Dighe Shweta Anil

Software Engineering

(Code : 214454)

Semester IV - Information Technology (Savitribai Phule Pune University)

Strictly as per the New Credit System Syllabus (2019 Course)
Savitribai Phule Pune University w.e.f. academic year 2020-2021

M. A. Ansari

Ph.D. (Computer Engineering) (Pursuing)
Assistant Professor,
Department Of Computer Engineering,
Smt. Kashibai Navale College of Engineering,
Vadgaon Budruk, Pune - 411041.

Dr. Vinod V. Kimbahune

M.E. (IT), Ph.D. (Computer Engineering)
Associate Professor,
Department of Computer Engineering,
Smt. Kashibai Navale College of Engineering,
Vadgaon Budruk, Off Sinhgad Road.
Pune - 411041

Dr. Shafi K. Pathan

Ph.D. (Computer Engineering)
Professor, Department of Computer Engineering,
Smt. Kashibai Navale College of Engineering,
Vadgaon Budruk, Off Sinhgad Road,
Pune - 411041

Prasad B. Chaudhari

M.E. (Computer)
Assistant Professor,
Department of Computer Engineering,
Smt. Kashibai Navale College of Engineering,
Vadgaon Budruk, Off Sinhgad Road,
Pune - 411041

TechKnowledge Publications

PE111A Price ₹ 235/-

CO5 : *Use quality attributes and testing principles in software development life cycle.*

CO6 : *Discuss recent trends in Software engineering by using CASE and agile tools.*

COURSE CONTENTS

Unit I : Introduction To Software Engineering (06 Hours)

Software Engineering Fundamentals : Nature of Software, Software Engineering Practice, Software Process, Software Myths.

Process Models : A Generic Process Model, Linear Sequential Development Model, Iterative Development Model, The incremental Development Model

Agile software development : Agile manifesto, agility principles, Agile methods, myth of planned development, Introduction to Extreme programming and Scrum.

Agile Practices : test driven development, pair programming, continuous integration in DevOps, Refactoring

(Refer Chapters – 1, 2 and 3)

Case Study : An information system – Library Management system

Mapping of Course Outcomes for Unit I : CO1

Unit II : Requirements Engineering & Analysis (06 Hours)

Requirements Engineering : User and system requirements, Functional and non-functional requirements, requirements engineering (elicitation, specification, validation, negotiation) prioritizing requirements (Kano diagram), requirement traceability matrix (RTM)

Software Requirements Specification (SRS) : software requirements Specification document, structure of SRS, writing a SRS, structured SRS for online shopping,

Requirements Analysis : Analysis Model, data modeling, scenario based modeling, class based modeling, Flow oriented modeling, behavioral modeling-Introduction to UML diagrams (Refer Chapter – 4, 5 and 6)

Case Study : Library Management system

Mapping of Course Outcomes for Unit II : CO2

Unit III : Design Engineering (07 Hours)

Design Engineering : Design Process & quality, Design Concepts, design Model, Pattern-based Software Design. Architectural Design : Design Decisions, Views, Patterns, Application Architectures,

Component level Design : component, Designing class based components, conducting component-level design, User Interface Design : The golden rules, Interface Design steps& Analysis, Design Evaluation

(Refer Chapters – 7, 8 and 9)

Case Study : Web App Design / Library Management System

Mapping of Course Outcomes for Unit III : CO3

Unit IV : Project Planning, Management And Estimation (07 Hours)

Project Planning : Project initiation, Planning Scope Management, Creating the Work Breakdown Structure, scheduling: Importance of Project Schedules, Developing the Schedule using Gantt Charts, PERT/ CPM

Project Management : The Management Spectrum, People, Product, Process, Project, The W5HH Principle, Metrics in the Process and Project Domains, Software Measurement : size &function-oriented metrics (FP & LOC), Metrics for Project

Project Estimation : Software Project Estimation, Decomposition Techniques, Cost Estimation Tools and Techniques, Typical Problems with IT Cost Estimates. **(Refer Chapters - 10, 11 and 12)**

Case Study : Project Management tool like OpenProj or MS Project

Mapping of Course Outcomes for Unit IV : CO4

Unit V : Software Quality And Testing (07 Hours)

Quality Concepts : Quality, software quality, Quality Metrics, software quality dilemma, achieving software quality

Software Testing : Introduction to Software Testing, Principles of Testing, Test plan, Test case, Types of Testing, Verification & Validation, Testing strategies, Defect Management, Defect Life Cycle, Bug Reporting, debugging.

(Refer Chapters - 13 and 14)

Case Study : Software testing tool like selenium

Mapping of Course Outcomes for Unit V : CO5

Unit VI : Formal Methods Recent Trends In Software Engineering (07 Hours)

Recent Trends in SE : SCM, Risk Management, Technology evolution, process trends, collaborative development, software reuse, test-driven development, global software development challenges, CASE – taxonomy, tool-kits, workbenches, environments, components of CASE, categories (upper, lower and integrated CASE tools), Introduction to agile tools Jira, Kanban **(Refer Chapter - 15)**

Case Study : CASE software / HP Quality Center (QC) / Jira

❑❑❑

UNIT I

UNIT II

User and system requirements, Functional and non-functional requirements, requirements engineering (elicitation, specification, validation, negotiation) prioritizing requirements (Kano diagram), requirement traceability matrix (RTM).

Analysis Model, data modeling, scenario based modeling, class based modeling, Flow oriented modeling, behavioral modeling-Introduction to UML diagrams

UNIT III

Design Process & quality, Design Concepts, design Model, Pattern-based Software Design.

UNIT IV

Project initiation, Planning Scope Management, Creating the Work Breakdown Structure, scheduling, Importance of Project Schedules, Developing the Schedule using Gantt Charts, PERT/ CPM

The Management Spectrum, People, Product, Process, Project, The W5HH Principle, Metrics in the Process and Project Domains, Software Measurement : size &function-oriented metrics (FP & LOC), Metrics for Project

Chapter 14 : Software Testing 14-1 to 14-35

Introduction to Software Testing, Principles of Testing, Test plan, Test case, Types of Testing, Verification & Validation, Testing strategies, Defect Management, Defect Life Cycle, Bug Reporting, debugging.

UNIT VI

SCM, Risk Management, Technology evolution, process trends, collaborative development, software reuse, test-driven development, global software development challenges, CASE - taxonomy, tool-kits, workbenches, environments, components of CASE, categories (upper, lower and integrated CASE tools), Introduction to agile tools Jira, Kanban.

❑❑❑

1 SOFTWARE ENGINEERING FUNDAMENTALS

Unit - I

Syllabus

Nature of Software, Software Engineering Practice, Software Process, Software Myths

1.1 Introduction to Software Engineering

University Questions

Q. What is software Engineering? **SPPU : May 12, Dec.19, 3 Marks**

Q. Define software engineering. **SPPU : Dec. 12, 4 Marks**

- Computer software has become an integral part of our daily lives. It helps in all sorts of businesses and decision makings in business. The application of computer software includes : Telecommunication, transportation, military, medical sciences, online shopping, entertainment industry, office products, education industry, construction business, IT industry, banking sector and many more.
- The software applications have a great impact on our social life and cultural life as well. Due to its widespread use, it is the need of time to develop technologies to produce high quality, user friendly, efficient and economical software.
- Computer software is actually a product developed by software engineers by making use of various software engineering processes and activities.
- Software consists of data, programs and the related documents. All these elements build a configuration that is created as a part of the software engineering process. The main motive behind software engineering is to give a framework for building software with better quality.
- We can say that the software has become the key element in all computer based systems and products.

1.2 Nature of Software

- The nature of software has great impact on software engineering systems. The general nature of software may describe the important characteristic : **Reliability**.
- If we consider the use of software in air traffic control system and space shuttle, then there should be not be any chances of failure of software. In these examples, if reliability is not taken into consideration, then the human life is at risk.

- In addition to this, there are four other important characteristics that describe the nature of software :

1. Absence of fundamental theory
2. Ease of change
3. Rapid evolution of technologies
4. Low manufacturing cost

1.2.1 Absence of Fundamental Theory

- If we consider the example of physics, we see there are fundamental laws of physics, but in software there are no such fundamental laws despite the researches done by the computer scientists.
- Because of this drawback, it is very difficult to do any reasoning about the software until it is developed. Thus the developers practice few software engineering standards that are not foolproof. But the codes written for developing software are following the discipline and some solid principles.

1.2.2 Ease of Change

- The software has the provision to be altered at any stage of time. Thus the software development organizations take the advantage of this feature. From the customer's point of view, the change is always required throughout its development cycle and even after its delivery to the customer.
- Since there are no basic rules of software, there is no rule available to accommodate the changes and its impact until it is developed. Thus we can say that the ease of change is a gift of God to the developers and the development organizations.

1.2.3 Rapid Evolution of Technologies

- In the modern age, the software development technologies and development environments are changing rapidly with an extreme speed. It becomes the need of the time to keep the software engineers updated and armed with latest technologies and skills.
- The software engineering standards must also be revised with the technology evolutions.

1.2.4 Low Manufacturing Cost

- The cost of software reproduction and installation is less as compared to a new development. Today nearly 80 percent of the software development contains only maintenance and only 20 percent is new development. This reflects that manufacturing a product is considerably involves very low cost.
- The software reusability is an important characteristic that benefits the development organizations a lot in manufacturing the software at low cost.

1.3 Software Engineering Practice

- Practice is collection of concepts, principles methods and tools those are considered while planning and developing the software.
- The software engineer implements this collection on daily basis for throughout development process of the software.
- Software engineers and managers apply the practice of software engineering.

Elements of Software Practice

- Following are the elements of software practice those are applied irrespective of the process model chosen for the development.
  - Concepts
  - Principles
  - Methods
  - Tools

1.3.1 The Essence of Practice

- George Polya has presented the concept "The Essence of Practice" in four sequential steps. These steps are :
  - To understand the problem (Communication and Analysis)
  - To plan the solution (Modeling and software design)
  - To carry out the plan (Coding)
  - To examine the result for accuracy (Testing and software quality assurance)

To understand the problem (Communication and analysis)

- This goal is achieved by the basic activities 'Communication with Customer' and 'Requirement analysis'.
  - Requirement analysis is a communication intensive activity. If there is misinterpretation of customer requirements, then obviously, the requirement analysis will be wrong.
  - Hence, wrong design will be formed and finally implementation will be wrong. So, customer will not get his expected results.
  - Hence with 'customer communication' data, functions and behaviour of the system to be implemented must be stated clearly during 'requirement analysis' activity.
  - In other words, practice is essential to understand problem statement.

To plan the solution (Modeling and software design)

This goal is achieved by activities Modeling and software design. Here following issues are considered.

Reusable components

- To implement required data, functions and behaviour of the system, the 'reusable' components are sear
  'Reusable components' are the components those developed for some specific application and can be reu
  development of other systems.
  - **Subsystems :** The main system requirements are divided into different groups so that by imple
    'subsystems' modularized solution can be achieved. By combining all subsystems, the combined
    produced by main system.
  - **Simple design :** The simple design is created so that design can be easily implemented using a
    programming language.

To carry out the plan (Code generation)

- This goal is achieved in 'Coding' step. The design is translated into code by using appropriate language. The focus is given on following issues :

- In addition to this, there are four other important characteristics that describe the nature of software :

1. Absence of fundamental theory
2. Ease of change
3. Rapid evolution of technologies
4. Low manufacturing cost

1.2.1 Absence of Fundamental Theory

- If we consider the example of physics, we see there are fundamental laws of physics, but in software there are no such fundamental laws despite the researches done by the computer scientists.
- Because of this drawback, it is very difficult to do any reasoning about the software until it is developed. Thus the developers practice few software engineering standards that are not foolproof. But the codes written for developing software are following the discipline and some solid principles.

1.2.2 Ease of Change

- The software has the provision to be altered at any stage of time. Thus the software development organizations take the advantage of this feature. From the customer's point of view, the change is always required throughout its development cycle and even after its delivery to the customer.
- Since there are no basic rules of software, there is no rule available to accommodate the changes and its impact until it is developed. Thus we can say that the ease of change is a gift of God to the developers and the development organizations.

1.2.3 Rapid Evolution of Technologies

- In the modern age, the software development technologies and development environments are changing rapidly with an extreme speed. It becomes the need of the time to keep the software engineers updated and armed with latest technologies and skills.
- The software engineering standards must also be revised with the technology evolutions.

1.2.4 Low Manufacturing Cost

- The cost of software reproduction and installation is less as compared to a new development. Today nearly 80 percent of the software development contains only maintenance and only 20 percent is new development. This reflects that manufacturing a product is considerably involves very low cost.
- The software reusability is an important characteristic that benefits the development organizations a lot in manufacturing the software at low cost.

1.3 Software Engineering Practice

- Practice is collection of concepts, principles methods and tools those are considered while planning and developing the software.

  The software engineer implements this collection on daily basis for throughout development process of the software.

  Software engineers and managers apply the practice of software engineering.

Elements of Software Practice

- Following are the elements of software practice those are applied irrespective of the process model chosen for the development.
  - Concepts
  - Principles
  - Methods
  - Tools

1.3.1 The Essence of Practice

- George Polya has presented the concept "The Essence of Practice" in four sequential steps. These steps are :
  - To understand the problem (Communication and Analysis)
  - To plan the solution (Modeling and software design)
  - To carry out the plan (Coding)
  - To examine the result for accuracy (Testing and software quality assurance)

To understand the problem (Communication and analysis)

- This goal is achieved by the basic activities 'Communication with Customer' and 'Requirement analysis'.
  - Requirement analysis is a communication intensive activity. If there is misinterpretation of customer requirements, then obviously, the requirement analysis will be wrong.
  - Hence, wrong design will be formed and finally implementation will be wrong. So, customer will not get his expected results.
  - Hence with 'customer communication' data, functions and behaviour of the system to be implemented must be stated clearly during 'requirement analysis' activity.
  - In other words, practice is essential to understand problem statement.

To plan the solution (Modeling and software design)

This goal is achieved by activities Modeling and software design. Here following issues are considered.

Reusable components

- To implement required data, functions and behaviour of the system, the 'reusable' components are searched. 'Reusable components' are the components those developed for some specific application and can be reused in development of other systems.
  - **Subsystems :** The main system requirements are divided into different groups so that by implementing 'subsystems' modularized solution can be achieved. By combining all subsystems, the combined result is produced by main system.
  - **Simple design :** The simple design is created so that design can be easily implemented using appropriate programming language.

To carry out the plan (Code generation)

- This goal is achieved in 'Coding' step. The design is translated into code by using appropriate programming language. The focus is given on following issues :

- All design details must reflect into coding i.e. actual implementation.
- Each module of coding must be correct to generate required output.

To examine the result for accuracy (Testing and quality assurance)

- This goal is achieved 'Testing and Assurance' activities. This step focus on following issues :
  - To check for expected results.
  - To check whether required testing tools are available.
  - To check for data, functions and behaviour of the system.
  - To check whether system is maintaining good quality.
  - To implement steps for software quality assurance.

1.3.2 Core Principles

The term 'principle' implies "some important law or assumption". In software engineering, the principles must be followed in every stage of software development.

For example

- The principles may be applied to 'software development' as whole.
- Principles those are applied to 'Communication' stage of software.
- Principles those are applied to 'Planning' stage of software.
- Principles those are applied to 'Analysis Modeling' stage of software.
- Principles those are applied to 'Design Modeling' stage of software.
- Principles those are applied to 'Construction' stage of software.
- Principles those are applied to 'Testing' stage of software
- Principles those are applied to 'Deployment' stage of software.

David Hooker's seven principles

David Hooker has proposed seven principles called as 'core principles'. These principles are applied to 'software development' as whole.

- **Principle Number 1 : The reason at all exists.**

  The complete software is one, which satisfies all the requirements by 'customer's point of view'. Hence before beginning a software project, be sure that the software has a business purpose. Hence first priority goes to customer's satisfaction.

- **Principle Number 2 : Keep simple but perfect.**

  The software design must be as simple as possible. If there are too many interfaces in the system, it's difficult to construct and implement such type of system. Hence design should be simple enough. But, simple design doesn't mean to skip the requirements. Hence design should be simple and perfect.

- **Principle Number 3 : Maintain the vision.**

  A clear vision is required for the success of the project. Hence maintain the architectural vision of software while designing it.

- **Principle Number 4 : What you produce, others will consume.**

  Other members in the team may use the software, which is analyzed, designed, and constructed by one person in a team. After deployment of software to the customer, the software 'enhancement' may be required as per customer's demand. Hence to maintain the software or to extend it, the software may be referred to develop other applications. Hence, the system must be transparent enough.

- **Principle Number 5 : Be Open to the future.**

  The solution for any problem must be generalized and not specific. The generalized solution can be reused for other applications. The technology changes day to day. Today's software gets outdated tomorrow. The programming languages used for software development may be outdated tomorrow. Hence the developed system must be generalized enough to absorb all these changes.

- **Principle Number 6 : Plan ahead for reuse.**

  'Reusability' is a property of good quality software. 'Reusable' software can be used in the development of other applications. Using Object Oriented programming languages like C++, it's possible to develop reusable modules. If reusable modules are available, then system can be developed very fast.

- **Principle Number 7 : Think for better solutions.**

  When we think about something, we get more ideas. If we are having solution and we think for better solution, the better idea will come out. This can improve the quality of system.

1.4 Software Process

University Questions

Q. What is software process? SPPU : May 12, May 13, 3 Marks

Q. Explain software Engineering process framework activities. SPPU : May 16, 5 Marks

- A software process can be illustrated by the following Fig. 1.4.1. In the Fig. 1.4.1, a common process framework is exhibited. This framework is established by dividing overall framework activities into small number of framework activities.
- And these small activities are applicable to the entire project irrespective of its size and complexity. A framework is a collection of task sets and these task sets consists of :
  - Collection of small work tasks
  - Project milestones, deliverable i.e. actual work product and
  - Software quality assurance points

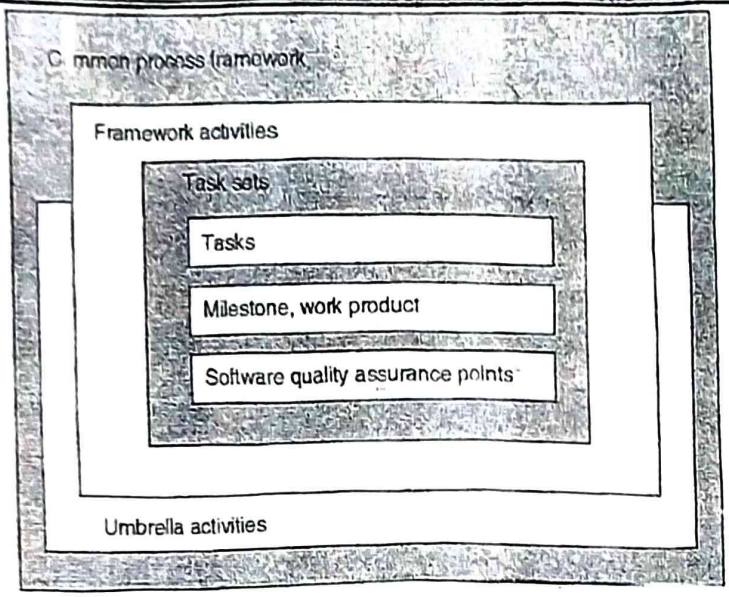


Fig. 1.4.1 : A software process

- There are various activities called **umbrella activities** are also there and these activities are associated throughout the development process.
- The umbrella activities include :

1. Software project tracking and control
2. Risk management
3. Software Quality Assurance (SQA)
4. Formal Technical Reviews (FTR)
5. Measurement
6. Software Configuration Management (SCM)
7. Reusability Management
8. Work product preparation and production

- All these umbrella activities actually constitute the process model. Also the umbrella activities are independent and occur throughout the process

1.4.1 Umbrella Activities

University Question

Q. What are framework and umbrella activities? **SPPU : May 12, 3 Marks**

The framework described in generic view of software engineering (Fig. 1.4.1 : A software process) is complemented by number of Umbrella activities. Typical **umbrella activities** are :

1. **Software project tracking and control**
   - Developing team has to assess project plan and compare with predefined schedule.
   - If project plan doesn't match with predefined schedule, necessary actions are taken to maintain the schedule.

2. **Risk management**

   Risk is event that may or may not occur. But if that event happens, it causes some unwanted outcomes. Hence proper management of risk is required.

3. **Software Quality Assurance (SQA)**

   - SQA is nothing but planned and systematic pattern of activities those are required to give guarantee of software quality.
   - For example during software development, meetings are conducted in every stage of development to find out defects and suggest improvement to yield good quality software.

4. **Formal Technical Reviews (FTR)**

   - FTR is a meeting conducted by technical staff. The purpose of meeting is to detect quality problems and suggest improvements.
   - The technical staff focuses on quality from customer's point of view, i.e. what customer exactly wants.

5. **Measurement**

   - It includes the efforts required to measure the software.
   - Software can not be measured directly. It is measured by some direct measures (e.g. cost, lines of code, size of software etc) and indirect measures (e.g. quality of software, which is measured in terms of other factors. Hence it is an indirect measure of software.)

6. **Software Configuration Management (SCM)**

   It manages the effects of change throughout the software process.

7. **Reusability management**

   - It defines criteria for product reuse.
   - If software components developed for certain application can be used in development of other applications, then it's good quality software.

8. **Work product preparation and production**

   It includes the activities required to create documents, logs, forms, lists and user manuals for developed software.

1.5 Software Myths

University Questions

Q. State and Explain different myths SPPU : Dec. 16, 5 Mark

Q. State and explain any Five Myths of software development with reality. SPPU : May 19, 5 Mar

- Dictionary meaning of myth is 'fable stories'. It is a fiction, imagination or it is a thing or story whose exi is not verifiable. In relation with computer software myth is nothing but the misinformation, misunderstar or confusions propagated in software development field.

- In software development the myths can be considered as three level myths.

1. Management level myths
2. Customer level myths
3. Practitioner level myths

1.5.1 Management Level Myths (or Manager Level Myths)

University Questions

Q. Explain in detail software_myths : Management myths. SPPU : May 12, May 13, 3 Marks

Q. What are the management myths ? SPPU : Dec. 12, 4 Marks

➢ **Myth**

Manager thinks "there is no need to change approach to software development. We can develop same kind of software that we have developed ten years ago".

➢ **Reality**

- Application domain may be same but quality of software need to improve according to customer's demand.
- The customer demands change time to time.

➢ **Myth**

- We can buy software tools and use them.

➢ **Reality**

- Software tools are readymade software that helps in creation of other software e.g. Microsoft front page/ Microsoft Publisher. All these software can be used for web development.
- Rational rose used to draw UML diagrams (e.g. use cases, sequential, class diagrams etc).
- Such software tools are available for every stage of software development (Requirement analysis, designing, coding, testing etc). But majority of software developers do not use them. Moreover, these tools also have some limitations.

➢ **Myth**

Manager thinks "when needed, we can add more programmers for faster software development".

➢ **Reality**

- When new peoples are added to the project, the training must be given to such newcomers.
- Hence people previously working on that project spend time for training people.
- Hence project can not be completed fast just by adding more people at any time during project development.

➢ **Myth**

If the developer outsource the software project to a third party, he can be tension free and wait for the project to done smoothly.

➢ **Reality**

When an organization is unable to manage and control the software projects internally, it will also be very difficult for them to get the project done by outsourcing it.

➢ **Myth**

There is a book that contains standards and procedures for developing software, it will provide the developer everything that he needs in the development process.

➢ **Reality**

The rule book exists. But the questions arise :

- Is it actually used by the developers ?
- Are software developers aware of this type of book of standards and procedures ?
- Does it contain all the modern engineering practices ?
- Is it foolproof ?
- Does it focus on quality ?
- Does it streamlined to timely delivery ?
- In most of the cases, the simple answer to all these queries is a big "NO".

➢ **Myth**

- The organization has state-of-the-art development tools.
- They have the latest configuration computers for their developers to provide the good platform to produce their work efficiently.

➢ **Reality**

- In actual scenario the latest hardware configuration computer will not help in producing high-quality software.
- But the Computer-Aided Software Engineering (CASE) tools are very important for achieving good quality software.
- In most of the organizations the software developers do not these CASE tools effectively and efficiently.

1.5.2 Customer Level Myths

University Question

Q. Discuss the software myths and realities in customer perspective. **SPPU : Dec. 19, 6 Marks**

➢ **Myth**

Only the general statement is sufficient and no need to mention detail project requirements.

➢ **Reality**

- Only general statement is not sufficient for project development. Other detail project requirements are also essential.
- Customer must provide other information, design details, validation criteria.
- Hence during complete software development process customer and developer communication is essential.

➢ **Myth**

Project requirements continuously change but can be easily accommodated in software.

➢ **Reality**

- If change is requested in early stages of software development, it can be easily carried out. But if change is requested in later stages, cost of change rapidly increases.
- If change is requested in maintenance, modifications are required in design, coding, testing.
- Hence cost of modification rapidly increases. Thus changes can't be easily absorbed. They result in increase in cost.

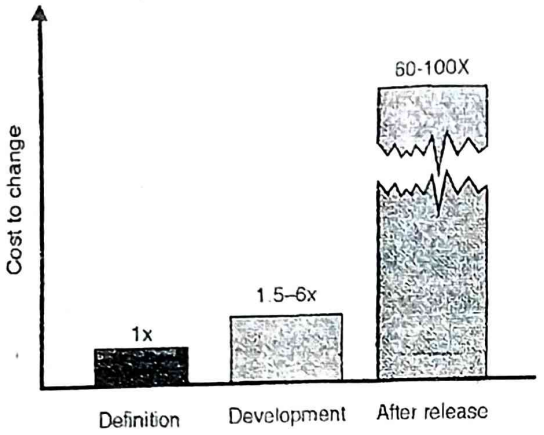


Fig. 1.5.1 : The impact of change

1.5.3 Practitioner Level Myths (or Developer Level Myths)

University Questions

Q. Explain in detail software myths: Practitioner's myths. SPPU : May 12, May 13, 4 Marks

Q. Discuss practitioner's myths of software development. SPPU : Dec. 15, 5 Marks

- **Myth**

  Practitioners think "Once we write program and get into work, our job is over".

- **Reallty**

  Almost 60 to 80 percent work is required after delivering the project to the customer for the first time.

- **Myth**

  Until I get program running, I have no way of assessing its quality.

- **Reallty**

  - The most effective quality assurance mechanisms can be applied during project development. The formal Technical Reviews are conducted to assure the quality
  - Formal Technical Review is meeting conducted by technical staff. FTR is applied in any stage of software development (Requirement analysis, designing, coding, testing etc).
  - FTR is not problem solving activity but it is applied to find out defects in any stage of software development. These defects can then removed to improve the quality of software.

- **Myth**

  When project is successful, deliverable product is only working program.

- **Reality**

  Working program is just part of software product. Actual project delivery includes all documentations, help files and guidance for handling the software by user.

Myth

The software engineering process creates larger and unnecessary documentation and ultimately it will slow down the process.

- Reality
  - Software engineering the process that creates the quality and it is the not the process of only creating the documents.
  - The good quality of the product will reduce the extra work involved in rework.
  - And finally reducing the overhead of rework will lead to quicker development to complete the desired work on scheduled time.

1.6 The Characteristics of Software

University Question

Q. What are the characteristics of software? **SPPU : May 12, Dec. 12, May 13, Dec. 19, 3 Marks**

- Software is having some characteristics, those are totally different from hardware characteristics or the industrial manufacturing :
  - Software is developed or engineered and it is not manufactured like other hardware products.
- There exist some similarities between development of software and manufacturing of hardware :
  - In both the cases quality is achieved by good design but manufacturing phase of hardware can introduce quality problems that are absent in software.
  - Both activities depend on people but relationship between people applied and work done is different in both the cases.
  - Both the cases require the 'construction of product', but approaches are different.
  - Software does not wear out.
  - Note that, wear out means process of losing the material.
  - Hardware components are affected by dust, temperature and other environmental maladies. Stated simply, hardware can wear out. However software does not influenced by such environmental means.
  - When hardware component wear out, it can be replaced by spare part. But there are no spare parts for software. Any software error indicates error in design or coding of that software. Hence software maintenance involves more complexity than hardware maintenance.
  - Mostly software is custom built rather than assembled from existing components.
  - Computer hardware can be assembled from existing components as per our requirements. But that is not the case for software. Software is developed according to customer's need.

Review Questions

Q. Why Computer Software is important?

Q. Explain the term Computer Software in brief.

Q. List and explain the characteristics that describe the nature of software.

Q. What are various Umbrella Activities associated in the Development Process?

Q. Describe Common Process Framework with the help of diagram.

Q. What is software myth? Give an example?

Q. List and explain practitioners myths and what are their corresponding realities?

Q. State a software myth each which customers, developers and project manager believe. What is the reality in each case?

□□□

2 PROCESS MODELS

Unit - I

Syllabus

A Generic Process Model, Linear Sequential Development Model, Iterative Development Model, The incremental Development Model.

2.1 Generic Process Model

University Questions

Q. What is generic process model ? Explain its activities. **SPPU : May 16, 5 Marks**

Q. Write short note on Generic process model. **SPPU : Dec. 16, 5 Marks**

A software process is collection of various activities. There are five generic process framework activities:

1. Communication
2. Planning
3. Modeling
4. Construction
5. Deployment

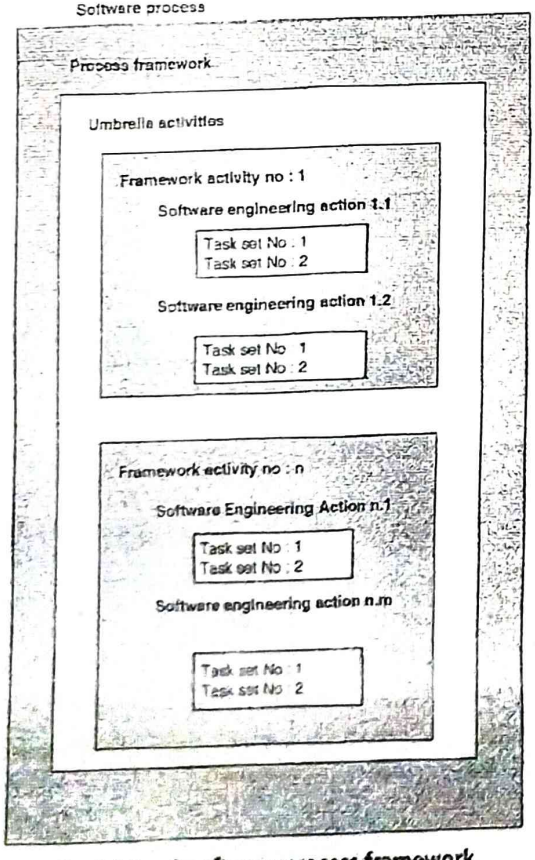


Fig. 2.1.1 : A software process framework

2.2 Linear Sequential Development Model (Waterfall Model)

University Question

Q. Explain in detail all the process phases of waterfall process model and state merits/demerits of the same. **SPPU : May 12, 3 Marks**

- This model is also called as **'Linear sequential model'** or **'Classic life cycle model'**. Classic life cycle paradigm begin at system level and goes through analysis, design, coding, testing and maintenance.

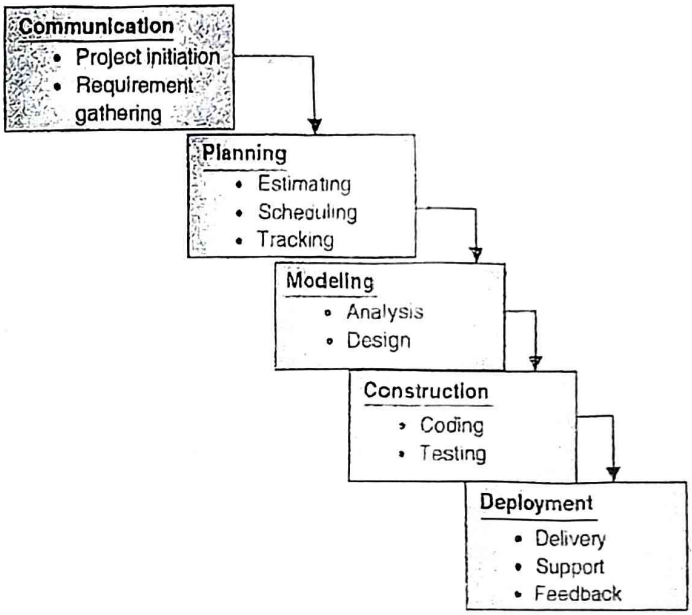


Fig. 2.2.1 : Linear Sequential Development Model

- An alternative design for 'Linear Sequential Model' is as follows :

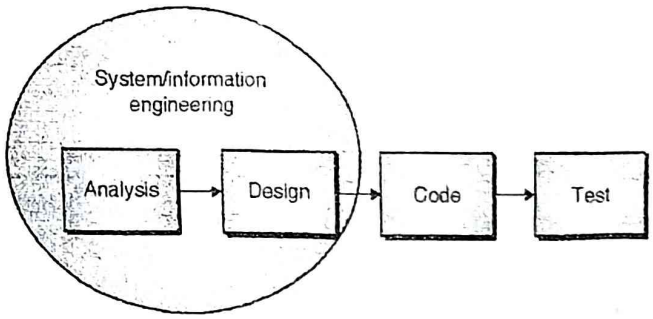


Fig. 2.2.2 : The linear sequential model

1. **Communication**
   - The software development process starts with communication between customer and developer.
   - According to waterfall model customer must state all requirements at the beginning of project.
2. **Planning**

   It includes complete estimation (e.g. cost estimation of project) and scheduling (complete timeline chart for project development) and tracking.

3. Modeling

   - It includes detail requirement analysis and project design (algorithm, flowchart etc).
   - Flowchart shows complete pictorial flow of program whereas algorithm is step-by-step solution of problem.

4. Construction

   It includes coding and testing steps :

   i) **Coding :** Design details are implemented using appropriate programming language.

   ii) **Testing :** Testing is carried out to check whether flow of coding is correct, to check out the errors of program e.g. in C program just by pressing F7 key we check step by step execution of program or by using "Add-Watch" we add the variables and watch the values of variables, to check whether program is giving expected output as per input specifications.

5. Deployment

   - It includes software delivery, support and feedback from customer.
   - If customer suggest some corrections, or demands additional capabilities then changes are required for such corrections or enhancement

Merits / Advantages

1. This model is very simple and easy to understand and use.
2. Waterfall model is systematic sequential approach for software development. Hence it is most widely used paradigm for software development
3. In this approach, each phase is processed and completed at one time and thus avoids phases overlapping.
4. It is very easy to manage since all the requirements are very well understood in the beginning itself.
5. It establishes the milestones whenever the products are produced or reviews available.

Demerits / Disadvantages

1. Problems in this model remain uncovered until software testing.
2. One of the disadvantages of this approach is 'blocking states'. In blocking state, the members of development team waits for other members to complete the dependent tasks. Due to this, huge amount of time spent in waiting rather than spending time on some productive work. In today's world, the software work is fast paced and thus waterfall model is not useful.
3. According to this model customer must state all his requirements at the beginning stage of development, which is difficult for the customer.
4. This model is step-by-step systematic sequential approach. Ultimately, customer gets the working version of software too late. Hence customer requires patience.
5. This approach is actually not realistic and does not match with real projects.
6. Also it does not incorporate the risk assessment.
7. Finally it is useful for smaller projects only where requirements are well understood in the beginning only.

2.3 Iterative Development Model : Component Based Development

- Component Based Development has many characteristics of spiral model. It is evolutionary in nature and includes iterative approach for software development.
- This model composes the applications from pre-packaged software components i.e. from existing software modules.
- Modeling and construction activity begin with identification of candidate components. These components can designed as either conventional software modules or object oriented classes or packages.
- The component is nearly independent and replaceable part of system.
- A package of classes is a collection of objects that work together to achieve some result.

Component Based Development includes following steps :

- Available components-based products are researched and evaluated for application domain.
- Component integration issues are considered.
- Software architecture is designed to accommodate the components.
- Components are integrated i.e. combined into architecture.
- Testing is carried out.
- Important feature of Component Based Development is that this model leads to 'software reuse'.
- Reusable components means the components i.e. software modules those are developed for specific application can be reused in development of other application projects. Reusability provides many benefits for software development.

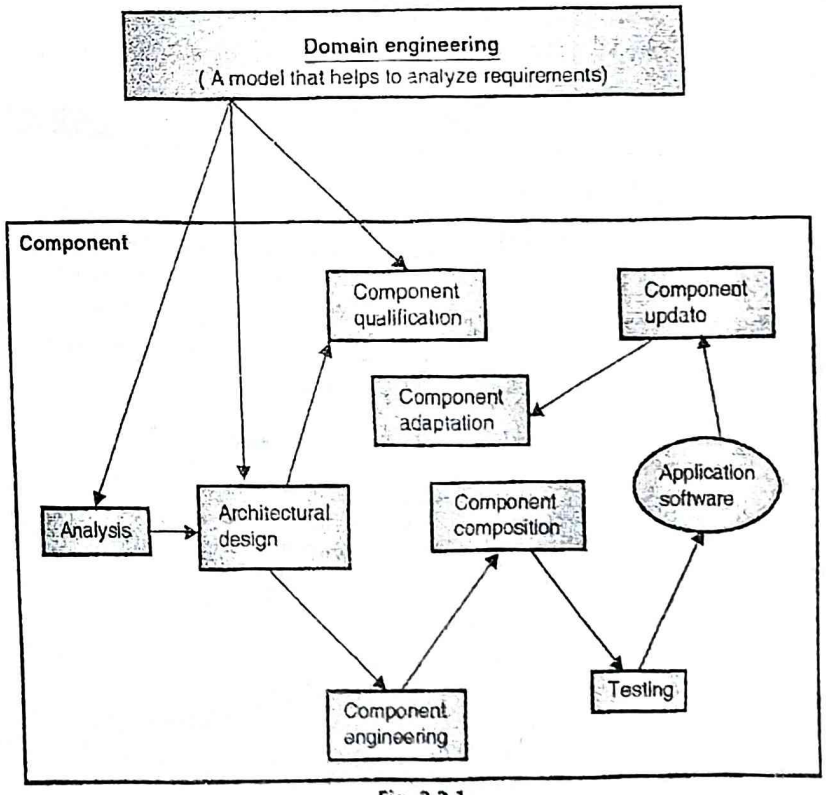


Fig. 2.3.1

Component Based Software Engineering (CBSE) includes following steps :

- 'Domain Engineering' creates a model of application domain that is used as a basis of analyzing the requirements.
- Software architecture provides inputs for design of the application.
- Finally after reusable components are purchased, selected from existing library, or constructed as a part of domain engineering, they are made available to software engineering during component-based development.

Merits / Advantages

1. The component based development model supports reusability.
2. Due to reusability, there is reduction in development cycle time.
3. Ultimately the cost of overall development reduces substantially.
4. There is significant increase in productivity.

Demerits / Disadvantages

1. The component based development model suffers from several problems. One of the serious problems is late discovery of the errors due to component interface or architectural mismatches.
2. The problem in encapsulation occurs due to functional overlapping.
3. It is very difficult to find that in which component a particular function will be implemented.
4. It is very difficult to achieve a complete separation of process from component.
5 This approach can not be fully utilized if a development organization does not adopt the principles of component based software engineering.

2.4 The Incremental Development Model

University Question

Q. Explain the incremental software process model in detail. **SPPU Dec. 16, 5 Marks**

- The incremental model combines the elements of waterfall model applied in an interactive fashion.
- The first increment is generally a core product. Each increment produces the product, which is submitted to the customer. The customer suggests some modifications.
- The next increment implements customer's suggestions and some additional requirements in previous increment. The process is repeated until the product is finished.
- Consider the development of word processing software (like MS Word or Word Star) using incremental model.

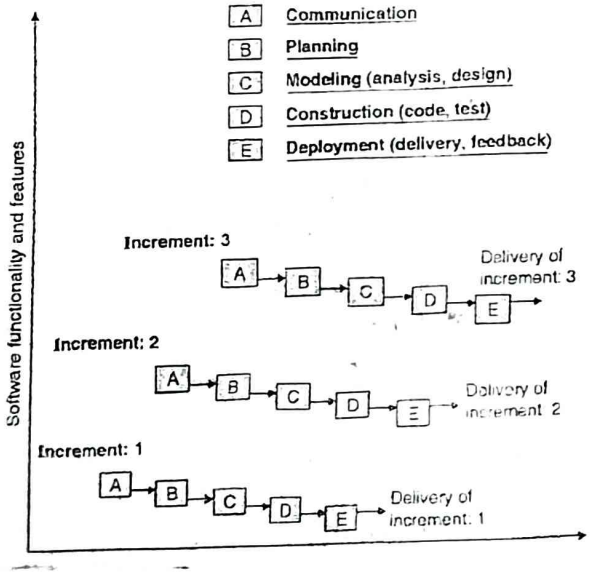


Fig. 2.4.1 : Project calendar time

Increment - 1

Basic file management functions like : New, Open, Save, Save as etc. can be implemented in first increment

Increment - 2

- More sophisticated editing and document production capabilities can be implemented in second increment.
- For example rulers, margins and multiple font selection can be implemented.

Increment - 3 Spelling, grammar checking and auto correction of words is implemented in third increment.

Increment - 4

The final advanced page layout capabilities are developed in fourth increment. And so on till the product is finished.

Merits / Advantages

1. The incremental model is useful when the size of development team is small in the beginning or some team members are not available. Thus early increments can be implemented with small size of team.
2. If the product satisfies the client, then the size of team can be increased.
3. Also increments can be properly planned to handle all types of technical risks. For example some application may require a new hardware that is presently not available. In this situation the increments can be planned to avoid the use of that hardware.
4. The initial product delivery is faster and cost of development is low.
5. The customers can respond to the functionalities after each increment and come up with feedback.

Demerits / Disadvantages

1. The cost of finished product may be increased in the end beyond the cost estimated.
2. After each increment, the customer can demand for additional functionalities that may cause serious problems to the system architecture.
3. It needs a very clear and complete planning and design before the whole system is broken into small increments.

2.5 Comparison between Evolutionary and Incremental Models

University Question

Q. Compare between Evolutionary and Incremental Models. **SPPU - May 19, 5 Marks**

Sr. No.	Evolutionary process models	Incremental models
1.	Evolutionary software processes do not establish the maximum speed of the evolution. Due to this development process becomes slow.	Incremental models develops the software quickly and each increment produces the product, which is submitted to the customer. The customer suggests some modifications.
2.	Evolutionary process models lack flexibility, extensibility, and high quality.	Incremental models provides flexibility and extensibility and new modifications are incorporated in each increment.

Sr. No.	Evolutionary process models	Incremental models
3.	It is less popular.	It is more popular.
4.	Time does not allow a full and complete system to be developed.	It provides complete and full developed systems.
5.	Example : Prototyping, Spiral and Concurrent models.	Example : Water fall model, Incremental models, RAD Model

Review Questions

Q. What is generic process model? Explain its activities.

Q. What are the disadvantages of the water fall model?

Q. Explain the waterfall model with work products of each activity.

Q. Explain advantages and disadvantages of waterfall model.

Q. Explain the incremental model with advantages and disadvantages?

□□□

3 AGILE SOFTWARE DEVELOPMENT & PRACTICES

Unit - I

Syllabus

Agile manifesto, agility principles, Agile methods, myth of planned development, Introduction to Extreme programming and Scrum. Test driven development, pair programming, continuous integration in DevOps, Refactoring

3.1 Introduction to Agile Software Development Process

- In today's modern world, businesses spread across the globe in each sphere of life. It has become the global phenomenon. Due to this changing environment, the client must respond to new opportunities and to the volatile market conditions, and to the arrival of new products and services.
- The software application is actually the heart of all the business operations. So according to the rapid changing rules and business ideas, it should respond quickly. Thus the developer is supposed to develop the new software quickly to meet the desired requirements in the stipulated time.
- In fact, in such rapidly changing market condition, rapid development of the software and its urgent delivery is the need of the time and it is supposed to be the most critical requirement of any software system.
- Normally most of the clients are even ready to compromise the quality of software and the requirements at the cost of faster development of the software and its quick delivery.
- Since all the businesses usually operate in the rapidly changing environment, it becomes highly impossible to gather complete set of requirements. The requirements elicited by the customers in the beginning are always changed since the customers find it difficult to predict the actual working of the software and the challenges to come across.
- Thus understanding the requirements quickly and accommodate the requirements change narrated by the customer quickly is the need of the time. Otherwise, delayed delivery of the software may make it unusable and out of date.
- Thus waiting for the complete requirements gathering from the customer will not work for the rapid development. Since the requirement of the customer changes with the progress of the software development process.
- Due to this reason, the conventional approaches like waterfall model or specification based processes will delay the final delivery of the software system.
- But in some cases like critical control system, the complete analysis of the requirement is mandatory else it may cost life. For such type of safety systems, plan-driven approach is always the best choice.
- Rapid software development processes are the ultimate choice for the development of useful products in quick time span. In this approach the software is developed as a series of increments.

- Agile methods are incremental development methods in which the increments are usually small the new versions of the system are created rapidly and handed over to the customers within the span of 15 to 20 days and their feedback is received for the next versions.

3.2 Agility Principles

University Questions

Q. Discuss agility principles used in agile development. SPPU : Dec. 14, 5 Marks

Q. What is Agility? SPPU : Dec. 19, 5 Marks

- An agile process model includes the concept of development along with a set of guidelines necessary for the development process. The conceptual design is necessary for the satisfaction of the customer. The concept is also necessary for the incremental delivery of the product. The concept or the philosophy makes development task simple. The agility team must be highly motivated, updated with latest skill sets and should focus on development simplicity.
- We can say that agile development is an alternative approach to the conventional development in certain projects.
- The development guidelines emphasize on analysis and design activities and continuous communication between developers and customers. An agile team quickly responds to changes. The changes may be in development, changes in the team members, and changes due to new technology. The agility can be applied to any software process. It emphasizes rapid delivery of operational software.
- The agile alliance has given twelve agility principles as follows :
  1. Satisfy customer through early and continuous delivery.
  2. Accommodate changing requirements.
  3. Deliver the working software frequently in shorter time span.
  4. The customer, business people and developers must work together on daily basis during entire development.
  5. Complete the task with motivated developers and facilitate healthy and friendly environment.
  6. Convey the message orally, face-to-face conversation.
  7. Working software is the primary measure of progress.
  8. Agile process promotes sustainable development.
  9. Achieve technical excellence and good design.
  10. There must be simplicity in development.
  11. The best architecture, requirements and design emerge from self-organizing teams.
  12. Every team should think how to become more effective. It should review regularly and adjust its behaviour accordingly.
- Thus the principles of agility may be applied to any software process. To achieve agile development, the team must obey all the principles mentioned above and conduct proper planning.
- All the agile software processes should address three important assumptions :
  - Difficult to predict in advance software requirements

- Design and construction are interleaved in most of the projects. It is difficult to predict design before construction.
- Analysis, design, construction and testing are not much predictable

- To address these assumptions of unpredictability, the agile development process must be adaptable. So an agile process must adapt incrementally.

3.2.1 Relation of Agility and Cost of Change

University Question

Q. What is relation of agility and cost of change? SPPU : Dec. 14, 5 Marks

Agility :

- Agility can be defined as
  - Effective response to change
  - Effective communication among stakeholders
  - Drawing customers onto team
  - Organizing team so that it is in control of the work performed
- In order to deliver agile software it includes following things
  - It depends what is required by customer
  - Recognizes plans are short lived
  - Develops software iteratively with focus on construction activity
  - Delivers multiple software increments.
  - Adapts to changes.

Cost of Change :

- Cost of change curve is shown in Fig. 3.2.1.
- This cost curve is for single production release.
- We need to consider cost of change for different production release.

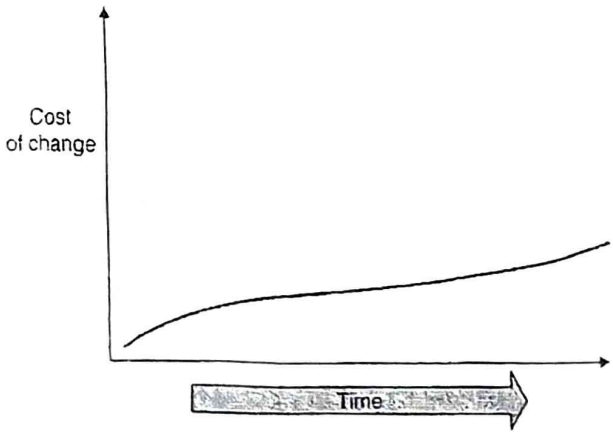


Fig. 3.2.1 : cost change in agile development

3.3 Agile Methods

- In the early days of software development, developer used to plan the project carefully, focus on quality assurance and employ the analysis and design methods supported by various CASE tools.
- This was the general practice of the software developer community that was responsible for the development of larger projects such as government projects etc.
- This larger projects use larger team and the team members may spread geographically across the world and they worked on the software for the longer period of time.
- Therefore, the development period may extend up to the 10 years from the initial specification to the final delivery of the product.
- Such a planed-driven approach will involve significant overheads in all the stages of the development processes like requirement specification, planning, designing, coding and documentation.
- In order to overcome this heavy weight plan-driven approach, the concept of agile methods was proposed. This methodology mainly focused on the product instead of focusing on the design and documentation part.
- All the agile methods trust on the incremental approach instead of the conventional waterfall approach. The incremental approach is the best approach where the customer requirements and the software requirements change rapidly or frequently.
- The philosophy behind agile methods is also observed in **Agile Manifesto** that is accepted by most of the leading software developers. The Agile Manifesto is discussed in the following section.

3.3.1 Agile Manifesto

University Questions

Q. What is Agile manifesto? **SPPU : Dec. 14, 5 Marks**

Q. Write the manifesto for agile software development. **SPPU : May 19, 5 Marks**

- The Agile Manifesto states that :
- We are uncovering better ways of developing software by doing it and helping others do it.
- Through this work we have come to value :
  - Individuals and interactions work over processes and tools
  - Working software takes responsibility of comprehensive documentation
  - Customer collaboration look after contract negotiation
  - Responding to change after having a good plan.
- Even though these agile methods are based on the concept of incremental development, quicker delivery and faster deployment, the agile manifesto gives different processes to achieve this.
- But most of the set of principles used by various experts that are based on agile manifesto are very much common.

3.3.2 Agile/XP methodology

University Question

Q. How Agile/XP methodology will help project managers? **SPPU : May 19, 5 Marks**

- Agile project management uses short development cycles called "sprints," each of which incorporates and adapts to stakeholder and customer feedback to produce an expertly-honed end product. Agile project management has become so popular partly due to the fast-paced nature of business today.
- With its focus on continued evolution and collaboration, the methodology targets organizations dealing with rapid to-market deadlines, shifting priorities, high stakeholder engagement, and a need for flexibility — in other words, most businesses today.
- Rather than spending six months developing a product or service that may be outdated by the time it hits the market, a company using Agile project management could release the first iteration within two weeks.
- They could then continue to release updated, adaptive versions over the next six months, resulting in a much more effective, relevant, and useful final deliverable. That's why Agile project management, which was originally developed for software companies, has since been adopted by a wide variety of industries, from financial services to transportation.

3.3.3 Benefits of Agile and XP Methodology project management

1. **Higher product quality**

   Because testing is integrated throughout the project development process, the team can perform regular checkups and find areas of improvement.

2. **Reduced risk**

   Agile project management virtually eliminates the chances of absolute project failure. Working in sprints allows teams to develop a working product from the beginning or fail fast and take another approach.

3. **Better visibility into project performance**

   Agile project management lets team members know how the project is progressing. Frequent Scrum meetings and sprint reviews provide increased transparency to everyone on the team.

4. **Increased project control**

   Team members have control throughout the project with more opportunities to test and adapt.

5. **Better project predictability**

   Breaking up the project into shorter sprints allows project managers to predict the exact cost, timeline, and resource allocation necessary for each sprint.

3.4 Myth of Planned Development

There are different myths in the agile development. Different myths are discussed below.

- **Agile development is methodology :** It is not methodology. It is set of values and principles which guide as a set of development methods. Different methods includes Adaptive software development, Agile modeling, Agile unified process, Crystal method, Disciplined agile delivery, Dynamic system development method, Extreme programming, Feature driven development, Learn software development, etc.
- **Agile is undisciplined :** Actually agile focus on individuals and interaction compared to process and tools. This is reason some user misinterprets agile process is undisciplined.
- **Agile has no planning :** Planning is everywhere. Planning can inhibit agility.

- **Agile has no documentation :** Main objective of agility is to remove documentation. It does not mean to fully removing the documentation. Agility focus on working software than the documentation. This is because of the dynamic nature of software development.
- **Agile has no upfront design :** It traces on simplified design. Developer generally implements only needed features.
- **Agile does not scale :** scaling software development is difficult. Some believe agile software will work for small projects but not for large projects. Agile process usually break large, complex projects into small modules. It means it is scalable.
- **Agile is just another fad :** agile process is used over the years which mean we cannot say it is fad. Fad means for time being and short-lived.

3.5 Introduction to Extreme Programming

University Questions

Q. Explain in detail extreme programming. SPPU : May 13, 5 Marks

Q. What is extreme programming ? SPPU : May 15, 5 Marks

The Extreme Programming is one of the most commonly used agile process models. All the agile process models obey the principles of agility and the manifesto of agile software development.

- The XP uses the concept of object oriented programming. This approach is preferred development paradigm.
- As in conventional approach, a developer focuses on the framework activities like planning, design, coding and testing, the XP also has set of rules and practices.
- Following Fig. 3.5.1 shows the Extreme Programming process and all the key XP activities are summarized.

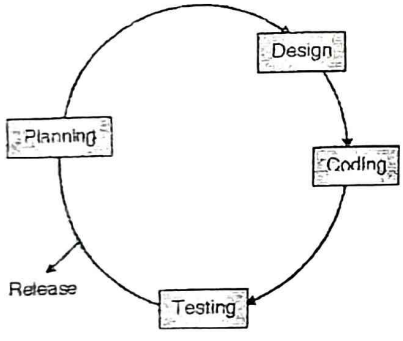


Fig. 3.5.1 : Extreme programming process

3.5.1 XP Values

University Questions

Q. List the drivers which are treated as XP values. SPPU : May 15, 5 Marks

Q. Discuss XP values. SPPU : Dec. 16, 5 Marks

- Following are a set of five values that establish a foundation for all work performed in context with XP.
  1. **Communication**
  2. **Simplicity**
  3. **Feedback**

4. Courage
5. Respect

- For any development process, there must be a regular meeting of developer and the customer. There should be a proper communication for requirement gathering and discussion of the concepts.
- The simple design can always be easily implemented in code.
- The feedback is an important activity which leads to the discipline in the development process.
- In every development projects, there is always a pressure situation. The courage or the discipline will definitely make the task easy.
- In addition to all the XP values, the agile should inculcate respect among all the team members, between other stake holder and with customers.

3.5.2 The XP Process

University Question

Q. Discuss XP process. **SPPU : Dec. 16, 5 Marks**

- The XP process encompasses four framework activities :

(i) Planning (ii) Design
(iii) Coding (iv) Testing

- **Planning** starts with requirement gathering activity that enables XP team to understand the business rules for the software. Customers and developers work together to decide the final requirement.
- **Design** of the product follows the KIS (Keep It Simple) in the XP environment. A simple design is always welcome over the complicated ones.
- **Coding** starts after the design work is over. Once the code is completed, it can be unit-tested immediately. The important concept during the coding activity in XP recommends that two people work together at one computer to create the code.
- **Testing** - All the individual unit tests are organized into a 'universal testing suite'. Integration and validation testing of the system can occur on daily basis. XP acceptances test, also called customer tests are specified by customer and focus on overall system features and functionality.

3.5.3 Industrial XP (IXP)

- Industrial extreme programming is defined as : "IXP is organic evolution of XP. It is customer-centric and filled with test – driven sprit." It differ most from the original XP in its greater inclusion of management, its expanded role for customers and its upgraded technical practices".
- IXP included six new practices that are designed to help XP projects.
  (i) Readiness assessment
  (ii) Project community
  (iii) Project chartering
  (iv) Test-driven management
  (v) Retrospectives
  (vi) Continuous learning

- Readiness as essment takes care that whether appropriate development environment exists to support IXP or not.
- The agile team should have the right candidates to ensure success.
- The IXP team itself will assess the project to determine the appropriate business justification.
- Test-driven management establishes a series of tests to find out the bugs and flaws.
- An IXP team conducts a specialized technical review after a software increment is delivered called retrospective.
- Continuous learning is essential as it is a vital part of continuous process improvement.

3.5.4 The XP Debate

- The critics have raised many issues with reference to XP practices and its evolution.
- Some of the issues discussed or debated are :
  - Requirements volatility.
  - Conflicting customer needs.
  - Requirements are expressed informally.
  - Lack of formal design.
- Due to the active involvement of customer in the agile team, changes are requested in an informal manner. This leads to most of the problems in the development phase. This is called as requirement volatility.
- Since in many projects, there are multiple customers with their own set of needs. This leads to conflicts in customer's need.
- Since requirements are expressed informally, it is difficult to build a system which should not have the inconsistencies.
- The extreme programming always prefers that design of all kinds should be informal. Due to this the design of complex systems can not ensure quality and maintainability. So XP lacks formal design, which is very important.

3.6 SCRUM

University Questions

Q. Explain how scrum works with basic diagram. **SPPU : May 19, 5 Marks**

Q. Explain SCRUM with the help of diagram. **SPPU : Dec. 19, 5 Marks**

- It is one of the agile software development methods. It is an iterative and incremental software development framework. It gives holistic and flexible development strategies.
- It enables teams to self organize by online collaborations of team members. Its key principle is to recognize that during project development customers can change their requirement and requirements of customers are unpredictable.
- It adopts empirical approach. It assumes problem cannot be fully understood or defined but focusing on maximizing team ability to deliver quickly.
- Different terminologies used in SCRUM process is as follows :
  - **SCRUM team :** It contains product owner, development team and scrum master.
  - **Product owner :** Person responsible for product backlog.

- **SCRUM master :** Person responsible for scrum process.
- **Development team :** Team of people responsible for delivering the product.
- Sprint burn down chart : Daily sprint progress.
- **Release burn down chart :** Chart of completed product backlog.
- **Product backlog :** List of priority wise requirement.
- **Sprint backlog :** List of task to be completed which are prioritized.
- **Sprint :** Period in which development occurs.
- **Spike :** Period used to research concept
- **Tracer bullet :** It is spike with current architecture, technology, best practices, etc.
- **Tasks :** Items added to sprint backlog at the beginning of sprint which are broken into hours.
- **Definition of done :** Exit criteria which decides product backlog items are completed or not.
- **Velocity :** Total efforts a team is capable of in a sprint.
- **Scrum-but :** It is exception to scrum methodology.

3.6.1 Process Flow

SCRUM process flow contains different stages. Different stages are as follows :

- **Setup**

  Setup environment contains the following :
  - Remove any customization from existing application.
  - Activate scrum process pack plug-in.
  - Assign scrum roles to users.
- **Create a product**
  - Product represents the functionality which is identified by owner.
  - Product contains different features which are needed for enhancement which are useful for customer satisfaction.
  - It can have few features or many features.
- **Create user stories**
  - These are product requirement created by product owner.
  - User stories are written in plain language. Less technical jargons are used.
  - It contains specific requirement that product should have.
  - Stories cannot be created without associating with product.
- **Create release**
  - It has start and end date in which number of development iterations are completed.
- **Create sprint**
  - It is basic unit of time in development process.
  - It can have any length. Typically it takes one to four weeks to complete.

- Scrum master creates one or more sprints.
- Sprints should release in given start and end dates.
- Scrum team need to complete all the stories which are committed.
- Scrum master expects all the stories are fully implemented and ready to release.

- **Plan sprint**
  - Team and scrum master need to decide on which stories they can commit to complete within a sprint.
  - Scrum master make sure that capability of sprint team matches the requirement of stories.
  - If capacity of the stories exceeds then scrum master add team members, remove stories or add sprint as and when needed.
  - Velocity chart is used in estimation process.
  - Velocity chart shows the historical record of number of completed points.
  - With the help of velocity chart scrum master get the idea of capacity of team.
  - Velocity chart is important tool in planning sprint.
- **Track sprint progress**
  - Scrum master is responsible for checking the progress.
  - He provides the progress report of the team.
  - Team members frequently update task and records.
- **Generate charts**
  - Progress of a sprint can be tracked with the help of different charts.
  - Chart is one of the important tools of scrum team.
  - Burn down chart compares ideal progress in sprint against the actual progress.
  - It is done on daily basis.

3.6.2 Scrum Roles

University Question

Q. Write and explain the role of the scrum master **SPPU : May 19, 5 Marks**

- Different roles are used in scrum process.
- Three different roles are as follows :

1. Product owner
2. Development team
3. Scrum master

1. **Product owner**
   - They are stakeholders of the customers.
   - They ensure teams delivers values to business.
   - They write customer centric items, rank customer centric item and add to product backlog.

- The scrum process needs to be one product owner. Sometimes they are part of development team.
- Role of the product owners in product requirement are as follows :
- Important role of product owner is communication with development team.
  - Identifying important requirement and communicating is important task of product owner.
  - They generally reduce the communication gap between team and stakeholders.
  - They demonstrate the solution to stakeholders.
  - They announce the different release of the product.
  - They communicate the team status.
  - Organizes the milestone review.
  - They educate the stakeholders in development process.
  - They negotiate priorities, scope, funding and schedule.

2. **Development team**
   - They are responsible for delivering the product.
   - Team generally consists of people of different skills.
   - Team size can be from between 3 to 9.
   - They includes member like analyst, designer, developer, tester, technical communicator, document writing, etc.
   - Development team is self organizing.
3. **Scrum master**
   - Scrum is facilitated by scrum master.
   - He is accountable for product goals and deliverables.
   - He is not a team lead or manager. He acts as a buffer between development team and distracting influences.
   - He is the enforcer of the scrum process.
   - He generally involves in key meeting and challenges the team to improve.
   - Project manager is responsible for people management but scrum master is not related to people management.

3.6.3 Scrum Cycle Description

- Scrum cycle is divided into different activities.
- Scrum cycle activities are as follows :

1. Define	2. Plan	3. Build
4. Review	5. Retrospect	

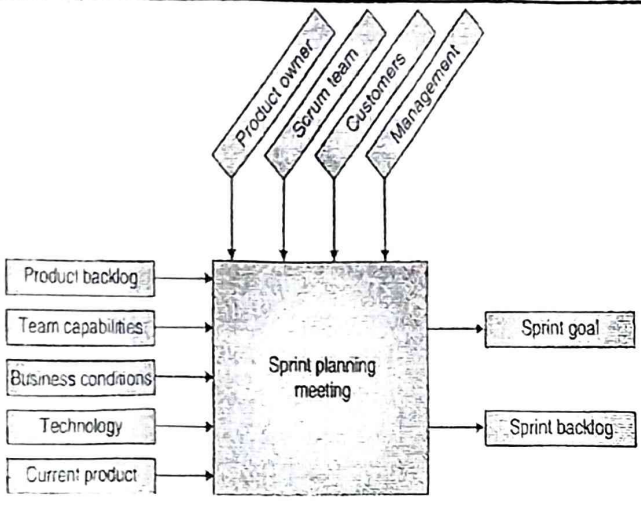


Fig. 3.6.2 : Sprint planning meeting

3.6.6 Sprint Backlog

- It is nothing but list of work development team need to complete in the sprint.
- This list is derived from product backlog.
- It contains list of user stories in sprint in the order of priority.
- It also contains the relative effort estimate.
- The task needed to develop every story is also maintained in the sprint backlog.
- In sprint planning meeting team selects some number of product backlog items.
- Product backlog items are selected in user story format.
- They also identifies user stories need to complete.
- Size of sprint backlog is selected by team.
- Sprint backlog can be maintained using spreadsheet.
- Example of sprint backlog is shown below.

Table 3.6.1

User Story	Tasks	Day 1	Day 2	Day 3	Day 4	Day 5	....
As a member, I can read profiles of other members so that I can find someone to date.	Code the ...	8	4	8	0		
	Design the ...	16	12	10	4		
	Meet with Mary about...	8	16	16	11		
	Design the UI	12	6	0	0		
	Automate tests...	4	4	1	0		
	Code the other ...	8	8	8	8		
As a member, I can update my billing information	Update security tests	6	6	4	0		
	Design a solution to ...	12	6	0	0		

User Story	Tasks	Day 1	Day 2	Day 3	Day 4	Day 5	...
	Write test plan	8	8	4	0		
	Automate tests	12	12	10	6		
	Code the ...	8	8	8	4		

3.6.7 Sprint Execution

- Sprint is basic unit of development in scrum development.
- Scrum make progress is series of sprint.
- During the sprint every day project status meeting occurs. This meeting is called as daily scrum meeting.
- Sprint execution is nothing but the work which is performed by scrum team to meet sprint goals.
- All the work which is necessary to deliver is performed.
- It accounts for majority of time during a sprint.
- It begins after sprint planning.
- It ends when sprint review starts.

3.6.8 Daily Scrum Meeting

- Every day during sprint project meeting occurs.
- This meeting is called as daily scrum meeting.
- It is helpful for schedule coming day's work.
- Scrum meeting has following guidelines :
  - All members presents for meeting with updates.
  - Usually meeting starts on time even if some members are absent.
  - Usual meeting time is at the morning.
  - Meeting time and venue are same every day.
  - Approximate meeting time is 15 minutes.
  - Generally core team member speak in the meeting.
- Different questions are answered by team. Some of the questions are as follows :
  - What have been done since yesterday?
  - What is today's planning?
  - Any impediments or stumbling blocks?
- Sample scrum meeting is shown below :

Table 3.6.2

Monday	Tuesday	Wednesday	Thursday	Friday
		Sprint 1 planning 1 pm. - 5 pm.	Daily scrum 9:15 - 9:30	Day scrum 9:15 - 0:30

Monday	Tuesday	Wednesday	Thursday	Friday
Daily scrum 9:15 - 9:30	Daily scrum 9:15 - 9:30 Backlog grooming 1 pm - 2 pm	Daily scrum 9:15 - 9:30	Daily scrum 9:15 - 9:30 Backlog grooming 1 pm - 2 pm	Daily scrum 9:15 - 9:30
Daily scrum 9:15 - 9:30	Daily scrum 9:15 - 9:30 Dry run and prep for sprint review (optional) 2 pm - 3 pm	Daily scrum 9:15 - 9:30 Sprint 1 review 10 am - 11 am Sprint 1 retrospective 11 am - 12 pm Celebration lunch 12 pm - 1 pm Sprint 2 planning 1 pm - 5 pm		

3.6.9 Maintaining Sprint Backlog and Burn-Down Chart

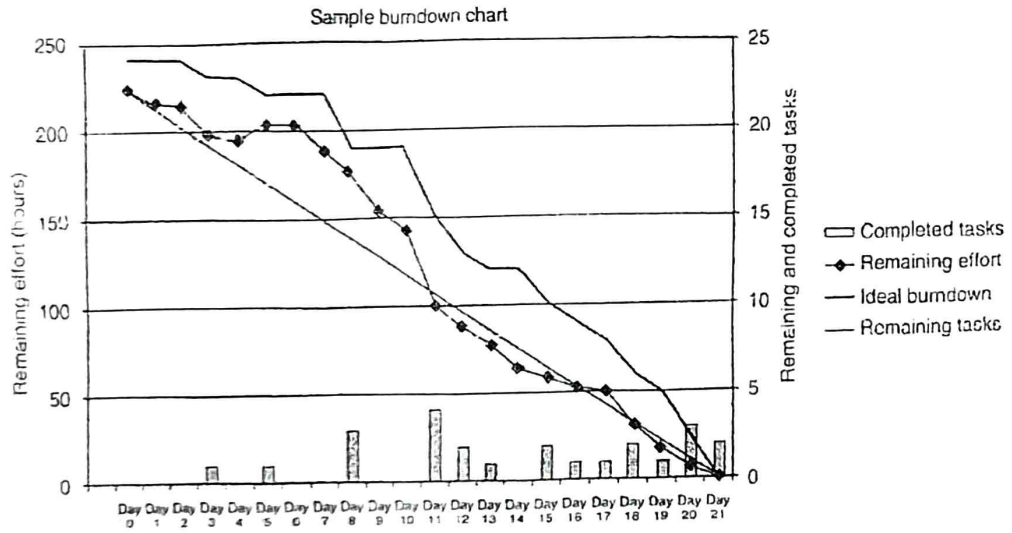


Fig. 3.6.3 : burn down chart

- Burn down is graphical representation of work left versus time.
- Pending work is on vertical axis and time is at horizontal axis.
- It is nothing but the run chart of outstanding work.
- It is useful in predicting when the work will be completed.
- It is used in agile software development particularly in scrum process.
- It can be applicable to any project.
- Sample burn down chart is shown in Fig. 3.6.3.

3.6.10 Sprint Review and Retrospective

- These meetings are held at the end of sprint cycle.
- These meetings are sprint review and retrospective meeting.
- In the sprint review meeting following points are discussed :
  - Review the completed work.
  - Review incomplete work.
  - Show the completed work to stakeholders.
  - Incomplete work not demonstrated.
  - Meeting is limited to four hours.
- In the sprint retrospective following points are considered :
  - Every team member reflects on past sprint.
  - Make continuous process improvement.
  - Two questions are asked in the sprint. One question is : What is well in the current sprint? What improvements can be done in next sprint?
  - The time limit for the sprint retrospective meeting is 3 hours.
  - This meeting is head by scrum master.

3.7 Agile Practices

Agile development is supported by number of practices. It includes the different area like requirement analysis, design, coding, testing, project management, process, quality, etc. Some of the important agile practices are as follows :

1. Pair Programming
2. Refactoring
3. Test Driven Development
4. Continuous Integration

3.7.1 Pair Programming

University Questions

Q. Describe XP concept of pair programming.	SPPU : Dec. 15, 5 Marks
Q. What is pair programming? What is role of pair programming in XP?	SPPU : Dec. 19, 5 Marks
Q. Write short note on : Pair Programming.	SPPU : May 19, 5 Marks

- It is one of agile practice is agile software development.
- In this method two programmer work together on one machine.
- One programmer types the code. Another programmer observes point and suggests the changes.
- Sometimes programmer can switch their role to each other.
- Observer programmer can suggest improvement in code.

- With this method programmer who writes the code can concentrate on efficient coding. Observer programming can think and suggest improvements.

Benefits of pair programming

There are numerous advantages of pair programming.

- **Economy :** It spends around 15% more time on programming. But designed code is efficient than that of individual coding. It also improves development time and support cost.
- **Design quality :** With pair design more ideas can be generated. As two programmers are working together we are able to check all possibilities during design. This can be accomplished as different programmer bring their prior experience together. They think differently and able to generate new ideas.
- **Satisfaction :** With pair programming programmer can enjoy work than that of individual programming. They are more confident in their solutions. Their confidence results in successful software.
- **Learning :** Ideas are shared among the programmer. They are able to learn from each other. They are able to share their knowledge with each other. It allows programmer to observe programming code and give the feedback accordingly.
- **Team building and communication :** It allows to share the problem with each other and able to find the solution on it. With this practice communication among team member increase with benefits in greater team building.

Remote pair programming

- It is part of pair programming.
- It allows programmer to be geographically apart from each other but connected through the network.
- It is also called as virtual pair programming or distributed pair programming.
- They generally work using shared editors, shared desktop, remote pair programming tools.
- Sometimes this method creates the problems which are not present in face to face programming.
- Some of the problem is remote pair programming are delay in communication, communication link failure problem, lack of coordination, etc.
- Different tools are provided for remote pair programming. Some of the tools are as follows :
  - Microsoft Lync
  - VNC
  - Screenhero
  - Skype
  - Terminal multiplexers (tmux a, screen -x)
  - Gobby
  - Saros Xpairtise
  - Floobits
  - Visual studio anywhere
  - Cloud

3.7.2 Refactoring

University Questions

Q. Describe XP concepts of refactoring. **SPPU : Dec. 15, 5 Marks**

Q. Write short note on refactoring. **SPPU : Dec. 16, 5 Marks**

- It is one of the agile practices in agile software development.
- It is process of restructuring the existing code. The changes are made without affecting external behaviour.
- It does not change the functional requirement of the code.
- It changes the non-functional requirement of the code. Non functional requirements are useful in efficiency, performance, throughput, etc.
- It has numerous advantages like improved readability, reducing complexity, reusability, etc.
- It results in creating more expressive software architecture which results in extensibility.
- Generally series of standards are applied for code refactoring.
- Some standards are used which are designed by organization and some other standards also used in code refactoring.
- They are generally identified by code smell.
- E.g. we have code which is very long or it is almost duplicate of another code. In this situation code refactoring can be applied where we can remove the code duplication. After refactoring the basic functionality of the code remain same.
- Code duplication can be removed by designing shared function. Shared by will be used whenever required instead of creating same copy of code again and again.
- Two main objectives of refactoring are maintainability and extensibility.
- With code maintainability, we can easily identify the bugs in the software as the code is properly documented.
- With code extensibility, new features can be added very easily.
- We need automatic unit test before applying code refactoring.

List of refactoring techniques

- Some of famous refactoring techniques we will discuss in this topic.
- Some techniques are applied to only certain languages or situations.
- Many development environments provide code refactoring.
- Some of the techniques are as follows.

1. **Techniques allowing more abstraction**
   - Code encapsulation where different fields can be accessed with getter and setter methods.
   - Code sharing can be achieved by generalizing types. If we make generalize code then chances of sharing that code increases.

- Replace type checking code with states
- Use polymorphism.

2. **Techniques for breaking code apart into more logical pieces**
   - Break the code into different modules. Modules increase the code reusability.
   - Extracting class results in moving code to new code.
   - Extracting methods results into moving methods to new methods.
3. **Techniques for improving name and location of code**
   - Move methods or field to appropriate class.
   - Change the name of methods or fields to user friendly name. Name should signify the purpose of method or variable.
   - Use concept of superclass from object oriented programming.
   - Use concept of subclass from object oriented programming.

Automated code refactoring

Some editors and IDE supports automated refactoring. Some of the software are listed below.

- IntelliJ IDEA
- Netbeans
- ReSharper
- Photran
- WebStorm
- JDeveloper
- Code rush
- Xcode
- Eclipse
- Visual studio
- Visual assist
- AppCode

3.7.3 Test Driven Development (TDD)

University Question

Q. Write short note on : Test driven development. **SPPU : May 19, 4 Marks**

- It is one of the agile practices in agile development.
- This software development process relies on repetition of short development cycles.
- Changes in software are quick and easy.
- Short development cycle includes writing automated test case which defines desired improvement or new function. Minimum coding is done to pass the test. Afterwards code are redesigned as per the standard.
- It is related to extreme programming.
- Test driven development is shown in Fig. 3.7.1

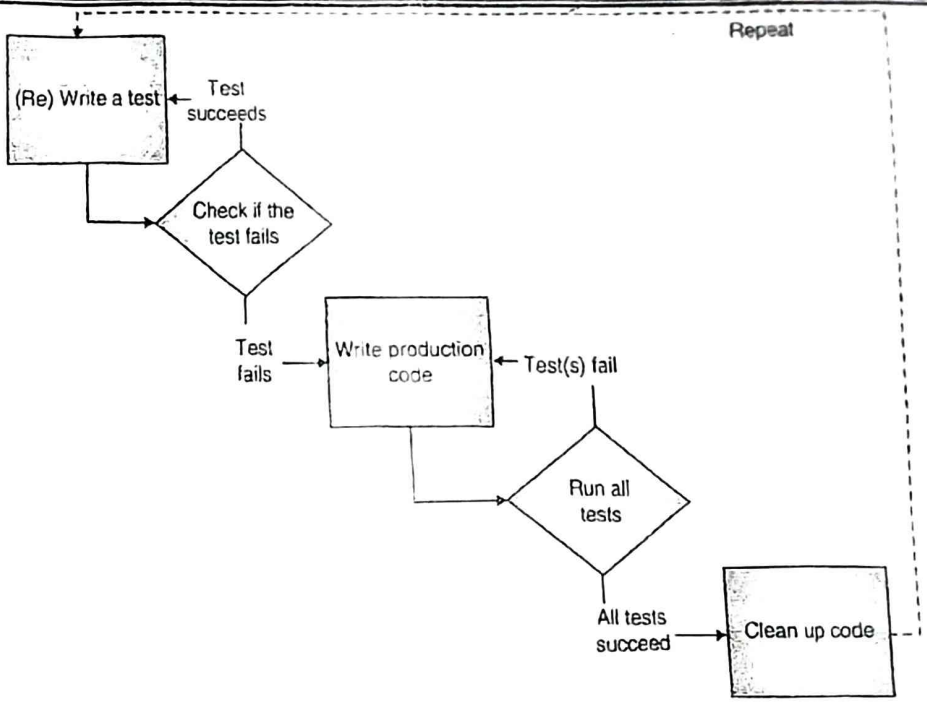


Fig. 3.7.1 : Test driven development

- Different phases of test driven development are as follows.

(a) Create test

- In test driven development new feature can be added by creating test.
- This test should get fail because this feature was not added previously.
- Before writing test, before should understand the test properly then only test should be added.
- Requirements can be understood using different techniques like use cases and user stories.

(b) Run test and see it fails

- In this phase we run the created test and check if it running properly or not.
- Test should get fail so that we can write code for it.

(c) Writing some code

- In this phase we write code for test to get pass.
- The code written is not perfect but it is useful to pass the test for time being.
- Afterwards this code will be improved.
- Purpose of this code is just for passing the test and not for permanent purpose.

(d) Run test

We run the test on created code.

(e) Refactor code

- Earlier we have written code which was not final.
- In this phase, code will improved and made efficient to fulfill all the coding standard.

(f) Repeat

- The process is repeated for another test also.
- Size of steps should be small.
- If the test fails then programmer need to undo the changes.

3.7.4 Continuous Integration in DevOps

- A set of practices that combines software development (Dev) and IT operations (Ops). It aims to shorten the systems development life cycle and provide continuous delivery with high software quality.
- DevOps is complementary with agile software development; several DevOps aspects came from the Agile Methodology.
- It is method of merging all developer copies of work together frequently.
- It is part of extreme programming.
- Main aim is to prevent integration problems.
- It is useful in iterative software development.
- Earlier it was used in coordination with test driven development.

Principles of continuous integration

- **Maintain code repository :** All the project source code should be properly stored in code repository. It is useful in revision control system.
- **Automate the build :** There should be facility for building the integration automatically. Single command should able to do all the activities. Different build tools like make can be used. Automated build includes integration.
- **Make build self testing :** After making the integration software should work as per specification. Testing facility after integration should be automated one.
- **Everyone commits to baseline every day :** With this facility every committer can reduce conflicting changes.
- **Every commit should be built :** system should build commits to current working version.
- **Keep the build fast :** Building process should be rapid.
- Other principles are as follows.
  - Test in clone of the production environment
  - Make it easy to get the latest deliverables
  - Everyone can see the results of the latest build
  - Automate deployment

Advantages

It has many advantages which are as follows.

- If unit test fails, developer can revert the code to bug free state of the code.
- Developers can detect and fix integration problem continuously.
- Early warning of broken or incomplete code.
- They can give early warning of conflicting changes.
- Code is available constantly.

- Immediate feedback can be given to developers
- Creating modules can be possible with continuous integration.

Software tools which supports continuous integration

- Apache Continuum
- Apache Gump
- Automated QA Automated Build Studio
- At lassian Software Systems Bamboo
- Build bot
- Build Hive
- CABIE
- Cascade
- Cruise Control
- Cruise Control.rb
- CruiseControl.NET
- Electric Cloud
- Final Builder Server
- Hudson
- Jenkins
- IBM Rational Team Concert
- IBM Rational Software SCLM
- TeamCity
- TinderboxTravis CI
- Xcode 5

3.7.5 Exploratory Testing Versus Scripted Testing

University Questions

Q. Compare scripted testing verses exploratory testing. **SPPU : Dec. 14, 5 Marks**

Q. Differentiate between exploratory testing and scripted testing. **SPPU : May 16, Dec. 16, 5 Marks**

- Scripted testing considers two roles namely test designer and tester. Test designer is usually designs the test cases. He is high skilled specialist. Tester executes the test case which is designed by test designer.
- In exploratory testing different roles are not considered. Test designing and actual testing is done by same person. Tester designs the test cases and executes it. According to previous results he may create more test case and execute it.
- Learning, test design and execution all are done by tester in exploratory testing.
- In script testing team can be formed of different employees where one of the members would be very high skilled one and other members can beginners. Cost of testing can be saved with script testing.
- In exploratory testing depends upon skill of the employees which results in high project cost.
- With scripted testing high planning is possible. We can predict for desired outcome with scripted testing. As scenarios are ready we execute the test cases precisely.
- In exploratory testing planning is impossible or very difficult. We cannot guarantee all the test cases will be executed or not.
- With scripted testing well documentation can be created. Test designer can document the test scenarios whereas tester can records the status of executed test With exploratory very less documentation is used.
- Exploratory testing is flexible whereas script testing is not.
- If some changes happen in the requirement then test designer need to change the scenarios. When scenarios are changed then only tester can executes the test cases.
- Exploratory testing is interesting than that of scripted testing.

Review Questions

Q. What are the different agility principles?

Q. Explain Pair programming in the context of agile software development.

Q. What are various benefits of pair programming?

Q. What do you mean by Remote pair programming?

Q. Explain Refactoring in the context of agile software development. What are different refactoring techniques ?

Q. What are different editors and IDE that supports automated refactoring?

Q. Write short note on test driven development.

Q. Explain test driven development (TDD) with block diagram. What are different phases of TDD?

Q. Explain Continuous integration in the context of agile software development. What are various principles of Continuous integration?

Q. List different advantages of Continuous integration.

Q. List Software tools which supports continuous integration.

Q. Compare Exploratory testing with scripted testing in the context of agile software development.

❑❑❑

4

Unit - II

REQUIREMENTS ENGINEERING

Syllabus

User and system requirements, Functional and non-functional requirements, requirements engineering (elicitation, specification, validation, negotiation) prioritizing requirements (Kano diagram), requirement traceability matrix (RTM).

4.1 User and System Requirements

- The fundamental definition of the requirements for a system is :
- "The set of descriptions listed for the system to accomplish the required services and the constraints on its operation."
- These requirements reflect the needs of customers for a system. The process of identifying the customers needs, analyzing, documenting and checking the services and constraints that a system should provide, is called requirements engineering (RE).
- Most of the time, the term 'Requirement' is not used in software organizations. But the requirement is illustrated as 'high-level, abstract statement of services that a system should provide for its users'.
- The problem exists when we talk about distinction between different levels of description. Primarily the requirements specification can divided into two important classes as follows :
  - User requirements
  - System requirements
- The **User requirements** are the statements narrated by the customer in simple natural language and with some useful diagrams.
- The customer describes what services the system should provide and the constraints on the operations and functions of the system.
- **The System requirements** are more detailed descriptions of the software system's functions, services, and operational constraints.
- The system requirements document also called as functional specification, should define exactly what is to be implemented. It is also the contract between the system buyer and the software developers. All these requirements are well documented on paper for future references.
- Following figure exhibits the readers of different types of requirements specification :

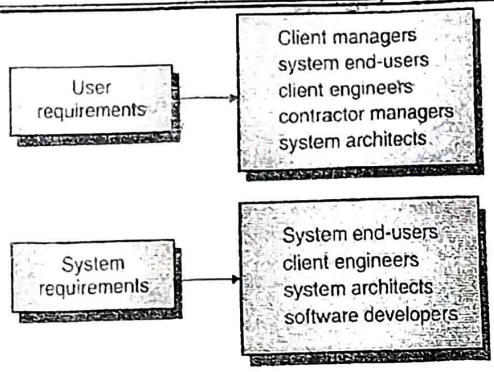


Fig. 4.1.1 Readers of different types of requirements specification

- The readers of the **user requirements** are interested only in identifying detailed requirements from the user that what facilities a system should provide to its end users.
- The readers of the **System requirements** need to know more precisely what the system will do because they are concerned with how it will support the business processes or because they are involved in the system implementation.

4.1.1 Importance of Requirement Engineering

University Question

Q. Explain the importance of Requirement Engineering. **SPPU : May 19, 5 Marks**

- The Requirement Engineering (RE) is the most important phase of the Software Development Life Cycle (SDLC). This phase is used to translate the imprecise, incomplete needs and wishes of the potential users of software into complete, precise and formal specifications. The specifications act as the contract between the software users and the developers.
- Therefore the importance of Requirement Engineering is enormous to develop effective software and in reducing software errors at the early stage of the development of software. Since Requirement Engineering (RE) has great role in different stages of the SDLC, its consideration in software development is crucial. There exist a number of approaches for requirement engineering.

4.2 Functional and Non-functional Requirements

University Question

Q. What are functional and non functional requirements of software? **SPPU : Feb. 15, May 16, 7 Marks**

- **The system requirements are classified into following two types :**
  - **Functional requirements**
  - **Non-functional requirements**
- **The functional requirements are the statements of the services that a system should provide and how the system must react to a given input and the system reacts in some situations. The functional requirement should also mention that a system should do nothing in some situations.**

- The non-functional requirements are treated as some constraints that a system should behave in some situations. Also the non-functional requirements are the explicit conditions that are put on the services and the functionalities of the system to behave. The non-functional requirements are applied on the whole system and not on the individual system features.

4.2.1 Functional Requirements

- Actually functional requirements state that how a system should behave and to work in a given situation. The functional requirements are often depended on the following parameters :
  - Type of the software under construction,
  - The users of the software,
  - The approach of writing the requirements by the organization.
- The functional requirements are always written in such a style that it is easily understood by all type of users.
- Some of the specific functional requirements give the expected input, desired output and the system functions.
- The functional requirements also specify the facilities provided by the software system. The functional requirements may have different levels of details. The functional requirements must be complete and consistent in nature.
- The complete functional requirement means it should define all the services that are needed by all categories of the users. And consistent means that the functional requirements should not have any contradictory statement i.e. for one requirement there should not be two definitions.
- But in larger systems and complex systems it is highly impossible to maintain completeness and consistency.

4.2.2 Non-functional Requirements

University Question

Q. Explain about various categories of non functional requirements & theirimportance. **SPPU : May 19, 5 Marks**

- The non-functional requirements are not directly related to the functions, services of the system and the desired outputs.
- The non-functional requirements are related to the system properties that are listed below :
  - Reliability
  - Speed of responses i.e. response time,
  - System performance,
  - System security,
  - System availability,
  - Portability,
  - Robustness,
  - Ease of use etc.
- In contrast to functional requirements, the non-functional requirements are more critical. Therefore, if functional requirements are failed then the whole system may be failed and become unusable.

- For example if a air system does not provide reliability, then it can not be used in actual practice. It will not be called as a safe system.
- The failure of an individual non-functional requirement may lead to failure of the whole system.

4.3 Introduction to Requirements Engineering

University Question

Q. What is requirements engineering ? **SPPU : May 15, 5 Marks**

- Requirements engineering helps software engineers and software developers to better understand the problem and the nature of the problem they are going to work on. It includes the set of tasks that lead to understanding of :
  - What will be business impact of the software ?
  - What the customer wants exactly ?
  - How end user will interact with the system ?
- Software engineers (sometimes also called as system engineer or analyst) and other project stakeholders (managers, customers and users), all participate in requirements engineering.
- Designing and building the elegant computer software will not satisfy customer's requirements. If requirements analysis is wrong, then design will be wrong.
- Ultimately coding will be wrong. Finally, software expectations will not match with outcomes. Hence requirements engineering should be carried out carefully.

4.4 Requirements Engineering

University Question

Q. Explain in detail requirement engineering task. **SPPU : Dec. 12, 3 Marks**

- The requirements engineering is carried out through the execution of following functions. Some of these requirements engineering functions occur in parallel.
- All these seven functions focus on the customer's needs and care must be taken to satisfy it. All these functions collectively form the strong base for software design and construction. These functions are :

1. Inception
2. Elicitation
3. Elaboration
4. Negotiation
5. Specification
6. Validation
7. Requirement management

4.4.1 Elicitation

University Question

Q. What is requirement elicitation? **SPPU : May 19, 5 Marks**

- Elicitation is a task that helps the customer to define what is required. 'Eliciting requirements' step is carried out by series of following steps :

1. Collaborative requirements gathering
2. Quality function deployment
3. User scenarios
4. Elicitation work product

4.4.1(A) Collaborative Requirements Gathering

- Gathering software requirements is team oriented activity. All the software team members, software engineering manager, members of marketing, and product engineering representatives all work together. The aim of this activity is :
  - To identify the problem.
  - To suggest the solution.
  - To negotiate different approaches.
  - To specify the preliminary set of solution requirements.
- The meeting for 'collaborative requirements gathering' is conducted to discuss all above issues.
- The basic guidelines for conducting a collaborative requirements gathering meeting are :
  - Meeting is conducted and attended by both software engineers as well as customers.
  - An agenda is suggested that is formal enough to cover all points those are to be discussed in the meeting.
  - A 'facilitator' may be customer, developer or outsider controls the meeting.
  - A definition mechanism including work sheets, charts, wall stickers, chat rooms, projectors is used.

4.4.1(B) Quality Function Deployment

- Quality function deployment is a technique that translates the customer's needs into technical requirements for software.
- In other words **'Quality Function Deployment'** defines the requirements in a way that maximizes customer satisfaction. Quality function deployment includes three types of requirements :

1. Normal requirements
2. Expected requirements
3. Exciting requirements

1. Normal requirements

University Question

Q. What is meant by normal requirements **SPPU : May 12, May 14, May 19, 3 Marks**

These are the requirements clearly stated by the customer. Hence these requirements must be present for customer's satisfaction.

For example

- Graphic displays.

- Specific system functions.
- Specified output formats.

2. **Expected requirements**

These requirements are implicit type of requirements. These requirements are not clearly stated by the customer but even then the customer expects them.

For example

- The developed system must provide easy human machine interaction.
- The system should be menu driven,
- All hot key buttons help should be provided.
- The system should be 'user friendly'.
- The system should be easy to install.

3. **Exciting requirements**

University Question

Q. What is meant by exciting requirements. **SPPU : May 12, May 14, May 19, 3 Marks**

These requirements are neither stated by the customer nor expected. But to make the customer more satisfied, the developer may include some unexpected requirements. For example, in word processing software development, only standard capabilities are expected. But, it will be a surprise for the customer if 'page layout capabilities' and 'advanced graphical features' are added.

4.4.1(C) Usage Scenarios

- It is just impossible to move into more technical software engineering activities until the software team understands how these functions and features will be used by different classes and end users.
- To understand this, the developer and users create a set of scenarios those will identify all these issues. The scenario is called as **'Use Cases'**. Details of Use Case diagrams are included in next chapter.

4.4.1(D) Elicitation Work Product

University Question

Q. Write the elicitation work products. **SPPU : May 19, 5 Marks**

- The work products produced by requirement elicitation depend upon the size of the system or the system to be built.
- The information produced as a consequence of requirements gathering includes :
  - A statement of need and feasibility.
  - A statement of scope for the system or product.
  - A list of customers, users and other stakeholders who participated in requirement elicitation.
  - Description of system's technical environment.
  - The list of requirements and domain constraints.
  - The set of scenarios.
  - Any prototype developed to define requirements clearly.

4.4.1(E) Elicitation Techniques

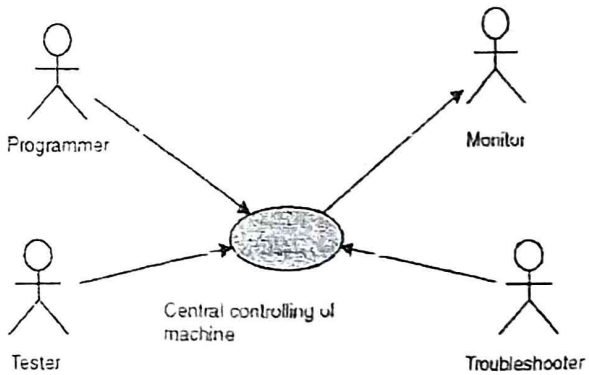
- These are the data collection techniques.
- They are used in cognitive science, knowledge engineering, linguistic management, psychology, etc.
- Knowledge is directly acquired through human being.
- It includes various techniques including interview, observations, participatory design, focus groups, etc.

4.4.1(F) Developing Use Cases

- A Use case exhibits the behaviour of the system according to the response received from any of the stakeholders. Alternatively a use case explains how the users will operate the system under any specific circumstances.
- Use cases are defined from actor's point of view. An actor is a role that refers the user or device that interacts with the software.
- The Use case diagram shows the relationship between actors (e.g. person, machine, another system etc) and use cases (sequence of actions i.e. procedures /functions).
- Note that the actor and end user are not necessarily the same thing. A typical user may play number of different roles when using the system, whereas an actor represents a class of external entities that plays just one role in the context of use case.

For example

- Consider the application where the machine operator handles control computer for manufacturing cell. Here, machine operator is a user.



Actors and use case for controlling the machine in manufacturing cell of factory

Fig. 4.4.1

- The software for control machine requires four different modes for interaction : Programming mode, test mode, monitoring mode, troubleshooting mode.
- Hence four actors can be defined : programmer, tester, monitor and trouble shooter.
- **There are two types of actors**

1. Primary actors
2. Secondary actors

1. **Primary actors**

   These actors interact with the system to achieve required system function and derive intended benefits from the system. They work directly with the software.

2. **Secondary actors**

   - These actors support the system so that primary actors can do their work. After identifying the actors the use cases can be developed. Jacobson suggests the number of questions those are answered by the use cases.
   - What are primary and secondary actors ?
   - What are the goals of actors (i.e. primary and secondary actors) ?
   - What are the preconditions that exist before the development begins ?
   - What are tasks and functions that are performed by actors ?
   - What are considerable exceptions ?
   - What are the possible variations in primary and secondary actor's interaction ?
   - What are the system information that a actor can acquire, produce ?
   - What are the system information that a actor can modify ?
   - Is it necessary for a actor to inform the system, the possible changes in the external environment ?
   - What is the desired information of a system that an actor want to know ?
   - Is it necessary for a system to inform the actor about unexpected changes ?

4.2 Specification

Throughout the analysis model, software engineer focuses **'what'** type of queries.

For example

- What is the information processed by the system ?
- What functions the system performs ?
- What is the behaviour of the system ?
- What interfaces the system defines ?

The **'analysis model'** acts as a bridge between **'system description'** and the 'design model'. After considering the requirements at system level in 'system description' the analysis model focuses on the system requirements.

The information, functions and behaviour of the system defined by 'analysis model' are translated into architectural, interface and component level designs in 'design modeling'.

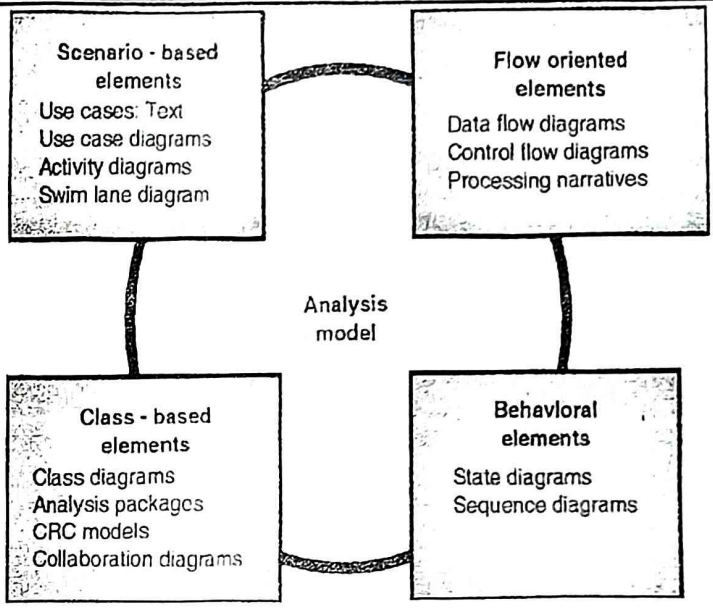


Fig. 4.4.2 : Elements of analysis model

Elements of Analysis Model

The elements of analysis model include :

1. Scenario-based elements
2. Flow oriented elements
3. Class based Elements
4. Behavioural elements

1. **Scenario-based elements**

They represent the system in user's point of view. Scenario-based elements are :

- Use case - Text
- Use case diagrams
- Activity diagrams
- Swim lane diagrams

2. **Flow oriented elements**

They provide necessary information that how data objects are transformed by processing the functions. Flow oriented elements are :

- Data flow diagrams
- Control Flow diagrams
- Processing narratives

3. **Class based elements**

They define the objects, attributes and relationships. Class based Elements are :

- Class diagrams
- Analysis diagrams

- CRC models
- Collaboration diagrams

4. **Behavioural elements**

They show the states and it is classes and impact of the events on these states. Class based elements are :

- State diagrams
- Sequence diagrams

4.4.2(A) Requirement Monitoring

- All the requirements should be monitored properly.
- Proper document should be created during collecting requirement.
- Developers need to check the requirement frequently and accordingly implement it.
- There are many chances that requirement changes frequently.
- We need to monitor all the requirement.

4.4.3 Validation

University Question

Q. How requirements are validated? **SPPU : May-12, May-14, May-19, 3 Marks**

- Once requirement model is built, it is checked for inconsistencies and ambiguities. Then the requirements are optimized.
- During the review of the requirements model, following questions arise :
  - Are all the requirements consistent ?
  - Whether requirements followed proper level of abstraction ?
  - Is the requirement really necessary ?
  - Are there any requirements which conflicts with one another ?
  - Is each requirement testable, once implemented ?
  - Does the requirement model properly reflect the information, function and behaviour of the system to be built ?
  - Does the requirements model use the requirements pattern to simplify the model ?
- These questions are answered to make sure that the requirements model is as per the customer's exact needs.

4.4.4 Negotiation

- In traditional development process, the requirement engineering has the following phases :
  - Inception
  - Elicitation
  - Elaboration
- These phases are enough to determine the customer's requirement. But, practically it rarely happens that all the requirements are met as per customer's need. In actual practice, one has to go for negotiation with one or more stakeholders.

- The importance of negotiation is to develop a project plan that meets customer's needs, also keeping in mind the other factors like time, people, budget etc.
- The best negotiation yields "win-win" result.
- Boehm has defined a set of negotiation activities at the beginning of each software process iteration :
  - Identification of the system or subsystem's key stakeholders.
  - Determination of the stakeholders' "wins conditions".
  - Negotiation of stakeholders' "wins conditions".
- After successful completion of these initial steps, we achieve a win-win result. Thus we can proceed to next software engineering activities.

4.5 Prioritizing Requirements (Kano diagram)

University Questions

Q. How to prioritize software requirements based on Kano Analysis? **SPPU : Dec. 15, May 19, 5 Marks**

Q. Explain Requirements Prioritization with the help of KANO diagram. **SPPU : Dec. 16, 5 Marks**

- This analysis allows us to prioritize the requirement as function of customer satisfaction.
- Kano defined four different methods for prioritizing the requirement.
- Different methods are as follows.
  - **Surprise and delight** : This capability differentiates the product from other product. More facility should be provided than that of competitor product.
  - **More is better :** More functionality should be provided.
  - **Must be:** functional barrier to entry without which user will not use the product.
  - **Better not be :** It represents things which dissatisfy customers.
- Kano diagram for prioritizing the requirement is as follows.

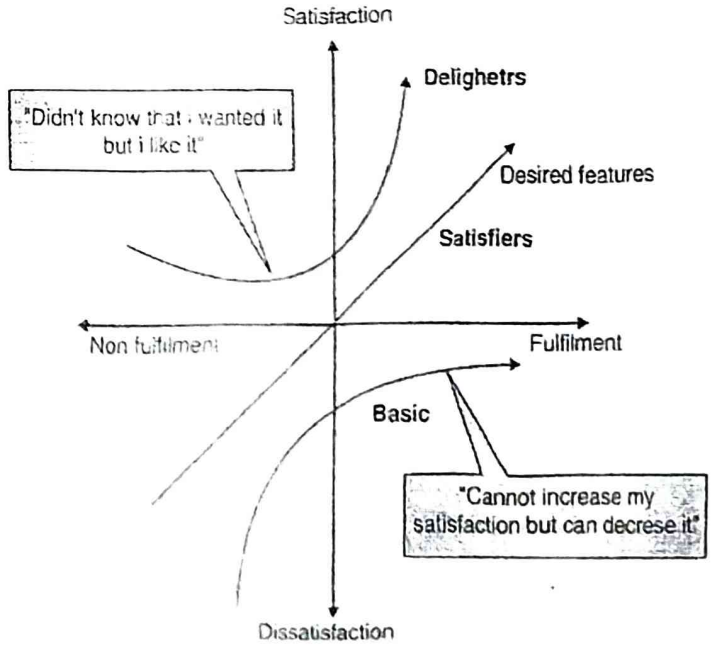


Fig. 4.5.1 : Kano model

4.6 Requirement Traceability Matrix - RTM (Requirement Management)

- Requirement management is a software engineering task that helps the project team members to identify the requirements, control the requirements, track the requirements and changes to requirements at any time as the project exceeds.
- The requirement management task starts with identification and each of the requirements is assigned a unique identifier.
- After the requirements finalization, the traceability table is developed. Traceability table relates the requirements to one or more aspects of the system or it is environment. The traceability tables are used as requirements database and are useful in understanding how change in one particular requirement will affect other parts or aspects of the system.
- Following are some examples of traceability table :
  - **Features traceability table** observes product features and make sure that how it is closely related to customer requirement.
  - **Source traceability table** is used to identify the source of each of the requirement.
  - **Dependency traceability table** observes the relationships among requirements.
  - **Subsystem traceability table** classifies the requirements as per the subsystem they govern.
  - **Interface traceability table** indicates the internal and external system interface relate as per the given requirement.

Requirement	Specific aspect of the system or its environment								
	A01	A02	A03	A04	A05				All
R01			✓		✓				
R02	✓		✓						
R03	✓			✓					✓
R04		✓			✓				
R05	✓	✓		✓					✓
Rnn	✓		✓						

Fig. 4.6.1 : Generic traceability table

4.7 Requirement Characteristic

University Question

Q. What are the characteristics that requirement must meet? **SPPU : Dec. 19, 6 Marks**

Following are characteristics that requirement must meet for any software development process :

- For effective testing, formal technical reviews must be conducted by the development team.
- Frequent communication between developer and customer resolves most of the complexities and confusion in the beginning itself. Thus before start of the testing process, number of errors may be uncovered and then fixed.

- Testing starts from the component level and then finally complete computer based system is integrated at the end.
- There are various testing techniques are available. So at different point of time, suitable testing technique may be applied.
- Testing may be conducted by the software developer itself for usually smaller projects. For the larger projects, an independent group(especially those people that are not the part of development team) may be hired for conducting an effective testing.
- The members of independent testing group may be the member of some other team or may be outsourced.
- Even though testing and debugging are two separate activities, debugging should be accommodated in testing strategies.
    - Any testing strategy should be able to conduct low level tests, since it verifies small source code modules and can be easily implemented.
    - It gives better precision. In addition to these low level tests, a testing strategy should also be able to conduct high level tests and it should be able to validate all the major functionalities as per customer's requirements.

Review Questions

Q. What are functional and non functional requirements of software?

Q. What do you mean by requirement inception?

Q. What do you mean by requirement negotiation?

Q. What are the work products of elicitation?

Q. Explain requirement analysis with example.

Q. What do you understand with the Validating requirements?

□□□

5

Unit - II

SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

Syllabus

Software requirements Specification document, structure of SRS, writing a SRS, structured SRS for online shopping.

5.1 Software Requirements Specification

- Basically SRS or software requirement specification is an official document or the statement of what the system developers should implement.
- It includes both :
  - System requirement.
  - User requirement.
- The SRS has a set of users like senior management of the organization, engineers responsible for developing the software.

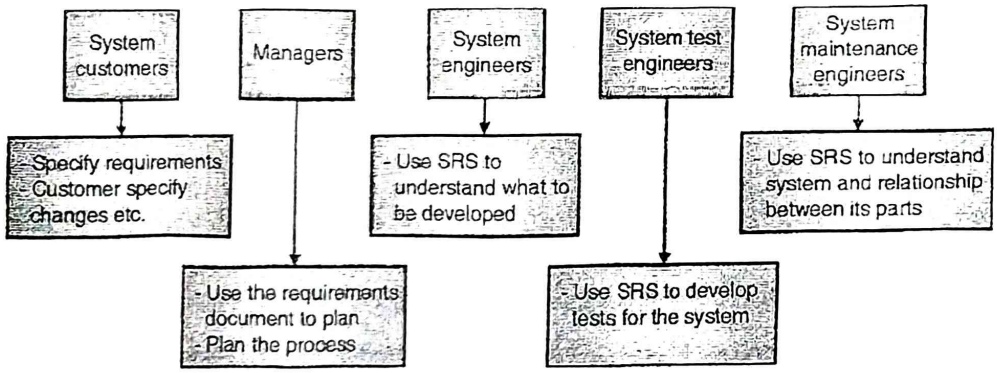


Fig. 5.1.1 : Users of SRS

- The details which are included in SRS depend on the type of system that is being developed and the type of the development process.
- A number of large organisations, such as IEEE have defined standards for software requirements document.
- The most widely used standard is IEEE/ANSI 830-1998. This standard suggests the following structure for requirements document :
  1. Introduction
     - Purpose of SRS

- Scope of product.
- Definitions, acronyms, abbreviations.
- References
- Overview

2. General description
   - Product perspective
   - Product Functions
   - User characteristics
   - General constraints
   - Assumptions and dependencies
3. Specific requirements
4. Appendices
5. Index.

- SRS is very useful when an outside contractor is developing the software system.
- For business system, where requirements are unstable, SRS play an important role.

5.2 Writing Software Requirements Specifications

- Proper software requirement gathering is very important at the start of the software development. Generally it is done by professional software developers.
- Document is created in the requirement analysis which is generally referred as SRS or software requirement specification document.
- It is first project deliverable.
- SRS records the end user needs in certain format. This is SRS is needed when actual software development process starts.

5.2.1 What is a Software Requirements Specification?

- Before starting software development requirement is captured from the end users or clients or customer. This requirement is written in certain format which is called as SRS. Generally this document is created before starting the development work.
- It signifies both developers and client understood what should be implemented in the software.
- All capabilities and functionalities are stored in SRS.
- Requirement document needs in every steps of software development.
- Analyst uses this document for designing the software. Developer uses this document during the coding whereas software tester uses this document to check the functionalities of the software.
- All remaining project documents are depends upon requirement document because of which it is referred as parent of all document.
- It contains functional and non functional requirements.

Good SRS have accomplishes following goals

- It gives feedback to customer.
- The large problem can be divided into different small components.
- It acts as input to design which is part of design specification.
- It acts as product validation check.

5.2.2 What Kind of Information Should an SRS Include?

Following things are part of SRS :

- Interfaces
- Functional capabilities
- Performance levels
- Data structures/Elements
- Safety
- Reliability
- Security/Privacy
- Quality
- Constraints and limitations

5.3 SRS Template

- Different existing SRS templates can be used in writing SRS documents.
- We can select the template which matches our requirement.
- Template will just acts as guiding principles for writing template. Proper changes need to be done whenever necessary in template.
- Table 5.3.1 shows SRS outline :

Table 5.3.1 : A sample of a basic SRS outline

1. **Introduction** : It contains different details like purpose of software, conventions used, target audiences, extra information, SRS team members, references.
2. **Overall Description** : Overall information of the project is given in this sections. It includes perspective, functions, different user classes, working environment, design constraints, assumptions about the software.
3. **External Interface Requirements** : It contains external interface information which includes user interface, hardware interface, software interfaces, communication protocols, etc.
4. **System Features** : Which system features needs to add is given in system features. It includes different features, features description, priority, action result, functionalities, etc.
5. **Other Nonfunctional Requirements** : Non functional requirement of the project is given in this section. It includes performance requirement, safety concerns, security constraints, quality factors, documentation, user documentation.
6. **Other Requirements** It includes Appendices, glossary of the software, etc.

5.3.1 Characteristics of an SRS

University Question

Q. Explain four desirable characteristics of a good Software Requirements Specification(SRS) document.

SPPU : Feb. 15, Feb. 16, 6 Marks

- **Complete :** All the project functionalities should be recorded by SRS.
- **Consistent :** SRS should be consistent
- **Accurate :** SRS defines systems functionality in real world. We need to record requirement very carefully.
- **Modifiable :** Logical and hierarchical modification should be allowed in SRS.
- **Ranked :** Requirement should be ranked using different factors like stability, security, ease or difficulty.
- **Testable :** The requirement should be realistic and able to implement.
- **Traceable :** Every requirement we must be able to uniquely identify.
- **Unambiguous :** Statement in the SRS document should have only one meaning.
- **Valid :** All the requirements should be valid.

5.4 Structured Specifications for an Insulin Pump Case Study

- Structured natural language is a way of writing system requirements where the freedom of the requirements writer is limited and all requirements are written in a standard way. This approach maintains most of the expressiveness and understandability of natural language but ensures that some uniformity is imposed on the specification.
- Structured language notations use templates to specify system requirements. The specification may use programming language constructs to show alternatives and iteration, and may highlight key elements using shading or different fonts.
- The Robertsons, a well known author, recommend that user requirements be initially written on cards, one requirement per card. They suggest a number of fields on each card, such as the requirements rationale, the dependencies on other requirements, the source of the requirements, supporting materials, and so on. This is similar to the approach used in the example of a structured specification shown in Table 5.4.1.

Table 5.4.1 : A structured specification of a requirement for an insulin pump

Function	Compute insulin dose: Safe sugar level.
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.
Inputs	Current sugar reading (r2), the previous two readings (r0 and r1).
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose—the dose in insulin to be delivered.

Destination	Main control loop.
Action	CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.
Requirements	Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin.
Post-condition	r0 is replaced by r1 then r1 is replaced by r2.
Side effects	None.

- To use a structured approach to specifying system requirements, you define one or more standard templates for requirements and represent these templates as structured forms. The specification may be structured around the objects manipulated by the system, the functions performed by the system, or the events processed by the system. An example of a form-based specification, in this case, one that defines how to calculate the dose of insulin to be delivered when the blood sugar is within a safe band, is shown in Table 5.4.2.

5.5 Tabular Specifications for an Insulin Pump Case Study

- Using structured specifications removes some of the problems of natural language specification. Variability in the specification is reduced and requirements are organized more effectively. However, it is still sometimes difficult to write requirements in a clear and unambiguous way, particularly when complex computations (e.g., how to calculate the insulin dose) are to be specified.
- To address this problem, you can add extra information to natural language requirements, for example, by using tables or graphical models of the system. These can show how computations proceed, how the system state changes, how users interact with the system, and how sequences of actions are performed.
- Tables are particularly useful when there are a number of possible alternative situations and you need to describe the actions to be taken for each of these. The insulin pump bases its computations of the insulin requirement on the rate of change of blood sugar levels.
- The rates of change are computed using the current and previous readings. Table 5.5.1 is a tabular description of how the rate of change of blood sugar is used to calculate the amount of insulin to be delivered.

Table 5.5.1 : Tabular specification of computation for an insulin pump

Condition	Action
Sugar level falling (r2 < r1)	CompDose = 0
Sugar level stable (r2 = r1)	CompDose = 0

Condition	Action
Sugar level increasing and rate of increase Decreasing ((r2 - r1) < (r1 - r0))	CompDose = 0
Sugar level increasing and rate of increase stable or increasing ((r2 − r1) ≥ (r1 − r0))	CompDose = round ((r2 − r1)/4) If rounded result = 0 then CompDose = MinimumDose

Review Question

Q. What is SRS? Why SRS is known as back-box specification of the system? What are major issues addressed by SRS?

Q. Explain different characteristics of Software Requirement Specification?

❑❑❑

6 REQUIREMENTS ANALYSIS

Unit - II

Syllabus

Analysis Model, data modeling, scenario based modeling, class based modeling, Flow oriented modeling, behavioral modeling-Introduction to UML diagrams.

6.1 Requirement Analysis

- Analysis model uses a combination of text and diagrammatic forms to depict requirements of data, functions and behaviour. So that it will be easy to understand overall requirements of the software to be built
- The **'Software Engineer'** also called as **'Analyst'** builds the model using requirements stated by the customer.
- Analysis model validates the software requirements and represent the requirements in multiple dimensions.
- Basic aim of analysis modelling is to create the model that represents the information, functions and behaviour of the system to be built.

6.1.1 Analysis Model

- Functions and behaviour of the system are translated into architectural, interface and component level designs in design modelling.
- Information, functions and behaviour of the system are represented using number of different diagrammatic formats.

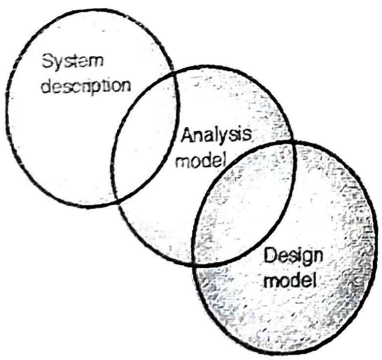


Fig. 6.1.1 : Analysis model, bridge in system description and design model

- As stated previously, the 'analysis model' acts as a bride between 'system description' and the 'design model'.
- The system description describes overall system functionality of the system including software, hardware, databases, human interfaces and other system elements. And the software design mainly focuses on application architectural, user interface and component level designs.

- Thus, all elements of analysis model are directly traceable to the parts of design model. Hence, the analysis model must have following three objectives :
  - To state clearly what customer wants exactly.
  - To establish the basis of the 'design model'. The information, functions and behaviour of the system defined by 'analysis model' are translated into architectural, interface and component level designs in 'design modeling'.
  - To bridge the gap between a system level description and software design.

6.1.2 Analysis Rules of Thumb

University Questions

Q. List all the rules of thumb. **SPPU : May 12, 3 Marks**

Q. What are the rules of thumb? **SPPU : May 13, 3 Marks**

These rules are

- The requirement analysis must state the requirements at business domain and at high level of abstraction. No need to state the details.
- Each element of analysis model must help for clear understanding of software requirements and must focus on information, functions and behaviour of the system
- Consideration of infrastructure and non-functional model should be delayed to design. For example : Define the databases if required for software under consideration. But no need to consider the details likes classes, functions and behaviour of the databases.
- Try to minimize the coupling throughout the system. The interconnections among different modules is called 'coupling'. That is the coupling measures the 'functional dependency' of different modules. Hence for good design, the interconnection among different components should be minimum.
- The analysis model provides value to all people concern to it.
- **For example :** Customer will use analysis model to validate his requirements. The Designer will use the analysis model for designing the 'design model'. End user can use same model for testing.
- The analysis model must be as simple as possible.
- Only simple model will help the easy understanding of software requirements. Hence analysis model should be simple enough.

6.1.3 Domain Analysis

University Question

Q. Explain domain analysis. **SPPU : Dec. 12, May 14, 3 Marks**

- By definition, software domain analysis is "the identification, analysis and specification of common requirements from a specific application domain." These common software domains can be reused for multiple projects within that application domain.
- In short, software domain analysis leads to 'reusable software components' in software development. Here, in specific application domain common objects, common classes, common frameworks can be identified and can be reused.

- Role of 'domain analyst' is same as job of toolsmith. The job of toolsmith is to design the tools those are used by many people for similar works. The role of domain analyst is to define reusable objects, classes, frameworks, those can be used by many people in similar applications.
- **For example :** 'The specific application domain' may be 'bus reservation system'. The software domains of 'Us reservation system' can be used for 'railway reservation system'.

Advantages

- Use of such software domain analysis saves the time.
- It also reduces the development cost.

6.1.4 Requirements Modeling Approaches

Following are different requirement modeling approaches :

- **Structured analysis :** It consists of **data** and the **process** that transforms data.
- **Object-oriented analysis :** It emphasizes on the classes that collaborates with one another to focus on customer's requirements.

6.2 Data Modelling

University Question

Q. What is data modeling ? **SPPU : May 15, 2 Marks**

- Analysis modeling begins with data modeling. Here all data objectives that are processed within the system, their attributes and relationship between different data objects are specified.
- The relationship between different data objects is also considered in terms of number of occurrences i.e. cardinality.
- Also the relationship is optional or mandatory is represented by modality.

6.2.1 Data Objects

University Questions

Q. Discuss in short: Data objects in data models **SPPU : Dec. 12, May 14, 3 Marks**

Q. Explain term in data modeling : Data objects. **SPPU : May 15, 3 Marks**

- Data object is the representation of composite information. The composite information is defined as the object that has various different properties or different attributes.
- For a data object, the encapsulation of only data is present. It means here is no reference between data object and operations on the data.
- **For example,** in the table, car is a data object with properties name, colour and prize. For example data object car can be represented in tabular form as follows :

Table 6.2.1 : Tabular representation of data object

Name	Color	Prize
Honda Amaze	Silver	6 Lac
Santro	Yellow	3.5 Lac
Alto 800	Maroon	3 Lac
Hyundai i20	Blue	11 Lac

6.2.2 Data Attributes

University Question

Q. Explain term in data modeling: Data attributes. **SPPU : May 15, 3 Marks**

Each data object is having set of attributes. That is data attributes define the properties of data object can have of the following characteristics :

1. Name an instance of data object.
2. Describe the instance.
3. Make reference to another instance in another table.

6.2.3 Relationship

University Question

Q. Explain term in data modeling: Relationships. **SPPU : May 15, 3 Marks**

- Relationship indicates how data objects are related to one another.

  For example: Customer purchases the car. Here, 'purchase' is the relation.

- The relationship between data object can be represented as :

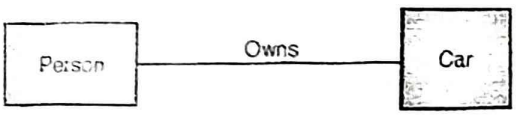


Fig. 6.2.1 : The relationship between data objects

6.2.4 Cardinality and Modality

University Question

Q. Discuss in short: Cardinality and modality in data models. **SPPU : Dec. 12, May 14, 3 Marks**

- We represent the pictorial relationship between two objects as shown in Fig. 6.2.2.

Fig. 6.2.2

- Here, the relation shows that object X is related to object Y.

- But, this relation does not provide enough information for "how many occurrences of object X are related to how many occurrences of object Y".
- Hence such types of details are provided by cardinality and modality.

Cardinality

- Cardinality specifies number of occurrences of one object related to number of occurrences of another object.
- In other words, cardinality refers "how many occurrences of object one object are related to how many occurrences of another object".
- That is the maximum number of object relationship is represented by cardinality. Cardinality can be expressed as :

One to one (1 : 1)

- It implies that one occurrence of object one is related to one occurrence of another object.
- **For example:** An engineering college is having only one principal.

One to many (1 : n)

- It implies that one occurrence of object one is related to many occurrence of another object.
- **For example** : One class may have many students.

Many to many (m : n)

- It implies that many occurrence of object one is related to many occurrence of another object.
- **For example :** An uncle may have many nephews while a nephew may have many uncles.

Modality

A modality of relationship is zero if occurrence of relationship is optional and modality of relationship is 1 if occurrence of relationship is mandatory (i.e. compulsory). Thus, the modality specifies the minimum number of relationship.

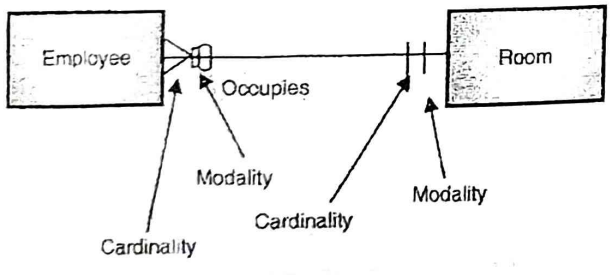


Fig. 6.2.3 : Relationship showing data objects with maximum and minimum number of occurrences

The symbol for showing the number of occurrences is :

Here, the relationship shows that "Exactly one (maximum 1 and minimum 1) room is occupied by zero or many (maximum many and minimum 0) employees or the relationship may be read as "zero or many employees occupy exactly one room".

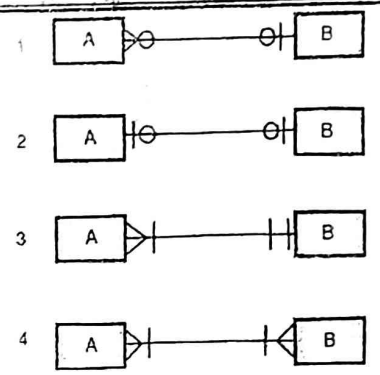


Fig. 6.2.4 : 1 to 4 showing relationship between different objects with different combinations of cardinality and modality

- Fig. 1 : Cardinality is n : 1 and both roles are optional.
- Fig. 2 : Cardinality is 1 : 1 and both roles are optional.
- Fig. 3 : Cardinality is n : 1 and both roles are mandatory.
- Fig. 4 : Cardinality is m : n and both roles are mandatory

Example of entity relationship diagram

The Fig. 6.2.5 shows the entity relationship diagram including cardinality and modality. Now, each relation can be read with maximum and minimum number of occurrences of that object.

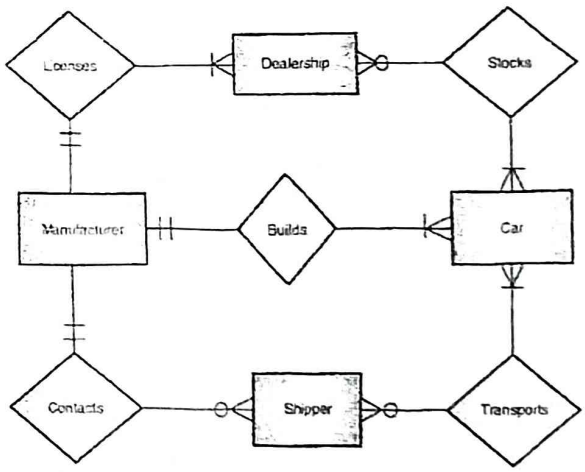


Fig. 6.2.5 : Entity Relationship diagram

For example

From the Fig. 6.2.5 we can read the relations with number of occurrences as follows :

- A manufacturer builds a car.
- A manufacturer contacts to zero or many shippers.

6.3 Introduction to UML Diagram (Scenario Based Modeling)

University Question

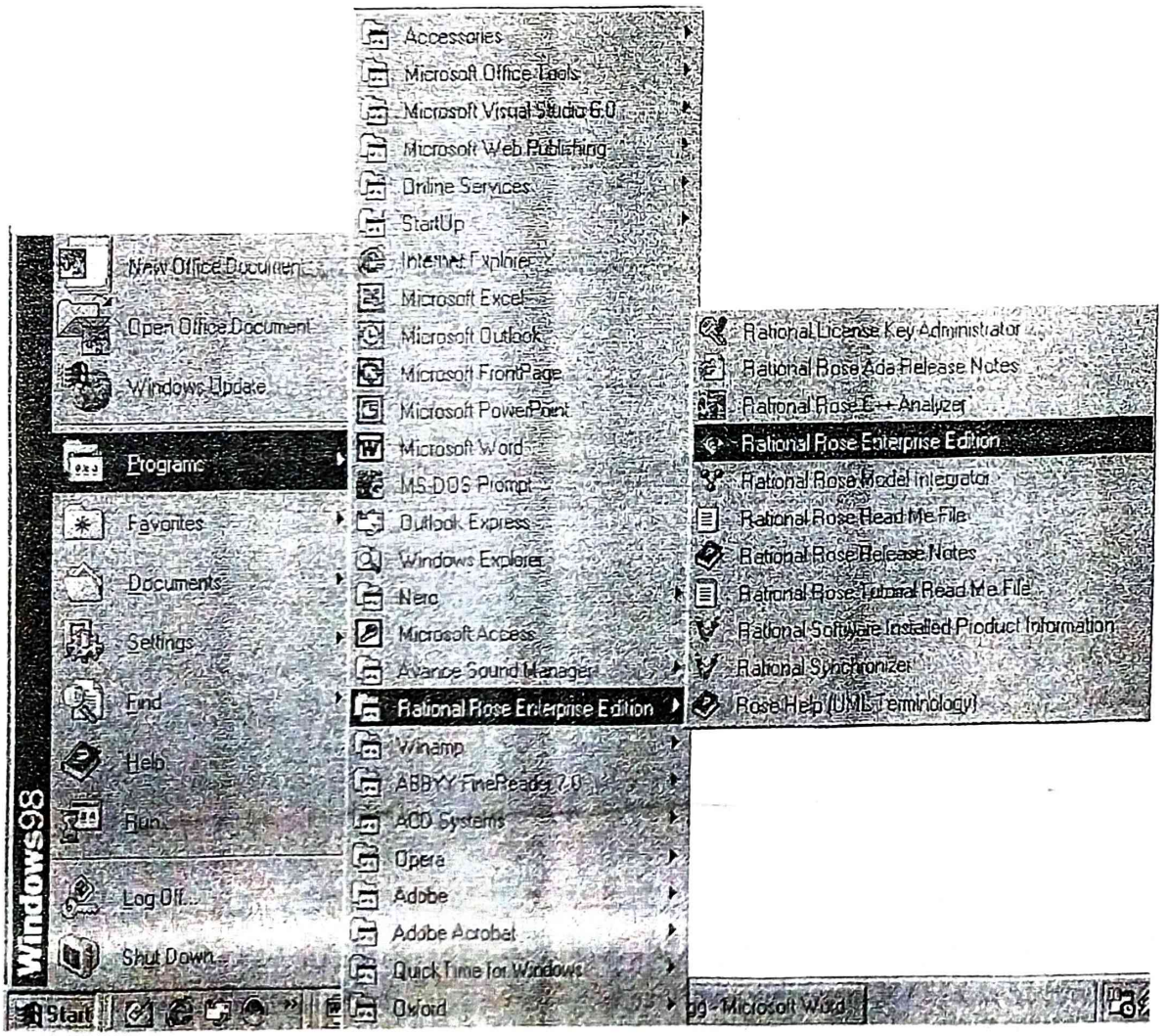
Q. Describe the steps of scenario based modeling with a suitable example. **SPPU : May 15, 5 Marks**

- Analysis modeling with UML begins with the creation of scenarios.
- In Scenario Base modeling the system is represented in user's point of view. Scenario-based elements are :
  1. Use case diagrams
  2. Activity diagrams
  3. Swim lane diagrams

6.3.1 Diagramming in UML

- During 1990 Rumbaugh, Booch and Jacobson combinely presented a Unified Modeling Language (UML) that includes notations for modeling and development of OO systems.
- By 1997 UML became a industry standard for Object Oriented software development and Rational corporation developed automated tools to support UML methods (i.e. Rational Rose software).

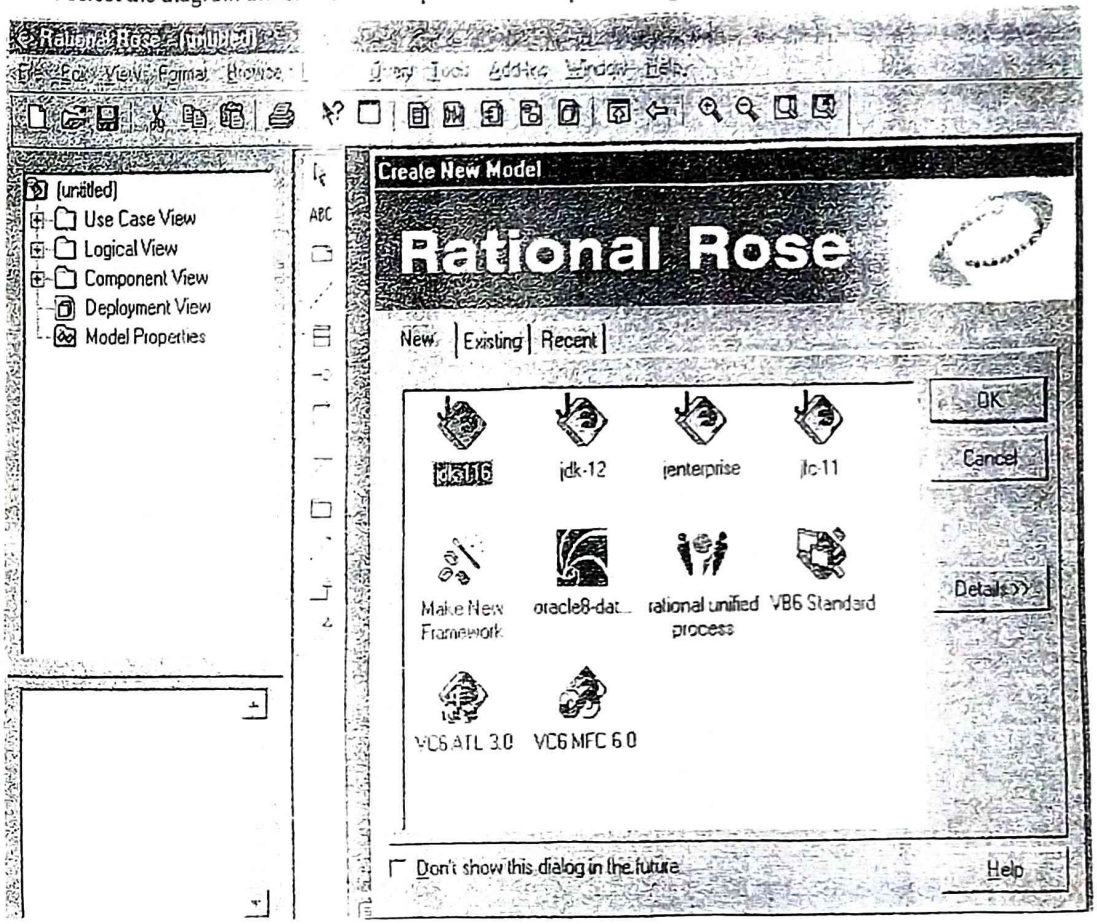
Using Rational Rose for UML diagrams

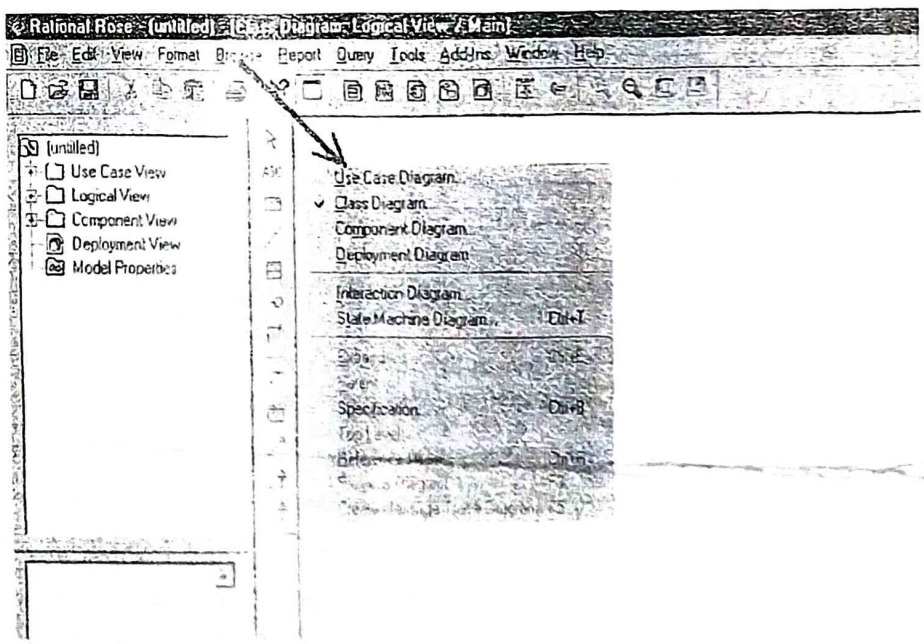


- Install Rational Rose on your PC.

  Go to start menu and open "Rational Rose Enterprise Edition".

- Now select the diagram under "Browse" option to draw specific diagram :





6.3.2 Developing Use Cases Diagram

University Questions

Q. Explain in detail UML diagram stating purpose and applicability : Use- Case diagram. SPPU : May 12, May 13, 3 Marks

Q. What is Use-case diagram? Illustrate it with some suitable example. SPPU : Dec. 19, 4 Marks

- To start developing a set of use-cases, the functions or the activities performed by actor are listed. The list of Use-cases may be obtained from one of the following sources :
  - From customer communication.
  - End user communication.
  - By evaluating the activity diagrams.
  - Different notations used for use case diagrams are :

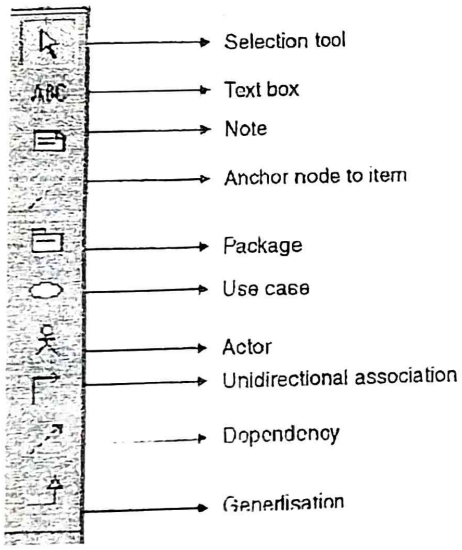


Fig. 6.3.1 : **Notations in use case diagram**

Example of use case diagram

Problem statement : To develop the Use case diagram for 'food ordering system' in hotel.

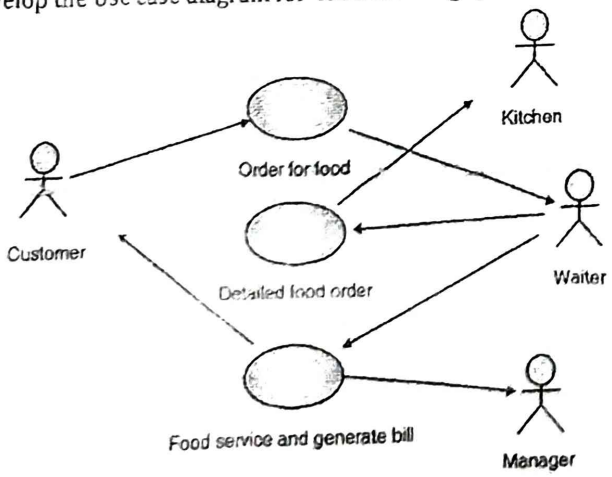


Fig. 6.3.2 : **Use case diagram for food ordering system**

Explanation

Customer orders for the food. The waiter receives this order. The waiter then forms detailed order. The detailed order is submitted to the kitchen and desired service is provided to the customer. Finally bill is produced. The bill is submitted to customer and bill report is sent to manager.

6.3.3 Developing Activity Diagram

University Questions

Q. Explain in detail UML diagrams stating purpose and applicability: Activity diagram. **SPPU : May 12, 3 Marks**

Activity diagrams are same similar to that of flowcharts. The notations used in activity diagram are :

- Rounded rectangles imply specific system functions.
- Arrows represent flow through the system.
- Decision diamonds are used to depict branching decisions.

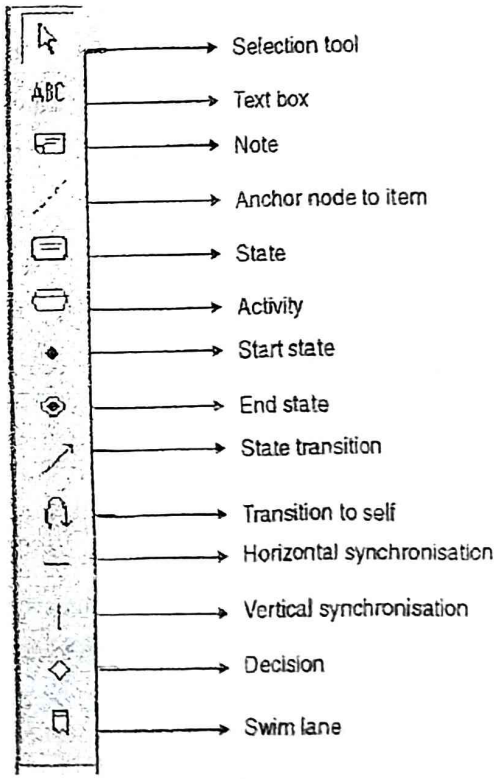


Fig. 6.3.3 : Notations in activity diagram

Example of activity diagram

Problem statement : To develop the activity diagram for 'Building Plan'.

Explanation

- First 'dig foundation' activity will be completed. After completing foundation, two activities 'build walls' and 'connect service' are two activities those can run parallel.
- After completing these activities 'build roof' activity can be started.

- While 'wiring' and 'carpentry' is going on sequentially, at the same time 'fit windows' activity can be completed parallel. Finally, 'painting' activity is completed.

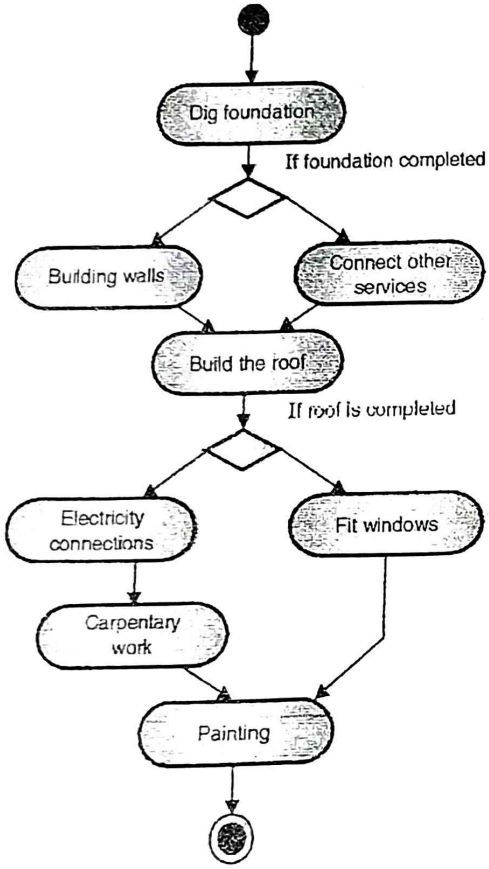


Fig. 6.3.4

6.3.4 Swim Lane Diagram

- Swim lane diagrams are the variation in activity diagrams. Swim lane diagrams allows to represent, the flow of activities described by use cases.
- At the same time these diagrams indicate which actors are involved in specific functions.
- We can also show the interaction between these different activities shown in different lanes.
- To show such parallel activities, diagram is divided vertically into lanes, similar to lanes of swimming pool. Hence these diagrams are called as 'swim lane' diagram.

Example of swim lane diagram

Problem statement

To develop the swim lane diagram for 'purchase equipment order'.

Explanation

First, Head of the department will submit the requirement for purchasing the equipments.

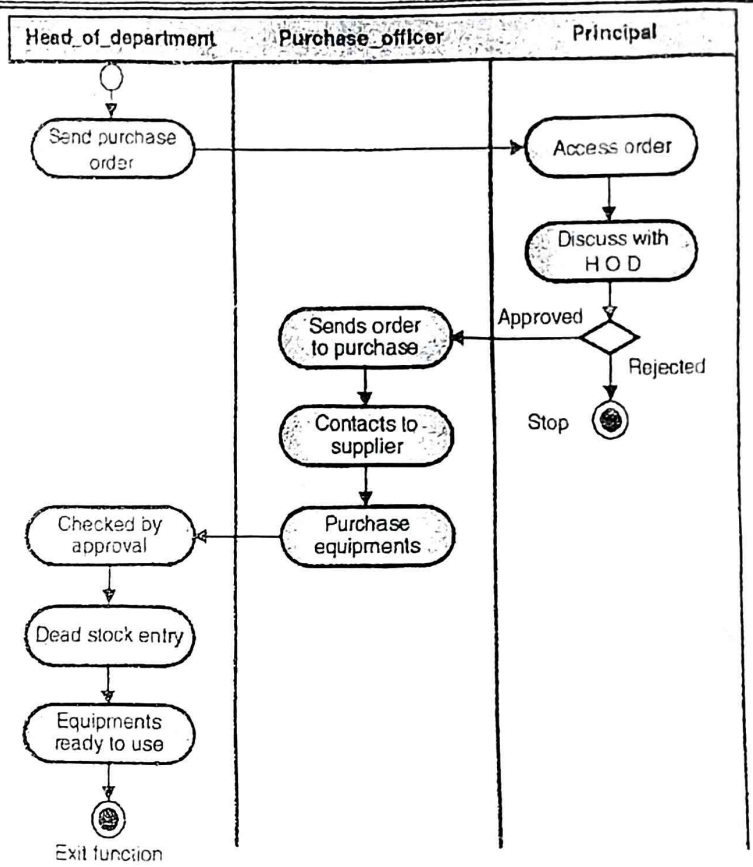


Fig. 6.3.5

- The order is reviewed by the principal.
- If the principal permits the proposal, the order is forwarded to purchase officer.
- Purchase officer now contacts to supplier.
- After purchasing the required equipments, they are checked by H.O.D.
- Finally, the dead stock entry is done in dead stock of the department.
- Purchased equipments are ready for use.

6.3.5 Class Diagram

Example class diagram

The class diagram for computer based system is shown with two important symbols :

6.3.5(A) Aggregation

Aggregation denotes super class to subclasses relationship.

For example in Fig. 6.3.6, we can say that Monitor, Printer and Plotter are subclasses of super class 'Output Device'.

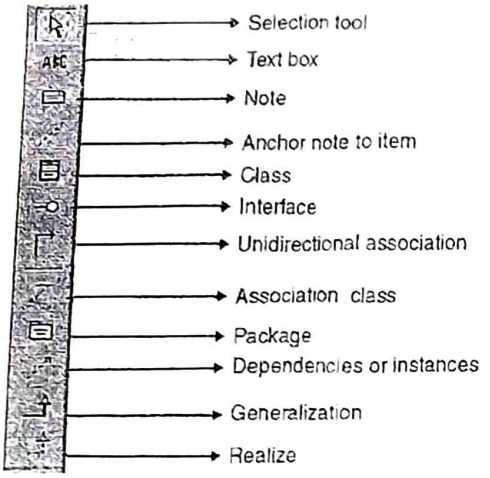


Fig. 6.3.6 : Notations in class Diagram

6.3.5(B) Generalization

This symbol is used to denote 'is part of' or 'is composed of' relation.

For example : In Fig. 6.3.7 we can say that class CPU is composed of classes RAM, Control Unit and ALU.

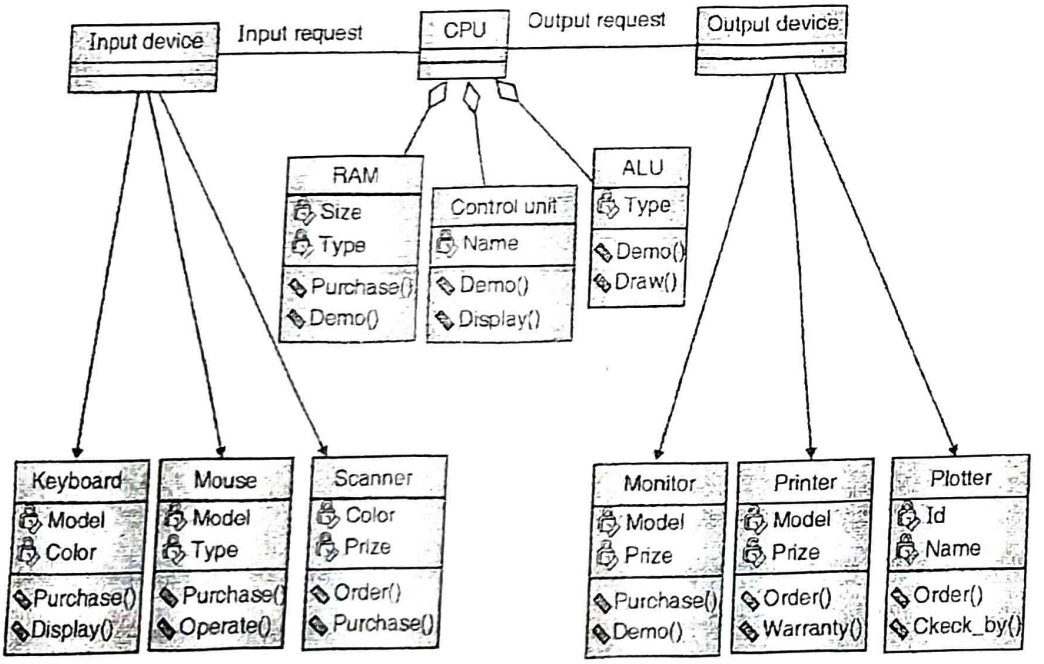


Fig. 6.3.7

3.5(C)Associations and Dependency

ociations

Two classes may be related to one another. This relation is called as 'association'. For example : In puter based system shown in Fig. 6.3.8, class 'input device' is associated with class 'CPU' and class 'CPU' is iated with class 'output device'.

Multiplicity

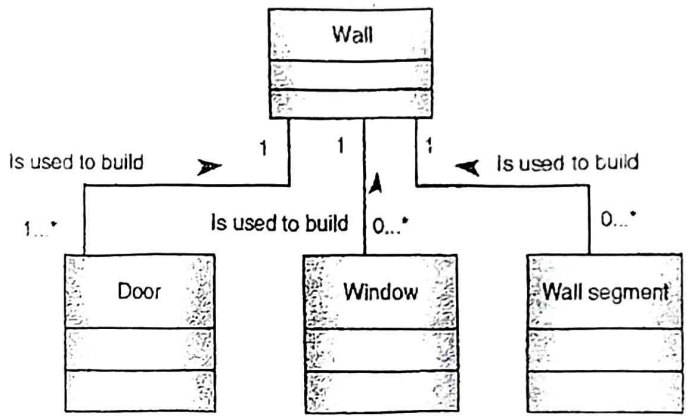


Fig. 6.3.8

The association is further defined by indicating multiplicity. It shows how many occurrences of one class are related to how many occurrences of another class. For example, from Fig. 6.3.8, we can read that 1...* doors are used in building one wall.

Dependency

Sometimes the application requires showing the dependencies between the classes.

For example : The librarian will get access to the 'server system' if he enters the correct 'password'. This dependency is shown in Fig. 6.3.9.

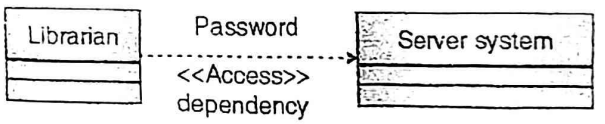


Fig. 6.3.9

Analysis Packages

Package : The package is used to assemble a collection of related classes.

For example : The analysis model of video game may include following packages :

- Environment
- Characters

In package environment we can have following related classes :

- Tree
- Landscape
- Road
- Wall
- Bridge

- Building
- Visual effect
- Scene

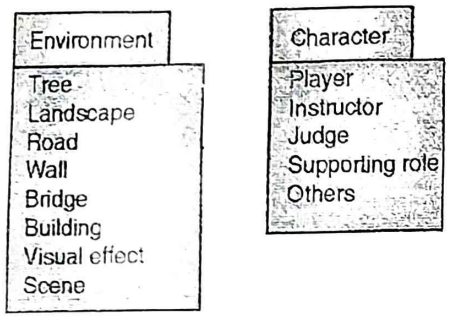


Fig. 6.3.10 : Environment and characters packages in video game

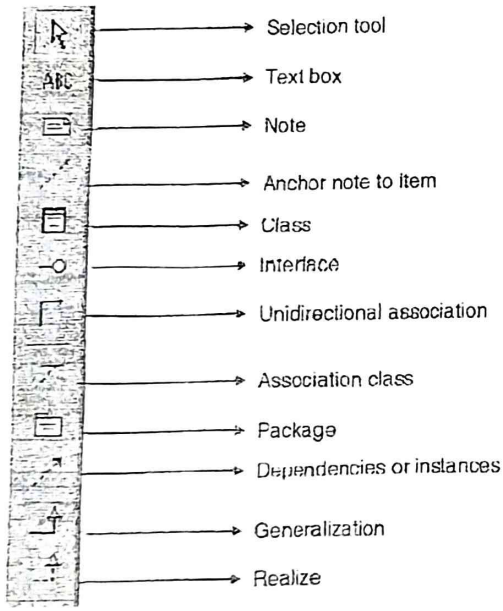


Fig. 6.3.11 : Notations in package diagram

6.4 Class Based Modeling

- As we have discussed earlier that object oriented software engineering approach focuses on component level design and analysis classes and interfaces between the classes, in class based component the detailed description of attributes, operations and their interfaces are emphasized.

The design details are discussed in the following sections.

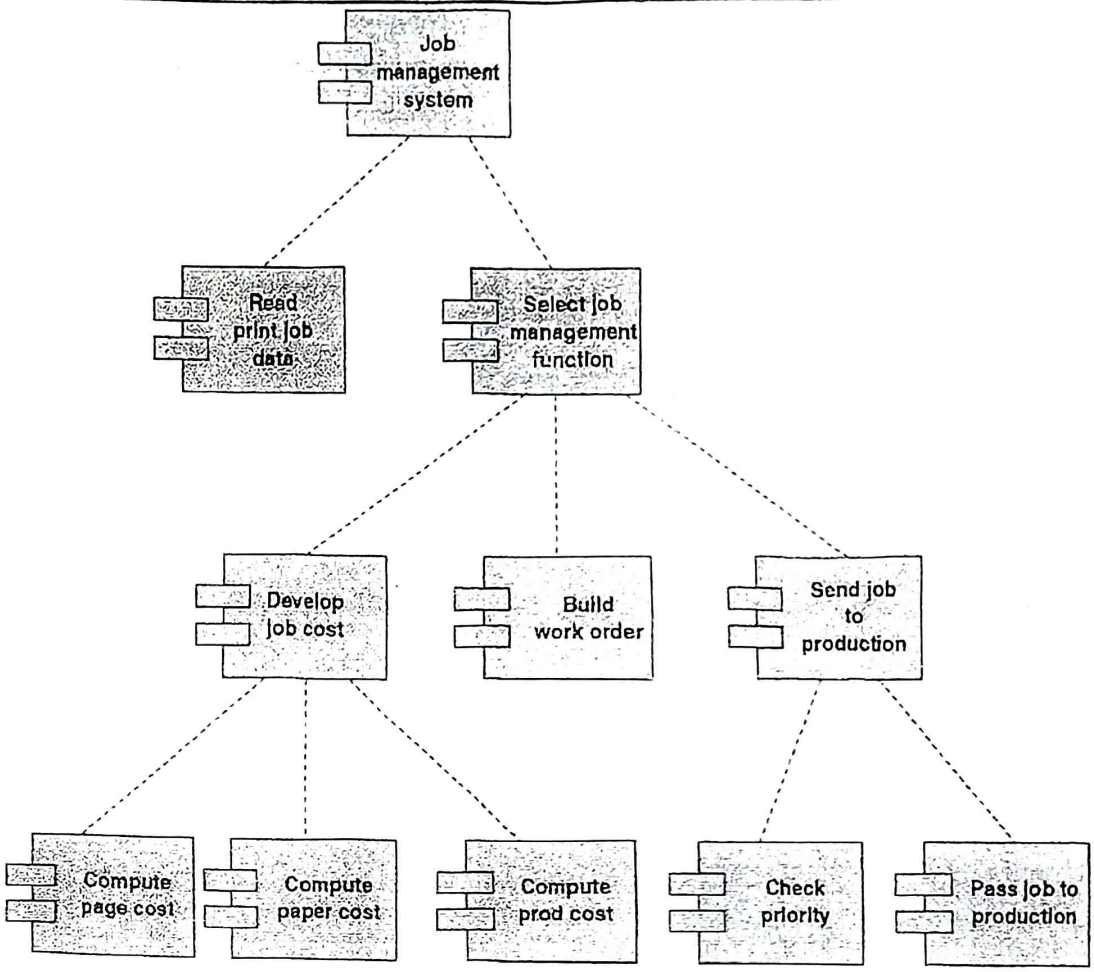


Fig. 6.4.1 : Structure chart for a conventional system

6.4.1 Basic Design Principles

Following are some basic design principles for class-based components:

1. **The Open-Closed Principle (OCP) :** In OCP any module must be available for extension and modification. The designer should specify the component in such a way that it can be extended without any modification in the internal structure of the component.
2. **The Liskov Substitution Principle (LSP) :** In this principle the subclasses should be substitutable for the base classes. This particular principle is suggested by Liskov. The LSP suggests that any class derived from the base class must imply the contract between the base classes and their components.
3. **Dependency Inversion Principle (DIP) :** In DIP the design depends on abstractions and not on concretion. The abstractions are the place where design is allowed to extend without any further complications.
4. **The Interface Segregation Principle (ISP) :** In this principle many client specific interfaces are better than general purpose interfaces. The component level design are organised into sub-systems or packages and suggests additional packaging principle for each component.
5. **The Release Reused Equivalency Principle (REP) :** In this the reuse of each module is actually the release of module. The classes or components are designed for reuse. It is always better to make a group of reusable classes i.e. packages that are easy to manage.

6. **The Common Closure Principle (CCP)** : In this principle the classes that changed together belongs together i.e. there is a cohesion between the classes.
7. **The Common Reused Principle (CRP)** : In CRP the classes that are not reused together, they should not be group together also.

6.4.2 Conducting Component-Level Design

Following are the component-level design steps for an object oriented systems :

- Identify those design classes that are related to problem domain only.
- Now identify those classes that are related to infrastructure domain. The classes and components in this step include GUI components, OS components and object and data management components.
- Identify all the design classes that do not have reusable components and elaborate these design classes. In this step, the message details are also specified when the components or the classes collaborate. Also identify proper interfaces for each of the components and elaborate their attributes and define the data types of the attributes and also define the data structures necessary to implement the components.
- Describe the databases and files i.e. persistent data sources. Identify the classes required to manage these persistent data sources.
- Develop the behavioural representations for classes or the components and elaborate them appropriately.
- Elaborate the deployment diagrams in order to give additional implementation details.
- Always refine every component level design and consider alternative design solutions also.

6.5 Flow Oriented Modeling

They provide necessary information that how data objects are transformed by processing the functions. Flow oriented elements are :

1. Data Flow Diagrams
2. Control Flow Diagrams
3. Control Specifications
4. Process Specifications

6.5.1 Data Flow Model

- Data Flow Diagram (DFD) is also called as 'Bubble chart' is a graphical technique, which is used to represent information flow, and transformers those are applied when data moves from input to output.
- DFD represents system requirements clearly and identify transformers those becomes programs in design. DFD consists of series of bubbles joined by lines.
- DFD may be further partitioned into different levels to show detailed information flow e.g. level 0, level 1, level 2 etc.

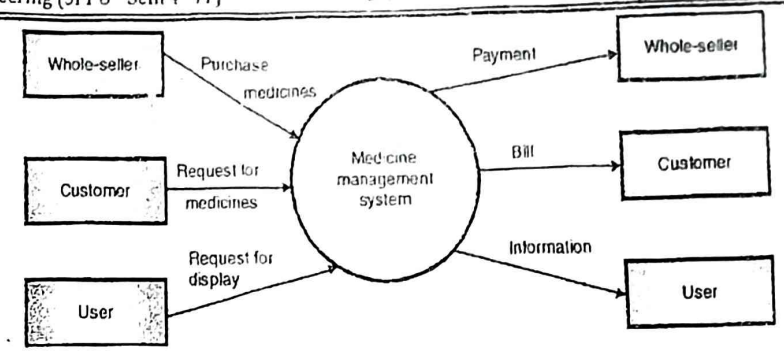


Fig. 6.5.1 : Data flow diagram for medicine managements system

- DFD focuses on the fact 'what data flow' rather 'how data is processed'.

Example of data flow diagram

The data flow diagram developed for medicine management system includes the external entities : Whole-seller, Customer and User. The DFD shows following requirements with proper flow of data from one entity to other.

- The medicines are purchased from whole-seller.
- The customer can request for the medicines in medicine store.
- User can request for display of the current status of the items in medicine.
- When medicines are purchased from whole-seller, the system generates the payment for whole-seller.
- When medicines are sold to customer, the system generates the bill for customer.
- The system also presents the required information to the user as per his requirements.

Developing level 1 DFD

- DFDs are used to represent information flow, and the transformers those are applied when data moves from input to output.
- To show the data flow with more details, the DFD is further extended to level 1, level 2, level 3 etc. as per the requirements.
- The typical value for the DFD is seven. Any system can be well represented with details up to seven levels.
- The level 0 and level 1 DFD for the 'Food Ordering System' in the restaurant' are shown in Figs. 6.5.2 and 6.5.3.

Notations used in DFD

- The circle represents the process.
- The rectangle represents the external entity like customer, whole-seller etc.
- The labelled arrows indicate incoming and outgoing data flow.
- The open rectangle shows the database or file.

DFD for food ordering system

Here customer, kitchen and restaurant managers are external entities.

- The 'food order system' accepts the food order from the customer and forwards the order to the kitchen.
- When the service is provided to the customer, the system generates the bill.
- A copy of customer bill can be submitted to the manager as a part of restaurant management report.

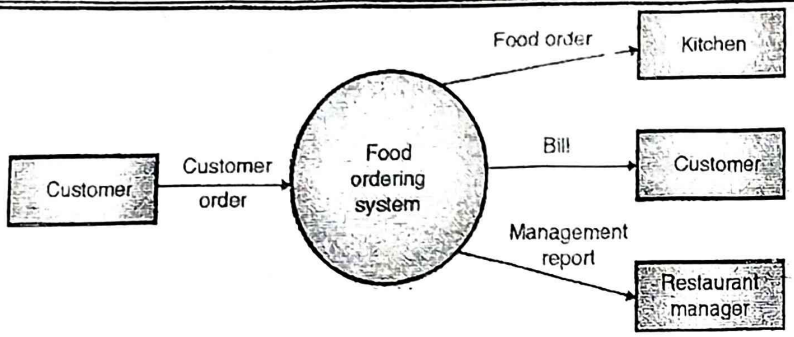


Fig. 6.5.2 : Level 0 DFD for 'food ordering system' in restaurant

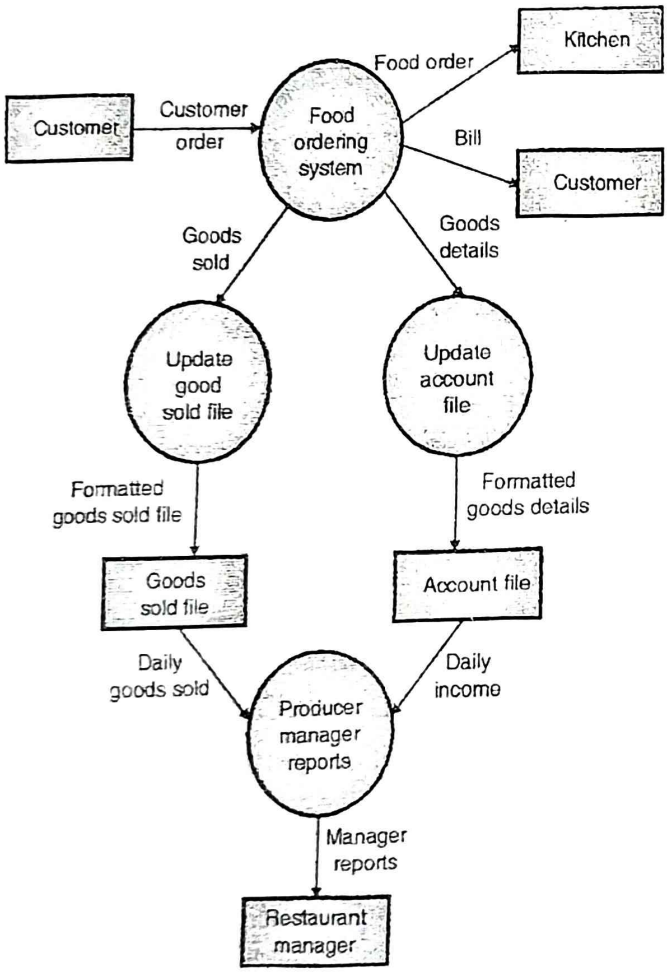


Fig. 6.5.3 : Level 1 DFD for 'food ordering system' in restaurant

This level 0 DFD can be extended to level 1 DFD to show more details showing exact data flow and processes (i.e. transformers).

6.5.2 Control Flow Model

The large class of applications having following characteristics requires control flow modeling :

- The applications those are driven by the events rather than data.
- The applications those produce control information rather than reports or displays.

- The applications those process information in specific time.

The control item or event is implemented as Boolean value. For example, true or false, on or off, 1 or 0.

6.5.3 Control Specifications

The control specification represents the behaviour of the system. The behaviour of the system can be represented using 'state transition diagram' which is also called as 'state chart diagram'.

A state chart diagram

A state chart diagram shows the state machine that consists of states, transitions, events and activities. This diagram shows how system makes the transition from one state to another state on occurrence of particular event.

State

A state of an object is defined as the condition that is satisfied in the life of an object and once the condition is satisfied, the object performs some activity and it waits for some events to occur.

Event

An event causes the system to make transition from one state to another. The notations used in state chart diagram are :

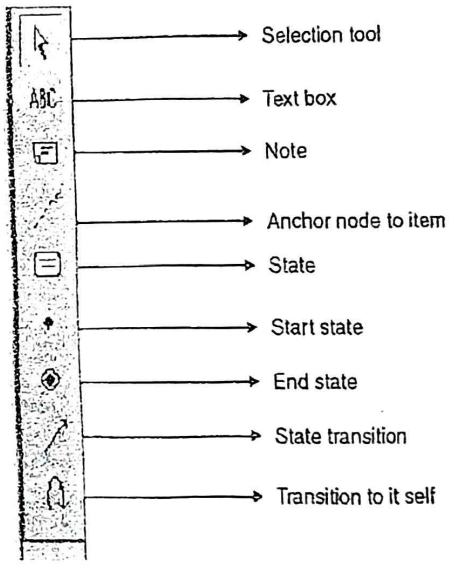


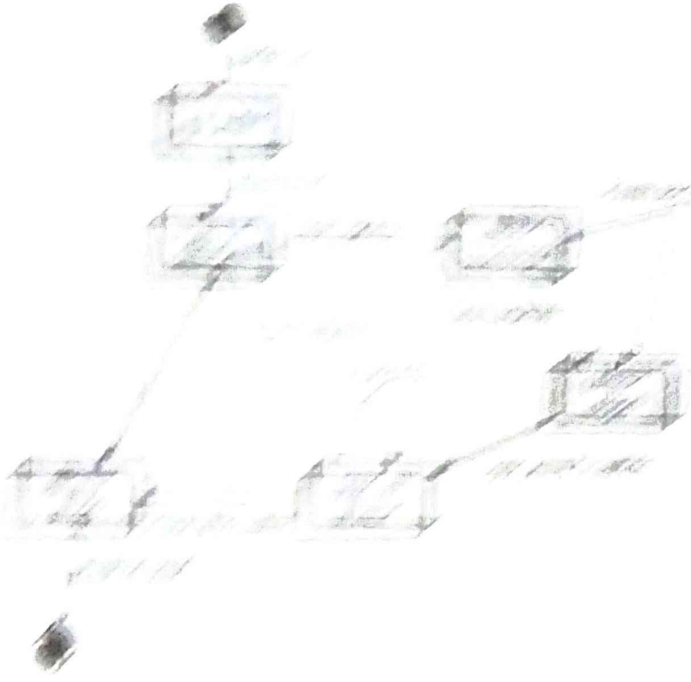
Fig. 6.5.4 : **Notations in state chart diagram**

Example of state chart diagram

- **Problem statement :** To develop the state chart diagram for 'washing machine':

Explanation

- First, fill the water in washing machine. Then take detergent in. Now, motor is in running state.
- The 'motor running' state can go in ' motor stops' state in one of the two cases :
  - When user of washing machine will press the stop switch or
  - Time expires



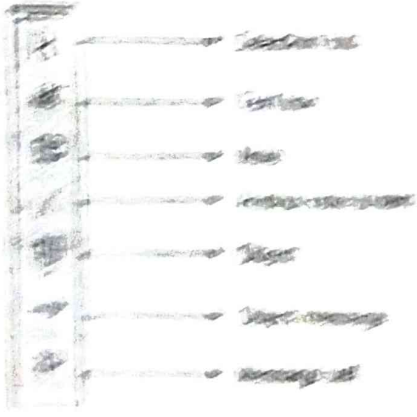


Fig 4.6.1 : Notations used in Sequence Diagram

Example of sequence diagram

- To develop the sequence diagram for **'Computer Based System'**.

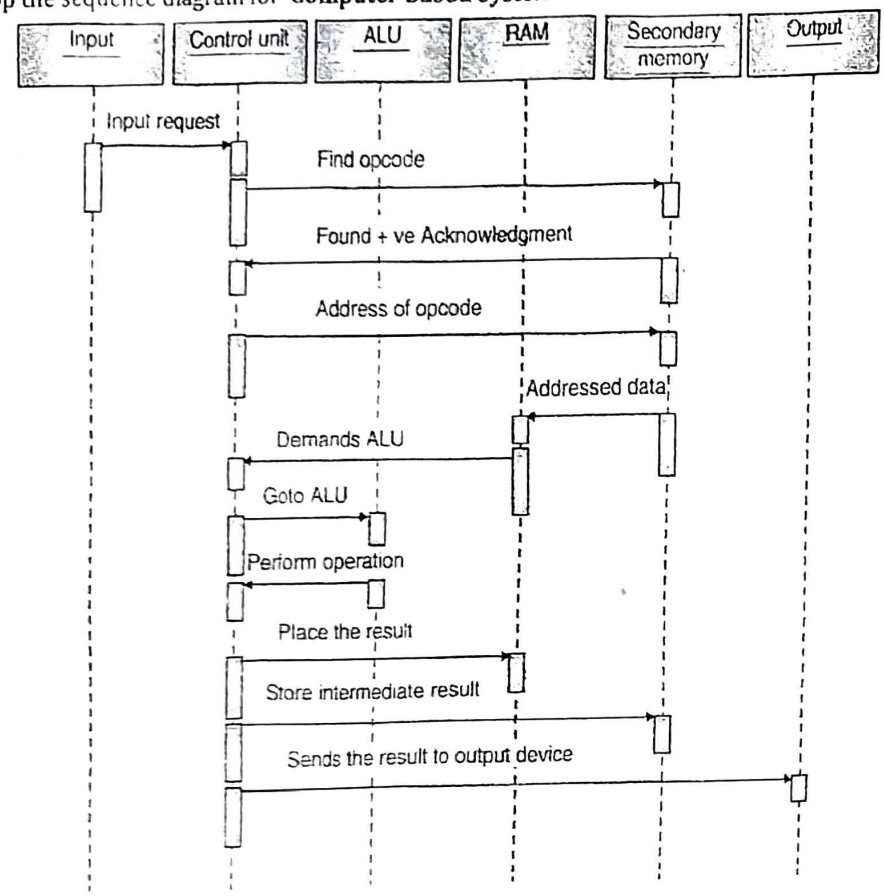


Fig. 6.6.2 : Sequence diagram for Computer Based System

6.6.3 State Machine Diagram with Orthogonal States

- One of the important components of bahavioural model is UML state machine diagrams that represent the active states and events between the active states.
- A state machine is defined as a behaviour that is exhibited by the sequences of states of an object during its lifetime. These states exist due to occurrence of events.
- The state machines are modelled by the dynamic aspects of a system. During the modeling phases, the state machine specifies the lifetime of the instances of a class. It also specifies the lifetime of a use case may the whole system.
- All these instances respond to the events like operations, signals or the pastime. Whenever an event occurs, any of the activity may take place based on the current state of an object.
- The activity in state machine is defined as an ongoing non-atomic execution within it. These activities cause some action that result in changing the state of an object or the model. The actions may be defined as executable atomic computations.
- The state of an object satisfies any condition during its lifetime and due to this it performs some activity and wait for some events to occur.

- We see tate machine by two points of view :
  - By putting emphasis on flow of control in different activities and
  - By putting emphasis on the states itself by using state chart diagrams
- A well structured state machine is very similar to a well structured algorithm in the sense that it adaptable to environment, it is simple and efficient and it is easy to understand.
- A state machine is generally used to model the behaviour of the elements of a class, use-case or the complete system.
- A state machine is viewed in two different ways :
  - By using activity diagrams or
  - By using the state machine diagrams
- Here the activity diagram focuses on the activities occurring within an object and state machine diagrams focus on the behaviour of an object.
- The UML diagrams illustrate the graphical representation of states, its transitions, events and their actions as depicted in Fig. 6.6.3.

6.6.3(A) Orthogonal States

- Following diagram shows that the orthogonal regions are entered in the state machine diagram. Basically the state with two or more than two regions is called as orthogonal state.
- A transition is a path that shows the change of state is occurring. In the following diagram two regions are represented. Both the regions finish in parallel and reach the final node i.e. S6. This is an example of a state machine with orthogonal states.

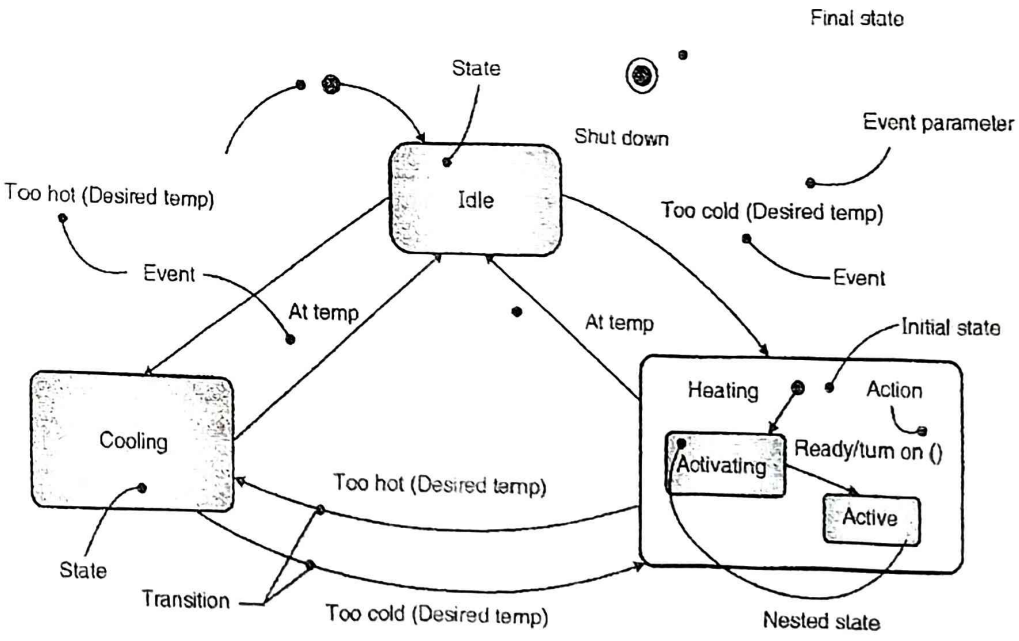


Fig. 6.6.3 : State machines

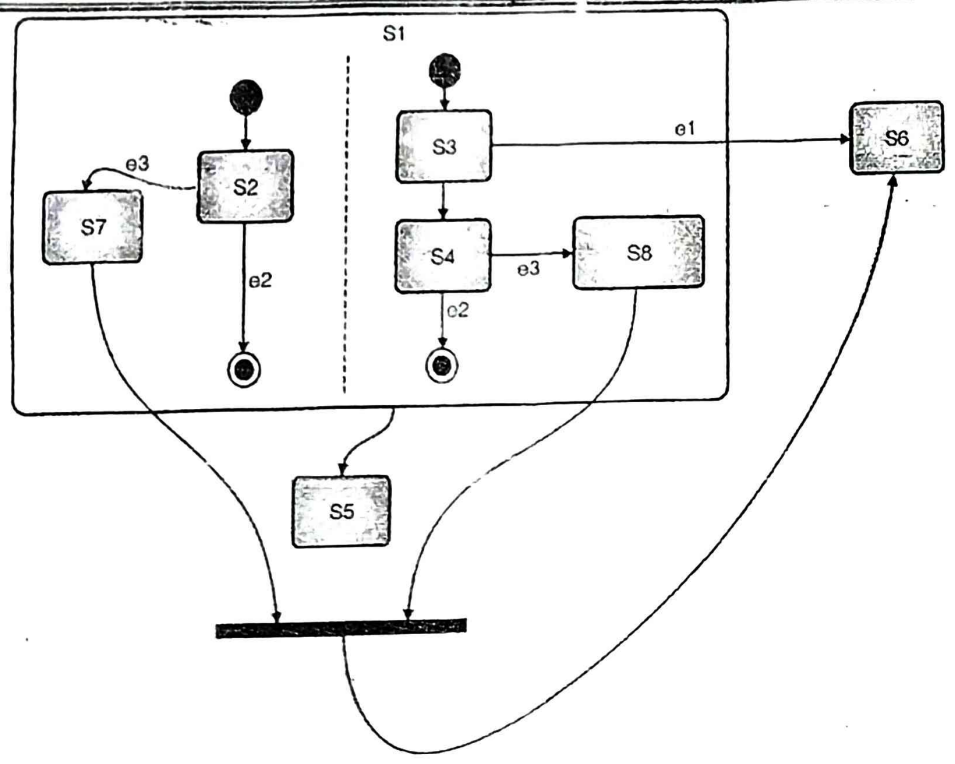


Fig. 6.6.4 : State machine with orthogonal state

Review Question

Q. List all the rules of thumb.

Q. What are the rules of thumb?

Q. Explain domain analysis

Q. Describe the steps of scenario based modeling with a suitable example

Q. Explain in detail UML diagram stating purpose and applicability : Use - Case diagram.

Q. Explain in detail UML diagrams stating purpose and applicability : Activity diagram.

Q. What is data modeling?

Q. Explain term in data modeling: Data attributes.

Q. Discuss in short: Data objects in data models.

Q. Explain term in data modeling : Relationships

Q. Discuss in short: Cardinality and modality in data models.

Q. Discuss in short: Data objects in data models.

Q. Explain in detail UML diagrams stating purpose and applicability : State diagram

□□□

Model Question Paper (In sem) Unit I & II

Model Question Paper (In Sem.)

Software Engineering

Semester IV – Information Technology (Savitribai Phule Pune University)

Time : 1 Hour **Maximum Marks : 30**

Instructions to the candidates :

1) **Answer Q. 1 or Q. 2, Q. 3, or Q. 4.**
2) **Neat diagrams must be drawn wherever necessary.**
3) **Figure to the right indicates full marks.**
4) **Make suitable assumptions if necessary.**

Q. 1	(a)	What are Core Principles of Software Practice? Explain in detail. **(Refer Section 1.3.2)**	**(5 Marks)**
	(b)	What are framework and umbrella activities? (Refer Section 1.4.1)	**(5 Marks)**
	(c)	State and Explain different myths. (Refer Section 1.5)	**(5 Marks)**

OR

Q. 2	(a)	What is generic process model? Explain its activities. (Refer Section 2.1)	**(5 Marks)**
	(b)	Explain in detail all the process phases of waterfall process model and state merits/demerits of the same. **(Refer Section 2.2)**	**(5 Marks)**
	(c)	What is relation of agility and cost of change? (Refer Section 3.2.1)	**(5 Marks)**
Q. 3	(a)	What are functional and non functional requirements of software? **(Refer Program 4.2)**	**(5 Marks)**
	(b)	What is meant by normal requirements and exciting requirements? **(Refer Section 4.4.1)**	**(5 Marks)**
	(c)	How to prioritize software requirements based on Kano Analysis? **(Refer Section 4.5)**	**(5 Marks)**

OR

Q. 4	(a)	Explain four desirable characteristics of a good Software Requirements Specification (SRS) document? **(Refer Section 5.3.1)**	**(5 Marks)**
	(b)	What is a Software Requirements Specification? (Refer Section 5.2.1)	**(5 Marks)**
	(c)	What are the characteristics that requirement must meet? (Refer Section 4.7)	**(5 Marks)**

□□□

7

Unit - III

DESIGN ENGINEERING

Syllabus

Design Process & quality, Design Concepts, design Model, Pattern-based Software Design.

7.1 Introduction to Design Engineering

- The goal of design engineering is to produce a model or representation that should show the firmness, delight and commodity.
- In order to accomplish this task, a designer should practice diversification and after that convergence.
- Diversification means; stock of all alternatives, the raw material of design like components, component solutions, and knowledge.
- From this, diverse set of information is assembled and the designer must pick and choose elements from the stock that meet the requirements.
- As this point, alternatives are considered and rejected, and the design engineer converges on one particular configuration of components, and thus the creation of the final product.

7.2 Design Process

- It is process of designing the software to achieve the intended goal of the software.
- Software design is done with the help of set of primitive components.
- It refers to all activity in conceptualizing, framing, implementing, commissioning, etc.
- It sometimes refers to activity which comes in between requirement analysis and programming or coding.
- It always refers to finding solutions or problem solving.
- It includes algorithms, diagram, formulae design, low level component, etc.
- It is one of the important phases in software development phase.
- Good design generally results in successful software implementation.
- It contains semi standard methods like UML. Unified modelling language is language in software engineering for modelling the software. It uses different diagrams like activity diagram, use case diagram and sequence diagram, etc.
- In software design process, the design patterns and architectural styles are also used.

7.3 Design Quality

University Question

Q. What are the software design quality attributes and quality guidelines? **SPPU : Dec. 19, 7 Marks**

Following are some important guidelines for evaluation of a good design :

7.3.1 Quality of Design Guidelines

- A design should be in such a way that gives recognisable architectural styles and patterns and should be composed of components of good design characteristics.
- A design should be modular in nature i.e. it consist of small partitions or sub systems.
- A design should represent data, architecture, interface and components in distinct way.
- A design should contain appropriate data structure and recognisable data patterns.
- A design should represent independent functional characteristics.
- A design should create interfaces that must reduce complexity of relations between the components.
- A design should be derived by using a reputable method.
- A design should use a notation that must lead to effective communication and understandings.
- These are some good design guidelines and should be used through the application of design principles and methodology.

7.3.2 The Quality Attributes

University Question

Q. Explain the quality attributes, considered in software design. **SPPU : Dec. 12, 8 Marks**

Following are some quality attributes that are given the name as **"FURPS"** define the good software design :

- **Functionality** is an important aspect of any software system. It is generally evaluated by the feature set and capabilities of that software.
- Following are general features that a system is supposed to deliver.
- **Usability** is best evaluated by considering following important factors :
  - Human factors
  - Overall aesthetics
  - Consistency and documentation.
- **Reliability** is assessed by measuring following parameters :
  - Frequency and severity of failure
  - Accuracy of output results
  - Mean-time-to-failure (MTTF)
  - Recovery from failure and
  - Predictability of the program
- **Performance** is evaluated by considering following characteristics :
  - Speed
  - Response time
  - Resource consumption

F - Functionality
U - Usability
R - Reliability
P - Performance
S - Supportability

- Throughput and
- Efficiency

- **Supportability** collectively combines following three important attributes:
  - Extensibility
  - Adaptability
  - Serviceability
- Most commonly, above three attributes can be better defined by the term **maintainability**. In addition to these attributes, following are some more attributes with which a system can be installed easily and the problems can be found out easily :
  - Testability,
  - Compatibility, and
  - Configurability.
- It also contains more attributes which are as follows :
  - Compatibility
  - Extensibility
  - Fault tolerance
  - Modularity
  - Reusability
  - Robustness
  - Security
  - Portability
  - Scalability

7.4 Design Concepts

Following is a set of fundamental software design concepts has evolved over the history of software engineering :

1. Abstraction	2. Architecture
3. Patterns	4. Modularity
5. Information Hiding	6. Functional Independence
7. Refinement	8. Refactoring
9. Design Classes	

7.4.1 Abstraction

- When we consider a modular solution to the problems, there are many levels of abstraction possible. In the highest level of abstraction, there is a solution that is stated in broad terms using the language of the problem environment.
- The lower levels of abstraction are used for detailed description and hence a more detailed description of the solution is provided at lower levels.
- A data abstraction is actually a collection of data that describes a data object.

7.4.2 Architecture

University Question

Q. Explain the following design concept : Architecture. **SPPU : May 13, 2 Marks**

- Software architecture means the complete structure of the software. In number of ways, this structure provides basis for conceptual integrity for a system.
- In its simplest form, the architecture is the complete structure or arrangement of the program components in such a way that they interact with each other. The data structures may be used by these components. All these components are the building blocks of the overall structure of the application being developed.
- One goal of software design is to derive an architectural framework of a system. The architectural design can be represented using one or more of a number of different models.

7.4.3 Patterns

- A pattern is a thing, which conveys the essence of a proven solution to a recurring problem within a certain context.
- Stated in another way, a design pattern describes a design structure that solves a particular design problem within a specific context.

7.4.4 Modularity

University Question

Q. Explain the following design concept : Modularity. **SPPU : May 13, 2 Marks**

- The modularity consists of software architecture and design patterns i.e. software is divided separately according to name and addresses of components and sometimes it is called as modules that are integrated to fulfil problem requirements.
- The modularity is defined as the modularization of a single attribute of software into number of small parts such that these parts can be easily managed.
- The number of variables, span of reference, number of control paths and overall complexity will make the understanding of the program nearly impossible.
- To explain this point in detail, consider the following discussion based on observations of human problem solving.
- Consider C(x) be a function of a problem x that defines the its complexity, and E(x) be a function of the problem that defines the effort required to solve a given problem x. Let there are two problems namely p_1 and p_2, if

$$C(p_1) > C(p_2) \quad \text{...(7.4.1)}$$

it follows that,

$$E(p_1) > E(p_2) \quad \text{...(7.4.2)}$$

- For a general case, this result is definitely obvious. But it takes more time to solve a difficult and complex problem. Here an interesting characteristic is uncovered through observations and experimentation in human problem solving i.e.,

$$C(p_1 + p_2) > C(p_1) + C(p_2) \quad \text{...(7.4.3)}$$

- The Equation (7.4.3) shows that the complexity of a problem that combines two problems p_1 and p_2 is greater than the complexity when the problems are considered separately. Now consider the Equation (7.4.3) and the condition implied by Equations (7.4.1) and (7.4.2), it follows that :

$$E(p_1 + p_2) > E(p_1) + E(p_2) \quad \text{...(7.4.4)}$$

- This equation leads to a strategy called divide and conquer. It is always easy to solve a complex problem when it is divided in small sub-problems that are easily manageable..
- The result expressed in Equation (7.4.4) has important implications with regard to modularity and software. It is an argument for modularity. It is possible to conclude from Equation (7.4.4) that if we subdivide software into smaller modules then the effort required to develop it will become considerably small.
- Referring to Fig 7.4.1 the effort (cost) to develop an individual software module does decrease as the total number of modules increases.
- For a given the set of requirements, the modules that can be broken into smaller manageable size. When the size of the problem is large, the number of modules will be increased and the effort (cost) associated with integrating the modules also increases.
- These characteristics are exhibited by using the following curve i.e. a total cost or effort curve shown in the Fig. 7.4.1. The M is number of modules that would result in minimum development cost, but there is no any proper and sophisticated method to predict M with assurance.

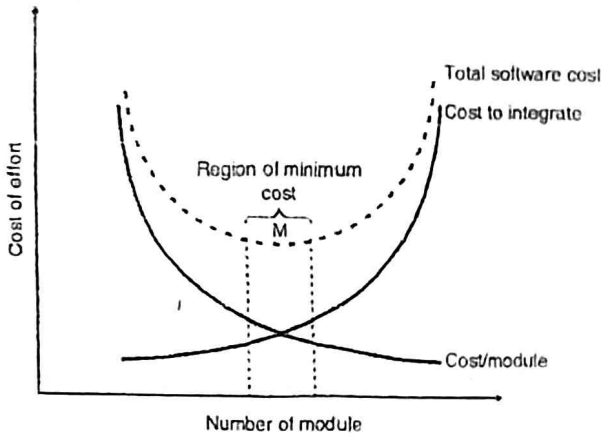


Fig. 7.4.1 : Modularity

- The curves shown in Fig. 7.4.1 provide some useful guidance when modularity is taken into consideration.
- We should divide the problem into modules, but utmost care must be taken to stay in the vicinity of M. It is always better to avoid under-modularity or over-modularity.
- There are five criteria defined for an effective modular system :

1. **Modular decomposability :** A complex problem may be solved very easily by decomposing into smaller sub problems. The concept of modular solutions makes it very easy and effective.
2. **Modular composability :** Different modules and components are integrated to produce a new system is called as modular composability.
3. **Modular understandability :** While integrating a system, if each of the individual modules or components are understood well, then it become very easy to build the system once again.

4. **Modular continuity :** It is always better to make changes module-wise instead of making changes system-wise. By doing this, the side effect of the changes can be minimized.
5. **Modular protection**
   - For any abnormal condition within a module, the effects are within that module only. Thus the side effects are minimized and problems will not spread outside that module. This phenomenon is called as modular protection.
   - After providing security to the modular design, it is also important to focus on rigidity in its implementation.
   - Any of the overhead caused by the sub programs is not acceptable. This overhead will affect the performance of the system.
   - In these situations, the products are designed using modularity and the coding of system is in-line.
   - The source code may not be seen modular initially but in-line code philosophy will be useful in the modular system.

7.4.5 Information Hiding

- The modules of the larger problem must be specified and designed properly so that information (i.e. algorithms and data) present within a module is not available to other modules that do not require such information.
- Hiding ensures that effective modularity can be achieved by defining independent modules that have proper communication among them and only that information necessary to achieve software function is shared.

7.4.6 Functional Independence

University Question

Q. What do you mean by the term cohesion and coupling in the context of software design? How are these concepts useful in arriving at a good design of a system? **SPPU - Dec. 16, 2 Marks**

- The functional independence concept is a type of modularity and related to the concepts of abstraction and information hiding.
- In functional independence each module addresses a specific sub function of requirements and has a simple interface. This can be viewed from other parts of the program structure.
- Independence is assessed using following two qualitative criteria :
  1. Cohesion
  2. Coupling
- Cohesion is a relative functional strength of a module and coupling is the relative interdependence among modules.

1. Cohesion
   - Cohesion is an extension of the concept of information hiding. In cohesion, a module performs only one task within a software procedure i.e. requires only a little interaction with procedures that are in other parts of a program.

- Cohesion can be represented as a "spectrum". High collision is always recommended but mid-range of the spectrum is often acceptable. The cohesion scale is nonlinear.
- That is, low-end cohesiveness is worse than middle range, which is nearly same high-end cohesion.
- In a regular practice, designers are not required to be bothered about categorizing cohesion in a particular module. Rather, the overall concept should be understood properly and low-end cohesion must be avoided when modules are designed.
- At the low-end of the spectrum which is actually not desirable, we encounter a module that performs different tasks that are loosely related to each other. These modules are termed as coincidentally **cohesive.**
- A module that performs tasks that are related logically is logically cohesive.
- When a module contains tasks that are related by the fact that all must be executed with the same span of time, the module exhibits temporal cohesion.
- Moderate levels of cohesion are close to each other in the degree of module independence. Procedural cohesion exists when processing elements of a module are related and are executed in a specific order. When all the processing elements do concentrate on only one area of a data structure, then communicational cohesion is available. High-end cohesion is characterized by a module that performs one single and unique procedural task.

2. **Coupling**

- Coupling is a measure of inter-relationship among various modules in a software structure. Coupling always depends on the interface complexity among modules i.e. the point at which reference or entry is made to a module. It also depends on what data is passed across the interface among modules.
- In software design, we look for minimum possible coupling. If the connectivity between different modules of software is made simple then it is bit easier to understand.
- The simple connectivity will also avoid "ripple effect" up to certain extent. The ripple effect is introduced when the errors occurring at one particular location in the system and propagating through a system.

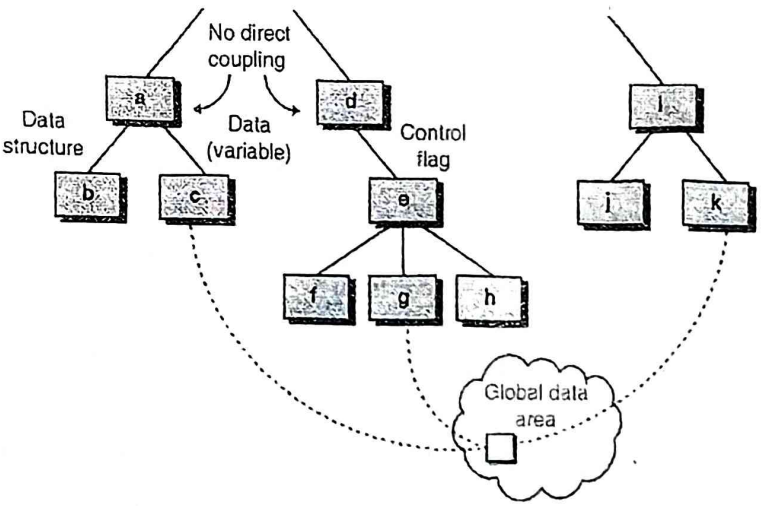


Fig. 7.4.2 : Types of coupling

In the Fig. 7.4.2, various examples of different types of module coupling are illustrated :

- Modules **a** and **d** are children of different modules. Each of them is not related to each other and thus there is no direct coupling occurs.
- Module **c** is child of module **a**. It is accessed through a conventional argument list. All the data are passed through this argument list.
- When the argument list is simple, the data can be passed and there is one to one correspondence between the items exist. The low data coupling or simply low coupling is demonstrated through this structure.
- When a part of data structure apart from simple argument, is passed through a module interface, then stamp coupling is exhibited. This is actually a variation of data coupling only. It is illustrated in the Fig. 7.4.2 and it occurs between **b** and **a**.
- The coupling is characterized by passage of control between modules at moderate levels. In most of the software designs, control coupling is common and is exhibited in Fig. 7.4.2 where a "control flag" is passed between modules **d** and **e**.
- When modules are tied to some environments that are external to software, then the high levels of coupling may occur. Like an I/O can couple a module to some devices, formats, and different communication protocols. This is external coupling and is very essential. It must be limited to only a small number of modules in a structure.
- Like low coupling, high coupling my also occurs when different modules reference a global data area. This is called common coupling also, and is shown in Fig. 7.4.2 modules **c**, **g**, and **k** share their data item in a global data area. The module **c** initializes the items.
- After that the module **g** computes and later on updates these items. Now assume that an error has been occurred and module **g** updates the item incorrectly.
- In processing the module, **k** reads the wrong item and during its processing, it fails and ultimately the software is aborted.
- The reason behind this abort is module **k** and the actual cause is module **g** since it has updated the item incorrectly.
- Diagnosing the problems is time consuming and difficult. The diagnosis in structures is always with the common coupling. The use of global data is not always bad, but the developer may use them in coupling. The designer should know well the impacts of these couplings and care against them to protect.
- The coupling occurs when a module uses the data present in some another module. It can use control information also during the coupling.
- The concept of content coupling used when all the subroutines or data structures within a module are compulsory asked to involve in coupling. The content coupling must be avoided
- The content coupling modes may occur due to design decisions made during development of structure. The variants of the coupling discussed, may be introduced during coding phase.

- For example : Compiler coupling ties source code to specific (and often nonstandard) attributes of a compiler; Operating System (OS) coupling ties design and resultant code to operating system "hooks" that can create havoc when OS changes occur.

7.4.7 Refinement

University Question

Q. Explain the following design concept : Refinement. SPPU : May 13, 3 Marks

- The refinement is software design concept that uses a top-down strategy. In this strategy, the procedural details of a program are refined in various levels.
- The hierarchy is produced in refinement process. This hierarchy is generated by modularizing the statements of a program, for example procedural abstraction. This modularization is completed step by step in a function or the program.

7.4.8 Refactoring

University Questions

Q. What do you understand by refactoring? SPPU : May 15, 3 Marks

Q. Explain refactoring. SPPU : Dec. 15, 3 Marks

An important design activity, refactoring, is a reorganization technique that simplifies the design (or code) of a component without changing its function or behavior or in other words, refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code [design] yet improves its internal structure.

7.4.8(A) Importance of refactoring

University Question

Q. Give the importance of refactoring in improving quality of software. SPPU : May 15, Dec. 15, 3 Marks

- The refactoring is used in improving the quality of software.
- Following are some points that prove the importance of refactoring :
  - By improving the quality of the code, working becomes easier. By using code refactoring, the addtion of new code and maintenance of the code becomes easy.
  - The programmer do not write perfect codes, hence refactoring improves the code over the period of time.
  - Refactoring play important role in splitting long functions into reasonably small sub functions.
  - It helps in creating reusable codes.
  - Error handling is much easier and within control.

7.4.9 Design Classes

- Each of the software team should define a set of design classes. All of these classes should describe the elements of problem domain and that should focus the customer requirement.

- A design class should :
  - Refine the analysis classes that are implemented.
  - Create a new set of design classes that must support the business solutions.

7.4.10 Differentiation between Abstraction and Refinement

University Question

Q. Differentiation between abstraction and refinement **SPPU : Dec. 12, 4 Marks**

Sr. No.	Abstraction	Refinement
1.	A description of something that omits some details that are not relevant called abstraction.	A detailed description that is even not relevant.
2.	It provides compact design process.	It allows more flexible design process.
3.	When we consider a modular solution to any problem, many levels of abstraction can be posed.	Stepwise refinement is a top-down design strategy.
4.	The highest level of abstraction generally states a solution by using the language of the problem environment.	A program is usually created by refining the levels of procedural detail.
5.	Similarly the lower levels of abstraction gives more detailed description of the solution. A data abstraction is nothing but a collection of data that explains a data object.	The hierarchy in refinement is created by partitioning a macroscopic statement of function i.e. a procedural abstraction. This partitioning is done stepwise till all the programming language statements are covered.

7.5 The Design Model

- The design model can be viewed in two different dimensions as illustrated in Fig. 7.5.1.
- The process dimension indicates the evolution of the design model as design tasks are executed as part of the software process.
- The abstraction dimension represents the level of detail as each element of the analysis model is transformed into a design equivalent and then refined iteratively.
- In the Fig. 7.5.1, the dashed line indicates the boundary between the analysis and design models.
- The elements of the design model use many of the UML diagrams. These diagrams are refined and elaborated as the part of design.

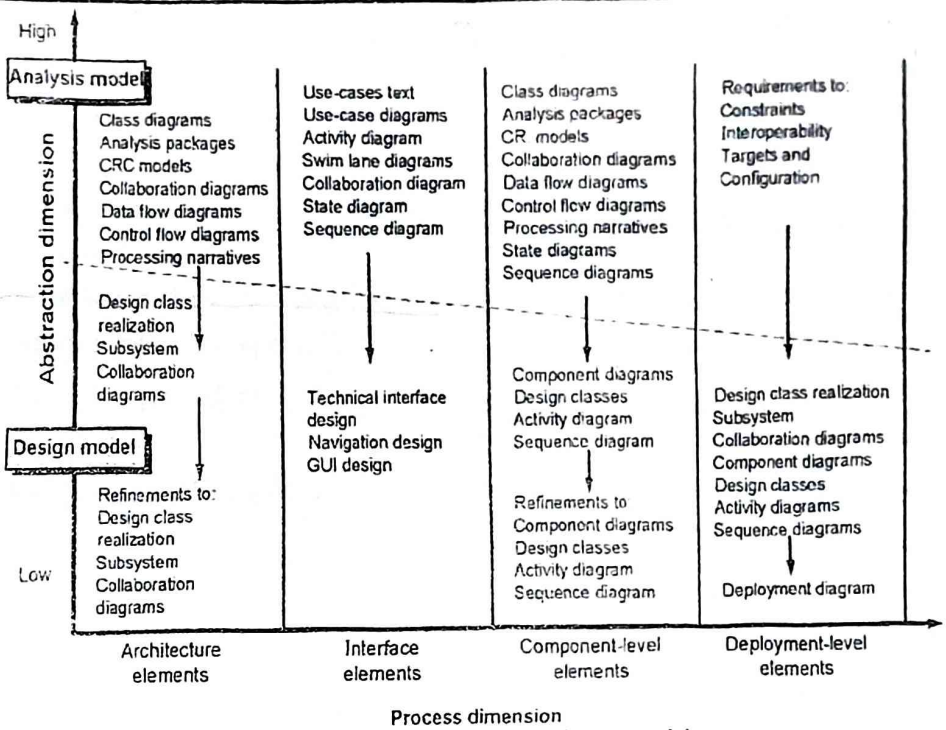


Fig. 7.5.1 : Dimensions of the design model

7.5.1 Data Design Elements

- There are several software engineering activities that are available and used throughout the software development processes. Similarly the data design or data architecting also creates a model of data and information that is presented at the highest level of abstraction. The highest level of abstraction refers to the customer view of data or the user's view of data.
- Now there is a refinement of data model into more implementation-specific representations and that can be processed by the computer-based system.
- In most of the software systems, the architecture of the data has a deep influence on the architecture of the software that processes it.

7.5.2 Architectural Design Elements

- The architectural design very similar to the floor plan of a building. The floor plan exhibits the overall layout of the rooms, the open spaces, the size of various room, their shape, dimension, and relationship among them and the doors and windows that provide entry and exit of the rooms.
- The floor plan provides the complete idea and view of the building. The architectural design elements also provide an overall view of the software.
- The architectural model is generally derived from following three main sources :
  - The information related to the application domain for the software.

- The analysis model elements like DFDs (Data Flow Diagrams), analysis classes and relationships and collaborations among them, and
- The architectural styles.

7.5.3 Interface Design Elements

- The floor plan of a building provides the drawings for various parts like doors, windows and various utilities for that building. In the same context the interface design for a software system provides such specification.
- The floor plan shows the size, shape, colour etc. of doors and windows as per their requirements. The door and windows must operate in the proper manner.
- The utilities like water connection, electrical, gas and telephone connections are spread among different rooms as per the floor plan.
- Following are three important elements of interface design :
  - UI (The User Interface)
  - External interfaces and
  - Internal interfaces among various design components.
- All these interface design elements helps the software to communicate internally as well as externally and enable collaboration among these components to describe the software architecture.

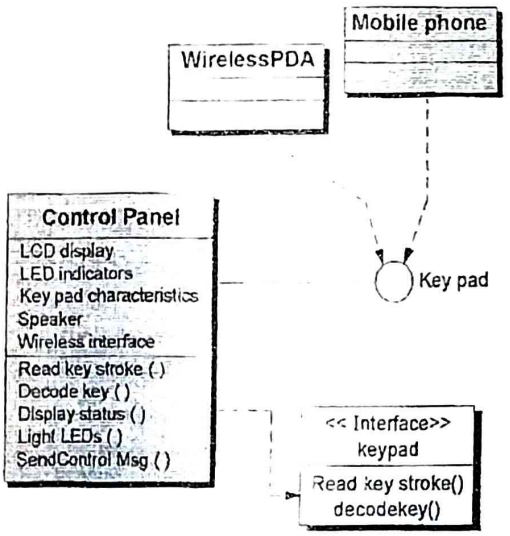


Fig. 7.5.2 : UML interface representation for control panel

7.5.4 Component-Level Design Elements

- If we consider the component-level design for software, then it is equivalent to the detailed drawings along with its specifications for each of rooms separately in a building.
- These drawings show various connections like electrical wiring and plumbing in each of the room separately. The location of switch board and other accessories in the room.

- The detailed design also describes which flooring is to be used, and which modeling is to be applied, and all the details related to room. Thus the component-level design for software completely explains the internal detail of each and every component of the software.
- In order to accomplish this, the component-level design gives the data structures for all local data objects along with algorithmic detail for all these processing that occurs within a component. It also elaborates the interface used to access to all component operations i.e. behaviours.

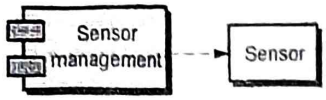


Fig. 7.5.3 : UML component diagram for sensor management

7.5.5 Deployment-Level Design Elements

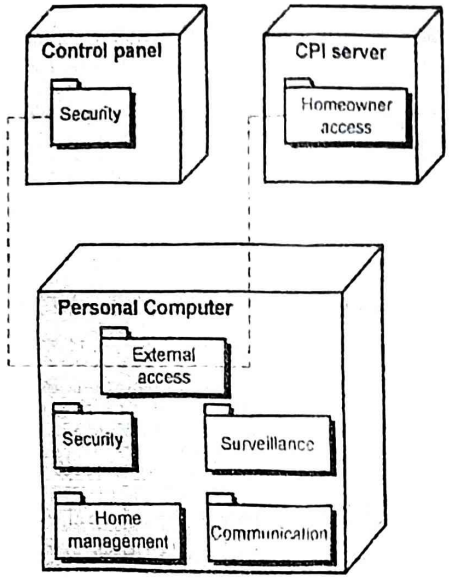


Fig. 7.5.4 : UML deployment diagram

- Deployment-level design elements indicate how software functionality and subsystems will be allocated within the physical computing environment that will support the software.
- During this design a UML deployment diagram is created as shown in the Fig. 7.5.4.
- It has three computing environment as shown below :
  - The **personal computer** that implements security, observation and management.
  - The **CPI server** manages the access rights.
  - The **control panel** manages the security.

7.5.6 Translating Requirements Model to Design Model

- The software design is residing actually at the technical kernel of software development and engineering and it is applied irrespective of the software process model that is employed.

- Starting with the requirement analysis and models, the software design is the last activity or the last action and it is actually the platform for coding and testing.
- The elements of the analysis model give the information essential to create the four design models that are required for a full specification of design.
- The flow of information during software design is illustrated in Fig. 7.5.5. The design produces data or class design, an interface design, an architectural design, and a component design.
- The data or class design is translated from analysis class models to design class realizations and the appropriate data structures required to implement the software.
- Here more detailed class design is created as each of the software components is developed.
- The relationship between major structural elements of the software is defined by the architectural design. The design patterns and the architectural styles can be used to achieve the requirements defined for the system.
- The interface design also describes the way software communicates with systems and with the end users who use it. An interface shows a flow of information.
- The component-level design translates structural components of the software architecture into a procedural description of software components.

7.5.7 Guidelines for the Data Design

- Following guidelines are implemented for a good data specification and good data design :
  - The systematic data design principles are applied to all the functions and their behaviour. These principles are applied to the data also.
  - All the data structures and operations to be performed must be identified.
  - A proper data dictionary should be established.
  - The date structures should be known to those modules only that use it.
  - A library of data structures and operations should be applied.
  - The design must be supported by the programming language.

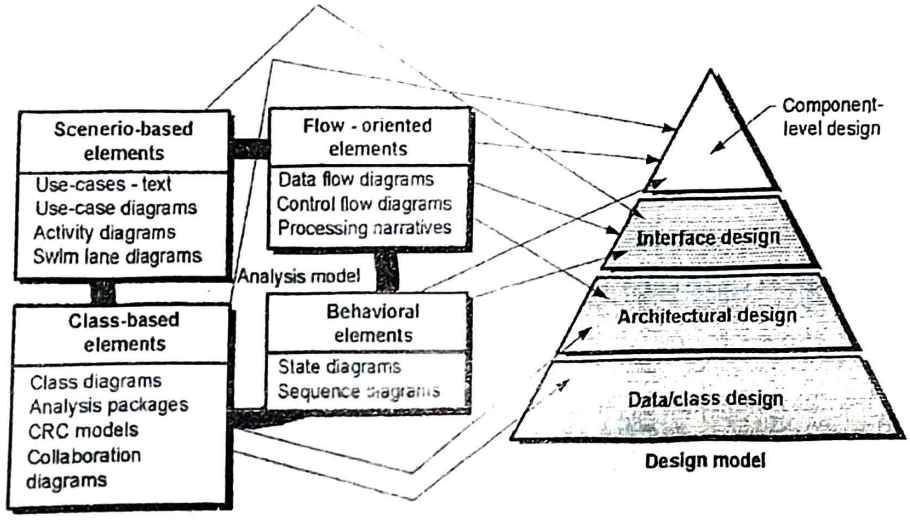


Fig. 7.5.5 : Translating the analysis model into the design model

7.6 Pattern-Based Software Design

- In software engineering, reusability is most important characteristic. The software developer looks for every opportunity to reuse the existing code.
- A good designer is the one who has the ability to see the pattern that characterizes a problem and the corresponding pattern available to combine in creating the final solution to the given problem.
- Throughout the software development process, the software developer always searches the existing design pattern to match with the pattern required in the solution.

7.6.1 Describing a Design Pattern

- In software engineering, developer uses thousands of design patterns. For example an electronics engineer uses some extremely complex integrated circuit (IC) to design some new circuit. Also a mechanical engineer can use pulley-based design pattern to engineer some new machine.
- The design pattern can be described using some standard template. This template can be described using following attributes :
  - **Pattern name** : The name should short, simple and effective to describe the design pattern, so that software developer should be able to use it quickly.
  - **Synonyms** : It should provide list of synonyms.
  - **Intent** : The intent of the design must be clear, what actually it does.
  - **Examples illustrated** : to motivate the use, illustrate the pattern by example.
  - **Structure** : It shows the classes required to implement the pattern.
  - **Responsibilities** : The responsibilities of the classes must be described.
- These are some examples of descriptions for a design pattern template. In addition to these attributes, collaborations, consequences, applicability etc. can also be mentioned.
- All these attributes actually represent the characteristics of the design pattern and any design pattern can be searched from the database using these attributes.

7.6.2 Using Patterns in Design

- In software design process, the software developer uses design patterns throughout the design process.
- The software engineer conducts a detailed examination and analysis on the given problem at various levels of abstractions to check whether any of the following design patterns can be used in the software design process to speed up the design process :
  - **Architectural Patterns** : This design pattern describes the overall structure of the software. The design pattern exhibits various relationships between subsystems and components. It also specifies the rules for these relationships.
  - **Idioms** : These are the language based patterns also called as coding patterns. The idioms usually implement the algorithmic element of a component. It also implements some mechanism for communication between components.

- **Design patterns** : It focuses on some specific element of design such as aggregation of components, in order to solve the design problems or the relationships among different components. It also focuses on component to component communication.

- Each of the above mentioned patterns differ in the level of abstraction and differ in degree to which it provides the guidance for coding in this example.

7.6.3 Frameworks

- For implementing the design work, sometimes it becomes essential to provide the implantation specific infrastructure. This infrastructure is called as framework.
- A framework is not considered as architectural pattern but it is a skeleton to be adapted to a specific problem domain. In object oriented environment, framework may be considered as the group of classes that cooperate to each other.
- To make the development process effective, frameworks are used as it is and some additional design elements are added complete the software design.

Review Question

Q. Differentiation between abstraction and refinement

Q. What are the guidelines that are implemented for a good data specification and good data design?

Q. Explain the quality attributes, considered in software design.

Q. Explain the following design concept : Modularity, Abstraction, Refinement.

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

Q. Explain the following design concept : Architecture

Q. What do you understand by refactoring?

Q. Explain refactoring.

Q. Give the importance of refactoring in improving quality of software.

Q. What are different dimensions of design model?

□□□

8

Unit - III

ARCHITECTURAL DESIGN & COMPONENT LEVEL DESIGN

Syllabus

Design Decisions, Views, Patterns, Application Architectures.

8.1 Introduction to Architectural Design

- Architectural design is backbone of any software system design and it is responsible for the overall structure of the system.
- In nearly all the models of software process, architectural design is considered as the first stage in the software design and development process.
- The output of the architectural design process is an architectural model that explains the overall structure of the system and also explains the working of the system and the communication among various components of the system.
- In case of agile development processes also, the first stage of development process is defining the system architecture. Incremental approach of the software development is not generally successful whereas the concept of refactoring of the components bit easy. But refactoring of the system architecture is expected to be expensive.
- In order to understand the system architecture, Fig. 8.1.1 provides a simple model of architecture for a remote control car :

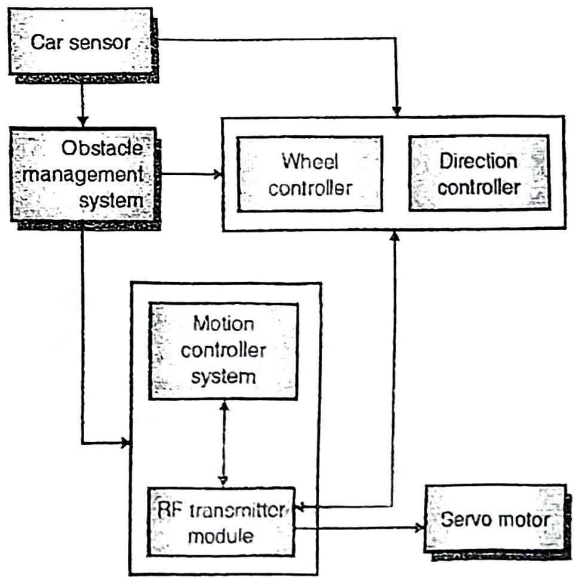


Fig. 8.1.1 : The architecture of a remote control car

- The Fig. 8.1.1 exhibits the abstract model of architecture of a remote control car system that will help to understand the functioning of the car.

- In the Fig. 8.1.1 shown, the interconnection betweer the components used to develop the remote control car system.
- It uses RF transmitter module to actually control the complete system. The sensor present in the car acts as RF receiver module which accepts the commands sent from the RF transmitter module.
- Various controllers listed below are actually used to control the functioning of the remote control car system :
  - Wheel controller
  - Direction controller
  - Motion controller, and
  - Obstacle management system that controls the obstacle and changes the direction to avoid the collision.
- The most important part of the remote control car system is "Servo Motor" that is responsible for overall functioning of the system.
- In actual practice, there is some similarities and overlap among the requirement engineering process and the system specification. The architectural design and system specification should avoid including the design information.
- Instead architectural decomposition is essential to organize the system specification and to provide the structure of the system.
- Therefore, as part of the requirements engineering process, the abstract system model is proposed so that various functions and feature is shown clearly. Therefore, the decomposition is used to show the requirements and features of the system with stakeholder (e.g. customer, developer, designer or the end users).
- Normally software architecture can be designed into two levels of abstraction as follows :
  - Small software design architecture
  - Large software design architecture
- **Architecture in the small** is related to the architecture of individual programs which may be small in size. This program is decomposed into smaller components. This type of architecture is related to mostly program architectures.
- At this level, we are concerned with the way that an individual program.
- **Architecture in the large** is related to architecture of complex systems that includes overall system, the entire program and the components of the program.
- **Software architecture** is an important characteristic since it is responsible for various features like performance, robustness, distributability, and maintainability of a system. The distributability feature is mainly the part of distributed systems over the networked lines.
- The non-functional system requirements are totally depend on the system architecture whereas functional system requirements are dependent on the individual components.
- Following are some of the advantages, if a software designer properly design, document and implement the software architecture :
  - **Communication among various stakeholders :** Actually any architecture is considered as the highest level of abstraction and presentation of the system is important and it can be focused by discussion among various stakeholders of the system.

- **System analysis :** If we want to propose system architecture explicitly in the beginning only, then software development requires more analysis.
- Mostly the system architectural design decisions have greater impact on achieving the critical requirements like performance, reliability, and maintainability.
- **Reusability at large-scale :** A model or the system architecture is considered as compact and manageable description of the system. For all the systems having similar requirements uses same system architecture and this is the reason why it can support reusability at large scale.

8.2 Architectural Design Decisions

- Architectural design is assumed to be an intellectual process and a software organization tries its level best to satisfy the functional requirements and non-functional requirements as well.
- All the activities present within the process depends on the type of the system under construction and specific requirement elicited by the customers. It also depends on the system architect previous experiences.
- Therefore the architectural design is not a sequence of activities rather it is assumed to be the series of decisions made.
- Thus the architectural design process requires number of structural decisions that have impact on the system and its development processes.
- Based on the past experiences of architectural designers, they have various questions about the system and are listed below :
  - Is there any generic architecture present, so that it can be used as template for the system development ?
  - How the components of the architecture can be decomposed into subcomponents?
  - What architectural organization is best for delivering the non-functional requirements of the system?
  - Which strategy will be best suited to control the operation of various components of the system ?
  - How the evaluation of architectural design is done ?
  - How documentation is carried out ?
  - What will be the basic methodology to design the system ?
  - What are the architectural patterns that can be used ?
  - How distributability feature is implemented ?
- Even if every software system is unique one, but the system with the same domain have similar design architectures.
- Therefore it is evident that software developer or the software designer decides what parts of the architectural design can be reused during the development process.
- For embedded systems and systems designed for personal computers, there is usually only a single processor and you will not have to design a distributed architecture for the system. However, most large systems are now distributed systems in which the system software is distributed across many different computers.
- The architecture of a software system may be based on a particular architectural pattern or style. An architectural pattern is a description of a system organization, such as a client-server organization or a layered architecture.

- Since there is a close relationship between non-functional requirements for a software system, the software architecture, architectural style and structure depends on the non-functional system requirements. We observe following non-functional requirements :
  - Performance
  - Security
  - Safety
  - Availability
  - Maintainability
- Evaluating an architectural design is difficult because the true test of architecture is how well the system meets its functional and non-functional requirements when it is in use.
- However, you can do some evaluation by comparing your design against reference architectures or generic architectural patterns. The description of the non-functional characteristics of architectural patterns can also be used to help with architectural evaluation.

8.3 Architectural Views

- The architectural model of the system focuses on the functional and non-functional requirements and design. Also the architectural design is well documented for further design of any software system and implementation of the system in future.
- The architectural design can also be used as the foundation for the evolution of the system. Following are two important issues that are related to the architectural design :
  1. What views of the architectural design are actually useful in designing and documenting the system's architecture?
  2. What are the notations that will be used to describe the architectural model or architectural design of the system ?
- It is highly impossible to describe all the information about the architecture of the system in one single model since each of the model represent or show only one view of the system.
- Some of the models represent how a system is decomposed into different modules, some of them show how different processes interact each other during the run time, some of them might represent that how system components can spread across the network using distributed system.
- In all of the above case, all these views are extremely useful in different situations. Hence both the design and documentation must be represented using multiple views of the software architecture.
- There are different opinions as to what views are required in a well-known 4+1 view model of software architecture which suggests that there should be four fundamental architectural views, which are helpful in different scenarios.
- Software architecture takes into account the design and implementation of the software structure.
- From different architectural elements, the architecture is chosen to satisfy functionality and performance requirements. The non-functional requirements are also taken into consideration like scalability, reliability, availability and portability.

- Software architecture also deals with the style and aesthetics. In the Fig. 8.3.1 software architecture is illustrated as a model presenting five main views.

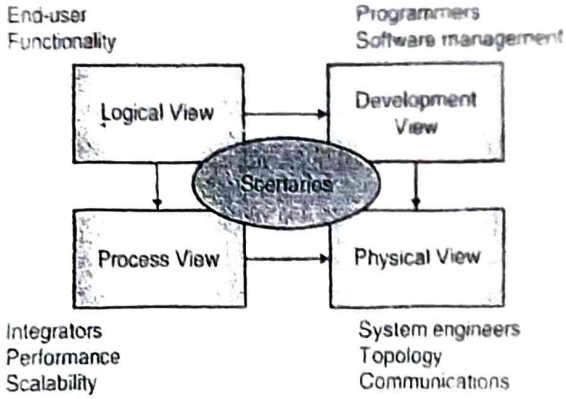


Fig. 8.3.1 : 4 + 1 View Architecture

- **The logical view**, which is the object model of the design (when an object-oriented design method is used),
- **The process view**, which captures the concurrency and synchronization aspects of the design,
- **The physical view**, which describes the mapping(s) of the software onto the hardware and reflects its distributed aspect,
- **The development view**, which describes the static organization of the software in its development environment.
- The description of architecture, the decisions made can be organized around these four views, and then illustrated by a few selected use cases, or scenarios which become a fifth view. The architecture is in fact partially evolved from these scenarios as we will see later.
- The "4 + 1" view model is quite "generic" : other notations and tools, other design methods can be used, especially for the logical and process decompositions, but we have indicated the ones we have used with success.
- In actual practice, the conceptual views are developed during the design process and are very much useful in supporting the architectural decision making.
- There is some form of communication in a system to various stakeholders of the software system. During the design process, as different scenarios come across, multiple views for those different scenarios or for different aspects of the system may also be developed.
- It is not possible at all to cover all the aspects of the system in a single description but it is possible to use architectural patterns or architectural styles for different views of a system.
- A software architects can also use the UML for architectural description. The UML was designed for describing object-oriented systems and, at the architectural design stage, you often want to describe systems at a higher level of abstraction. Object classes are too close to the implementation to be useful for architectural description.
- Many researchers have proposed the use of more specialized architectural description languages (ADLs) in order to describe system architectures. The basic elements of ADLs are components and connectors, and they include rules and guidelines for well-formed architectures. However, because of their specialized nature, domain and application specialists find it hard to understand and use ADLs.

- Users of agile methods claim that detailed design documentation is mostly unused. It is, therefore, a waste of time and money to develop it. Most of the developers agree with this view and they think that, for most systems, it is not worth developing a detailed architectural description from these four perspectives.
- We should develop the views that are useful for communication and not worry about whether or not your architectural documentation is complete. However, an exception to this is when you are developing critical systems, when you need to make a detailed dependability analysis of the system. We may need to convince external regulators that your system conforms to their regulations and so complete architectural documentation may be required.

8.4 Architectural Patterns

University Question

Q. Discuss architectural patterns in details. **SPPU : Dec. 16, 6 Marks**

- The concept of patterns is well known for presenting, sharing, and reusing knowledge about software systems. This is more commonly used nowadays. Architectural patterns were proposed earlier in the 1990s.
- An architectural pattern can defined as a stylized, abstract description of good practices, which has been tried and tested on various systems and in different environments.
- Therefore, an architectural pattern describes a system organization that was already proved successful in various past systems.
- The knowledge of strengths and weaknesses of pattern must be documented properly and also there should be a proper documentation that in which situation, it is not proper to use the said pattern.

8.4.1 Software Architecture

University Question

Q. What do you mean by software architecture ? **SPPU : Feb. 16, 2 Marks**

- When we discuss the architecture of a building, it is the manner in which the various components of the building are integrated to form a complete structure.
- It is the way in which the building fits into its environment and meshes with other buildings in its vicinity.
- Also visual impact of the building, and the way textures, colours and materials are combined to create the external appearance and the internal living environment.
- Similarly we can define the software architecture concept. The software architecture of a program or computing system is actually the structures of the system that consists of following attributes:
  - Software components,
  - The externally visible properties of software components, and
  - The relationships among them.
- The architecture is basically not the operational software. But, it is a representation only that enables a software engineer to :
  - Analyze the effectiveness of the design.
  - Consider architectural alternatives, and
  - Reduce the associated risks.

- This definition of software architecture focuses on the role of "software components" in any architectural representation. A software component can be as simple as a program module or an object oriented class, in the context of architectural design. The architecture can also be extended to include databases and middleware that enables the configuration of a network having clients and servers.

8.5 Application Architectures

- Any software application is developed to satisfy the business or organizational needs. If we observe, then we conclude that all the businesses are common in their functioning.
- For example :
  - Any business needs people to run it.
  - It generates invoices for the sales.
  - It keeps track of the accounts.
  - It keeps track of the delivery of the products ordered.
  - Facilitates online payments.
  - Provides facility to accept the returns etc.
- The list is endless.
- Now consider the example of mobile phone companies. They provide following functionalities :
  - Connect to calls
  - Manage the mobile towers and their networks.
  - Generate bills for the customers.
- If we observe these two businesses, then we conclude that both the businesses have many things in common. This commonness actually triggers the development of software architectures that describes the structure and organization of particular types of software systems.
- The application architecture includes basic the characteristics of these systems or the class of the system and thus the common architectural structure can be reused when developing new systems of the same type.
- The application architecture may be re-implemented when developing new systems but, for many business systems, application reuse is possible without reimplementation. For example, Enterprise Resource Planning (ERP) systems from companies such as SAP and Oracle, and vertical software packages (COTS) for specialized applications etc., in different areas of business. In these systems, a generic system is configured and adapted to create a specific business application.
- Consider the example of supply chain management system providing different types of products. Any business involved in supply chain management system with some different products can use the same ERP for their business.
- Thus a software designer can use the application model without any hesitation for the systems which share common business rules and policies.

- There are some application systems that look very different but many of these superficially dissimilar applications actually have lot of common functionalities in them. Thus these types of dissimilar businesses can adapt the same application architecture.
- Following are good examples of such types of applications, we describe their architectures :
  - Transaction processing systems
  - Language processing systems

8.5.1 Transaction Processing Systems

- Transaction processing systems are developed to facilitate the transaction executed by the customers. It process customer requests for information from a database, or requests to update a database (e.g. **online banking systems**).
- Basically, a database transaction is a series of operations performed by the user and it is considered as one single transaction. All of the operations in a transaction have to be completed before the database changes are made permanent.
- The atomicity property of database system ensures that failure of the system or the failure of transaction should not lead to any inconsistencies in the database.
- Consider the very famous example of withdrawing money from ATM banking system that uses the following sequence of the operations :
  - Get details of the customer's account from the database
  - Check account balance
  - Update the balance after withdrawal
  - Send the command to release the requested cash
- Until and unless all these steps are completed successfully, the transaction is supposed to be incomplete and the account balance remains same as it was before the start of the transaction.
- Usually all transaction processing systems are interactive in nature where users make asynchronous requests for the services.
- Fig. 8.5.1 exhibits the conceptual architectural structure of transaction processing system for ATM banking system.

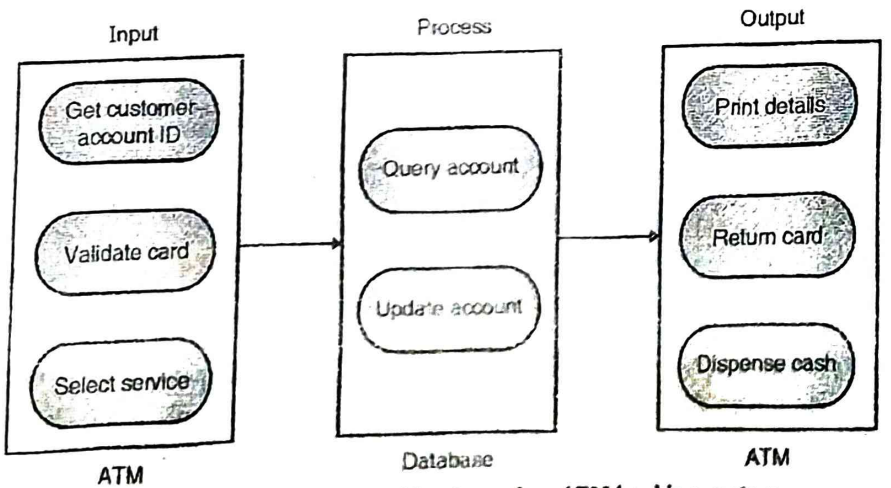


Fig. 8.5.1 : The software architecture of an ATM banking system

- The sequence of operations is illustrated in the Fig. 8.5.1 for the software architecture of an ATM banking system.
- The input and output components are implemented as software in the ATM and the processing component is part of the bank's database server. The architecture of this system shown, illustrates the functions of the input, process, and output components.

8.5.2 Language Processing Systems

- The main task of "language processing systems" translates a natural or artificial language into another language. Also programming languages may also process the sequence of instructions and produce the code.
- In software engineering, compilers translate an artificial programming language into machine code. Other language-processing systems may translate an XML data description into commands to query a database or to an alternative XML representation.
- Natural language processing systems may also translate one natural language to another natural language e.g., English to Marathi or Hindi.
- The architecture for a language processing system for a programming language is explained in the Fig. 8.5.2. The source language instructions define the program to be executed and a translator converts these into instructions for an abstract machine.
- These instructions are then interpreted by another component that fetches the instructions for execution and executes them using data from the environment. The output of the process is the result of interpreting the instructions on the input data.

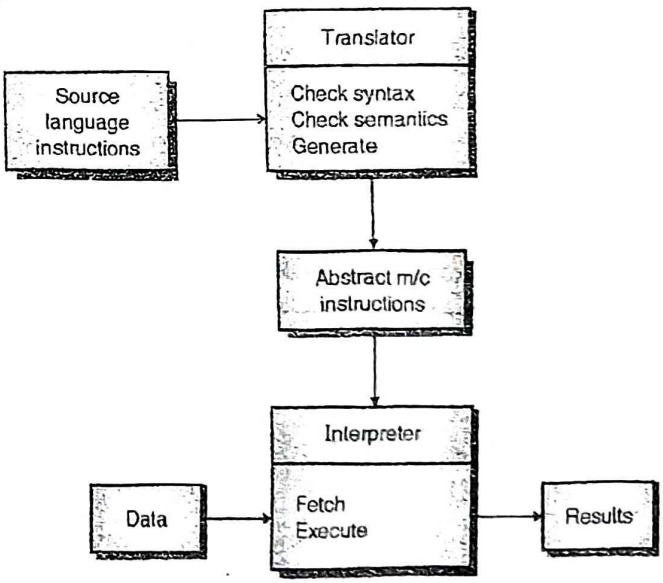


Fig. 8.5.2 : The architecture of a language processing system

- The compilers have a generic architecture that includes the following components :
  - **A lexical analyzer** : It takes input language tokens and converts them to an internal form.
  - **A symbol table** : It holds information about the names of entities (variables, class names, object names, etc.) used in the text that is being translated.

- **A syntax analyzer :** It checks the syntax of the language being translated. It uses a defined grammar of the language and builds a syntax tree.
- **A syntax tree :** It is an internal structure representing the program being compiled.
- **A semantic analyzer :** It uses information from the syntax tree and the symbol table to check the semantic correctness of the input language text.
- **A code generator :** It 'walks' the syntax tree and generates abstract machine code.

- The generic architecture for the compilers used in the programming languages shown in the Fig. 8.5.3 :

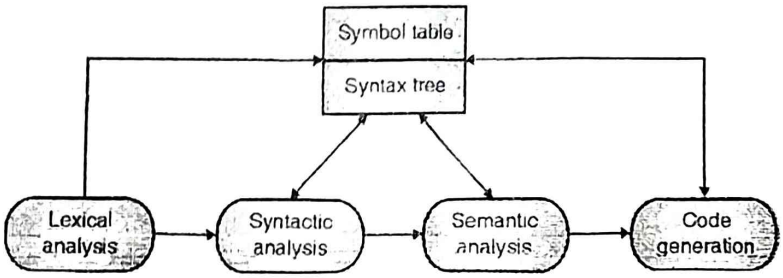


Fig. 8.5.3 : A pipe and filter compiler architecture

- There are alternative architectural patterns that may be used in a language processing system. Compilers can be implemented using a composite of a repository and a pipe and filter model.
- In compiler architecture, the symbol table is a repository for shared data. The phases of lexical, syntactic, and semantic analysis are organized sequentially, as shown in Fig. 8.5.3, and communicate through the shared symbol table.
- This pipe and filter model of language compilation is effective in batch environments where programs are compiled and executed without user interaction; for example, in the translation of one XML document to another.
- It is less effective when a compiler is integrated with other language processing tools such as a structured editing system, an interactive debugger. In this situation, changes from one component need to be reflected immediately in other components. It is better, therefore, to organize the system around a repository, as shown in Fig. 8.5.4.

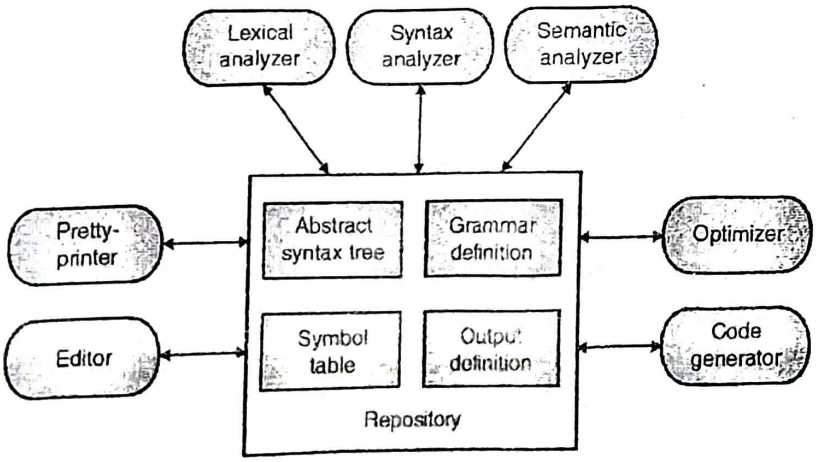


Fig. 8.5.4 : A repository architecture for a language processing system

- This Fig. 8.5.4 illustrates how a language processing system can be part of an integrated set of programming support tools. In this example, the symbol table and syntax tree act as a central information repository. Tools or tool fragments communicate through it.
- Other information that is sometimes embedded in tools, such as the grammar definition and the definition of the output format for the program, have been taken out of the tools and put into the repository

8.6 Conducting Component level Design

- Component level design comes after architectural design is completed. It is possible to represent the component level design by using some programming languages these programs can be created by using the architectural design model.
- In fact component level design is an alternative approach to the architectural design approach.
- A component is the basic building block of a software system, it refers the OMG specifications and is defined as a modular and deployable part of a computer system and it consists of different sets of interfaces.
- Following are different views of a component design :

1. An object oriented view
2. The conventional view
3. The process related view

1. An object oriented view
   - In object oriented software engineering a component is a set of classes which defines the attributes and the operations with respect to implementations.
   - It consists of all interfaces i.e. Messages that help to communicate and collaborate a different design classes.
   - To explain the process of design a software engineer collects the customer's requirements and performs requirement engineering and analysis. He defines the attributes and operations during analysis and architectural design.
   - The component is defined within the software architecture giving its attributes and operations and the details of components is enough to implement.
   - The object oriented software consists of details of all the messages used for communication between the classes.

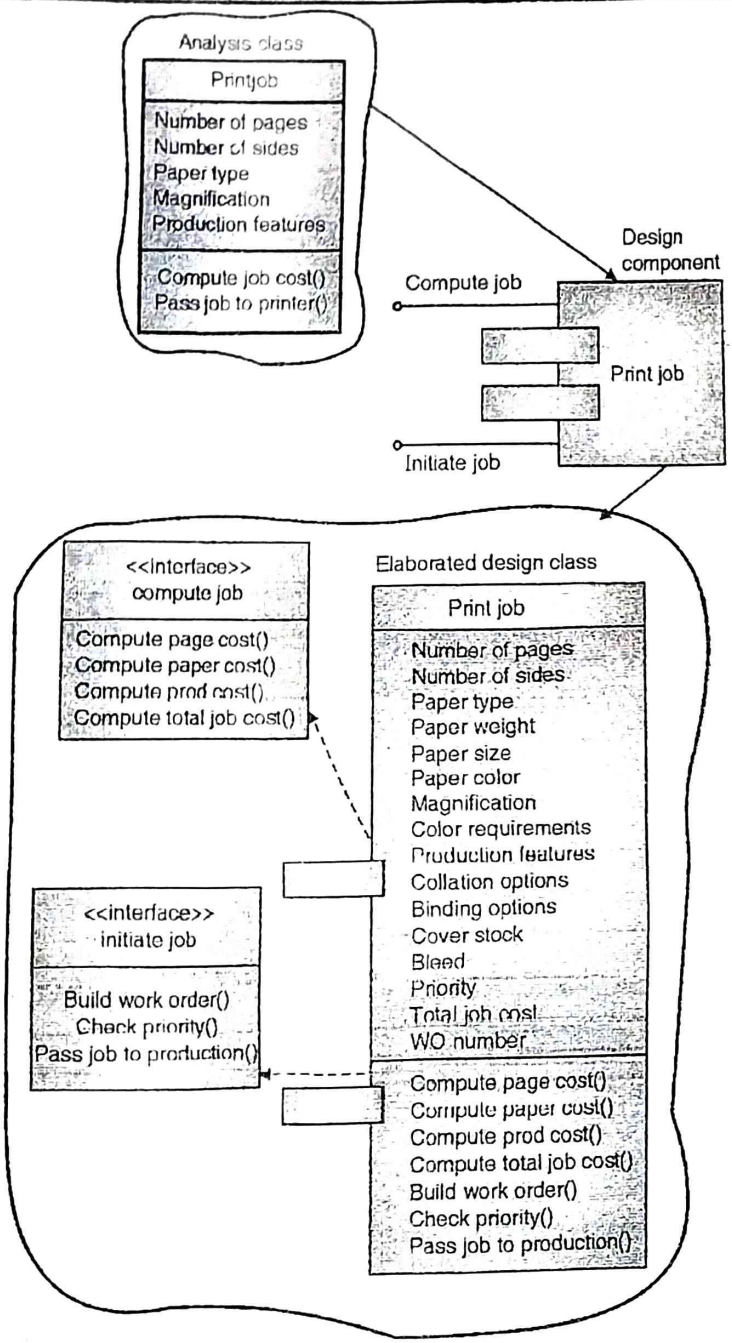


Fig. 8.6.1 : Elaboration of design component

2. **The conventional view**

- In conventional method for software engineering, a component consist of a functional element of a program or a module that has processing logic, data structures required to implement the interfaces between the components.
- To explain the process of design for conventional components we consider a sophisticated photo copying of analysis modeling discussed in architectural design.
- In component level design each subsystem is illustrated in the Fig. 8.6.2.

- The data flow and control object and their interfaces are illustrated very well in the diagram. The data structure is also explained properly and the algorithm allows to complete its intended function using stepwise refinement approach.

3. **The process-related view**

- In process related approach, the main focus is on the reusability of the existing software components.
- As the architectural design is ready, components are chosen and the complete descriptions of their interface are illustrated and are available to the designer.

8.7 Designing Class-based Components

- As we have discussed earlier that object oriented software engineering approach focuses on component level design and analysis classes and interfaces between the classes, in class based component the detailed description of attributes, operations and their interfaces are emphasized.
- The design details are discussed in the following sections.

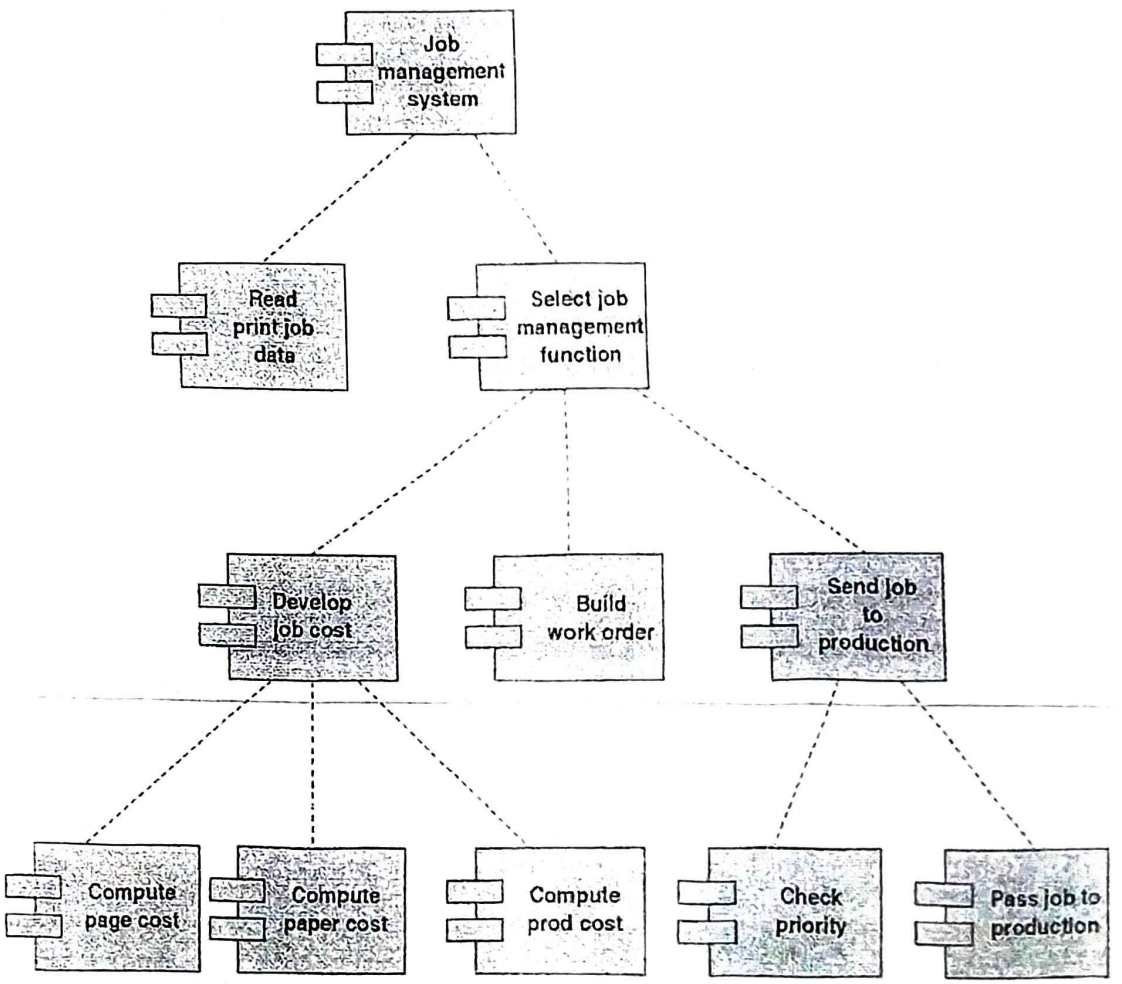


Fig. 8.7.2 : Structure chart for a conventional system

8.7.1 Basic Design Principles

Following are some basic design principles for class-based components :

1. **The Open-Closed Principle (OCP) :** In OCP any module must be available for extension and modification. The designer should specify the component in such a way that it can be extended without any modification in the internal structure of the component.
2. **The Liskov Substitution Principle (LSP) :** In this principle the subclasses should be substitutable for the base classes. This particular principle is suggested by Liskov. The LSP suggests that any class derived from the base class must imply the contract between the base classes and their components.
3. **Dependency Inversion Principle (DIP) :** In DIP the design depends on abstractions and not on concretion. The abstractions are the place where design is allowed to extend without any further complications.
4. **The Interface Segregation Principle (ISP) :** In this principle many client specific interfaces are better than general purpose interfaces. The component level design are organised into sub-systems or packages and suggests additional packaging principle for each component.
5. **The Release Reused Equivalency Principle (REP) :** In this the reuse of each module is actually the release of module. The classes or components are designed for reuse. It is always better to make a group of reusable classes i.e. packages that are easy to manage.
6. **The Common Closure Principle (CCP) :** In this principle the classes that changed together belongs together i.e. there is a cohesion between the classes.
7. **The Common Reused Principle (CRP) :** In CRP the classes that are not reused together, they should not be group together also.

8.7.2 Component-Level Design Steps

Following are the component-level design steps for an object oriented systems :

- Identify those design classes that are related to problem domain only.
- Now identify those classes that are related to infrastructure domain. The classes and components in this step include GUI components, OS components and object and data management components.
- Identify all the design classes that do not have reusable components and elaborate these design classes. In this step, the message details are also specified when the components or the classes collaborate. Also identify proper interfaces for each of the components and elaborate their attributes and define the data types of the attributes and also define the data structures necessary to implement the components.
- Describe the databases and files i.e. persistent data sources. Identify the classes required to manage these persistent data sources.
- Develop the behavioural representations for classes or the components and elaborate them appropriately.
- Elaborate the deployment diagrams in order to give additional implementation details.
- Always refine every component level design and consider alternative design solutions also.

Review Question

Q. Discuss architectural patterns in details

Q. What do you mean by software architecture?

Q. Write a short notes on : Architectural Views

Q. Explain software architecture with an example of an ATM banking system

Q. Explain Language Processing Systems with an example

Q. Draw repository architecture for a language processing system.

Q. What are Component-Level Design Steps?

□□□

9 USER INTERFACE DESIGN

Unit - III

Syllabus

The golden rules, Interface Design steps & Analysis, Design Evaluation.

9.1 User Interface Design

- Success of most of the software depends upon user interface design.
- It is part of human computer interaction. In user interface human controls the software.
- As two different entities are trying to communicate with each other, one is human being and other is computer.
- We need to understand nature of human being designing user interface.
- Many factors are considered before designing user interface which includes type of user, age of user, education level, purpose of software, etc.
- User interface design is created by considering human type also like child, grown up person, senior citizen, educated person, non educated persons, etc.
- Good interface design is user friendly and user can communicate very easily with that software.
- User interface can be at the hardware and software level also.
- At hardware level, type of button provided can be considered as user interface. E.g. in washing machine, button provided, display provided can be considered as hardware user interface.
- At software level we can provide user interface using programming. Different menus, options, radio button, list boxes, text boxes, button, etc are provided at the software for user interface.
- If the software is used by illiterate person then user interface should be in terms of pictures and some visual effects.
- If user is disabled then we need to design interface accordingly. E.g. for blind users we need to provide interface using sound effects. Different biometric factors can be used for user interface design for disabled users.
- User convenience is given at most priority in user interface design.
- If interface is designed for mobile users then also we need to consider different factors like type of mobile, type of user, internet bandwidth, battery life, processing power, screen size, type of input, etc.
- It needs good understanding of user needs.
- In this topic we focus on user interface design of software and not of the hardware.
- Various aspects of interface design has been discussed. Different principles, methods are discussed.
- Plenty of examples are taken for better understanding.

9.1.1 Type of User Interface

- It means how user can communicate with computer systems. Two fundamental approaches are used to interface with the computer system. User can communicate with computer system using the command interface and graphical user interfaces.
- User can communicate with computer using different commands provided by the system. In another approach user interact by using user friendly graphical user interface. Graphical user interface provides the menu and icon based facility which can be used using keyboard and mouse.

1. Command Interpreter
2. Graphical User Interfaces

1. **Command Interpreter**
   - In command interpreter, user communicates with computer system with the help of commands.
   - In Unix/Linux system the command interpreter is known as shell. Shell is nothing but the interface between user and operating system.
   - All the shells give similar facility with only minor changes. Selecting the particular shell totally depends on user's requirement. Command interpreter just accept the commands, call the corresponding program and execute it.
   - When we type the "cd" command, operating system search the program for the "cd" command and get it executed. Many commands are provided by the operating system. The kind of command supported is depends upon the operating system.
   - The commands are implemented in different manner. In the first approach, the command interpreter itself contains the code for the command.
   - Whenever we type any command it will execute directly. The size of command interpreter depends upon the number of command supported by the command interpreter.
   - In the second approach separate program is created for each and every command supported by the system. Whenever we need to execute any command operating system search the program for it, load it into memory and get it executed.
   - The program for commands is stored on certain locations so that operating system can search it quickly. The approach is used in Linux system.
   - In Linux system all the programs for the command are stored in the "/bin" directory. In short system provides program for each and every command provided.
   - Executing the command means executing the corresponding program.
2. **Graphical User Interfaces (GUI)**
   - Another approach to interface with the computer system is through a user friendly graphical user interface or GUI.
   - Instead of directly entering commands through a command-line interface, a GUI allows a mouse-based, window-and-menu based system as an interface.

- GUI gives the desktop environment v here mouse and keyboard can be used. Mouse can be moved to certain location and when we click on some location operating system calls the corresponding program. Depending on the mouse pointer location, mouse click will invoke the program to execute. For every mouse click the corresponding program is start executing.
- GUI was initially used in the Xerox Alto computer. But the popularity of GUI is after 1980 in apple Macintosh operating system.
- Macintosh operating system had many changes in the initial GUI.
- Microsoft's first version of Windows-version LO-was based upon a cur interface to the MS-DOS operating system
- Two kinds of interfaces are available in UNIX system which is X-Windows systems and Common Desktop Environment (CDE).
- Various open source project have made a significant development in GUI environment. The popular open source projects are K Desktop Environment (or KDE) and the GNOME desktop by the GNU project. The KDE or GNOME works on UNIX and Linux systems.
- Which method of interface to use is totally depends on the user preferences. Generally Unix/Linux user uses the command line interface as it provides very powerful collection of shell commands. Most of Windows user uses the GUI environment which is very much user friendly one and even they are not aware about command based interface.

More interface types

- Direct manipulation interface
- Web based interface
- Touch screens
- Touch user interface
- Hardware interface
- Attentive user interface
- Batch interface
- Conversational user interface agent
- Crossing based interface
- Gesture interface
- Intelligent user interface
- Multi-screen interface
- Non command user interface
- Object oriented user interfaces
- Tangible user interface
- Text based user interface
- Voice user interface
- Natural language based interface

9.1.2 Characteristics of Good User Interface

- There are different characteristics of good user interface :
  - Clarity
  - Concision
  - Familiarity
  - Responsiveness
  - Consistency
  - Aesthetics
  - Efficiency
  - Attractive
  - Forgiveness

9.1.3 Benefits of Good Interface Design

- Higher revenue
- Increased user efficiency and satisfaction
- Reduced development costs
- Reduced support costs.

9.2 The Golden Rules

University Questions

Q	State and explain any four principles.	SPPU : Dec. 13, 6 Marks
Q.	What are the interface design principles and guidelines?	SPPU : May 14, 8 Marks
Q.	Enlist the golden rules for User Interface Design.	SPPU : Dec. 19, 6 Marks

- The following three rules form the basis for a set of user interface design principles and guidelines :
  1. Place the user in control
  2. Reduce the user's memory load
  3. Make the interface consistent
- These golden rules and guidelines actually form the basis for a set of user interface design principles thatguide this important software design action.

9.2.1 Place the user in Control

- The main motive behind using interface constraints and restrictions are to simplify the mode of interaction
- In most of the cases, the developer may introduce constraints and limitations to simplify the implementaton of the interface.
- The ultimate result could be an interface that is easy to develop, but frustrating to use.
- There are a number of design principles that allow the user to maintain control.

- **Define the interaction modes in such a way that the user is not forced to perform unnecessary or undesired actions :**
  - Consider the example of word processor for spell check. If user selects spell check in its current interaction, then a word-processor menu should moves it to a spell-checking mode.
  - The software should not force the user to remain in spell-checking mode if the user wishes to make some small text edit along the way. There should be less effort in entering and exiting spell check mode.
- **The interaction should be flexible**
  - For different users, different interaction preferences must be provided. For example, the software should allow a user to interact with the system with different input devices like keyboard, mouse, a digitizer pen, or voice recognition commands.
  - It is not necessary that all the actions are supported by all means of input devices, but some actions supports only specific mechanism only. Consider drawing a shape using keyboard will be too difficult, but mouse will be a good option.
- **User interaction should be interruptible and undoable**
  - Even for a long sequence of actions, the user must be able to interrupt the sequence and can perform some another task without losing any data or work.
  - There must be undo options available in the software.
- **The different levels of difficulty in interaction for different classes of users must be customized**
  - The users find that they perform some sequence of interactions repeatedly. Instead of writing the same sequence of interaction again and again, it is always recommended to use a macro mechanism.
  - The software should provide such a mechanism to facilitate interaction.
- **Hide complexities from the casual user**
  - The main benefit of the user interface is that it takes its user to a virtual world while interacting. The software must support this feature.
  - The design complexities like use of operating system, functions in file management, or other hidden complexities in computing technology.
  - The interface should never require a user to interact at a level which is inside the machine. For example, the user never asked to type operating system commands in the application software.
- **The direct interaction with objects that appear on the screen**
  - The user should have feel of using an object like a physical object. i.e. a design must be in such a way that user is able to perform all the function and do manipulation as he do in a physical object or physical thing.
  - For example, an application interface allows its users to stretch or resize an object as it is a physical object i.e. scaling should be a direct implementation of direct manipulation.

9.2.2 Reduce the User's Memory Load

University Question

Q. What are the design principles for reducing the user's memory load in user interface design?

SPPU : Dec. 12, 6 Marks

Some design principles are there, that enable an interface to reduce the user's memory load :

- **Reduce the demand for short-term memory**
  - When users are performing a complex task, then the demand of short-term memory might be significant.
  - The user interface must be designed in a way that reduces the requirement to remember its past actions and results. i.e. it should be able to reduce the short-term memory use.
  - This feature is achieved by providing visual clues that will help a user to remember past actions, rather than a need to recall them.
- **Establish meaningful defaults**
  - The defaults values will be an added advantage especially for average users in the start of application. But the advanced user should have a provision to set their default individual preferences.
  - There should be reset option i.e. factory reset option available in mobile phones. So that user can once again obtain the default values.
- **Define shortcuts that can easily remembered**

  The shortcuts should be defined in such a way that it can easily be remembered by the users. For example, Control P is used for print command.

  It is easy to remember this shortcut since P is the first letter of print command
- **The interface screens must be analogous to a real world object**

  A bill payment system should use a checkbook and check register metaphor to help its users to complete their transactions.
- **The information should be disclosed in a progressive fashion**
  - The interface must be designed in such a way that it gives bubble tips about a task, an object at a high level of abstraction.
  - Then the detailed information should be given when user shows interest in that i.e. when user clicks on that bubble help icon.

9.2.3 Make the Interface Consistent

The interface should present and acquire information in a consistent fashion. Some design principles that help make the interface consistent :

- **The system should allow its users to put the ongoing task into a meaningful context**
  - Most of the interfaces implement complex layers of interactions with dozens of screen images.
  - It is important to provide indicators like window titles, consistent colour codes and graphical icons that helps the user to know the context of the work at hand.
  - In addition, the user should be able to determine where he has come from and what alternatives exist for a transition to a new task.
- **The system should maintain the consistency within a family of applications**

  A set of applications (or products) should all implement the same design rules so that consistency is maintained for all interaction.

- **The new systems should not change any model that is created by the user's expectation, unless there is some genuine reasons**
  - Once a particular interactive sequence has become a standard, whether it is right or wrong, the user community expects the same in all other applications. The best example is: Control S for saving a file.
  - Any modification in use of Control S will lead to confusions. For example, if we use Control S for scaling, it leads to confusion.

9.2.4 Necessity of a Good User Interface

University Question

Q. What is the necessity of a good user interface ? **SPPU : May 12, 3 Marks**

- If the application is very difficult to operate and the user is forced to commit mistakes, then the company can lose its customers.
- The difficulty in operating the application may lead to frustration for the customers or the end users. Ultimately the company may lose business.
- Even if the application has a great computational potential and facilitate great features and functionalities, it may its users due to difficult user interface.
- These are the reasons that an application must have a good user interface.

9.3 Shneiderman's 8 Golden Rules for UI Analysis

University Question

Q. What are the rules to keep in mind while designing a user interface ? **SPPU : May 12, 3 Marks**

As we rightly said user interface plays crucial role in software development. Properly user interface is needed for all the software. To improve usability of software, it is very important to well design the user interface. This topic discusses about 8 golden rules of UI design by Ben Shneiderman.

1. Strive for consistency
2. Enable frequent users to use shortcuts
3. Offer informative feedback
4. Design dialog to yield closure
5. Offer simple error handling
6. Permit easy reversal of actions
7. Support internal locus of control
8. Reduce short-term memory load

1. **Strive for consistency**

It is very necessary to have consistent sequences of actions. If we are using similar operations then consistent sequence of operation should be there. Related terminology and help should be provided for related operation.

2. **Enable frequent users to use shortcuts**

   When we are using the same software consistently, we should enable users to use the more shortcuts to increase the efficiency. Shortcut interaction is more effective than mouse or other type of interaction. Consider in banking system, employees use the same software over the years.

   They can perform many tasks using shortcut to increase the speed of the work. Expert user can make use of this facility.

3. **Offer informative feedback**

   System need to be interactive with the users. For small operation feedback does not play important role but for lengthy operation feedback plays important role. Consider during the long installation, system should display the percentage progress to the users. Feedback can be given with the help of text message, progress bars, some live images, etc.

4. **Design dialog to yield closure**

   System should provide the sequence of action to users. The actions should be grouped together. Consider the example of photo slide show software. This software should provide group of button to navigate the photo like start, end, current, change album, start slideshow, end slideshow, zoom in, zoom out, etc. For group of actions, certain feedback should be there.

5. **Offer simple error handling**

   Error handling mechanism should be simple and easily understood by all types of users. Error messages should be displayed in simple and non technical language. First of all we need to design such a system which does not generate errors. If error happens at the rarest of rare situation then simple error handling should be there.

6. **Permit easy reversal of actions**

   For every action or for almost all action the reversal facility should be there. With this facility user can freely work system as the reversal facility is provided by the system. Simple examples are back button, recycle bin in computer, undo button, etc. If users delete something by mistake then that contents should be easily able to retrieve. If some action is happened by mistake then undo option should be there.

7. **Support internal locus of control**

   Design the system to make users the initiators of actions rather than the responders. User should easily able to locate the information without much overhead.

8. **Reduce short-term memory load**

   We should minimize the information user need to carry from one screen to another screen.

9.4 Interface Analysis and Design Models

- In fact, the process for analysis and design of a user interface starts with the development of different models of system function.
- In this the human and computer-oriented tasks are required to achieve system function. These tasks are described properly and all the design issues are also considered. There are various tools available and areused to prototype and ultimately implement the design model; and the result is evaluated by end-users for quality.

9.4.1 Interface Analysis and Design Models

- Following are four different models generated while analyzing and designing user interface :
  - **User model** is established by a human engineer or the software engineer,
  - **Design model** is created by the software engineer,
  - **User's mental model** or the system perception model created by end-user.
  - **Implementation model** is created by the implementers of the system.
- But each of these models may differ significantly in actual practice.
- This is the role of an interface designer is to patch these differences and derive a consistent representation of the interface.
- The user model establishes the profile of end-users of the system. In order to build a good user interface, the design must start with an understanding of intended users of the system and considering their profiles including age, sex, education, cultural, physical abilities, ethnic background, motivation, goals and personality etc. In addition to this, the users can be categorized as follows :
  - **Naive Users :** User having no technical and syntactical skills.
  - **Knowledgeable and intermittent users :** The users having semantic knowledge of the application but relatively weak in syntactic information.
  - **Knowledgeable and frequent users :** Users having good semantic and good syntactic knowledge.
- A design model of the complete system includes the data, architectural interface, and detailed procedural representations of the software. The requirements specification establishes different constraints that will help to define the user.

9.4.2 User Interface Design Process

University Questions

Q. Explain the user interface design process. **SPPU : Dec. 13, Dec. 14, 8 Marks**

Q. Justify " The analysis and design process for user interface is iterative". **SPPU : Feb. 16, May 16, 6 Marks**

Q. Describe the User Interface analysis and design process with diagram and explain interface design element. **SPPU : Dec. 16, 5 Marks**

- The analysis and design process for user interfaces is iterative and can be represented using a spiral model.
- Referring to Fig. 9.4.1, the user interface analysis and design process encompasses four distinct framework activities :
  1. User, task, and environment analysis and modeling.
  2. Interface design.
  3. Interface construction (implementation).
  4. Interface validation.

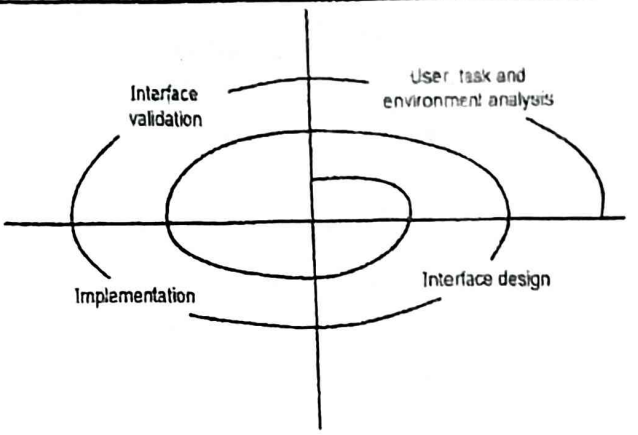


Fig. 9.4.1 : The user interface design process

- The spiral shown in Fig. 9.4.1 implies that each of these tasks will occur repeatedly along with each of the passes of spiral to represent the additional explanation of requirements and the final design.
- In many cases, the construction activity includes the prototyping that is the only practical way to validate the design.
- Interface analysis always focuses on the profile of the users. These are categorized as : skilled users, business level users etc and the class of users having general capabilities are recorded.
- Once general requirements have been defined, a more detailed task analysis is conducted. These tasks are identified and are performed by the user to reach the goals of the system.
- The analysis of the user environment focuses on the physical work environment.
- The information gathered as part of the analysis activity is used to create an analysis model for the interface. Using this model as a foundation, the design activity starts.
- The ultimate goal of interface design is to define interface objects and their actions that will help user to perform all defined tasks.

9.5 Interface Design Steps and Analysis

University Questions

Q. Explain the user interface design steps. **SPPU : Dec. 12, 8 Marks**

Q. Describe the User Interface analysis and design process with diagram and explain interface design element. **SPPU : Dec. 16, 5 Marks**

- Once the interface analysis is completed, all the tasks (or objects and actions) required by the end-user have to be identified in detail, and the interface design activity starts. Interface design, is an iterative process like all other software engineering design processes.
- Each of the user interface design step occurs number of times, each elaborating and refining information developed in the previous step.
- However so many user interface design models have been proposed, all of them suggest some combination of the following steps :
  1. Using information developed during interface analysis; define interface objects and actions (operations).
  2. Define events (user actions) that will cause the state of the user interface to change. Model this behaviour.

3. Depict each interface state, as it wil actually look to the end-user.
4. Indicate how the user interprets the state of the system from information provided through the interface.

- In some cases, the interface designer may begin with sketches of each interface state (i.e., what the user interface looks like under various circumstances) and then work backward to define important design information like objects, actions etc.
- Regardless of the sequence of design tasks, the designer must always follow the rules for user interface design.

9.5.1 Applying Interface Design Steps

- The most important step in interface design is defining the interface objects and the actions that are applied to them. To fulfil this use-cases are parsed.
- That is, a description of a use-case is written. Nouns (objects) and verbs (actions) are isolated to create a list of objects and actions.
- Once the objects and actions are defined and elaborated iteratively, they are classified by their types. Later on the target objects, source objects, and application objects are identified.
- The source object is dragged and dropped onto a target object. For an example the report icon is dragged and dropped onto a printer icon. The outcome of this action is to create a hard copy of report.
- The application objects represent the data that are not directly modified as part of screen interaction. For example, in a mailing list, the names are stored for mailing. The list can be sorted or merged but it can not be dragged and dropped through user interaction.
- When all the objects and actions are defined then the screen layout is performed. The screen layout is an interactive process in that graphical design, icons and definition of descriptive screen text, titles of windows and specification is conducted.

9.5.2 User Interface Design Patterns

- In today's era, the sophisticated graphical user interfaces are very common and a wide variety of user interface design patterns has emerged. The design pattern is an abstraction that illustrates a design solution to a specific design problem.
- The design pattern examples are presented in the side bar also has the component-level design along with the design classes, their attributes, operations and interfaces between them.

9.5.3 Interface Design Issues

University Questions

Q. What do you mean by application accessibility and internationalization in user interface design? **SPPU : Dec. 14, 4 Marks**

Q. Explain in detail the user interface design issues. **SPPU : Apr. 17, 5 Marks**

As the design of a user interface evolves, four common design issues almost always surface :

- **Response time :** Response time is one of the performance attributes of all the systems. This is the main complaint for many interactive applications. Generally, the system response time is measured from the point when user presses the enter key to the point when the software responds with the expected output or expected action.

- **Help facilities :** The users of the system require help every now and then. In short, in all of the software system, the help facility or the user manual is essential for the smooth use of software
- Some of the software provides on-line help facilities also to get the problem solved without closing the application or the interface.
- **Error handling :** The large number of error messages and warnings are actually irritating and produces unnecessary hindrance in the operation. Until and unless the error messages or warnings are very critical, it should not be popped up. The critical messages like
- "Do you want to delete all data ?" is essential to be included. Thus increased use of error messages and warnings will increase the frustration of users.
- **Menu and command labelling :** Earlier the typed command were very common but now-a-days, the common mode of interaction point and pick interfaces. It has reduced the trust on typed commands. In most of the software, in addition to typed command, menu labels are used as a common mode of interaction.
- **Application accessibility :** The computing applications are used everywhere and the developer should take care in designing the user interface and it should include easy access for all the needs of users. The physically disabled people should also be able to use the application in an easy way.
- **Internationalization :** The software developers should take into consideration the design in such a way that it is useful for end-users from all around the world. The system must support the languages spoken by all these users. The user interface must be designed for generic use.

9.5.4 Interface Design Evaluation

- After development of an operational user interface prototype, the evaluation process to assess whether the system satisfies the need of the user.
- Evaluation can be an informal "test drive," in that a user gives feedback to a formally designed study that uses statistical methods for the evaluation of questionnaires completed by a population of end-users.
- In the following Fig. 9.5.1, the interface evaluation cycle is exhibited :

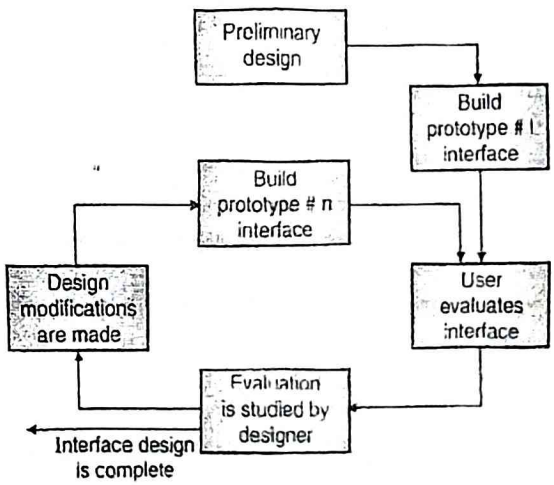


Fig. 9.5.1 : The interface design evaluation cycle

- After completion of the design model, a first-level prototype is created. This prototype is tested by the user, who provides the designer with direct feedback about the power of the interface.
- The prototyping approach is effective, but it is not possible to evaluate the quality of a user interface before a prototype is built.
- If potential problems can be detected and fixed in the early stages, the number of loops through the evaluation cycle will be minimized and development is faster.
- Once a design model of the interface is ready, the evaluation criteria can be applied.
- After development of the first prototype, the developer collects a variety of qualitative and quantitative data that will help in assessing the interface.
- Questionnaires should be distributed to users to collect qualitative data related to prototype.

9.6 Design Evaluation

- In design evaluation, a prototyping model is created and it is handed over to user to determine whether it is developed as per the requirement given by the client.
- The end user can only evaluate the product and check that it meets the requirement.
- After receiving the feedback from the user, it is studied well by the developer and it the feedback is valid then the modifications are done by the developer.
- This user interface evaluation is illustrated by the Fig. 9.6.1 :

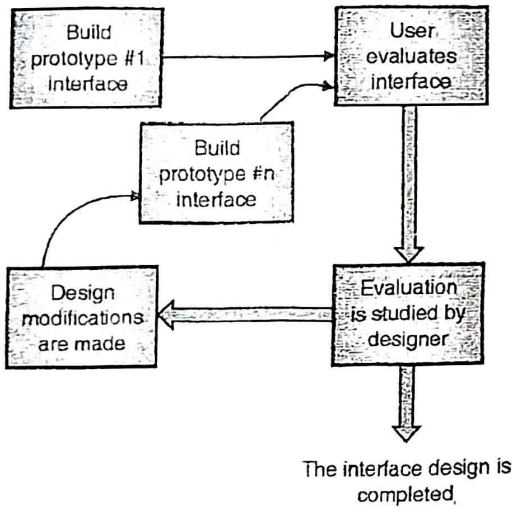


Fig. 9.6.1 : The interface design evaluation cycle

- In addition to user feedback, the developer can also use other evaluation techniques as listed follows :
  - Questionnaires
  - Rating sheets
- The developer can use all these useful information for prototype refinement. Initially prototype no. 1 is created and it is evaluated by the user. After modifications are effected, the prototyping model no. 2 becomes ready, and it is further evaluated by the user and this process continues until all the requirements are satisfied.

9.7 WebApp Interface Design

For every interface design, either for traditional software or for WebApp, should have the usability characteristics like response time, help facilities, error handling, menu and command labelling, application accessibility, etc.

9.7.1 WebApp Design Principles

University Questions

Q. Highlight the principles specific to the design of a Web Application Interface. SPPU : May 12, 5 Marks

Q. Enlist and explain the Webapp design principles in detail. SPPU : Apr. 17, 5 Marks

Following are some WebApp design principles :

- Anticipation means it should let the user know its next step or next move.
- Communication let the user know the status of any activity initiated by him through some text message or through some image.
- **Consistency** must be maintained throughout the WebApp. For example navigation controls, icons or menus must be consistent. With respect to its colour, shape throughout the design.
- **Efficiency** : The design of the interface must optimize the end user's work efficiency.
- **Flexibility** is the integral part of the interface design. The user may explore the application in some random fashion. The system must be flexible enough to accommodate the needs of the user.
- Focus the WebApp interface should stay focus on the user tasks at hand.
- **Readability** : Information presented should be readable to all the users.
- **Learnability** : The application must be designed to minimize learning time.
- Main the work integrity through the system.
- Latency reduction should minimize the waiting time for user and use that waiting for some internal operation to complete.

Review Question

Q. Write a short notes on : User Interface Design

Q. What are the types of User Interface? Explain each with example

Q. State and explain characteristics of good user interface. What are the benefits of good user interface design?

Q. What are the interface design principles and guidelines?

Q. State and explain any four principles.

Q. What are the design principles for reducing the user's memory load in user interface design?

Q. What is the necessity of a good user interface?

Q. What are the rules to keep in mind while designing a user interface?

Q. Explain the user interface design process.

Q. Justify "The analysis and design proces for user interface is iterative".

Q. Describe the User Interface analysis and design process with diagram and explain interface design element.

Q. Explain the user interface design steps.

Q. Describe the User Interface analysis and design process with diagram and explain interface design element.

Q. What do you mean by application accessibility and internationalization in user interface design?

Q. Explain in detail the user interface design issues.

Q. Highlight the principles specific to the design of a Web Application Interface.

Q. Enlist and explain the Webapp design principles in detail.

□□□

10

Unit - IV

PROJECT PLANNING

Syllabus

Project initiation, Planning Scope Management, Creating the Work Breakdown Structure, scheduling : Importance of Project Schedules, Developing the Schedule using Gantt Charts, PERT/ CPM

10.1 Introduction to Project Planning

- The project planning is an important activity performed by the project managers in order to estimate the following entities for their optimum use :
  - The resources required for the project under development.
  - The cost of the project
  - The schedule for in-time completion of the project
- All these estimates are made in the stipulated amount of time in the starting of the project itself. Later on these estimates can be updated time to time. In order to achieve the objectives of project planning, following activities are carried out :
  - Defining software scope
  - Estimating resources

10.2 Project Initiation

- The Project Initiation Phase is the first phase in the Project Management Life Cycle, as it involves starting up a new project. We start a new project by defining its objectives, scope, purpose and deliverables to be produced.
- We also hire our project team, setup the Project Office and review the project, to gain approval to begin the next phase.
- Overall, there are six key steps that we need to take to properly initiate a new project. These Project Initiation steps are listed below :
  - Develop a Business Case
  - Undertake a Feasibility Study
  - Establish the Project Charter
  - Appoint the Project Team
  - Set up the Project Office
  - Perform a Phase Review

10.2.1 Business Case

- A Business Case justifies the start-up of a project. It includes a description of the business problem or opportunity, the costs and benefits of each alternative solution, and the recommended solution for approval.
- The Business Case is referred to frequently during the project, to determine whether it is currently on track. And at the end of the project, success is measured against the ability to meet the objectives defined in the Business Case.

10.2.2 Feasibility Study

- A Project Feasibility Study is an exercise that involves documenting each of the potential solutions to a particular business problem or opportunity. Feasibility Studies can be undertaken by any type of business, project or team and they are a critical part of the Project Life Cycle.
- The purpose of a Feasibility Study is to identify the likelihood of one or more solutions meeting the stated business requirements.
- In other words, if you are unsure whether your solution will deliver the outcome you want, then a Project Feasibility Study will help gain that clarity. During the Feasibility Study, a variety of 'assessment' methods are undertaken. The outcome of the Feasibility Study is a confirmed solution for implementation.

10.2.3 Project Charter

- A Project Charter outlines the purpose of the project, the way the project will be structured and how it will be successfully implemented. The Project Charter describes the project vision, objectives, scope and deliverables, as well as the Stakeholders, roles and responsibilities.
- The Project Charter defines the vision and boundaries for the project, as well as the high level roadmap. In addition, the Project Charter also defines the scope of the project, within which the deliverables are produced.

10.2.4 Project Team

- A Project Job Description defines the objectives and responsibilities of a particular role on a project.
- A Project Job Description should be completed every time a new role is identified. The Project Job Description should clearly state the objectives and responsibilities of the role and where it fits within the organizational structure.

10.2.5 Project Office

The Project Office Checklist lists everything you need to do, to set up a Project Management Office. A Project Management Office is the physical premises within which project staff (e.g. the Project Manager and support staff) reside.

The Project Office also contains the communications infrastructure and technologies required to support the project.

0.2.6 Phase Review

A Project Review is an assessment of the status of a project, at a particular point in time. The first time in the project life cycle that a project review is undertaken is at the end of the first project phase, called "Initiation".

- During this project review, a decision is made as to whether or not the team has met the objectives and is approved to proceed to the next project phase, being the "Planning" phase.

10.3 Planning Scope Management

- A software project manager is normally confused with a condition in the beginning of a software engineering project i.e. quantitative estimates and an organized plan. They are required and necessary but proper information is not available.
- A detailed analysis of software requirements may provide all the necessary information for estimates, but analysis often takes too much time to complete, even weeks or months. Requirements may be changing regularly as the project proceeds.
- Therefore, the developer and the manager should examine the product and the problem. It is intended to solve at the beginning of the project itself. At a minimum, the scope of the product must be established and bounded.
- The first software project management endeavor is the finding out of software scope. Scope can be defined by answering the following simple questions:
  - **Context :** How the software system can be developed to fit into a large system or some business context and what are the constraints that should be considered as a result of the context considered ?
  - **Information objectives :** What data objects are produced as output through the software developed from customer's point of view ? And what data objects will be used for the input ?
  - **Function and performance :** What are the functions that the software should perform in order to translate input into output ? Is there any special performance characteristic that is supposed to be addressed ?
- There should be a clear and comprehensive software project scope defined and that should be understandable at all the levels (i.e. the management and technical levels).
- Generally the software scope describes the following :
  - Performance
  - Function
  - The data and control used to define the constraints
  - Reliability
  - Interfaces.
- Functions are described in the statement of scope are assessed and evaluated to provide details in estimation. The cost and schedule are two important parameters and they are functionality oriented.

10.3.1 Obtaining Information Necessary for Scope

- In start of a development process, it is always good and necessary to define the scope of the project. The frequent communication between the customer and developer builds the foundation for defining scope properly. The frequent meetings between the customer and developer are very important to bridge the gap in all sorts of communication.
- Following are the set of questions that focuses on the goals and objectives :
  - Know the person behind the request for this work.
  - See who will use the solution?

- What will be the commercial benefit of a solution?
- Is there any other resource of solution?

- These questions clear the understanding of problems and customers can raise their doubts.

10.3.2 Feasibility

- After defining and identifying the scope of the project, the developer should look at the scope and should estimate the feasibility whether the project is feasible or not.
- Otherwise, after spending lot of time working on the project, it is found that the scope is not feasible to develop. Hence it is always necessary to check the feasibility in the starting of the development process itself to avoid the losses.

10.3.3 A Scoping Example

- The frequent communication between the customer and developer is always useful in defining the functions, data and control, performance and the constraints and all the related information.
- The person planning the project looks into the details of statement of the project and derives all the necessary information. This particular process is called as decomposition and it has the following functions :
  - Read the input barcode
  - Read the tachometer reading
  - Database look-up should be done
  - Produce the control signal
  - Maintain all the record
- The planner (the person who plans) interacts with all the elements of computer-based system and he considers all the complexities in each of the interfaces in order to find any effect on the cost, resources and schedule.

10.4 Creating the Work Breakdown Structure

- Scheduling of different engineering activities can use the available project scheduling tools and techniques.
- Following are some project scheduling tools and techniques :
  - PERT (Program evaluation and review technique)
  - CPM (Critical path method)
- These are two project scheduling methods that can be applied to the software development process.
- Both tools use the data and information received from the earlier developments. They perform the same planning activities as performed in earlier projects. These activities are listed as follow :
  - Estimating the effort
  - Decomposing the product function
  - Selecting suitable process model
  - Decomposing task.
- The interdependencies among various task are also called as **work breakdown structure** (WBS). The WBS can be defined for single functions and for the whole project as well.
- Both tools allow to determine the critical path, duration of the projects, and time estimate for the individual activity, and helps in calculating the "boundary times".

- The boundary time calculations are very important in defining the software project schedules.

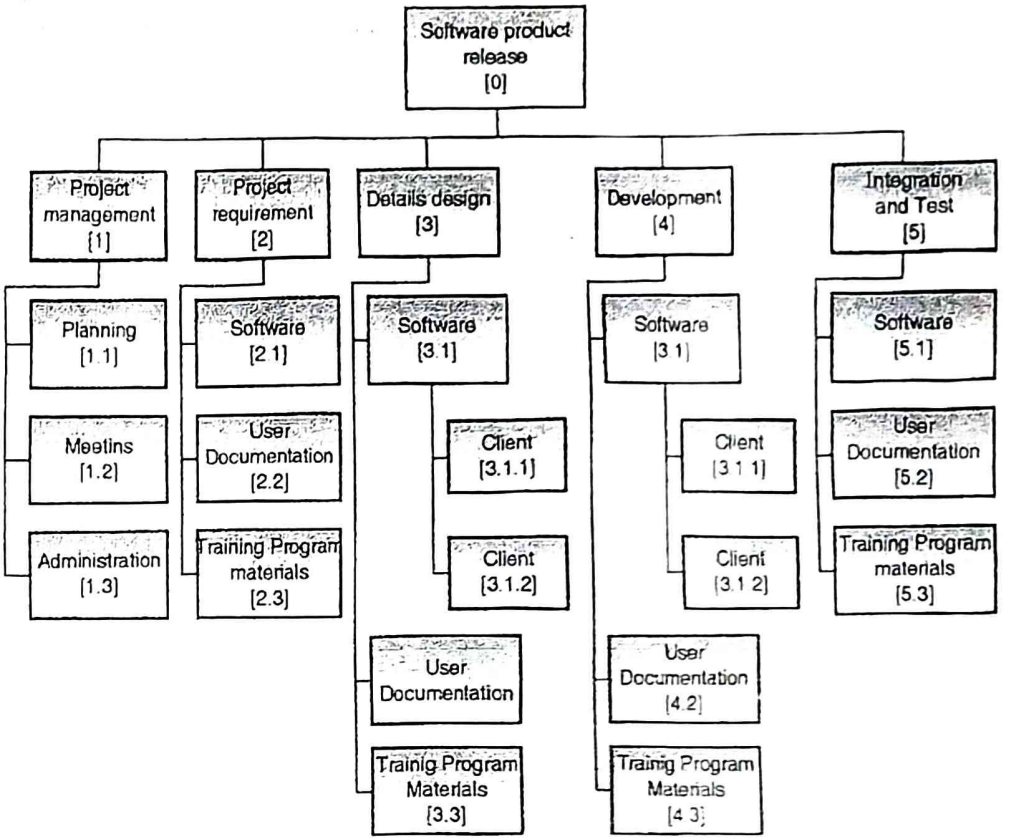


Fig. 10.4.1 : A sample Work Breakdown Structure (WBS)

10.5 Project Scheduling

10.5.1 The Structure of Estimation Models

- The estimation model is generally derived from regression analysis on the data that are collected from previously collected software projects. The structure of estimation models take the following form :

$$E = A + B \times (e_v)^C \quad \ldots(1)$$

Where A, B, and C are nothing but the empirically derived constants. The constant E is an effort in person-months, and the estimation variable is e_v.

- In addition to the relationship described by Equation (1), most of estimation models use project adjustment component that enables the effort E. Generally the effort E is adjusted by the following project characteristics :
  - Problem complexity
  - Staff experience
  - Development environment etc.
- Various LOC-oriented estimation models that have been proposed in the literature are as follows :

1) $E = 5.288 \times (KLOC)^{1.047}$

(Doty model for KLOC > 9)

2) $E = 3.2 \times (KLOC)^{1.05}$

(Boehm simple model)

3) $E = 5.2 \times (KLOC)^{0.91}$

(Walston-Felix model)

4) $E = 5.5 + 0.73 \times (KLOC)^{1.16}$

(Bailey-Basili model)

- Following are some FP-oriented models proposed :

1) $E = -37 + 0.96\ FP$ (Kemerer model)

2) $E = -91.4 + 0.355\ FP$

(Albrecht and Gaffney model)

3) $E = -12.88 + 0.405\ FP$

(small project regression model)

- After observing all these models, we come to conclusion that different results are possible for the same values of LOC or FP

10.5.2 The COCOMO II Model

University Question

Q. Explain the COCOMO-II estimation model. **SPPU : May 14, 5 Marks**

- **Barry Boehm** has devised a software estimation models hierarchy having the name as COCOMO i.e. **CO**nstructive **CO**st **MO**del. The COCOMO model has been evolved into more comprehensive model called COCOMO II. It is actually having a hierarchy of estimation models that focus the following areas :
  - **Application composition model :** It is used in the early stages of development, when user interfaces, system interaction, performance assessment and technology evaluation are prime important.
  - **Early design stage model :** Once the requirements are stabilized and basic architecture is constructed, then early design stage model is used.
  - **Post-architecture stage model :** It is used during development of the software.
- COCOMO II models also require sizing information like other estimation models for the software. Following three sizing options are available :
  - Object points
  - Function points and
  - Lines of source code
- The COCOMO II model uses object points that are an indirect software measure. They are is computed using the counts of
  - Screenshots taken at user interface
  - Reports and
  - Components required in developing the application.
- The screenshots or the reports are classified into any of the following three complexity levels :
  - Simple
  - Medium
  - Difficult

- The client and server data tables are required to create the screenshots and reports. The complexity is defined as the function of these reports.

Table 10.5.1 : Complexity weighting for object types

Object type	Complexity weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component			10

- After the determination of complexity, the number of screenshots, reports, and components object point are weighted as per the table shown in Table 6.8.1 In component-based development when software reuse is implemented, then the percent of reuse (% reuse) is estimated and the object point count is adjusted according to the following equation :

NOP = (object points) × [(100 – % reuse)/100] where NOP → New Object Points.

Table 10.5.2 : Productivity rate for object points

Developer's experience/capability	Very low	Low	Nominal	High	Very high
Environment maturity / capability	Very low	Low	Nominal	High	Very high
PROD	4	7	13	25	50

- In order to derive the estimate of effort, a "productivity rate" should be derived first. They are based on the computed NOP value. The Table 6.8.2 shown above presents the productivity rate. These productivity rates are illustrated for various levels of developer's experience as well as various levels of environment maturity.

PROD = NOP/person-month

- After determining the productivity rate, an estimate of project effort can be derived as follows :

Estimated effort = NOP/PROD

10.5.3 The Software Equation

- The software equation is a model that considers a specific distribution of effort over the project development life. The software equation model is a multivariable model. This model is derived from productivity data collected from nearly 4000 contemporary software projects. Depending on all these data, the estimation model takes the following form :

$$E = [LOC \times B^{0.333}/P]^3 \times (1/t^4) \quad \ldots(2)$$

Where,

E = Person-months effort or person-years efforts

t = Duration of the project in months or years

B = The special skills factor

P = Productivity parameter

- The productivity parameter reflec s the following characteristics :
  - Process maturity
  - Management practices
  - Good software engineering practices
  - Programming languages used
  - Software environment
  - The skill set of team members
  - The experience of team members, and
  - The complexity of the software
- Typical values might be P = 2000 for development of real-time embedded software,
- P = 10,000 for telecommunication and systems software; P = 28,000 for business systems applications. The productivity parameter is derived using historical data collected from previous development efforts.
- The software equation has following two independent parameters :
  - An estimate of size and
  - An indication of project duration.

10.6 Importance of Project Schedules

- To organize and complete your projects in a timely, quality and financially responsible manner, you need to schedule projects carefully.
- Effective project scheduling plays a crucial role in ensuring project success. To keep projects on track, set realistic time frames, assign resources appropriately and manage quality to decrease product errors.
- This typically results in reduced costs and increased customer satisfaction. Important factors include financial, documentation, management and quality assurance.

Financial

- Project scheduling impacts the overall finances of a project. Time constraints require project managers to schedule resources effectively. This is particularly true when resources must have highly specialized skills and knowledge in order to complete a task or when costly materials are required.
- Completing a project in a short time frame typically costs more because additional resources or expedited materials are needed. With accurate project scheduling, realistic estimates and accurate projections prevent last-minute orders that drive up costs.

Documentation

- Creating a comprehensive work breakdown structure allows you to create a chart, such as a Gantt chart, that lists the project tasks, shows dependencies and defines milestones. Management consultant Henry Gantt designed this type of chart to show a graphic schedule of planned work. Its role in business projects is to record and report progress toward project completion.
- Your project schedule also allows you to assign human resources to the work and evaluate their allocation to ensure you have the appropriate levels of utilization. You may also develop a program evaluation and review technique chart, or PERT chart, to help you analyze project tasks.

Management

- Effective project managers conduct regular meetings to get status reports. They use project scheduling meetings to check in with their team members and prevent costly misunderstandings.
- These regular meetings ensure that work flows from one process to the next and that each team member knows that he needs to do to contribute the project's overall success.

Quality

- Project scheduling ensures one task gets completed in a quality manner before the next task in the process begins. By assuring that quality measures meet expectations at every step of the way, you ensure that managers and team members address problems as they arise and don't wait until the end.
- No major issues should appear upon completion because you've established quality controls from the very beginning of the scheduling process.
- Effective project managers understand that ensuring quality control involves managing risks and exploiting opportunities to speed up the schedule when possible to beat the competition and achieve or maintain a competitive edge with a more reliable product.

10.7 Developing the Schedule using Gantt Charts

- A time-line chart, also called Gantt chart, is generated on the basis of the start date inputs for each task.
- Fig. 10.7.1 shows an example Gantt chart. It elaborates the project schedule.
- In the left hand column of the Fig. 10.7.1, all the project tasks are listed. The horizontal lines indicate the duration of the task.
- For concurrent activities, multiple horizontal lines are used.

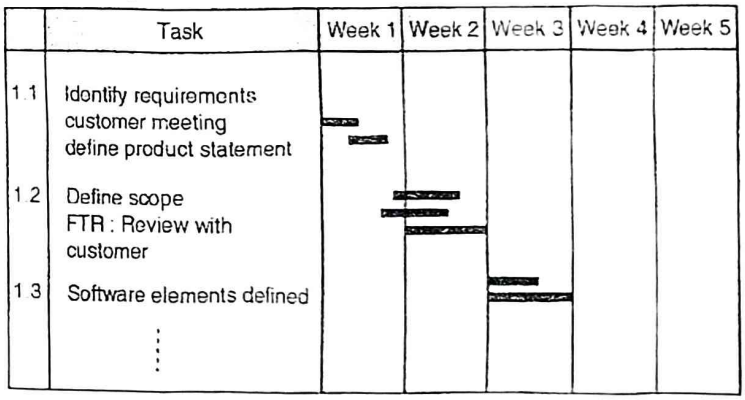


Fig. 10.7.1 : An example time-line chart

10.7.1 Tracking the Schedule

- The project schedule is nothing but a road map that defines the tasks and the milestones to be tracked.
- Tracking is achieved in different ways as follows :
  - Conduct periodic meeting to find out the progress status and problems.
  - Evaluate the process.
  - Check whether all the activities are completed within the deadline or by the schedule date.

- Compare the planned date and actual start date for each activity.
- Meet practitioners informally to assess the progress.

- All these tracking techniques are used by the project managers.

10.7.2 Schedule and Cost Slippage

- A schedule slippage in project planning is described as missing a deadline and project is not completed in the estimated time. In order to avoid schedule slippage, the project manager must plan the project very carefully so that there should not delays in the schedule.
- The time-line charts or Gantt charts are very useful tools to plan the project in an effective manner.
- If there is slippage in schedule, then ultimately there is slippage in the cost of the project or the budget of the project.
- If the schedule slippage is detected in some projects, then project starts floating towards the upper limits of project deadline. The float in a schedule means delays and leads time mentioned in the project schedule. It is the responsibility of a project manager to diagnose the problems and find the root causes of the slippage and take necessary actions to correct the problems.
- Practically it is very difficult to achieve the deadline estimated. Following are some important tasks to be accomplished in order to avoid schedule slippage and cost slippage :
  - Identify the basic reasons behind slippage.
  - Forecast a probable delivery date.
  - Acknowledge the problems of slippage and find suitable remedy for each of the problems.
  - Try to use some method to neutralize the slippage.
  - Always discuss customers with the current progress of the project.
- **Causes of schedule and cost slippage :**

  Following are three major causes of schedule and cost slippage :
  - Poor estimation of schedule and cost.
  - Poor requirement analysis and requirement gathering, and
  - Management issues

10.8 Project Scheduling Tools and Techniques : PERT/ CPM

- Scheduling of different engineering activities can use the available project scheduling tools and techniques.
- Following are some project scheduling tools and techniques :
  - CPM (Critical Path Method)
  - PERT (Program Evaluation and Review Technique)
- These are two project scheduling methods that can be applied to the software development process.
- Both tools use the data and information received from the earlier developments.
- Both tools allow to determine the critical path, duration of the projects, and time estimate for the individual activity efforts taken, duration.

10.8.1 CPM (Critical Path Method)

University Question

Q. What is critical path? Explain with example how critical path method is used to estimate the total project duration.

SPPU : May 19, 5 Marks

- The critical path method (CPM) is a project modeling technique developed in the late 1950s. CPM is commonly used with all forms of projects, including construction, aerospace and defence, software development, research projects, product development, engineering, and plant maintenance, among others.
- Any project with interdependent activities can apply this method of mathematical analysis. The first time CPM was used for major skyscraper development was in 1966 while constructing the former World Trade Centre Twin Towers in NYC.
- Although the original CPM program and approach is no longer used, the term is generally applied to any approach used to analyze a project network logic diagram.
- The essential technique for using CPM is to construct a model of the project that includes the following :
  1. A list of all activities required to complete the project
  2. The time (duration) that each activity will take to complete.
  3. The dependencies between the activities and,
  4. Logical end points such as milestones or deliverable items.
- Using these values, CPM calculates the longest path of planned activities to logical end points or to the end of the project, and the earliest and latest that each activity can start and finish without making the project longer.
- This process determines which activities are "critical" (i.e., on the longest path) and which have "total float" (i.e., can be delayed without making the project longer).
- In project management, a critical path is the sequence of project network activities which add up to the longest overall duration, regardless if that longest duration has float or not. This determines the shortest time possible to complete the project.
- A project can have several, parallel, near critical paths; and some or all of the tasks could have 'free float' and/or 'total float'. An additional parallel path through the network with the total durations shorter than the critical path is called a sub-critical or non-critical path. Activities on sub-critical paths have no drag, as they are not extending the project's duration.
- CPM analysis tools allow a user to select a logical end point in a project and quickly identify its longest series of dependent activities (its longest path). These tools can display the critical path (and near critical path activities if desired) as a cascading waterfall that flows from the project's start (or current status date) to the selected logical end point.

10.8.2 PERT (Program Evaluation and Review Technique)

- The program (or project) evaluation and review technique, commonly abbreviated PERT, is a statistical tool, used in project management, which was designed to analyze and represent the tasks involved in completing a given project.
- It is commonly used in conjunction with the critical path method (CPM).

- PERT is a method of analyzing the tasks involved in completing a given project, especially the time needed to complete each task, and to identify the minimum time needed to complete the total project.
- PERT was developed primarily to simplify the planning and scheduling of large and complex projects. It was developed for the U.S. Navy Special Projects Office in 1957 to support the U.S. Navy's Polaris nuclear submarine project.
- It was able to incorporate uncertainty by making it possible to schedule a project while not knowing precisely the details and durations of all the activities.
- It is more of an event-oriented technique rather than start- and completion-oriented, and is used more in projects where time is the major factor rather than cost.
- It is applied to very large-scale, one-time, complex, non-routine infrastructure and Research and Development projects.
- The first step to scheduling the project is to determine the tasks that the project requires and the order in which they must be completed. The order may be easy to record for some tasks (e.g. When building a house, the land must be graded before the foundation can be laid) while difficult for others (there are two areas that need to be graded, but there are only enough bulldozers to do one).
- Additionally, the time estimates usually reflect the normal, non-rushed time. Many times, the time required to execute the task can be reduced for an additional cost or a reduction in the quality.
- PERT planning involves the following steps :

1. Identify the specific activities and milestones.
2. Determine the proper sequence of the activities.
3. Construct a network diagram.
4. Estimate the time required for each activity.
5. Determine the critical path.
6. Update the PERT chart as the project progresses

1. **Identify the specific activities and milestones**
   - The activities are the tasks required to complete a project. The milestones are the events marking the beginning and the end of one or more activities.
   - It is helpful to list the tasks in a table that in later steps can be expanded to include information on sequence and duration.
2. **Determine the proper sequence of the activities**
   - This step may be combined with the activity identification step since the activity sequence is evident for some tasks.
   - Other tasks may require more analysis to determine the exact order in which they must be performed.
3. **Construct a network diagram**
   - Using the activity sequence information, a network diagram can be drawn showing the sequence of the serial and parallel activities.
   - Each activity represents a node in the network, and the arrows represent the relation between activities.

- Software packages simplify this step by automatically converting tabular activity information into a network diagram.

4. Estimate the time required for each activity

- Weeks are a commonly used unit of time for activity completion, but any consistent unit of time can be used.
- A distinguishing feature of PERT is its ability to deal with uncertainty in activity completion time. For each activity, the model usually includes three time estimates.
  (i) **Optimistic time** – generally the shortest time in which the activity can be completed. It is common practice to specify optimistic time to be three standards deviations from the mean so that there is a approximately a 1% chance that the activity will be completed within the optimistic time.
  (ii) **Most likely time** – the completion time having the highest probability. Note that this time is different from the expected time.
  (iii) **Pessimistic time** – the longest time that an activity might require. Three standard deviations from the mean is commonly used for the pessimistic time.
- PERT assumes a beta probability distribution for the time estimates.
- For a beta distribution, the expected time for each activity can be approximated using the following weighted average :

  Expected time = (Optimistic + 4 x Most likely + Pessimistic) / 6
- This expected time may be displayed on the network diagram.
- To calculate the variance for each activity completion time, if three standard deviation times were selected for the optimistic and pessimistic times, then there are six standard deviations between them, so the variance is given by:

  [(Pessimistic - Optimistic) / 6]

5. Determine the critical path

- The critical path is determined by adding the times for the activities in each sequence and determining the longest path in the project. The critical path determines the total calendar time required for the project.
- If activities outside the critical path speed up to slow down (within limits), the total project time does not change. The amount of time that a non – critical path activity can be delayed without the project is referred to as a slack time.
- If the critical path is not immediately obvious, it may be helpful to determine the following four quantities foe each activity :

  ES - Earliest Start time

  EF - Earliest Finish time

  LS - Latest Start time

  LF - Latest Finish time
- These times are calculated using the expected time for the relevant activities. The earliest start and finish times of each activity are determined by working forward through the network and determining the earliest time at which an activity can start and finish considering its predecessors activities.

- The latest start and finish times are the latest times that an activity can start and finish without delaying the project. LS and LF are found by working backward through the network. The difference in the latest and earliest finish of each activity is that activity's slack. The critical path then is the path through the network in which none of the activities have slack.
- Since the critical path determines the completion date of the project, the project can be accelerated by adding the resources required to decrease the time for the activities in the critical path. Such a shortening of the project sometimes is referred to as **project crashing**.

6. **Update the PERT chart as the project progresses**
   - Make adjustments in the PERT chart as the project progresses. As the project unfolds, the estimated times can be replaced with actual times.
   - In cases where there are delays, additional resources may be needed to stay on schedule and the PERT chart may be modified to reflect the new situation.

10.8.2(A) Advantages using PERT

PERT is useful because it provides the following information :

- Expected project completion time;
- Probability of completion before a specified date;
- The critical path activities that directly impact the completion time;
- The activities that have slack time and that can be lend resources to critical path activities;
- Activity start and end date.

Review Questions

Q. Why Project Planning is important?

Q. List the activities in the software project planning.

Q. What is the objective of project planning?

Q. Explain Project Initiation in details.

Q. Explain Planning Scope.

Q. Write a short notes on Planning Scope Management.

Q. Explain Work Breakdown Structure.

Q. Explain Project Scheduling.

Q. What is the Importance of Project Schedules?

Q. How Schedules are developed using Gantt Charts?

Q. What are different Project Scheduling Tools and Techniques?

UUU

11 PROJECT MANAGEMENT

Unit - IV

Syllabus

The Management Spectrum, People, Product, Process, Project, The W5HH Principle, Metrics in the Process and Project Domains, Software Measurement: size &function-oriented metrics(FP & LOC), Metrics for Project

11.1 The Management Spectrum

University Questions

Q. Explain the role of people, product and process in project management. **SPPU : May 12, May 13, May 15, 8 Marks**

Q. Explain four major sections of project management plan **SPPU : Dec. 19, 4 Marks**

- Management spectrum for effective software focuses on the four P's :

1. People	2. Product
3. Process	4. Project

- How these four P's can be used in the management spectrum? It is interesting to note that the order can never be arbitrary.

11.1.1 The People

University Question

Q. Explain the term the people of management spectrum. **SPPU : May 12, May 19, 4 Marks**

- Software engineering institute has developed model named the People Management Capability Maturity Model (PM-CMM). The PM-CMM has been developed to deploy the talent needed to improve their software development capability.
- Not only to deploy talent but also to enhance the readiness of software organizations to undertake increasingly complex applications by helping to grow, attract, motivate the talent needed to improve their software development capability and to continue to hold the talent to improve the development capabilities.
- Also helping to retain the talents that are needed to improve the development capabilities.
- The PM-CMM defines the following key practice areas for software people :
  - Recruiting
  - Selection
  - Performance management
  - Training
  - Compensation

- Career development
- Organization and work design
- Team or culture development

- The PM-CMM is very close to the software capability maturity model integration. It guides organization in creation of a mature software process.
- There are various groups that are involved for the most needed communication and co-ordination issues required for the effective software. All these groups can be categorized as under :

1. Stake holders
2. Team leaders
3. Software team
4. Agile teams
5. Co-ordination and communication issues

11.1.1(A) Stake Holders

University Questions

Q. Explain the term in brief: Stakeholders. **SPPU : May 13, 2 Marks**

Q. What are the categories of stakeholders? What are the characteristics of effective project manager? **SPPU : May 14, 8 Marks**

Stake holders are the people who are directly or indirectly involved in the software process and software project. To make the effective development process, the project team must be organized in such a way that maximizes each person's skills and capabilities. This is the job of the team leader that looks into each and every corner of the process. The stake holders can be categorized into one of five categories required :

1. Senior managers
2. Project managers
3. Developers
4. Customers
5. End users

1. **Senior managers** have lot of influence on the project and they are the people who define business issues.
2. **Project managers** can organize, plan motivate and control the developers who develop the software project. Actually, all these are the technical people.
3. **Developers** have the technical skills that are important to engineer a product or application.
4. **Customers** are one of the important stakeholders who specify the requirements. These requirements are later developed. Also some other stakeholders are also there who have interest in the outcome of the software.
5. Finally the **end-users** who are going to use the software.

11.1.1(B) Team Leaders

- Competent practitioners often make poor team leaders because project management is a people-intensive activity. But unfortunately in many cases individuals just fall into a project manager role and become the project managers accidentally.
- The **MOI** (Model of Leadership) is used by the project managers that has the following characteristics to make a project manager very effective:
  - **Motivation** : The ability to encourage the team to produce their best ability.

- **Organization :** The ability to organize the ongoing processes in such a way that will help the initial concept to mould into a final product.
- **Ideas or innovation :** The ability to encourage the team to create the innovative ideas let them feel to be more creative in their tasks.

- All the successful project leaders apply a **problem solving management style** i.e. a software project manager should be able to concentrate on understanding the problem and finding the solution, managing the flow of ideas and thoughts. At the same time, manger should let everyone in the team to know that quality is important parameter and can not be compromised.
- One other view of the characteristics that defines an effective project manager should emphasizes following four key factors :
  - **Problem solving :** A project manager should find out technical and organizational issues.
  - **Managerial identity :** A project manager should take the charge of the project. He should have overall control and should allow only good technical team members to do their own styles.
  - **Achievement :** To enhance the productivity of a project team, he should take initiative and demonstrate the function through his own actions.
  - **Influence and team building :** A project manager should have the ability to read the faces and capability of his team members i.e. face reading.

11.1.1(C) Software Team

- The people those are directly involved in a software project are within the project manager's scope. The structure of a good team depends on the management methodology adopted in the organization, the number of people who will inhabit the team and their skill levels, and the overall complexity of the question.
- Following are seven project factors that must be considered when planning the structure of software engineering teams :
  - The ability to solve the difficulty of the problem.
  - The overall size of the final programs in lines of code.
  - The amount of time that the team will stay together.
  - The degree of modularity.
  - The quality and reliability of the system.
  - The delivery date rigidity.
  - The degree of communication required for the project.
- To achieve a high-performance team :
  - Team members must have trust in one another.
  - The distribution of skills must be appropriate to the problem.
  - An unorthodox or independent-minded person may have to be excluded from the team, if team unity is to be maintained.
- The aim and idea for every project manager is to create a team that exhibits unity among the team members.

11.1.1(D) Agile Teams

- The term agile means very active. As a cure to many of the problems that have afflicted software project work, in recent years, agile software development has been put forward. The agile philosophy provides the following :
  - It encourages customer satisfaction by early incremental delivery of software.
  - It provides small highly motivated project teams.
  - It consists of informal methods.
  - It also provides minimal software engineering work products.
  - It results in overall development simplicity.
- The small sized and highly motivated project team, also called an agile team, adopts many of the characteristics of successful software project teams.

11.1.1(E) Co-ordination and Communication Issues

There are some factors because of which, software projects get into problems. These factors are based on the following features of modern software :

1. **Scale**

   Since the scale of most of the development efforts is large, therefore it leads to confusion, complexity and significant difficulties in coordinating team members.

2. **Uncertainty**

   Uncertainty is very common and it results in a continuing stream of changes that will increase or decrease the size of project team.

3. **Interoperability**

   - Interoperability is one of the important characteristics of many software applications. The new software should be able to communicate with the existing system and satisfy the predefined constraints and conditions implemented by the software.

- The following important characteristics of software are the facts of life :
  1. Scale 2. Uncertainty
  3. Interoperability
- To handle these characteristics very effectively, the development team must devise some method for communicating and coordinating the team members.
- The formal communication is achieved through formal meetings like writing and non-interactive communication channels. The informal communication is more personal focusing individuals. All the members of a software team must share their ideas on ad hoc basis and must ask for any help required for the problems encountered on a daily basis.

11.1.2 The Product

University Question

Q. Explain the term the product of management spectrum. **SPPU : May 12, May 19, 4 Marks**

- The product objectives and scope must be established before a planning a project. All the technical and management constraints and difficulties should be identified and their alternative solutions should be considered.
- Without this information, it is impossible to define reasonable and accurate estimation of the cost, and conduct an effective assessment of risk and a realistic breakdown of project tasks.
- In order to define the product objectives and its scope, there should proper coordination between the software developer and customer and they must meet.
- In most of the cases, this particular activity begins as part of the system engineering or business process engineering and continues as the first step in software requirements engineering.
- Objectives identify the overall goals for the product. These objectives do not take into consideration, how these goals will be achieved. The scope identifies the primary data and functions and the behaviours that characterize the product, and more importantly, attempts to bound these characteristics in a quantitative manner.

11.1.3 The Process

University Questions

Q. Explain the term the process in project management. **SPPU : May 19, 4 Marks**

- A comprehensive plan can be drafted using software process framework in software development process.
- All these framework activities can be applied to all projects either small sized or large sized and regardless of the complexity of software projects
- Following factors enable the framework activities to be adapted to the characteristics of the software project :
  - The tasks set
  - Work products
  - Milestones
  - Quality assurance points
- And finally, the umbrella activities mentioned below overlay the process model :
  - SQA (Software Quality Assurance)
  - SCM (Software Configuration Management)
  - Measurement
- Generally all these umbrella activities are independent of any one framework activity and they are applied throughout the development process.

11.1.4 The Project

University Question

Q. Explain the term the project in project management. **SPPU : May 19, 4 Marks**

- Planned and controlled software projects are conducted for one and only reason because it is the only way known to manage complexity.
- Project failure rate remains higher than it should be even though the success rate for software projects has improved. One of the reasons is that the customers lose interest quickly (because what they have requested was not really as important as they first thought), and the projects are cancelled.

- The software engineers and the software project manager must follow certain guidelines in order to avoid project failure :
  - Heed a set of common warning signs
  - Understand the critical success factors that lead to good project management and
  - Develop a common sense approach for planning, monitoring, and controlling the project.

11.2 The W5HH Principle

We need an organizing rule that balances down to stipulate simple project plans for simple projects. An approach that addresses project objectives, milestones and schedules, responsibilities, management and technical approaches, and required resources is called as the W5HH principle. W5 means five 'Wh' questions. And next HH means two 'How' questions. After a series of questions that lead to a definition of key project characteristics and the resultant project plan :

Why is the system being developed ?

The answer to this question enables all parties to assess the validity of business reasons for the software work. Stated in another way, does the business purpose justify the expenditure of people, time, and money ?

What will be done ?

The answer to this question establishes the task set that will be required for the project.

When will it be done ?

The answer to this question helps the team to establish a project schedule by identifying when project tasks are to be conducted and when milestones are to be reached.

Who is responsible for a function ?

Earlier in this chapter, we noted that the role and responsibility of each member of the software team must be defined. The answer to this question helps to accomplish this.

Where they are organizationally located ?

Not all roles and responsibilities reside within the software team itself. The customer, users, and other stakeholders also have responsibilities.

How will the job be done technically and managerially ?

Once product scope is established, a management and technical strategy for the project must be defined.

How much of each resource is needed ?

- The answer to this question is derived by developing estimates.
- W5HH principle is applicable regardless of the size or complexity of a software project. The questions noted provide an excellent planning outline for the project manager and the software team.

11.3 Metrics in the Process and Project Domains

University Question

Q. Explain in detail software process and project metrics. **SPPU : May 15, Dec. 15, 9 Marks**

Process metrics or measurements are acquired across all projects and over long intervals of time. Their intent is to provide a set of process indicators that lead to long-term software process improvement. Project metrics permit a software project manager to perform the following :

- Assess the status of an ongoing project
- Track potential risks
- Uncover problem areas before they go "critical"
- Adjust work flow or tasks
- Evaluate the project team's ability to control quality of software work products.

11.3.1 Process Metrics

- The only logical and reasonable way to escalate any process is to measure specific traits of the process, develop a set of understandable metrics based on these attributes, and then use the metrics to provide pointers that will lead to a plan for improvement.

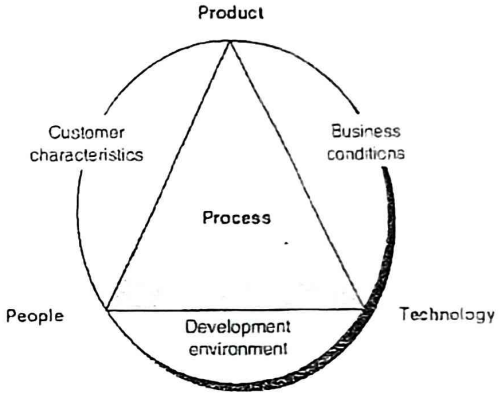


Fig. 11.3.1 : Determinants for software quality and organizational effectiveness

- In the Fig. 11.3.1, the process resides at the center of a triangle. The three corners of the triangle are actually the factors that have a profound impact on software quality and the organizational performance. They are :
  - Product
  - People
  - Technology
- The skill set and motivation of the team members are shown to be the most significant factor in quality and performance. The other parameters like complexity of the product, also has a greater impact on quality and team performance. And finally the technology will also have some impact.

11.3.2 Project Metrics (Metrics For Project)

- Actually the projects metrics are tactical ones i.e. the project metrics and the indicators derived from them are generally used by a project manager and a software team in order to adapt workflow and various activities.
- The very first application of metrics on various software projects is observed as estimation.
- With the progress of a project, the measures of effort and actual time spent are compared to the original estimates and schedule.
- As soon as technical work begins, other significant project metrics may also be taken into consideration.

- In addition to these project metrics, the errors detected during each task are also tracked.
- The intent of project metrics is twofold :
  - First, to avoid delays and mitigate potential problems and risks.
  - Secondly, the project metrics are used to assess the product quality on an ongoing basis and whenever necessary can modify the technical approach to improve quality.

11.4 Software Measurement

- We have seen that software measurement can be classified in two ways:
  - **Direct measures** of the software process and product (e.g. Lines of Code (LOC) produced).
  - **Indirect measures** of the product that includes functionality, quality, complexity, efficiency, reliability, maintainability, and many other abilities.
- Project metrics can be consolidated to create process metrics that are public to the software organization as a whole. However, if the measures are normalized then it is possible to create software metrics that enable comparison to broader organizational averages. Both size-oriented and function-oriented metrics are normalized in this manner.

11.4.1 Size-Oriented Metrics

University Question

Q. Explain size oriented metrics. **SPPU : May 14, 4 Marks**

- Size-oriented software metrics are derived by normalizing quality and productivity measures. This metric also consider the size of the software product. If any software organization keeps the simple records of software development process, then a table containing the size-oriented measures can be created. The example of such a table is shown in Fig. 11.4.1.

Project	LOC	Effort	S(000)	Pp. doc.	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
.	.	.	.	.	.		
.	.	.	.	.	.		
.	.	.	.	.	.		

Fig. 11.4.1 : Size-oriented metrics

- This table has the list of each software development projects that are already completed over the past few years and it keeps the record of their corresponding measures for project.

- To develop metrics for a project that has the similar metrics compared to other projects, the developer can select the lines of codes as a standard normalization value.
- In fact, the size-oriented metrics are not considered as foolproof metrics and it is globally not accepted as the best metrics for software process.

11.4.2 Function-Oriented Metrics (FP and LOC)

- The second important metric is function-oriented software metrics that use an amount of the work delivered by the application as a standard normalization value. The most commonly used function-oriented metric is Function Point (FP). The computation of the function point depends on the information domain and complexity of the software project.
- FP is function oriented software metric that is programming language independent. It used in various applications that are using conventional and nonprocedural languages. It depends on the data known in the beginning of the project development or early evolution of the project. These are the reasons why FP is a good and attractive estimation approach.
- The computation of FP depends on subjective data rather than objective data. Later on, it may be difficult to keep the counts of the information domain and FP will be a simple number and it would have no physical meaning.
- LOC metric is a software metric used to measure the size of a computer program by counting the number of lines in the text of program's source code.
- LOC is typically used to predict the amount of effort that will be required to develop a program.

11.4.3 Reconciling LOC and FP Metrics

- The Lines of Code (LOC) and Function Points and their relationship are based on the programming language used to implement the software. The relationship also depends on the quality of the design. Various researches have been observed in the past that tried to relate LOC and FP measures.
- Table 11.4.1 provides a rough estimates of LOC required to develop a FP in different programming languages.

Table 11.4.1 : LOC per function point

Programming language	Average	Medium	Low	High
Access	35	38	15	47
ASP	62	-	32	127
Assembler	337	315	91	694
C	162	109	33	704
C++	66	53	29	178
COBOL	77	77	14	400
FORTRAN	-	-	-	-
FoxPro	32	35	25	35
Informix	42	31	24	57

Programming language	Average	Medium	Low	High
Java	63	53	77	-
JavaScript	58	63	42	75
JSP	59	-	-	-
Lotus Notes	21	22	15	25
Oracle	30	35	4	217
PeopleSoft	33	32	30	40
Perl	60	-	-	-
Powerbuilder	32	31	11	105
Smalltalk	26	19	10	55
SQL	40	37	7	110
Visual Basic	47	42	16	158

- A review of these data indicates that one LOC of C++ provides approximately 2.4 times the functionality (on average) as one LOC of C.
- Practically the FP and LOC based metrics are considered as accurate measure of software development effort and cost.

11.4.4 Comparison between FP and LOC

FP	LOC
1. FP is specification based.	1. LOC is an analogy based.
2. FP is language independent.	2. LOC is language dependent.
3. FP is user-oriented.	3. LOC is design-oriented.
4. It is extendible to LOC.	4. It is convertible to FP (backfiring)

11.4.5 Object-Oriented Metrics

- The above discussed conventional software project metrics (i.e. LOC or FP) are also used to estimate the object-oriented software projects. The problem with these approaches is that they do not provide schedule, effort and adjustments required.
- Following are the sets of object-oriented metrics for the object-oriented projects suggested :

  Number of scenario scripts : The scenario script is actually a detailed sequence of steps used in interaction between the end-user and the product.

  - **Number of key classes :** The key classes are highly independent components defined in object-oriented analysis. These key classes sit in the centre of the problem domain.

- **Number of support classes :** The support classes are used in system implementation but they are not directly involved in the problem domain. The examples of support classes are : User interface classes, database access classes and database manipulation classes.
- **Average number of support classes per key class :** Normally the key classes are known in the beginning of the project itself. The support classes are defined throughout the development process. If the average number of support classes per key class is known in a problem domain then the estimation would be much simpler.
- **Number of subsystems :** Actually a subsystem is an aggregation of keys class and its support class to define a function that is visible to the user. Once these subsystems are accurately identified, then it would be very easy to make a schedule and the related work among project staff.

11.4.6 Integrating Metrics within the Software Process

- It is a survey that most of the software developers do not go for software measurement. Here we present certain important points that will depict the significance of software measurement.
- If developer do not measure, then it becomes too difficult to find out whether there is an improvement or not. Actually there is no any way to find out the improvement except software measurement. If there is no improvement, then the progress of the project is lost.
- During the development phase of a project and at technical level software metrics provides lot of benefits.
- Once the development is completed then developer would like to find out answers for various questions like :
  - Which of the requirements may change?
  - How much testing should be done for each component ?
  - Which of the components are prone to errors ?
- If a developer has done software measurement in the beginning of development process, then it is very easy to find out the answers for all of these questions.
- Thus we can say that software metrics acts as a technical guide for development process. It is always better to integrate metrics within the software process.
- Also by **establishing a baseline for metrics**, the immediate benefits can be obtained from process, project, products i.e. at technical levels.
- The **metrics baseline** consists of the history or the past data of various software development projects as shown in Fig. 11.4.1 (Size-oriented metrics).
- The metric baseline must have following attributes for effective process improvements :
  - All the data must be accurate and there should not be any blind guesses.
  - If possible, collect the data from various projects so that comparison becomes easy.
  - All the measures must be consistent.
  - The application under development should be similar to application estimated.
- The process of establishing the metric baseline is exhibited in the Fig. 11.4.2.

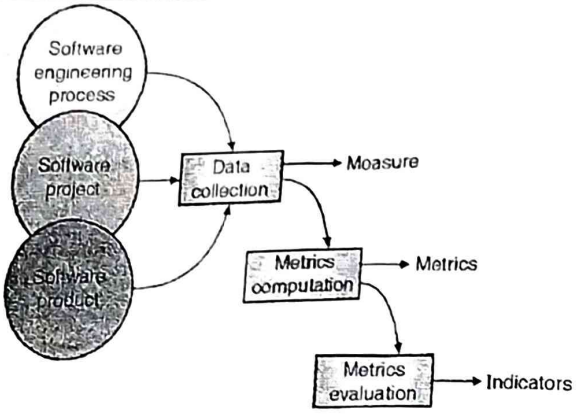


Fig. 11.4.2 : Establishing a baseline

Review Question

Q. Explain the role of people, product and process in project management.

Q. Explain the term the people of management spectrum.

Q. Explain the term in brief : Stakeholders.

Q. What are the categories of stakeholders? What are the characteristics of effective project manager?

Q. Explain the term the product of management spectrum.

Q. Explain the W5HH Principle.

Q. Explain in detail software process and project metrics.

Q. Explain size oriented metrics.

Q. Write a short notes on : Object-Oriented metrics.

❑❑❑

12

Unit - IV

PROJECT ESTIMATION

Syllabus

Software Project Estimation, Decomposition Techniques, Cost Estimation Tools and Techniques, Typical Problems with IT Cost Estimates.

12.1 Software Project Estimation

- Software cost and effort estimation will never be an exact science. Too many variables are there like human, technical, environmental, political that can affect the ultimate cost of software and effort applied to develop it.
- However, software project estimation can be transformed from a black art to a series of systematic steps that provide estimates with acceptable risk.
- To achieve reliable cost and effort estimates, a number of options arise :
  - Delay estimation until late in the project (obviously, we can achieve 100% accurate estimates after the project is complete!).
  - Base estimates on similar projects that have already been completed.
  - Use relatively simple decomposition techniques to generate project cost and effort estimates.
  - Use one or more empirical models for software cost and effort estimation.

12.2 Decomposition Techniques

- Software project estimation is a form of problem solving, and in most cases, the problem to be solved (i.e., developing a cost and effort estimate for a software project) is too complex to be considered in one piece.
- For this reason, we decompose the problem, recharacterizing it as a set of smaller (and hopefully, more manageable) problems.
- We use following two types of Decomposition Techniques :
  - Problem decomposition, and
  - Process decomposition

12.2.1 Problem Decomposition

- Problem decomposition is an activity that is present during software requirements analysis. The problem decomposition, sometimes called as **partitioning** or **problem elaboration.**
- During defining the scope of software, the problem is not fully decomposed. The decomposition activity is applied in following two important areas :
  - The functionality that is expected to be delivered and
  - The process that helps to deliver it.

- Generally all the human beings prefer a divide-and-conquer scheme when they face any complex problem. It can be stated in this way that a complex problem is divided into smaller sub problems that can be managed very easily.
- This technique is often applied in the beginning of the project planning.
- During the evolution of the statement of scope, the first level of partitioning occurs naturally. For example, the project team members learn that the marketing team has communicate with potential customers and found that the following functions should be the part of automatic copy editing required :
  - Spell checking
  - Sentence grammar checking
  - Reference checking for large documents
  - Section and chapter reference validation for large documents.
- All these features represent a sub function that is supposed to be implemented in software. Each feature can be further refined if the decomposition will make planning easier.

12.2.2 Process Decomposition

- An outline for project planning is ascertained by the process framework. It is adapted by allocating a task set that is appropriate to the project. The framework activities that characterize the software process are applicable to all software projects. The problem is to select the process model that is appropriate for the software to be engineered by a project team.
- The project manager should decide that which of the process model is most appropriate for :
  - The customers who request the product and the people who will do the work
  - The characteristics of the software product
  - The project environment in which the software team works
- When a process model is selected, the team then defines a preliminary project plan based on the set of process framework activities. After establishing the preliminary plan, the process decomposition can be started. A complete plan must be created, that reflects the work tasks required to populate the framework activities.
- It is always recommended that a software team should be adaptable in selecting the software process model that is most appropriate for the project and all the software engineering tasks can be implemented once the process model is chosen.
- Usually small projects can be completed using the linear sequential approach.
- If time constraints are imposed very strictly then the problem might be heavily modularized into several small models and generally RAD model would be the correct choice.
- If the deadline is to be followed very strictly then it is fact that full functionality may not be delivered in the first version and in this scenario, an incremental approach would be most suitable approach.
- If projects encounter with following characteristics, then some other process models might be selected :
  - Uncertain requirements
  - Breakthrough technology
  - Difficult customers
  - Significant reuse potential

- Once the process model has been finalized, the process framework is adapted to it. In all the cases, the generic framework activities like planning, communication, modeling, construction, and deployment can be used
- It will work for the following models :
  - Linear models
  - Iterative and incremental models
  - Evolutionary models and
  - Even for concurrent or component assembly models
- The process framework is invariant and serves as the basis for all software work performed by a software organization.
- But the actual work tasks do always vary. The process decomposition begins when the project manager asks "how do we accomplish this framework activity?"
- For example : A small, relatively simple project may require the following work tasks for the communication activity :
  - Develop list of clarification issues.
  - Meet with customer to address clarification issues
  - Jointly develop a statement of scope.
  - Review the statement of scope with all concerned
  - Modify the statement of scope as required.
- These events may occur in less than 48 hours. These events represent that a process decomposition is appropriate for the small and relatively simple project. It should be noted that work tasks must be adapted to the specific needs of the project.

12.3 Cost Estimation Tools and Techniques

- Estimating size or resources is one of the most important topics in software engineering and IT. You will not deliver according to expectations if you don't plan, and you cannot plan if you don't know the underlying dependencies and estimates.
- An estimate is a quantitative assessment of a future endeavor's likely cost or outcome. People typically use it to forecast a project's cost, size, resources, effort, or duration.
- Today's software market, with its increasing reliance on external components and adapted code, has led to new kinds of technologies for estimation, a practice that has moved from mere size-based approximations to functional and component estimates.
- Standards are starting to evolve as well, because these estimates play such a crucial role in business and enormous amounts of money are at stake.
- Unfortunately, people often confuse estimates with goals or plans. For instance, they schedule projects according to needs instead of feasibility, or commit to something "very urgent and important" before checking how this "urgency" relates to current commitments and capacity planning.

- In fact, most failures in software projects come from belatedly understanding and considering the important difference between goals, estimates, and plans.
- Two of the most important ingredients for proper estimation are people and historical data. They are interrelated more than you might expect because most organizations lack historical data, thus they form estimates primarily through analogy and experience.
- It works if you have experienced people who periodically measure and put their estimates versus actual data into a historical database. Tools can help reducing the time and costs involved in data gathering, as well as supporting reports, risk management, and scenario analysis.
- Following are some Software Estimation Tools :
  - **SLIM-Estimate** : It uses a proven top-down approach that minimizes the input information required to produce fact based, defensible estimates. In addition to software cost estimation, SLIM-Estimate's high level of configurability accommodates the many different design processes used by developers.
  - **SystemStar** : It offers two types of models :
    - COCOMO - Software Cost Estimation Model
    - COSYSMO - Systems Engineering Cost Estimation Model

12.4 Typical Problems with IT Cost Estimates

- Every project proposal needs realistic cost estimates. Similarly, every project manager is dependent on realistic cost estimates to allow for successful cost management.
- Budgeting and cost control is very critical but also challenging under uncertainty. Uncertainty means we do not have all information about the future, and assumptions we make today may come out differently in reality as the project progress.
- Complexity is a fundamental issue here. Simple systems are easy to understand and represent in a model. Thus, simple projects are no challenge in an analytical sense, although they too can be prone to misunderstandings and mistakes.
- The complex system is a different matter altogether. There are four types of complexity :
  - **Structural complexity** : Seeing how projects fit together and how interdependencies pose risk and uncertainties.
  - **Technical complexity** : Maturity of technology and how problems are solved through the design of processes or products.
  - **Directional complexity** : Alignment of people's objectives and motivation, and
  - **Temporal complexity** : Bringing on project parts or components at the right time and the handling of changes, especially in design, as well as cultural understanding of time.
- Project management (PM) is traditionally considered a means of planning and control of activities producing a unique product or service. A major challenge in planning is realistic cost estimation.
- This is true in IT-development projects based on a comprehensive review of previous cost estimation papers and articles.

Review Question

Q. Write a short notes on : Software Project Estimation.

Q. What are various Decomposition Techniques? Explain each in detail.

Q. Write a short notes on : Cost Estimation.

Q. What are various cost estimation tools and techniques?

Q. What are the typical problems with IT cost estimates? Why every project manager is dependent on realistic cost estimates to allow for successful cost management?

□□□

13

Unit - V

QUALITY CONCEPTS

Syllabus

Quality, software quality, Quality Metrics, software quality dilemma, achieving software quality.

13.1 Quality

Quality is nothing but **'degree of goodness'**. Software quality cannot be measured directly. Good quality software has following characteristics :

1. **Correctness** is degree to which software performs its required function.
2. **Maintainability** is an ease with which software is maintained. Maintenance of software implies change in software. If software is easy to maintain, then the quality of that software is supposed to be good.
3. **Integrity** is a security provided so that unauthorized user can not access data or information. e.g. password authentication.
4. **Usability** is the efforts required to use or operate the software.

13.2 Software Quality

University Questions

Q.	What are the software quality factors ? Explain any four.	SPPU : May 14, 12 Marks
Q.	Explain any four quality measures of software.	SPPU : Dec. 14, 8 Marks
Q.	What is the need for Software Quality ?	SPPU : Dec. 15, May 16, 4 Marks
Q.	What is software quality ?	SPPU : May 15, Dec. 15, 4 Marks
Q.	What is software quality? Explain software quality dimensions/parameters.	SPPU : May 19, 6 Marks

- The most software developers will agree that high-quality software is an important goal. In the most general sense, software quality is conformance to explicitly stated, functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.
- The definition emphasizes three important points :
  1. Software requirements are the basis for software quality measurement. If requirements are not satisfied then obviously it will yield low quality software.
  2. For the development processes, some standards are specified that define the development criteria. These development criteria will guide software engineering process. If the criteria are not followed properly then the quality of the product will be quite low.

3. There are two types of requirements :
   - Implicit requirements and
   - Explicit requirements

   Even if the explicit requirements are considered, the quality may be degraded if it fails to meet implicit requirements.

- Software quality is a complex mix of factors that will vary across different applications and the customers who request them.
- Software quality factors are identified and the human activities required to achieve them are described as follows :

1. McCall's Quality Factors
2. ISO 9126 Quality Factors

13.2.1 McCall's Quality Factors

University Question

Q. Explain different McCall's quality factors. **SPPU : Dec. 15, May 16, 4 Marks**

- The factors that affect software quality can be categorized in two broad groups :
  - Factors that can be directly measured (e.g. defects uncovered during testing) and
  - Factors that can be measured only indirectly (e.g. usability or maintainability)
- The categorization of factors that affect software quality shown in Fig. 13.2.1, Focus on three important aspects of a software product :
  1. Its operational characteristics,
  2. Its ability to undergo change and
  3. Its adaptability to new environments

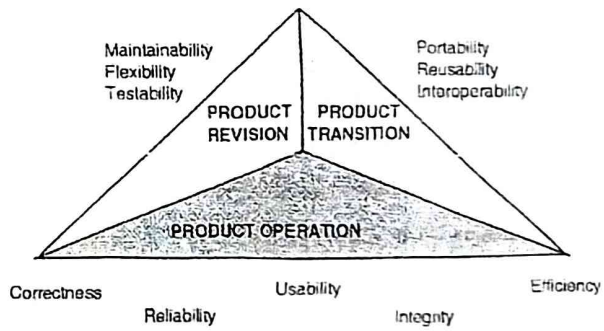


Fig. 13.2.1 : McCall's software quality factors

- Referring to the factors noted in Fig. 13.2.1, provide the following descriptions :
  1. **Correctness :** The correctness is the quality factor that assesses the software for its correct functioning according to the specification and customer's requirements.
  2. **Reliability :** It is the quality attribute that evaluates the software for its working according to the desired precision.

3. **Usability** : The software usability is defined as the degree to which the application software is easy to use. It defines the efforts taken to operate and learn, submit input and interpret output.
4. **Efficiency** : It is defined as the amount of various resources required to execute a given program. The computing resources are memory devices, processing devices and input output devices etc.
5. **Maintainability** : It is defined as the effort needed to uncover the errors and the problems in functioning of a program and resolve these errors.
6. **Integrity** : The integrity of a system is defined as the efforts taken to manage access authorization. The integrity of a system can be ensured by controlling all the accesses to it.
7. **Flexibility** : The ease with which a program can be modified and extended in future.
8. **Testability** : Amount of time and the efforts taken to test a program.
9. **Portability** : Shifting a program from one environment to another.
10. **Reusability** : How existing codes can be reused in future developments according to the scope of the software.
11. **Interoperability** : Ability to connect one system to the another system.

13.2.2 ISO 9126 Quality Factors

University Questions

Q. Explain the following quality factors: i) Maintainability ii) Reusability. **SPPU : May 12, 4 Marks**

Q. Explain ISO 9126 Quality Factors. **SPPU : Dec. 16, 5 Marks**

The ISO 9126 standard was developed in an attempt to identify quality attributes for computer software. The standard identifies six key quality attributes :

1. **Reliability** : The software should be available for the use and should satisfy following key attributes :
   - Maturity
   - Fault tolerance i.e. ability to withstand against failures
   - Recoverability i.e. the system should regain its original state once the failure occurs. It should also ensure the data integrity and consistency while recovering from failure states.
2. **Portability** : Portability means the system should be able to work in different hardware and software environments according to the following attributes :
   - Installability
   - Adaptability
   - Conformance
   - Replaceability
3. **Efficiency** : Efficiency means making the optimal use of the system resources like memory devices, processing devices and input output devices etc. It should take into consideration following attributes :
   - Time behaviour and
   - Resource behaviour.

4. **Maintainability** : Maintainability is defined as the ease with which the software may be repaired and ready to use once again. The maintainability should take into consideration following attributes
   - o Changeability
   - o Testability
   - o Stability and
   - o Analyzability
5. **Functionality** : The degree to which the application software satisfies the customer's needs and requirements according to the following attributes :
   - o Accuracy of working model
   - o Suitability of the product
   - o Interoperability (i.e. the product can be connected to any system),
   - o Compliance and
   - o Security
6. **Usability** : The software usability is defined as the degree to which the application software is easy to use according to the following usability attributes :
   - o Understandability of the software by the end-uses
   - o Learnability means learning the use of the product and
   - o Operability.

- The ISO 9126 factors do not support direct measurement of quality, but provides indirect measurement.
- It provides a very good checklist for evaluating the quality.

13.3 Quality Metrics

- Quality metrics are a key component of an effective quality management plan and are the measurements used in ensuring customers receive acceptable products or deliverables. Quality metrics are used to directly translate customer needs into acceptable performance measures in both products and processes.
- Project managers must be able to assess the progress, efficiency, and performance of their projects and metrics are the means which allow project managers to do this. However, it is important to note that metrics must be established in an effort to directly improve the product or processes involved in the project.
- The Software Quality metrics are classified into three broad categories :

1. Product metrics
2. Process metrics
3. Project metrics

- Software quality metrics are a subset software metrics that focus on the quality aspects of the product, process, and project. These are more closely associated with process and product metrics than with project metrics.

13.3.1 Product Metrics

- Describes the characteristics of the product such as size, complexity, design features, performance, and quality level.

- The terms measure, measurement, and metrics are often used interchangeably.
- Within the software engineering context, a measure provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of a product or process.
- Measurement is the act of determining a measure.
- The IEEE Standard defines metric as "a quantitative measure of the degree to which a system, component, or process possesses a given attribute."
- Measurement occurs as the result of the collection of one or more data points (e.g., a number of component reviews and unit tests are investigated to collect measures of the number of errors for each).
- Software metric relates the individual measures in some way, e.g., the average number of errors found per review or the average number of errors found per unit test.

13.3.1(A) The Challenge of Product Metrics

- Over the past three decades, many researchers have attempted to develop a single metric that provides a comprehensive measure of software complexity.
- Dozens of complexity measures have been proposed, each takes a somewhat different view of what complexity is and what attributes of a system lead to complexity. There is a need to measure and control software complexity.
- If we consider a metric for evaluating an attractive car, this is really very complicated to measure, because everybody will be having their own definition of attraction. In case of software metric, the problem occurs.
- Each of the "challenges" is a cause for caution, but it is no reason to dismiss product metrics. Measurement is essential if quality is to be achieved.

13.3.1(B) Measurement Principles

- Before we introduce a series of product metrics, it is important to understand basic measurement principles. Measurement process that can be characterized by five activities :
  - Formulation : The derivation of software measures and metrics appropriate for the representation of the software that is being considered.
  - Collection : The mechanism used to accumulate data required to derive the formulated metrics.
  - Analysis : The computation of metrics and the application of mathematical tools.
  - Interpretation : The evaluation of metrics in an effort to gain insight into the quality of the representation.
  - Feedback : Recommendations derived from the interpretation of product metrics transmitted to the software team.
- Software metrics will be useful only if they are characterized effectively and validated so that their worth is proven. The following are some principles that can be proposed for metrics characterization and validation :
  - A metric should have desirable mathematical properties. That is, the metric's value should be in a meaningful range. Also, a metric that purports to be on a rational scale should not be composed of components that are only measured on an ordinal scale.
  - When a metric represents a software characteristic that increases when positive traits occur or decreases when undesirable traits are encountered, the value of the metric should increase or decrease in the same manner.
  - Each metric should be validated empirically in a wide variety of contexts before being published or used to make decisions.

13.3.2 Process Metrics

- This metrics describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.
- Process metrics or measurements are acquired across all projects and over long intervals of time. Their intent is to provide a set of process indicators that lead to long-term software process improvement.
- Project metrics permit a software project manager to perform the following :
  - Assess the status of an ongoing project,
  - Track potential risks,
  - Uncover problem areas before they go "critical,"
  - Adjust work flow or tasks, and
  - Evaluate the project team's ability to control quality of software work

13.3.2(A) Process Metrics and Software Process Improvement

- The only logical and reasonable way to escalate any process is to measure specific traits of the process, develop a set of understandable metrics based on these attributes, and then use the metrics to provide pointers that will lead to a plan for improvement.

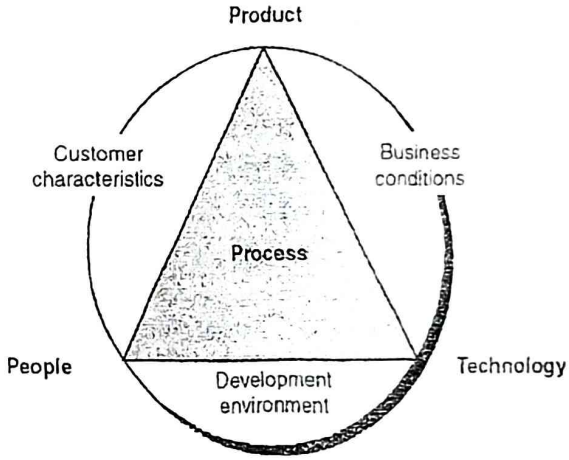


Fig. 13.3.1 : Determinants for software quality and organizational effectiveness

- Referring to Fig. 13.3.1, process sits at the centre of a triangle connecting three factors that have a profound influence on software quality and organizational performance.
- The skill and motivation of people has been shown to be the single most influential factor in quality and performance. The complexity of the product can have a substantial impact on quality and team performance. The technology that populates the process also has an impact.

13.3.3 Project Metrics

- Software project metrics are tactical. i.e., project metrics and the indicators derived from them are used by a project manager and a software team to adapt project workflow and technical activities.
- The first application of project metrics on most software projects occurs during estimation.

- As a project proceeds, measures of effort and calendar time expended are compared to original estimates and the project schedule.
- As technical work commences, other project metrics begin to have significance.
- In addition, errors uncovered during each software engineering task are tracked.
- The intent of project metrics is twofold.
  - First, to avoid delays and mitigate potential problems and risks.
  - Second, project metrics are used to assess product quality on an ongoing basis and, when necessary, modify the technical approach to improve quality.

13.4 Software Quality Dilemma

- If you produce a software system that has terrible quality, you lose because no one will want to buy it.
- If on the other hand you spend infinite time, extremely large effort, and huge sums of money to build the absolutely perfect piece of software, then it's going to take so long to complete and it will be so expensive to produce that you'll be out of business anyway.
- Either you missed the market window, or you simply exhausted all your resources.
- So people in industry try to get to that magical middle ground where the product is good enough not to be rejected right away, such as during evaluation, but also not the object of so much perfectionism and so much work that it would take too long or cost too much to complete.

13.5 Achieving Software Quality

- Software quality doesn't just appear. Software quality is the result of good project management and solid engineering practice.
- This comes into play in four broad activities :
  - Software engineering methods
  - Project management techniques
  - Quality control
  - Quality assurance

13.5.1 Software engineering methods

- If you expect to build high quality software, you must understand the problem to be solved.
- You must also be capable of creating a design that conforms to the problem while at the same time exhibiting characteristics that lead to software that exhibits the quality dimensions and factors discussed earlier.
- If you follow sound methods, the likelihood of creating high quality software will increase dramatically

13.5.2 Project management techniques

- If a project manager uses estimation to verify delivery dates are achievable, schedule dependencies are understood and the team resists the temptation to use shortcuts, and risk planning is conducted so problems do not breed chaos, software quality will be affected in a positive way.
- The project plan should include explicit techniques for quality and change management.

13.5.3 Quality control

- Quality control encompasses a set of software engineering actions that help to ensure that each work product meets its quality goals.
- Models are reviewed to ensure that they are complete and consistent • Code may be inspected to uncover and correct errors before testing commences.
- A series of testing steps is applied to uncover errors in processing logic, data manipulation, and interface communication.
- A combination of measurement and feedback allows a software team to tune the process when any of these work products fail to meet quality goals

13.5.4 Quality assurance

- Quality assurance establishes the infrastructure that supports solid software engineering methods, rational project management, and quality control actions.
- Quality assurance also consists of a set of auditing and reporting functions that assess the effectiveness and completeness of quality control actions.
- The goal of quality assurance is to provide management and technical staff with the data necessary to be informed about product quality.
- In doing so, they can gain insight and confidence that actions taken to achieve product quality are working

Review Question

Q. What are the software quality factors? Explain any four.
Q. Explain any four quality measures of software.
Q. What is the need for Software Quality?
Q. What is software quality?
Q. Explain different McCall's quality factors.
Q. Explain the following quality factors :
   i) Maintainability
   ii) Reusability
Q. Explain ISO 9126 Quality Factors.
Q. Describe the difference between process and project metrics
Q. What are different metrics are available for software quality control? How they control quality of software?

□□□

14 SOFTWARE TESTING

Unit - V

Syllabus

Introduction to Software Testing, Principles of Testing, Test plan, Test case, Types of Testing, Verification & Validation, Testing strategies, Defect Management, Defect Life Cycle, Bug Reporting, debugging.

14.1 Introduction to Software Testing

University Question

Q. What is software testing ? **SPPU : Dec. 13, Dec. 14, 4 Marks**

- Testing is an important activity in software development process. Before the start of the development, testing is planned and is conducted very systematically. In order to implement the testing, there are various testing strategies defined in the software engineering literature.
- All these strategies provide a testing template to the developers. The testing templates should possess following characteristics that is applicable to any software development process :
  - For effective testing, formal technical reviews must be conducted by the development team.
  - Frequent communication between developer and customer resolves most of the complexities and confusion in the beginning itself. Thus before start of the testing process, number of errors may be uncovered and then fixed.
  - Testing starts from the component level and then finally complete computer based system is integrated at the end.
  - There are various testing techniques are available. So at different point of time, suitable testing technique may be applied.
  - Testing may be conducted by the software developer itself for usually smaller projects. For the larger projects, an independent group (especially those people that are not the part of development team) may be hired for conducting an effective testing.
  - The members of independent testing group may be the member of some other team or may be outsourced.
  - Even though testing and debugging are two separate activities, debugging should be accommodated in testing strategies.
- Any testing strategy should be able to conduct low level tests, since it verifies small source code modules and can be easily implemented.
- It gives better precision. In addition to these low level tests, a testing strategy should also be able to conduct high level tests and it should be able to validate all the major functionalities as per customer's requirements.

14.2 Software Testing Fundamentals

Testability

- The software developer builds a software system or a product keeping **testability** in mind. The testability enables an individual to design the test cases with ease. He can design the effective test cases for effective testing.
- The software testability is defined as : "how easily a software can be tested effectively". It is a degree with which software supports testing. If the software testability is high, it means testing is easy.

Operability

- The meaning of operability is the characteristic of a system that makes it working.
- For software, it is the fact that if it works properly then the testing can be conducted more efficiently.
- The system may have some bugs, if it has no bugs, it will block the execution of tests.
- The product keep on evolving in its functionality i.e. simultaneous development and testing can be performed.

Observability

- Observability means an ability to see or observe. In software, the observability is an ability to see what is being tested. The outputs are generated for each of the input values. The outputs are usually distinct.
- All the system states and variables are visible. It can be queried during the execution of the software. Also previous system states and variables are also visible and it can be done by maintaining the transaction logs.
- All the factors that affect the output are also visible to the developer. Any incorrect output can be easily identified by the developer. But the internal errors are automatically detected by using self testing mechanisms. These internal errors are also reported automatically and the source code is accessible.

Controllability

- If we can control the software properly then the testing can be automated and optimized very well. Various outputs can be generated by using some combination of input. Also the complete code is executed by using some combination of input.
- The states of software and hardware and variables used, can be controlled by the developer itself.
- The formats of input and output should be consistent and structured.
- All tests cases can be easily specified, automated, and reproduced.

Decomposability

- Controlling the scope of testing means, finding the cause of the problem quickly and once it is done, smarter retesting can be done.
- Since the software is developed from independent modules, these independent software modules can be tested separately.

Simplicity

- If amount of errors are less, then very quickly we can conduct and complete testing.
- The functional simplicity means :
  - The functionality list provided by the customer is the minimum necessity to satisfy requirements.

- The structural simplicity means :
  - Architecture is partitioned to control the propagation of new errors.
- The code simplicity means :
  - Some coding standard is used for convenient and simple testing and maintenance.

Stability

- If changes or the modifications are less, then the disruptions to testing will be minimized.
- Modifications to the software are not very frequent.
- Modifications are controlled.
- Changes to the software also validate existing tests.
- The software always recovers from all types of failures.

Understandability

- If we have more information, then we can conduct tests in smart way.
- The design of the program is well understood by the user.
- All the dependencies and interrelationship among internal, external, and shared components are very well understood.
- All the modifications to the design are properly communicated.
- The technical documentation is important and it is easily accessible. It is well organized. Also it is specific and in details. It is accurate.

Software engineer develops a software configuration i.e. programs, data, and documents. All these attributes are amenable to testing.

4.2.1 Test Characteristics (Attributes of good test)

Following are some attributes of a "good" test :

A good test has a high probability of finding an error.

A good test is not redundant. There is no point in conducting a test that has the same purpose as another test.

A good test should be "best of breed", i.e. the test that has the highest likelihood of uncovering a whole class of errors should be used.

A good test should be neither too simple nor too complex.

.3 Principles of Testing

niversity Question

What are the principles of software testing ? **SPPU : Dec. 15, 5 Marks**

A good software engineer must understand the basic principles for software testing before he can apply various ods to effective test cases. Following are the set of testing principles that have been suggested :

All tests should be traceable to customer requirements : Its single goal is to uncover faults by triggering failures. Any inference about quality is the responsibility of quality assurance but beyond the scope of testing. It ollows that the most severe defects are those that cause the program to fail to meet its requirements.

2. **Tests should be planned long before testing begins :** Test planning can begin as soon as the requirements model is ready. All tests can be planned and designed before any code has been generated.
3. **The Pareto principle applies to software testing :** Pareto principle implies that 80 percent of all errors uncovered during testing will likely be traceable to 20 percent of all program components.
4. **Testing should begin "in the small" and progress toward testing "in the large." :** The first tests planned and executed generally focus on individual components. As testing progresses, focus shifts in an attempt to find errors in integrated clusters of components and ultimately in the entire system.
5. **Exhaustive testing is not possible :** The number of path permutations for even a moderately sized program is exceptionally large. Therefore it is impossible to execute every combination of paths during testing.
6. **To be most effective, testing should be conducted by an independent third party :** The software engineer who created the system is not the best person to conduct all tests for the software. Therefore, to make the testing effective, it is advisable to conduct the testing from the third person who is not involved in the development process.

14.4 Testing Life Cycles

- Software Testing Life Cycle (STLC) is the testing process which is executed in systematic and planned manner. In STLC process, different activities are carried out to improve the quality of the product. Let's quickly see what all stages are involved in typical Software Testing Life Cycle (STLC).
- Following steps are involved in Software Testing Life Cycle (STLC). Each of the steps has their own Entry Criteria and deliverable :
  - Requirement Analysis
  - Test Planning
  - Test Case Development
  - Environment Setup
  - Test Execution
  - Test Cycle Closure
- Ideally, the next step is based on previous step or we can say next step cannot be started unless and until previous step is completed. It is possible in the ideal situation, but practically it is not always true.

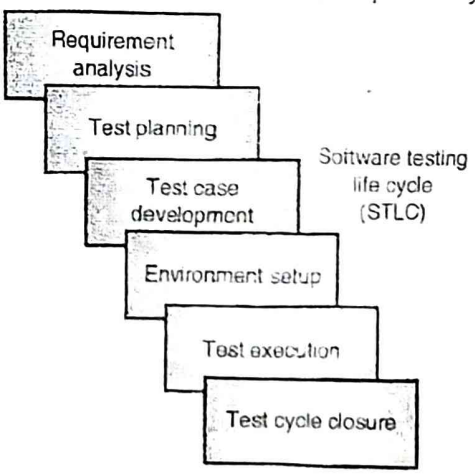


Fig. 14.4.1 : Software Testing Life Cycle (STLC)

14.4.1 Requirement Analysis

- Requirement Analysis is the very first step in **Software Testing Life Cycle (STLC)**. In this step Quality Assurance (QA) team understands the requirement in terms of what we will testing & figure out the testable requirements.
- If any conflict, missing or not understood any requirement, then QA team follow up with the various stakeholders like Business Analyst, System Architecture, Client, Technical Manager/Lead etc. to better understand the detail knowledge of requirement.
- From very first step QA involved in the where STLC which helps to prevent the introducing defects into Software under test. The requirements can be either Functional or Non-Functional like Performance, Security testing.

14.4.2 Test Planning

- Test Planning is most important phase of Software testing life cycle where all testing strategy is defined. This phase also called as Test Strategy phase.
- In this phase typically Test Manager (or Test Lead based on company to company) involved to determine the effort and cost estimates for entire project.

14.4.3 Test Case Development

- The test case development activity is started once the test planning activity is finished. This is the phase of STLC where testing team write down the detailed test cases.
- Along with test cases testing team also prepare the test data if any required for testing. Once the test cases are ready then these test cases are reviewed by peer members or QA lead.

14.4.4 Test Execution

- Once the preparation of Test Case Development and Test Environment setup is completed then test execution phase can be kicked off.
- In this phase testing team start executing test cases based on prepared test planning & prepared test cases in the prior step.

14.4.5 Test Cycle Closure

- Call out the testing team member meeting & evaluate cycle completion criteria based on Test coverage, Quality, Cost, Time, Critical Business Objectives, and Software.
- Discuss what all went good, which area needs to be improve & taking the lessons from current STLC as input to upcoming test cycles, which will help to improve bottleneck in the STLC process.

14.5 Test Plan

- A Test Plan is a detailed document that describes the test strategy, objectives, schedule, estimation, deliverables, and resources required to perform testing for a software product. Test Plan helps us determine the effort needed to validate the quality of the application under test. The test plan serves as a blueprint to conduct software testing activities as a defined process, which is minutely monitored and controlled by the test manager.

- It lays out the overall objective and scope of the tests to be run. The purpose of a test plan is to set expectations and align the team for success throughout the entire testing process.
- A test plan does not include the individual test cases, as it is meant to be a higher-level document. A test plan ultimately aligns teams to deliver a more reliable product to their users by orchestrating a smooth testing process.
- Test planning is the first thing that should happen in the software testing life cycle. The test plan document is a living and breathing artifact – it is dynamic in the sense that it should always be up to date. So, when plans change, the test plan should be updated in order to maintain an accurate set of information for the team.

14.5.1 Test strategy vs. Test plan

- It is not uncommon to see "test strategy" and "test plan" compared to one another or sometimes used interchangeably. It's often the case that the test strategy is written as part of the test plan, in its own dedicated section of the master test plan.
- A test strategy is often used at the organization level, it is a static document about the project or product. The decision to test is strategic in itself, and the test strategy lays out the "what" and "why" of testing within the organization. The test strategy rarely changes, hence its static nature. A test strategy is usually created by a project or product manager.
- A test plan is used at the project level and is a dynamic document catered to the uniqueness of the project. It's more specific in nature as it focuses on the when, how and who of testing. A test plan is usually created by the test lead or test manager.

14.5.2 The importance of a test plan

- A test plan helps you and your team get on the same page. It serves as a framework and guide to ensure your testing project is successful, and it helps you control risk.
- The very act of writing a test plan helps you think through things in a way you might not consider without writing them down. Therefore, the value of writing a test plan is tremendous by itself.
- A test plan is a great way to communicate a testing project's objectives across the entire team. This is especially useful in a world of remote workers, where testing teams might be spread across the globe, and asynchronous communication is more common.

14.5.3 How to write a test plan ?

- Writing your first test plan might be difficult, but it will start to feel natural the more you do it. A manager or team lead is usually responsible for writing the test plan, while others on the team contribute and support the process.
- Following are four tips for writing a good test plan :
  1. **Keep it concise** : It's easy for a test plan to become overly verbose. A short and succinct test plan will be more effective and easier to read. However, every project is different. There is no set length a test plan should be – it varies by team and project. More complex projects will have longer test plans than simpler projects. A super long test plan is more likely to be ignored, so focused on brevity whenever possible.
  2. **Keep it organized** : A test plan is often scanned for specific information throughout the testing project. It's important to keep the information organized so teammates can easily find what they're looking for.

3. **Make it easy to read** : Similar to keeping the test plan organized, you need to make it easy to read. Write the test plan with your audience in mind. Keep the language simple so that anyone can read the test plan and understand it.
4. **Make it easy to update** : Plans change. A testing project is likely to change at some point, and the test plan document will need to be updated in order to remain accurate. An inaccurate test plan isn't much good. Write and store the test plan in a way that's easy to update.

14.6 Test Case

- A TEST CASE is a set of actions executed to verify a particular feature or functionality of your software application. A Test Case contains test steps, test data, precondition, postcondition developed for specific test scenario to verify any requirement. The test case includes specific variables or conditions, using which a testing engineer can compare expected and actual results to determine whether a software product is functioning as per the requirements of the customer.
- Test cases help guide the tester through a sequence of steps to validate whether a software application is free of bugs, and working as required by the end user.

14.6.1 How to write test cases for software

1. Use a Strong Title
   - A good test case starts with a strong title. As a best practice, it's good to name the test case along the same lines as the module that you are testing. For example, if you're testing the login page, include "Login Page" in the title of the test case.
   - In some cases, if the tool you're using doesn't already do this, it might makes sense to include a unique identifier in the title of the test case as well. so the identifier can be referenced instead of a long title
2. Include a Strong Description
   - The description should tell the tester what they're going to test. Sometimes this section might also include other pertinent information such as the test environment, test data, and preconditions/assumptions.
   - A description should be easy to read and immediately communicate the high-level goal of the test.
3. Include Assumptions and Preconditions
   - You should include any assumptions that apply to the test and any preconditions that must be met prior to the test being executed.
   - This information can include which page the user should start the test on, dependencies on the test environment, and any special setup requirements that must be done before running the test. This information also helps keep the test steps short and concise.
4. Keep the Test Steps Clear and Concise
   - Test cases should be simple. Keep in mind, the person who wrote the test case might not be the same person who executes the test itself. The test steps should include the necessary data and information on how to execute the test.
   - This is perhaps the most important part of a test case. Keep this section clear and concise, but don't leave out any necessary details. Write the test case so that anyone can go in and perform the test.

5. Include the Expected result
   - The expected result tells the tester what they should experience as a result of the test steps.
   - This is how the tester determines if the test case is a "pass" or "fail"
6. Make it Reusable
   - A good test case is reusable and provides long-term value to the software testing team. When writing a test case, keep this in mind.
   - You can save time down the road by re-using the test case instead of re-writing it.

14.6.2 Benefits of Writing Test Cases

- The very practice of writing test cases helps prepare the testing team by ensuring good test coverage across the application, but writing test cases has an even broader impact on quality assurance and user experience.
- Test cases will help uncover a lot of information early on; information that can ultimately improve the quality of your product.
- Here are some additional benefits to writing test cases :
  - You'll find gaps in the design early
  - You'll find usability issues
  - New hires can pick up and test the application without much, or any training
  - It builds empathy for the users of your product
  - It helps you and others learn the product
- Writing test cases puts you in your user's shoes, which builds empathy for the individuals who will actually be using your product. This empathy can easily trickle back into the design and development process, and have a broad impact. As you write test cases, you'll identify gaps and areas for improvement, things that don't quite make sense, and these things can be addressed before the application is released into production.
- Writing test cases means any new testers that are hired can easily get up to speed on the product without much training. After all, test cases outline exactly how to use the product and what is expected as a result of different actions.
- A good set of test cases accessible by other team members makes it easier for others to learn about the product as well. Think of customer support, for example. The support team can browse test cases to understand how upcoming features are going to work. They can use those test cases to write technical documentation and help content.

14.7 Types of Testing

- Any of the software applications can be tested in the following two ways :
  - It assures that all the functions specified by the customer are fully functional and again searching for more errors in each of the functions.
  - All the internal operations are working as per the specifications. All the internal components have been traversed properly to uncover the errors.

- In the above two approaches, first test approach is called as black box testing and the second approach is known as white box testing. Thus we can categorize the testing approaches into the following two types :
  - White-Box Testing
  - Black-Box Testing

14.7.1 White-Box Testing

University Question

Q. What do you understand by white box testing ? **SPPU : Dec. 16, 7 Marks**

- **White-box testing also called as glass-box testing.** It employs the test case design concept that uses the control structures. These control structures are described as component-level design to derive the test cases.
- By using the white-box testing, the software developer derive test cases that has the following characteristics :
  - It guarantees the traversal of all independent paths in a module at least once. It refers the concept of traversal of tree i.e. each of the nodes in a tree must be traversed at least once.
  - Evaluate all logical decisions and find whether they are true or false.
  - Evaluate all the loops to check their boundaries and operational bounds, and
  - Evaluate all internal data structures to confirm their validity.
- Different white-box testing methods are :
  1. Basis Path Testing
  2. Control Structure Testing

14.7.1(A) Basis Path Testing

University Questions

Q. What is basis path testing ? What is cyclomatic complexity ? How is it determined for a flow graph ? Illustrate with an example **SPPU : May 12, 8 Marks**

Q. Basis path testing covers all statement in program module. Justify with example. **SPPU : Dec. 12, 8 Marks, Dec. 16, 7 Marks**

Q. What is cyclomatic complexity ? How is it determined for a flow graph ? Illustrate with an example. **SPPU : May 14, May 19, Dec.19, 8 Marks**

Q. How test cases are derived in basis path testing ? **SPPU : Dec. 14, 6 Marks**

Q. What are the main objective of basis path testing ? Explain in detail. **SPPU : May 15, 9 Marks**

- Basis path testing use the concept of white-box testing. It enables the developer to design test cases in order to derive a logical complexity measure of a procedural design. To define a basis set of execution paths, the developer uses this complexity measure as a guide.
- In this, test cases designed are guaranteed to execute all the statements in the program at least once.
- Following are the examples of basis path testing :
  1. Flow graph notation
  2. Independent program paths
  3. Deriving test cases
  4. Graph matrices

1. Flow Graph Notation

- The flow graph depicts logical control flow using the notation illustrated in Fig. 14.7.1. Each structured construct has a corresponding flow graph symbol.
- To illustrate the use of a flow graph, we consider the procedural design representation in the Fig. 14.7.2(a).
- In the Fig. 14.7.2, to show the program control structure, a flowchart is used. Fig. 14.7.2(b) there is mapping of flowchart into a flow graph.
- Refer Fig. 14.7.2(b), to show the flow graph node are represented by circles. These circles represent one or more procedural statements.
- A single node encompasses a sequence of process boxes and a decision diamond.
- The edges or links represent flow of control and are denoted by the arrows on the flow graph. These arrows are similar to flowchart arrows.

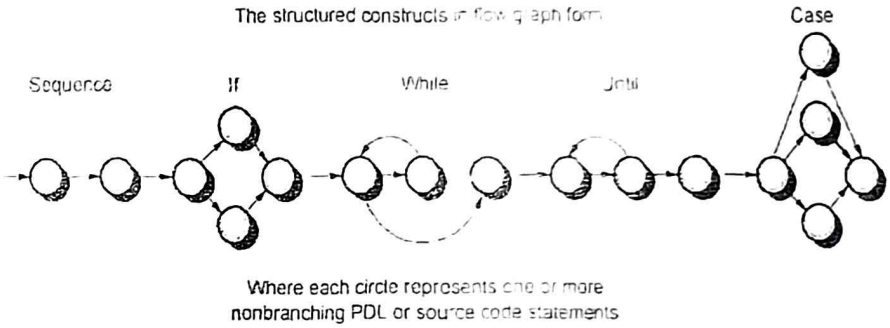
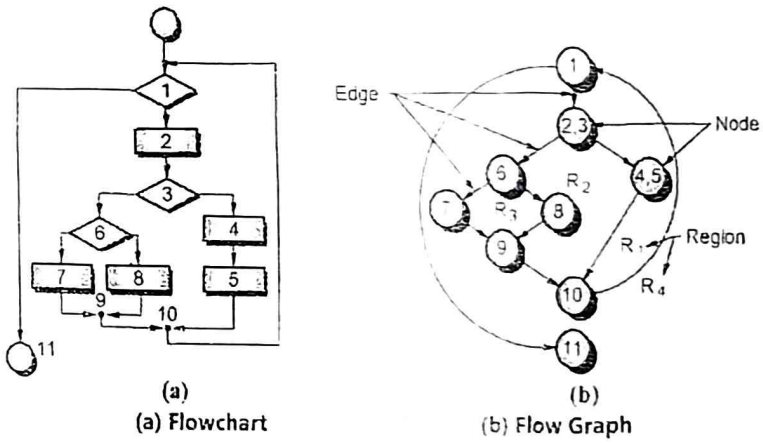


Fig. 14.7.1 : Flow graph notation



(a) Flowchart (b) Flow Graph

Fig. 14.7.2

- Even if, a node does not represent any procedural statements, an edge must terminate at a node.
- The areas surrounded by edges and nodes are referred as regions. When we consider regions, we should include the area outside the graph as a region.
- Whenever in a procedural design, the compound conditions are encountered, the creation of a flow graph becomes more complex.
- In a conditional statement, a compound condition is encountered whenever one or more Boolean operators are present. The examples of Boolean operators are logical OR, AND, NAND, NOR etc.

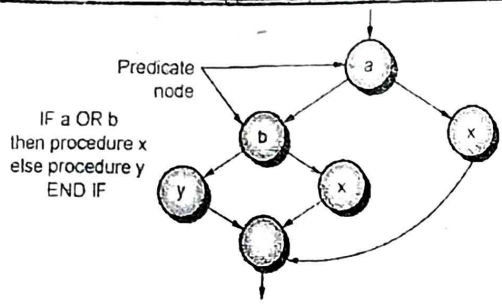


Fig. 14.7.3 : Compound logic

- The PDL (Program Design Language) segments are translated into the flow graph as shown in Fig. 14.7.3. For each of the conditions a and b, a separate node is created in the statement (i.e. IF a OR b).
- In the above flow graph, each node that contains a condition is known as predicate node. The predicate node is characterized by two or more edges coming from it (predicate node).

2. Independent Program Paths

- An independent path is defined as a path in the program that introduces at least one new set of processing statements or introduces a new condition.
- When an independent path is stated in terms of a flow graph, it must move along at least one edge and that edge should not have been traversed before defining the path.
- For example, A set of independent paths for the flow graph are illustrated in Fig.14.7.2 :

  Path 1 : 1→ 1

  Path 2 : 1→2→3→4→5→10→1 →11

  Path 3 : 1→2→3→6→8→9→10→1→11

  Path 4 : 1→2→3→6→7→9→10→1→11

- We observe that, each of the new path introduces a new edge.

  The path : 1 →2 →3→4→5→10→1→2→3→6→8→9→10→1→11

  is not an independent path since it is a combination of some previously specified paths only. It has not traversed any new any new edge.

- The paths 1, 2, 3, and 4 are considered as a basis set for the flow graph in Fig. 14.7.2, i.e. if test cases are designed to execute these paths (i.e. a basis set), then each of the statements in the program will be executed at least once, and these condition will be executed to find whether they are true or false.
- The basis set is not unique. Actually, different basis sets can be derived for a given procedural design.
- **Cyclomatic complexity** provides quantitative measure of the logical complexity of the program. The **Cyclomatic complexity** is a software metrics.
- In the context of the basis path testing, the value of Cyclomatic complexity is calculated and they are the independent paths in the basis set. The Cyclomatic complexity provides an upper bound for the test cases and it guarantees that all the statements are executed at least once.
- The Cyclomatic complexity is considered as a foundation in graph theory. It is calculated in one of the following three ways :
  - How many regions are related to the Cyclomatic complexity ?

- Cyclomatic complexity V(G) for a graph G is defined as :

  $$V(G) = E - N + 2$$

- In the above equation, E = the number of flow graph edges, and

  N = the number of flow graph nodes.

- Cyclomatic complexity, V(G), for a flow graph, G, is also defined as :

  $$V(G) = P + 1$$

- In the above equation, P = the number of predicate nodes contained in the flow graph G.

- Referring once more to the flow graph in Fig. 14.7.2, the Cyclomatic complexity can be computed using each of the algorithms just noted :

  (1) The flow graph has four regions.

  (2) V(G) = 11 edges – 9 nodes + 2 = 4.

  (3) V(G) = 3 predicate nodes +1 = 4.

3. Deriving Test Cases

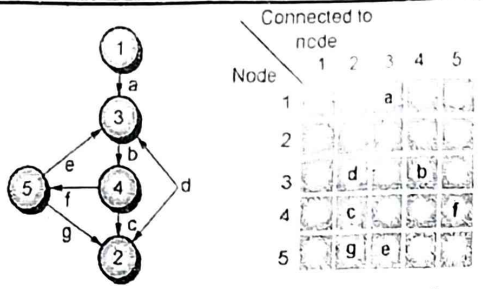
- The basis path testing method can be applied to a procedural design or to source code. In this section, we present basis path testing as a series of steps.
- The following steps can be applied to derive the basis set :
  - Using the design or code as a foundation, draw a corresponding flow graph.
  - Determine the Cyclomatic complexity of the resultant flow graph.
  - Determine a basis set of linearly independent paths.
  - Prepare test cases that will force execution of each path in the basis set.
- One important point to remember is that some of the independent paths cannot be tested alone. But the combination of data is required to traverse the path. This can not be achieved in the normal flow of the program. In these cases, the independent paths are tested as part of another path test.

4. Graph Matrices

University Question

Q. Explain graph matrix. **SPPU : May 13, 4 Marks**

- A graph matrix is a tool that supports basis path testing. A graph matrix is defined as a square matrix whose size is equal to the number of nodes on the flow graph. The size of matrix means the number of rows and columns of the square matrix.
- Each of the row and column of graph matrix corresponds to an identified node. The matrix entries as shown in Fig. 14.7.4 correspond to an edge between nodes (i.e. connections). An example of a simple flow graph and its corresponding graph matrix is shown Fig. 14.7.4.
- Each of the nodes on the flow graph is denoted by some numbers and the edges are represented by some letters as shown in following Fig. 14.7.4.
- For example : Node 3 in the following diagram is connected to the node 4 by the edge b. This can be represented in graph matrix by writing "b" in the corresponding cell.



Flow graph Graph matrix

Fig. 14.7.4 : Graph Matrix

- The graph matrix is actually a tabular representation of a flow graph. By adding a link weight to each of the matrix entry, it becomes a powerful tool for testing program control structure.

14.7.1(B) Control Structure Testing

University Question

Q. Differentiate between condition and loop testing. **SPPU : May 16, 4 Marks**

- Basis path testing is one of the techniques for control structure testing.
- Since basis path testing is not sufficient, other variations on control structure testing are discussed in the following sections. The examples of control structure testing are :

1. Condition Testing
2. Data Flow Testing
3. Loop Testing

1 Condition Testing

- The condition testing is one of the test case design method. In condition testing, all the logical conditions in the program or the program module are tested.
- An example of a simple condition may be a Boolean variable or a relational expression. The expression may contain a single NOT operator.
- Any relational expression has the following simple form :

  E_1 < the relational operator > E_2

  Where, E_1 and E_2 are two arithmetic expressions and between them, any of the following relational operator may be present :

  < → Less than

  < = → less than or equal to

  ≠ → Non-equality

  > → Greater than

  ≥ → Greater than or equal to
- A condition may also be a compound condition and it consists of two or more simple conditions, Boolean operators and parentheses.

2. Data Flow Testing

- In dataflow testing approach, a test paths of a program is selected based on the locations of definitions and uses of variables in the program.
- To explain the data flow testing method, consider that each of statement in a program is assigned a unique number. Also the functions and global variables do not modify their parameters.
- Consider a statement S (its unique statement number) as follows :

  DEFX(S) ={X | statement S contains a definition of X}

  USEX(S) = {X | statement S contains a use of X}
- If statement S is an "if" or "loop" statement, then its DEFX set is empty and its USEX set depends on the condition of statement S.
- The variable X at statement S is the live at statement S', if there is a path from statement S to statement S'. Also it should not contain any other definition of X.

3. Loop Testing

University Question

Q. Explain Loop Testing methods. **SPPU : May 13, 4 Marks**

- Loops are the important part of nearly all the programs in the software. Normally the developers pay very little attention this while conducting tests.
- The loop testing technique is in the category of a white-box testing. It focuses mainly on the validity of loop constructs. In the following section, we define four classes of loops

1. Simple loops

Following tests can be applied to simple loops, where n is the maximum number of passes through the loop :

- Skip the whole loop.
- Only one pass allowed.
- Only two passes allowed.
- m passes allowed through the loop where $m < n$.
- Also, the possibilities of n – 1, n and n+1 passes allowed.

2. Concatenated loops

This concatenated loop approach can be implemented by using the approach discussed in simple loops. In this each of the loops is independent of other. When loops are not independent, this approach is not suitable and instead nested loop approach is suggested as discussed in the next point.

3. Nested loops

The nested loop approach is an extension of simple loop approach. By nesting the loops, the number of testing levels will be increased automatically. This might result in number of tests which may not be practically possible. Following are some approaches that are suggested to reduce the number of tests :

- Start with the inner most loops and set all the other loops at minimum levels.

- Now conduct the simple loop approach to innermost loop while hold other with their minimum interaction value.
- Move outward to conduct simple test to the next loop and keeping its out loop value at minimum.
- Continue the same process till all the loops are exercised.

4. **Unstructured loops (Fig. 14.7.5)**

Whenever it is required, the loops should be redesigned by using structure using structured programming constructs.

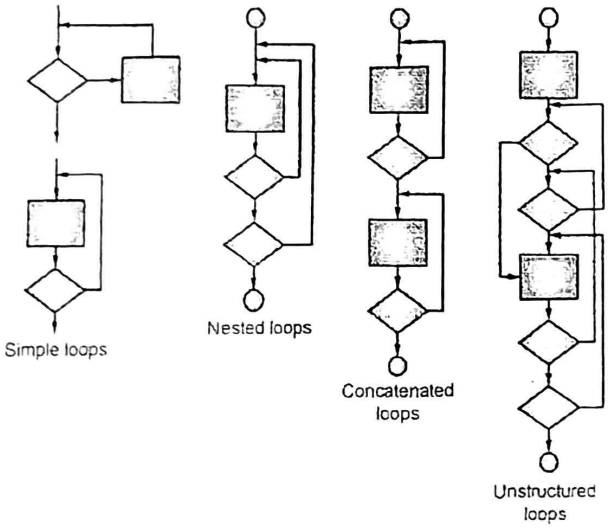


Fig. 14.7.5 : Classes of loops

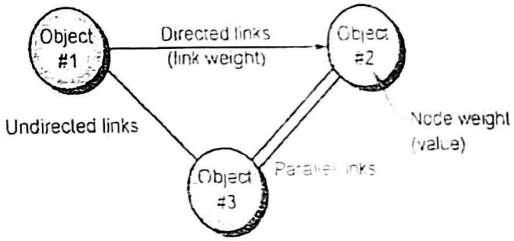
14.7.2 Black-Box Testing

- **Black-box testing is also known as behavioural testing.** It focuses only on the functional requirements of the software.
- The black-box testing helps the software developer to derive the sets of input conditions. These input conditions exercise all the functional requirements for a program.
- Black-box testing is a different approach and it can not be the alternative or the replacement of white-box testing approach. It is a complementary testing method that detects a different class of errors compared to white-box testing methods.
- Following are categories of errors that can be detected by using black-box testing approach :
  - The performance errors or the behavioural errors
  - Missing or incorrectly defined functions
  - Data structures incorrectly defined and external data base access errors,
  - Errors occurred during initialization and termination.
  - Errors occurred during interface.
- In contrast to white-box testing, where testing begins early in the testing process, the black box testing is applied in the final stages of testing.
- Different black-box testing methods are :

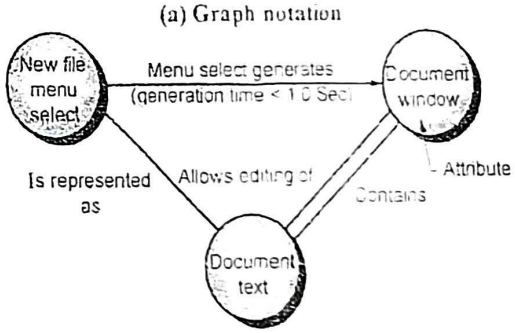
1. Graph-Based Testing Method
2. Equivalence Partitioning
3. Boundary Value Analysis
4. Orthogonal Array Testing

14.7.2(A) Graph-Based Testing Method

- The software testing starts with drawing the graphs of some important objects and their relationships. After this, the software developer performs the series of tests that cover the graph in such a way that each of the object and its relationship is covered and errors are detected.
- In order to complete these steps, the software developer starts with a graph that represents collection of objects and the relationships among them. The nodes of the graph are objects and links denotes relationships.
- In Fig. 14.7.6(a), a symbolic representation of a graph is exhibited. The circles in the graph are nodes and they are connected by links. There are different types of links as follows
  - A **directed link** denoted by an arrow, indicates that a relationship can moves in one direction only.
  - A **bi-directional link,** it is also called as symmetric link i.e. the relationship is applied in both the directions.
  - A **Parallel link,** is used in cases where various relationships exist among graph nodes.



(a) Graph notation



(b) Simple example

Fig. 14.7.6

14.7.2(B) Equivalence Partitioning

- Equivalence partitioning is a testing approach that employs black-box testing method. It divides the input domain of a program into various classes of data. Then these classes are used for deriving the test cases.
- In an ideal test case, this approach detects all the errors single-handedly from different classes of errors.
- The equivalence partitioning approach defines a test case that detects classes of errors and thus reducing the efforts of conducting more test cases.

- The equivalence classes that are generated by partitioning are defined according to the following guidelines :
  - If the input conditions specify a range then one valid and two invalid equivalence classes can be defined.
  - If an input condition requires some specific value then also one valid and two invalid equivalence classes are defined.
  - If an input condition specifies a member of a set then in this case, one valid and one invalid equivalence class are defined.
  - If an input condition is a Boolean value, then also one valid and one invalid class are defined.
- By applying these guidelines for the derivation of equivalence classes, test cases for each input domain data object can be developed and executed.

14.7.2(C) Boundary Value Analysis

University Question

Q. Explain boundary value analysis testing. **SPPU : Dec. 16, 3 Marks**

- Generally large number of errors occurs at the boundaries of the input domain and not at the "center" of it.
- This is the reason why we require Boundary Value Analysis (BVA). It has been developed and used as a testing technique by the developers. Boundary value analysis selects and conducts different test cases to test the boundary values of input domain.
- BVA is a test case design method in which the equivalence partitioning takes place. In spite of selecting some element of an equivalence class, the boundary value analysis actually selects the test cases at the "edges" of the class.
- Here in boundary value analysis approach, the focus is not only on input conditions but also it derives and conducts the test cases from the output values.
- Following are some guidelines for boundary value analysis approach :
  - If an input condition select a range between a and b then test cases should be designed beyond the values a and b. It means the input values may be just below and just above the values of a and b.
  - If an input condition selects the minimum and maximum numbers, then values just above and just below the minimum and maximum values are also tested.
  - The above two guidelines are applied to output conditions.
  - If internal program or its data structures prescribed some boundary values like an array of 100 numbers, then the care must be taken to conduct the test cases up to 100 entries only i.e. its boundary.
- Most of the software developers perform boundary value analysis up to some degree only. After applying all these guidelines on boundary value analysis testing, then it will give more complete testing and higher probabilities of finding errors.

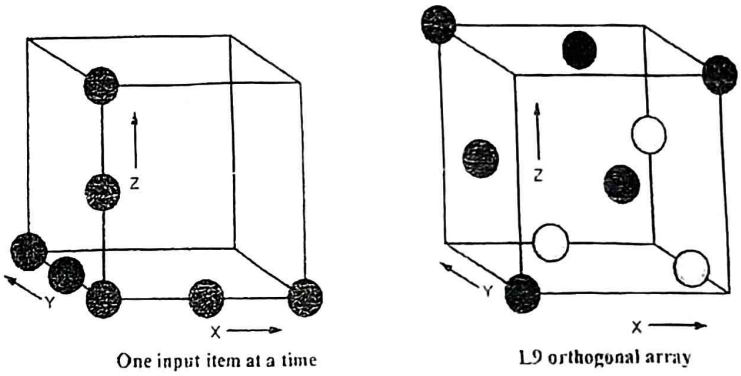
14.7.2(D) Orthogonal Array Testing

University Question

Q. Explain Orthogonal Array testing. **SPPU : Dec. 16, 3 Marks**

- In various applications, the input domain is relatively limited and hence **orthogonal array testing** is a good option in such situations. In orthogonal array testing, the problems in which the input domain is small but it is large enough to accommodate the exhaustive testing.

- The orthogonal array testing approach is especially used in detecting errors that are caused by the faulty logic within a software component.
- To explain the comparison between orthogonal array testing and more conventional approaches, consider the following system in which there are three input values X, Y and Z.
- All these input values have three discrete values associated with each of it. The probability of test cases is $3^3 = 27$.
- In the following Fig. 14.7.7, a geometric view of the possible test cases associated are explained by considering the input values i.e. X, Y, and Z.



One input item at a time L9 orthogonal array

Fig. 14.7.7 : A geometric view of test cases

- Refer the Fig. 14.7.7, in that one input value at a time is changing in accordance with the input axis. The results obtained are generally having less coverage of the input values.
- An L9 orthogonal array of test cases is created whenever an orthogonal array testing is conducted. Generally a L9 orthogonal array has an important property called as **balancing property.** In this the test coverage more complete with the input domain.

14.7.3 Differentiation between White-box and Black-box Testing

University Question

Q. Differentiate between white box and black box testing. **SPPU : May 12, Dec. 12, May 15, 4 Marks**

Sr. No.	White box testing	Black box testing
1.	White-box testing also called as glass-box testing. It employs the test case design concept that uses the control structures. These control structures are described as component-level design to derive the test cases.	Black-box testing is also known as behavioral testing. It focuses only on the functional requirements of the software.
2.	It guarantees the traversal of all independent paths in a module at least once. It refers the concept of traversal of tree i.e. each of the nodes in a tree must be traversed at least once.	The black-box testing helps the software developer to derive the sets of input conditions. These input conditions exercise all the functional requirements for a program.

Sr. No.	White box testing	Black box testing
3.	It evaluates all logical decisions and finds whether they are true or false. It evaluates all the loops to check their boundaries and operational bounds. It evaluates all internal data structures to confirm their validity.	It evaluates performance errors or the behavioural errors. It evaluates missing or incorrectly defined functions. It evaluates incorrect data structures and external database errors. It evaluates errors occurred during initialization and termination and interface errors also.
4.	The white box testing begins early in the testing process.	The black box testing is applied in the final stages of testing
5.	Different white-box testing methods are : Basis Path Testing, Control Structure Testing	Different black-box testing methods are : Graph-Based Testing Method, Equivalence Partitioning, Boundary Value Analysis, Orthogonal Array Testing

14.8 Verification and Validation

University Question

Q. Explain validation testing. **SPPU : May 14, 3 Marks**

- Verification and validation is a set of activities that ensures that all customer's requirements are fulfilled by the software and all the functions are implemented and working correctly. The Verification and Validation (V&V) is basically one of the elements of software testing.
- The validation ensures that the software is developed and it is traceable to all the customer's requirements.
- The Boehm defines verification and validation in following way :
  - Verification means evaluating whether the developer build product in a right way ?
  - Validation means the evaluating whether developer built right product ?
- The V&V consists of different activities that are also the part of SQA (Software Quality Assurance).
- The software quality assurance consists of following important activities :
  - FTR (Formal Technical Reviews) : Frequent meetings between developer and customer to ensure effective communication on technical part of the software.
  - Performance monitoring.
  - Quality and configuration audits
  - Algorithm analysis
  - Development testing
  - Feasibility study
  - Database review
  - Simulation
  - Qualification testing and installation testing
  - Documentation review
  - Usability testing

14.8.1 Difference between Verification and Validation

University Question

Q. Differentiate between verification and validation **SPPU : May 12, Dec. 13, 4 Marks, Dec. 15, May 16, 4 Marks**

Sr. No.	Verification	Validation
1.	Verification is a set of activities that ensures that all customer's requirements are fulfilled by the software and all the functions are implemented and working correctly.	The validation ensures that the software is developed and it is traceable to all the customer's requirements.
2.	The Verification is basically one of the elements of software testing.	The Validation is also one of the elements of software testing.
3.	Verification means evaluating whether the developer build product in a right way ?	Validation means evaluating whether developer built right product ?
4.	Verification is static.	Validation is dynamic.
5.	Verification evaluates all the documentation, the planning, the coding, requirements and specifications.	Validation means the evaluation of the complete product itself.
6.	Verification takes place before validation.	Not vice versa.
7.	**The inputs for verification** are : Technical meetings, various issues, review meeting etc.	**The input for validation** are : the complete testing of the product itself.
8.	**The output of verification :** Flawless documents, perfect plans and nearly completed specifications along with requirement.	**The output of validation :** The perfect final product.

14.9 Testing Strategies

- For the conventional software, there are various test strategies available for use. In first one, the team waits for a product development to be completed and then start conducting tests.
- In another approach, the developer conducts the test on a daily basis as the development proceeds and one part of the program is completed.
- These strategies are chosen by the development team based on the product and convenience of the team. The choice may among two approaches discussed above.
- The team starts with an incremental view of testing and test individual program units. Then moving ahead with integration of the units, and finalizing the tests of overall system.

Different types of tests are mentioned below :

1. Unit Testing
2. Integration Testing
3. Validation Testing
4. System Testing

14.9.1 Unit Testing

University Question

Q. Explain unit testing. **SPPU : May 12, 4 Marks**

In unit testing, the main focus is on assessment of smallest unit of the product design like component of the software or module. The unit testing has limited scope and it emphasizes on internal processing logic and data structures. Multiple modules and components can be tested in parallel.

Unit test considerations

- The unit testing is illustrated diagrammatically as follow in Fig. 14.9.1 :

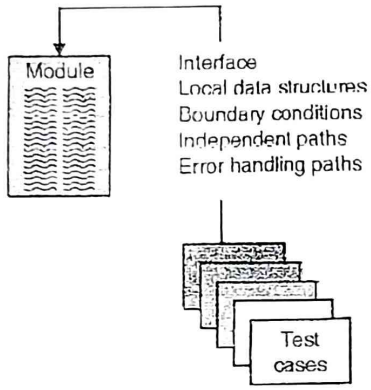


Fig. 14.9.1 : Unit test

- The test strategy is conducted on each of the **module interface** to assess the proper flow of input and output and its correctness as per the requirements.
- Next, the **local data structures** are assessed to ensure its integrity in the execution of the algorithm.
- All the **independent paths** (also called basis paths) are also evaluated through the control structure and it keeps track on the program and makes sure that at least each of the statements in a module should have been executed once.
- **Boundary conditions** are also tested so that the software works properly within the boundaries or within the limits. All the error **handling paths** are tested.
- Thus the **boundary testing** is one of the significant unit testing methodology.

Unit test procedures

- In parallel with coding step, usually unit testing is conducted. Before the start of coding, unit test can be conducted. This method is most commonly used in agile development process.
- The design information gives sufficient help for the developers to establish the test cases and uncover the errors. The developers always couple the set of results with the test cases.

Unit test environment

- The unit test environment is illustrated in Fig. 14.9.2

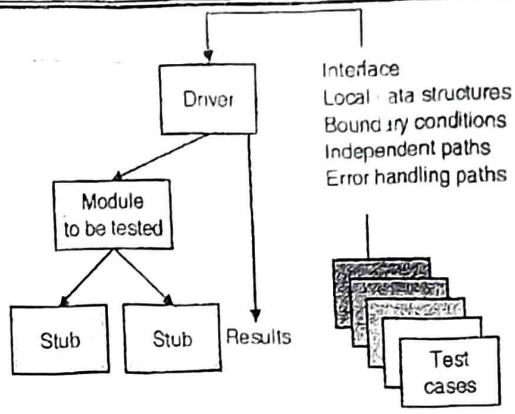


Fig. 14.9.2 : Unit test environment

- In most of the applications, a driver is considered as the "main program". This main program itself accepts all the test case data and pass that data to the component under test and the results are also printed side by side
- In the above diagram, we observe that the stubs replace the modules of the program and are tested next to driver. The stub is also considered to be the subprogram. These subprograms make an interface with the main program to complete the test.
- Logically both the driver and stubs are the software that are written but not submitted to the customer and thus are considered as the overhead. It is always recommended to keep these overhead simple to reduce the cost But overhead can not be made simple in all the cases and hence the unit testing can be postponed to the integration testing.

14.9.2 Integration Testing

Q. What do you understand by Integration Testing? Explain objectives ofintegration testing.. **SPPU : Dec. 19, 6 Marks**

- Integration testing is used for constructing the software architecture. It is a systematic approach for conducting tests to uncover interfacing errors. The main objective behind integration testing is to accept all the unit tested components and integrate into a program structure as given by the design.
- It is often observed that there is a tendency to attempt always non-incremental approaches. All the components are integrated in the beginning and the program is tested as a whole.
- In this approach a set of errors are encountered and correction seems to be very difficult due to isolation of causes and the complications by program since program is expanded over several modules. After correction of these errors, some new may appear and the process continues. It looks like an infinite loop.
- Small increments of the program are tested in an incremental approach. In incremental integration approach, the error detection and correction is quite simple. In this all the interfaces are tested completely and thus a systematic test strategy may be applied.
- Following are different incremental integration strategies :

(1) **Top-down integration**

  - Top-down integration testing is one of the incremental testing strategies for construction of the software architecture.

- All the modules are combined by moving top to down direction through the control hierarchy. It begins with the main program or the main control module. The flow is from main module to its subordinate modules in depth-first or breadth-first manner.
- Depth-first integration is illustrated in Fig. 14.9.3, and it integrates all the components on most of the control path of the program.

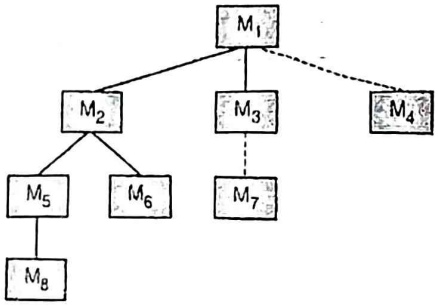


Fig. 14.9.3 : Top-down integration

(2) Bottom-up integration

- In bottom-up integration testing, the testing begins with the construction and components at the lowest level. These components are called as atomic modules.
- Since the components are integrated from the bottom to top, the required processing is always available for components subordinate in all the levels. Here in this approach the need of stub is completed eliminated. The use of stub was actually a disadvantage due to overhead.
- Following steps are required for implementation of bottom-up integration strategy :
- The low-level components are integrated to form clusters that can perform sub function of a specific software.
- A driver is needed to coordinate all the test case inputs and outputs.
- Later all the clusters are tested.
- Then, drivers are removed and all the clusters are integrated once again from bottom to top direction in the program.
- Integration follows the pattern illustrated in Fig. 14.9.4.

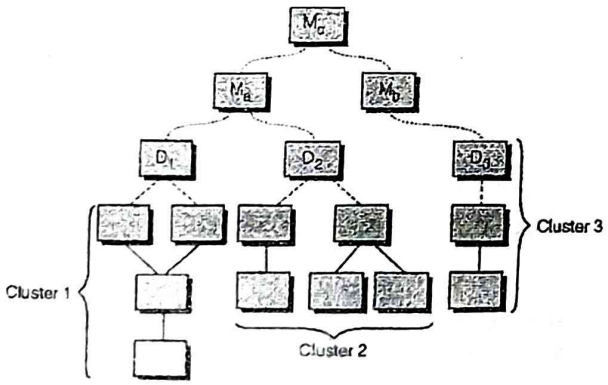


Fig. 14.9.4 : Bottom-up integration

- The components are integrated in such a fashion that they can form clusters 1, 2 and 3. These clusters are tested by using a driver (marked by a dashed block).

- The components in clusters 1 and 2 are subordinate to module M_a. The drivers D_1 and D_2 are then removed and the clusters are connected directly to M_a.
- In the same way, the driver D_3 for the cluster 3 is removed and then cluster 3 is directly connected to module M_b. Here both the modules, M_a and M_b are finally integrated with the component M_c and so on.

Problems associated with Top down approach of testing

- In incremental integration testing approach, a set of errors are encountered and correction seems to be very difficult due to isolation of causes and the complications by program since program is expanded over several modules. After correction of these errors, some new may appear and the process continues. It looks like an infinite loop.
- Top-down integration testing is one of the incremental testing strategies for construction of the software architecture.
- The main problem with top down approach is that the test conditions are nearly impossible or they are extremely difficult to create.
- Stub modules are complicated.

Comments on Integration Testing

- The selection of any integration strategy is based on the software characteristics or the project schedule.
- Generally a combined approach i.e. both the top-down and bottom-up are used. It is called as sandwich testing. The top-down tests can be implemented for upper levels and bottom-up tests are implemented for subordinate levels of the program. Actually the sandwich testing is the best compromise.

Integration Test Documentation

- The complete test plan for integration testing is documented in a test specification.
- This document encompasses a test plan and a test procedure that is actually a work product of the software process, and it becomes the part of the software configuration.

14.9.3 Validation Testing

- Validation is defined by various means, but a very simple definition is that a software is developed in such a way that it fulfils customer's expectations and all the functions of the software are working properly. Validation is only successful when software works reasonably according to customer's expectation.
- The reasonable expectations are written properly in the SRS (Software Requirements Specification) documents.

14.9.3(A) Validation Test Criteria

- The software validation is done by conducting a series of tests that a show satisfies the customer's requirements.
- A **test plan** draws the classes of tests that are to be conducted, and a **test procedure** will define specific test cases.
- In validation test criteria, both the plan and procedure are designed in such a way that they ensure all functional requirements are satisfied. It also ensures all behavioural characteristics are achieved. The performance requirements must be attained and documentation must be correct. In addition the usability and other requirements are meeting properly.

- In each validation test case, following two possible conditions exists :
  (1) The performance characteristic is meeting the specification and is accepted, or
  (2) Any deviation from the specification is detected and a deficiency list is generated.
- The error detection at this stage is rarely corrected before the scheduled delivery.
- It is a better practice to negotiate with the customer to find some solution for the deficiencies detected.

14.9.3(B) Configuration Review

- An important part of the validation process is concept of configuration review.
- The motive behind configuration review is ensuring that all the elements of the software configuration developed properly and they are catalogued too. All these information is necessary for maintenance phase of life cycle.

14.9.3(C) Acceptance Testing

- The acceptance test is conducted by the end-user rather than software developer. The need is to find the range of input values by the customer. Thus the acceptance test is well planned and is executed systematically. The series of acceptance test are usually conducted to validate the requirements.
- The acceptance testing is conducted over a period of weeks or months and thereby detecting the cumulative errors. These errors may degrade the system over time.
- If the software is built as a product that is to be used by different customers, then it is practically not possible to perform formal acceptance tests with each of the user.
- In short, when customized software is developed for one customer, then a series of acceptance tests are necessary to validate all the requirements by the customer.
- The acceptance testing can be classified as :
  1. Alpha testing 2. Beta testing

14.9.3(D) Alpha and Beta Testing

University Question

Q. Differentiate between alpha and beta testing. **SPPU : May 19, Dec. 19, 6 Marks**

- It is very difficult for a software developer to visualize how the end user will actually use his program. The instructions for use may be read but not understood properly or some wrong meaning can be drawn. Also some combinations of input may be regularly used and the output that is correct for tester but it is not satisfied by the user in actual practice.
- When customized software is developed for one customer, then a series of acceptance tests are necessary to validate all the requirements by the customer.
- The acceptance test is conducted by the end-user rather than software developer. The need is to find the range of input values by the customer. Thus the acceptance test is well planned and is executed systematically. The series of acceptance test are usually conducted to validate the requirements.
- The acceptance testing is conducted over a period of weeks or months and thereby detecting the cumulative errors. These errors may degrade the system over time.

- If the software is built as a product that is to be used by different customers, then it is practically not possible to perform formal acceptance tests with each of the user.
- In such a scenario, most of the software developers use **alpha and beta testing** to find those errors that can be detected the end-user only.
- The **alpha test** is conducted at developer's end by end-users. The end users work on the software and developer has a keen observation on all interactions of user with the software in an environment that is controlled by the developer. The developer keep on recording all the errors and problems encountered. Alpha tests are generally conducted in a controlled environment.
- The **beta test** is conducted at end-user place. Here the developer is generally not present. Hence the beta test is supposed to be the "live" application of the software.
- This environment is not controlled by the developer in contrast to the alpha testing that is done at developer's end.
- In beta testing, the end-user will record all the problems (may be a real one or imaginary) that are encountered and will report to the developer at regular basis.
- After getting the reports from the end user, the developer will make modifications and then prepare the new release of the software and deliver to the customer.

Difference between Alpha Testing and Beta Testing

Sr. No.	Alpha testing	Beta testing
1.	The alpha test is conducted at developer's end by end-users.	The beta test is conducted at end-user place. Here the developer is generally not present. Hence the beta test is supposed to be the "live" application of the software.
2.	The end users work on the software and developer has a keen observation on all interactions of user with the software in an environment that is controlled by the developer.	This environment is not controlled by the developer in contrast to the alpha testing that is done at developer's end.
3.	The developer keep on recording all the errors and problems encountered. Alpha tests are generally conducted in a controlled environment.	In beta testing, the end-user will record all the problems (may be a real one or imaginary) that are encountered and will report to the developer at regular basis.
4.	Developer does not know the types of users and hence unable control the errors.	In beta testing, user reports the problems and thus developer is able to correct errors.
5.	In this, developer may modify the codes at his site before the release.	In this, the developer may modify after getting the feedback from user and prepare the next release.

14.9.4 System Testing

- The **"finger-pointing"** is a classic system testing problem. It occurs whenever an error is detected, and for each of the system element, the developers blame each other for the problem.

- **System testing** is a series of different tests and the main reason of these tests is to test the complete computer-based system. The purpose of each of the tests is to verify all the system elements properly and ensure they are integrated properly and perform required functions.
- Types of System Testng
- Recovery Testing
- Security Testing
- Stress Testing

14.9.4(A) Recovery Testing

University Question

Q. What do you mean by recovery testing ? **SPPU : Dec. 13, 3 Marks**

- The computer-based systems may recover from faults that are occurring and can resume back the normal processing within a stipulated time. But in some cases, the system should be fault tolerant i.e. it should continue to functioning despite failure.
- In some other cases, the system must recovered back from the failure and corrected within a stipulated period of time.
- If not done so, then some drastic economic damage will take place. For example online shopping sites, banking sites where the data is very sensitive and critical.
- In recovery testing, a system is forcefully sent to failed state in a variety of way and testing is done to recover from those states. The motive is to test that the system is recovering properly from the failure state.
- If it is an automatic recovery, then re-initialization, data recovery, restart and check-pointing mechanisms are evaluated for correcting the errors. If the recovery requires human intervention, then the Mean-Time-To-Repair (i.e. MTTR) is calculated to find whether it is in acceptable limits or not.

14.9.4(B) Security Testing

- All the computer-based systems that handles sensitive information or causes the actions that can harm or benefit any individuals is tested for illegal attacks to the system.
- The examples of illegal activities are :
  - Hackers who try to attack systems for sport
  - Detained employees who try to attack the system for revenge
  - Some mischievous individuals who try attack system for illicit personal gain.
- Thus security testing verifies whether a protection mechanism is built or not. The mechanism should protect the system from illegal attacks and unauthorized access.
- In security testing, the tester may work as an intruder that tries to attack the system to check its security. The tester can do anything to check the system's security. For example the tester may try to obtain passwords and may attack the system to break down the defences that were constructed.
  - Overload the system by series of requests to cause denial of service attack.
  - Purposely cause system errors.
  - Attack the system during recovery.

- Browse through insecure data.
- Find the key to enter into the system.

- Good security testing will definitely attack the system.

14.9.4(C) Stress Testing

- In stress testing, the test cases are conducted to bring the programs into abnormal situations purposely. Why this is done ? Because before the delivery, the attempt is to uncover every possible error by trying to fail the system by taking abrupt inputs.
- Stress testing is executed in such a way that the system demands resources to be in abnormal frequency, volume or quantity.
- For example
  - Input data rates are increased to determine how input will respond.
  - Test cases that cause excessive searching of data on disk are created.
  - Test cases that cause problems in memory management are created.
  - Tests cases are created that generate interrupts.
  - Test cases are created that require excessive memory or other excessive resources.

14.10 Defect Management

University Question

Q. Write a short note on defect management **SPPU : May 19, 4 Marks**

- Software development teams and software testing teams have numerous choices of defect management tools to help support their software defect efforts. Selecting and utilizing an effective tool is really only part of an overall defect management system.
- From a high-level view, defect management systems are made up of a combination of some defect management tools or tool and a defect management process.
- These two primary components work together to support each other. Ignore either one, and sub-optimal results can be expected.
- Following is an overview of a typical defect management process and the key features to look for in an effective defect management tool. All of this information will be useful to review.

14.10.1 Defect Management Process

High Level Steps in a typical defect management process

- The typical defect management process includes the following high-level process steps. When implemented inside of a specific organization, each of these high-level steps would have more detailed standard operating procedures along with policies to carry out the details of the process.

1. **Identification** - This step involves the discovery of a defect. Hopefully, the person discovering the defect is someone on the testing team. In the real world, it can be anyone including the other individuals on the project team, or on rare occasions even the end-customer.

14.12 Bug Reporting

- Software testing process is a systematic and planned activity and it is an integral part of the development process.
- The test cases are conducted, and a strategy can be defined, and the results can be evaluated as per the requirements.
- Debugging occurs when a test case is successfully conducted. It means that the test case uncovers various errors, and the debugging process can be started. The debugging process attempts to eliminate all the errors that were uncovered in the testing process.
- Even if debugging is an orderly process, it is considered to be an art.

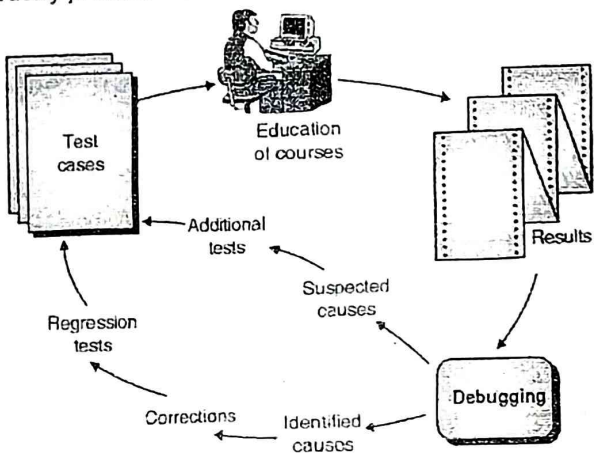


Fig. 14.12.1 : The debugging process

14.12.1 Debugging

University Question

Q. Explain the debugging process. SPPU : May 13, 8 Marks, Dec. 13, 6 Marks

- Debugging is not a testing process but it is the consequence of testing. In the Fig. 14.12.1, the debugging process starts with conduction of a test case.
- The results are evaluated and a difference between expected and actual performance is encountered.
- The debugging process has one of two outcomes as follows :
  (1) The reason for causing error are found and corrected, or
  (2) The reason for causing the error will not be found.
- In the case 2, the person performing debugging will be suspicious about a cause. Conducting the test cases will help to validate that suspicion, and the person can work to correct the errors in iterative fashion.
- It is observed that the debugging is a difficult process. Sometimes human psychology is required in addition to software technology. Following are some characteristics of bugs that will provide some clues to clear the doubts about debugging process :
  - The cause or the symptom can be geographically remote.
  - The symptom may be unavailable temporarily, while some other errors are corrected.

- The symptom may also be caused by non-errors (e.g., round-off inaccuracies).
- The symptom may be caused by some human error and that can not be easily traced.
- The symptom may be due to timing problems rather than processing problems.
- It is very difficult to reproduce input conditions accurately. For example real-time applications where order of the input values is indeterminate.
- The symptom can also be intermittent i.e. not continuous. This scenario is very common in embedded systems.
- The symptom can also be caused by number of tasks running on different processors in a distributed system scenario.

- In the debugging process, numerous error are encountered that may be mild annoying or catastrophic.
- The examples of mild annoying errors are :
  - Incorrect output format
- The examples of catastrophic errors are :
  - System fails
  - Physical damage
  - Economical losses
- In debugging process, finding the cause of the errors is more important that finding the errors itself. The software developer sometimes eliminates an error and at the same time, two more errors are introduces.

14.12.2 Psychological Considerations

- Debugging is considered to be the natural process and some people are good in debugging and some are not good in debugging. Thus there is large variance in the debugging process. It is evident from the various reports that the developers with the same education and same experiences have different debugging capabilities.
- Based on human abilities of debugging, Shneiderman gave following statements :
  - Debugging is the most frustrating phase of development. It consists of problem solving and brain teasers, along with some annoying recognition that you made a mistake.
  - The anxiety and unwillingness to admit the possibility of errors, usually increases the difficulty of the task. But finally all the bugs are corrected.
  - It is quite difficult to "learn" debugging process as there are number of approaches exists for problem solving.

14.12.3 Debugging Approaches

- Regardless of the approach that is chosen, debugging has one simple objective i.e. to detect and correct the cause of error. The objective can be achieved by different evaluation techniques.
- Bradley has described the debugging approach in the following way :
  - Debugging is an application of scientific method.
  - In debugging, the basic idea is to locate the problem's source by partitioning it.

Consider the following non-software example

- Suppose a lamp in a house is not working, it means that lamp is faulty. If everything in the house is not working, it means there is problem in the main circuit breaker. If I see in my neighbourhood, everything is blackout, then I can find there is problem in the main connection in my area and so on.
- In this scenario I have to start correcting the problems from top and reach to the specific location.
- This hypothesis may be useful in conducting the test.
- Following are three important categories for debugging approaches :

(1) Brute force
(2) Backtracking
(3) Cause elimination

(1) Brute force

- The Brute force is the most common and least efficient method for finding the cause of a software error.
- This category of debugging i.e. Brute force debugging methods is applicable generally when all the other methods are failed. In this method, computer find the errors itself since memory dumps are used and thus run-time traces are invoked. In this, the program is loaded with the WRITE statements.
- In the complicated and confused pool of information, the developer is able to reach to the clue to find the cause of errors.
- The mass information produced will lead to success finally. Sometimes it may lead to wasted of time and effort. Therefore first, the thought should be expended.

(2) Backtracking

- Backtracking category of debugging is more common approach and is used successfully in especially small programs. In this approach, the hint or the point where errors are seen is selected as a starting point and the source code is traced backward. The backtracking is a manual process and continues till the cause of the error is found.
- In this category, the task is increased as compared to previous approaches. The number of source code lines are increased and sometimes the large number of backward paths are unmanageable.

(3) Cause elimination

- The third category of debugging use the concept of binary partitioning. The cause elimination approach is achieved by induction or deduction.
- Data that causes errors or somehow that data is related to the error occurrence, are organized properly to find the actual cause of potential errors. In this category of debugging, a "cause hypothesis" is used. In this, any previously mentioned data is taken to prove or disprove the hypothesis. In this, the list of causes is prepared and finally the tests are conducted to eliminate these errors.
- If tests prove the cause hypothesis then the data is refined to find the errors.
- These debugging approaches are supported by various debugging tools available to make it more effective.
- Various debugging compilers are applied and various dynamic debugging aids called as tracers are used.

- Also the automatic test case generators, cross-reference maps and memory dumps are also used to make this category of debugging more effective. But one should be remembered that these tools can not be the substitute of debugging process, they are only the supporting tools.
- As soon as the errors are found, they are eliminated, but during the correction of these errors, some new errors can be introduced. So instead of some good thing, more damage can be caused in this process.
- **Van Vleck** stated following three simple questions that must be answered before correction starts :

1. Check whether the cause of error is reproduced in some another part of the program?

   A) In most of the cases, the program errors are caused by erring logic and this erroneous logic may also be reproduced in another part of the program. The explicit consideration should be taken for all the logical pattern. This consideration may discover the errors.

2. What will be the next error, if I fix one ?

   B) So before any correction is made, the tester should assess the source code to find the use of any logic in another part of the same program. A utmost care is taken in correction so that interrelated logic or coupling should not cause any new errors.

3. What precaution should be taken to prevent the error at the first place only?

   C) By using proper SQA (Software Quality Assurance) approach, this question can be answered. If the process and the product, both are corrected simultaneously then all the errors may be eliminated from the current programs as well as future programs.

Review Question

Q. What is software testing?

Q. What are attributes of good test?

Q. What are characteristics of good test?

Q. What are various Software Testing Fundamentals?

Q. Explain Software Testing Life Cycle (STLC) in details?

Q. What are different phases of software testing?

Q. What is unit testing? Explain the unit testing process.

Q. Explain the integration testing approaches.

Q. Explain and compare incremental integration testing strategies.

Q. What do you understand by the term integration testing? Which types of defects are uncovered during integration testing?

Q. What do you understand by integration testing? Explain objectives of integration testing.

Q. Explain Top-down integration testing in detail

Q. What are the problems associated with top down approach of testing

Q. Explain bottom up integration testing strategy in detail

Q. What do you mean by recovery testing?

Q. What do you mean by performance testing?

Q. Write a short notes on :

1 Security Testing

2 Stress Testing

Q. Explain regression testing.

Q. What do you mean by acceptance testing?

Q. Explain acceptance testing.

Q. What is the difference between alpha testing and beta testing?

Q. What do you understand by white box testing?

Q. What is basis path testing? What is cyclomatic complexity? How is it determined for a flow graph? Illustrate with an example.

Q. Basis path testing covers all statement in program module. Justify with example.

Q. What is cyclomatic complexity? How is it determined for a flow graph? Illustrate with an example.

Q. How test cases are derived in basis path testing?

Q. What are the main objective of basis path testing? Explain in detail.

Q. Explain graph matrix?

Q. Differentiate between condition and loop testing.

Q. Explain Loop Testing methods.

Q. Explain boundary value analysis testing.

Q. Explain Orthogonal Array testing.

Q. Differentiate between white box and black box testing.

Q. Explain validation testing.

Q. Differentiate between verification and validation.

Q. Explain unit testing.

Q. Explain smoke testing.

Q. Differentiate between regression and smoke testing.

Q. Write a short note on : Defect Management.

Q. Write a short note on : Defect Life Cycle.

Q. Explain the debugging process.is software testing?

□□□

15 RECENT TRENDS IN SOFTWARE ENGINEERING

Unit - VI

Syllabus

SCM, Risk Management, Technology evolution, process trends, collaborative development, software reuse, test-driven development, global software development challenges, CASE - taxonomy, tool-kits, workbenches, environments, components of CASE, categories (upper, lower and integrated CASE tools), Introduction to agile tools Jira, Kanban.

15.1 Software Configuration Management

University Questions

Q. Write a short note on: Software Configuration Management. — SPPU : May 13, May 14, 5 Marks

Q. What do you understand by Software Configuration Management (SCM)? — SPPU : May 19, 5 Marks

Q. Explain Software Configuration Management (SCM) process. — SPPU : May 19, 6 Marks

- Basically the output of any software process contains the information based on various inputs. These output can be broadly divided into three important categories as follow :
  - The computer programs
  - The work products and
  - The data that consist of the information produced.

  All these information parts collectively called as software configuration.
- If each configuration item simply led to other items, little confusion would result. During the process, change takes place which may be unfortunate. But change can occur any time and for any undefined reason. The change is the only constant.
- The first law of system engineering says that "No matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle".
- Following are four basic sources of change :
  - Every new business conditions and market conditions may cause the change in product requirement and business rules definitions.
  - Customers may require change in data presented by the information system. The customers may also demand various functionality delivered by their product of interest. The customers seek some new services. All these changes are customer oriented.
  - As the business grows or its downsizing may cause various changes in priorities of the project and also restructuring of the team happens.
  - The budget and time scheduling constraints are also important factors for change in system or product.

- SCM is a set of activities used for managing the change during the life cycle of computer software. The software configuration management can also be considered as a software quality assurance activity during the development process.

15.1.1 SCM Basics

University Questions

Q. What are elements that exist when an effective SCM system is implemented? Explain each in detail. SPPU : May 12, 5 Marks

Q. Write short note on; Elements of a Configuration management system. SPPU : Dec. 14, 6 Marks

- Four basic elements that should exist when a configuration management system is developed :
  1. **Component elements :** The tools in the file management system uses the software configuration item.
  2. **Process elements :** The process elements or the procedures uses effective approach towards the change management in engineering and use of computer software.
  3. **Construction elements :** The automated tools are used in construction or the development process and ensuring the validated components should be assembled.
  4. **Human elements :** In order to make the effective use of SCM, the team makes the use of various tools and process feature.
- These elements are not mutually exclusive. For example, component elements work in conjunction with construction elements as the software process evolves. Process elements guide many human activities that are related to SCM and might therefore be considered human elements as well.

1. **Baselines**

   The change is the only constant in software development life cycle. The customer want to modify the requirement as the model gets ready. Since in the beginning, even customer is not fully aware of the product requirement. As the development begins, customers need lot of changes in the requirement.

   Once customers modify their requirement, it is now manager who modifies the project strategy.

   - Actually as time passes, all the people involved in the product development process come to know exact need, the approach and how it will be done in the prescribed amount of time.
   - The additional knowledge is required to know the exact requirement. It is very difficult for most of the software engineers to accept this statement that most of the changes are justified.
   - The baseline is SCM that help in development process without affecting much the schedule and control the changes.
   - The IEEE provides a baseline as follows :

     "A specification and requirement that is agreed upon between customer and developer is a basis for the product development and these requirements can be changed only through change control procedures".

2. **Software Configuration Items**

   - Basically SCI i.e. software configuration item is the integral part of software engineering development process. It is a part of large specification or we can say that one test case among large suite of test cases.
   - In fact the SCI is a document or the program component like C++ functions or a Java applet.

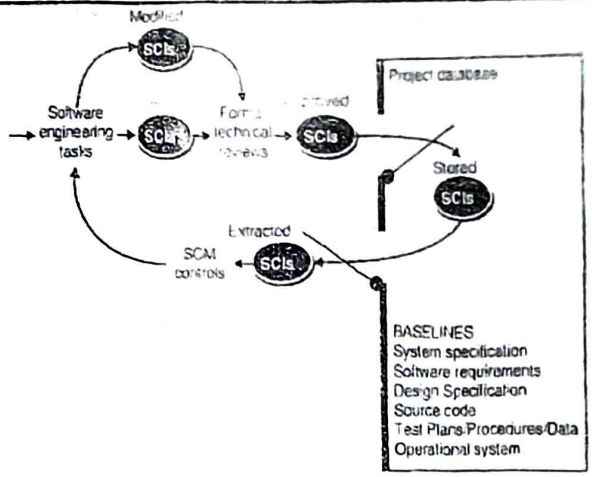


Fig. 15.1.1 : Baselined SCIs and the project database

- Most of the organizations use software tools under configuration control to help development process. In fact the editors, compilers, browsers and various automated tools are the integral part of software configuration.
- In fact the SCIs are catalogued in the project database with a single name and they form configuration objects. These objects are configuration object and it has a name, attribute and it has certain relationship with other configuration objects.
- Referring to Fig. 15.1.2, the configuration objects, Design Specification, Data Model, Component N, Source Code and Test Specification are each defined separately.

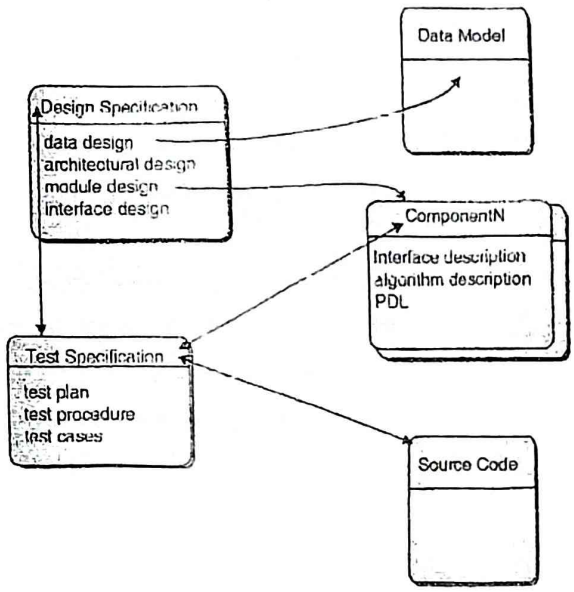


Fig. 15.1.2 : Configuration objects

15.1.2 SCM Repository

University Question

Q. What is software configuration management repository? **SPPU - May 15, Dec. 15, May 16, 5 Marks**

- In the early days of software engineering, software configuration items were maintained as paper documents (or punched computer cards), placed in file folders or three-ring binders, and stored in metal cabinets.

- This approach was problematic for many reasons:
  - To find a SCI is a difficult task when it is needed.
  - To determine which items were changed and by whom. It is always challenging.
  - To develop a new version from an existing program is prone to errors and time consuming too.
  - To describe detailed relationship is actually impossible.
- As discussed earlier that SCIs are catalogued in the project database with a single name, they reside in the repository. The repository is a database that stores the software engineering information. The software developer or engineer interacts with the repository by using built in tools within repository.

1. The Role of the Repository

University Question

Q. Discuss role of SCM repository. **SPPU : May 15, Dec. 15, 5 Marks**

- The SCM repository is the set of mechanisms and data structures that allow a software team to manage change in an effective manner. The repository will perform all the fundamental operations of database management system and in addition it will perform the following operations :
  1. **Data integrity :** Data integrity validates all the entries to the repository and make sure that the consistency among various objects intact and takes care of all the modifications takes place. It will also ensure cascading modifications i.e. change in one object causes change in a dependent object also.
  2. **Information sharing :** It is mechanism for sharing information among various developers and between various tools. These tools manage and control multi-user access to data, and locks or unlock objects to retain its consistency.
  3. **Tool integration :** Tool integration is a data model that can be used by many software engineering tools to control access to the data, and performs appropriate configuration management functions.
  4. **Data integration :** Data integration provides database functions that allow various SCM tasks to be performed on one or more SCIs.
  5. **Document standardization :** Document standardization is an important task for defining the objects. This standardization is a good approach for making software engineering documents.
  6. **Methodology enforcement :** Methodology enforcement defines an (E-R model) i.e. entity-relationship model available in the databases i.e. repository. This model may be used as a process model for software engineering. It is mandatory that the relationships and objects must define before building the contents of the repository.
- To achieve these functions, the database is used as a meta-model. This meta-model exhibits the information and how this information is stored in the databases i.e. repository. This meta-model also checks data security and integrity.

2. General Features and Content

University Question

Q. Discuss features of SCM repository. **SPPU : May 15, Dec. 15, 5 Marks**

- The contents of databases and features of databases are considered from two perspective :
  - What data is to be stored in the databases
  - What services are provided by the databases

- A detailed breakdown of types of representations, documents, and work products that are stored in the repository is presented in Fig. 15.1.3

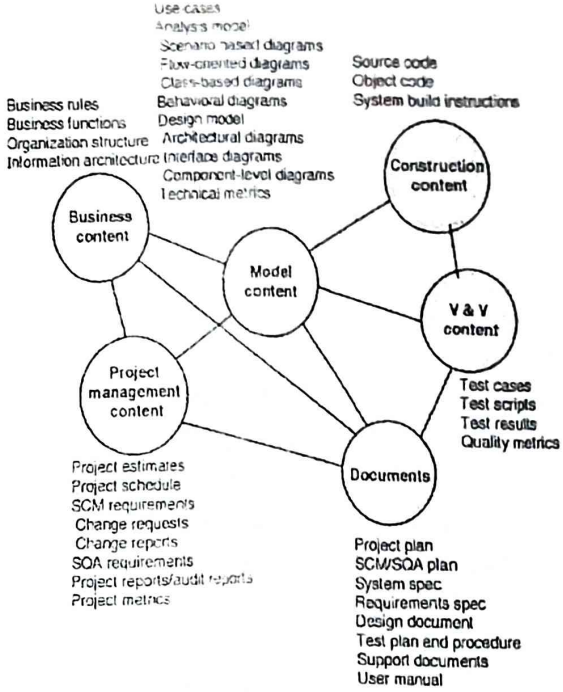


Fig. 15.1.3 : Content of the repository

- A robust repository provides two different classes of services :
  1. The same types of services that might be expected from any sophisticated database management system and
  2. Services that is specific to the software engineering environment.
- A repository that serves a software engineering team should :
  - Integrate with or directly support process management functions;
  - Support specific rules that govern the SCM function and the data maintained within the repository;
  - Provide an interface to other software engineering tools; and
  - Accommodate storage of sophisticated data objects (e.g., text, graphics, video, audio).

15.1.3 SCM Features

To support SCM, the repository must have a tool set that provides support for the following features :

1. Versioning
2. Dependency tracking and change management
3. Requirements tracing
4. Configuration management
5. Audit trails

1. **Versioning**
   - In the development process, as the project progresses, various versions of the product will be developed and the database will save all these versions.
   - The repository will keep track of these versions in order to make effective management of the product delivery or the product releases.
   - The history of all releases will be used by the developers to make effective testing and debugging.
2. **Dependency tracking and change management**
   - The databases or the repository stores various relationships among the configuration objects.
   - The relationships may be between entities and processes, or between application design and component design and between all the design elements etc.
   - Some relationships are optional and some of the relationships are mandatory relationships that have various dependencies.
   - So to keep the track of previous history and relationships is very important for the consistency of the information present in the databases. The new releases of the final product are also dependent on the history stored in the repository. This will be useful in the improvement process.
3. **Requirements tracing**
   - The requirement tracing will provide the ability to trace all the design components and releases. This tracing results from a specific requirement called as forward tracing.
   - It will also useful in finding that which requirements are fulfilled properly from the ready product. This is called as backward tracing.
4. **Configuration management**
   - The configuration management is a facility to keep the track of various configurations under development.
   - The series of configurations is used as the project milestones and future releases.
5. **Audit trails**
   - The audit trails keeps additional information (i.e. meta-data) like the changes made by whom and when. Also it stores the reasons why changes have been made.
   - The source of each change in the development must be stored in the repository.

15.1.4 SCM Process

University Questions

Q. What is the objective of SCM? SPPU : Dec. 12, 4 Marks

Q. Explain SCM process in detail. SPPU : May 16, 4 Marks

- The SCM (Software Configuration Management) process consists of series of task to monitor the control on changes being occurred. The main objectives of these tasks are as follows :
  1. To identify all individual items that can define software configuration collectively.
  2. Manage the changes taking place in various individual items.
  3. To handle different versions or releases of product.
  4. To maintain the quality of the software under construction over the period of time.

- A process that achieves these objectives must be characterized in a manner that enables a software team to develop answers to a set of complex questions
- How does a software team identify the discrete elements of a software configuration?
- How an organization manages the changes being done in existing release or the existing version? The modification should be incorporated efficiently.
- How an organization keeps the track and control of new releases?
- In an organization, who is responsible for authorizing all these changes?
- The mechanism used to let others know the changes taking place and implemented?
- These questions lead us to the definition of five SCM tasks - identification, version control and change control, configuration auditing, and reporting, as illustrated in Fig. 15.1.4.

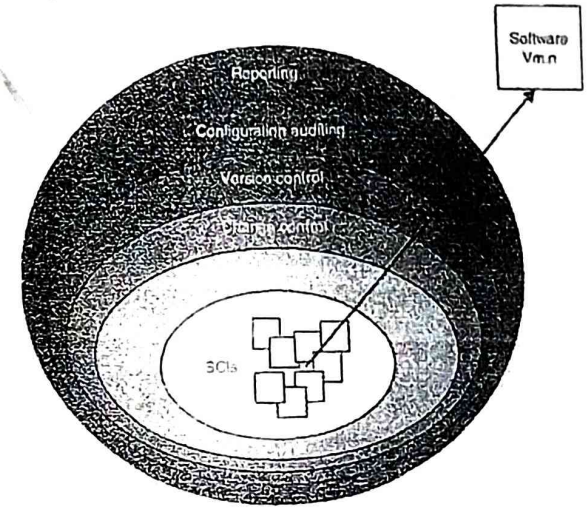


Fig. 15.1.4 : Layers of the SCM process

1. **Identification of Objects in the Software Configuration**

- To control and manage software configuration items, each should be separately named and then organized using an object-oriented approach. Two types of objects can be identified :

1. Basic objects and
2. Aggregate objects.

1. **Basic object :** A basic object is a unit of information that has been created by a software engineer during analysis, design, code, or test.
2. **Aggregate object :** An aggregate object is a collection of basic objects and other aggregate objects.

- Since each of the object in the product has some distinct features that make the object different from other objects. The object has a unique name, description and list of resources associated with it. The name of the object should be very clear and distinct.
- The description of the object should identify the type of software configuration item i.e. SCI represented by that object.

2. Version Control

- The version control actually controls the new releases or new versions. It combines the procedures and tools in order to control various versions of configuration objects.
- Any version control management system has four major capabilities that is integrated in the version control system itself :
  1. The repository will store all the related configuration objects.
  2. The repository will store all the versions of configuration objects.
  3. It has a facility to provide the related information about configuration objects so that a software engineer will construct a new version based on those information.
  4. To track all the issues in development process by using a special tracking facility in the version control.

3. Change Control

University Question

Q. Write short note on: Change control process. **SPPU : May 12, May 14, 5 Marks**

- In a development of a larger software system, the changes may be uncontrolled and it leads to a complex situation. In such projects the change control is done partially by human and partially by some automated tools. In such a complex situation human intervention is very much necessary.
- The change control process is explained in the following Fig. 15.1.5
- The change request is first submitted and then evaluated by a technical support staff by taking into consideration its potential side effects and the overall impact on other objects in the product. The other parameters to be evaluated are system functions, the cost of project etc.
- Based on the result of the evaluation treated as a change report, the implementation is taken into consideration. This report is submitted by change control authority i.e. CCA. The CCA is a person or a group who has the final authority to take decision on any changes and their priority.
- After approval from CCA, a change order called ECO (engineering change order) is generated for each of the change.
- The object to be changed can be placed in a directory that is controlled solely by the software engineer making the change.
- These version control mechanisms, integrated within the change control process, implement two important elements of change management :
  - Access control and
  - Synchronization control.
- Before an SCI become a baseline, the changes should be applied. The developer will look after whether the changes are justified or not by project. The technical requirement must check properly.
- After approval from CCA, a baseline may be created and change control is implemented.

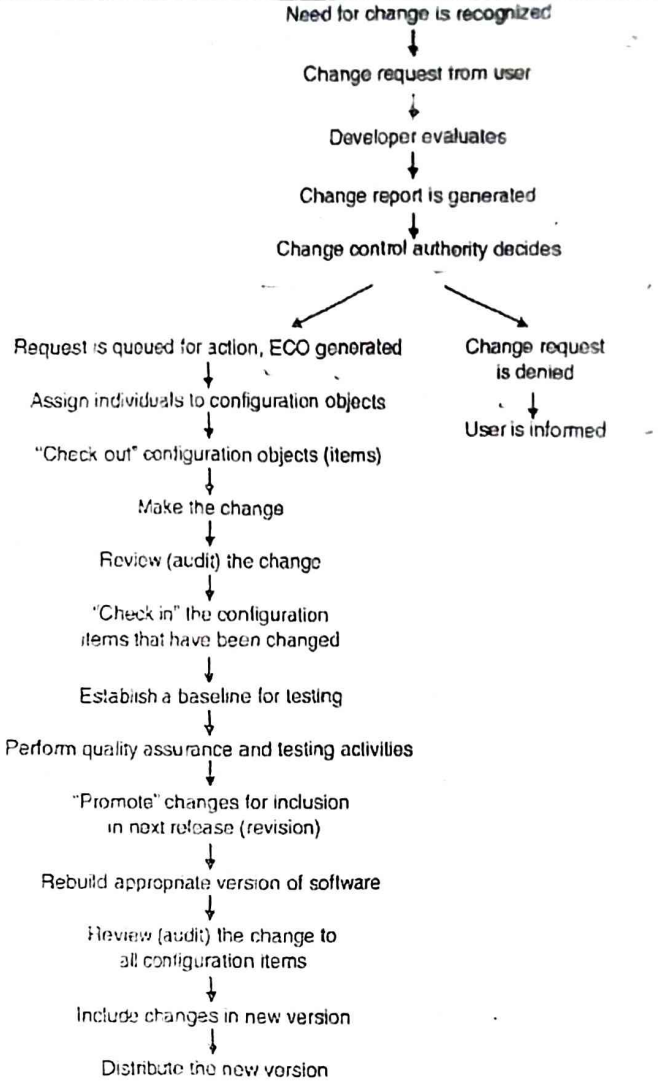


Fig. 15.1.5 : The change control process

- Once the final product is released, the formal changes must be instituted as per the figure 15.1.5. This formal change is outlined.
- The CCA plays an active role in second and third layers of change control based on size of the project.

4. **Configuration Audit**

- A software configuration audit addressees the following questions:
  - Whether the change mentioned in the ECO applied or not ? Incorporated any additional modifications?
  - Whether the formal technical review conducted or not to assess technical correctness ?
  - Whether the software process followed or not and software engineering standards properly applied ?
  - Whether the changes are "highlighted" in the SCI ?
  - Whether SCM procedures for noting the change, recording it, and reporting it been followed or not ?
  - Whether all SCIs properly updated ?
- In some of the cases, the configuration audit questions are asked as part of a FTR (formal technical review) . Still SCM is a formal activity. The SCM audit is conducted separately. These activities are performed by the quality assurance group.

5. **Status Reporting**

- Configuration status reporting is a SCM task that has following questions to answer :
  - What had happened? (Specify the changes made).
  - Who did it? (Specify the responsible person or authority approving the changes).
  - When did it happen? (Specify the time of occurrence of the change)
  - Anything else is affected based on the changes made?
- The CSR report is generated on regular basis to keep the developers aware of the changes made and the history of the changes made. It is very much useful in new releases or constructing new versions.

15.1.5 Importance of SCM

- Configuration Management (CM) is an area in software development and engineering that generally gets somewhat less attention than it really needs. Configuration Management is also critical for day to day reliability, performance and security of the product or solution.
- We can also consider Configuration Management a part of a systems engineering process that consistency of the software product after its development phase.
- In reality Configuration Management should be a part of every phase of the SDLC cycle in some respects and for that matter every part of a robust production environment.
- Every software project should have a practice of defining how project configuration management should be addressed.
- Configuration management encompasses the practices and tooling to automate the delivery and operation of infrastructure. Configuration Management is actually part of a solid change management program.
- Configuration management solutions commonly model infrastructure, continually monitor and enforce desired configurations, and automatically remmediate any unexpected changes or configuration drift.

15.2 Risk Management

University Questions

Q. Explain risk management process and various types of risks with suitable example. **SPPU : May 12, 6 Marks**

Q. What are the types of risks? Explain in brief. **SPPU : Dec.12, 6 Marks**

Q. Explain Software Risk Management. **SPPU : May 13, 6 Marks**

- Whenever we start any business or any development process, we take into consideration the risks involved in accomplishing that task. Similarly when software development process is started, it is main job of a software manager to look into all types of possible risks involved in the entire process.
- It involves focusing on the possible risks that could affect the project development process :
  - Schedule of the development process
  - Quality of the application under construction
- It is the responsibility of a project manager to look into the matter and take necessary action to avoid these risks. All the results of risk analysis and analysis of consequences of risk occurring should be well documented in the planning of the project itself.

- If the risk management is effective, then it becomes easier to handle all the problems and it is ensured that the project schedules and budget are within acceptable limits and there is no schedule slippage and ultimately no budget slippage.
- The always threaten the project development processes and the software under construction.
- Following are three important categories of risk :

1. Project risks
2. Product risks
3. Business risks

1. **Project risks**
   - These are the risks that directly affect the schedule of the project and the resources involved in the development process.
   - **Example of project loss :** Loss of an experienced developer and designer.
2. **Product risks**
   - The product risks affect the quality and performance of the application built.
   - **Example of product risks :** Failure of a purchased component to perform as per expectation.
3. **Business risks**
   - The business risks are affecting the organization those develop and process the software.
   - **Example of business risks :** A competitor of the organization introducing a new product.

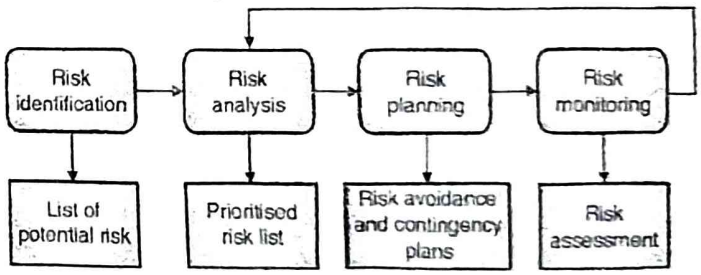


Fig. 15.2.1 : The Risks management process

15.2.1 Reasons for Project Delay

Following are few common reasons for project delay :

- One major factor that has been identified as reasons for project delay in most projects is design errors.
- It is important to note that proper representation of client's requirement and the blue print to achieving good technical input to project execution are usually mapped out base on project designs. Thus a design with errors practically means wrong or insufficient representation of project deliverables.
- Delay in project could be as a result of scope change. Scope is the term that defines the entire deliverables that is expected at the end of a project. Therefore, logically, it can be said that all project plans, estimation, schedule, quality and base lines are usually designed base in the initial project scope. Thus, any change in the project scope during execution will mean that the entire initial project plan will have to be reviewed such that a reviewed budget, schedule and quality will have to be developed.

- Another major reason for delay in project is inappropriate and inadequate procurement and faulty contractual management system. Contracts read-out virtually every aspect of a business correlation, including payment terms, pricing, and service levels. Therefore a contract that has not highlighted the entire project scenario may lead to dispute in the contract system.
- The complexity of project could also be a contributing factor to delay and cost overrun. Complexity could be defined in terms of the size of the project, most mega projects tend to have relatively long implementation period when compared to small projects.
- The post execution phase of a project contains potential factors that can lead to delays and cost overrun. Being the very last part of the project life-cycle, it is often been ignored even by organizations, especially in multi-project environments.
- Slow closeout could be seen as dragging the various handover activities course by unresolved disputes linked with client acceptance, contracts and procurement, change order issues not resolved, final change orders not issued, poor close out of final account, poor documentation of project success and lessons learnt, slow client acceptance and failing to close the work order can allow unexpected delay.

15.3 Technology Evolution

University Question

Q. Write short note on Technology Evolution. **SPPU : Dec. 18, May 19, 4 Marks**

- The capability to design quality software and implement modern information systems is at the core of economic growth in the 21st century. To use this potential is only possible when adequate human resources are available and when modern software engineering methods and tools are used.
- The recent years have witnessed rapid evolution of software engineering methodologies, including the creation of new platforms and tools which aim to shorten the software design process, raise its quality and cut down its costs.
- This evolution is made possible through ever-increasing knowledge of software design strategies as well as through improvements in system design and code testing procedures. At the same time, the need for broad access to high-performance and high-throughput computing resources necessitates the creation of large-scale, interactive information systems, capable of processing millions of transactions per seconds. These systems, in turn, call for new, innovative distributed software design and implementation technologies.

15.4 Process Trends

- It is style of software development which is focused on public availability and communication.
- Usually communication happens using internet.
- It is used in freeware, open source software and commons based peer production.
- It is also used in agile development model.
- It is generally used in development of free software.
- It is very compatible with free software.
- They meets online for software development.
- It is evolution of integrated development environment.

- Typical functionalities are as follows :
  - Version control system
  - Bug tracking system
  - Todo list
  - Mailing list
  - Document management system
  - Forum
- They also involve users in the collaborative development.
- It is used in most technological disciplines.

15.4.1 Model-driven development

- It is software design approach for development of software system.
- It provides guidelines for structuring of specifications.
- It is kind of domain engineering.
- It is launched by object management group.
- It defines system functionalities using platform independent model.
- Related standards are as follows.
  - Unified modeling language (UML)
  - Meta object factory (MOF)
  - XML metadata interchange (XMI)
  - Enterprise distributed object computing (EDOC)
  - Software process engineering metamodel (SPEM)
- Different tools are used in model driven architectures
  - Creation tools
  - Analysis tools
  - Transformation tools
  - Composition tools
  - Test tool
  - Simulation tools
  - Metadata management tools
  - Reverse engineering tools
- Model driven development is shown in Fig. 15.4.1.

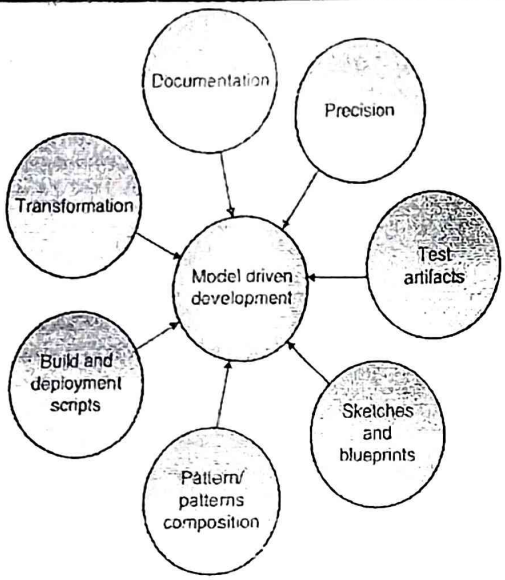


Fig. 15.4.1 : Model driven architecture

15.4.2 Test-driven development

- It is software development process which relies on repetition of very short development cycle.
- First, developer writes test cases which define a desired improvement.
- It is related to test first programming concept of extreme programming.
- Test driven development cycle is shown in Fig. 15.4.2

1. Add test
2. Run all tests
3. Write some code
4. Run tests
5. Refactor code
6. Repeat the process.

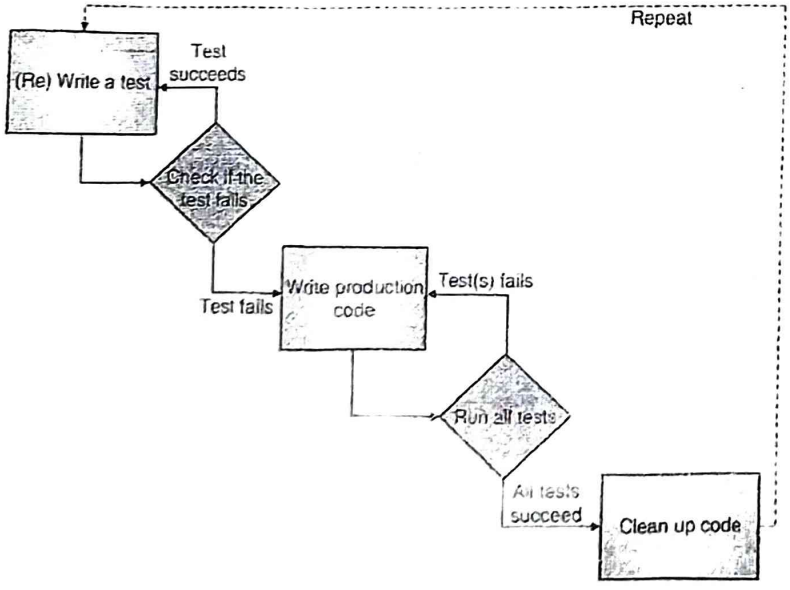


Fig. 15.4.2 : Test Driven development

15.4.3 Challenges of global software development

University Question

Q. Explain in brief global software challenges. **SPPU : May 19, 4 Marks**

- Communication breakdown
- Coordination breakdown
- Control breakdown
- Cohesion barriers
- Culture clash
- Ability to gain market share.
- Greater flexibility and variable staffing model
- Lower cost labor.
- Access to broader set of skilled workers.
- Ability to leverage outsourcing providers with specialized skill.
- Possibility of establishing a presence in geographies that may become new market for product.
- Ability to gain competitive edge.
- Business driver and global delivery challenges are shown in Fig. 15.4.3.

15.4.4 Business drivers and global delivery challenges

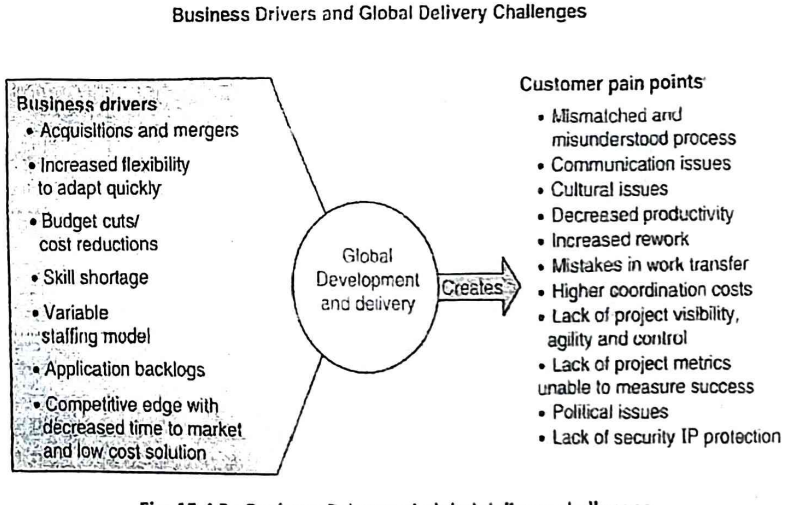


Fig. 15.4.3 : Business Driver and global delivery challenges

- Misunderstood processes or mismatched processes.
- Communication issues can lead to misunderstanding, omissions, errors and rework.

15.5 Collaborative Development

University Question

Q. Write short note on Collaborative Development **SPPU : Dec. 18, 4 Marks**

- A **collaborative development environment** (CDE) is an online meeting space where a soft vare development project's stakeholders can work together; no matter what timezone or region they are in, to discuss, document, and produce project deliverables.
- Although growing from a tool base in the software development sector, the CDE has been taken up in other sectors, with teams typically geographically dispersed, where it is beneficial to be able to collaborate across the web, including automotive and aeronautical engineering, movie production, and civil engineering.

15.6 Software Reuse

- Software reuse is a term used for developing the software by using the existing software components. Some of the components that can be reuse are as follows;
  - Source code
  - Design and interfaces
  - User manuals
  - Software Documentation
- Software reuse is the process of implementing or updating software systems using existing software assets.
  - The systematic development of reusable components
  - The systematic reuse of these components as building blocks to create new system
- It is great to know about the kinds of artefacts associated with software development that can be used again. Almost all artefacts associated with software development, including project plan and test plan, can be used again. However, the important items that can be effectively used again are,
  - Requirements specification
  - Design
  - Code
  - Test cases
  - Knowledge

15.6.1 Advantages of software reuse

- **Less effort :** Software reuse requires less effort because many components use in the system are ready made components.
- **Time-saving :** Re-using the readymade components is time saving for the software team.
- **Reduce cost :** Less effort, and time saving leads to the overall cost reduction.
- **Increase software productivity :** when you are provided with readymade components, then you can focus on the new components that are not available, just like readymade components.
- **Utilize fewer resources :** Software reuse save many sources just like effort, time, money etc.
- **Leads to a better quality software :** Software reuse save our time and we can consume our more time on maintaining software quality and assurance.

15.6.2 Problem in software reuse

- The principles, methods, and skills required to develop reusable software cannot be learned effectively by generalities and platitudes.
- To succeed in-the-large, reuse efforts must address both technical and non-technical issues.
- It's easier and more cost effective to develop and evolve networked applications by basing them on reusable distributed object computing middleware, which is software that resides between applications and the underlying operating systems, network protocol stacks, and hardware.

15.7 CASE (Computer-Aided Software Engineering)

- Computer-aided software engineering (CASE) tools are developed to help software engineers, managers and all the software practitioners in all the software activities related to the software development process. The CASE tools are actually built to automate software management activities. It also assists a software engineer in requirement analysis, design, coding and testing, debugging.
- Software engineering is considered to be very difficult. The CASE tools reduce the amount of effort required to develop a product. In most of the project, these tools play very important role to achieve the benefits. In addition to these benefits, tools also provide different way of looking at software engineering information that improves the software quality.
- CASE tools help a software engineer or a practitioner in developing high-quality products. It also produces additional customized work due to these automation tools. Without CASE tools the thing might have been too tough.
- Computer-aided software engineering provides a platform where a software engineer automates all the manual activities. This improves the insight of engineering to produce better products. Thus CASE ensures the quality before the actual product is built.

15.7.1 CASE tools

University Question

Q. Write short note on CASE Tools **SPPU : May 19, 6 Marks**

- CASE tools are class of software which is useful to automate the different activities in life cycle. They are useful in different software engineering phases. They are useful in requirement analysis, design, coding, testing, etc. Different CASE tools are given below.
  - Business process engineering tools
  - Project planning tools
  - Risk analysis tools
  - Project management tools
  - Requirement tracing tools
  - Documentation tools
  - System software tools
  - Quality assurance tools
  - Database management tools

- Prototyping tools
- Program analysis tools
- Web development tools
- Static analysis tools
- Dynamic analysis tools
- Test management tools
- Client-server testing tools
- Re-engineering tools

7. Functions of CASE tools
   - Analysis
   - Design
   - Code generation
   - Documentation
8. Types of CASE Tools
   - The general types of CASE tools are listed below:
     - Diagramming tools
     - Computer display and report generators
     - Analysis tools
     - Central repository
     - Documentation Generators
     - Code generators

15.7.2 CASE - taxonomy

Different terms involved in the CASE tools are as follows. These tools are useful to understand the CASE in details.

15.7.2(A) Workbenches

- Workbench integrates different CASE tools in one application.
- It is useful to support certain process activity.
- They generally have homogeneous and consistent interface.
- They have easy invocation of tools and tools chains.
- CASE workbenches can be classified into following categories.
  - Business planning and modeling
  - Analysis and design
  - User interface development
  - Programming

- Verification and validation
- Maintenance and reverse engineering
- Configuration management
- Project management
- Design management

15.7.2(B) Tool-kits

- It is collection of products which are [illegible]
- Support provided by tool-kits is limited to programming, configuration management, project management [illegible]
- It is extended from basic set of operating system tools.

15.7.2(C) Environments

- It is nothing but the collection of CASE tools and workbenches.
- It is useful in supporting software process.
- It is classified into different categories based on basis of integration.
  - Toolkits
  - Language centered
  - Integrated
  - Fourth generation
  - Process centered

15.7.2(D) Components of CASE

- Components of CASE tools is shown in Fig. 15.7.1.

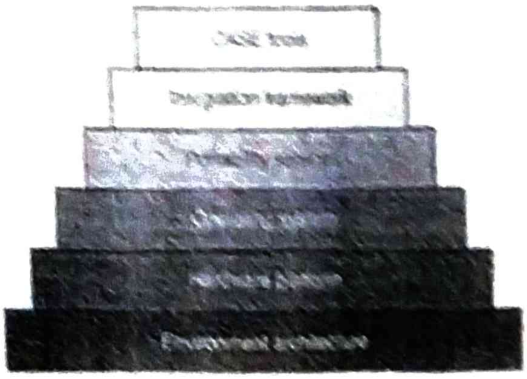


Fig. 15.7.1 : Components of CASE Tools

- It consists of different blocks like environment architecture, hardware platform, operating system, portability services, integration framework and CASE tools.

15.7.2(E) Categories

- It is used in wide variety of software development life cycle methods.
- It is also useful in project identification and selection.
- It is useful in project initiation, planning and design.
- Components of CASE tools are classified into 3 main categories which are as follows:
  - Upper CASE tools
  - Lower CASE tools
  - Integrated CASE tools

1. **Upper CASE Tools**
   - These are the tools which supports software development from implementation.
   - It mainly focuses on analysis phase.
   - It also focus on design phase.
   - It includes diagramming tools, report and form generators and analysis tools.
2. **Lower CASE Tools**
   - It is useful in supporting implementation and integration tasks.
   - It is useful in database schema generation, program generation, implementation, testing, configuration, etc.
3. **Integrated CASE Tools**
   - It is also called as ICASE.
   - It supports both upper CASE tools and lower CASE tools.
   - If we consider the example of designing the form and building the database to support it at the same time.
   - Different tools are available for creating diagrams, forms, and reports.
   - It is also supporting analysis, reporting, code generation, etc.

15.8 Introduction to Agile Tools

Following are most commonly used toolset for agile processes :

(1) JIRA
(2) Kanban

15.8.1 JIRA

- JIRA is an agile tool used for issue tracking and project management by over 25,000 customers in 122 countries in the world.
- JIRA lets you prioritize, assign, track, report and audit your issues from software bugs and help-desk tickets to project tasks and change requests.
- JIRA is offered in three packages :
  - JIRA Core includes the base software.
  - JIRA Software is intended for use by software development teams and includes JIRA Core and JIRA Agile.

- JIRA Service Desk is intended for use by IT or business service desks.

- The main features of JIRA for agile software development are the functionality to plan development iterations, the iteration reports and the bug tracking functionality.
- JIRA is a commercial software product that can be licensed for running on-premises or available as a hosted application. Pricing depends on the maximum number of users.

JIRA platform

- Every JIRA application (JIRA Software, JIRA Service Desk, and JIRA Core) is built on the JIRA platform. The JIRA platform provides a set of base functionality that is shared across all JIRA applications, like issues, workflows, search, email, and more.
- You can extend and modify this functionality via the integration points provided by the JIRA platform, including the JIRA REST APIs, webhooks, plugin modules, etc
- The pages in this section will help you learn about the JIRA platform and how to develop for it. You'll find information on JIRA's architecture and JIRA add-ons, guides to developing for different parts of JIRA, JIRA Data Centre, and more.

JIRA Architecture

- JIRA is a web application written in Java. It is deployed as a standard Java WAR file into a java Servlet Container such as Tomcat.
- JIRA uses Open Symphony's WebWork 1 to process web requests submitted by users. Please note that WebWork 1, not 2, is used. WebWork 1 is a MVC framework similar to Struts. Each request is handled by a WebWork action which usually uses other objects, such as utility and Manager classes to accomplish a task.
- JIRA uses JSP for the View layer. So most of HTML that is served to the user as the response to their web request is generated by a JSP. Therefore, to generate a response the WebWork action uses a JSP.

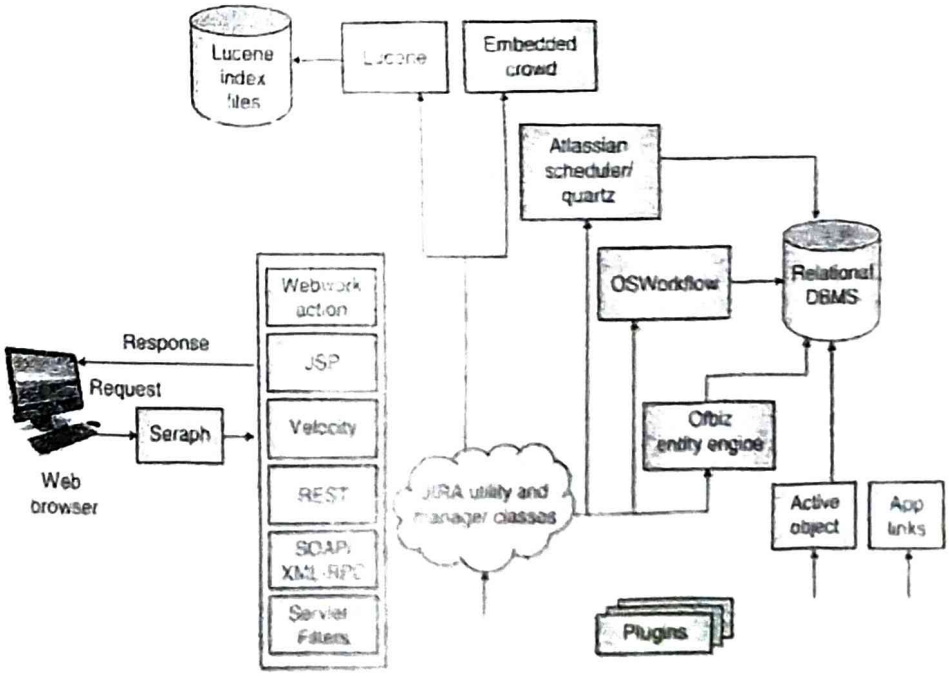


Fig. 15.8.1 : JIRA architecture

JIRA Database Schema

- To generate schema information for the JIRA database, e.g. PDF file, follow the instructions below. You can generate schema information in pdf, txt and dot formats. Note, if you want to generate the schema in PDF format, you need to have Graphviz installed.
  (1) Download the attached plugin:
  - For JIRA 5 : JIRA-schema-diagram-generator-plugin-1.0.jar
  - For JIRA 6 : JIRA-schema-diagram-generator-plugin-1.0.1.jar
  - For JIRA 7 : JIRA-schema-diagram-generator-plugin-1.1.0.jar

  (2) Install the plugin in your JIRA instance by following the instructions on Managing JIRA's Plugins.
  - Go to the JIRA administration console and navigate to System > Troubleshooting and Support > Generate Schema Diagram
  - (tick) Keyboard shortcut: g + g + start typing generate
  - Enter the tables/columns to omit from the generated schema information, if desired.
  - If you want to generate a pdf, enter the path to the Graphviz executable.
  - Click Generate Schema.
  - The 'Database Schema' page will be displayed with links to the schema file in txt, dot and pdf format.

JIRA Mobile Connect

JIRA Mobile Connect (JMC) is an iOS library that can be embedded into any iOS app to provide following extra functionality:

1. **Real-time crash reporting**, have users or testers submit crash reports directly to your JIRA instance.
2. **User or tester feedback** views that allow users or testers to create bug reports within your app.
3. **Rich data input**, users can attach and annotate screenshots, leave a voice message, and have their location sent.
4. **Two-way communication with users**, thank your users or testers for providing feedback on your app.

JIRA Applications

- The JIRA family of applications currently consists of JIRA Software, JIRA Service Desk, and JIRA Core. A JIRA application is built on the JIRA platform, and extends and modifies the base functionality provided by the JIRA platform.
- For example, the JIRA Software application provides software development features that are not part of the JIRA platform, such as agile boards, linked development tool information (e.g. commits, builds, etc), and software project templates.

JIRA APIs

- The JIRA platform provides both Java APIs and REST APIs that you can use to interact with JIRA programmatically. These APIs are common to all JIRA applications.
- In addition, JIRA Software and JIRA Service Desk provide APIs for application-specific functionality. For example, the JIRA Software REST API has methods for creating sprints, creating boards, retrieving epics, etc.

15.8.2 Kanban

- Kanban is a popular framework used by software teams practicing agile software development. It is enormously prominent among today's agile software teams, but the kanban methodology of work dates back more than 50 years.
- In the late 1940s Toyota began optimizing its engineering processes based on the same model that supermarkets were using to stock their shelves.
- Supermarkets stock just enough products to meet consumer demand, a practice that optimizes the flow between the supermarket and the consumer.
- Because inventory levels match consumption patterns, the supermarket gains significant efficiency in inventory management by decreasing the amount of excess stock it must hold at any given team.
- Meanwhile, the supermarket can still ensure that the given product a consumer needs is always in stock.
- Agile software development teams today are able to leverage these same JIT principles by matching the amount of work in progress (WIP) to the team's capacity.
- This gives teams more flexible planning options, faster output, clearer focus, and transparency throughout the development cycle.
- While the core principles of the framework are timeless and applicable to almost any industry, software development teams have found particular success with the agile practice.
- In part, this is because software teams can begin practicing with little to no overhead once they understand the basic principles.
- Unlike implementing kanban on a factory floor, which would involve changes to physical processes and the addition of substantial materials, the only physical things a software teams need are a board and cards, and even those can be virtual.

Kanban Boards

- The work of all kanban teams revolves around a kanban board, a tool used to visualize work and optimize the flow of the work among the team.

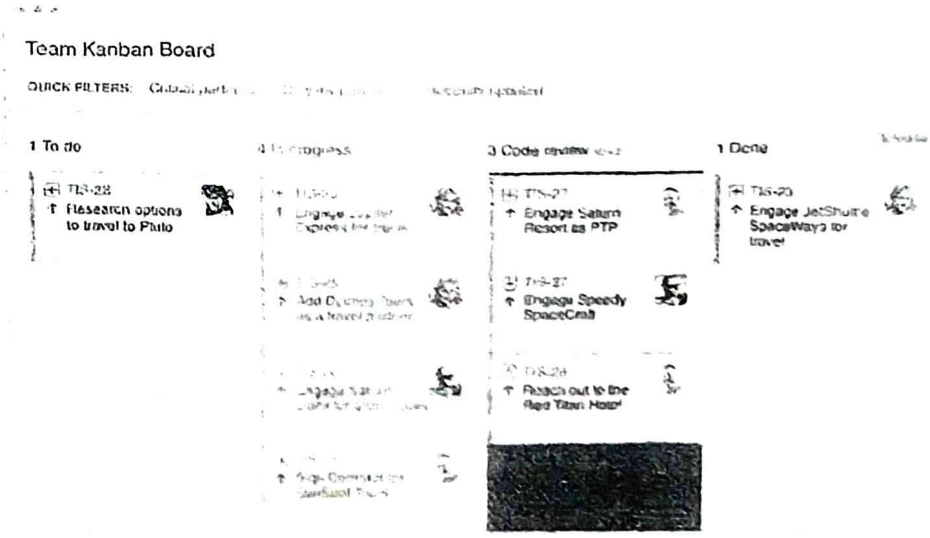


Fig. 15.8.2 : Kanban boards

- While physical boards are popular among some teams, virtual boards are a crucial feature in any agile software development tool for their traceability, easier collaboration, and accessibility from multiple locations.
- Regardless of whether a team's board is physical or digital, their function is to ensure the team's work is visualized, their workflow is standardized, and all blockers and dependencies are immediately identified and resolved.
- A basic kanban board has a three-step workflow: To Do, In Progress, and Done. However, depending on a team's size, structure, and objectives, the workflow can be mapped to meet the unique process of any particular team.
- The kanban methodology relies upon full transparency of work and real-time communication of capacity, therefore the kanban board should be seen as the single source of truth for the team's work.

Kanban Cards

- In Japanese, kanban literally translates to "visual signal." For kanban teams, every work item is represented as a separate card on the board.
- The main purpose of representing work as a card on the kanban board is to allow team members to track the progress of work through its workflow in a highly visual manner.
- Kanban cards feature critical information about that particular work item, giving the entire team full visibility into who is responsible for that item of work, a brief description of the job being done, how long that piece of work is estimated to take, and so on.
- Cards on virtual kanban boards will often also feature screenshots and other technical details that is valuable to the assignee.
- Allowing team members to see the state of every work item at any given point in time, as well as all of the associated details, ensures increased focus, full traceability, and fast identification of blockers and dependencies.

The Benefits of Kanban

- Kanban is one of the most popular software development methodologies adopted by agile teams today. Kanban offers several additional advantages to task planning and throughput for teams of all sizes.
- Planning flexibility : A kanban team is only focused on the work that's actively in progress. Once the team completes a work item, they pluck the next work item off the top of the backlog.
- The product owner is free to reprioritize work in the backlog without disrupting the team, because any changes outside the current work items don't impact the team.
- As long as the product owner keeps the most important work items on top of the backlog, the development team is assured they are delivering maximum value back to the business. So there's no need for the fixed-length iterations you find in scrum.
- Shortened cycle times : Cycle time is a key metric for kanban teams. Cycle time is the amount of time it takes for a unit of work to travel through the team's workflow-from the moment work starts to the moment it ships.
- By optimizing cycle time, the team can confidently forecast the delivery of future work.
- Overlapping skill sets lead to smaller cycle times. When only one person holds a skill set, that person becomes a bottleneck in the workflow. So teams employ basic best practices like code review and mentoring help to spread knowledge.

- Shared skills mean that team members can take on heterogeneous work, which further optimizes cycle time. It also means that if there is a backup of work, the entire team can swarm on it to get the process flowing smoothly again.
- For instance, testing isn't only done by QA engineers. Developers pitch in, too.
- In a kanban framework, it's the entire team's responsibility to ensure work is moving smoothly through the process.
- Fewer bottlenecks : Multitasking kills efficiency. The more work items in flight at any given time, the more context switching, which hinders their path to completion.
- That's why a key tenant of kanban is to limit the amount of work in progress (WIP). Work-in-progress limits highlight bottlenecks and backups in the team's process due to lack of focus, people, or skill sets.
- For example, a typical software team might have four workflow states: To Do, In Progress, Code Review, and Done. They could choose to set a WIP limit of 2 for the code review state.
- That might seem like a low limit, but there's good reason for it: developers often prefer to write new code, rather than spend time reviewing someone else's work.
- A low limit encourages the team to pay special attention to issues in the review state, and to review others work before raising their own code reviews. This ultimately reduces the overall cycle time.
- Visual metrics : One of the core values is a strong focus on continually improving team efficiency and effectiveness with every iteration of work.
- Charts provide a visual mechanism for teams to ensure they're continuing to improve. When the team can see data, it's easier to spot bottlenecks in the process (and remove them).
- Two common reports kanban teams use are control charts and cumulative flow diagrams.
- A control chart shows the cycle time for each issue as well as a rolling average for the team.

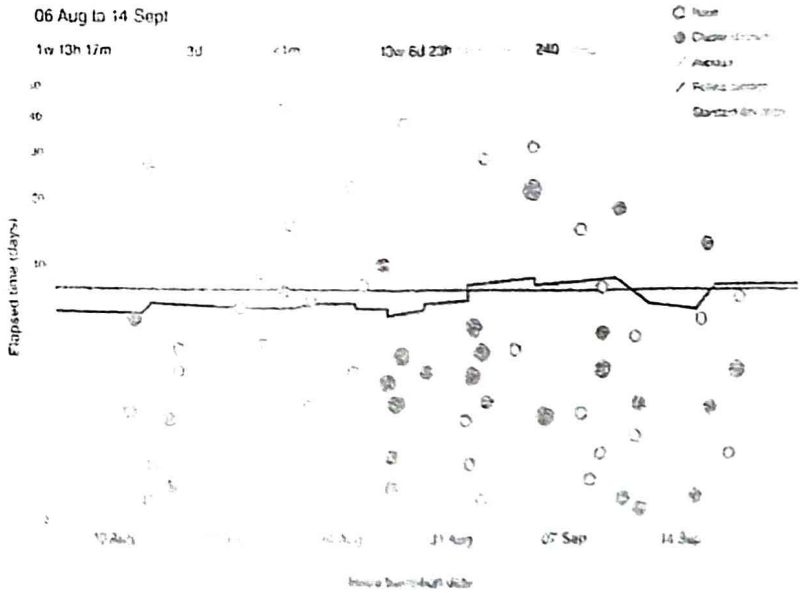


Fig. 15.8.3 : Control chart

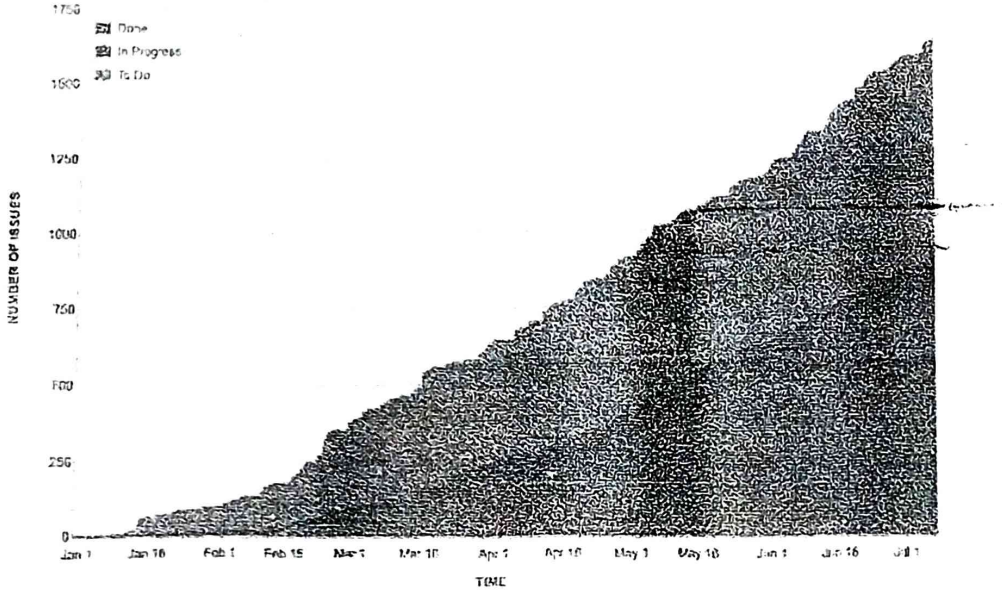


Fig. 15.8.4 : Cumulative flow diagram

- A cumulative flow diagram shows the number of issues in each state. The team can easily spot blockages by seeing the number of issues increase in any given state. Issues in intermediate states such as "In Progress" or "In Review" are not yet shipped to customers, and a blockage in these states can increase the likelihood of massive integration conflicts when the work does get merged upstream.
- Continuous delivery : We know that continuous integration–the practice of automatically building and testing code incrementally throughout the day–is essential for maintaining quality. Now it's time to meet continuous delivery (CD). CD is the practice of releasing work to customers frequently–even daily or hourly. Kanban and CD beautifully complement each other because both techniques focus on the just-in-time (and one-at-a-time) delivery of value.
- The faster a team can deliver innovation to market, the more competitive their product will be in the marketplace. And kanban teams focus on exactly that: optimizing the flow of work out to customers.

Comparison between Kanban and Scrum

Kanban	Scrum
In Kanban, there is continuous flow.	In Scrum approach, we have regular fixed length sprints (i.e. 2 weeks)
Continuous delivery or at the team's discretion.	Delivery at the end of each sprint if approved by the product owner.
Cycle time is the key metrics in Kanban approach.	Velocity is the key metrics.
Changes can happen at any time.	Teams should strive to not make changes to the sprint forecast during the sprint. Doing so compromises learning around estimation.
Roles : No existing roles. Some teams enlist the help of an agile coach.	Roles : Product owner, scrum master, development team etc.

Review Questions

Q. Write a short note on : Software Configuration Management.

Q. Explain risk management process and various types of risks with suitable example

Q. What are the types of risks? Explain in brief.

Q. Explain Software Risk Management.

Q. Write short note on Technology Evolution.

Q. Write short note on Collaborative Development.

Q. What are the risks associated with software projects? How do project managers manage such risks?

Q. Mention the reasons for project delay.

Q. Compare between Kanban and Scrum.

□□□

Model Question Paper (End Sem.)

Software Engineering

Semester IV – Information Technology (Savitribai Phule Pune University)

Time : $2\frac{1}{2}$ Hour | Maximum Marks : 70

Instructions to the candidates :

1) **Answer Q. 1 or Q. 2, Q. 3, or Q. 4. Q.5 or Q.6, Q.7 or Q.8**
2) **Neat diagrams must be drawn wherever necessary.**
3) **Figure to the right indicates full marks.**
4) **Make suitable assumptions if necessary.**

Q. 1 (a) What are the software design quality attributes and quality guidelines? **(Refer Section 7.3)** **(7 Marks)**

(b) Explain the quality attributes, considered in software design? **(Refer Section 7.3.2)** **(6 Marks)**

(c) What do you understand by refactoring? Give the importance of refactoring in improving quality of software. **(Refer Section 7.4.8)** **(5 Marks)**

OR

Q. 2 (a) Discuss architectural patterns in details. (Refer Section 8.4) **(6 Marks)**

(b) Enlist the golden rules for User Interface Design? **(Refer Section 9.2)** **(5 Marks)**

(c) What are the design principles for reducing the user's memory load in user interface design? **(Refer Section 9.2.2)** **(7 Marks)**

Q. 3 (a) Explain the COCOMO-II estimation model? **(Refer Section 10.5.2)** **(4 Marks)**

(b) What are the Importance of Project Schedules? **(Refer Section 10.6)** **(5 Marks)**

(c) What is critical path? Explain with example how critical path method is used to estimate the total project duration. **(Refer Section 10.8.1)** **(8 Marks)**

OR

Q. 4 (a) Explain the role of people, product and process in project management. **(Refer Section 11.1)** **(6 Marks)**

(b) What are the categories of stakeholders? What are the characteristics of effective project manager? **(Refer Section 11.1.1)** **(5 Marks)**

(c) Explain in detail software process and project metrics **(Refer Section 11.3)** **(6 Marks)**

Q. 5 (a) What are the software quality factors? Explain any four. **(Refer Section 13.2)** **(5 Marks)**

(b) Explain different McCall's quality factors. **(Refer Section 13.2.1)** **(6 Marks)**

(c) How will you achieve Software Quality? **(Refer Section13.5)** **(7 Marks)**

OR

Q. 6 (a) State and Explain Software Testing Fundamentals? (Refer Section 14.2) **(6 Marks)**

(b) What are the principles of software testing? (Refer Section 14.3) **(6 Marks)**

(c) Write a short note on : Test Plan. (Refer Section 14.5) **(6 Marks)**

Q. 7 (a) Explain Software Configuration Management (SCM) process. (Refer Section 15.1) **(4 Marks)**

(b) Explain risk management process and various types of risks with suitable example? (Refer Section 15.2) **(7 Marks)**

(c) Write short note on Technology Evolution. (Refer Section 15.3) **(5 Marks)**

OR

Q. 8 (a) Write short note on : Test-driven development? (Refer Section 15.4.2) **(6 Marks)**

(b) Explain in brief global software challenges. (Refer Section 15.4.3) **(6 Marks)**

(c) What are the Benefits of Kanban? (Refer Section15.8.2) **(5 Marks)**

□□□