SUBJECT CODE : CCS356

Strictly as per Revised Syllabus of

ANNA UNIVERSITY

Choice Based Credit System (CBCS)

Semester - VI (CSE / IT / CS&BS)

OBJECT ORIENTED SOFTWARE ENGINEERING

Mrs. Anuradha A. Puntambekar

M.E. (Computer)

Formerly Assistant Professor in

P.E.S. Modern College of Engineering,

Pune

TECHNICAL PUBLICATIONS

SINCE 1993

An Up-Thrust for Knowledge

SYLLABUS

Object Oriented Software Engineering - [CCS356]

UNIT I SOFTWARE PROCESS AND AGILE DEVELOPMENT

Introduction to Software Engineering, Software Process, Perspective and Specialized Process Models - Introduction to Agility - Agile process-Extreme programming - XP Process - Case Study. **(Chapter - 1)**

UNIT II REQUIREMENTS ANALYSIS AND SPECIFICATION

Requirement analysis and specification - Requirements gathering and analysis - Software Requirement Specification - Formal system specification - Finite State Machines - Petrinets - Object modelling using UML - Use case Model - Class diagrams - Interaction diagrams - Activity diagrams - State chart diagrams - Functional modelling - Data Flow Diagram - CASE TOOLS. **(Chapter - 2)**

UNIT III SOFTWARE DESIGN

Software design - Design process - Design concepts - Coupling - Cohesion - Functional independence - Design patterns - Model-view-controller - Publish-subscribe - Adapter - Command - Strategy - Observer - Proxy - Facade - Architectural styles - Layered - Client Server - Tiered - Pipe and filter - User interface design - Case Study. **(Chapter - 3)**

UNIT IV SOFTWARE TESTING AND MAINTENANCE

Testing - Unit testing - Black box testing - White box testing - Integration and System testing - Regression testing - Debugging - Program analysis - Symbolic execution - Model Checking - Case Study. **(Chapter - 4)**

UNIT V PROJECT MANAGEMENT

Software Project Management - Software Configuration Management - Project Scheduling- DevOps : Motivation - Cloud as a platform - Operations - Deployment Pipeline : Overall Architecture Building and Testing - Deployment - Tools - Case Study. **(Chapter - 5)**

TABLE OF CONTENTS

UNIT I

Chapter - 1 Software Process and Agile Development (1 - 1) to (1 - 46)

1.1 Introduction to Software Engineering ..... 1 - 2
1.1.1 Defining Software ..... 1 - 2
1.1.2 Software Characteristics ..... 1 - 2
1.1.3 Categories of Software ..... 1 - 4
1.2 Goals and Objectives of Software ..... 1 - 5
1.3 Difference between Software Product and Program ..... 1 - 6
1.4 Layered Technology ..... 1 - 6
1.5 Software Process ..... 1 - 7
1.5.1 Common Process Framework ..... 1 - 7
1.5.2 Capability Maturity Model (CMM) ..... 1 - 8
1.6 Prescriptive Process Models ..... 1 - 9
1.6.1 Need for Process Model ..... 1 - 10
1.6.2 Waterfall Model ..... 1 - 10
1.6.3 Incremental Process Model ..... 1 - 13
1.6.3.1 Incremental Model ..... 1 - 13
1.6.3.2 RAD Model ..... 1 - 14
1.6.4 Evolutionary Process Model ..... 1 - 16
1.6.4.1 Prototyping ..... 1 - 16
1.6.4.2 Spiral Model ..... 1 - 18
1.6.4.3 Concurrent Development Model ..... 1 - 21
1.7 Specialized Model ..... 1 - 28
1.7.1 Component based Development ..... 1 - 28
1.7.2 Formal Methods Model ..... 1 - 29
1.7.3 Aspect Oriented Software Development ..... 1 - 29
1.8 Introduction to Agility ..... 1 - 32
1.9 Agile Process ..... 1 - 32
1.9.1 Agile Principles ..... 1 - 34
1.10 Extreme Programming ..... 1 - 35
1.10.1 XP Values ..... 1 - 35

1.10.2 Process ..... 1 - 36
1.10.3 Industrial XP ..... 1 - 39
1.11 Two Marks Questions with Answers ..... 1 - 40

UNIT II

Chapter - 2 Requirements Analysis and Specification (2 - 1) to (2 - 178)

2.1 Introduction to Software Requirements ..... 2 - 3
2.2 Functional and Non Functional Requirements ..... 2 - 4
2.2.1 Functional Requirements ..... 2 - 4
2.2.1.1 Problems Associated with Requirements ..... 2 - 4
2.2.2 Non Functional Requirements ..... 2 - 5
2.2.2.1 Types of Non Functional Requirements ..... 2 - 5
2.2.2.2 Domain Requirements ..... 2 - 7
2.2.3 Difference between Functional and Non Functional Requirements ..... 2 - 7
2.3 Requirements Engineering Process ..... 2 - 12
2.4 Feasibility Studies ..... 2 - 15
2.5 Requirement Gathering and Analysis ..... 2 - 16
2.5.1 Stakeholders ..... 2 - 17
2.5.2 Requirement Elicitation and Analysis Process ..... 2 - 17
2.5.3 Requirement Discovery ..... 2 - 18
2.6 Software Requirement Specification ..... 2 - 26
2.6.1 Characteristics of SRS ..... 2 - 29
2.6.2 Example of SRS ..... 2 - 30
2.7 Formal System Specification ..... 2 - 44
2.7.1 Concept of Formal Technique ..... 2 - 44
2.7.2 Merits and Limitations of Formal Methods ..... 2 - 45
2.7.3 Model-Oriented Approach and Property-Oriented Approach ..... 2 - 46
2.7.4 Axiomatic Specification ..... 2 - 47
2.7.5 Algebraic Specification ..... 2 - 48
2.8 Finite State Machines ..... 2 - 50
2.8.1 Formal Definition ..... 2 - 51
2.8.2 Representation ..... 2 - 51
2.8.3 Types of Finite State Machines ..... 2 - 52
2.8.4 Applications of Finite State Machine ..... 2 - 52

2.8.5 Deterministic Finite State Machine ..... 2 - 52
2.8.6 Non Deterministic Finite State Automata(NFA) ..... 2 - 54
2.9 Petri Nets ..... 2 - 55
2.10 Object Modeling using UML ..... 2 - 58
2.10.1 What is UML ? ..... 2 - 59
2.10.2 UML Diagrams ..... 2 - 61
2.11 Use Case Model ..... 2 - 62
2.11.1 Actors, Scenarios and Use Cases ..... 2 - 64
2.11.2 Use Cases and Use Case Model ..... 2 - 65
2.11.3 Relationships in Use Cases ..... 2 - 67
2.11.3.1 Include ..... 2 - 68
2.11.3.2 Extend ..... 2 - 69
2.11.3.3 Generalization ..... 2 - 70
2.11.4 When to Use Use Cases ? ..... 2 - 71
2.12 Class Diagrams ..... 2 - 72
2.12.1 Attributes ..... 2 - 74
2.12.2 Relationships ..... 2 - 74
2.12.2.1 Associations ..... 2 - 75
2.12.2.2 Inheritance ..... 2 - 77
2.12.3 Aggregation and Composition ..... 2 - 78
2.12.4 Visibility ..... 2 - 79
2.12.5 Qualified Association ..... 2 - 80
2.12.6 Association Class ..... 2 - 80
2.12.7 Singleton Class ..... 2 - 81
2.12.8 Active Class ..... 2 - 81
2.12.9 When to use Class Diagrams ? ..... 2 - 82
2.12.10 Examples on Class Diagram ..... 2 - 83
2.13 Interaction Diagrams ..... 2 - 95
2.13.1 System Sequence Diagram ..... 2 - 95
2.13.2 Relationship between Sequence Diagram and Use Cases ..... 2 - 97
2.13.3 Collaboration Diagram ..... 2 - 100
2.13.3.1 Objects and Links ..... 2 - 100
2.13.3.2 Messages and Stimulus ..... 2 - 101
2.13.3.3 Active Objects ..... 2 - 102
2.13.3.4 Communication Diagram ..... 2 - 103
2.13.4 Strengths and Weakness of Sequence and Communication Diagrams ..... 2 - 104

2.13.5 Examples on Interaction Diagram ..... 2 - 104
2.13.6 When to use Sequence Diagram ? ..... 2 - 114
2.13.7 When to use Communication Diagram ? ..... 2 - 115
2.14 Activity Diagrams ..... 2 - 115
2.14.1 Activity and Actions ..... 2 - 116
2.14.2 Initial and Final Activity ..... 2 - 116
2.14.3 Activity Edge ..... 2 - 117
2.14.3.1 Object Flows ..... 2 - 117
2.14.4 Decision and Merge Points ..... 2 - 117
2.14.5 Fork and Join ..... 2 - 118
2.14.6 Activity Partitions or Swimlane ..... 2 - 119
2.14.7 Object Flow ..... 2 - 119
2.14.8 When to use Activity Diagrams ? ..... 2 - 121
2.14.9 Examples on Activity Diagram ..... 2 - 122
2.15 State Chart Diagrams ..... 2 - 132
2.15.1 Events, States and Transitions ..... 2 - 132
2.15.2 Applying State Machine Diagram ..... 2 - 133
2.15.3 More Notations ..... 2 - 134
2.15.4 When to use State Diagram ? ..... 2 - 141
2.16 Functional Modelling ..... 2 - 141
2.17 Data Flow Diagram ..... 2 - 143
2.17.1 Rules for Designing DFD ..... 2 - 143
2.17.3 Difference between Structured Analysis and Object Oriented Analysis ... 2 - 173
2.18 Two Marks Questions with Answers ..... 2 - 173

UNIT III

Chapter - 3 Software Design (3 - 1) to (3 - 68)

3.1 Software Design ..... 3 - 2
3.1.1 Designing within the Context of Software Engineering ..... 3 - 2
3.2 Design Process ..... 3 - 3
3.3 Design Concepts ..... 3 - 5
3.3.1 Abstraction ..... 3 - 5
3.3.2 Modularity ..... 3 - 6
3.3.3 Architecture ..... 3 - 8

3.3.4 Refinement ..... 3 - 8
3.3.5 Pattern ..... 3 - 8
3.3.6 Information Hiding ..... 3 - 9
3.3.7 Functional Independence ..... 3 - 9
3.3.7.1 Cohesion ..... 3 - 9
3.3.7.2 Coupling ..... 3 - 10
3.3.8 Refactoring ..... 3 - 11
3.3.9 Design Classes ..... 3 - 11
3.4 Design Patterns ..... 3 - 13
3.4.1 Creational Pattern ..... 3 - 14
3.4.2 Structural Pattern ..... 3 - 15
3.4.3 Behavioural Pattern ..... 3 - 16
3.5 Model-View-Controller ..... 3 - 17
3.6 Publisher-Subscriber Pattern ..... 3 - 19
3.7 Adapter ..... 3 - 20
3.8 Command ..... 3 - 22
3.9 Strategy ..... 3 - 27
3.10 Observer ..... 3 - 31
3.11 Proxy ..... 3 - 32
3.12 Façade ..... 3 - 35
3.13 Role of Architecture ..... 3 - 37
3.14 Architectural Styles ..... 3 - 39
3.15 Layered Architectural Style ..... 3 - 39
3.16 Client Server Style ..... 3 - 40
3.17 Tiered Architecture ..... 3 - 41
3.18 Pipe and Filter ..... 3 - 41
3.19 User Interface Design ..... 3 - 43
3.19.1 Golden Rules ..... 3 - 43
3.19.1.1 Place the User in Control ..... 3 - 44
3.19.1.2 Reduce the User's Memory Load ..... 3 - 45
3.19.1.3 Make the Interface Consistent ..... 3 - 45
3.19.2 User Interface Analysis and Design ..... 3 - 46
3.19.2.1 Interface Analysis and Design Model ..... 3 - 47
3.19.2.2 The Process ..... 3 - 48
3.19.3 Interface Analysis ..... 3 - 49

3.19.3.1 User Analysis ..... 3 - 49
3.19.3.2 Task Analysis and Modelling ..... 3 - 50
3.19.3.3 Analysis of Display Content ..... 3 - 53
3.19.3.4 Analysis of Work Environment ..... 3 - 54
3.19.4 Interface Design ..... 3 - 54
3.19.4.1 Application of Interface Design Steps ..... 3 - 55
3.19.4.2 User Interface Design Pattern ..... 3 - 58
3.19.4.3 Design Issues ..... 3 - 58
3.20 Two Marks Questions with Answers ..... 3 - 61

UNIT IV

Chapter - 4 Software Testing and Maintenance (4 - 1) to (4 - 62)

4.1 Introduction to Testing ..... 4 - 2
4.1.1 Testing Objectives ..... 4 - 2
4.1.2 Testing Principles ..... 4 - 2
4.1.3 Why Testing is Important ? ..... 4 - 2
4.2 Internal and External Views of Testing ..... 4 - 5
4.3 Black Box Testing ..... 4 - 5
4.3.1 Equivalence Partitioning ..... 4 - 6
4.3.2 Boundary Value Analysis (BVA) ..... 4 - 7
4.4 White Box Testing ..... 4 - 13
4.4.1 Condition Testing ..... 4 - 13
4.4.2 Loop Testing ..... 4 - 14
4.4.3 Basis Path Testing ..... 4 - 16
4.5 Testing Strategies ..... 4 - 34
4.6 Unit Testing ..... 4 - 36
4.7 Integration Testing ..... 4 - 37
4.7.1 Top Down Integration Testing ..... 4 - 39
4.7.2 Bottom Up Integration Testing ..... 4 - 40
4.7.3 Regression Testing ..... 4 - 41
4.7.4 Smoke Testing ..... 4 - 41
4.8 System Testing ..... 4 - 42
4.8.1 Recovery Testing ..... 4 - 43
4.8.2 Security Testing ..... 4 - 43
4.8.3 Stress Testing ..... 4 - 43

4.8.4 Performance Testing ..... 4 - 43
4.9 Validation Testing ..... 4 - 44
4.9.1 Acceptance Testing ..... 4 - 44
4.10 Debugging Techniques ..... 4 - 46
4.10.1 Testing Vs. Debugging ..... 4 - 47
4.11 Program Analysis ..... 4 - 48
4.12 Symbolic Execution ..... 4 - 50
4.13 Model Checking ..... 4 - 51
4.14 Two Marks Questions with Answers ..... 4 - 52

UNIT V

Chapter - 5 Project Management **(5 - 1) to (5 - 88)**
5.1 Software Project Management ..... 5 - 2
5.2 Management Spectrum ..... 5 - 2
5.2.1 The People ..... 5 - 2
5.2.2 The Product ..... 5 - 3
5.2.3 The Process ..... 5 - 3
5.2.4 The Project ..... 5 - 3
5.3 People ..... 5 - 4
5.3.1 Stakeholder ..... 5 - 4
5.3.2 Team Leaders ..... 5 - 5
5.3.3 Software Team ..... 5 - 5
5.3.4 Agile Team ..... 5 - 6
5.3.5 Co-ordination and Communication Issues ..... 5 - 7
5.4 Product ..... 5 - 8
5.4.1 Software Scope ..... 5 - 8
5.4.2 Problem Decomposition ..... 5 - 8
5.5 Process ..... 5 - 9
5.5.1 Melding the Product and the Process ..... 5 - 9
5.5.2 Process Decomposition ..... 5 - 10
5.6 Project ..... 5 - 11
5.7 The W5HH Principle ..... 5 - 12
5.8 Project Estimation ..... 5 - 13

5.8.1 Software Sizing ..... 5 - 13
5.8.2 Problem based Estimation ..... 5 - 14
5.8.3 LOC based Estimation ..... 5 - 15
5.8.4 Example of LOC based Estimation ..... 5 - 16
5.8.5 Function Oriented Metrics ..... 5 - 19
5.8.6 Example of FP based Estimation ..... 5 - 21
5.9 Make Buy Decision ..... 5 - 28
5.9.1 Outsourcing ..... 5 - 30
5.10 COCOMO I Model ..... 5 - 31
5.11 COCOMO II Model ..... 5 - 36
5.12 Software Configuration Management ..... 5 - 44
5.12.1 SCM Basics ..... 5 - 45
5.12.1.1 Goals in Change Management ..... 5 - 45
5.12.1.2 Elements of Configuration Management System ..... 5 - 45
5.12.1.3 Baselines ..... 5 - 46
5.12.1.4 Software Configuration Items ..... 5 - 47
5.12.2 SCM Repository ..... 5 - 47
5.12.2.1 Role of Project Repository ..... 5 - 48
5.12.2.2 Features ..... 5 - 48
5.12.3 SCM Process ..... 5 - 49
5.12.3.1 Identification of Objects in Software Configuration ..... 5 - 49
5.12.3.2 Change Control ..... 5 - 50
5.12.3.3 Version Control ..... 5 - 52
5.12.3.4 Configuration Audit ..... 5 - 53
5.12.3.5 Status Reporting ..... 5 - 54
5.13 Project Scheduling ..... 5 - 54
5.13.1 Relationship between People and Effort ..... 5 - 55
5.13.2 Task Sets ..... 5 - 56
5.13.3 Task Network ..... 5 - 59
5.13.4 Time Line Chart ..... 5 - 60
5.13.5 Tracking Schedule ..... 5 - 61
5.13.6 Earned Value Analysis ..... 5 - 62
5.14 DevOps ..... 5 - 71
5.14.1 Why DevOps ? ..... 5 - 71
5.14.2 Motivation ..... 5 - 72
5.14.3 Benefits ..... 5 - 72
5.14.4 Agility and DevOps ..... 5 - 73

5.14.5 Deployment Pipeline ..... 5 - 74
5.14.6 Overall Architecture ..... 5 - 77
5.14.7 DevOps Lifecycle ..... 5 - 78
5.14.8 Tools ..... 5 - 78
5.15 Cloud as a Platform ..... 5 - 79
5.15.1 Operations ..... 5 - 81
5.16 Two Marks Questions with Answers ..... 5 - 82

Solved Model Question Paper **(M - 1) to (M - 4)**

UNIT I

1 Software Process and Agile Development

Syllabus

Introduction to software engineering, Software process, Perspective and specialized process models, Introduction to Agility, Agile process, Extreme programming, XP process.

Contents

1.1 Introduction to Software Engineering ..... **Dec.-13, 17,** ..... Marks 4
1.2 Goals and Objectives of Software
1.3 Difference between Software Product and Program
1.4 Layered Technology ..... **May-22,** ..... Marks 7
1.5 Software Process ..... **Dec.-10,17,22,** ..... Marks 7
1.6 Prescriptive Process Models **May-05,06,09,15,16,17,18,22,**
..... **Dec.-06,09,11,16,17,19,**
..... **Dec.-20,22,** ..... Marks 16
1.7 Specialized Model **Dec.-13,14,15,17,**
..... **May-16,19,** ..... Marks 16
1.8 Introduction to Agility
1.9 Agile Process ..... **Dec.-16,19, May-19,** ..... Marks 4
1.10 Extreme Programming ..... **May-19,22, Dec.-20,22,** ..... Marks 13
1.11 Two Marks Questions with Answers

1.1 Introduction to Software Engineering

AU : Dec.-13, 17, Marks 4

*"Software engineering is a **discipline** in which **theories, methods** and **tools** are applied to develop professional software **product**."*

In software engineering the **systematic** and **organized approach** is adopted. Based on the nature of the problem and development constraints various **tools and techniques** are applied in order to develop **quality software**.

The definition of software engineering is based on two terms :

- **Discipline :** For finding the solution to the problem an Engineer applies appropriate theories, methods and tools. While finding the solutions, Engineers must think of the organizational and financial constraints. Within these constraints only, he/she has to find the solution.
- **Product :** The software product gets developed after following systematic theories, methods and tools along with the appropriate management activities.

1.1.1 Defining Software

Software is nothing but a collection of computer programs and related documents that are intended to provide desired features, functionalities and better performance.

Software products may be :

1. **Generic** - That means developed to be sold to a range of different customers.
2. **Custom** - That means developed for a single customer according to their specification.

1.1.2 Software Characteristics

Software development is a logical activity and therefore it is important to understand basic characteristics of software. Some important characteristics of software are :

- **Software is engineered, not manufactured**
  - Software development and hardware development are two different activities.
  - A good design is a backbone for both the activities.
  - Quality problems that occur in hardware manufacturing phase can not be removed easily. On the other hand, during software development process such problems can be rectified.
  - In both the activities, developers are responsible for producing qualitative product.

- **Software does not ware out**
  - In early stage of hardware development process the failure rate is very high because of manufacturing defects. But after correcting such defects the failure rate gets reduced.
  - The failure rate remains constant for some period of time and again it starts increasing because of environmental maladies (extreme temperature, dusts and vibrations).
  - On the other hand software does not get affected from such environmental maladies. Hence ideally it should have an *"idealized curve"*. But due to some **undiscovered errors** the failure rate is high and drops down as soon as the errors get corrected.
  - Hence in failure rating of software the *"actual curve"* is as shown below :

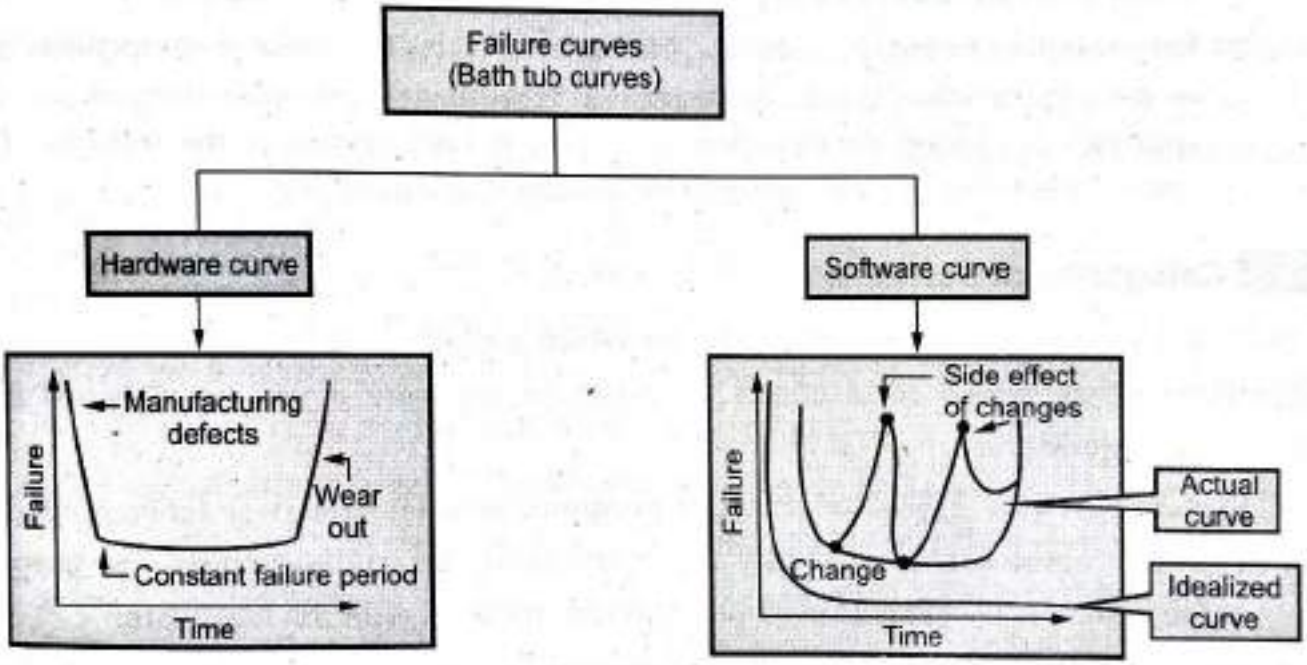


Fig. 1.1.1 Failure curves for hardware and software

  - During the life of software if any change is made, some **defects** may get **introduced**.
  - This causes failure rate to be high. Before the curve can return to original steady state another change is requested and again the failure rate becomes high.
  - Thus the failure curve looks like a spike. Thus frequent changes in software cause it to deteriorate.
  - Another issue with software is that there are ***no spare parts for software***.
  - If hardware component wears out it can be replaced by another component but it is not possible in case of software.
  - Therefore **software maintenance** is **more difficult** than the hardware maintenance.

- **Most software is custom built rather than being assembled from components**
  - While developing any hardware product firstly the circuit design with desired functioning properties is created.
  - Then required hardware components such as ICs, capacitors and registers are assembled according to the design, but this is not done while developing software product.
  - Most of the software is custom built.
  - However, now the software development approach is getting changed and we look for reusability of software components.
  - It is practiced to reuse algorithms and data structures.
  - Today software industry is trying to make library of reusable components.
  - **For example :** In today's software, GUI is built using the reusable components such as message windows, pull down menus and many more such components. The approach is getting developed to use in-built components in the software. This stream of software is popularly known as *component engineering*.

1.1.3 Categories of Software

Software can be applied in a situation for which a predefined set of procedural steps (algorithm) exists. Based on a complex growth of software it can be classified into following categories.

- **System software** - It is collection of programs written to service other programs. Typical programs in this category are compiler, editors and assemblers. The purpose of the system software is to establish a communication with the hardware.
- **Application software** - It consists of standalone programs that are developed for specific business need. This software may be supported by database systems.
- **Engineering/Scientific software** - This software category has a wide range of programs from astronomy to volcanology, from automative stress analysis to space shuttle orbital dynamics and from molecular biology to automated manufacturing. This software is based on complex numeric computations.
- **Embedded software** - This category consists of program that can reside within a product or system. Such software can be used to implement and control features and functions for the end-user and for the system itself.
- **Web applications** - Web application software consists of various web pages that can be retrieved by a browser. The web pages can be developed using programming languages like JAVA, PERL, CGI, HTML, DHTML.

- **Artificial Intelligence software** - This kind of software is based on knowledge based expert systems. Typically, this software is useful in robotics, expert systems, image and voice recognition, artificial neural networks, theorem proving and game playing.

Review Questions

1. *What is the impact of reusability in software development process?* **AU : Dec.-17, Marks 4**
2. *Write a note on the unique characters of a software.* **AU : Dec.-17, Marks 3**

1.2 Goals and Objectives of Software

While developing software following are common objectives.

1. **Satisfy users requirements** - Many programmers simply don't do what the end user wants because they do not understand user requirements. Hence it becomes necessary to understand the demand of end user and accordingly software should be developed.
2. **High reliability** - Mistakes or bugs in a program can be expensive in terms of human lives, money, and customer relation. For instance, Microsoft has faced many problems because earlier release of windows has many problems. Thus software should be delivered only if high reliability is achieved.
3. **Low maintenance costs** - Maintenance of software is an activity that can be done only after delivering the software to the customer. Any small change in software should not cause restructuring of whole software. This indicates that the design of software has poor quality.
4. **Delivery on time** - It is very difficult to predict the exact time on which the software can be completed. But a systematic development of software can lead to meet the given deadline.
5. **Low production costs** - The software product should be cost effective.
6. **High performance** - The high performance software are expected to achieve optimization in speed and memory usage.
7. **Ease of reuse** - Use same software in different systems and software.

Environments reduce development costs and also improve the reliability. Hence reusability of developed software is an important property.

1.3 Difference between Software Product and Program

Sr. No.	Program	Software product
1.	Programs are developed by individual user and it is used for personal use.	Software product is developed by multiple users and it is used by large number of people or customers.
2.	Programs are small in size and possessing limited functionality.	Software product consists of multiple program codes, related documents such as SRS, design documents user manuals, test cases and so on.
3.	Generally only one person uses the program, hence there is a lack of user interface.	Good graphical user interface is most required by any software product.
4.	Program is generally developed by **programmer**.	Software product is developed by **software engineers** who are large in number and work in a team. Therefore systematic approach of developing software product must be applied.
5.	**For example :** Program of sorting n elements.	**For example :** A word processing software.

1.4 Layered Technology

AU : May-22, Marks 7

- Software engineering is a layered technology. Any software can be developed using these layered approaches. Various layers on which the technology is based are **quality focus layer, process layer, methods layer, tools layer**.
- A disciplined **quality management** is a backbone of software engineering technology.
- **Process layer** is a foundation of software engineering. Basically, process defines the framework for timely delivery of software.

Fig. 1.4.1 Layered technology

- In **method layer** the actual method of implementation is carried out with the help of requirement analysis, designing, coding using desired programming constructs and testing.
- Software **tools** are used to bring automation in software development process.
- Thus software engineering is a **combination** of process, methods, and tools for development of quality software.

Review Question

1. *Software engineering is a layered approach. Justify.* **AU : May-22, Marks 7**

1.5 Software Process

AU : Dec.-10,17,22, Marks 7

Software process can be defined as the structured set of activities that are required to develop the software system.

The fundamental activities are :

- Specification
- Design and implementation
- Validation
- Evolution

A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

1.5.1 Common Process Framework

The process framework is required for representing the common process activities.

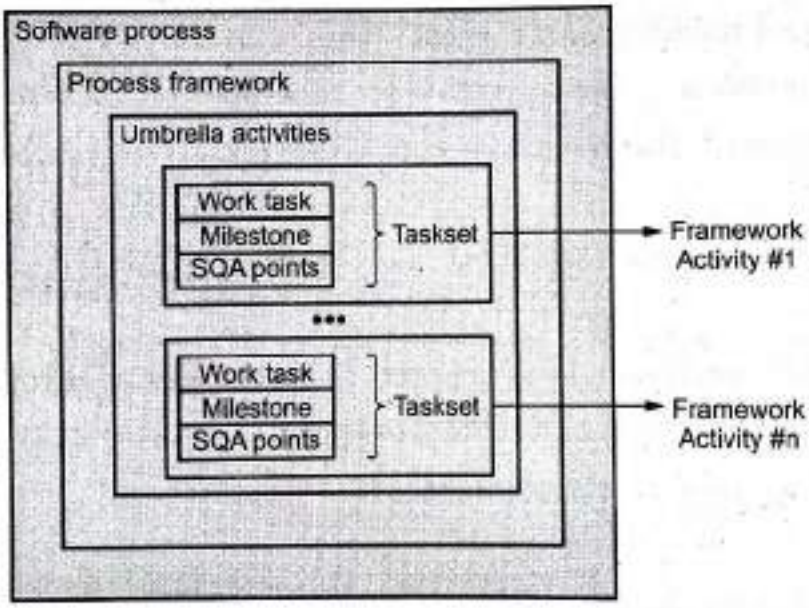


Fig. 1.5.1 Software process framework

As shown in Fig. 1.5.1, the software process is characterized by process framework activities, task sets and umbrella activities.

Process framework activities

- Communication
  - By communicating customer requirement gathering is done.
- Planning - Establishes engineering work plan, describes technical risks, lists resource requirements, work products produced and defines work schedule.
- Modeling - The software model is prepared by :
  - Analysis of requirements
  - Design

- Construction - The software design is mapped into a code by :
  - Code generation
  - Testing
- Deployment - The software delivered for customer evaluation and feedback is obtained.

Task sets - The task set defines the actual work done in order to achieve the software objective. The task set is used to adopt the framework activities and project team requirements using :

- Collection of software engineering work tasks
- Project milestones
- Software quality assurance points

Umbrella activities - The umbrella activities occur **throughout the process**. They focus on project management, tracking and control. The umbrella activities are

1. **Software project tracking and control** - This is an activity in which software team can **assess progress** and take corrective action to **maintain schedule**.
2. **Risk management** - The **risks** that may affect project outcomes or quality can be **analyzed**.
3. **Software quality assurance** - These are activities required to **maintain** software **quality**.
4. **Formal technical reviews** - It is required to **assess engineering work products to** uncover and **remove errors** before they propagate to next activity.
5. **Software configuration management** - Managing of configuration process when any **change** in the software occurs.
6. **Work product preparation and production -** The activities to create models, documents, logs, forms and lists are carried out.
7. **Reusability management -** It defines criteria for work product reuse.
8. **Measurement** - In this activity, the process can be defined and collected. Also **project and product measures** are used to assist the software team in delivering the required software.

1.5.2 Capability Maturity Model (CMM)

- The Software Engineering Institute (SEI) has developed a comprehensive process meta-model emphasizing **process maturity**. It is predicated on a **set of system and software capabilities** that should be present when organizations reach different levels of process capability and maturity.

- The Capability Maturity Model (CMM) is **used** in assessing how well an organization's processes allow **to complete and manage** new software projects.
- Various process maturity levels are :

Level 1 : Initial - Few processes are defined and individual efforts are taken.

Level 2 : Repeatable - To track cost schedule and functionality basic project management processes are established. Depending on earlier successes of projects with similar applications necessary process discipline can be repeated.

Level 3 : Defined - The process is standardized, documented and followed. All the projects use documented and approved version of software process which is useful in developing and supporting software.

Level 4 : Managed - Both the software process and product are quantitatively understood and controlled using detailed measures.

Level 5 : Optimizing - Establish mechanisms to plan and implement change. Innovative ideas and technologies can be tested.

Thus CMM is used for improving the software project.

Review Questions

1. *Explain the CMMI model to access the organization level.* **AU : Dec.-17, Marks 7**
2. *Discuss in brief about the typical activities involved under umbrella activities in software engineering.* **AU : Dec.-22, Marks 7**

1.6 Prescriptive Process Models

AU : May-05,06,09,15,16,17,18,22, Dec.-06,09,11,16,17,19,20,22, Marks 16

- **Definition of Process Model :** The process model can be defined as the **abstract representation of process**. The appropriate process model can be chosen based on abstract representation of process.
- The software process model is also known as **Software Development Life Cycle (SDLC) Model** or software paradigm.
- These models are called **prescriptive process models** because they are following some rules for correct usage.
- In this model various activities are carried out in some specific sequence to make the desired software product.
- Various prescriptive process models are -

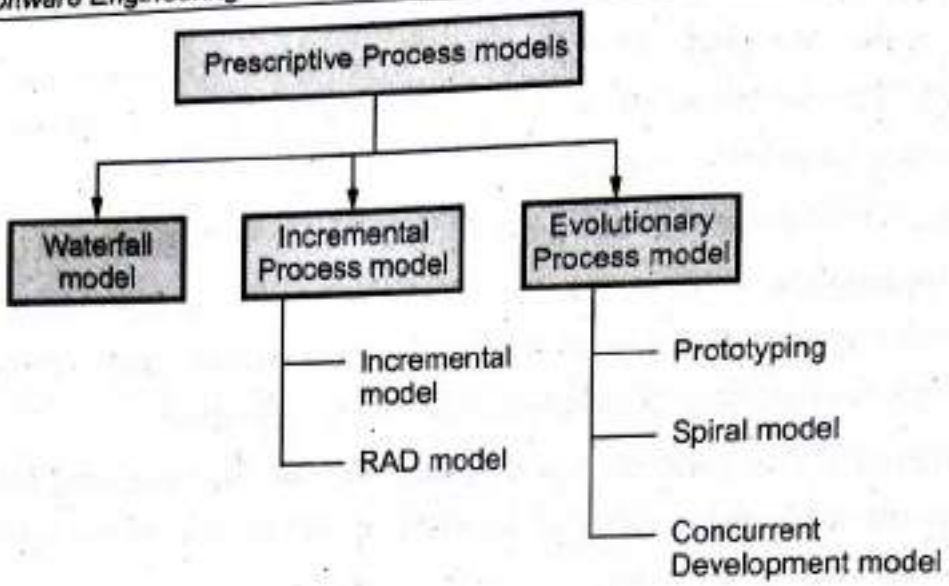


Fig. 1.6.1 Prescriptive process model

1.6.1 Need for Process Model

The software development team must decide the process model that is to be used for software product development and then the entire team must adhere to it. This is necessary because the software product development can then be done systematically. Each team member will **understand - what is the next activity** and **how** to do it. Thus **process model** will bring the **definiteness and discipline** in overall development process.

Every process model consists of **definite entry and exit criteria** for **each phase**. Hence the transition of the product through various phases is definite. If the process model is not followed for software development then any team member can perform any software development activity, this will ultimately cause a chaos and software project will definitely fail without using process model, it is difficult to monitor the progress of software product. Let us discuss various process models one by one.

1.6.2 Waterfall Model

- The waterfall model is also called as '**linear-sequential model**' or '**classic life cycle model**'. It is the oldest software paradigm. This model suggests a systematic, **sequential approach** to software development.
- The software development starts with **requirements gathering phase**. Then progresses through **analysis**, **design**, **coding**, **testing** and **maintenance**. Following figure illustrates waterfall model.
- In **requirement gathering and analysis** phase the **basic requirements** of the system must be understood by software engineer, who is also called **Analyst**. The **information domain**, **function**, **behavioural requirements** of the system are

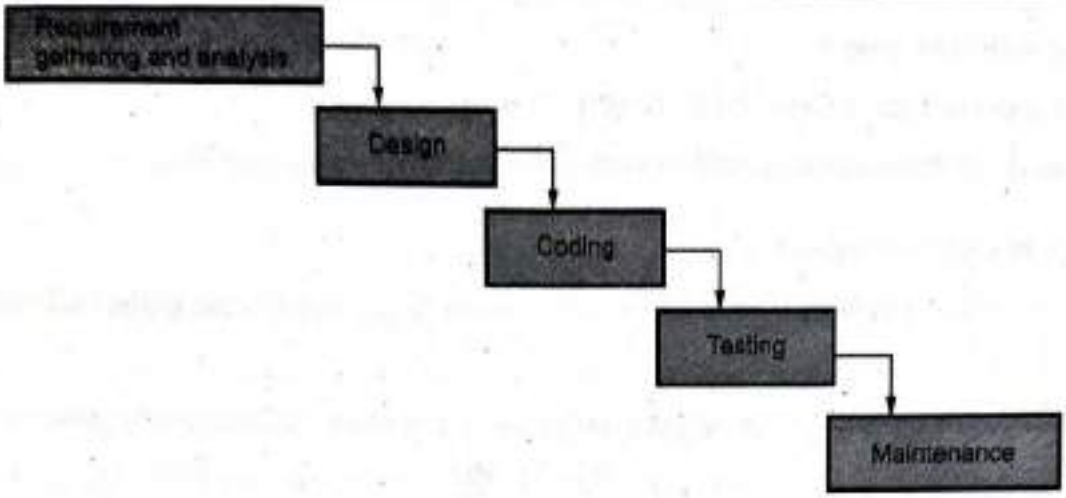


Fig. 1.6.2 Waterfall model

understood. All these requirements are then **well documented** and discussed further with the customer, for reviewing.

- The **design** is an intermediate **step between** requirements analysis and coding.

  Design focuses on program attributes such as -

  - Data structure
  - Software architecture
  - Interface representation
  - Algorithmic details.

  The requirements are translated in some easy to represent form using which coding can be done effectively and efficiently. The design needs to be documented for further use.

- **Coding** is a step in which design is translated into **machine-readable form**. If design is done in sufficient detail then coding can be done effectively. **Programs** are created in this phase.
- **Testing** begins when coding is done. While performing testing the major focus is on **logical internals of the software**. The testing ensures execution of all the paths, functional behaviours. The purpose of testing is to **uncover errors, fix the bugs** and **meet the customer requirements**.
- **Maintenance** is the **longest life cycle phase**. When the system is installed and put in practical use then error may get introduced, correcting such errors and **putting it in use** is the **major purpose** of maintenance activity. Similarly, **enhancing system's services** as new requirements are discovered is again maintenance of the system.

This model is widely used model, although it has many drawbacks. Let us discuss benefits and drawbacks.

Benefits of waterfall model

1. The waterfall model is **simple to implement.**
2. For implementation of **small systems** waterfall model is useful.

Drawbacks of waterfall model

There are some **problems** that are encountered if we apply the **waterfall model** and those are :

1. It is difficult to follow the sequential flow in software development process. If some changes are made at some phases then it may cause some confusion.
2. The requirement analysis is done initially and sometimes it is not possible to state all the requirements explicitly in the beginning. This causes difficulty in the project.
3. The customer can see the **working model** of the project only **at the end**. After reviewing of the working model; if the customer gets dissatisfied then it causes serious problems.
4. Linear nature of waterfall model induces **blocking states**, because certain tasks may be dependant on some previous tasks. Hence it is necessary to accomplish all the dependant tasks first. It may cause long waiting time.

Example 1.6.1 *Explain how water-fall model is applicable for the development of the following systems :*

a) University accounting system

b) Interactive system that allows railway passengers to find time and other information from the terminals installed in the station.

Solution :

a) University accounting system : If the software developers who have the experience in developing the account systems then building university account system based on existing design could be easily managed with water-fall model.

b) Interactive system that allows railway passengers to find time and other information from the terminals installed in the station.

For developing such interactive system, all the requirements must be correctly identified and analyzed before the designing of the project. The requirements of end-users must be correctly and un-ambiguously understood by the developers prior to design phase. Once the requirements are well defined then using disciplined design, coding and testing phases the required system can be built using water-fall model.

1.6.3 Incremental Process Model

In this model, the initial model with limited functionality is created for user's understanding about the software product and then this model is refined and expanded in later releases.

1.6.3.1 Incremental Model

- The incremental model has same phases that are in waterfall model. But it is iterative in nature. The incremental model has following phases.

  1. Analysis 2. Design
  3. Code 4. Test

- The incremental model delivers series of releases to the customer. These releases are called **increments**. More and more functionality is associated with each increment.

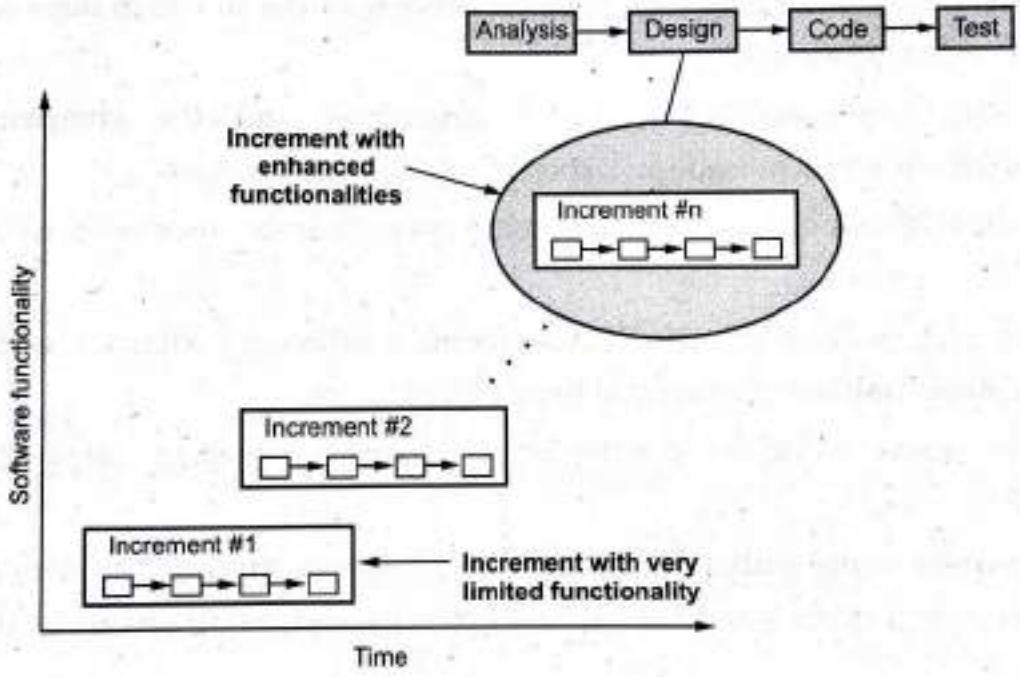


Fig. 1.6.3 The incremental model

- The first increment is called **core product**. In this release the basic requirements are implemented and then in subsequent increments new requirements are added.
- The word processing software package can be considered as an example of incremental model. In the first increment only the document processing facilities are available. In the second increment, more sophisticated document producing and processing facilities, file management functionalities are given. In the next increment spelling and grammar checking facilities can be given. Thus in incremental model progressive functionalities are obtained with each release.

When to choose it ?

1. When requirements are reasonably well-defined.
2. When overall scope of the development effort suggests a purely linear effort.
3. When limited set of software functionality needed quickly.

Merits of incremental model

1. The incremental model can be adopted when there are less number of people involved in the project.
2. Technical risks can be managed with each increment.
3. For a very small time span, atleast core product can be delivered to the customer.

1.6.3.2 RAD Model

- The RAD Model is a type of incremental process model in which there is **extremely short development cycle.**
- When the requirements are fully understood and the **component based construction** approach is adopted then the RAD model is used.
- Using the RAD model the fully functional system can be developed within **60 to 90 days.**
- Various phases in RAD are Requirements Gathering, Analysis and Planning, Design, Build or Construction and finally Deployment.
- Multiple teams work on developing the software system using RAD model parallely.
- In the **requirements gathering** phase the developers communicate with the users of the system and understand the business process and requirements of the software system.
- During **analysis and planning** phase, the analysis on the gathered requirements is made and a planning for various software development activities is done.
- During the **design** phase various models are created. Those models are Business model, data model and process model.
- The **build** is an activity in which using the existing software components and automatic code generation tool the implementation code is created for the software system. This code is well tested by its team. The functionalities developed by all the teams are integrated to form a whole.
- Finally the deployment of all the software components (created by various teams working on the project) is carried out.

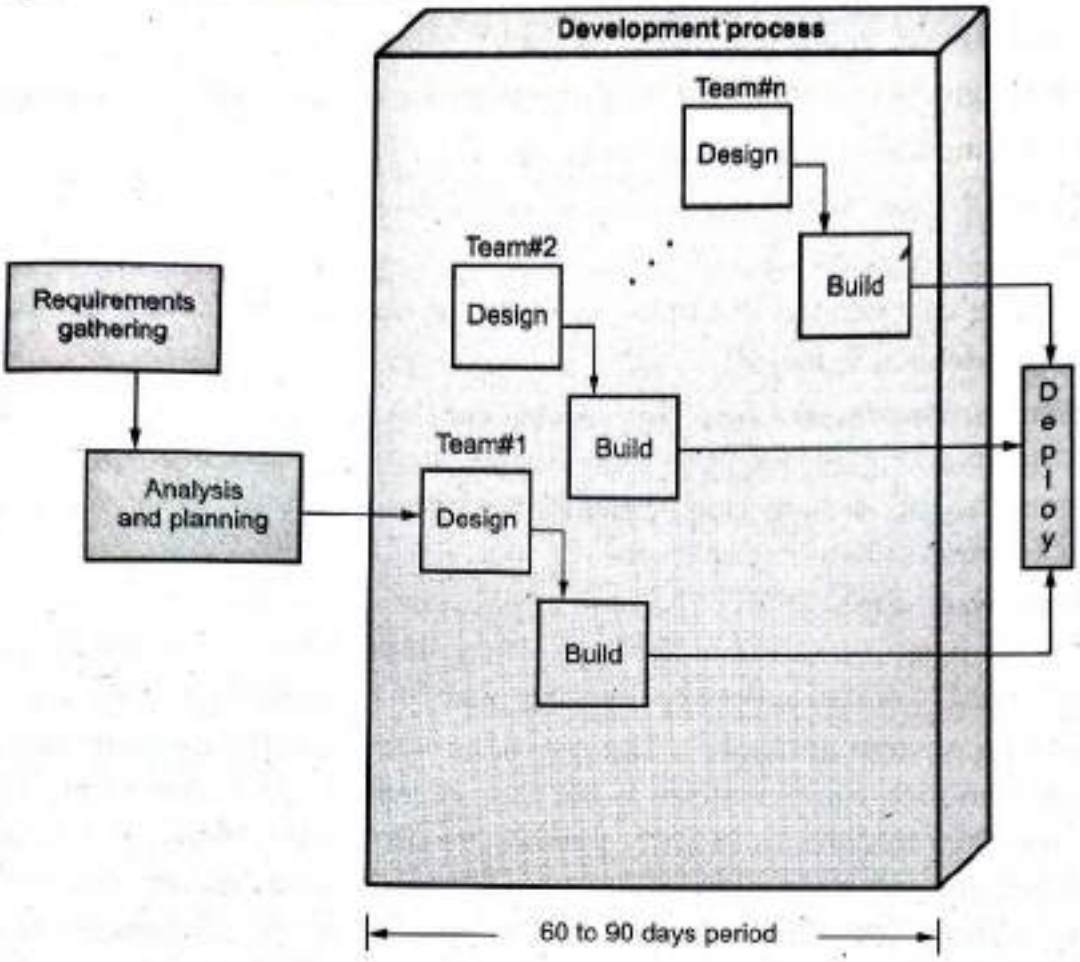


Fig. 1.6.4 Rapid application development

Drawbacks of rapid application development

1. It requires **multiple teams** or large number of people to work on the scalable projects.
2. This model requires **heavily committed developer** and customers. If commitment is lacking then RAD projects will fail.
3. The projects using RAD model requires **heavy resources**.
4. If there is no appropriate modularization then RAD projects fail. Performance can be problem to such projects.
5. The projects using RAD model find it difficult to adopt new technologies.

Example 1.6.2 *Which type of applications suit RAD model ? Justify your answer.*

Solution : The RAD model is suitable for information system applications, business applications and the for systems that can be modularized because of following reasons -

1. This model is similar to waterfall model but it uses very short development cycle.
2. It uses component-based construction and emphasises reuse and code generation.

3. This model uses multiple teams on scaleable projects.
4. The RAD model is suitable for the projects where technical risks are not high.
5. The RAD model requires heavy resources.

Example 1.6.3 *Provide three examples of software projects that would be amenable to incremental model. Be specific*

Solution : There can various examples of software projects that would be amenable to incremental model. For instance -

1. **Banking software service :** This service can be personal service. That means for personal banking system the incremental model can be used. In later state of increments, this system can implement insurance service, home loans and some other features of banking services.
2. **Web browser application :** The base application can be developed and distributed. This is the basic increment of the application. In the later increments the plugins can be provided to enhance the experience of web browser applications.
3. **Operating system software :** The operating system software providing the basic system handling functionalities is the first increment. After the release of the basic versions then updates or security patches are provided to the customer in the form of increments. Various distribution package in the form of versions such as basic home edition, premium, ultimate and so on can be the increments of operating system software.

1.6.4 Evolutionary Process Model

While developing the software systems, it is often needed to make modifications in earlier development phases or the tasks sets. If the development process is linear or in a straight line (from requirements gathering to deployment) then the end product will be unrealistic. In such cases, the **iterative approach** needs to be adopted. The evolutionary process model is **iterative model.**

1.6.4.1 Prototyping

- In prototyping model initially the requirement gathering is done.
- Developer and customer define overall objectives; identify areas needing more requirement gathering.
- Then a quick design is prepared. This design represents what will be visible to user in input and output format.
- From the quick design a prototype is prepared. Customer or user evaluates the prototype in order to refine the requirements. Iteratively prototype is tuned for satisfying customer requirements. Thus prototype is important to identify the software requirements.

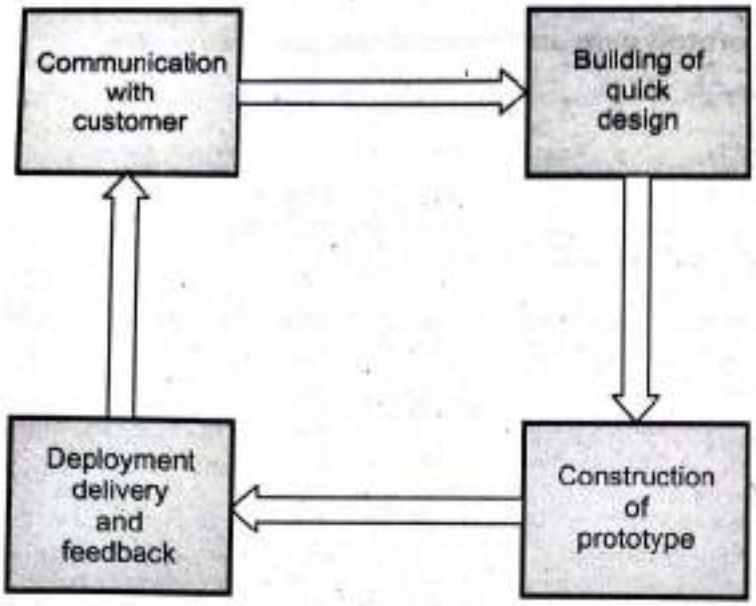


Fig. 1.6.5 Prototyping

- When working prototype is built, developer use existing program fragments or program generators to throw away the prototype and rebuild the system to high quality.
- Certain classes of mathematical algorithms, subset of command driven systems and other applications where results can be easily examined without real time interaction can be developed using prototyping paradigm.

When to choose it ?

- Software applications that are relatively easy to prototype and that always involve Human-Computer Interaction (HCI), the prototyping model is suggested.
- A general objective of software is defined but not detailed input, processing or output requirements. Then in such a case prototyping model is useful.
- When the developer is unsure of the efficiency of an algorithm or the adaptability of an operating system then prototype serves as a better choice.

Drawbacks of prototyping

1. In the first version itself, customer often wants "few fixes" rather than rebuilding of the system whereas rebuilding of new system maintains high level of quality.
2. The first version may have some compromises.
3. Sometimes developer may make implementation compromises to get prototype working quickly. Later on developer may become comfortable with compromises and forget why they are inappropriate.

Comparison between prototyping and incremental process model

Sr. No.	Prototyping	Incremental process model
1.	Some requirements are gathered initially, but there may be change in requirements when the working prototype is shown to the customer.	The requirements are precisely defined and there is no confusion about the final product of the software.
2.	The development team has adequate domain knowledge. Similarly they can adopt the new technologies if product demands.	The development team with less domain knowledge can be accommodated due to iterative nature of this model. The change in technology in the later phase can not be tolerated.
3.	All the end-users are involved in all phases of development.	All the end-users need not be involved in all the phases of development.
4.	There can be use of some reusable software components in project development process.	There is no use of reusable components in development process.

1.6.4.2 Spiral Model

- This model possess the **iterative nature** of prototyping model and controlled and **systematic approaches** of the linear sequential model.
- This model gives efficient development of incremental versions of software. In this model, the software is developed in series of increments.
- The sprial model is divided into a number of **framework activities**. These framework activities are denoted by **task regions**.
- Usually there are **six tasks regions**. The spiral model is as shown in Fig. 1.6.6. (See Fig. 1.6.6 on next page)
- Spiral model is realistic approach to development of **large-scale systems** and software. Because customer and developer better understand the problem statement **at each evolutionary level**. Also **risks** can be identified or rectified at each such level.
- In the **initial pass**, product specification is built and in subsequent passes around the spiral the prototype gets developed and then more improved versions of software gets developed.
- During **planning phase**, the cost and schedule of software can be planned and adjusted based on feedback obtained from customer evaluation.
- In spiral model, project **entry point axis** is defined. This axis represents starting point for different types of projects.
- For instance, concept development project will start at core of spiral and will continue along the spiral path. If the concept has to be developed into actual project then at entry point 2 the product development process starts. Hence entry

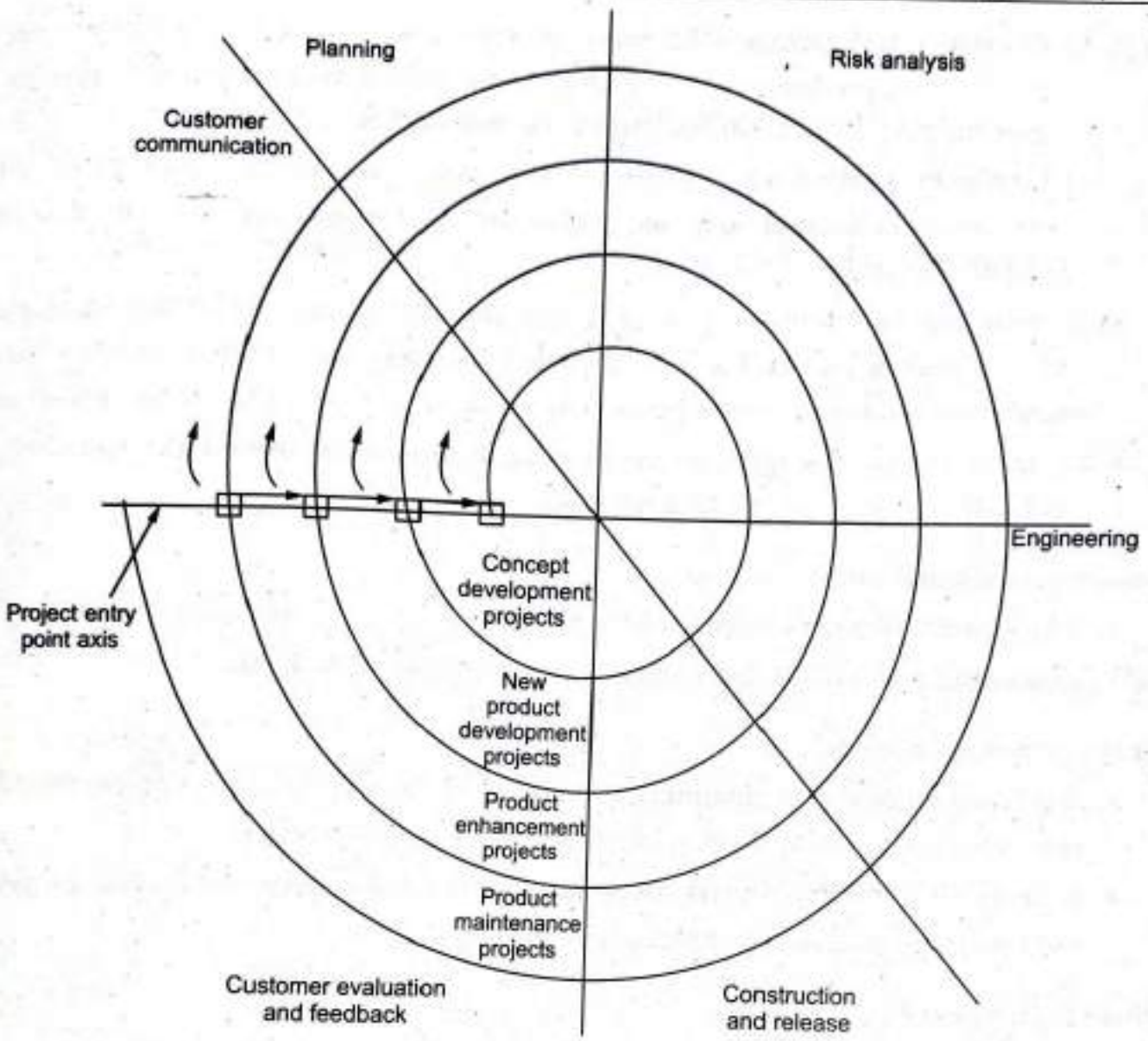


Fig. 1.6.6 Spiral model

point 2 is called product development project entry point. The development of the project can be carried out in iterations.

- The task regions can be described as :

i) **Customer communication** - In this region, it is suggested to establish customer communication.

ii) **Planning** - All planning activities are carried out in order to define resources time line and other project related activities.

iii) **Risk analysis** - The tasks required to calculate technical and management risks are carried out.

iv) **Engineering** - In this task region, tasks required to build one or more representations of applications are carried out.

v) **Construct and release** - All the necessary tasks required to construct, test, install the application are conducted. Some tasks that are required to provide user support are also carried out in this task region.

vi) **Customer evaluation** - Customer's feedback is obtained and based on customer evaluation required tasks are performed and implemented at installation stage.

- In each region, number of **work tasks** are carried out depending upon the characteristics of project. For a small project relatively small number of work tasks are adopted but for a complex project large number of work tasks can be carried out.
- In spiral model, the software engineering team **moves around the spiral** in a clockwise direction beginning at the core.

Advantages of spiral model

- Requirement changes can be made at every stage.
- **Risks can be identified** and rectified before they get problematic.

Drawbacks of spiral model

- It is based on **customer communication**. If the communication is not proper then the software product that gets developed will not be up to the mark.
- It demands considerable **risk assessment**. If the risk assessment is done properly then only the successful product can be obtained.

When to choose it ?

1. When the prototypes for the software functionality are needed.
2. When requirements are **not very clearly defined** or complex.
3. When the **large or high budget** projects need to be developed.
4. When the **risk assessment** is very critical and essential
5. When project is not expected within a specific limited time span.

Comparison between Spiral model and Prototyping model

Sr. No.	Spiral model	Prototyping model
1.	The development team with less domain knowledge can be accommodated due to iterative nature of this model. The change in technology in the later phase can not be tolerated.	The development team has adequate domain knowledge. Similarly they can adopt the new technologies if product demands.

2.	All the end-users need not be involved in all the phases of development.	All the end-users are involved in all phases of development.
3.	Funding are not stable for the projects that can be developed using spiral model.	Funding are stable for these type of projets.
4.	The requirements that are gathered and analyzed are high reliability requirements.	Some requirements are gathered initially, but there may be change in requirements when the working prototype is shown to the customer.

Difference between waterfall model and spiral model

Sr. No.	Waterfall model	Spiral model
1.	It requires well **understanding** of **requirements** and familiar technology.	It is developed in **iterations**. Hence the requirements can be identified at new iterations.
2.	**Difficult** to accommodate **changes** after the process has started.	The required changes can be made at every stage of **new version**.
3.	Can accommodate iteration but indirectly.	It is iterative model.
4.	**Risks** can be identified **at the end** which may cause failure to the product.	**Risks** can be identified and **reduced** before they get problematic.
5.	The customer can see the **working model** of the project only **at the end**. After reviewing of the working model; if the customer gets dissatisfied then it causes serious problems.	The customer can see the **working product at certain stages** of iterations.
6.	**Customers prefer** this model.	**Developers prefer** this model.
7.	This model is good for **small systems**.	This model is good for **large systems**.
8.	It has **sequential** nature.	It has **evolutionary** nature.

1.6.4.3 Concurrent Development Model

- The concurrent development model is also called as **concurrent engineering.**
- In this model, the framework activities or software development tasks are represented as **states.**
- The **modeling** or **designing** phase of software development can be in one of the states like *under development, waiting for modification, under revision* or *under review* and so on.
- Fig. 1.6.7 represents these states.

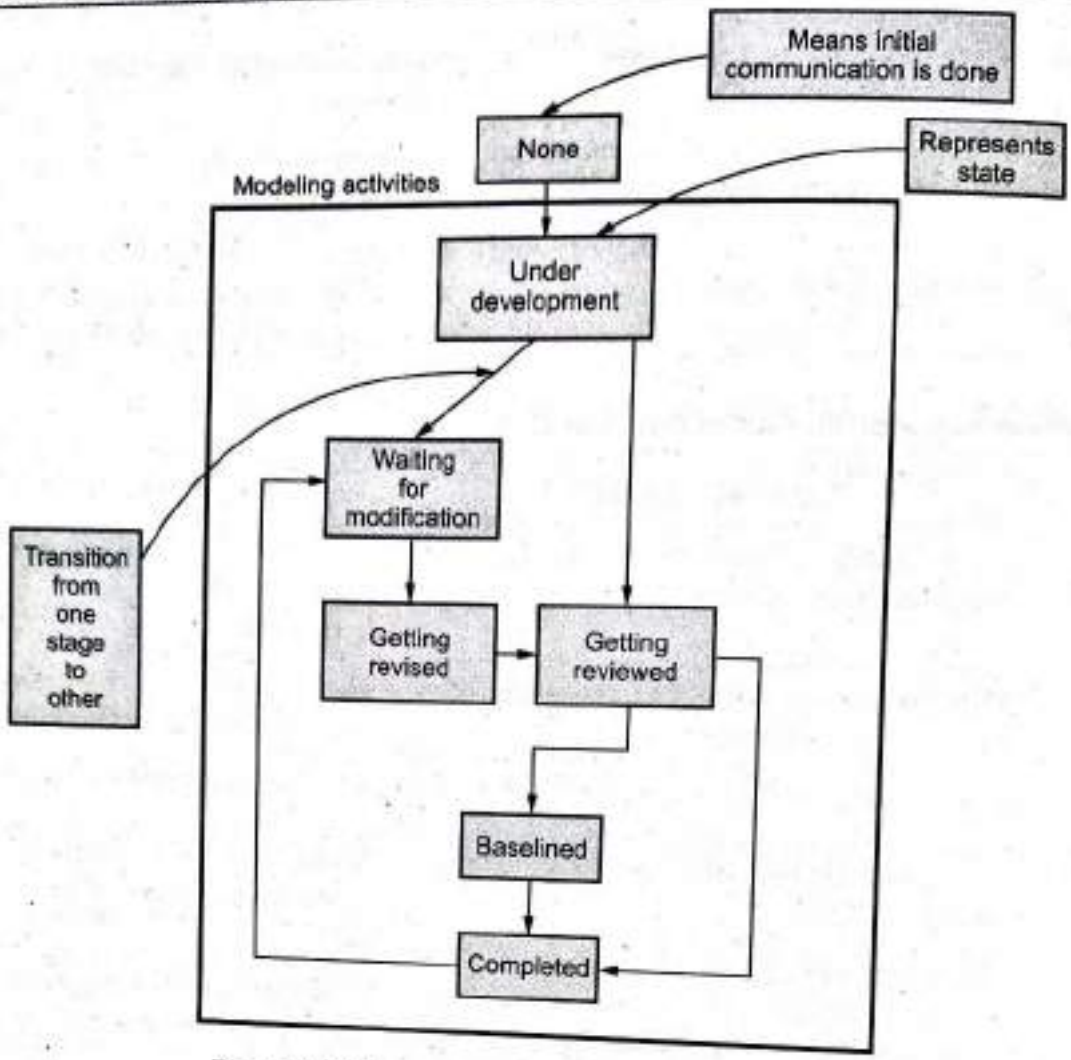


Fig. 1.6.7 Concurrent development model

- All the software development activities exist concurrently in this model but these activities can be in various **states.**
- These states make transitions. That is during **modeling,** the transition from **under development** state to **waiting for modification** state occurs.
- This model basically defines the **series of events** due to which the transition from one state to another state occurs. This is called **triggering.** These series of events occur for every software development activity, action or task.
- This model defines various activities that occur concurrently and a network of activities is defined.

Advantages

1. All types of software development can be done using concurrent development model.
2. This model provides accurate picture of current state of project.
3. Each activity or task can be carried out concurrently. Hence this model is an efficient process model.

Example 1.6.4 *Compare and contrast the different life cycle models.* **AU : Dec.-11, Marks 6**

Solution :

Waterfall model	Spiral model	Prototyping model	Incremental model
Requirements must be clearly understood and defined at the beginning only.	The **requirements** analysis and gathering can be done in iterations because requirements get changed quite often.	**Requirements analysis** can be made in the later stages of the development cycle, because requirements get changed quite often.	The **requirements analysis** can be made in the later stages of the development cycle.
The **development team** having the adequate experience of working on the similar project is chosen to work on this type of process model	The **development team** having less experience of working on the similar projects is allowed in this process model	The **development team** having less experience of working on the similar projects is allowed in this process model.	The **development team** having the adequate experience of working on the similar project is chosen to work on this type of process model.
There is **no user involvement** in all the phases of development process.	There is **no user involvement** in all the phases of development process.	There is **user involvement** in all the phases of development process.	There is **user involvement** in all the phases of development process.
When the requirements are reasonably well defined and the development effort suggests a purely linear effort then the **waterfall model is chosen.**	Due to iterative nature of this model, the risk identification and rectification is done before they get problematic. Hence for handling real time problems the **spiral model is chosen.**	When developer is unsure about the efficiency of an algorithm or the adaptability of an operating system then the **prototyping model is chosen.**	When the requirements are reasonably well defined, the development effort suggests a purely linear effort and when limited set of software functionality is needed quickly then the **incremental model is chosen.**

Example 1.6.5 *For the scenario described below, which life cycle model would you choose? Give the reason why you would choose this model.*

You are interacting with the MIS department of a very large oil company with multiple departments. They have a complex legacy system. Migrating the data from this legacy system is not an easy task and would take a considerable time. The oil company is very particular about processes, acceptance criteria and legal contracts. **AU : Dec.-11, Marks 5**

Solution : Legacy applications are database management systems. The software companies normally migrate these applications to some relational databases. For data migrating projects a cohesive method that enables the developer to cycle or spiral through

the migration process is required. In this migrating process following steps need to be followed -

1. Analyze the data.
2. Extract the data which is to be transformed.
3. Transform the extracted data.
4. Validate the transformed data.
5. Load the validated data to the target system.

These above mentioned steps are repeated until the migration is successfully completed. The risk management must be done properly. All these factors suggest that the spiral model is suitable for the given project.

Example 1.6.6 *Describe at least one scenario where*
1. RAD model would be applicable and not the waterfall model.
2. Waterfall model is preferable to all other models. **AU : Dec.-11, Marks 10**

Solution : 1. RAD model would be applicable and not the waterfall model :

Following are some scenario where RAD model would be applicable and not the waterfall model.

A) The projects in which users are involved in all phases.

B) The projects in which users are experts of problem domain.

C) The project is enhancement of existing system.

2. Waterfall model is preferable to all other models :

In the following scenario waterfall model is preferable to all other models.

The project for which requirements are easily understandable and defined.

Example 1.6.7 *How does a spiral model represent a process suitable to represent a real time problem ?* **AU : CSE, May-05, Marks 8**

Solution : Spiral model represents a process suitable to represent a real time problem because of following reasons -

1. Software evolves as the project progresses. And at every evolutionary level the risks are identified and managed and risks are reduced at every stage.
2. It enables the developer to apply the prototype approach at any stage in the evolution of the product. It helps in adopting the approach systematic stepwise development of the product.
3. The iterative frameworks help in analyzing the product at every evolutionary stage.

4. The spiral model demands a direct consideration of technical risks at all stages of project. The risks are reduced before they get problematic.

Example 1.6.8 *Which type of applications suit RAD model ? Justify your answer.*

AU : May-06, Marks 10

Solution : The RAD model is suitable for information system applications, business applications and the for systems that can be modularized because of following reasons -

1. This model is similar to waterfall model but it uses very short development cycle.
2. It uses component-based construction and emphasises reuse and code generation.
3. This model uses multiple teams on scaleable projects.
4. The RAD model is suitable for the projects where technical risks are not high.
5. The RAD model requires heavy resources.

Example 1.6.9 *Discuss the major differences between software life cycle model and a process model.*

AU : CSE, Dec.-06, Marks 4

Solution :

Software life cycle model	Process model
This model is based on common four activities : analysis, design, code and testing.	This model is based on problem definition, technical development, software integration and existing status.
The software development process can be clearly and systematically defined in phases.	The software development process cannot be clearly defined in phases.
Customer interaction is possible in every stage of software development process in this model.	Customer interaction is only at initial stage of software development.
Large scale projects can be handled using this approach.	Large scale projects may be caught in chaotic situation using this approach.

Example 1.6.10 *As you move outward along with process flow path of the spiral model, what can we say about the software that is being developed or maintained ?*

AU : CSE, May.-09, Marks 4

Solution : When software engineering team moves around the spiral, the first circuit around the spiral results in development of product specification. The subsequent passes around the spiral might be used to develop prototype in more subsequent manner. In each pass, through planning region, some adjustments to project plan are made. Cost and schedule adjustments can also be made according to customer feedback.

Example 1.6.11 *A software project which is considered to be very simple and the customer is in position of giving all the requirements at the initial stage, which process model would you prefer for developing the project ?* **AU : IT, Dec.-09, Marks 16**

Solution : The linear sequential model or a waterfall model is appropriate when a simple project has to be developed with already known requirements.

Waterfall model - Refer section 1.6.2

Example 1.6.12 *Assume that you are the technical manager of a software development organization. A client approached you for a software solution. The problems stated by the client have uncertainties which lead to loss if it not planned and solved. Which software development model you will suggest for this project – justify. Explain that model with its pros and cons and neat sketch.* **AU : Dec.-16, Marks 16**

Solution : The uncertainties in problem statement leads to risks in the project. Hence the spiral model can be used for development of such project.

Spiral model - Refer section 1.6.4.2.

Example 1.6.13 *What is the role of user participation in the selection of a life cycle model?* **AU : May-16, Marks 8**

Solution : Due to users' participation , their understanding about the project or system increases. Following are the issues that depict the role of user participation in selection of life cycle model.

1. Prototype model and RAD model are the life cycle model in which **user involvement** is expected in **all the phases.**
2. There is no user involvement expected in all the phases of waterfall, spiral model and evolutionary development model.
3. The waterfall model, spiral model, iterative enhancement model or evolutionary development model **do not require limited user participation.**
4. RAD model and Prototype model does not allow limited user participation.
5. Prototype model, RAD Model, Evolutionary development model require that the **users must be experts of problem domain.** They should understand the system properly well in advance.
6. For the selection of waterfall model, it is **not** necessary that the **user has expertise of problem** domain.
7. The prototype, iterative enhancement model, spiral model and Evolutionary model can be selected by the users **having no previous experience of participation in similar projects.**
8. For selection of waterfall model and prototype model, the users must have **participated earlier in similar type of projects.**

Example 1.6.14 *Describe the type of situations where iterative enhancement model might lead to difficulties.* **AU : May-16, Marks 8**

Solution :

1. In the prototyping model, during the first prototype itself, the customer often wants **few fixes** rather than rebuilding of new system.
2. Sometimes developer may make implementation compromises to get prototype working quickly. But later on the developer may become **comfortable with compromises** and forget why they were inappropriate.
3. The spiral model is mainly based on customer communication. If the **communication** is **not proper** then the software product that gets developed will not be up to the mark.
4. During iterative enhancement cycle of project development, if the **risk assessment** is **not done** properly then the product will become total failure.

Example 1.6.15 *What is process model ? Describe the process model that you would choose to manufacture a car. Explain by giving suitable reasons.* **AU : May-17, Marks 13**

Solution : Process model : Refer section 1.6.

The process model that can be used for car manufacture is **spiral model.**

Spiral model : Refer section 1.6.4.2.

Reasons :

1) This is a model in which **risks** can be **identified** and rectified before they get problematic.
2) The **iterative framework** of this model helps in analyzing the problem at every evolutionary stage.
3) The required **changes can be made at every stage** of new version.
4) This model is **good for large systems.**

Review Questions

1. *Neatly explain the following process models and write their advantages and disadvantages. i) Spiral model, ii) Rapid application development model.* **AU : May-15, Marks 8**
2. *Discuss the prototyping model. What is the effect of designing a prototype on the overall cost of the software project ?* **AU : May-16, Marks 8**
3. *Which process model is best suited for risk management ? Discuss in detail with an example. Give the advantages and disadvantages of the model.* **AU : Dec.-16, Marks 16**
4. *What is the significance of the spiral model when compared with other models.* **AU : Dec-17, Marks 3**
5. *Which software process model is good for risk management ? Explain the model. Describe how the model is used to layout the objectives, risks and plans for quality improvement.* **AU : May-18, Marks 16**

6. *Outline the spiral life cycle model with a diagram.* **AU : Dec.-19, Marks 13, May-22, Marks 7**
7. *Compare and contrast waterfall model, spiral model and iterative model.* **AU : Dec.-20, Marks 13**
8. *Discuss about the waterfall model and spiral model with advantages and disadvantages* **AU : Dec.-22, Marks 7**

1.7 Specialized Model

AU : Dec.-13,14,15,17, May-16,19, Marks 16

The specialized models are used when only collection of specialized technique or methods are expected for developing the specific software.

Various types of specialized models are -

1. Component based development
2. Formal methods model
3. Aspect oriented software development

Let us discuss them in detail.

1.7.1 Component based Development

- The commercial off-the-shelves components that are developed by the vendors are used during the software built.
- These components have specialized targeted functionalities and well defined interfaces. Hence it is easy to integrate these components into the existing software.
- The component based development model makes use of various characteristics of **spiral model.** This model is evolutionary in nature. That means the necessary changes can be made in the software during the each iteration of software development cycle.
- Before beginning the modelling and construction activity of software development the **candidate component** must be searched and analyzed. The components can be simple functions or can be object oriented classes or methods.
- Following steps are applied for component based development -
  - Identify the component based products and analyze them for fitting in the existing application domain.
  - Analyze the component integration issues.
  - Design the software architecture to accommodate the components
  - Integrate the components into the software architecture.
  - Conduct comprehensive testing for the developed software.
- **Software reusability** is the major advantage of component based development.
- The **reusability** reduces the development cycle time and overall cost.

1.7.2 Formal Methods Model

- This model consists of the set of activities in which the formal mathematical specification is used.
- The software engineers specify, develop and test the computer based systems using the mathematical notations. The notations are specified within the formal methods.
- **Cleanroom software engineering** makes use of the formal method approach.
- The **advantage** of using formal methods model is that it overcomes many problems that we encounter in traditional software process models. **Ambiguity, incompleteness and inconsistency** are those problems that can be overcome if we use formal methods model.

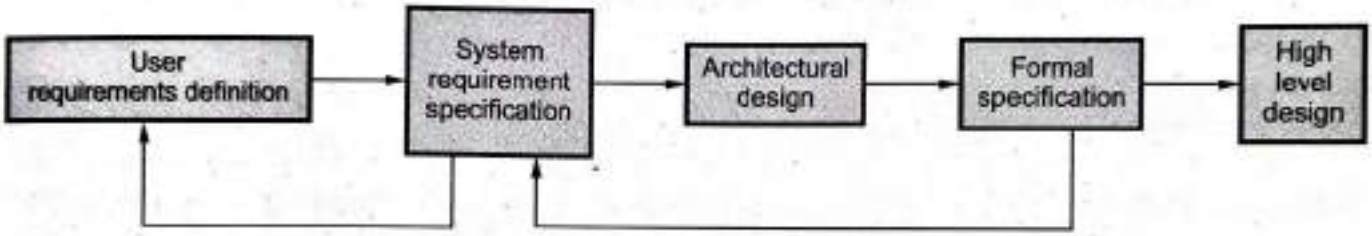


Fig. 1.7.1 Activities for formal method model

- The formal methods model offers **defect-free** software. However there are some **drawbacks** of this model which resists it from getting used widely. These drawbacks are
  - The formal methods model is **time consuming** and **expensive**.
  - For using this model, the developers need the strong **mathematical background** or some extensive training.
  - If this model is chosen for development then the **communication** with customer becomes very **difficult**.

1.7.3 Aspect Oriented Software Development

- In traditional software development process, the system is decomposed into multiple units of **primary functionality**. But there are other issues of **concern** that do not fit into these primary functionalities. Later on this becomes programmers' duty to code modules corresponding to the primary functionality and to incorporate all other concerned issues wherever appropriate.
- Programmers need to keep in mind all the things that need to be done, how to deal with each issue, the problems associated with them and the correct execution. Due to these concerns there are chances of appearing serious problems during

application development and maintenance. The coding process for **realizing a concern** becomes very critical.

- **Aspect-Oriented Software Development** focuses on the identification, specification and representation of **cross-cutting concerns and their modularization** into separate functional units as well as their automated composition into a working system.
- **Aspectual requirements** define these cross-cutting concerns that have impact on the software architecture.
- **Aspect Oriented Software Development (AOSD)** is often referred as **Aspect oriented programming.**
- It is a relatively new software engineering paradigm and is not matured enough. But is likely that it will adopt the characteristics of both the spiral and concurrent process models.

Example 1.7.1 *What are pros and cons of using mathematical approach for software development?* **AU : Dec.-15, Marks 6**

Solution : Pros : 1) The mathematical approach is useful in building scientific model, critical systems or simulation of real time systems.

2) The ambiguity or incosistency problems can be eliminated if mathematical approach is adopted.

3) Developers are exceptionally committed to the project.

4) The controlled and optimized system can be developed using mathematical approach.

Cons :

1) The developers need the strong mathematical background or some extensive training.

2) The methods of software development are time consuming and expensive.

3) Many times-if mathematical approach is adopted for software development, then communication with customer becomes very difficult.

Example 1.7.2 *Compare the following life cycle models based on their distinguishing factors, strengths and weaknesses, - Waterfall Model, RAD Model, Spiral Model and Formal Methods Model. (Present in the form of table only - use diagrams wherever necessary)*

AU : Dec.-13, 14, Marks 16; May-19, Marks 13

OR

Elucidate the key features of the software process models with suitable examples.

AU : May-16, Marks 8

Solution :

Waterfall model	RAD model	Spiral model	Formal methods model
Requirements must be **clearly understood** and defined at the beginning only.	**Requirements** must be **clearly understood** and defined at the beginning only.	The requirements analysis and gathering can be done in iterations because **requirements get changed quite often.**	**Requirements** must be **clearly understood** and defined at the beginning only.
The **development team** having the adequate experience of working on the similar project is chosen to work on this type of process model.	The **development team** having the adequate experience of working on the similar project is chosen to work on this type of process model.	The **development team** having less experience of working on the similar projects is allowed in this process model.	The **development team** having the adequate experience of working on the similar project is chosen to work on this type of process model.
If the development team has **less domain knowledge** or if it new to the technology then such a team is allowed for this kind of process model.	If the development team has **less domain knowledge** or if it new to the technology then such a team is **not allowed** for this kind of process model.	If the development team has **less domain knowledge** or if it new to the technology then such a team is allowed for this kind of process model.	If the development team has **less domain knowledge** or if it new to the technology then such a team is **not allowed** for this kind of process model.
There is **no user involvement** in all the phases of development process.	There is **user involvement** in all the phases of development process.	There is **no user involvement** in all the phases of development process.	**Limited community** makes **use** of formal methods model for their projects because this methodology is based on mathematical theorems, formal methods and automata theory.
Types of Projects : When the **requirements** are reasonably **well defined** and the development effort suggests a purely linear effort then the waterfall model is chosen	When there is a **tight project schedule** or project is **enhancement** of the **existing projects** or if there is use of **reusable components** in the project then for developing such projects this process model is suitable.	Due to iterative nature of this model, the risk identification and rectification is done before they get problematic. Hence for handling real time problems the **spiral model is chosen.**	Whenever there is a need to build the **scientific models** based on mathematical techniques, or **simulation** of some **real time systems** or when there is a need to build the systems that can contribute to the reliability and robustness (normally **critical systems**) then the formal methods models are used.
Diagram : Refer Fig. 1.6.2.	**Diagram :** Refer Fig. 1.6.4.	**Diagram :** Refer Fig. 1.6.6.	**Diagram :** Refer Fig. 1.7.1.

Review Question

1. *Explain the component based software development model with a neat sketch.* **AU : Dec.-17, Marks 9**

1.8 Introduction to Agility

- The Agile Manifesto, also called the **Manifesto for Agile Software Development**, is a formal declaration of **four key values** and **12 principles** to guide an iterative and people-centric approach to software development.
- The agile methods were developed to overcome the weakness of conventional software engineering.
- The agile manifesto is represented by following Fig. 1.8.1.

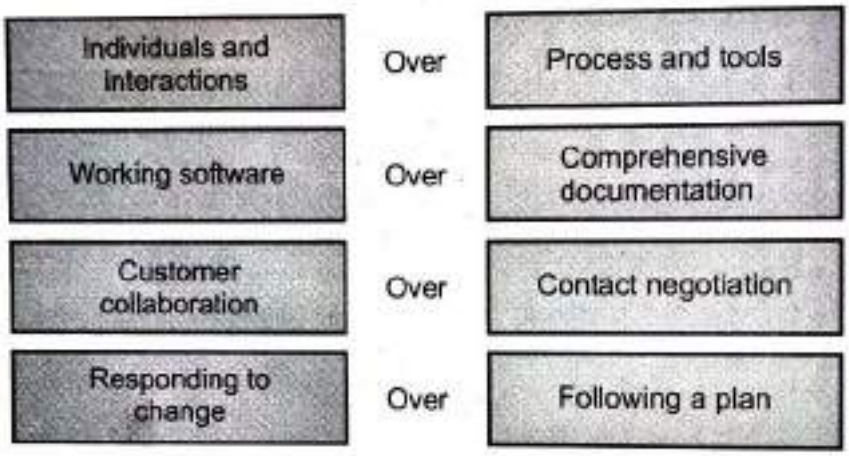


Fig. 1.8.1 Agile manifesto

1.9 Agile Process

AU : Dec.-16, 19, May-19, Marks 4

- In 1980's the **heavy weight, plan based software** development approach was used to develop any software product.
- In this approach too many things are done which were not directly related to software product being produced.
- This approach was rigid. That means if requirements get changed, then rework was essential. Hence new methods were proposed in 1990's which are known as agile processes.
- The agile processes are the light-weight methods are **people-based** rather than plan-based methods.
- The agile process forces the development team to **focus** on **software** itself rather than design and documentation.
- The agile process believes in **iterative method.**

- The aim of agile process is to **deliver** the working model of software **quickly** to the customer.
- **For example :** Extreme programming is the best known of agile process.

Conventional Software Development Methodology

- As the software project makes the progress, the cost of the changes increases non linearly.
- It is easy to accommodate changes during the requirement gathering stage. At this stage to accommodate the changes - usage scenarios are modified, list of functions can be extended, or written specification be edited.
- As the progresses and if the customer suggest the changes during the testing phase of the software development life cycle then to accommodate these changes the architectural design needs to be modified and ultimately these changes will affect other phases of software development cycle. These changes are actually costly to execute.

Agile Methodology

- The agile method proponents claim that if the software development is carried out using the agile approach then it will allow the software team to accommodate changes late in a software project without dramatic cost and time impact.
- In other words, if the incremental delivery is combined with agile practices such as continuous unit testing and pair programming then the cost of changes can be controlled.
- The following graph represents the how the software development approach has a strong influence on the development cost due to changes suggested.

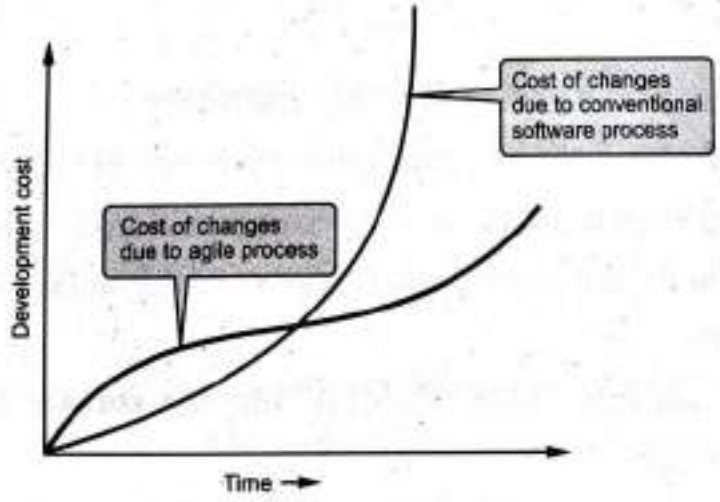


Fig. 1.9.1 Influence of software development approach on agile process

Pros

1. **Efficient Delivery :** Agile software development approach emphasizes on delivering the small increments of working software quickly.
2. **Flexibility :** The Agile software development methodology allows for changes and adaptions to be made throughout the development process.
3. **Continuous Improvement :** Agility encourages continuous reflection and improvement in the software.
4. **Transparency :** During agile software development, daily stand-up meetings makes all the team members and stakeholders aware of the project's progress, challenges and decisions.
5. **Reduced Risk :** Transparency helps to identify and mitigate risks early on. When everyone is aware of the project's challenges, they can work together to develop solutions.

Cons

1. **Need for Disciplined Team :** Agile team requires more communication and collaboration for successful software development. This is difficult for teams that are new to Agile.
2. **More Collaboration and Communication :** For Agile Software Development, there needs more collaboration and communication which is challenging for large and distributed projects.
3. **Lack of Documentation :** Agile prioritizes working software over comprehensive documentation, which can be a disadvantage in situations where documentation is essential, such as in highly regulated industries.

1.9.1 Agile Principles

There are famous 12 principles used as agility principles -

1. Satisfy the customer by early and continuous delivery of valuable software.
2. The changes in the requirements must be accommodated. Even though the changes occur late in the software development process, the agile process should help to accommodate them.
3. Deliver working software quite often. Within the shorter time span deliver the working unit.
4. Business people and developers must work together throughput the project.

5. Motivate the people who are building the projects. Provide the environment and support to the development team and trust them for the job to be done.
6. The working software is the primary measure of the progress of the software development.
7. The agile software development approach promote the constant project development. The constant speed for the development of the product must be maintained.
8. To enhance the agility there should be continuous technical excellence
9. The proper attention to be given to technical excellence and good design.
10. The simplicity must be maintained while developing the project using this approach.
11. The teams must be the self-organizing team for getting best architecture, requirements and design.
12. At regular intervals the team thinks over the issue of becoming effective. After the careful review the team members adjusts their behavior accordingly.

Review Questions

1. *List the principles of agile software development.* **AU : Dec-16, Marks 4**
2. *Define Agility. List any five principles of agility.* **AU : May-19, Marks 5**
3. *What is agility ? Elaborate the agile principles.* **AU : Dec.-19, Marks 13**

1.10 Extreme Programming

AU : May-19,22, Dec.-20,22, Marks 13

Extreme Programming (XP) is one of the best known agile process. It is suggested by Kent Beck in 2000.

1.10.1 XP Values

Beck defined the set of five values that serve as a basis for the work performed in XP. These values are -

1. **Communication** : The effective communication must be established between software developers and stakeholders in order to convey the important concepts and to get the important feedback.
2. **Simplicity** : XP focuses on the current needs instead of futures needs to incorporate in the design. Hence the XP believes that the Software design should be simple.
3. **Feedback** : The feedback for the software product can be obtained from the developers of the software, customers, and other software team members.

4. **Courage :** The strict adherence to certain XP practices require courage. The agile XP team must be disciplined to design the system today, recognize the future requirements and make the changes dramatically as per the demand.
5. **Respect :** By following the above states XP values the agile team can win the respect of stakeholders.

1.10.2 Process

- The extreme programming process is explained as follows -
- Customer specifies and priorities the system requirements. Customer becomes one of the important members of development team. The developer and customer together prepare a **story-card** in which customer needs are mentioned.
- The developer team then aims to implement the scenarios in the story-card.
- After developing the story-card the development team breaks down the total work in **small tasks**. The efforts and the estimated resources required for these tasks are estimated.
- The customer priorities the stories for implementation. If the requirement changes then sometimes unimplemented stories have to be discarded. Then release the complete software in **small** and frequent **releases.**
- For accommodating new changes, **new story-card** must be developed.
- Evaluate, the system along with the customer.
- This process is demonstrated by the following Fig. 1.10.1.

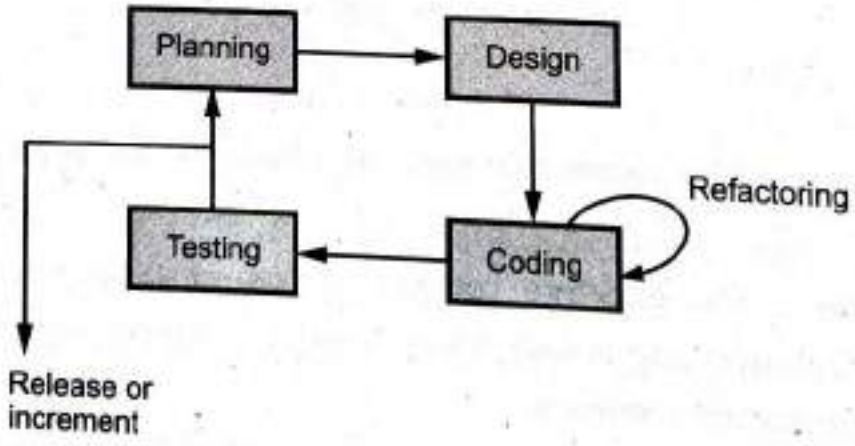


Fig. 1.10.1 XP process

- Various rules and practices used in extreme programming are as given below –

	XP Principle	Description
Planning	User story-cards	Instead of creating a large requirement document user stories are written by the customer in which what they need is mentioned.
	Release planning	A release plan for overall project is prepared from which the iteration plan can be prepared for individual iteration.
	Small releases	The developer breaks down the user stories into small releases and a plan for releasing the small functionalities is prepared.
	Iterative process	Divide the development work into small iterations. Keep the iteration of nearly constant length. Iterative development helps in quick or agile development.
	Stand up meetings	The stand up meetings must be conducted for the current outcomes of the project.
Designing	Simple design	Simple design always takes less time than the complex design. It is always good to keep the things simple to meet the current requirements.
	Spike solution	For answering the tough technical problems create the spike solutions. The goal of these solutions should be to reduce the technical risks.
	Refactoring	Refactoring means reductions in the redundancy, elimination of unused functionalities, redesign the obsolete designs. This will improve the quality of the project.

Coding	Customer availability	The most essential requirement of the XP is availability of the customer. In Extreme programming the customer not only helps the developer team but it should be the part of the project.
	Paired programming	All the code to be included in the project must be coded by groups of two people working at the same computer. This will increase the quality of coding.
	Collective code ownership	By having collective code ownership approach the everyone contributes new ideas and not any single person becomes the bottleneck of the project. Anyone can change any line of code to fix a bug or to refactor.
Testing	Unit testing	The test-first development is done in XP. The test framework that contains the automated test case suite is used to test the code. All the code must be tested using unit testing before its release.
	Continuous integration	As soon as one task is finished integrate it into the whole system. Again after such integration unit testing must be conducted.
	No overtime	Working overtime lose the spirit and motivation of the team. Conduct the release planning meeting to change the project scope or to reschedule the project.

As mentioned earlier in Extreme Programming instead of creating large requirement documents the user story-card is prepared. For example : If a project is for creating a banking software then the sample story card can be prepared as follows -

1. Customer makes the balance enquiry.
2. Customer withdraws some amount.
3. Customer deposits some amount.

Banking software checking balance, depositing and withdrawing
When the customer wants to **check the balance** then he just enters his account number and the total amount present in his account is displayed to him.
Then customer can **withdraw** some amount from his account. The limit of amount to be withdrawn must be specified. If the customer crosses that limit then he must be warned. The amount withdrawn must be deducted from the available balance.
The customer can **deposit** the desired amount in his account. The deposited amount must be added to the available balance.

Note that, the user story is broken into various tasks and simple classes must be created for these tasks. From above example it is clear that the useful class in the project could be *'account'* . Various data attributes can be *account_no, customer name, address, balance* and the useful functions can be *deposit(), withdraw(), check_balance().*

Each task can be separately tested using the automated test cases.

Example 1.10.1 *What is spike solution in XP ?*

Solution : The Extreme Programming (XP) is one of the known agile process. In this process, to answer complex and difficult technical or design problems spike solution is created.

The spike solution is very simple program to explore potential solution. The spike is bulid to only address the problem under examination and all other things can be overlooked.

The spike solution reduces the risk of technical problem and increases the reliability of user story's estimate.

1.10.3 Industrial XP

- The Industrial XP (IXP) can be defined as the organic evolution of XP. It is customer centric. It has expanded role for customers and advanced technical practices.
- Various new practices that are appended to XP to create IXP are as follows -

1. **Readiness Assessment :** In this assessment, following issues are assessed -
   i. Proper environment must be available to support XP.
   ii. Team should contain appropriate and skilled stakeholder
   iii. The organization should support quality programs and continuous improvement.
   iv. The organizational culture should support new values of agile team.
2. **Project Community :** Skilled and efficient people must be chosen as the agile team members for the success of the project. The team is referred as the **community** when extreme programming approach is considered. The project **community** consists of **technologies, customers** and **other stakeholders** who play the vital role for the success of the project. The **role** of the community members must be **explicitly defined.**
3. **Project Chartering :** Project chartering means **assessing** the justification for the project as a business application. That means, the IXP team assess whether the project satisfies the goals and objectives of the organization.

4. **Test Driven Management :** For assessing the state of the project and its progress the industrial XP needs some **measurable criteria**. In test driven management the project is tested with the help of these measurable criteria.
5. **Retrospectives :** After delivering the software increment, the **specialized review** is conducted which is called as **retrospective**. The intention of retrospectives is to improve the industrial XP process.
6. **Continuous Learning :** The team members are inspired and encouraged to learn new methods and techniques that can **improve the quality** of the product.

Review Questions

1. *Explain the phases in extreme programming process.* **AU : May-19, Marks 8**
2. *Define agile programming. Explain the 12 practices of extreme programming.* **AU : Dec.-20, Marks 13**
3. *Explain Pros and Cons of agile software development.* **AU : May 22, Marks 6**
4. *Discuss about the various practices of extreme programming.* **AU : May-22, Marks 6**
5. *Briefly explain the 12 principles of extreme programming.* **AU : Dec.-22, Marks 6**
6. *State the principles to be followed to achieve agility in software engineering and mention the type of programming that are widely used for agile software development.* **AU : Dec.-22, Marks 6**

1.11 Two Marks Questions with Answers

Q.1 Write the IEEE definition of software engineering. **AU : Dec.-17, May-19**

Ans. : Software engineering is defined as the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.

Q.2 What is Software ? List the characteristics. **AU : May-18**

Ans. : Software : Software is a collection of computer programs and related documents that are intended to provide desired features, functionalities and better performance.

Characteristics :

1) Software is engineered not manufactured.

2) Software does not ware out

3) Most software is custom built rather than being assembled from components.

Q.3 What are two type of software products ? **AU : May-12**

Ans. : There are two types of software products -

1. **Generic** - These products are developed and to be sold to the range of different customers.

2. **Custom** - These type of products are developed and sold to the specific group of customers and developed as per their requirements.

Q.4 What is software engineering ? AU : Dec.-13

OR

Define software engineering. AU : Dec.14,22, May-22

Ans. : Software engineering is a discipline in which theories, methods and tools are applied to develop professional software product.

Q.5 Software doesn't wear out. Justify. (Refer section 1.1.2) AU : Dec.-13

Q.6 Distinguish between process and methods. AU : IT, May-04, 22

Ans. : Software process can be defined as the structured set of activities that are required to develop the software system. Various activities under software process are-

- Specification
- Design and implementation
- Validation
- Evolution

Method is used mainly in object-oriented programming, the term method refers to a piece of code that is exclusively associated either with a class (called class methods) or with an object (called instance methods).

Q.7 Why software architecture is important in software process ? AU : IT, May-05

Ans. : The system architecture defines the role of hardware, software, people, database procedures and other system elements. The architectural design of the system help the developer to understand the system as a whole. Hence system architecture must be built before specifications are completed. Thus architectural design is important in software engineering.

Q.8 What are the drawbacks of rapid application development life cycle model ?

(Refer section 1.6.3.2) AU : CSE, May-05

Q.9 List out the problems encountered in linear sequential model.

(Refer section 1.6.2) AU : May-06

Q.10 What is meant by 'blocking states' in linear sequential model ? AU : IT, Dec.-06

Ans. : The linear nature of linear sequential model brings a situation in the project that some project team members have to wait for other members of the team to complete the dependent tasks. This situation is called "blocking state" in linear sequential model. For example, after performing the requirement gathering and analysis step the design process can be started. Hence the team working on design stage has to wait for gathering of all the necessary requirements. Similarly the programmers can not start coding step unless and until the design of the project is completed.

Q.11 What are the advantages of prototyping model ? AU : CSE, Dec.- 06

Ans. : Advantages of prototyping model -

1. The working model of the system can be quickly designed by construction of prototype. This gives the idea of final system to the user.
2. The prototype is evaluated by the user and the requirements can be refined during the process of software development.
3. This method encourages active participation of developer and user.
4. This type of model is cost effective.
5. This model helps to refine the potential risks associated with the delivery of the final system.
6. The system development speed can be increased with this approach.

Q.12 Write any two software engineering challenges. AU : IT, May-07, Dec.-22

Ans. : The key challenges faced by software engineering are -

1. Delivery times challenge

There is increasing pressure for **faster delivery** of software. When the complexity of the systems that we develop increases, this challenge becomes harder.

2. Heterogeneity challenge

Sometimes systems are distributed and include **a mix of hardware and software.** This implies that software systems must **cleanly integrate** with other different software systems, built by different organizations and team which may be using different hardware and software platform.

Q.13 Identify in which phase of the software life cycle the following documents are delivered.

a) Architectural design b) Test plan

c) Cost estimate d) Source code document.

Ans. : AU : IT, May-07

	Document	Phase
(a)	Architectural design	Design
(b)	Test plan	Testing
(c)	Cost estimate	Project management and planning
(d)	Source code document	Coding

Q.14 Define the terms product and process in software engineering. AU : IT, May-07

Ans. : The product in software engineering is a standalone entity that can be produced by development organization and sold on the open market to any customer who is able to buy them. The software product consists of computer programs, procedures, and associated

documentation (documentation can be in hard copy form or it may be in visual form). Some of the examples of software product are databases, word processors, drawing tools.

The process in software engineering can be defined as the structured set of activities that are required to develop software system. Various activities under software process are -

- Specification
- Design and implementation
- Validation
- Evolution

Q.15 What are the phases encompassed in the RAD model ? AU : CSE, Dec.-07

Ans. : Various phases in the RAD model are

1. Business modelling 2. Data modelling 3. Process modelling
4. Application generation 5. Testing and turnover

Q.16 Define a system and computer based system. AU : CSE, Dec.-07

Ans. : **System :** A system can be defined as a purposeful collection of inter-related components working together to achieve some common objectives.

Computer based system : The computer based system can be defined as a set or an arrangement of elements that are organized to accomplish some predefined goal by processing information.

Q.17 Which process model leads to software reuse ? Why ? AU : IT, Dec.-07

Ans. : The object oriented model is used for the software reuse because - this model is based on the incremental development of the software product. This can be done in one or more iterations.

Q.18 State the benefits of waterfall life cycle model for software development. AU : IT, May-08

Ans. : 1. The waterfall model is simple to implement.

2. For implementation of small systems the waterfall model is used.

Q.19 How does "Project Risk" factor affect the spiral model of software development ? AU : CSE, May-09

Ans. : The spiral model demands considerable risk assessment because if a major risk is not uncovered and managed, problems will occur in the project and then it will not be acceptable by end user.

Q.20 Define software. AU : IT, Dec.-09

Ans. : Software is a collection of computer programs and related documents that are intended to provide desired features, functionalities and performance. A software can be of two types - 1. Generic software and 2. Custom software.

Q.21 Write down the generic process framework activities that is applicable to any software project. (Refer section 1.5.1) AU : Dec.-10

Q.22 What is software process model ? On what basis it is chosen ?

Ans. : The software process model can be defined as abstract representation of process. It is based on nature of software project.

Q.23 What is software process ? AU : May-13, Dec.-19

Ans. : Software process can be defined as the structured set of activities that are required to develop the software system. The fundamental activities are -

1. Specification 2. Design and Implementation 3. Validation 4. Evolution

Q.24 Write the process framework and Umbrella activities AU : May-15

Ans. : Process Framework : Process framework is required for representing the common process activities. The process framework activities are -

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

Umbrella Activities : The umbrella activities focus on project management, tracking and control. These activities are -

1. Software Project tracking and Control
2. Risk Management
3. Software Quality Assurance
4. Formal Technical Review
5. Software Configuration Management
6. Work Production Preparation and production
7. Reusability Management
8. Measurement

Q.25 What are the pros and cons of Iterative software development models ? AU : Dec.-06, 15

Ans. : Pros : 1) The changes in requirements or additions of functionality is possible.

or

2) Risks can be identified and rectified before they get problematic.

Cons : 1) This model is typically based on customer communication. If the communication is not proper the software product that gets developed will not be exactly as per the requirements.

2) The development process may get continued and never finish.

Q.26 What led to the transition from product oriented development to process oriented development to process oriented development ? AU : May -16

Ans. : The software process model led to the transition from product oriented development to process oriented development.

Q.27 Mention the characteristics of software contrasting it with characteristics of hardware.

AU : May -16

Ans. : 1) Software is engineered and not manufactured.

2) Software does not ware out. 3) The software is custom built rather than being assembled from components.

Q.28 If you have to develop a word processing software product, what process model will you choose ? Justify your answer.

AU : Dec.-16, May-17

Ans. : The incremental process model will be used to develop word processing software product.

Justification : 1) The working software can be generated quickly and early during the software life cycle.

2) The customers can respond to its functionalities after every increment.

Q.29 Depict the relationship between work product, task, activity and system.

AU : Dec.-16

Ans. :

- Each framework activity under umbrella activities of software process framework consists of various task sets.
- Each task set consists of work tasks, work products, quality assurance points and project milestones. The task sets accomodate the needs of the system getting developed.

Q.30 List two deficiencies in waterfall model. Which process model do you suggest to overcome each efficiency ?

AU : May-17

Ans. : i) It is difficult to define all the requirement at the beginning of project, this model is not suitable for accomodating any change.

To overcome this efficiency : Prototyping model.

ii) It does not scale up to large projects.

To overcome this efficiency : Spiral model

Q.31 If you have to develop a word processing software product, what process model will you choose ? Justify your answer.

AU : May-18

Ans. : The incremental model can be used for developing a word processing software product. This is because the basic functionality of word editing tasks can be developed and verified from the customer in the initial increments. Then in subsequent increments the advanced editing features can be added.

Q.32 Compare prototyping approaches in a software process. AU : May-18

Ans. : There are two prototyping approaches in a software process -

1) **Evolutionary Prototyping -** In this approach of system development, the initial prototype is prepared and it is then refined through number of stages to final stage.

2) **Throw-away Prototyping -** Using this approach a rough practical implementation of the system is produced. The requirement problems can be identified from this implementation. It is then discarded. System is then developed using some different engineering paradigm.

Q.33 List any two agile process models. AU : May-19

Ans. : 1) Rational Unified Process Model(RUP) 2) SCRUM.

Q.34 Define evolutionary prototype. AU : Dec.-19

Ans. : Evolutionary prototyping is a software development method where the developer or development team first constructs a prototype. After receiving initial feedback from the customer, subsequent prototypes are produced, each with additional functionality or improvements, until the final product is built.

Q.35 List out the goals of software engineering. *(Refer section 1.2)* AU : Dec.-20

Q.36 What are the various categories of software ? *(Refer section 1.1.3)* AU : Dec.-20

❑❑❑

UNIT II

2 Requirements Analysis and Specification

Syllabus

Requirement analysis and specification - Requirements gathering and analysis - Software Requirement Specification - Formal system specification - Finite State Machines - Petrinets - Object modelling using UML - Use case Model - Class diagrams - Interaction diagrams - Activity diagrams - State chart diagrams - Functional modelling - Data Flow Diagram - CASE TOOLS.

Contents

2.1 Introduction to Software Requirements

2.2 Functional and Non Functional Requirements ..... **Dec.-13,17,22, May-13,** ..... Marks 11

2.3 Requirements Engineering Process..... **May-15,17, Dec.-15,19,** ..... Marks 16

2.4 Feasibility Studies ..... **May-17,22, Dec.-17,** ..... Marks 8

2.5 Requirement Gathering and Analysis ..... **Dec.-16,22,** ..... **May-08,17,18,19,** ..... Marks 16

2.6 Software Requirement Specification..... **Dec.-11,19, 20,** ..... **May-13,14,16,19,22,** ..... Marks 16

2.7 Formal System Specification

2.8 Finite State Machines

2.9 Petri Nets ..... **Dec.-19,** ..... Marks 13

2.10 Object Modeling using UML..... **May-11,14, Dec.-18,** ..... Marks 16

2.11 Use Case Model ..... **May-11,12,13,14,18,19,** ..... **Dec.-11,12,13,15,16,17,18,**..... Marks 16

2.12 Class Diagrams ..... **May-11,12,16,17,18,19,** ..... **Dec.-11,12,13,14,15,16,17,18,**... Marks 16

2.13 Interaction Diagrams	.....	**May-11,14,18,19,**
	.....	**Dec.-13,14,15,16,17,18,** ..... Marks 16
	.....	**Dec.-13,18,** ..... Marks 16
2.14 Activity Diagrams	.....	**Dec.-11,13,14,**
2.15 State Chart Diagrams	.....	**May-11,14,19,** ..... Marks 16
	.....	**May-16,** ..... Marks 8
2.16 Functional Modelling	.....	**May-08,10,15,17,18,19,**
2.17 Data Flow Diagram	.....	**Dec.-13,15,16,17,** ..... Marks 16
2.18 Two Marks Questions with Answers		

2.1 Introduction to Software Requirements

What is requirement engineering ?

Requirement engineering is the process of

- establishing the services that the customer requires from a system
- and the constraints under which it operates and is developed.

The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.

What is a requirement ?

A requirement can range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.

The requirement must be open to interpretation and it must be defined in detail.

Types of requirements

The requirements can be classified as

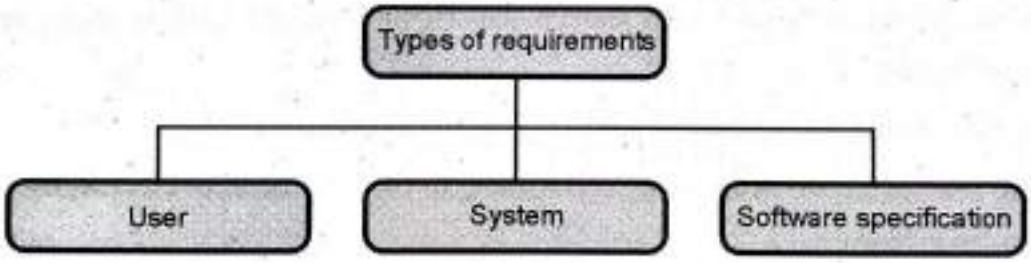


Fig. 2.1.1 Types of requirements

- **User requirements**

It is a collection of statements in natural language plus description of the services the system provides and its operational constraints. It is written for customers.

- **System requirements**

It is a structured document that gives the detailed description of the system services. It is written as a contract between client and contractor.

- **Software specification**

It is a detailed software description that can serve as a basis for design or implementation. Typically it is written for software developers.

2.2 Functional and Non Functional Requirements

AU : Dec.-13,17,22. May-13. Marks 11

Software system requirements can be classified as functional and non functional requirements.

2.2.1 Functional Requirements

- Functional requirements should describe all the required functionality or system services.
- The customer should provide statement of service. It should be clear how the system should react to particular inputs and how a particular system should behave in particular situation.
- Functional requirements are heavily dependant upon the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do but functional system requirements should describe the system services in detail.
- For example : Consider a library system in which there is a single interface provided to multiple databases. These databases are collection of articles from different libraries. A user can search for, download and print these articles for a personal study.

  From this example we can obtain functional requirements as -

  1. The user shall be able to search either all of the initial set of databases or select a subset from it.
  2. The system shall provide appropriate viewers for the user to read documents in the document store.
  3. A unique identifier (ORDER_ID) should be allocated to every order. This identifier can be copied by the user to the account's permanent storage area.

2.2.1.1 Problems Associated with Requirements

- **Requirements imprecision**
  1. Problems arise when requirements are not precisely stated.
  2. Ambiguous requirements may be interpreted in different ways by developers and users.
  3. Consider meaning of term 'appropriate viewers'.
- **User intention** - Special purpose viewer for each different document type;
- **Developer interpretation** - Provide a text viewer that shows the contents of the document.

- **Requirements completeness and consistency -**

The requirements should be both complete and consistent. *Complete* means they should include descriptions of all facilities required. *Consistent* means there should be no conflicts or contradictions in the descriptions of the system facilities.

Actually in practice, it is impossible to produce a complete and consistent requirements document.

2.2.2 Non Functional Requirements

- The non functional requirements define system properties and constraints.
- Various properties of a system can be : Reliability, response time, storage requirements. And constraints of the system can be : Input and output device capability, system representations etc.
- Process requirements may also specify programming language or development method.
- Non functional requirements are more critical than functional requirements. If the non functional requirements do not meet then the complete system is of no use.

2.2.2.1 Types of Non Functional Requirements

The classification of non functional requirements is as given below. (See Fig. 2.2.1 on next page)

Product requirements

These requirements specify how a delivered product should behave in a particular way. For instance: execution speed, reliability.

Organizational requirements

The requirements which are consequences of organizational policies and procedures come under this category. For instance: process standards used implementation requirements.

External requirements

These requirements arise due to the factors that are external to the system and its development process. For instance : interoperability requirements, legislative requirements.

In short, non functional requirements arise through

i) User needs
ii) Because of budget constraints
iii) Organizational policies
iv) The need for interoperability with other software or hardware systems
v) Because of external factors such as safety regulations.

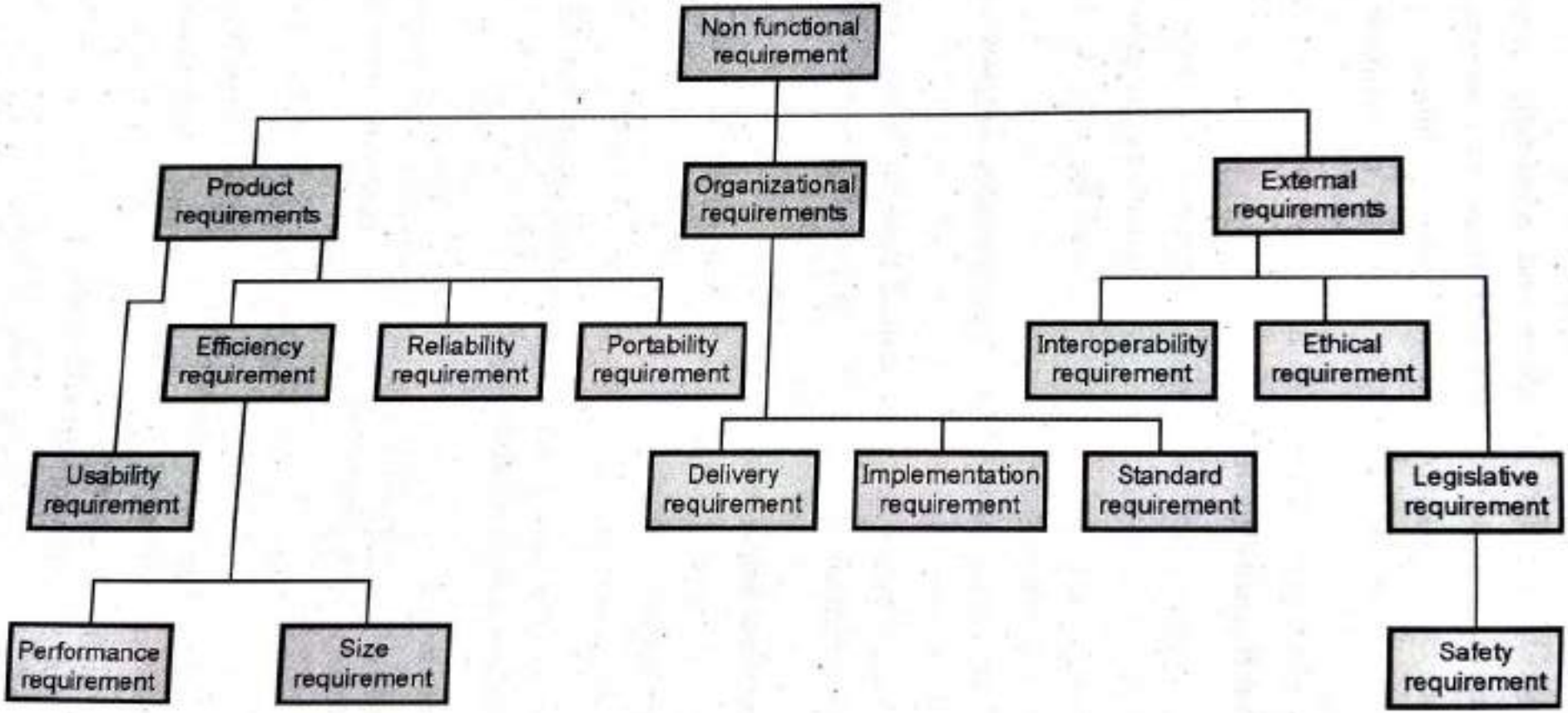


Fig. 2.2.1 Types of non functional requirement

Metrics used for specifying the non functional requirements

Property	Metric
Speed	Events per response time processed transactions per second.
Size	Kilo bytes.
Reliability	Mean time to failure. Rate of failure. Occurrence availability.
Robustness	Time to restart after failure. Probability of events causing failure.
Portability	Number of target statements.

2.2.2.2 Domain Requirements

- Domain requirements are derived from the application domain of the system instead of specific user needs.
- These requirements make use of domain terminologies specific to the existing domain concept.
- The domain requirements may be in the form of new functional requirements, constraints on existing functional requirement or guidance on how to carry out certain computation.
- These are the specialised requirements and hence software engineers find it difficult to co-relate the domain requirements with the system requirements.
- It is important to specify the domain requirements otherwise the system will not work properly.
- **Example : Domain requirements for the library system.**
- There should be user interface for handling the databases. These interfaces should be according to some international standard.
- If there is copyright restriction on some document then it should get printed locally on the server. The copies of such document should not get created.

2.2.3 Difference between Functional and Non Functional Requirements

Functional requirements	Non functional requirements
The functional requirements specify the features of the software system	The non functional requirements specify the properties of the software system.

Functional requirements describe what the product must do	Non functional requirements describe how the product should perform.
The functional requirements specify the actions with which the work is concerned.	The non functional requirements specify the experience of the user while using the system.
Example : For a library management system, allowing user to read the article online is a functional requirement.	Example : For a library management system, for a user who whishes to read the article online must be authenticated first.

Metrics used for specifying Non Functional Requirements

The metrics used for specifying the non functional requirements are -

1. **Reliability :** The application should be highly reliable and it should generate all th updated information in correct order.
2. **Availability** : Any information must be quickly available from computer to th authorized user.
3. **Security :** The application must be password protected. This feature is essential t avoid any unauthorized access to the application.
4. **Maintainability :** The application should be maintained in such a manner that i any change in requirement occurs then it should be easily incorporated in a individual module.
5. **Extensibility :** The application should be maintained in such a manner that if an new requirement occurs then it should be easily incorporated in an individua module.
6. **Portability :** The application should be portable on the desired operating system.
7. **Reusability :** The application should have the ability that a segment of source cod can be used again to add new functionalities with slight or no modification Reusable modules and classes reduce implementation time.
8. **Resource utilization** : The application should make use of resources to its maximum capability.

Example 2.2.1 *For the requirement given below, identify stakeholders, functional and non-functional requirements*

A software is to be built that will control an Automated Teller Machine (ATM). The ATM machine services customers 24 × 7. ATM has a magnetic stripe reader for reading an ATM card, a keyboard and display for interaction with the customer, a slot for depositing envelopes, a dispenser for cash, a printer for printing receipts and a switch that allows an operator to start/stop a machine.

The ATM services one customer at a time. When a customer inserts an ATM card and enters the Personal Identification Number (PIN), the details are validated for each transaction. A customer can perform one or more transactions. Transactions made against each account are recorded so as to ensure validity of transactions.

If PIN is invalid, customer is required to re-enter PIN before making a transaction, If customer is unable to successfully enter PIN after three tries, card is retained by machine and customer has to contact bank.

The ATM provides the following services to the customer :

1) Withdraw cash in multiples of 100

2) Deposit cash in multiples of 100

3) Transfer amount between any two accounts.

4) Make balance enquiry.

5) Print receipt.

Each of the above transactions must be made within 60 seconds in case the time exceeds 60 seconds, then the transaction is cancelled automatically. Also, if the machine is not used for more than two minutes after entry of card, the card is retained by the machine. An operator panel with a key-operated switch allows an operator to start and stop the servicing of customers. When the switch is moved to the "off position, the machine will shut down, so that the operator may remove deposit envelopes and reload the machine with cash, blank receipts, etc. The operator is required to verify and enter the total cash on hand before starting the system from this panel.

AU : Dec.-13, Marks 11

Solution : i) Stakeholders

1. Bank customer
2. Service operator
3. Hardware and software maintenance engineers
4. Database administrators
5. Banking regulators
6. Security administrator

ii) Functional requirements

1. There should be the facility for the Customer to insert a card.
2. The system should first validate card and PIN.
3. The system should allow the customer to deposit amount in the bank.
4. The system should dispense the cash on withdrawal.
5. The system should provide the printout for the transaction.
6. The system should make the record of the transactions made by particular customer.
7. On invalid PIN entry for three times the card should be retained by the system.
8. The cash withdrawal is allowed in multiple of 100.
9. The cash deposition is allowed in multiple of 100.
10. The customer is allowed to transfer amount between the two accounts.
11. The customer is allowed to know the balance enquiry.
12. The customer is allowed to get the printout for desired transaction.
13. The system should be efficient.

iii) Non functional requirements

1. Each of the transaction should be made within 60 seconds. If the time limit is exceeded ,then cancel the transaction automatically.
2. If there is no response from the bank computer after request is made within the minutes then the card is rejected with error message.
3. The bank dispenses money only after the processing of withdrawal from the bank. That means if sufficient fund is available in user's account then only the withdrawal request is processed.
4. Each bank should process the transactions from several ATM centers at the same time.
5. The machine should be loaded with sufficient fund in it.

Example 2.2.2 *List the stakeholders and all types of requirements for an online train reservation system.* **AU : Dec.-17, Marks 7**

Solution :

1) Passenger
2) Database Administrator
3) Booking Clerk
4) Bank

Functional requirements

1) The online booking made by the customer should be associated with the account.
2) Booking confirmation must be sent to the user on specified contact details.
3) The system should provide a search option to the user so that user can search for required train and for required number of reservations.
4) The system should allow the user to make online payments.
5) The system should allow user to book for tickets.
6) The system should allow the user to cancel the booking and half of the amount paid by the customer must be refunded to him.

Non functional requirements

1) The system must be available to the user for 24*7.
2) The system should accept the payments via different payment methods like PayPal, Wallets, Cards, vouchers etc.
3) Use of captcha and encryption to avoid bots from booking tickets.
4) The response time of the system while searching, booking or cancellation operation should be very less.

Example 2.2.3 *Illustrate the basic issues in non-functional requirement and how it is more critical than functional requirements.* **AU : Dec.-22, Marks 6**

Solution : Basic issues : Refer to Metrics used for non-functional requirements from section 2.2.3

In many cases, the non-functional requirements are more critical than the functional requirements. The non-functional requirements define the success and acceptance of the system. Users may compromise on some functional requirements but they can not compromise system performance, reliability, security and user-friendliness. Failure to these requirements leads to more operational costs and even business failure.

Review Question

1. *Explain the metrics used for specifying non functional requirements.* **AU : May-13, Marks 8**

2.3 Requirements Engineering Process

AU : May-15, 17, Dec.-15,19, Marks 16

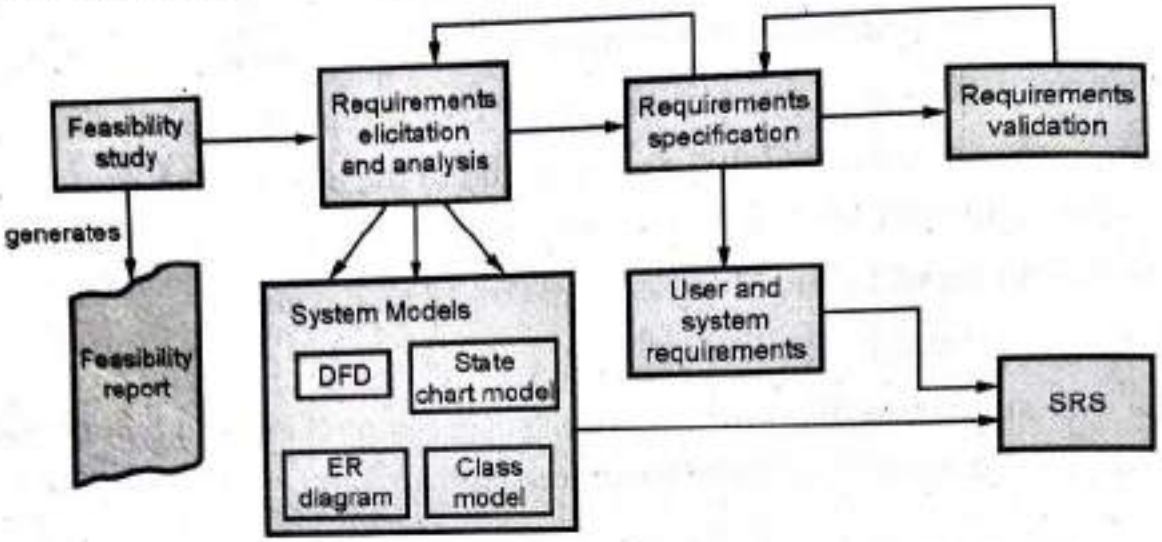


Fig. 2.3.1 Requirement engineering process

- A Requirement Engineering is a process in which various activities such as **discovery, analysis** and **validation** of system requirements are done.

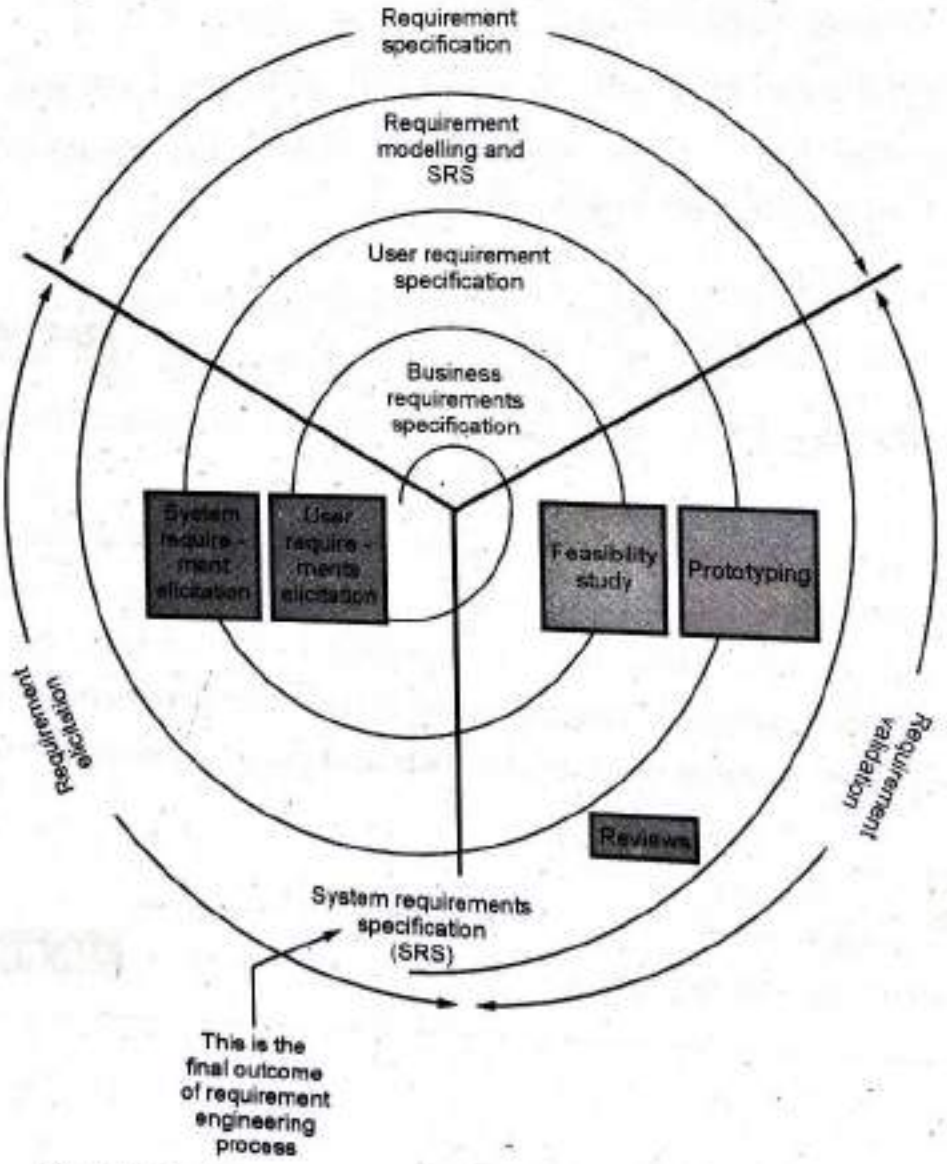


Fig. 2.3.2 Spiral model of requirements engineering process

- It begins with feasibility study of the system and ends up with requirement validation. Finally the requirement document has to be prepared.
- This process is a three stage activity where the activities are arranged in the iterative manner. In the early stage of this process most of the time is spent on understanding the system by understanding the high-level non functional requirements and user requirements. The spiral model of requirement engineering process is as shown in Fig. 2.3.2.

Requirement engineering process performs following **seven distinct functions** -

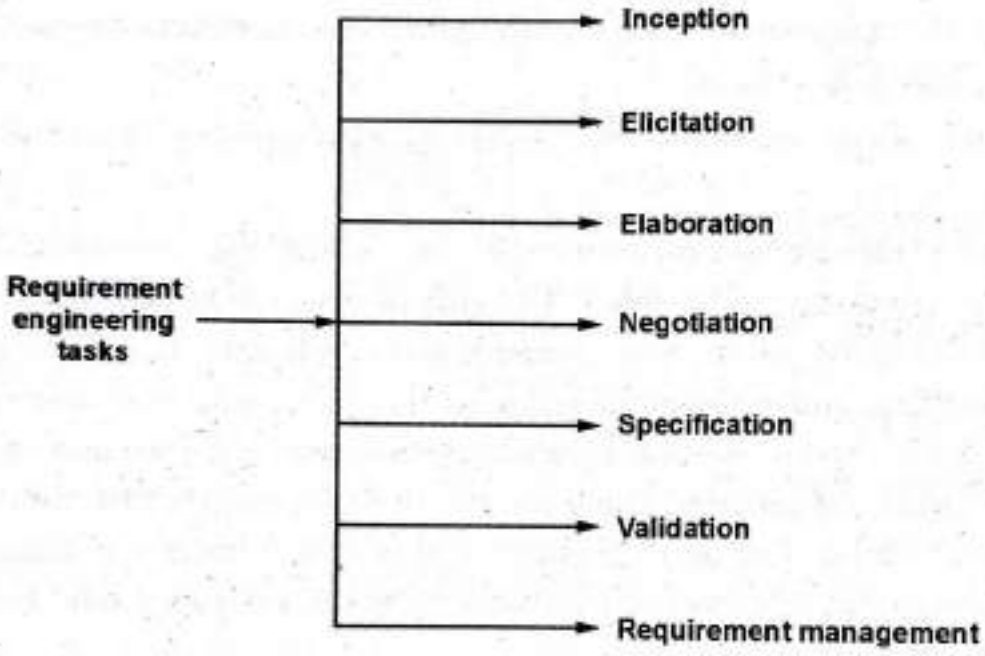


Fig. 2.3.3 Requirement engineering tasks

Let us now discuss these tasks in detail -

Inception : The inception means specifying the **beginning** of the software project. Most of the software projects get started due to business requirements. There may be potential demand from the market for a particular product.

The purpose of inception is to -

1. Establish the basic understanding of the project.
2. Find out all possible solutions and to identify the nature of the solution.
3. Establish an effective communication between developer and the customer.

Elicitation : Requirements elicitation means **requirements discovery.** Requirements elicitation is very difficult task.

Following are the reasons for : ***why it is difficult to understand customer wants ?*** -

1. Customer sometimes is unable to specify the **scope of the project.** Sometimes customers specify too many technical details and this may increase the confusion.

2. There is difficulty in **understanding the problem.** Sometimes customer could not decide what are their needs and wants. Sometimes they have got poor understanding of capabilities and limitations of the existing computing environment.

Sometimes customers find it **difficult to communicate** with the system engineer about their needs. Sometimes customers may have got **some conflicting requirements.** This ultimately results in specifying **ambiguous requirements.**

3. As project progresses the needs or requirements of the customers changes. This creates a problem of volatility.

To overcome these problems the requirements gathering must be done very systematically.

Elaboration : Elaboration is an activity in which the information about the requirements is **expanded** and **refined**. The **goal** of elaboration activity is to prepare a technical model of software functions, features and constraints. The elaboration consists of several **modelling and refinement tasks**. In this process several **user scenarios** are created and refined. During elaboration, each user scenario is parsed and various **classes** are identified. These classes are nothing but the business entities that are visible to end user. Then the **attributes** and **services** (functions) of these classes are defined. Then the relationship among these classes is identified. Finally the **analysis model** gets developed during the elaboration phase.

Negotiation : Sometimes customer may **demand for more** than that is achieved or there are certain situations in which customer demands for something which cannot be achieved in **limited business resources**. To handle such situations requirement engineers must convince the customers or end users by solving various **conflicts**.

Specification : A specification can be a **written document, mathematical** or **graphical model, collection of use case scenarios** or may be the **prototypes.** There is a need to develop a **standard specification** in which requirements are presented in consistent and understandable manner.

Validation : Requirement validation is an activity in which requirement specification is analyzed in order to ensure that the requirements are specified unambiguously. If any inconsistencies, omissions and errors are identified then those are corrected or modified during the validation.

Requirement management : Requirements management is the process of managing changing requirements during the requirements engineering process and system development.

Example 2.3.1 *What does win-win mean in the context of negotiation during requirements engineering activity ?* **AU : Dec.-19, Marks 4**

Solution : From customer point of view the win situation is when the customer gets satisfied when most of its requirements or expectations from the system gets satisfied. From developers point of view the win situation is when the system can be developed within the given budget and within the given schedule.

Review Questions

1. *What is requirements engineering ? Explain in detail the various processes in requirements engineering.* **AU : May-17, Marks 13**
2. *Write about the following requirements engineering activities.*
   (i) Inception (ii) Elicitation (iii) Elaboration (iv) Negotiation (v) Specification
   (vi) Validation (vii) Requirements management **AU : May-15, Marks (2+3+3+2+2+2+2)**
3. *Explain the software requirement engineering process with neat diagram.* **AU : Dec.-15, May-17, Marks 16**
4. *Discuss the distinct tasks involved in requirement engineering process.* **AU : Dec.-19, Marks 9**

2.4 Feasibility Studies

AU : May-17,22, Dec.-17, Marks 8

Definition : *A feasibility study is a study made to decide whether or not the proposed system is worthwhile.*

The **focus** of feasibility study is to check

- If the system contributes to **organizational objectives.**
- If the system can be engineered using **current technology.**
- If the system is within the given **budget.**
- If the system can be **integrated** with other useful systems.

The implementation of feasibility study is based on the information assessment (what is required), information collection and report writing.

While performing the feasibility study, following questionnaires to the people in the organization should be asked -

1. What if the system wasn't implemented ?
2. What are current process problems ?
3. How will the proposed system help ?
4. What will be the integration problems ?
5. Is new technology needed? With what skills ?
6. What facilities must be supported by the proposed system ?

Feasibility study should be done with the help of project managers who is going to handle that particular project, software engineers who are about to develop that system, technical experts and customers who will be using the system. Typically the feasibility study should be completed within **two to three weeks.**

Finally the **feasibility study report** has to be prepared.

Types of Feasibility Study

1. **Technical Feasibility** : It is the measure of technical resources such as hardware components, software techniques and skilled persons.
2. **Economical Feasibility** : It is the measure whether finances or funds are available for proposed system.
3. **Operational Feasibility** : It is a measure of how well the solution of problems or a specific alternative solution will work in the organization.
4. **Schedule Feasibility** : It is the establishment of time limit for completion of the project. This kind of feasibility is dependent upon available manpower and economical support for the project.

Thus the feasibility study helps in understanding the requirements of the system. Hence feasibility study affects the requirement collection implicitly or explicitly.

Review Questions

1. *Explain the feasibility studies. What are the outcomes ? Does it have implicit or explicit effects on software requirement collection ?* **AU : May-17, Marks 8**
2. *What is feasibility study ? How it helps in requirement engineering process ?* **AU : Dec.-17, Marks 3**
3. *What is feasibility study ? What are the feasibilities to be analyzed in software development ?* **AU : May 22, Marks 6**

2.5 Requirement Gathering and Analysis

AU : Dec.-16,22, May-08,17,18,19, Marks 16

After performing feasibility study the requirements elicitation and analysis can be done. Requirement **elicitation means discovery** of all possible requirements. After identifying all possible requirements the analysis on these requirements can be done. Software engineers communicate the end-users or customers in order to find out certain information such as: application domain, expected services from the system, the expected performance level of the system. From this information even constraints of the system can be decided.

2.5.1 Stakeholders

This is a commonly used term in software engineering. The stakeholder means the person(s) who will be affected by the system. For example : end-user, system maintenance engineers or software engineers can be stakeholders. Following is the list of **problems** encountered in understanding the requirements of the system.

Problems	Description
Unrealistic expectations	Stakeholders sometimes unable to specify what they want exactly. Sometimes they specify unrealistic demands.
Differences in the requirements	Different stakeholders specify different requirements. The requirement engineer has to resolve the conflicts in the requirements with proper communication with the stakeholders
Economic and business environment	The economic and business environment is dynamic due to which there may be change in the requirements or change in the stakeholders. Both these things may affect the requirement analysis process.
Political changes	Sometimes political factors affect heavily on the need for the system. Hence there may be change in the requirements or stakeholders which ultimately affect the requirement elicitation and analysis.

2.5.2 Requirement Elicitation and Analysis Process

The spiral model shown in Fig. 2.5.1 depicts the requirement elicitation and analysis process. (See Fig. 2.5.1 on next page)

The process activities are -

1. Requirement discovery : By having effective communication with the customers the requirements can be identified.

2. Requirements classification and discovery : All the unstructured requirements can be categorised systematically depending upon their nature. And they are arranged in groups.

3. Requirement prioritisation : There are some conflicting requirements. Hence the requirements are prioritised first. If there are some unrealistic requirements then negotiations are made and only realistic prioritised requirements are collected. If any conflict occurs then it its resolved by requirement engineers.

4. Requirement documentation : This is the specification of all the requirements. And an important requirement document is created.

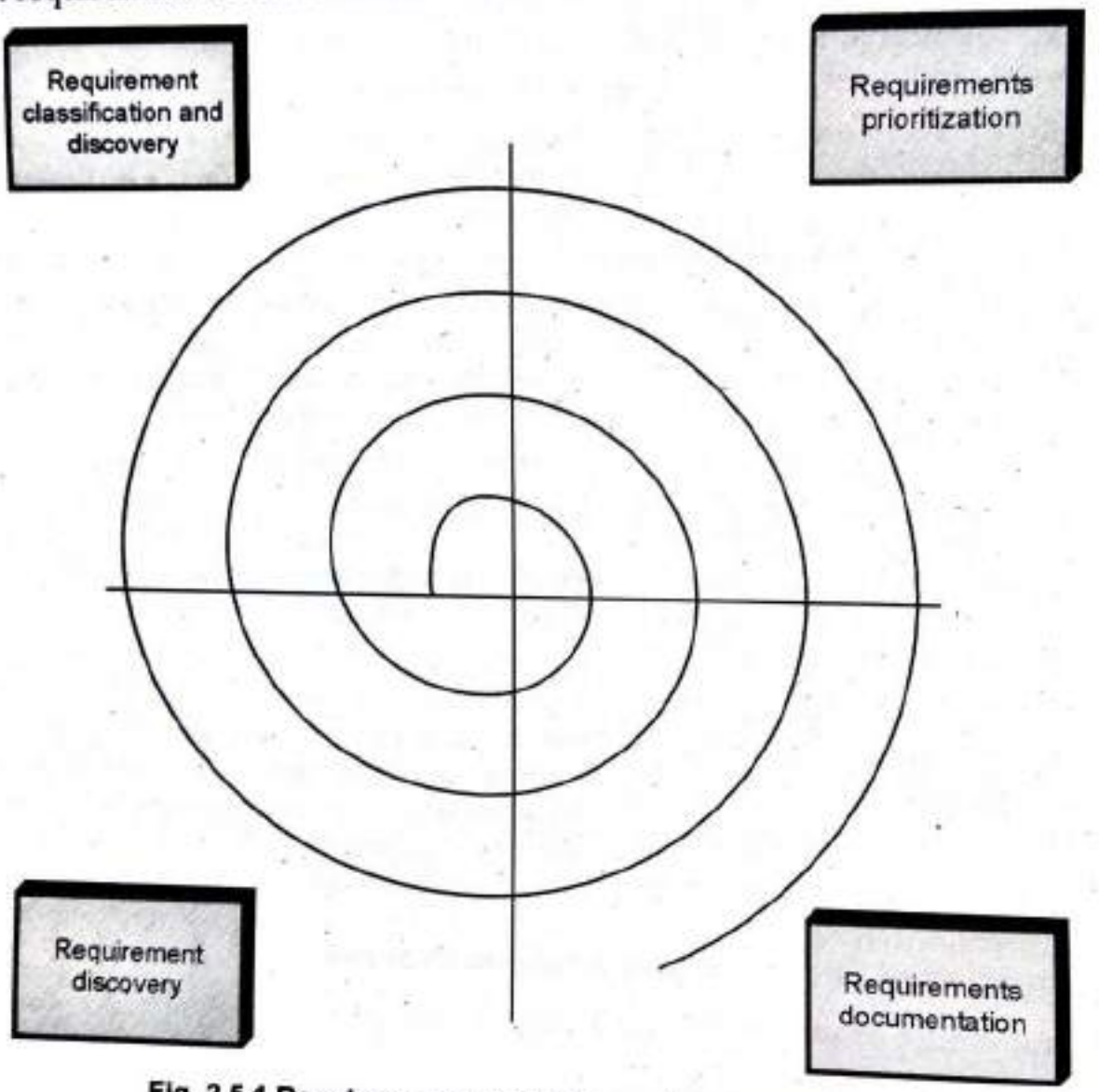


Fig. 2.5.1 Requirements elicitation and analysis process

2.5.3 Requirement Discovery

Requirement discovery means finding all relevant information about the system. The **sources of information** for requirement gathering are: documentation, system stakeholders and specification some other system which is of similar kind. Various **methods** of requirement discovery are conducting interviews, observations, creating use case scenarios. Let us discuss **various methods** of requirements discovery.

1. Interviewing

This is an effective method of requirement gathering. The requirement engineering team communicates the stakeholders by asking them various questions about the system and its use. From the answers the requirements can be identified. There are two types of interviews. (Refer Fig. 2.5.2.)

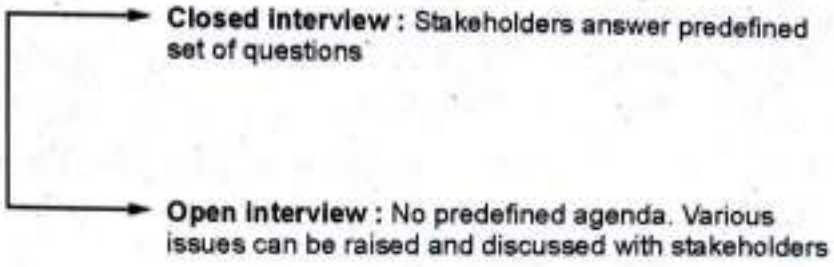


Fig. 2.5.2 Types of interview

Interviews are useful for understanding stakeholders but they are not much useful for understanding the application domain.

Characteristics of effective interviews

1. The interviews should be conducted in a free environment and they should be conducted with open minded approach. The requirement engineers should listen to stakeholders with patience. Similarly if stakeholders are expecting some unrealistic things about the system they should be ready to change their mind and ideas about the system.
2. Interviewee should start the discussion by asking questions and the requirements should be gathered together.

2. Scenario

- Scenario is the sequence of interactions made by the user with the software systems. Requirements engineers can use the information gained from this scenarios to formulate actual system requirements. The scenarios describe the interaction sessions.
- Each scenario covers one or more interaction sessions.
- Each scenario includes -
  - A description of what the system and the users want at the beginning of the scenario
  - A description of normal flow of events
  - A description of something that went wrong and handling of it
  - A description of other activities occurring in parallel
  - A description of finish state

For example : **Scenario for an article downloading in the library system.**

Initial Assumption	User logs in to the library system and searches for the desired article
Normal flow of events	1. When user find the required article and clicks the button for downloading the article, he might be displayed for either provide the subscriber information or request for payment for purchasing the article. 2. User either submits the subscription information or pays online for the article. 3. He might be displayed with copyright information and might be requested to fill up the copyright related form 4. User fills up the form and submits it. 5. The user is authenticated an required document can be presented to the user in pdf form. 6. User might select the print option for printing the required article
What went wrong	1. User fails to pay for the article 2. User makes mistakes in filling up the copyright form Both these events are handled by rejecting the download request.
Other activities	User can download other articles He might search for other articles
Finish State	User gets log out the library system.

3. View point

Viewpoint provides the perspective to the system and using these perspectives the requirements can be discovered. The viewpoint is also useful in classifying the stakeholders. Following are the types of viewpoints -

Interactor viewpoint : This viewpoint is useful for finding the interaction one system with other system. For example in ATM system the user of ATM is the interactor for the bank.

Indirect viewpoints : The user who is not using the system directly but its existence reflects on the requirements of the system. Such stakeholders form indirect viewpoint.

Domain viewpoints : The domain characteristics and constraints that affect on requirements of the system sets the domain viewpoints. For example in Library system the rules that are to be followed for reserving the book(the fine or dues should be paid already or book must be returned within 1 week etc.) form the domain view point.

From these viewpoints the requirements can be discovered. From interactor viewpoint the system requirements can be obtained. The indirect viewpoints help in getting the organizational requirements and constraints. The domain requirements provide domain constructs that need to be applied for the system.

View points in library management system is given by following Fig. 2.5.3

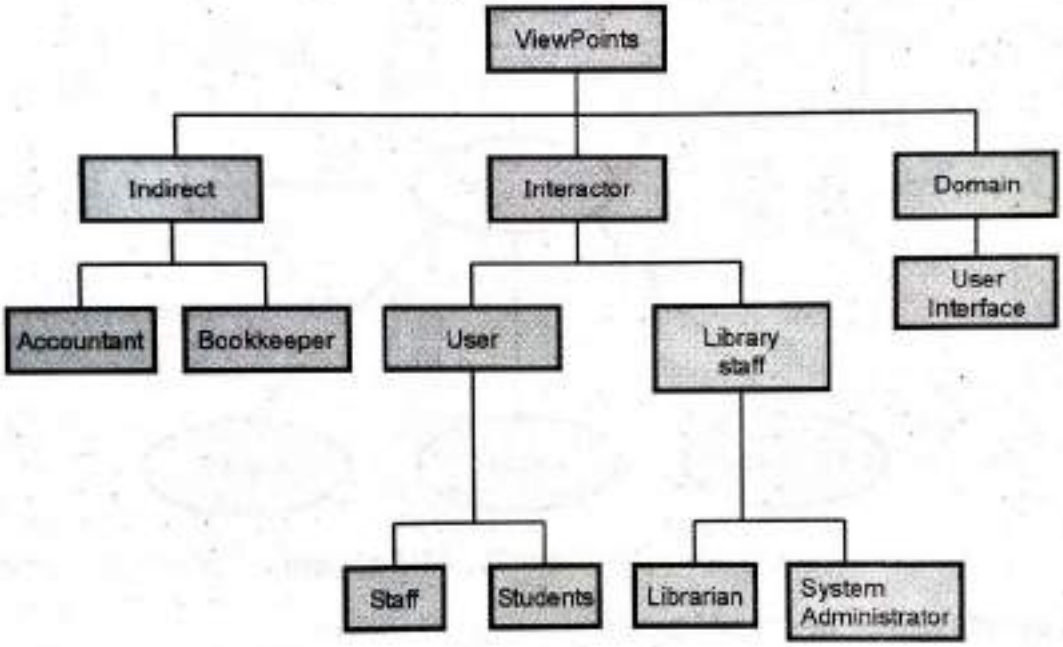


Fig. 2.5.3 Viewpoints for library system

4. Use cases

Use cases are the fundamental units of modelling language, in which functionalities are distinctly presented. The use case is a scenario based technique. Use cases help to identify individual interactions with the system. Use-cases are extensively used for requirements elicitation. By designing the proper use cases for different scenarios major reuirements of the system can be identified. The typical notations used in the use cases are as shown in Fig. 2.5.4.

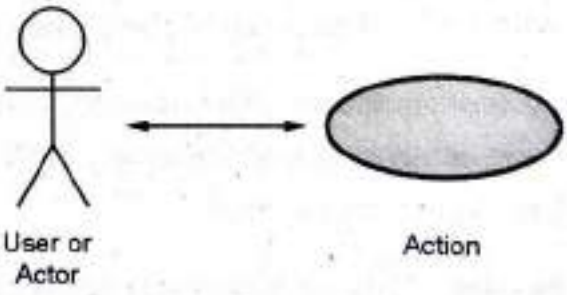


Fig. 2.5.4 Use-case notation

The use case for ATM system is as shown below.

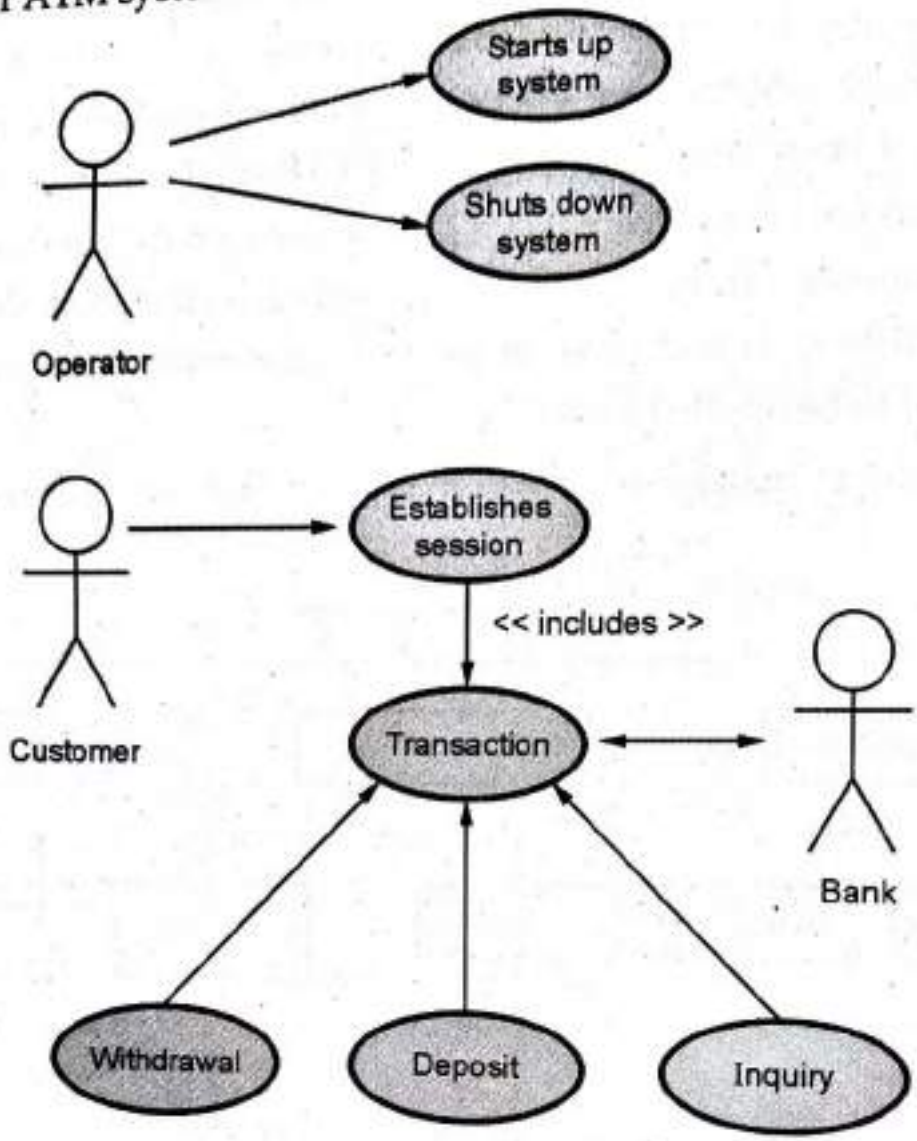


Fig. 2.5.5 Use cases for ATM system

Flow of events can be described as

System start up : The system is started when the operator turns on the switch. Initially the operator has to enter some amount of money in the cash dispenser. The connectio with the bank gets established. Then only the servicing customer can begin.

System shutdown : The operator can shutdown the system only after confirming that there is no customer currently operating the ATM system. Then the connection to th bank gets disconnected.

Session : This use case starts when the user inserts the card. ATM pulls the card inside and reads it. Then further inquiry information is displayed on the display panel .

Transaction : The transaction use case is comprised of amount withdrawal, deposit and inquiry.

The interaction diagram for system startup is as given below.

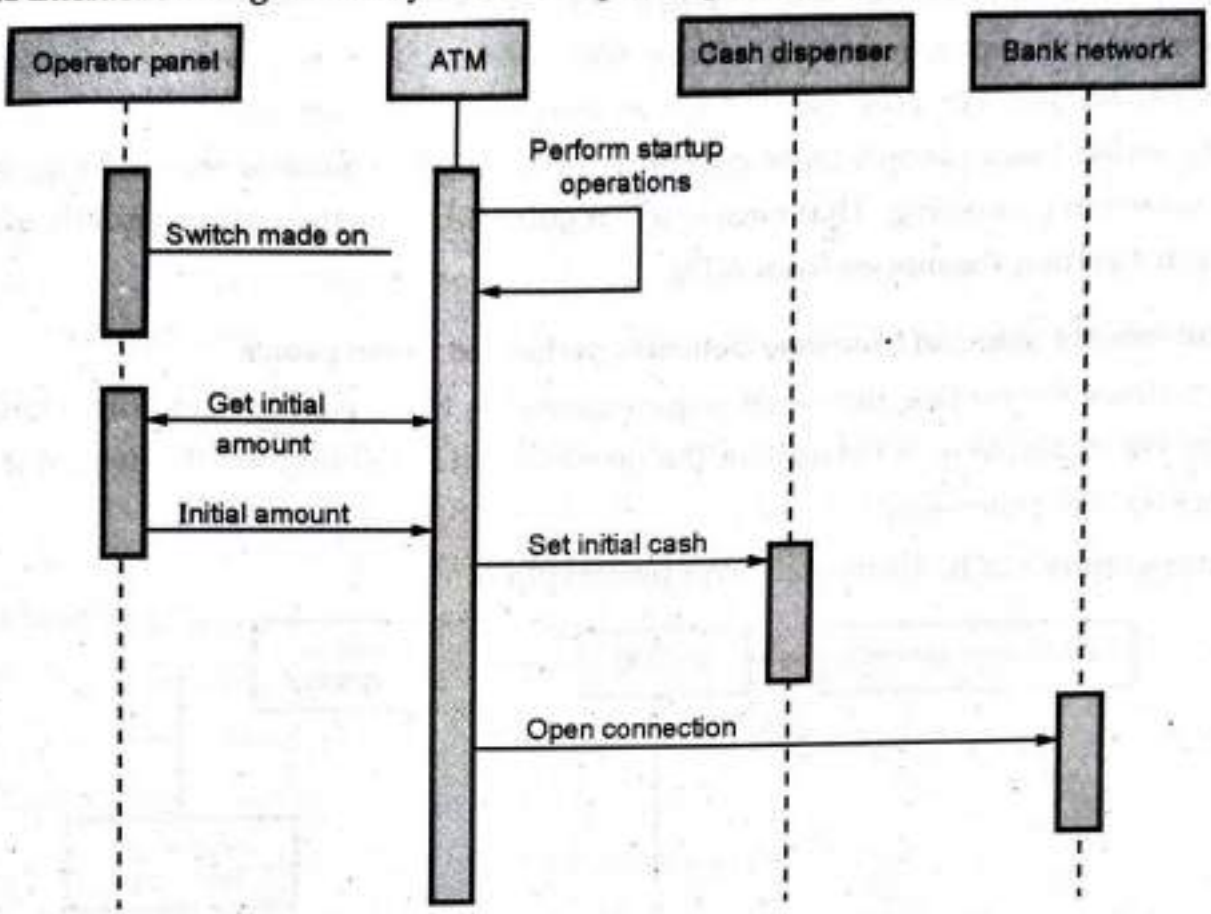


Fig. 2.5.6 Interaction diagram

This diagram depicts the sequence of operations that must be performed in order to accomplish the task.

5. Ethnography

Ethnography is a technique of observation which is used to understand social and organizational requirements.

The system analyst imagines himself as a part of the working environment in which the system will be used. He then observes the day-to-day activities and notes down the needs and tasks required by the organization. The analysts thus finds the implicit requirements of the system.

People understand their own work, but they do not understand the relationship of that work with organization. Hence by ethnographic technique such requirements can be exposed.

Two types of requirements can be discovered by ethnography.

1. Requirements obtained from working style of people

These are the requirements that can be identified from the sequence of actions that a user is performing. For example in ATM system when the user enters the PIN, the card validity action takes place. Unless and until the card gets validated there should not be any transaction processing. That means, it is required that a valid card, valid PIN must be entered for getting the money from ATM.

2. Requirements obtained from inter-activities performed by the people

Sometimes for finding the social requirements the other people's activities should be known. For example in ATM system the operator can not shutdown the system if some transaction is in processing.

Enthography can be combined with prototyping

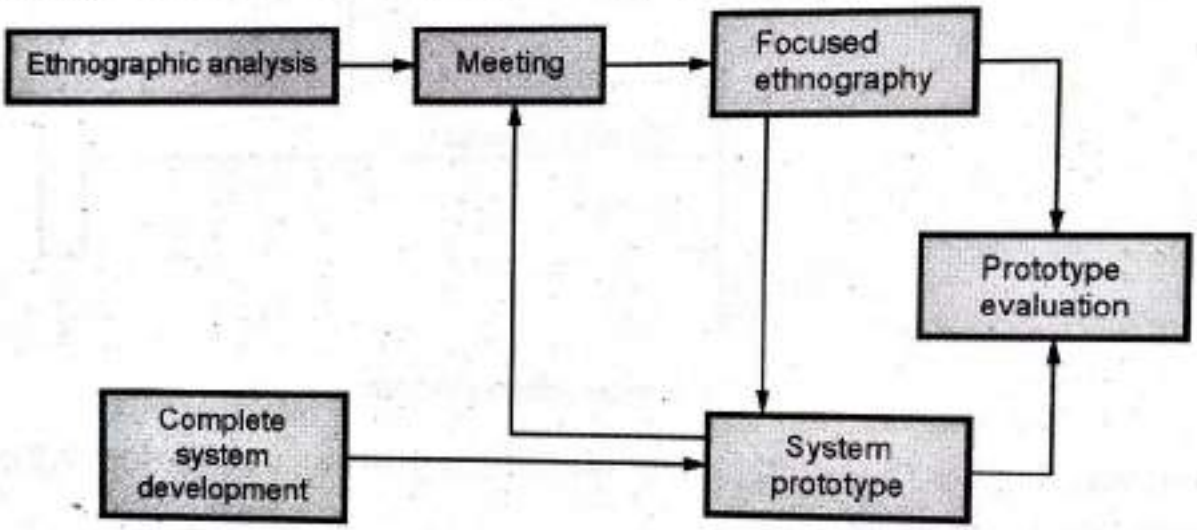


Fig. 2.5.7 Ethnographic process

The ethnography informs the development of protoype using this information prototype refinement cycles can be used.

Prototype focuses on ethnography by identifying problems and questions discussed by ethnographer.

Requirement elicitation problem

1. Stakeholders don't know what they want.
2. Stakeholders may have unrealistic expectations.
3. Stakeholders use their own language.
4. Different stakeholders may have different requirements.
5. Certain political factors may affect the requirements.
6. Business or economic changes create dynamic environment.

Example 2.5.1 *Perform requirements elicitation for watch system that facilitates to set time and alarm.*

Solution : Requirements elicitation is done for watch system by using following steps

1) **Requirements discovery :** By effective communication with customers the requirements can be identified.
2) **Requirements classification :** All unstructured requirements can be classified systematically and can be arranged in functional and non-functional requirements.
3) **Requirements prioritisation :** Some requirements are conflicting requirements. Hence they need to be priortised first. If some requirements are un-realistic, then they must be eliminated and only realistic requirements are collected.

The functional and nonfunctional requirements for watch system are -

Functional requirements

1) System allows to set time in 12:00 or 24:00 hour display format.
2) It allows to set alarm.
3) System allows to delete the already set alarm.
4) System allows to set timings for snooze.
5) System allows to stop alarm, during alarming.
6) The volume of alarm can be set by user.
7) It should allow to set more than one alarms.

Non-functional requirements

1) It should indicate battery low message if battery is low.
2) The system must be reliable and nobody should hack the system.

Review Questions

1. *What is requirements elicitation ? Briefly describe the various activities performed in requirements elicitation phase with an example of a watch system that facilitates to set time and alarm.* **AU : Dec.-16, Marks 16**
2. *What is requirement engineering ? State its process and explain requirements elicitation problem.* **AU : May-08, Marks 8**
3. *What is requirement elicitation ? Briefly describe various activities performed in requirements elicitation phase with an example of watch system that facilitates to set time and alarm.* **AU : Dec.-16, May-18, Marks 16**
4. *Write a note on what are difficulties in elicitation, requirement elicitation.* **AU : May-17, Marks 5**

5. *List any two techniques used for eliciting requirements. Compare the two techniques and list where each is applicable.* AU : May-19. Marks 13
6. *Write in detail about requirements gathering techniques.* AU : Dec.-22. Marks 7

2.6 Software Requirement Specification

AU : Dec.-11, 19, May-13,14,16,19,22. Marks 16

The software requirements document is the specification of the system. It should include both a definition and a specification of requirements. It is not a design document. As far as possible, it should set of what the system should do rather than how it should do it.

Software Requirements Specification

The software requirements provide a basis for creating the Software Requirements Specifications (SRS).

The SRS is useful in estimating cost, planning team activities, performing tasks, and tracking the team's progress throughout the development activity.

Typically software designers use IEEE STD 830-1998 as the basis for the entire Software Specifications. The standard template for writing SRS is as given below.

Document Title

Author(s)
Affiliation
Address
Date
Document Version

1. Introduction

1.1 Purpose of this document

Describes the purpose of the document.

1.2 Scope of this document

Describes the scope of this requirements definition effort. This section also details any constraints that were placed upon the requirements elicitation process, such as schedules, costs.

1.3 Overview

Provides a brief overview of the product defined as a result of the requirements elicitation process.

2. General description

- Describes the general functionality of the product such as similar system information, user characteristics, user objective, general constraints placed on design team.
- Describes the features of the user community, including their expected expertise with software systems and the application domain.

3. Functional requirements

This section lists the functional requirements in ranked order. A functional requirement describes the possible effects of a software system, in other words, *what* the system must accomplish. Each functional requirement should be specified in following manner

- **Short, imperative sentence stating highest ranked functional requirement.**

1. **Description**
   A full description of the requirement.
2. **Criticality**
   Describes how essential this requirement is to the overall system.
3. **Technical issues**
   Describes any design or implementation issues involved in satisfying this requirement.
4. **Cost and schedule**
   Describes the relative or absolute costs of the system.
5. **Risks**
   Describes the circumstances under which this requirement might not able to be satisfied.
6. **Dependencies with other requirements**
   Describes interactions with other requirements.
7. **... any other appropriate**

4. Interface requirements

This section describes how the software interfaces with other software products or users for input or output. Examples of such interfaces include library routines, token streams, shared memory, data streams, and so forth.

- **4.1 User Interfaces**
  Describes how this product interfaces with the user.

- **4.1.1 GUI**

  Describes the graphical user interface if present. This section should include a set of screen dumps to illustrate user interface features.

- **4.1.2 CLI**

  Describes the command-line interface if present. For each command, a description of all arguments and example values and invocations should be provided.

- **4.1.3 API**

  Describes the application programming interface, if present.

- **4.2 Hardware Interfaces**

  Describes interfaces to hardware devices.

- **4.3 Communications Interfaces**

  Describes network interfaces.

- **4.4 Software Interfaces**

  Describes any remaining software interfaces not included above.

5. Performance requirements

Specifies speed and memory requirements.

6. Design constraints

Specifies any constraints for the design team such as software or hardware limitations.

7. Other non-functional attributes

Specifies any other particular non functional attributes required by the system. Such as :

7.1 Security

7.2 Binary Compatibility

7.3 Reliability

7.4 Maintainability

7.5 Portability

7.6 Extensibility

7.7 Reusability

7.8 Application Compatibility

7.9 Resource Utilization

7.10 Serviceability

... others as appropriate

8. Operational scenarios

This section should describe a set of scenarios that illustrate, from the user's perspective, what will be experienced when utilizing the system under various situations.

9. Preliminary schedule

This section provides an initial version of the project plan, including the major tasks to be accomplished, their interdependencies, and their tentative start/stop dates.

10. Preliminary budget

This section provides an initial budget for the project.

11. Appendices

11.1 Definitions, Acronyms, Abbreviations

Provides definitions terms, and acronyms, can be provided.

11.2 References

Provides complete citations to all documents and meetings referenced.

2.6.1 Characteristics of SRS

Various characteristics of SRS are

- **Correct** - The SRS should be made up to date when appropriate requirements are identified.
- **Unambiguous** - When the requirements are correctly understood then only it is possible to write an unambiguous SRS.
- **Complete** - To make the SRS complete, it should be specified what a software designer wants to create a software.
- **Consistent** - It should be consistent with reference to the functionalities identified.
- **Specific** - The requirements should be mentioned specifically.
- **Traceable** - What is the need for mentioned requirement? This should be correctly identified.

2.6.2 Example of SRS

Software Requirements Specification For Attendance Maintenance System

Prepared by Anjali

December 1, 2014

Release 1.0

Version 1.0

Table of Contents

1. Introduction 1
   1.1 Purpose 1
   1.2 Scope 1
   1.3 Overview 1
2. General Description 1
   2.1 User Manual 1
3. Functional Requirements 1
   3.1 Description 1
   3.2 Technical Issues 2
4. Interface Requirements 2
   4.1 GUI 2
   4.2 Hardware Interface 3
   4.3 Software Interface 3
5. Performance Requirements 3
6. Design Constraints 4
7. Other Non-functional Attributes 4
   7.1 Security 4
   7.2 Reliability 5
   7.3 Availability 5
   7.4 Maintenability 5
   7.5 Reusability 5
8. Operational Scenarios 5
9. Preliminary Schedule 5
   6

1. Introduction

1.1 Purpose

This document gives detailed functional and non functional requirements for attendance maintenance system. The purpose of this document is that the requirements mentioned in it should be utilized by software developer to implement the system.

1.2 Scope

This system allows the Teacher to maintain attendance record of the classes to which it is teaching. With the help of this system Teacher should be in a position to send e-mail to the students who remain absent for the class. The system provides a cumulative report at every month end for the corresponding class.

1.3 Overview

This system provides an easy solution to the Teacher to keep track of student attendance, and statistics.

2. General description

This attendance maintenance system replaces the traditional, manual attendance system by which a lot of paper work will be reduced. The Teacher should able to view photograph of a student along with his attendance in his Laptop. This is the primary feature of this system. Another feature is that Teacher can be allowed to edit particular record at desired time. The system should produce monthly attendance report. And there should be facility to send an e-mail/warning to the student remaining absent in the class.

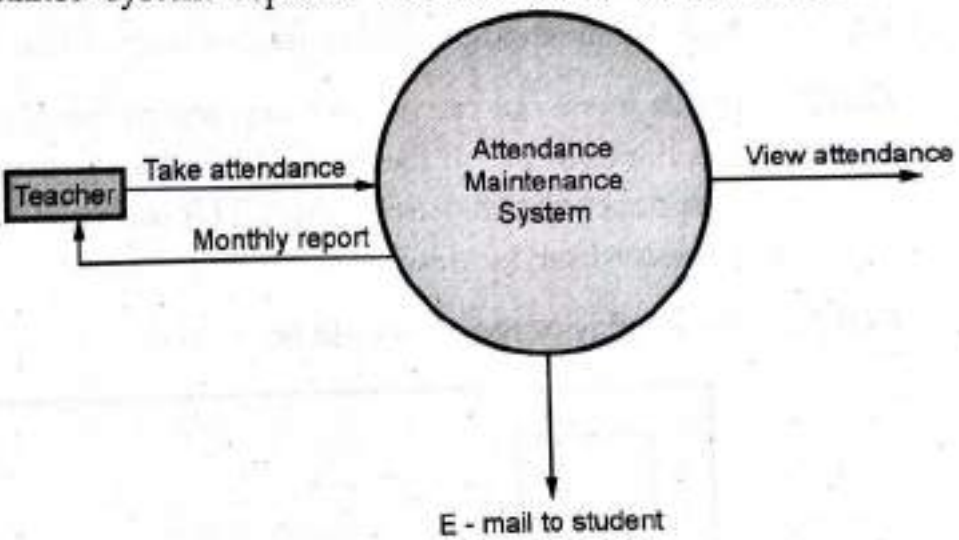


Fig. 2.6.1

Every Teacher should have Laptop with wireless internet connection. A Teacher may teach to different classes and a separate record for the corresponding classes should be maintained.

2.1 User Manual

The system should provide **Help** option in which how to operate the system should be explained. Also hard copy of this document should be given to the user in a booklet form.

3. Functional Requirements

3.1 Description

The identity of student is verified and then marked present at particular date and Time. The system should display student photograph along with their names for that corresponding class. The student may be marked present or absent depending upon his presence. The system should send e-mails to absent students. A statistical report should display individual's report or a cumulative report whenever required.

3.2 Technical Issues

The system should be implemented in VC++

4. Interface Requirements

4.1 GUI

GUI 1 : Main menu should provide options such as File, Edit, Report, Help.

GUI 2 : In File menu one can create a new record file or can open an existing record file. For example : If file *SECOMP_SEMI_07* is opened then we may view attendance record of SE COMPUTER class in year 2007 and a record for semester-I can be viewed.

GUI 3 : The display of record should be

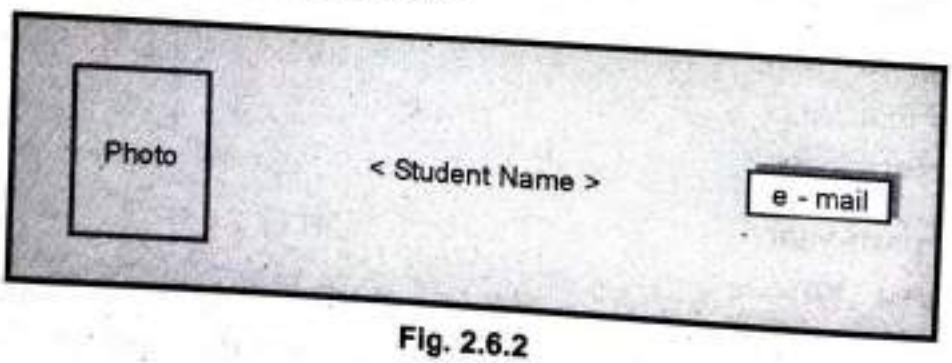


Fig. 2.6.2

The photo can be clicked to mark student present for particular class. The **e-mail** button can be clicked to send e-mail to the student being absent.

GUI 4 : Report option should display statistical report. It may be for particular student or for the whole class.

GUI 5 : Help option should describe functionality of the system. It should be written in simple HTML.

4.2 Hardware Interface

Hardware Interface 1 : The system should be embedded in the laptops.

Hardware Interface 2 : The laptop should use wireless Ethernet card to send e-mails. These Laptops should be the clients of departmental database server.

4.3 Software Interface

Software Interface 1 : Attendance maintenance system.

Software Interface 2 : The attendance database should be transmitted to departmental database server.

Software Interface 3 : E-mail message generator which generates standard message of absence.

Software Interface 4 : Report generators

5. Performance requirements

This system should work concurrently on multiple processors between the college hours. The system should support 50 users.

The e-mail should be sent within one hour after the class gets over.

The system should support taking attendance of maximum 100 students per class.

6. Design constraints

The system should be designed within 6 months.

7. Other non-functional attributes

7.1 Security

The teacher should provide password to log on to the system. He/she should be able to see the record of his/her class.

The password can be changed by the Teacher by providing existing password.

7.2 Reliability

Due to wireless connectivity , reliability can not be guaranteed.

7.3 Availability

The system should be available during college hours.

7.4 Maintenability

There should be a facility to add or delete or update Teachers and students for each semester.

7.5 Reusability

The same system will be used in each new semester.

8. Operational scenarios

There will be student database, Teacher database. The student database will contain students name, class, attendance, e-mail address, address, phone number.

The Teacher database will contain Teacher's name, classes taught, e-mail address phone number.

9. Preliminary schedule

The system has to be implemented within 6 months.

Importance of SRS : The software requirements provide a basis for creating the Software Requirements Specifications (SRS).

The SRS is useful in estimating cost, planning team activities, performing tasks, and tracking the team's progress throughout the development activity.

Example 2.6.1 *An independent trunk company wants to track and record its drivers driving habits. For this purpose the company has rented 800 phones numbers and has printed the numbers in the front, back and sides of all trucks owned by the company. Next to the 800 numbers a message is written "PLEASE REPORT ANY DRIVER OR TRUCK PROBLEM BY CALLING THIS NUMBER". The hacking company waits for you to develop a system that :*

i) Collects information from caller about the driver performance and behaviour as well as truck condition,

ii) Generates daily and monthly reports for each driver and truck management.

iii) Reports problems that require immediate action to an on-duty manager.

Analyse the problems that statement and list major functions to be incorporated with the SRS document.

AU : Dec.-11, Marks 8

Solution : The major functions to be included with SRS document are -

i) A) Creation of databases such as drivers database, trucks database, callers database.

B) Updation for above created databases.

C) Display and searching operations on databases

ii) A) Report generation (Daily) of trucks and drivers.

B) Report generation (Monthly) of trucks and drivers.

iii) A) Sending of SMS to corresponding driver about the problem.

B) Updation for duty hours if necessary.

C) Deletion of particular record (For truck database) if the truck is out of condition.

D) Insertion of new record (New truck arrival).

Example 2.6.2 *Develop the software requirements document for the following requirement. A coffee vending machine serves coffee to customers. A customer can choose a type of coffee among a list often options, supply the amount required and get served. Each coffee is prepared by adding units of hot water, coffee powder, milk and sugar. The recipe for each coffee is stored.* **AU : May-19, Marks 13**

Solution :

1. Introduction

1.1 Purpose

This document describes the interaction between the user and Coffee Vending Machine(CVM). The purpose of this document is that the requirements mentioned in it should be utilized by software developer to implement the system.

1.2 Scope

This system allows the customer to interact the vending machine. The purpose of the CVM is to accept payment for the customer, dispense coffee (this will have the options of added milk and / or sugar) and return any excess money to the customer as change.

1.3 Overview

This system dispenses the coffee of a user's choice.

2. General description

Coffee vending machine is a type of vending machine that dispenses hot coffee. Some of the machines, particularly older models, utilize powdered instant coffee mixed with hot water, and some of these offer condiments such as cream and sugar. Some models fresh-brew the coffee using hot water and ground coffee beans, and some also grind the coffee to order using coffee grinders installed in the machines. Some modern machines also provide other hot drinks such as tea, espresso, lattes, cappuccinos, mochas and hot chocolate. Some of the machines dispense canned coffee, and some dispense both hot coffee and iced coffee.

These coffee vending machines typically require payment, functioning as coin-operated machines.

2.1 Application Document

The document describes the interaction between the customer and the CVM. This document also provides the information about maintenance of the system.

3. Functional requirements

3.1 Description

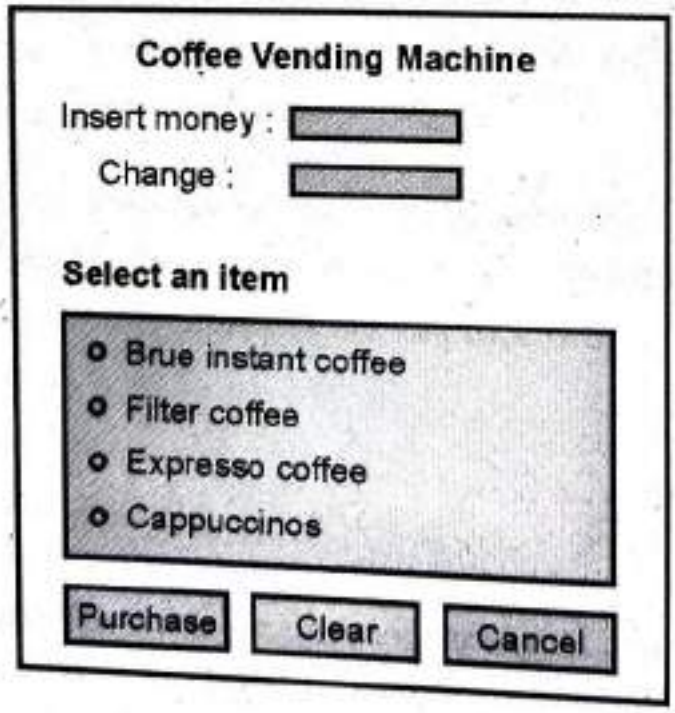
A coffee vending machine serves coffee to customers. A customer can choose a type of coffee among a list often options, supply the amount required and get served. Each coffee is prepared by adding units of hot water, coffee powder, milk and sugar. The recipe for each coffee is stored.

3.2 Technical Issues

The system should be implemented in Java

4. Interface requirements

4.1 GUI



4.2 Hardware Interface

Hardware Interface 1 : The system should be embedded in the laptops.

4.3 Software Interface

Software Interface 1 :

Interface Identification: Customer

Interface type: Person

Description : The customer is an actor who uses the Vending machine system to do the transactions

Software Interface 2 :

Interface Identification : Maintainer

Interface type : Person

Description : The maintainer is an actor who maintains and manage the vending machine system.

Software Interface 3 :

Interface Identification : Controller

Interface type : System

Description : The admin is an actor who controls the vending machine.

5. Performance requirements

Not Applicable,

6. Design constraints

The system should be designed within 6 months.

7. Other non-functional attributes

7.1 Security

Not applicable.

7.2 Reliability

If there exists a situation in which insufficient payment has been tendered, to meet the price of the chosen item. It is handled by refusing to dispense the required beverage until additional payment is inserted, sufficient to equal or exceed the cost of the selected beverage.

7.3 Availability

The coffee must be made available on correct payment.

7.4 Maintenability

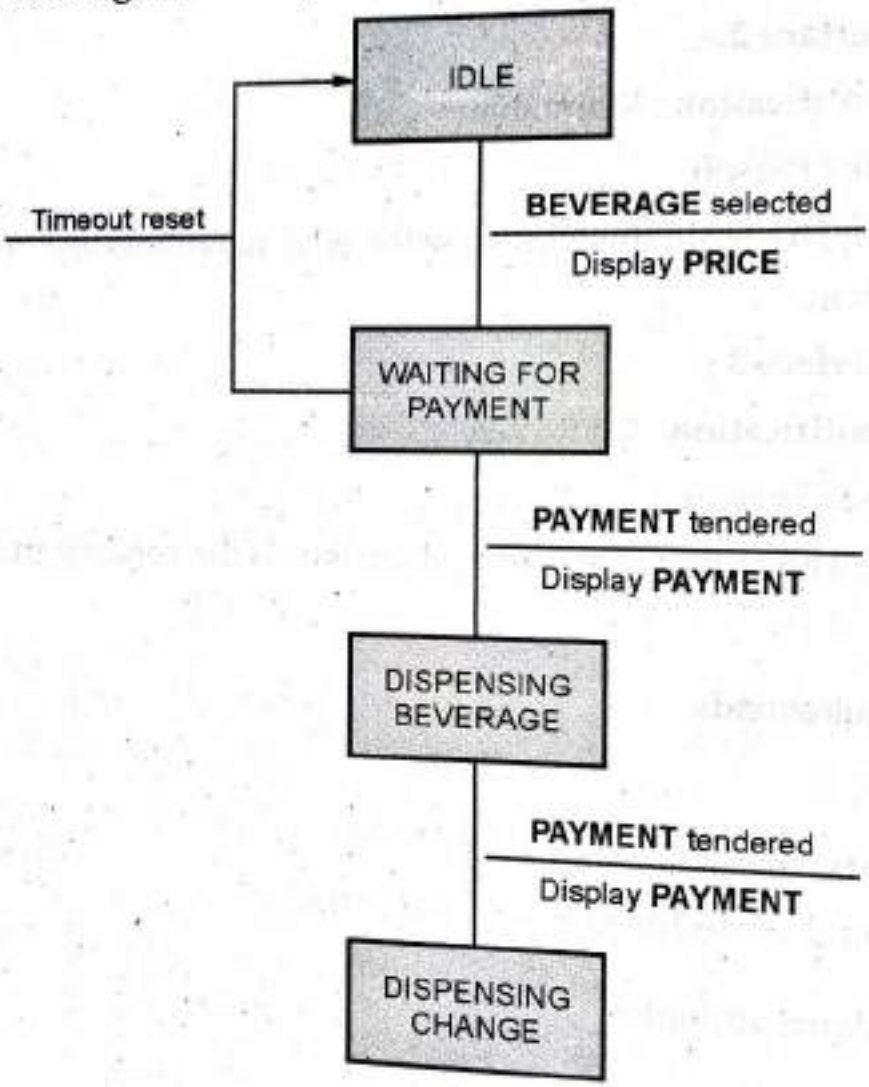
There should be a facility to clear the choice that is been made.

7.5 Reusability

The same system will be used for each type of coffee making.

8. Operational scenarios

Following is a state diagram that represents the operational scenario of the system.



Event List

The event list consists of three types of event : Direct flow of events, indirect flow o events, temporal(driven by clock or schedule)event.

Direct Event

(1) Customer makes the choice for the coffee.

(2) Customer tenders the payment.

(3) Coffee machine dispenses coffee.

(4) Coffee machine dispenses change.

Indirect Event

(1) Payment exceeds price

(2) Beverage not available.

(3) Hot water not available,

(4) Drinkable water not available.

Temporal Event

(1) Maintenance inspects and resupplies coffee machine

9. Preliminary schedule

The system has to be implemented within 6 months.

Example 2.6.3 *Prepare a software requirement specification document for "Library Management System"*

AU : Dec.-19, Marks 15

Solution :

1. Introduction

1.1 Purpose

This document gives the detail functional and non functional requirements for Library management system. Thee purpose of this document is that the requirements mentioned in it should be utilized by the software developer to implement the system.

This SRS will be used by the software engineers who are constructing the system and the users who will use the system. The hotel end users will be able to use this SRS as a test to see if the software engineers will be constructing the system to their expectations.

1.2 Scope

The library management system will automate the major library operations. Various subsystems will be student accounts management system, Book issuing system, Stock maintenance systems and fine calculation and management system.

1.3 Definitions, Acronyms and Abbreviations

- LMS - Library Management System
- SRS - Software Requirement Specification
- GUI - Graphical User Interface
- Stakeholder - The person who will participate in this system. For example Student, Staff, Librarian.

1.4 Overview

This system provides an easy solution for the students, staff and Librarian for accessing and managing the books in the library.

2. Overall description

The Library management system is developed for handling the activities for variou users such as Student, staff, librarian and library staff.

2.1 Product Perspective

This is a standalone system.

2.2 Hardware Interfaces

This system is placed in the Personal Computer in the library.

2.3 Software Interfaces

There are various databases in the system. These databases can be created in Oracle o MySQL. The hotel books and Student/Staff information is maintained by the LMS.

The book database include the Title of the book, ISBN number, publisher, edition number of copies, price,status(Old or new). The Student/staff database will include information such as first name, last name, address, phone number, designation(for staff) or class(for student), name of books issued, Library Card number, expects date of book return, fine to be paid(if any).

3. Functional requirements

The functional requirement will define the fundamental functioning that the system should perform.

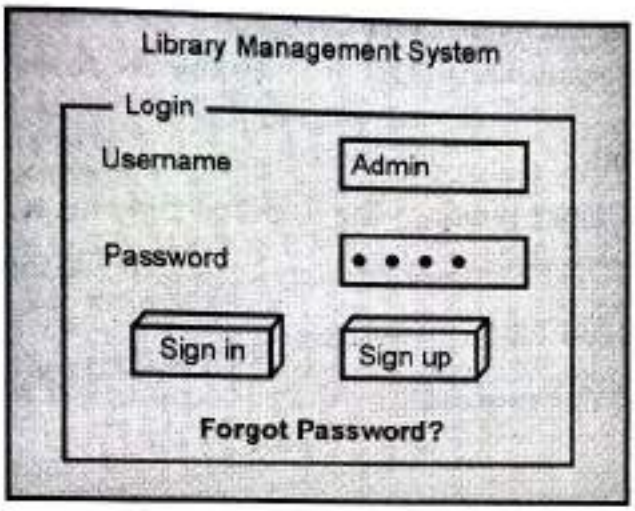
1. The system should record all the details of the student, staff and book.
2. The system should display the search results if the user searches for some book o article.
3. The system should allow to select the book for issuing.
4. The system should record the expected issue date and time of book.
5. The system should allow the modification in the student/staff/book information without reentering the entire information.
6. The system should allows to create an account for the new student.
7. The system should allow to cancel the account if the student leaves the college.
8. The system should display the availability of book or magazine.
9. The system shall allow to assign password to the users.
10. The system should keep track of the purchased items stock.
11. The member is allowed to reserve a book if it is currently unavailable.
12. The system should generate daily or monthly report for the stock.

13. The system should allow addition of book/magazine records if new books are purchased.
14. The system should allow deletion of book/magazine records if the books are old and in poor condition.

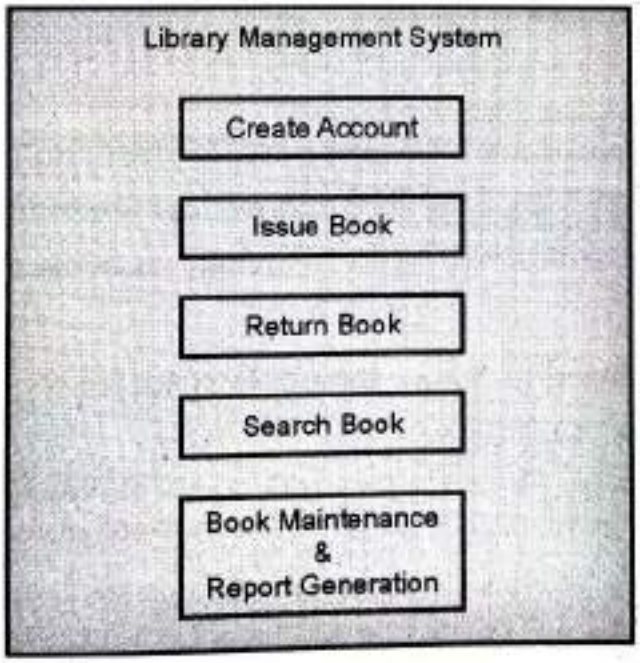
4. Interface requirements

4.1 GUI

GUI 1 : The login screen will be displayed so that the stakeholder of the system will enter the user name and password.



GUI 2 : The main menu will be displayed for Create Account, Search, Issue/Return Book, Book maintenance and Report generation.



GUI 3 : When the Create Account button is clicked then the form will be displayed This form will allow the user to enter the Student/Staff details such as last name, fir name, phone number, class and so on.

GUI 4 : If the user selects for the Search then the user will be allowed to search for th desired book.

4.2 Hardware Interfaces

The system should be embedded in the Computers in the library.

4.3 Software Interface

The system must be interfaced with Oracle or Access database.

5. Performance requirements

The performance requirement defines the response time for system functionality.

The load time for the GUI should not be more than four seconds.

The log in must be verified within 7 seconds.

The search query should be processed with in 3 seconds and response must be given.

6. Design constraints

The system must be designed as a standalone system and must be run on windo based system. The system can be developed in Java or Visual Basic. The database can b implemented in Oracle or MySQL.

7. Other non functional attributes

7.1 Security

The System administrator/Librarian must provide password to log on the system. Th password can be changed by the system administrator/Librarian. The Student should b allowed to the access the book database for searching of book. He/She is not allowed t modify the database.

The student/Staff is allowed to change their passwords for security purpose.

7.2 Reliability

The system must be reliable to prevent any unauthorized access.

7.3 Availability

The system should be available during the College hours.

7.4 Maintainability

There should be the facility to add/modify information about the Books, Student and Staff.

8. Operational scenarios

If a new students arrives in the library for membership then the librarian creates an account; issues him the library Card, User name and password.

If the student already has an account, he searches for the desired book/magazine. If the book is available then he makes the request to the librarian for issuing the book. The librarian issues the book and updates the database.

If the book is not available then the member reserves the book. On availability of the book, it is issued to the customer.

If the student/staff does not return the book on the specified date then he has to pay the fine.

The librarian updates the stock and the system generates the monthly report. The old and outdated books are removed from the library and accordingly the stock must be updated.

If new purchase is made then also librarian is allowed to update the book database.

9. Preliminary schedule

The system must be implemented within 6 months.

Review Questions

1. *Show the template of IEEE standard software requirements document.* **AU : May-13, Marks 8**
2. *What are the components of the standard structure for the software requirements document ? Explain in detail.* **AU May-14, Marks 8, May-18, Marks 16**
3. *Write the software requirement specification for a system of your choice.* **AU : May-14, Marks 8**
4. *Explain the organization of SRS and highlight the importance of each subsection.* **AU : May-16, Marks 8**
5. *Describe how Software requirements are documented. State the importance of documentation.* **AU : Dec.-20, Marks 13**
6. *What are the components of software requirement specification document? Explain.* **AU : May-22, Marks 7**

2.7 Formal System Specification

- A formal system specification for software requirements is an **unambiguous software system description.** In the formal system specification, **formal methods** are used to represent the requirements of the system.
- Formal methods provide us with tools to precisely describe a system and show that a system is correctly implemented. We say a system is correctly implemented when it satisfies its given specification.
- The specification of a system can be given either as a list of its **desirable properties** (property-oriented approach) or as an **abstract model** of the system (model-oriented approach).

2.7.1 Concept of Formal Technique

- A formal technique is a mathematical method used to specify a hardware or software system.
- The formal methods verify that the implementation satisfies the requirements specification, they prove the properties of the system without necessarily running the system.
- The formal specification language consists of the **syntactic domain, semantic domain** and a relationship called **satisfaction relation.**
- The formal techniques can be used at every stage of the software development life cycle (requirement specification, design, coding and implementation) to verify that one stage's output conforms to the previous stage's output.

Semantic domain

- Abstract Data Type(ADT) specification languages are used to specify algebras and programs.
- The programming languages are used to specify functions from input to output values.
- The distributed system specification languages are used to specify state sequences, event sequences, state transition sequences and finite state machines.

Syntactic domain

- The syntactic domain of formal specification language consists of alphabets, symbols and set of rules.
- These rules are used to construct **well-formed formula** using alphabets. Basically, these well formed formulas are used to specify the system.

Satisfaction relation

- For any model of a system, it is important to determine if elements of the semantic domain satisfy its specification. This satisfaction is determined by a function which is known as **semantic abstraction function**. The semantic abstraction function maps the elements of semantic domain to equivalent classes
- There are different specifications that are used to describe different aspects of the system. For example - a specification for describing system behavior or a specification describing the system structure.

2.7.2 Merits and Limitations of Formal Methods

Merits

1) Formal methods provide a **precise and unambiguous** way to describe the behavior and properties of a software system.
2) Formal methods can be used to **prove the correctness of a software system** concerning its specification. This is particularly valuable for safety-critical and mission-critical systems.
3) Formal specification can serve as a **clear and comprehensive documentation** of system requirements.
4) The formal methods can **identify errors in the specification** itself. This allows corrections before implementation.
5) The formal methods **allow rigorous analysis** of complex systems. Hence formal methods promote the construction of rigorous specification of the system.
6) The mathematical basis of formal methods facilitates **automating the analysis of specifications**. The possibility of automatic verification is one of the most important advantages of formal methods.

Limitations

1) Formal methods are **difficult to learn and use.**
2) It is **difficult to check the absolute correctness** of systems using theorem-proving techniques.
3) Formal techniques are **not able to handle complex problems.**
4) It can be challenging to create complete formal specifications that capture all aspects of a **real-world system.**

2.7.3 Model-Oriented Approach and Property-Oriented Approach

- The formal method is classified using two approaches - **Model oriented approach** and **Property-oriented approach.**

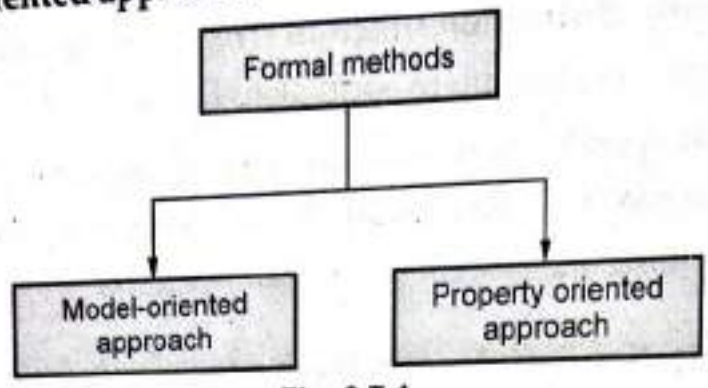


Fig. 2.7.1

- **In model oriented approach,** the system's behavior is represented by directly constructing a model of the system with the help of mathematical structures such as tuples, relations, functions, sets and sequences.
- **In property property-oriented approach,** system's behaviour is defined indirectly by stating its properties. These properties are specified in terms of a **set of axioms.**
- The property-oriented approach is more suitable for **requirements specification** and model-oriented approach is more suitable for system **design specification.**

Difference between Model-oriented approach and Property oriented approach

Sr. No.	Model-oriented approach	Property oriented approach
1.	The system's behavior is represented by an **abstract model** of the system.	The system's behavior is represented by stating the **properties** or requirements of the system.
2.	This approach is more suitable for the **design specification** of the system.	This approach is more suitable for **requirements specification** of the system.
3.	Mathematical **structures** such as tuples, relations, functions, sets and sequences are used in this approach.	The **set of axioms** is used in this approach.
4.	**A limited number of implementations** are possible using the model oriented approach.	**A large number of implementations** are possible using property property-oriented approach.
5.	**Making minor changes** in the specification is very **difficult** in model-oriented approach.	**Making changes** at any specification stage is easier in a property-oriented approach.

- There property oriented approach is classified into two categories - Axiomatic specification and algebraic specification

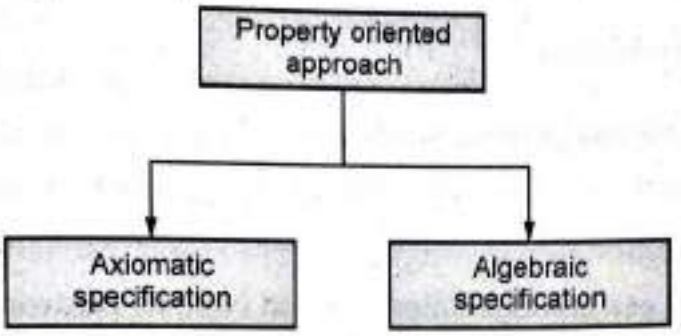


Fig. 2.7.2

2.7.4 Axiomatic Specification

- The axiomatic specification is based on formal logic or predicate calculus.
- The main purpose of this type of semantic method is - **formal program verification**
- **Axioms or inference rules** are defined for each statement type in the language to allow transformations of logic expressions into more formal logic expressions.
- An Axiom is a logical statement that is assumed to be true.
- An Inference Rule is a method of inferring the truth of one assertion based on the values of other assertions.

$$\frac{S_1, S_2, S_3, .. , S_n}{S}$$

- The **rule** states that if S_1, S_2, ... and S_n are true, then the truth of S can be inferred. The top part of an inference rule is called its **antecedent**; the bottom part is called it **consequent**.

Concept of assertion :

- The logic expressions are called **assertions**.
- An assertion before the statement or command is called a precondition. This condition states the relationships and constraints among variables that are true at that point in execution.
- An assertion following a statement is called postcondition.

<precondition> statement <postcondition>

- The axiomatic semantic is specified in the following manner

{P} statement {Q}

Where P is a precondition, Q is the postcondition.

- For example -

$$x = y+1 \qquad \{x>1\}$$

One possible precondition : $\{y > 10\}$

Axiomatic specification for assignment statement

The precondition and postcondition of an assignment statement together defi precisely its meaning.

Let x = E be a general assignment statement and N be its postcondition.

Then, its precondition, M, is defined by the axiom

$$M = N_x \rightarrow E$$

That means M is computed as N with all instances of x replaced by the expression E.

How to develop axiomatic specifications ?

Step 1 : Establish the range of input values over which the function should behave correctly.

Step 2 : Establish the constraints on the input parameters as a predicate.

Step 3 : Specify a predicate defining the condition which must hold on the output of the function.

Step 4 : Establish the changes made to the function's input parameters after execution of the function.

Step 5 : Combine all of the above into pre and post-conditions of the function.

2.7.5 Algebraic Specification

- In the algebraic specification technique, an object class or type is specified in terms of relationships that exist between the operations defined on that type.
- The algebraic specifications define a system as **heterogeneous algebra**. The heterogeneous algebra is a collection of different sets on which several operations are defined. Traditional algebras are **homogeneous**. A **homogeneous algebra** consists of a single set and several operations.
- The algebraic specification consists of four sections -
  1. **Type section :** In this section, the data types being used are specified.
  2. **Exception section :** In this section, exceptional conditions that may occur during the operations are defined.
  3. **Syntax section :** This section defines the signatures of the interface procedures. The collection of sets that form the input domain of an operator and the sort where the output is produced is called the signature of the operator.

4. **Equations section :** This section specifies the set of equations or rewrite rules. These rules define the meaning of the interface procedures.

The algebraic specification consists of the following **set of operators -**

1. **Basic construction operators :** These operators are used to create or modify entities of a type. For example - **create** and **append** are basic construction operators.
2. **Extra construction operators :** These are the construction operators other than the basic construction operators. For example - The **remove** is an extra construction operator.
3. **Basic inspection operators :** These operators evaluate attributes of type without modifying them. For example - eval, get are the basic inspection operators.
4. **Extra inspection operators :** These extra inspection operators are other than the basic inspection operators.

Example of algebraic specification

Following is an algebraic specification that represents the Cartesian coordinates. The operations defined on X and Y coordinates which evaluates the x and y attributes of an entity. The **IsEq** compares two entities for equality.

Types section

defines Coord

uses Integer, Boolean

Syntax section

$$\text{Create (Integer, Integer)} \rightarrow \text{Coord ;}$$

$$\text{X (Coord)} \rightarrow \text{Integer ;}$$

$$\text{Y (Coord)} \rightarrow \text{Integer ;}$$

$$\text{IsEq (Coord, Coord)} \rightarrow \text{Boolean ;}$$

Equations section

$$\text{X (Create (x, y))} = x$$

$$\text{Y (Create (x, y))} = y$$

$$\text{IsEq (Create (x1, y1), Create (x2, y2))} = ((x1 = x2) \text{ and } (y1 = y2))$$

Properties of algebraic specification

1) **Finite Termination Property :** This property of algebraic specification ensures that any sequence of operations applied to a data structure or system will terminate in a finite number of steps. In other words, it guarantees that there will be no infinite loops or non-terminating computations.
2) **Unique termination property :** The unique termination property of an algebraic specification states that any two sequences of operations involving the interface procedures of the specification will eventually terminate and produce the same result.
3) **Completeness :** The completeness property of an algebraic specification states that the specification is sufficient to define the behavior of the system for all possible inputs and outputs.

Pros and cons of algebraic specification

Pros

1) The algebraic specifications are based on mathematical structures. Hence they are unambiguous and specification.
2) Using algebraic specification, the effect of the arbitrary sequence of operations can be studied.

Cons

1) Algebraic specifications are hard to understand.
2) The algebraic specifications are difficult to integrate with programming languages.

2.8 Finite State Machines

- A finite state machine is a mathematical abstraction used to design algorithms.
- In simpler terms, a state machine will read a series of inputs. When it reads an input, it will switch to a different state. Each state specifies which state to switch to for a given input.
- A very good example of a finite state system is the control mechanism of the elevator. This mechanism only remembers the current floor number pressed, it does not remember all the previously pressed numbers.
- The finite state system is a very good design tool for programs such as text editors and lexical analyzer (which is used in compilers). The lexical analyzer is a program

that scans your program character by character and recognizes those words as tokens.

2.8.1 Formal Definition

A Finite State Machine (FSM) is a collection of 5-tuple $(Q, \Sigma, \delta\ q_0, F)$ where,

Q is a finite **set of states**, which is non empty.

Σ is **input** alphabet, indicates input set.

q_0 is an **initial state** and q_0 is in Q i.e. $q_0 \in Q$.

F is a set of **final states.**

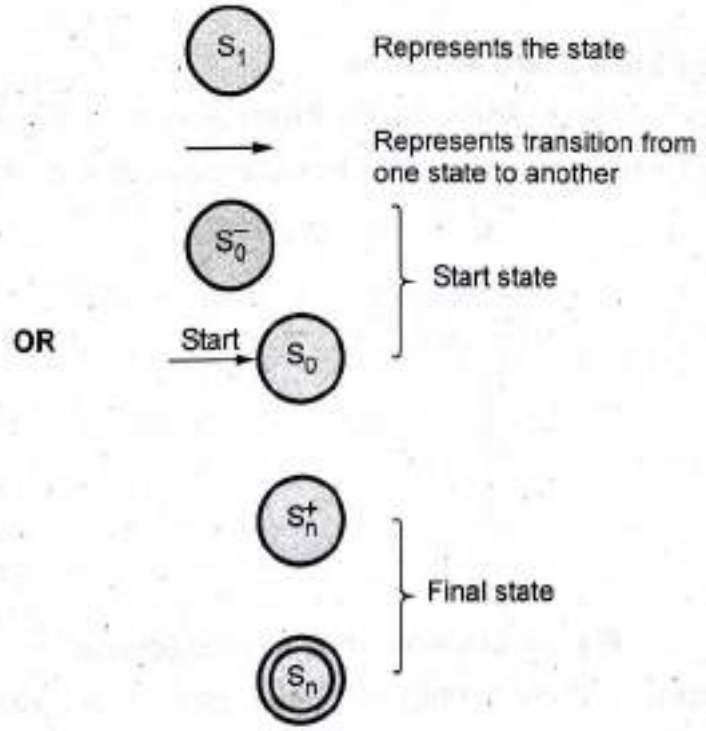
δ is a **transition function** or a mapping function. Using this function the next state can be determined.

2.8.2 Representation

A transition diagram or transition graph can be defined as collection of -

1) Finite set of states K.
2) Finite set of symbols Σ.
3) A non empty set S of K. It is called **start state.**
4) A set $F \subseteq k$ of **final states.**
5) A transition function $K \times A \rightarrow K$ with K as state and A as input from Σ^*.

The notations used in transition diagram are -



2.8.3 Types of Finite State Machines

There are two types of finite automata or finite state machine

i) Deterministic finite automata.

ii) Non deterministic finite automata.

The deterministic finite automata is deterministic in nature, in the sense, each move in this automata is uniquely determined on current input and current state. On the other hand, it is non deterministic in nature, in NFA i.e. non deterministic finite automata.

2.8.4 Applications of Finite State Machine

1. Finite automata are used in text processing in matching the pattern against the given text.
2. It is used in compilers for analyzing the syntax of the language.
3. The finite automata are also used in preparing the hardware designs.
4. The Context-Free Grammar (CFGs) are used in programming languages and artificial intelligence. Originally, CFGs were used in the study of the human languages. Cellular automata are used in the field of biology, for instance - representing the pine cones growth and pigmentation patterns.
5. Automata simulators are pedagogical tools used to teach, learn and research automata theory.
6. In switching theory, design and analysis of digital circuits automata theory is useful.

2.8.5 Deterministic Finite State Machine

The finite automata is called Deterministic Finite Automata(DFA) if there is only one path for a specific input from current state to next state. For example, the DFA can be shown as below.

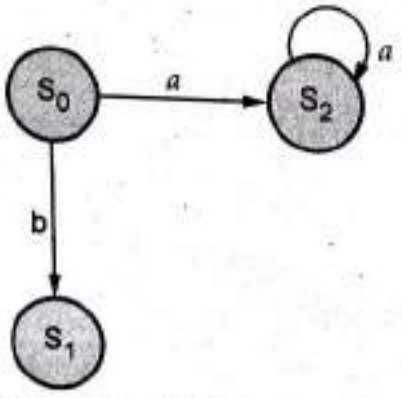


Fig. 2.8.1 Deterministic finite automata

From state S_0 for input 'a' there is only one path, going to S_2. Similarly from S_0 there is only one path for input b going to S_1.

Example 2.8.1 *Design a DFA which checks whether the given binary number is even.*

Solution : The binary number is made up of 0's and 1's. When any binary number ends with 0 it is always even and when a binary number ends with 1 it is always odd. For example,

0100 is a even number, it is equal to 4.

0011 is an odd number, it is equal to 3.

And so while designing FA we will assume one start state, one state ending in 0 and other state for ending with 1. Since we want to check whether given binary number is even or not, we will make the state for 0, the final state.

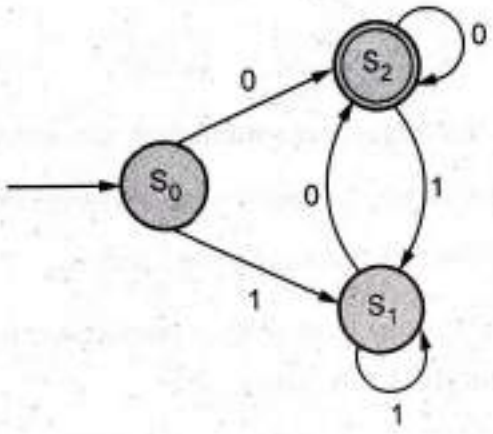


Fig. 2.8.2 For Ex. 2.8.1

The FA indicates clearly is a state which handles all the 1's and S_2 is a state which handles all the 0's. Let us take some input.

$$01000 \Rightarrow 0\underline{S_2}1000$$
$$01\underline{S_1}000$$
$$010\underline{S_2}00$$
$$0100\underline{S_2}0$$
$$01000S_2$$

Now at the end of input we are in final or in accept state so it is an even number.

Similarly let us take another input.

$$1011 \Rightarrow 1\underline{S_1}011$$
$$10\underline{S_2}11$$
$$101\underline{S_1}1$$
$$1011S_1$$

Now we are at the state which is a non final state.

2.8.6 Non Deterministic Finite State Automata(NFA)

The concept of non-deterministic finite automata is exactly reverse of determinis finite automata. The finite automata is called NFA when there exists many paths for specific input from current state to next state. The NFA can be shown as in Fig. 2.8.3.

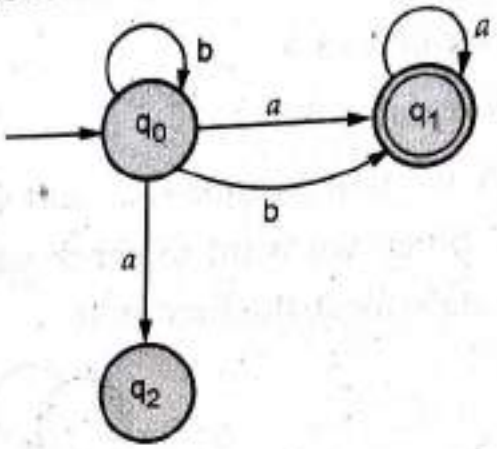


Fig. 2.8.3 Non-deterministic finite automata

Note that the NFA shows from q_0 for input a there are two next states q_1 and q Similarly, from q_0 for input b the next states are q_0 and q_1.

Thus it is not fixed or determined that with a particular input where to go next. Hen this FA is called non-deterministic finite automata.

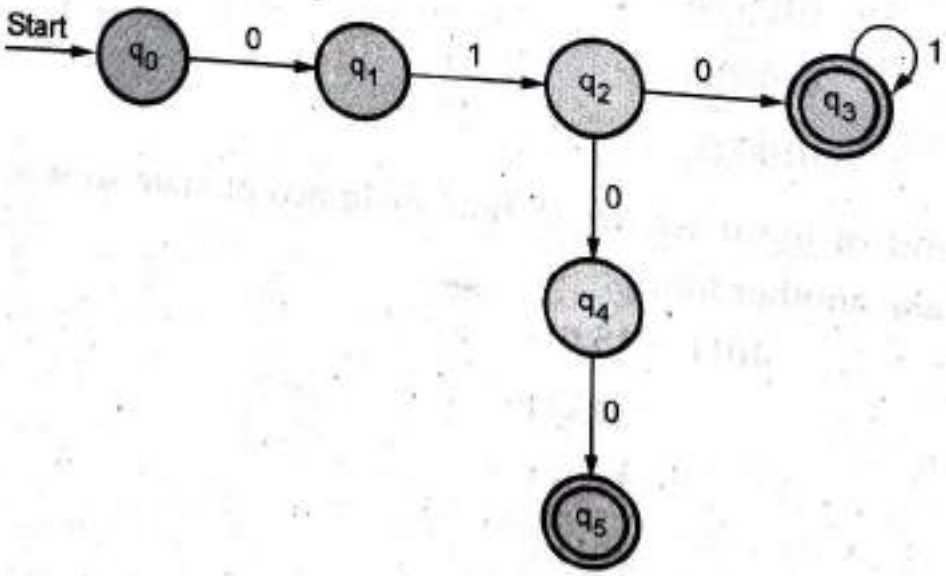
Example 2.8.2 *Construct NFA for the language*

$L = \{0101^n \cup 0100 \mid n \geq 0\}$

Over $\Sigma = \{0, 1\}$.

Solution :

Here in language L first three symbols are common i.e. 010. Hence the NFA can b drawn as



2.9 Petri Nets

AU : Dec.-19, Marks 13

- The Petri Nets are invented by Carl Adam Petri in 1962.
- The Petri Nets can be rigorously used to define the system. The Petri nets are more popularly used for distributed systems and systems with resource sharing.
- In Petri Nets there are three types of components - places (circles), transitions (rectangles) and arcs (arrows)

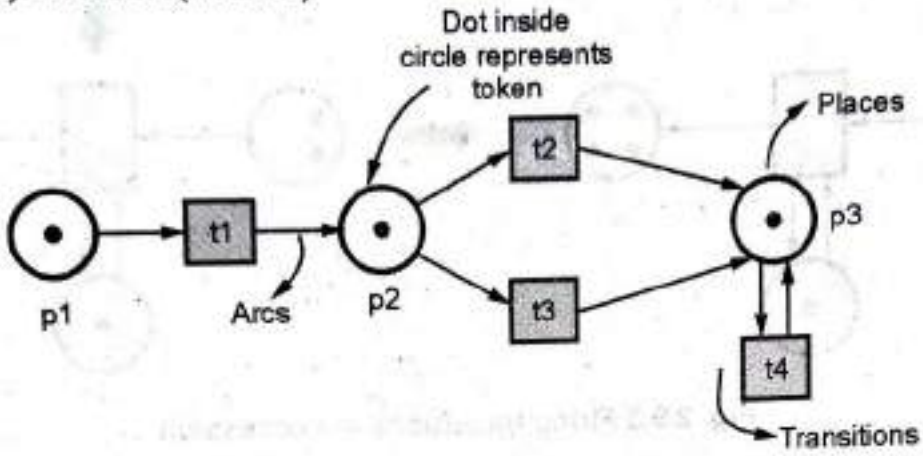


Fig. 2.9.1 Petri nets

- **Places :** Represent possible states of the system.
- **Transitions :** Cause the change in the states
- **Arc :** Connects a place with transition or transition with a place.

For example -

- Another equivalent notation used is a solid bar. This solid bar is used to denote transitions.
- The token represents the dynamic objects. The states having block dots are called marked Petri token. If there is a single dot inside the circle then that means it represent the one unit of certain resource.
- **Firing a transition :** Transitions take input from the input places and produce tokens in the output places. This is called firing a transition. For example

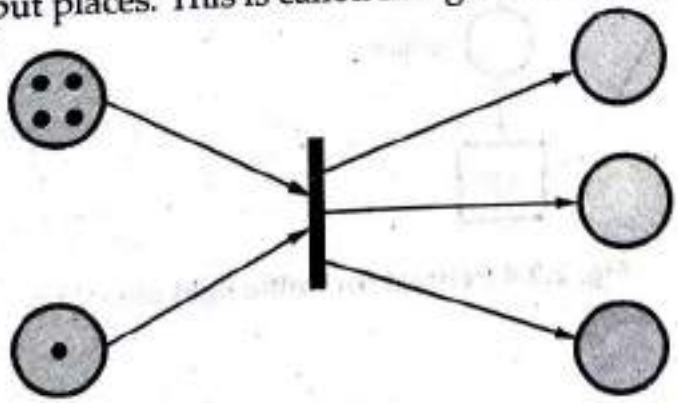


Fig. 2.9.2 Firing transitions

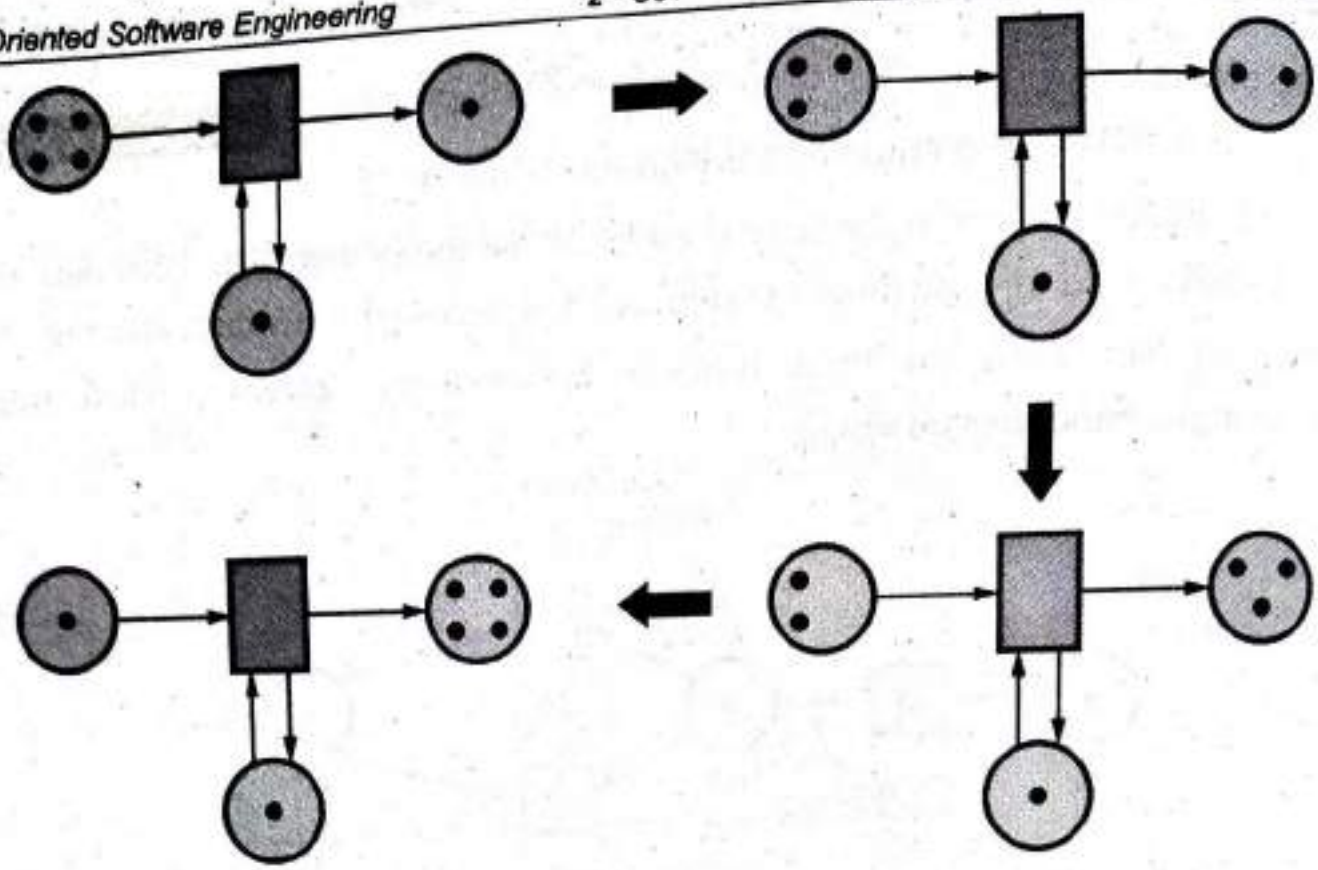


Fig. 2.9.3 Firing transitions in succession

Example 2.9.1 *Draw a petrinet for a traffic light switching system.*

Solution :

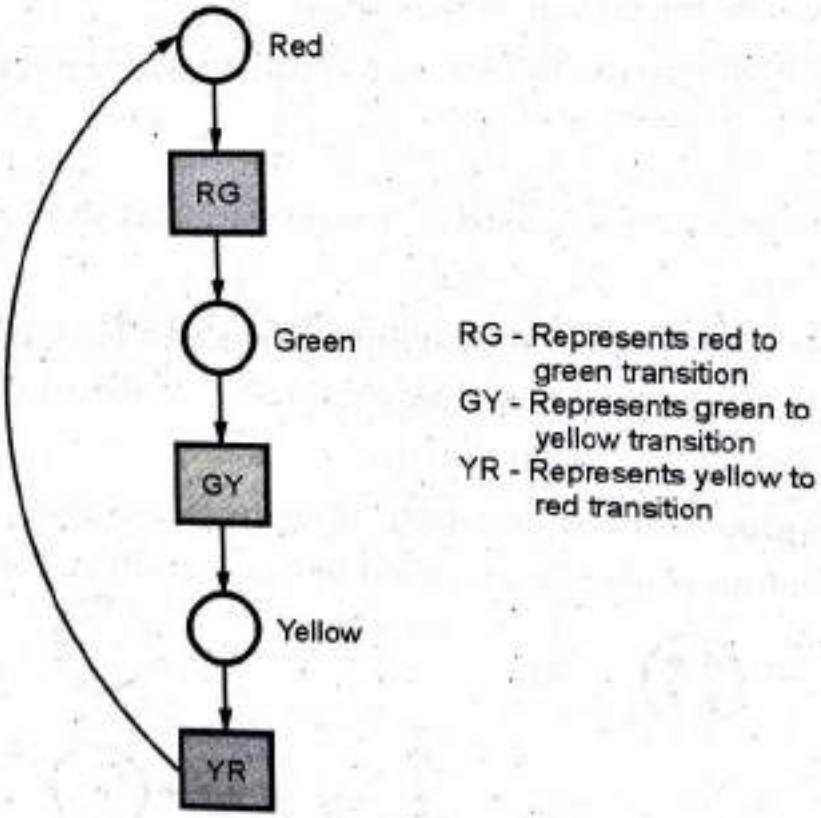
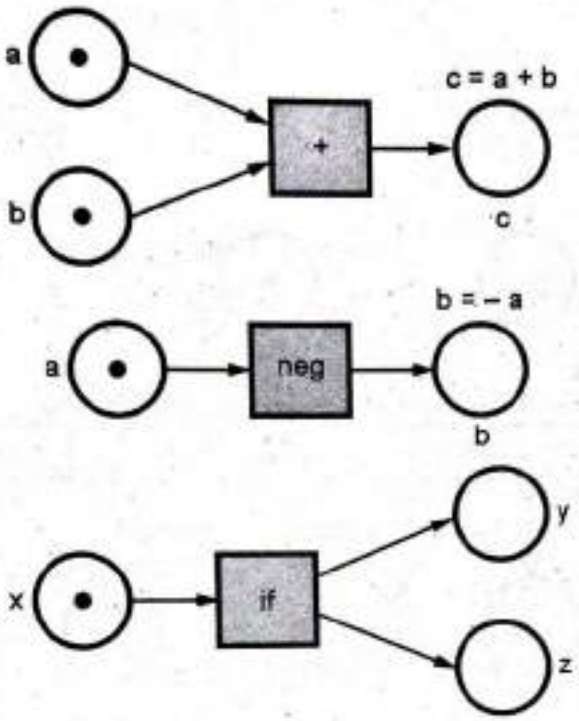


Fig. 2.9.4 Petrinet for traffic light signaling

Example 2.9.2 *Represent the operations i) $c = a + b$ ii) $b = -a$ iii) $x > 0$ then $y = x$ else $z = x$*

Solution :



Example 2.9.3 *Draw a Petri Net that depicts the operation of an "Automated Teller Machine". State the functional requirements you are considering.* **AU : Dec.-19, Marks 13**

Solution : Petri Net for demonstrating the operation of Automated Teller Machine.

(Refer Fig. 2.9.5 on next page)

Functional Requirements

1) The card should be valid
2) Pin entered by the user must be valid.
3) The amount to be withdrawn must be less than the required balance amount.
4) The balance amount must be updated and then displayed to the user after withdrawal of money.

Advantages of using Petri Nets

1. It specifies the precise definition of problem.
2. It represents the graphical workflow.
3. It is useful in allocating resources to the tasks.

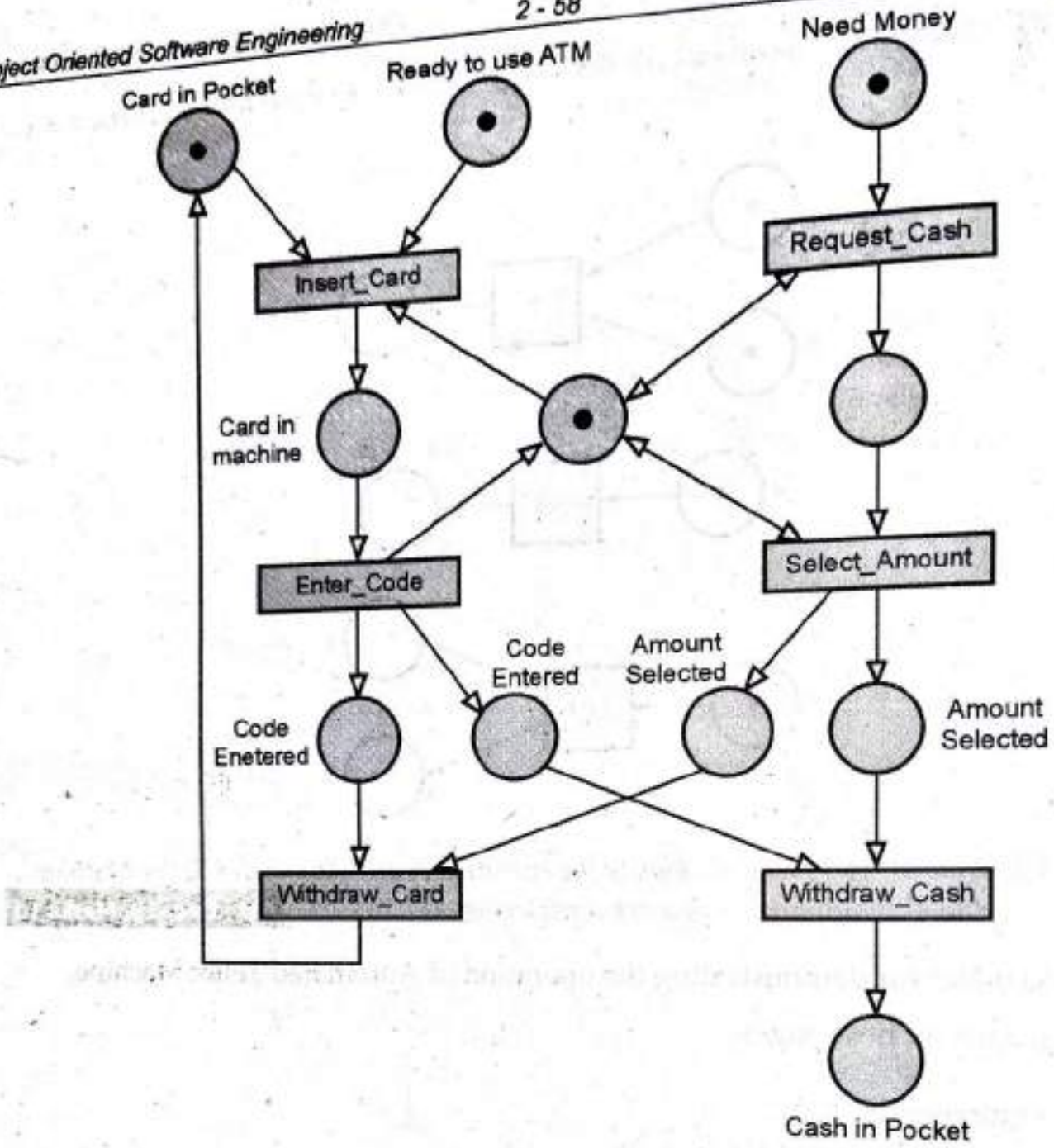


Fig. 2.9.5

2.10 Object Modeling using UML

AU : May-11, 14, Dec.-18, Marks 16

Object oriented analysis is the process of identifying and describing the objects for given problem instance. For example : For designing the banking systems **customer, Account, Branch** are the objects that must be described.

Object oriented design is the process of defining the objects and their collaboration with other objects in order to fulfill the requirements. For example in **banking system getCustomerInfo, Withdraw_Amount, Deposit_Amount** and so on are the methods that help in collaborating one object with another object. These methods actually help in fulfilling requirements. The methods of objects represent the behavior of the objects. This can be illustrated by the Fig. 2.10.1.

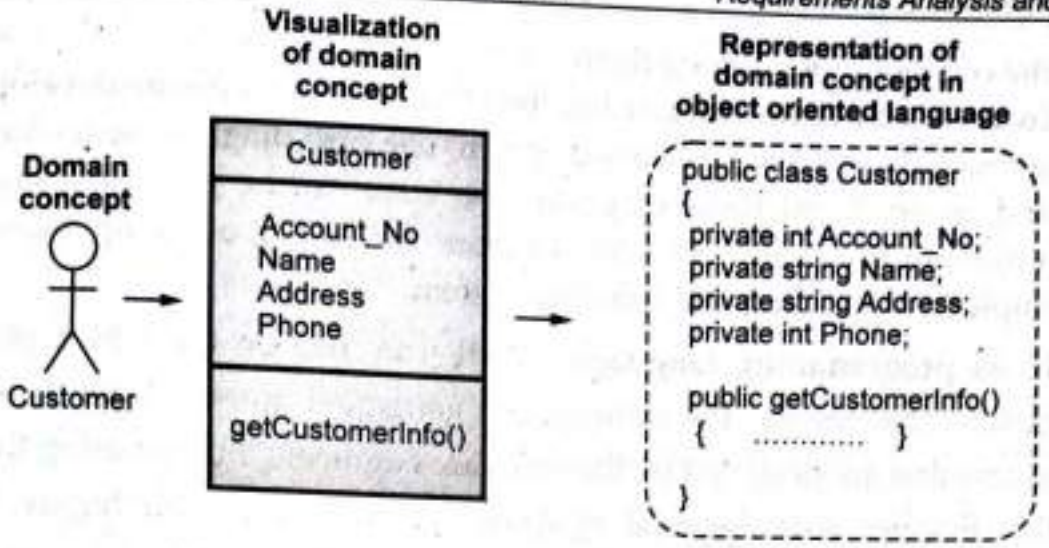


Fig. 2.10.1 Representation of object orientation

Key Point : *Object oriented analysis and design is the study of objects.*

2.10.1 What is UML?

Definition : The Unified Modeling Language (UML) is a standard diagramming notation for specifying, visualizing, constructing and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.

- **The goals of UML are -**

1. UML is intended to provide ready- to use and expressive visual modeling language in order to develop the effective system model.
2. The modeling must be independent of the programming language or platform but at the same time the automatic code that can be generated by this tool must support the modern programming languages.
3. This tool supports the high level concepts such as collaborations, framework, patterns and components.
4. It helps to integrate the organizational standards in the software systems.

- **According to Martin Fowler there are three ways of using UML -**

1. **UML as sketch :** For exploring the difficult part of the system or for representing the solution to particular problem, software designer create informal diagrams.
2. **UML as blueprint :** The UML can be used as a blueprint for two main reasons - either for reverse engineering or for forward engineering.
   - **In Reverse Engineering** the UML tool reads the source code and generates the class diagram, sequence diagram and so on. With the help of these diagrams the user can understand the structure of the system elements and

the collaborations among them.

- **In Forward Engineering,** using the UML tool the software developers draw some diagrams such as class diagram, use case diagram, sequence diagram and so on. From these diagrams the code can be generated manually or automatically. Later on, the software developer adds up more code to implement the complete software system.

3. **UML as programming language :** UML can also be used as a programming language. That means, the code can be generated automatically from the UML diagrams that are designed by the software engineers. But for using the UML to in this manner it is essential to draw the diagrams with highly logical and practical approach. Although one can use UML as a tool for programming language it is still immature to use it in this manner.

- **UML can be applied as three perspectives -**

1. **Conceptual perspective :** Using this perspective, the things in a real world situation are described by the UML diagram.

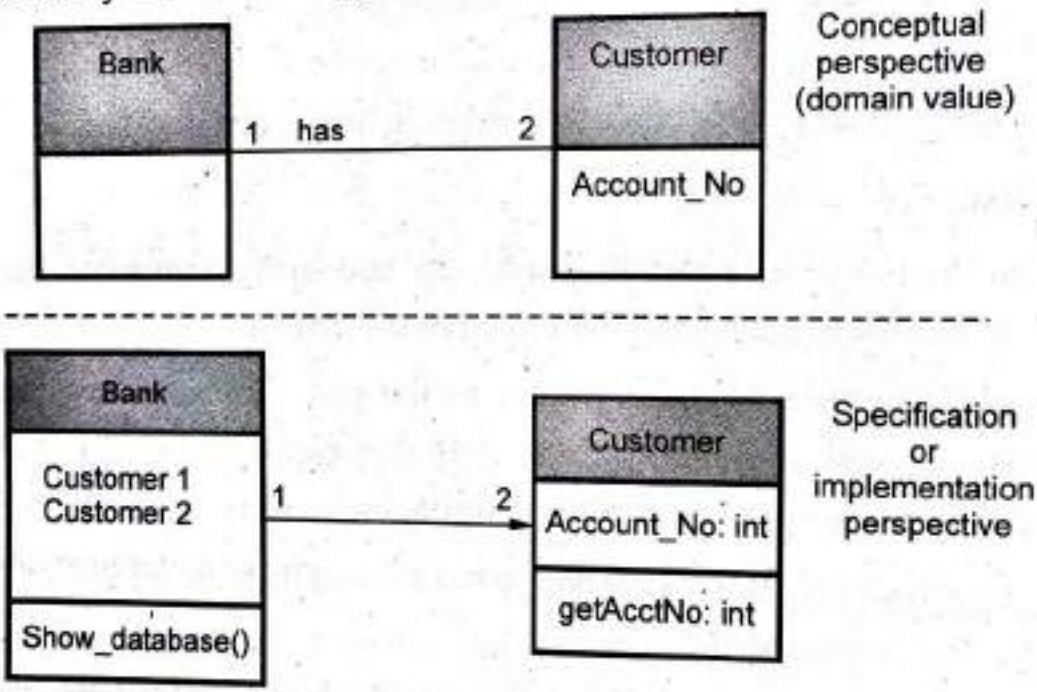


Fig. 2.10.2 Three perspectives of UML

2. **Specification perspective :** Using this perspective, the UML diagrams describe the software abstractions or components with specification or interface.
3. **Implementation perspective :** Using this perspective, the UML diagrams describe the particulars of implementation.

The most commonly used perspective is specification perspective. However, some of the software systems make use of implementation perspective.

• Meaning of class in different perspective

The classes used in the above described perspectives have three different names. Hence we define -

1. **Conceptual class :** This type of class represents the real world entities.
2. **Software class :** This type of class represents the specification or implementation perspective of a software component.
3. **Implementation class :** This type of class is implemented using some suitable object oriented programming language. The examples of object oriented programming languages are C++, Java.

2.10.2 UML Diagrams

- Diagrams are the **graphical representation** of the set of the elements.
- The diagrams can be drawn to represent the system from **different perspectives.**
- The same elements may appear in different diagrams.
- The diagram contains the collection of elements and relationship among these elements.
- Various **types of diagrams** that can be drawn in UML are -

1. Class diagram	2. Object diagram	3. Component diagram
4. Use case diagram	5. Sequence diagram	6. State diagram
7. Activity diagram	8. Deployment diagram	9. Package diagram

- **Class diagram :** It contains the set of **classes, interfaces, collaborations** and their **relationships**. The class diagrazm represents the **static design view** of the system. These diagrams are commonly drawn in UML.
- **Object diagram :** These diagrams show the **relationship among the different objects**. Objects are the instances of the classes, hence the object diagrams represent real or **prototypical cases**. These diagrams represent the **static design view** of the system.
- **Component diagram :** This diagram contains the collection of **components** and **connectors**. It represents the **static design view** of the system. It shows the encapsulated classes, interfaces, ports and internal structure of the system.
- **Use case diagram :** These diagrams are drawn to represent the **static design view** of the system. The use cases represent the **behaviour** of the system. It consists of collection of **use cases, actors**(special type of classes) and their **relationships.**

- **Sequence diagram** : These are kind of interaction diagrams consisting of set of **objects, roles** along the **messages**. They are specially useful for modeling the behaviour of the system.
- **State diagram :** The state diagrams represent the **states, transitions, activities** and **events**. The event-ordered behaviour of the object is emphasized.
- **Activity diagram :** This diagram represents the **flow of control** or data step by step occurring in computations. These diagrams emphasize on representation of functionality. These diagrams represent the **dynamic view** of the system.
- **Deployment diagram :** It addresses the static **deployment view** of the architecture. The run time processing of the components is represented by these diagrams.
- **Package diagram** : These diagrams show the **organizational units** and their dependencies using their semantics.

Review Questions

1. *What is object oriented analysis and design ?* **AU : May-11, Marks 2**
2. *List various UML diagrams and explain the purpose of each diagram.* **AU : May-14, Marks 16**
3. *How do you see the application of UML diagrams for Iterative Software Development ? Explain.* **AU : Dec.-18, Marks 13**

2.11 Use Case Model

AU : May-11,12,13,14,18,19, Dec.-11,12,13,15,16,17,18, Marks 16

- During requirement gathering the primary requirements of the system are understood. These basic requirements of the system are called the **functional requirements**.
- In order to understand the working of these functional requirements, the developers and users create a set of scenarios. These scenarios are nothing but the text stories. These text stories are called as **use cases**
- The use cases are the textual representation of particular scenario, whereas the use **case model** is the graphical representation of the scenario. Following table shows the template of the use case -

Use Case template	
Use Case	Write the name of the scenario.
Primary Actor	Specify the role of the entity who interacts with the system.
Goal in Context	Specify the purpose of this use case scenario.
Preconditions	The condition that is to be satisfied before the use case start.

Scenario	Sequence of steps that describe the main scenario.
Triggers	The triggering event due to which the function starts, is described.
Extensions points	The extension points describe the extend relationship.
Exceptions	Describe the exceptional situations that may occur during the execution of the use case.
Priority	Describe the necessity of the system to be implemented.
Open Issues	Certain additional issues that are required for execution of the scenario and that are not specified in the main scenario.

- For example : Following is an example of informal description or text story for withdrawal of money from ATM

" Customer wants to operate the ATM machine for performing some transactions. For instance customer wants to withdraw the money. He enters the ATM card and then types in the PIN. The system validates the customer. If the customer is the valid customer then the customer is allowed to do further transactions. Otherwise the card will be rejected. The valid customer enters the amount to be withdrawn. It is then checked if withdrawal amount < balance amount. If it is so, then the machine dispenses the desired amount. If there is no further transactions then the card is ejected. Customer collects the cash, statement and card."

Use Case template	
Use Case	ATM system
Primary Actor	Customer
Goal in Context	To monitor all the functionalities required to establish the session
Preconditions	ATM system has to be programmed and as to recognise and validate the PIN.
Trigger	On completion of every activity a beep has to be generated by the system.
Scenario	1. Customer observes the control panel. 2. Customer enters the ATM card 3. Customer enters the PIN. 4. Customer selects the operation(withdrawal, deposit, inquiry) 5. Customer collects the cash, statement, card etc.
Exceptions	1. The control panel is not ready. 2. PIN is incorrect. 3. Card is not recognised. 4. Insufficient balance 5. Limit for the transaction exceeds. 6. Total number of allowed transactions per day.

Priority	Essential and must be implemented in banking system.
Secondary Actor	Administrator
Open Issues	1. Is there a need to display any additional information by the control panel ? 2. For how many time the PIN is allowed to enter on incorrect supplement. 3. How much time is allotted to the customer to pick the items like cash or card after completion of operation ?

Key Point : *Use cases are not diagrams they are text.*

2.11.1 Actors, Scenarios and Use Cases

- Use cases are the fundamental units of modeling language in which functionalities are distinctly represented.
- The first step in writing use case is to identify the **actors**. The actors are the entities that use the system or product within the context of **behavior** of the system. Actors **represent the role of** the people as system gets operated. Actors can be a person, computer system or organization.

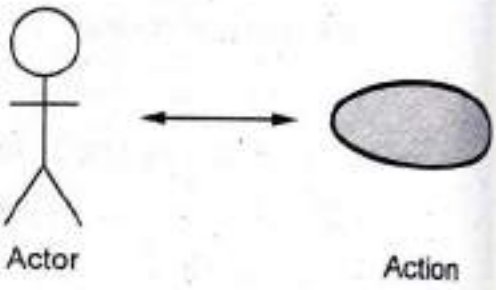


Fig. 2.11.1 Actor and Action representation

- Actor is anything that communicates with the system or product. It is external to the system. Every actor has one or more goals when using the system.
- **Example** : Customer, Inventory database, Accountant
- **Scenario** is a specific sequence of actions or interactions between actor and the system. It is also called as **use case instance**. The scenarios describe the text stories. For example : Scenario of issuing of a book from library, withdrawal of money from ATM.
- **Use case** a success or failure scenarios in which the actor communicates with the system in order to achieve certain functionality.
- For example : Online purchase system

Main scenario : User browses for the product on-line. Then user selects the desired product and adds it in shopping basket. System then displays the shopping basket with newly added product. Now user proceeds for the checkout. If the user is a new customer then it has to register himself as a new register, then this user might be sent with a password via E-mail. User then enters the password and submits the payment and shipping information. The payment is further validated by the system. Finally the system displays the order placed by the user and get it confirmed. The system then confirms the cell.

Alternate scenario : If the credit card information is incorrect, then asks the customer to try again or asks for giving the cash incase.

2.11.2 Use Cases and Use Case Model

- Use case modeling is done during the requirement analysis phase.
- Use cases are the text documents and the use case modeling is an act of writing the text and not the diagrams.
- In requirement analysis phase, along with use case modeling other documents such as glossary, business rules and standards can also be created.
- Use case modeling may include UML diagrams optionally. The use case diagram shows the actors and related functionalities.

Abstract and Concrete use Case

Abstract use case is use case which does not have complete declaration. The declaration in this use case is incomplete. An abstract use case is intended to be used by other use cases, e.g., as a target of generalization relationship.

For example : Refer Fig. 2.11.2 (a).

Use Cases that are self-contained and complete are said to be *concrete* use cases, in that they are actually instantiated by a particular actor(s). (Refer Fig. 2.11.2 (b))

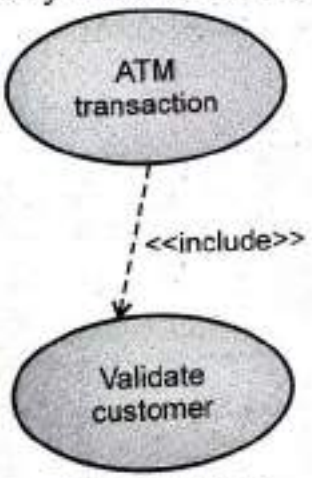


Fig. 2.11.2 (a)

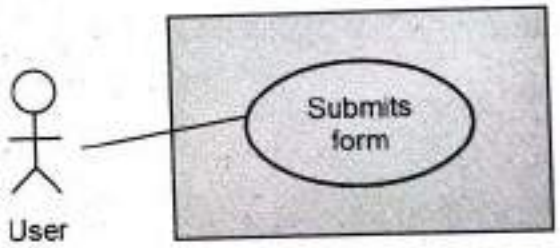


Fig. 2.11.2 (b)

Example 2.11.1 *By considering the library management system, perform the object oriented system development and give the use case model for the same (use include, extend and generalization).* **AU : Dec.-11, 12. Marks 16**

Solution :

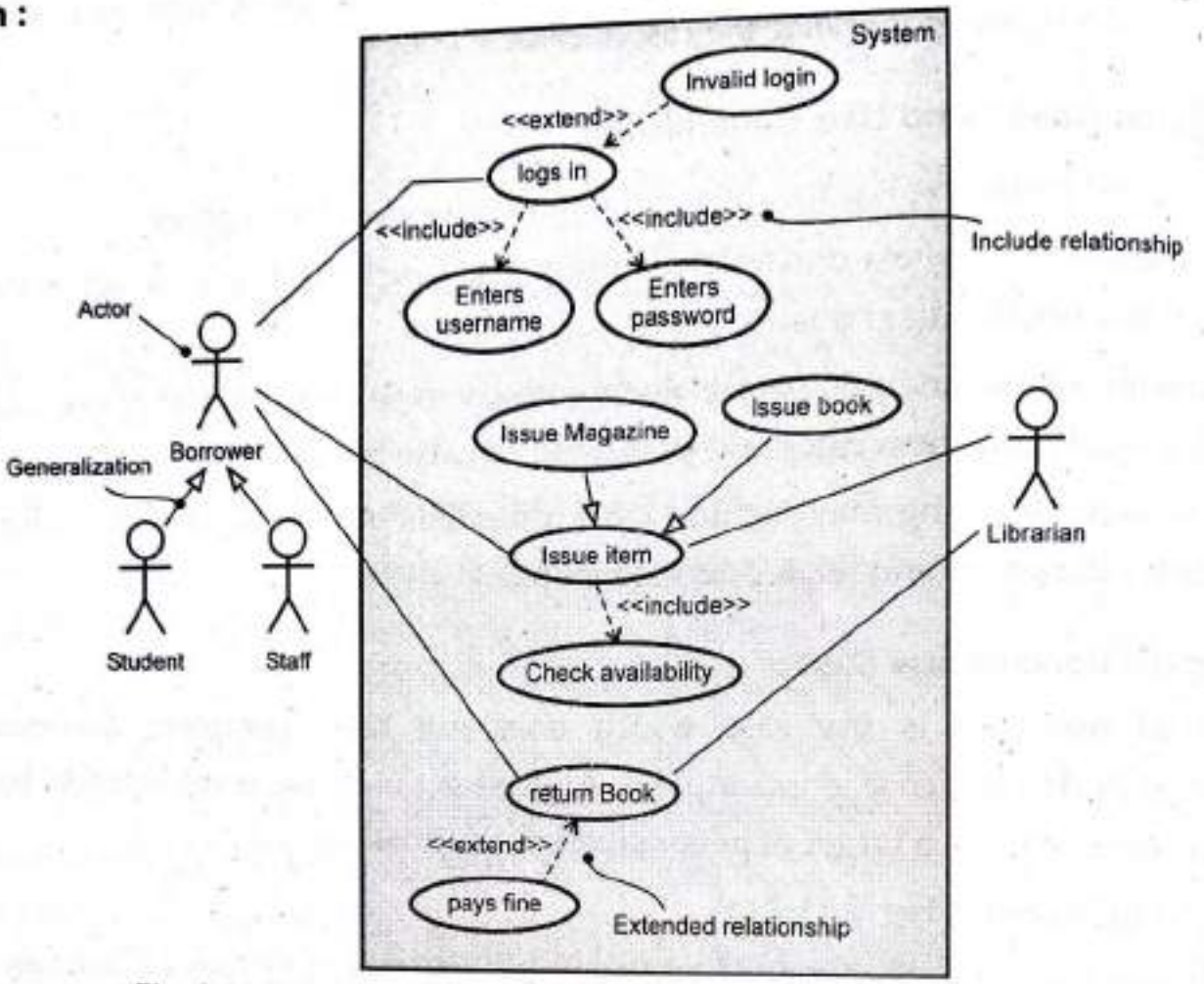


Fig. 2.11.3 Use case diagram for library management system

Example 2.11.2 *Design use case diagram for withdrawal of money from ATM system.*

Solution :

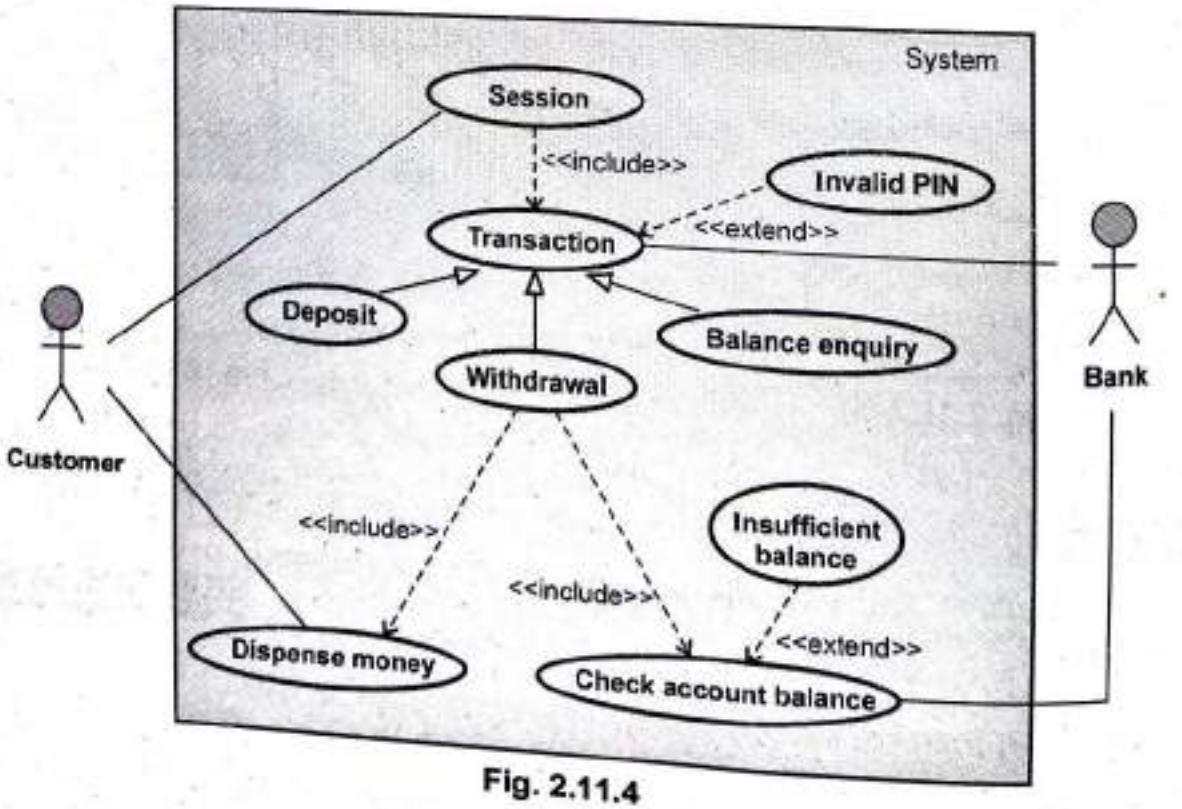


Fig. 2.11.4

Example 2.11.3 *Model a use case diagram for the following scenario : Deepthi Super market wants a subsystem to process the supply orders via the web. The user will supply via a form their name, password, account number and a list of supplies along with an indication of the quantities desired. The system will validate the input, enter the order into a database and generate a receipt with order number, expected ship date and total cost of the order. If the validation step fails, subsystem will generate an error message describing the cause of failure.*

AU : Dec.-17, Marks 8

Solution :

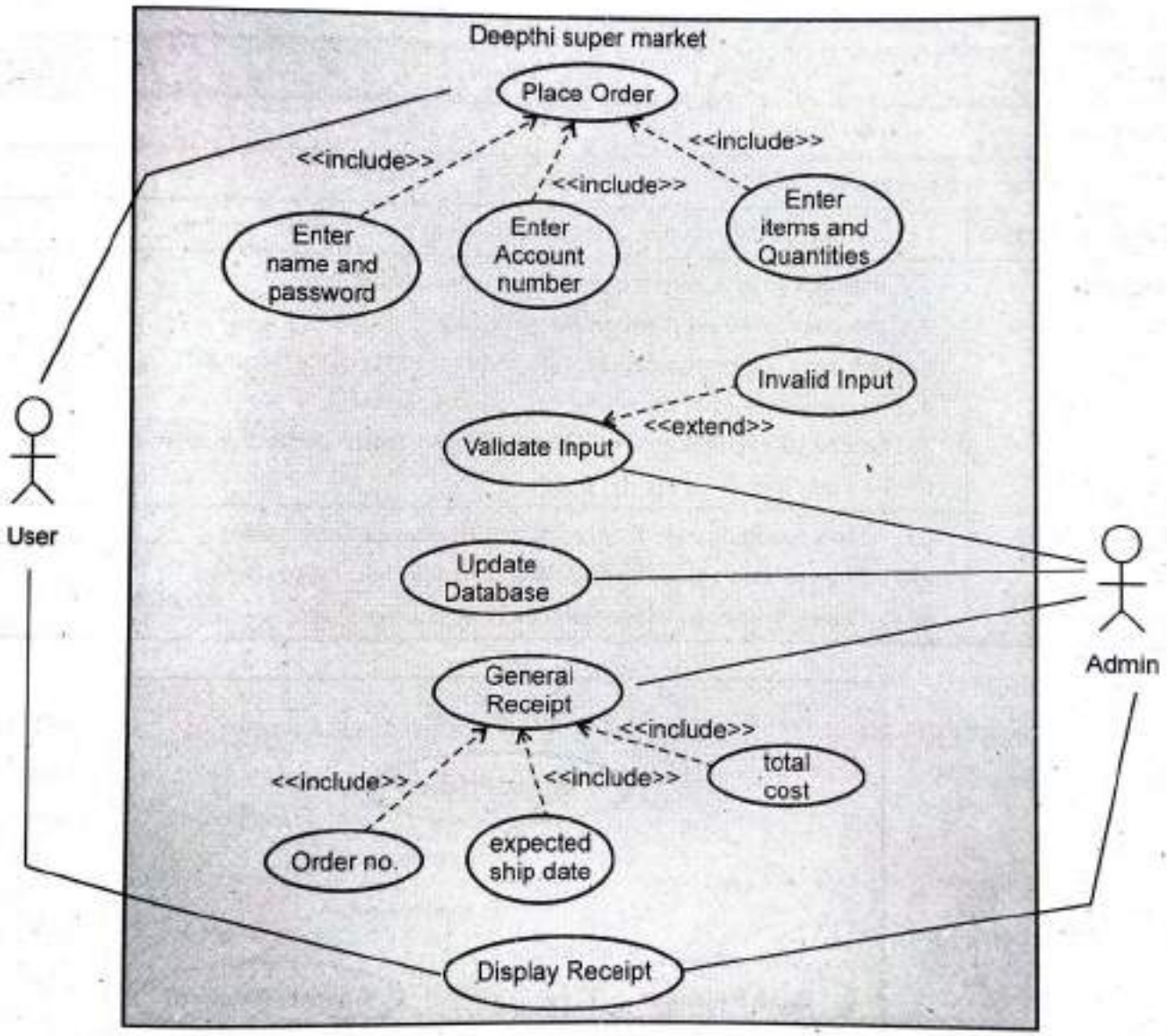


Fig. 2.11.5

2.11.3 Relationships in Use Cases

Several use cases can be related with each other. Similarly the use cases are also related to the actors who are intended to achieve the goal of the system. The use cases are related to any other entity by using include, extend and generalization relationship.

2.11.3.1 Include

- This is the most commonly used relationship which denotes that a given use case may include another. It is denoted by

<<Include>>

The use case at the arrow head position is the use case which is included by use case on the other side of the arrow. When there are multiple steps to carry out the single task then this task is divided into the sub-functions and these sub-functions are denoted by the use cases. For example -

Use Case template	
Use Case	On-line shopping
Primary Actor	Customer
Goal in Context	To monitor all the functionalities required to establish the session
Scenario	1. Customer logs in to purchase the product online 2. Customer browses through the products. 3. Customer selects the product and adds it to the shopping cart. 4. The contents of the shopping cart are displayed. 5. The product and shipping information is confirmed by the customer. 6. The customer pays for the product
Extensions	6a The customer pays by credit card (include relationship) 6b The customer pays by the cheque (include relationship) 6c The customer pays by cash (include relationship)

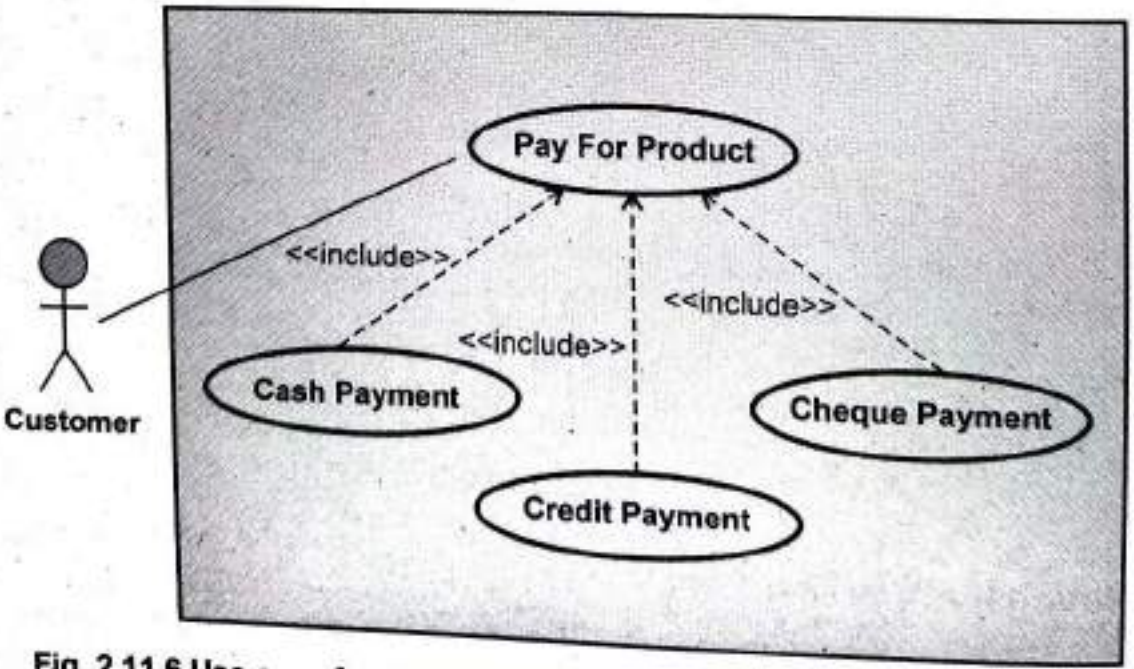


Fig. 2.11.6 Use case for payment for the product (Include-relationship)

2.11.3.2 Extend

- The extend relationship is used when an extended use case is connected to the base use case. While using the extend use the extension points can be explicitly shown in the base use cases. Following are the reasons why extensions occur -

1. When some part of the use case is optional and we need to show separate behavior of the system.

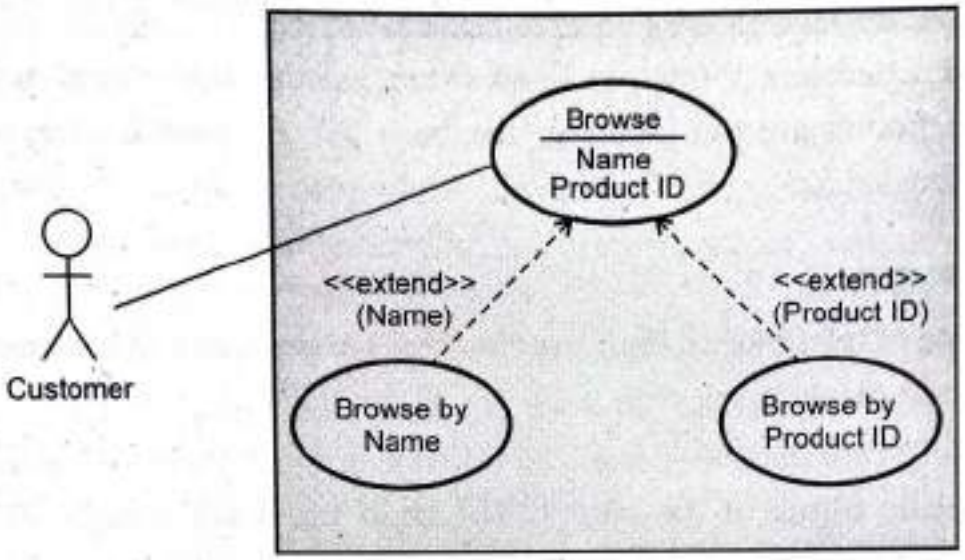
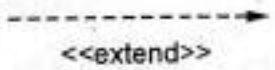


Fig. 2.11.7 Use case for browsing product (Extend - relationship)

2. When we want to show a subflow of the system when some specific conditions occur.

- This relationship is denoted by -



- The extension is conditional. That means its execution is dependent on what has happened while executing the base use case. These conditions are described by the extend relationship.
- The extension use case may access and modify attributes of the base use case
- Example : **Online purchase system,** when user browses for the product, it may browse either by the product name or by product ID.

Use Case template	
Use Case	On-line shopping
Primary Actor	Customer
Goal in Context	To monitor all the functionalities required to establish the session
Scenario	1. Customer logs in to purchase the product online

	2. Customer browses through the products. 3. ... 4. ... 5. The customer pays for the product
Triggers : Extensions points	Customer browses the catalog. 2a Customer browses the product by entering the product name 2b Customer browses the product by entering the product ID.

- Note that extension points are used when we use the extend relationship. The extension points are the labels in the base use case which are referred by the extending use case.

2.11.3.3 Generalization

- In this type of relationship there are two types of use cases - the parent use case and the child use cases.
- A parent use case may be specialized into one or more child use cases that represent more specific forms of the parent. The child use cases inherit all the structure behavior and relationships of parents.
- Generalization is used when we find two or more use cases having common structure and behavior and purpose.
- The test word used for generalization relationship is "**kind of**". It is denoted by

- For example : For the **Purchase system** the customer can place an order via telephone or using internet. The use case

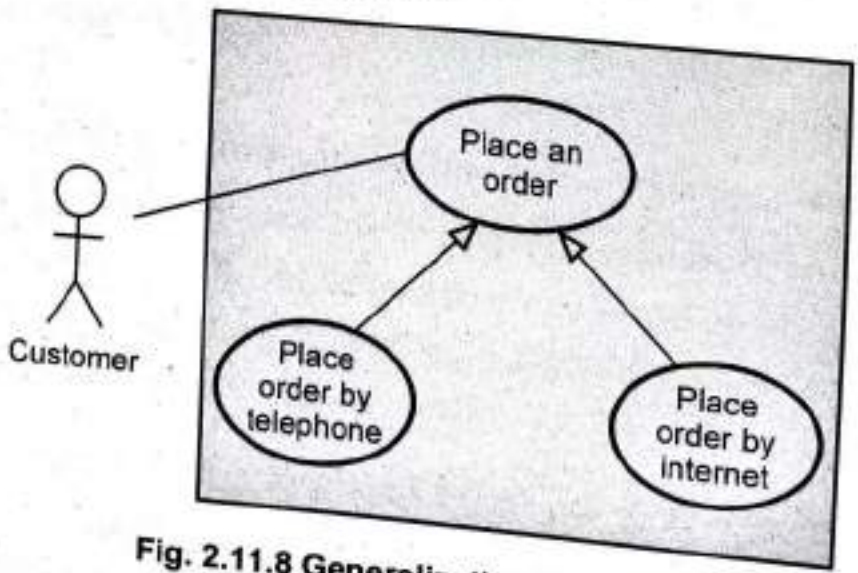


Fig. 2.11.8 Generalization Use case

2.11.4 When to Use Use Cases ?

- Use cases are used to model the behavior of the system. Following are some reasons why use cases are important -
  1. By modeling the **behavior of elements** using use cases will provide the outside view using which the inside view of the system can be constructed.
  2. Using use cases, developers can identify and **understand various system elements**.
  3. Use cases serve as the **basis for the test cases** that can be carried out during system development.
- Following are the steps that are used to model the behavior of the system -
  - Identify the actors that interact with the system. Actors interact with the system which is intended to perform some task.
  - Organize the actors by identifying the generalized and specific roles.
  - There are various ways by which the actor interacts with the system. There are some interactions in which the interactions change the state of the element. Some interactions occur when elements response to the event.
  - There are some exceptional ways in which each actor interact with the element.
  - For factoring the common behavior, make use of include relationship and for exceptional behavior make use of exclude relationship.
- Following is an example of the hospital management system in which the receptionist interacts with the hospital management for making admission to the hospital.

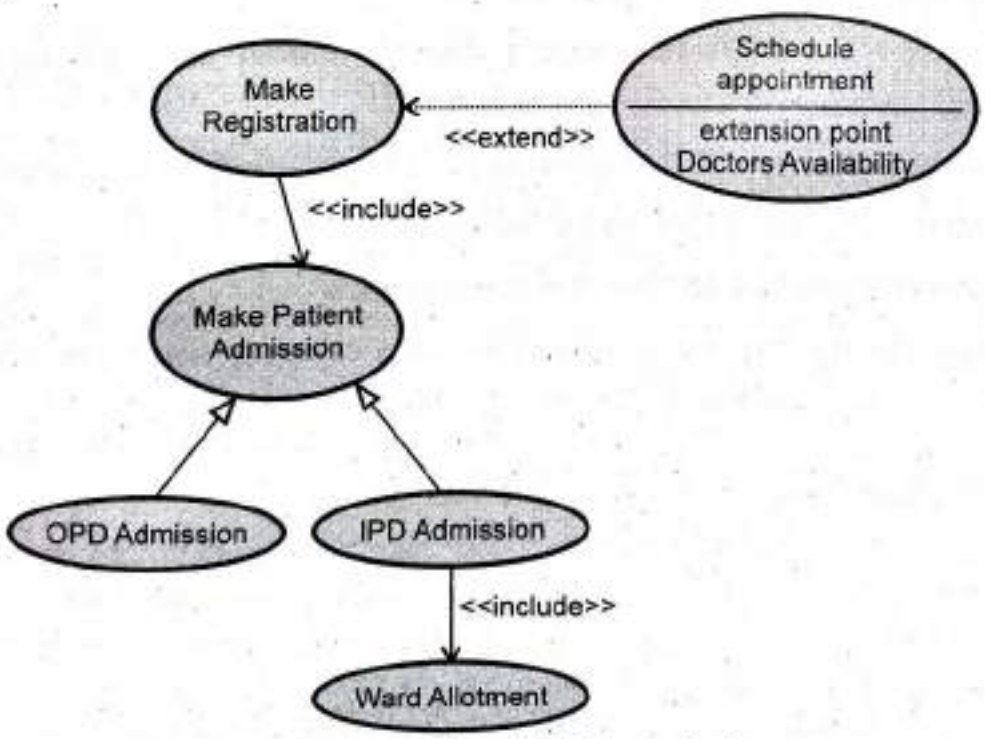


Fig. 2.11.9 Use case for admission to the hospital

Review Questions

1. *Explain with an example, how use case modeling is used to describe functional requirements. Identify the actors, scenarios and use cases for example.* **AU : May-11,12,13,14, Dec.-13,16, Marks 16**
2. *Explain about Use-Case Model for a case study of your choice.* **AU : Dec.-15, Marks 16**
3. *Explain with an example a concrete use case and an abstract use case.* **AU : Dec.-17, Marks 5**
4. *Explain in detail about use case diagrams.* **AU : May-18, Marks 6**
5. *Explain briefly the elements of Use Case diagram.* **AU : Dec.-18, Marks 13**
6. *Write the use of case modeling and explain in detail.* **AU : May-19, Marks 7**

2.12 Class Diagrams AU : May-11,12,16,17,18,19, Dec.-11,12,13,14,15,16,17,18, Marks 16

A class represents a collection of objects having the same characteristic properties that exhibit common behaviour. It gives the blueprint or description of the objects that can be created from it.

- There are **three strategies** that can be adopted for finding the conceptual classes -
  1. Reuse or modify existing models for finding the conceptual classes. This is the easiest way because in the existing model the domain models and data models are well designed. The common domains can be used to identify the conceptual classes.
  2. Use of category list.
  3. Finding out the name phrases from the problem description. The name phrases denote the conceptual classes.
- Following are the **steps to create a domain model used** for representing the conceptual classes.
  1. Find out the conceptual classes using the above described three strategies.
  2. Draw the UML class diagram for these classes.
  3. Add associations and attributes to these classes.
- Conceptual classes can be identified using the **category list**. Some commonly used categories are as shown in the following table -

Category	Example
Physical Objects	Ticket, Die, Board, Register, Item, Book.
Place of service	Store, Airport,ATM.

Catalog	ProductCatalog, BookCatalog.
Role	Passanger, Accountant, Student, Customer, Librarian, Administrator.
Financial Items	Credit Card, Cash
Schedules	DailySchedule, RepairSchedule.
Record keeping	Database, Register, Ledger.

- The conceptual classes can be identified by identifying the **noun phrases**. This technique is suggested by Abbot. This technique is called **linguistic analysis**.

Example : ATM System

Problem Description : Consider an ATM System in which the customer with an ATM card can establish session with the system. First of all he inserts the card into the system and enters the PIN. The console of the system reads the card and authenticates the customer. The system has console containing display and keyboard.

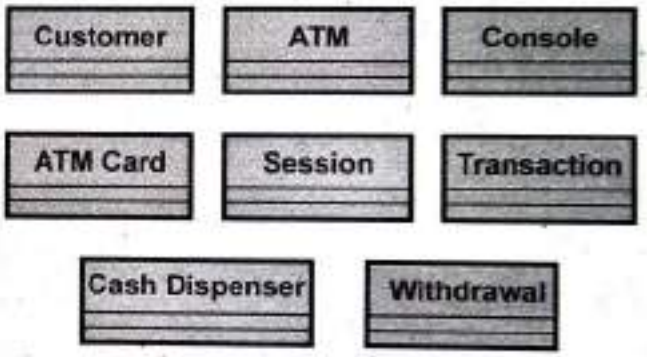


Fig. 2.12.1 Initial ATM system

The session with the system can be accomplished by the transaction. Using the withdrawal transaction the customer can get the requested amount through cash dispenser.

From above problem description, we can obtain the conceptual classes. The conceptual classes can be identified by identifying the noun phrases.

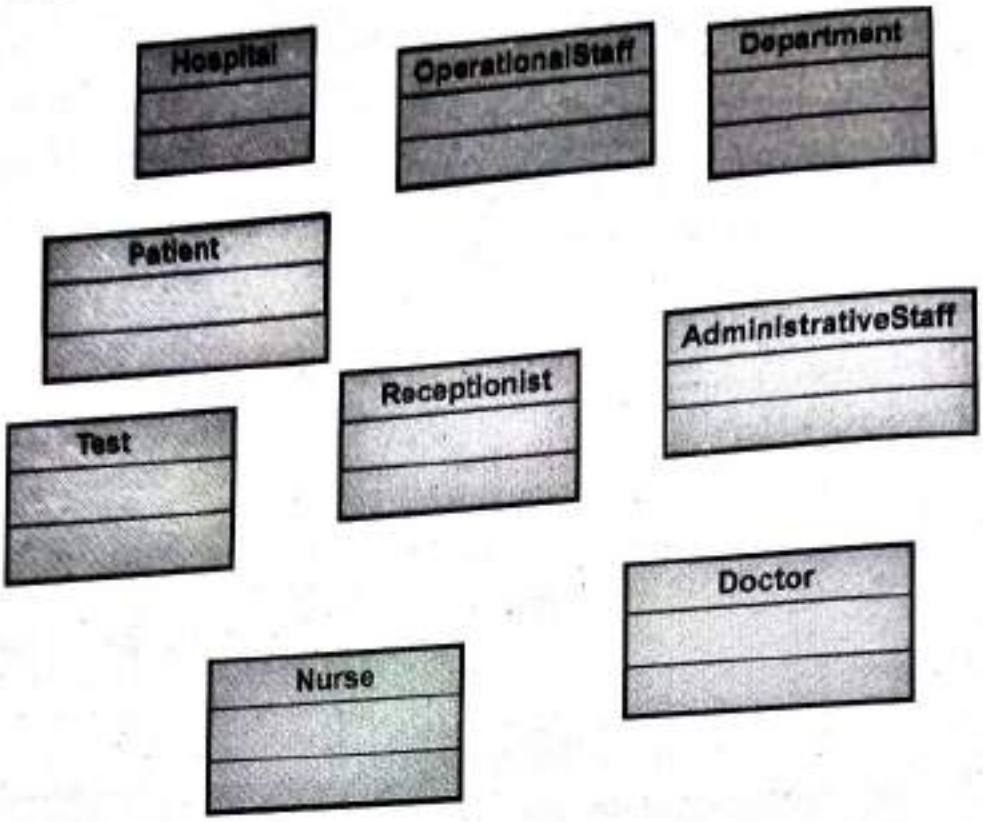
The UML class diagram using these conceptual classes can be - as shown in Fig. 2.12.1.

The associations and attributes for the above class diagram is not shown.

Example 2.12.1 *For the Hospital Management system draw conceptual class diagram(overall system)*

AU : Dec.-12, Marks 8

Solution :



2.12.1 Attributes

Attribute is a logical data value of an object. It is necessary to identify the attributes conceptual classes so that information required by the system can be represented.

The attributes are needed when some specific information needs to be remembere

For example -

- **Customer** needs **ID**
- **Account** has account type
- **Flight** has **Date/time**
- **Student** has **name and address**

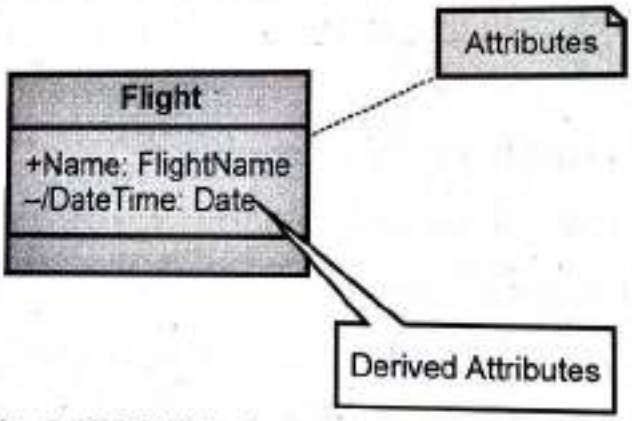


Fig. 2.12.2 Attributes and derived attributes

2.12.2 Relationships

Various relationships in class diagram are –

1. Associations
2. Inheritance
3. Aggregation
4. Composition

2.12.2.1 Associations

- The association is the most common type of relationship used UML. The association is shown between the classes or instance of classes. For example -

In above Fig. 2.12.3 the relationship between the customer and the bank account is shown by the association relationship. This can be read as **"One Customer can have multiple Accounts in the bank"**.

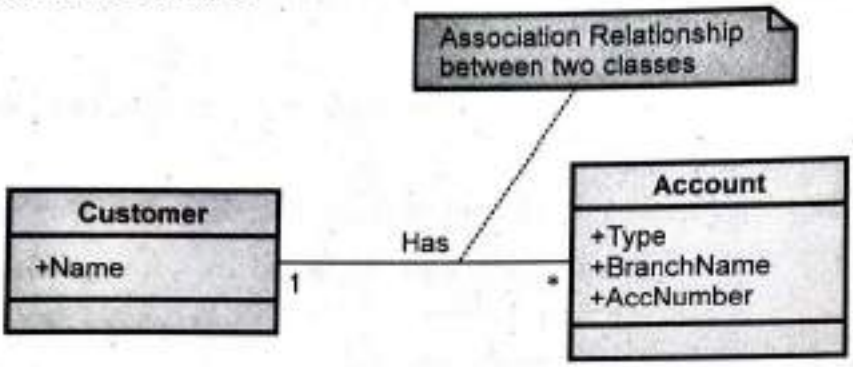


Fig. 2.12.3 Association relationship

• When to show an Association?

The association relationship is used in the system design when we want that the relation between two object has to remembered for some time.

The association relationship can be defined from the common association list. Use of too many association relationships in the class diagram must be avoided.

• How to represent Association in UML?

The association is represented by a line between the classes. The **association name** must be denoted by the string with first letter capital.

The ends of association must represent the multiplicity expression indicating the numerical relation between the classes.

- Using the simple association names like **Has** or **Uses** leads to less understanding of the domain model. Hence more appropriate verbs can be used to denote the association name. Most commonly used association names are -

1. A uses B
2. A has B
3. A is a member of B
4. A records B
5. A issues/returns B
6. A communicates with B
7. A is followed by B
8. A is owned by B

- **Example**

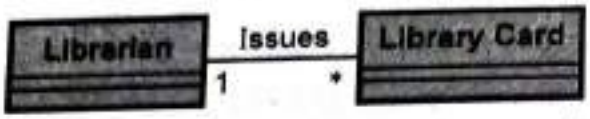


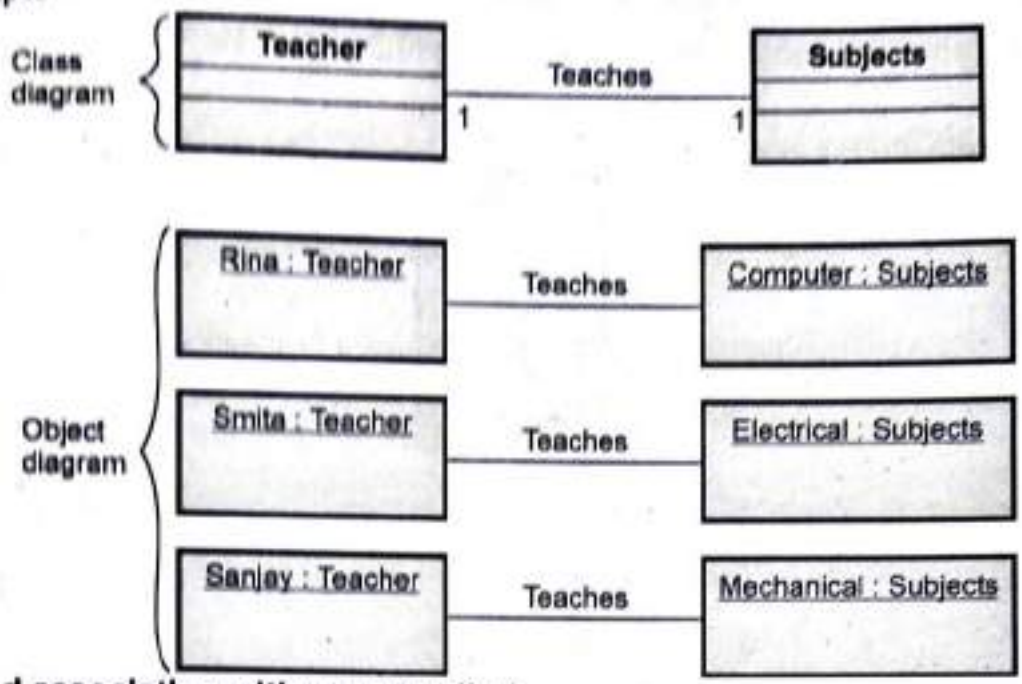
Fig. 2.12.4

- **What is multiplicity in an association?**
  - Multiple objects can be related to some objects. The multiplicity represents "how many" objects are connected.
  - The multiplicity can be written as expression. It describes "one" or "many" objects associated with other object.
  - The multiplicity is specified at the end of the association link.
  - Following are the notations that can be used to denote the multiplicity of associations -

Notations	Description
1	Only one instance
0..1	Zero or one instance
*	Many instances
0..*	Zero or many instances
1..*	One or many instances
2..10	You can specify the integer range

- There is a difference between two terms - the **multiplicity** and **cardinality**. The multiplicity is the constraint on the collection of the associated **objects** whereas the cardinality is the count of the objects that are in collection. The multiplicity is actually the constraint on the cardinality.
- While making the class model first decide about the classes and their associations and then think about their multiplicity.

For example -



- **How to find association with common list?**

Following table shows different associations that can be identified from the common association list -

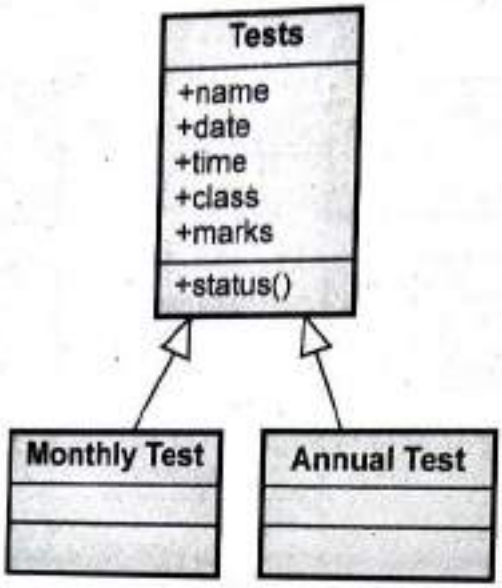
Category	Example
A and B are transactions and they are related with each other	**Reservation** is related to the **Confirmation** of Reservation
A is a role and B is transaction	**Customer** makes the **Payment** **Librarian** issues **Book** **Librarian** issues **Library Card**
A is logical or physical part of B	**Customer** has **Library Card**
A is logically or physically contained within B	**Book** is present in **Shelf**
A is a member of B	**Student** is a member of **Library**
A uses/manages/owes B	**Customer** uses **Account**

2.12.2.2 Inheritance

- Inheritance or **Generalization** is a relationship between superclass and one or more subclasses.
- This is a kind of relationship in which the properties of parent class are inherited by the child class.
- In this relationship the **superclass** is more **general** and the subclasses are more specific. The subclasses use the common properties and methods of the superclass but at the same time adds specific attributes, operations and associations.
- This is **is-a** kind of relationship because each instance of subclass is an instance of superclass.

- The inheritance or generalization is represented by an hollow arrow head as given below -

- For example



2.12.3 Aggregation and Composition

Aggregation and composition are the kind of associations.

Aggregation

Aggregation is used to represent the whole-part relationship. It normally posses the Has-a relationship. It is denoted as follows -

Fig. 2.12.5 Aggregation relationship

For example - DVD player and Car are the two classes that can be associated by the aggregate relationship. Note that the DVD player can exist without Car and Car can exist without the DVD player. We can read the relationship as "DVD player is a part of Car".

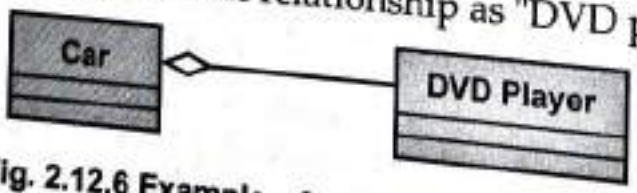


Fig. 2.12.6 Example - Aggregate relationship

Composition : Composition is a special kind of aggregation which represents the whole-part relationship. To be more specific, a **restricted** aggregation is called composition. When an object contains the other object, if the contained object cannot exist without the existence of container object, then it is called composition. It is denoted as follows -

Fig. 2.12.7 Composition relationship

For example - Steering and Car are the two classes those can be associated by composite relationship. This is restricted relationship because the Car can not exist without Steering.

We can read the relationship as "Steering is a part of Car"

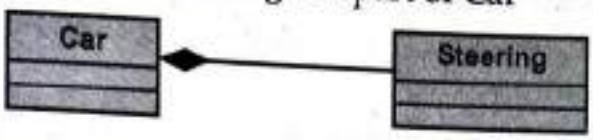
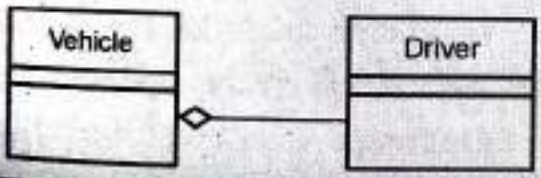
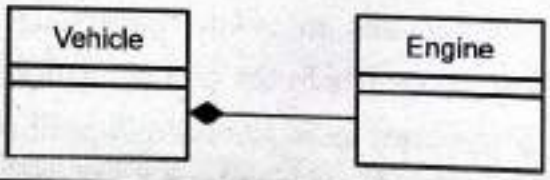


Fig. 2.12.8 Example - Composite relationship

Difference between aggregation and composition

Aggregation	Composition
The relationship between child and parent class is represented in aggregation relationship where in child can exist independently of the parent.	The relationship between child and parent class is represented in composition relationship where in child can not exist independently of the parent.
It is weak association.	It is strong association.
Deletion of owning class doesn't affect system.	If owning class is deleted it will significantly affect the system.
The hollow diamond is used in arrow notation.	The solid diamond is used in arrow notation.
For example 	For example 

2.12.4 Visibility

The visibility specifies the how the attributes and operations are visible in the system. There are four levels of visibility -

1. **public** : The public element is visible to all elements that can access the classifier. It is denoted by '+' symbol.
2. **Protected** : The protected element is visible to all the elements that have a generalization relationship to the namespace that owns it. **Protected** visibility is represented by '#' symbol.

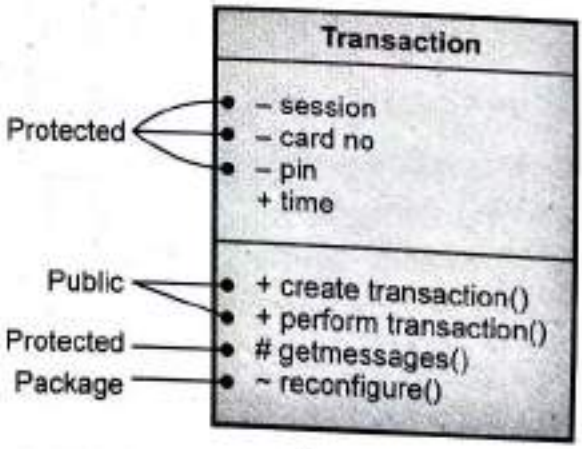


Fig. 2.12.9 Visibility

3. **private :** A private element is only visible inside the namespace that owns it. Private visibility is represented by '-' symbol.
4. **package :** Only the elements that are declared in the same package can use this feature. It is specified using the symbol '~'.

2.12.5 Qualified Association

The qualified association has **qualifier** which is used to select particular object from the set of objects. For example For selection of particular **Student** from the **Course** the qualifier will be **StudentID**

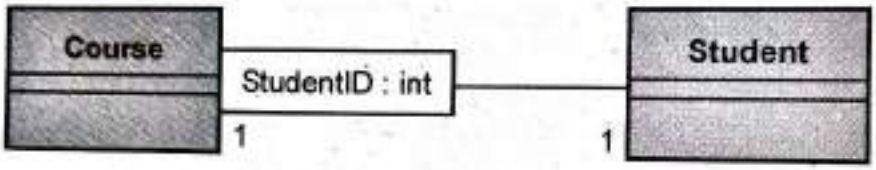


Fig. 2.12.10

A qualifier is a property which defines **selection key.** UML allows having a qualifier on each end of a single association. The multiplicity can be associated with the association link.

2.12.6 Association Class

The abstract class is a class that allows the association to be a class itself. That means when two classes are related with each other by an association link, then the association itself can have attributes and operations. Hence this association can be represented by a class. For example- in the following illustration the **Enrollment** is an association class for the classes **Student** and **Course.**

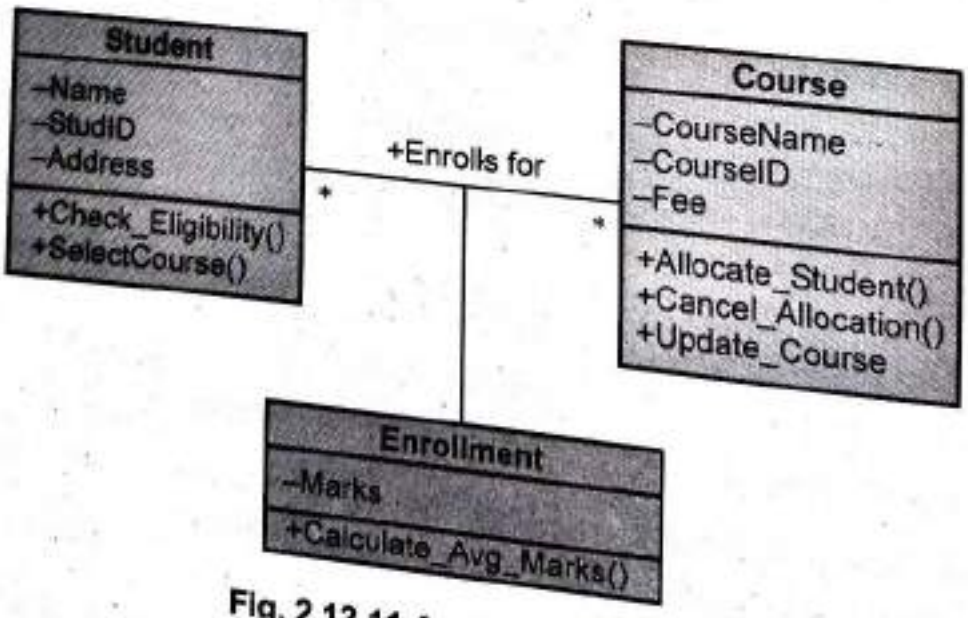


Fig. 2.12.11 Association class

2.12.7 Singleton Class

When there is a situation that only one instance of a class is instantiated then it is called singleton instance. In the UML class diagram such a class is marked with "1" at the upper right corner in the name compartment. Using singleton pattern exactly one object is needed to co-ordinate actions across the system. This concept restricts the instantiation to a certain number of objects.

For example -

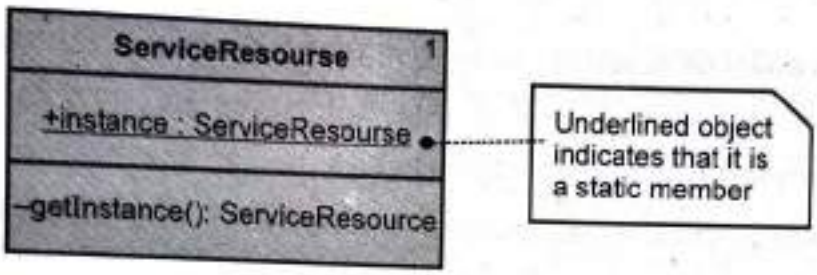


Fig. 2.12.12 Singleton class

2.12.8 Active Class

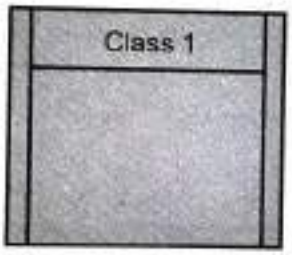


Fig. 2.12.13 Active Class

An Active Class indicates that, when instantiated, the Class controls its own execution. Rather than being invoked or activated by other objects, it can operate standalone and define its own thread of behavior. In other words this class initiates a thread or a process. For example - CommandProcessor is an active class when user submits some command then based on that command particular process is invoked for execution.

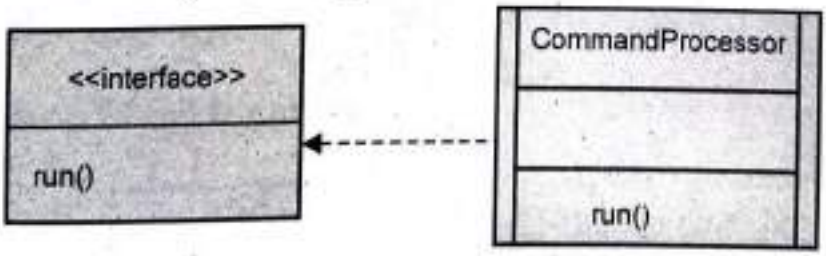


Fig. 2.12.14

2.12.9 When to Use Class Diagrams?

Following are uses of Class diagram :

1. **For modeling the vocabulary of the system the class diagram is used.**
   - When we want to decide which abstractions must be the part of the system and which abstractions should be outside the boundary, we design the class diagram.
   - The class diagram is used to represent the abstractions and their responsibilities.

2. **For modeling the collaborations of the system the class diagram is used.**
   - Collaborations is a collection of classes, interfaces and other elements that work together to provide some co-operative behavior.
   - The class diagram specifies such collaborative classes and their relationships.
   - For example -

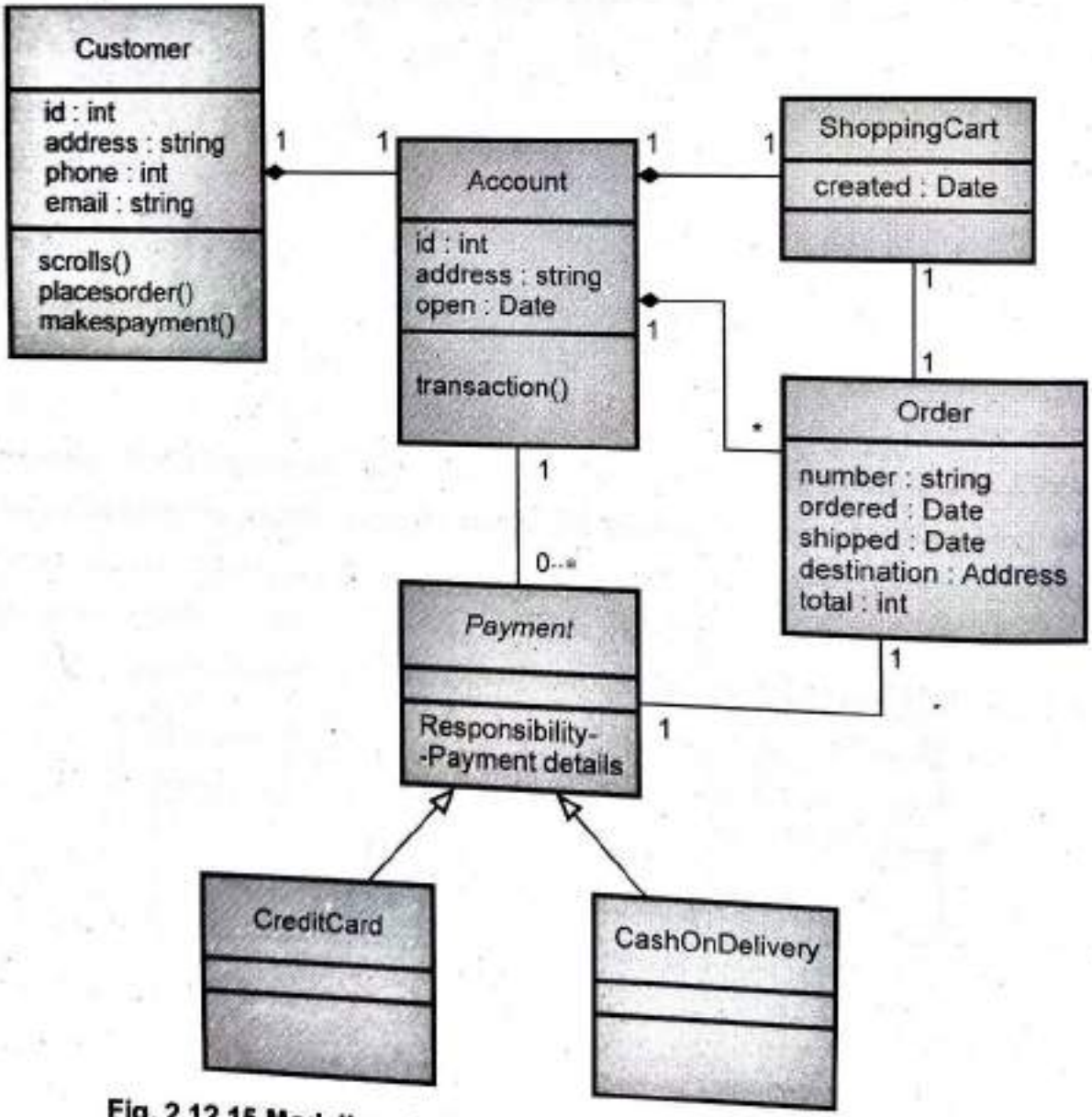


Fig. 2.12.15 Modeling simple collaborations for online shopping

3. **For modeling the logical database schema the class diagram is used.**
   - For conceptual design of the databases the schemas are used.
   - For example -

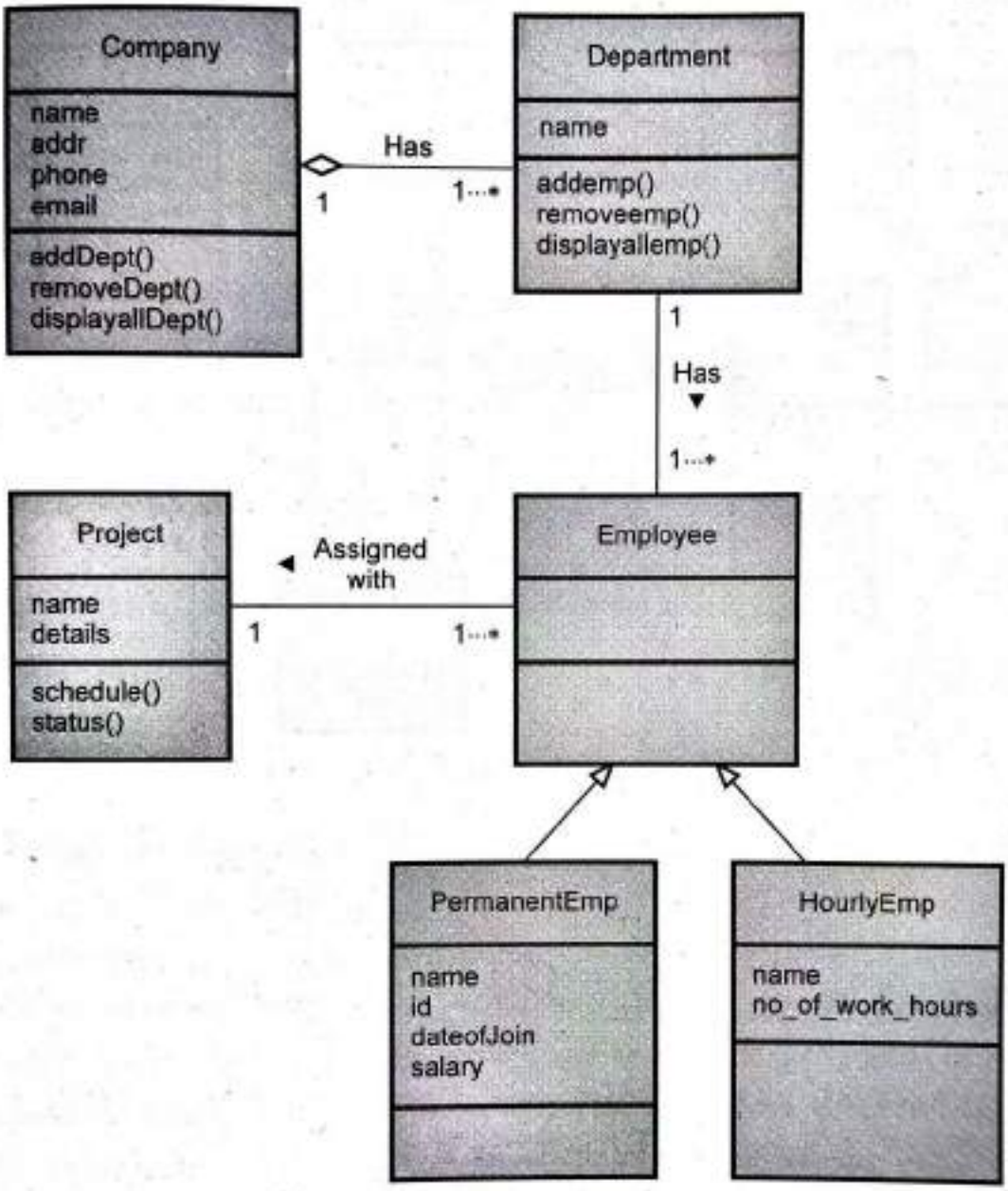


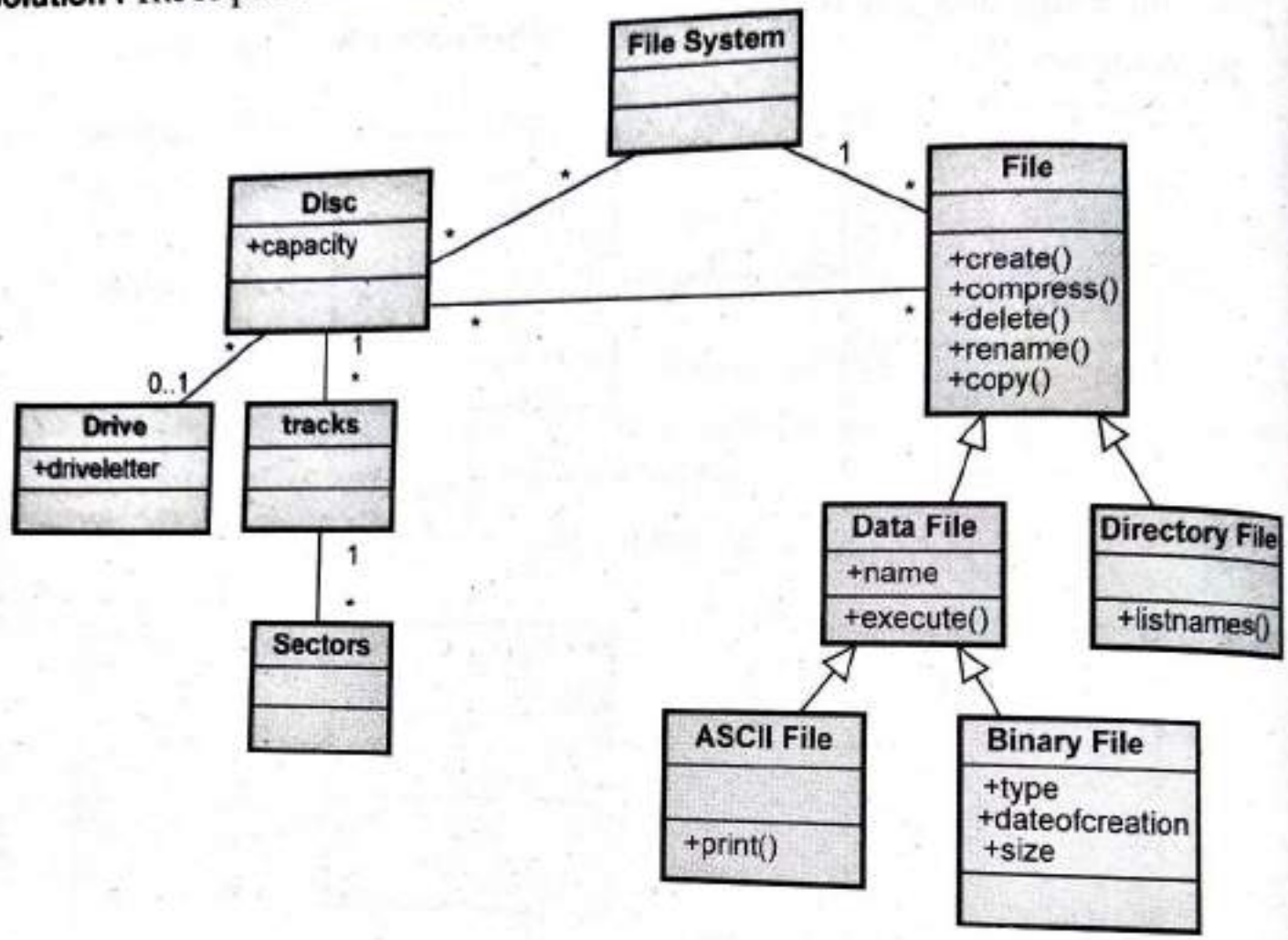
Fig. 2.12.16 Company Information System

2.12.10 Examples on Class Diagram

Example 2.12.2 *Prepare a class diagram for each group of classes. Add at least 10 relationships (association and generalizations) to each diagram.*

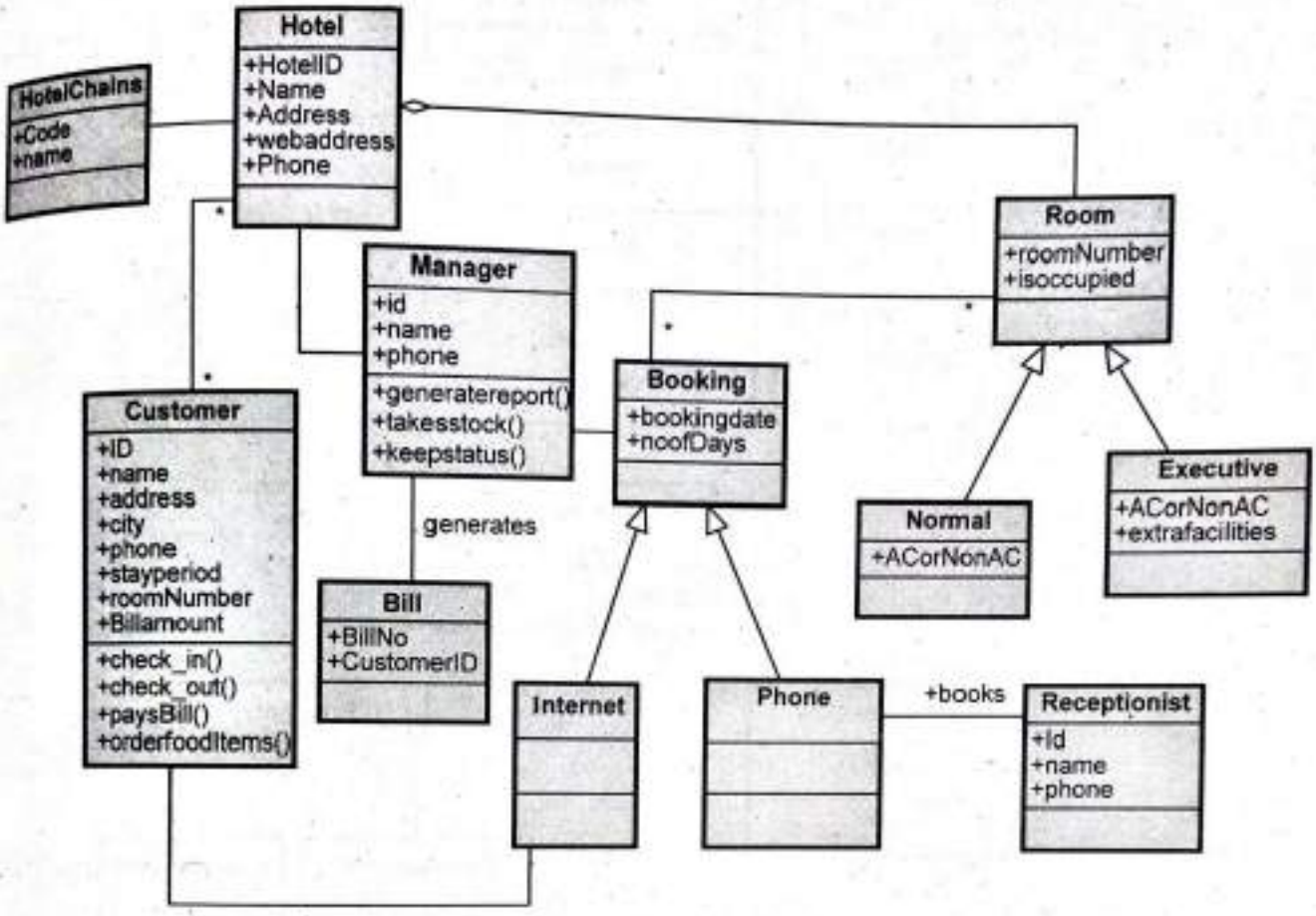
File system, file, ASCII file, binary file, directory file, disc, drive, track, sector

Solution : The required class diagram is given below -



Example 2.12.3 *Prepare a class model for the hotel management system. The system should support chain of hotels. A hotel contains two categories of rooms : executive and normal, both AC and non-AC. The customers of executive rooms can avail extra facilities like games, swimming, food service in rooms, etc. The booking is possible by Internet or by phone. If the booking is through phone, process is done by receptionist and if booking is done through Internet the process is carried out by customer through hotel website. Depending on the number of days customer stays, appropriate bill is generated. The bill also contains amount for transport, food and other facilities enjoyed by the customer along with necessary taxes. The manager should be able to generate reports like list of customers staying in the hotel, list of rooms empty, monthly/yearly income, etc.*

Solution :



Example 2.12.4 *Convert the following into a class diagram with appropriate classes, relationships, multiplicity.*

A computer program has many statements. An expression is a statement. A function is statement. An expression contains a variable, a constant and operator. A relational operator is an operator. An arithmetic operator is an operator. A function has an argument list, a return type.

Solution :

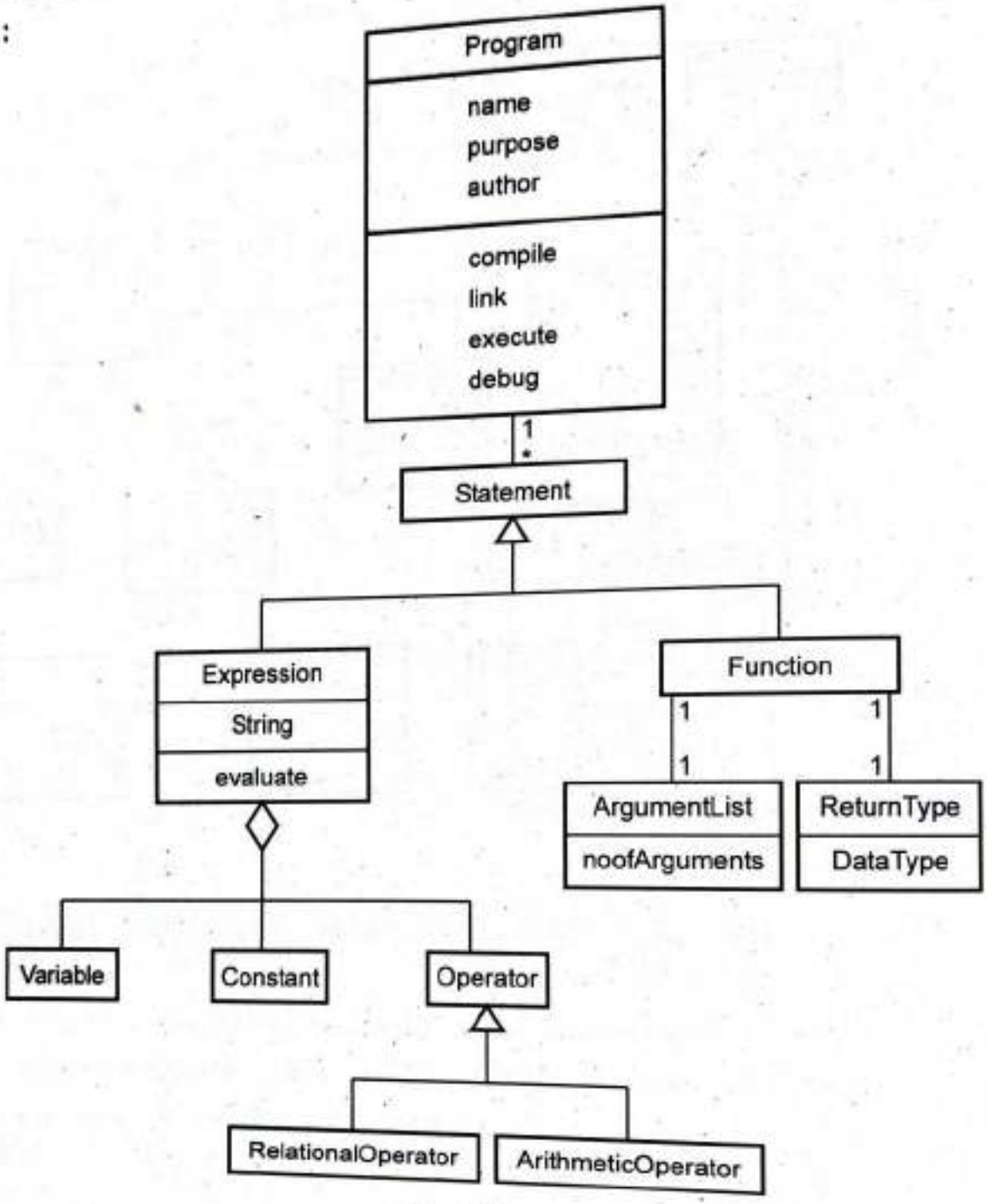


Fig. 2.12.17

Example 2.12.5 *Convert the following into a class diagram with appropriate classes, relationships, multiplicity.*

Car dealer sales cars. A car is owned by an owner. The owner has an address. The owner can be a person, a company or a bank. A car loan may be involved in the purchase of a car. Bank provides the loan.

Solution :

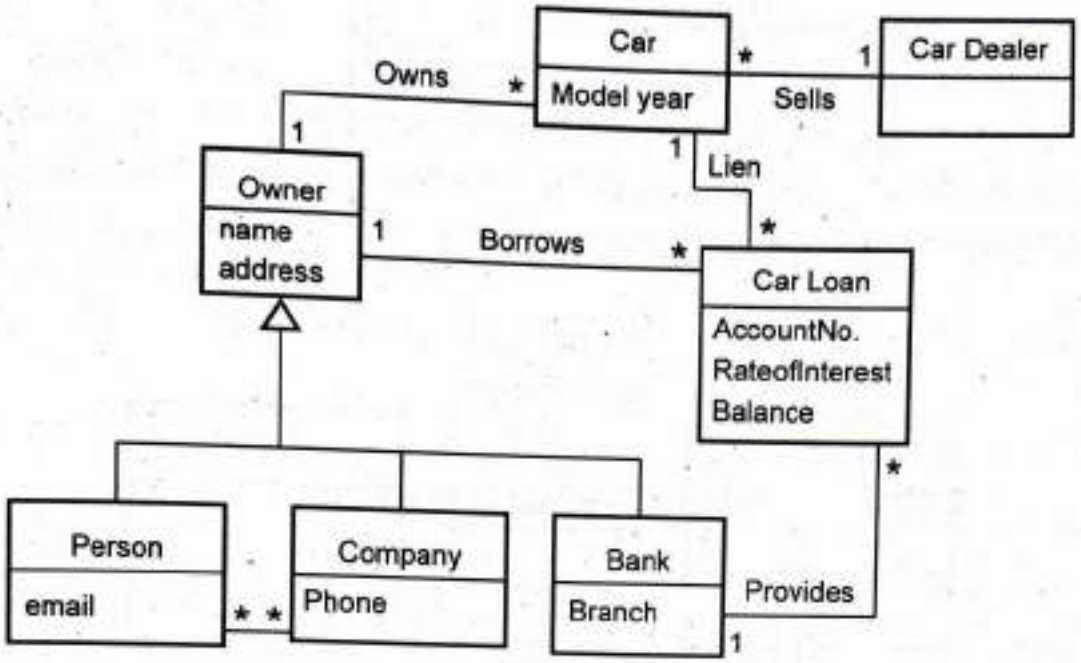


Fig. 2.12.18 Class diagram

Example 2.12.6 *Convert the following into a class diagram with appropriate classes, relationships, multiplicity.*

A drawing object is a text, a geometric object or a group. Group can contain at least two geometric objects. Point, polygon are geometrical objects. A polygon is composed of an ordered set of points.

Solution :

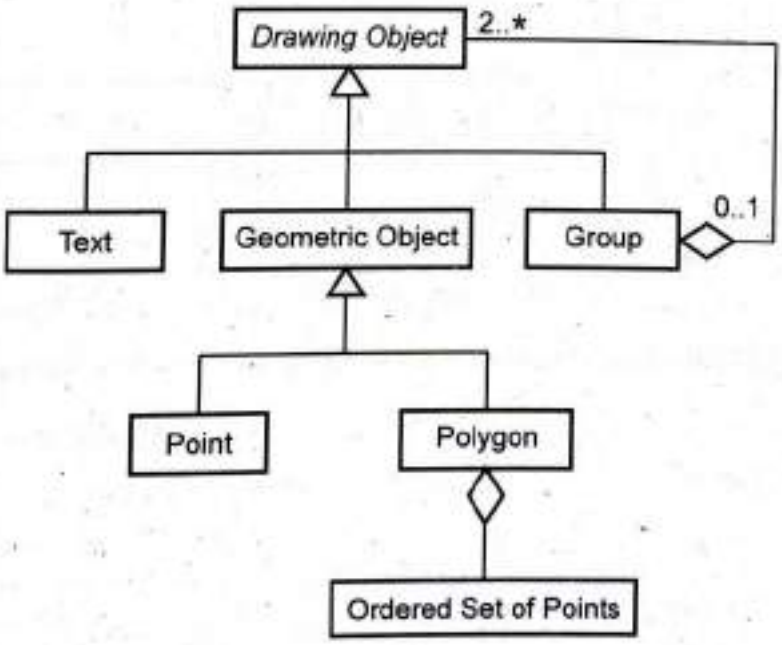


Fig. 2.12.19

Example 2.12.7 *Draw a class diagram for the given description with appropriate relationships, relationship names and multiplicity.*

A person owns multiple documents. Each document is composed of paragraphs that are, in turn, composed of characters. The copy operation propagates from document to paragraph to characters. Copying a paragraph copies all the characters in it. The operation does not propagate in reverse direction; a paragraph can be copied without copying the whole document.

Solution :

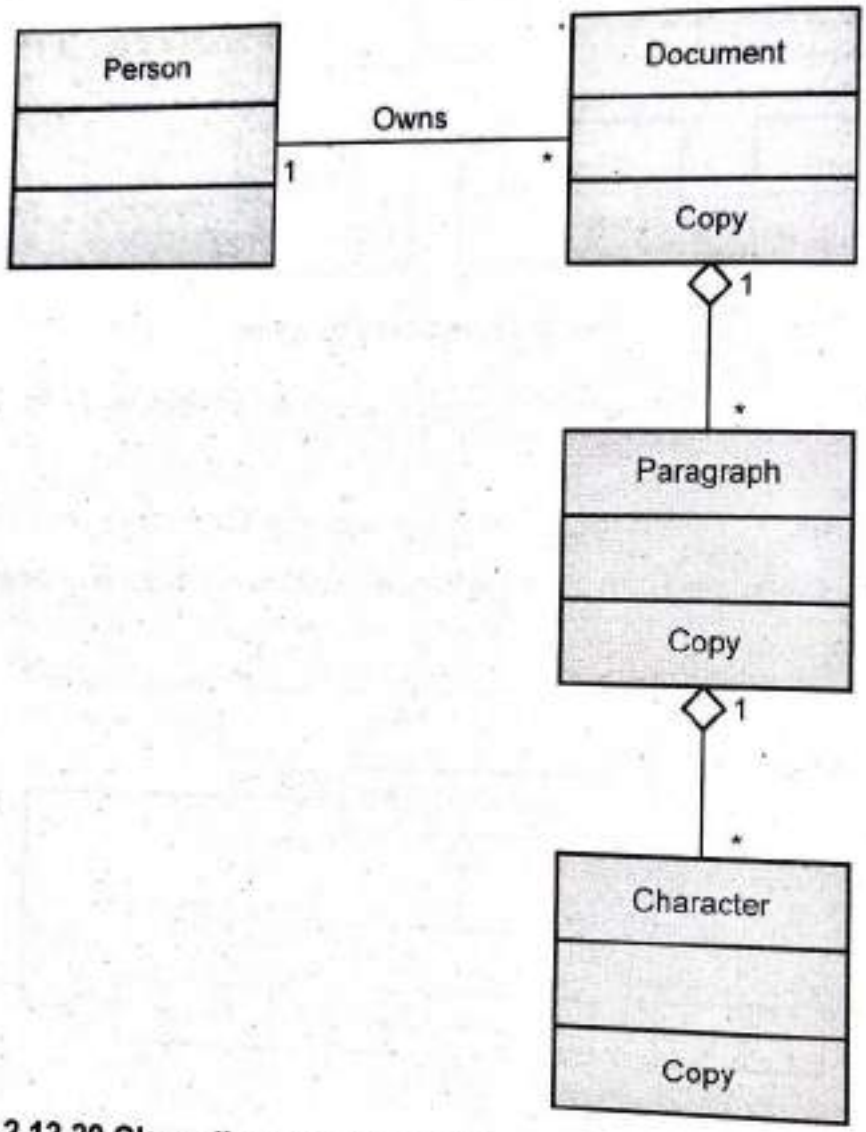


Fig. 2.12.20 Class diagram representing propagation operation

Example 2.12.8 *A project has three to five students. A projects has a guide can guide one to three projects. For this description draw a class diagram. From the class diagram draw an object to show two projects, seven students and one guide. Do not write any explanation, just draw the diagrams.*

Solution :

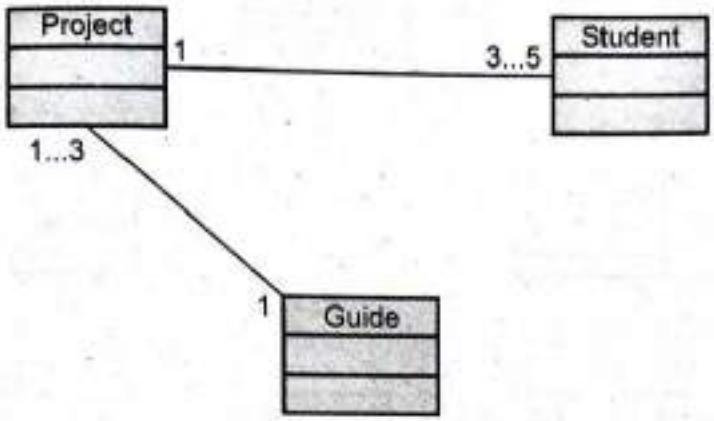


Fig. 2.12.21 (a) Class diagram

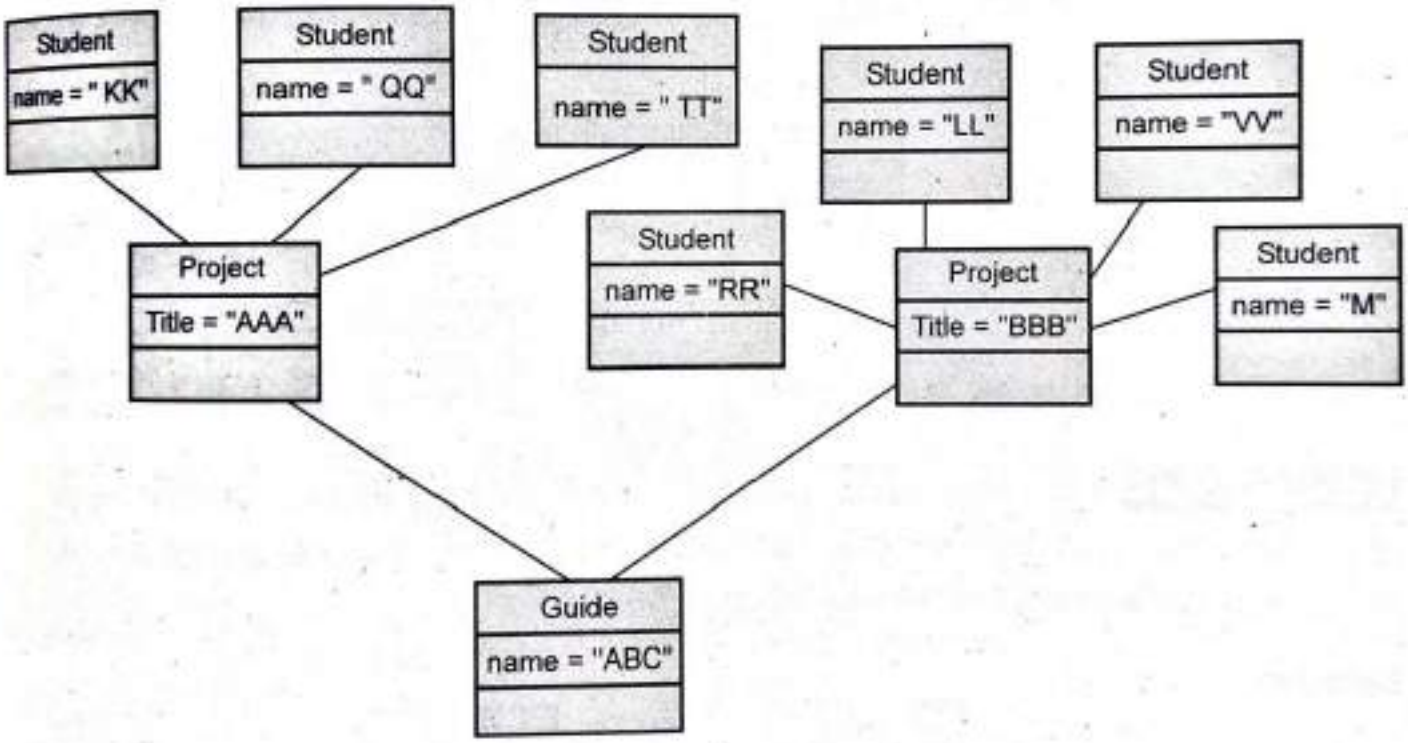


Fig. 2.12.21 (b) Object diagram

Example 2.12.9 *Convert the following description into a class diagram. Show classes, relationships and multiplicities.*

In a university there are different classrooms, offices and departments. A department has a name and it contains many offices. A person working at the university has a unique ID and can be a professor or an employee. A professor can be an associate or assistant professor and he/she is enrolled in one department. Offices and classrooms have a number ID, and a classroom has a number of seats. Every employee works in an office.

Solution :

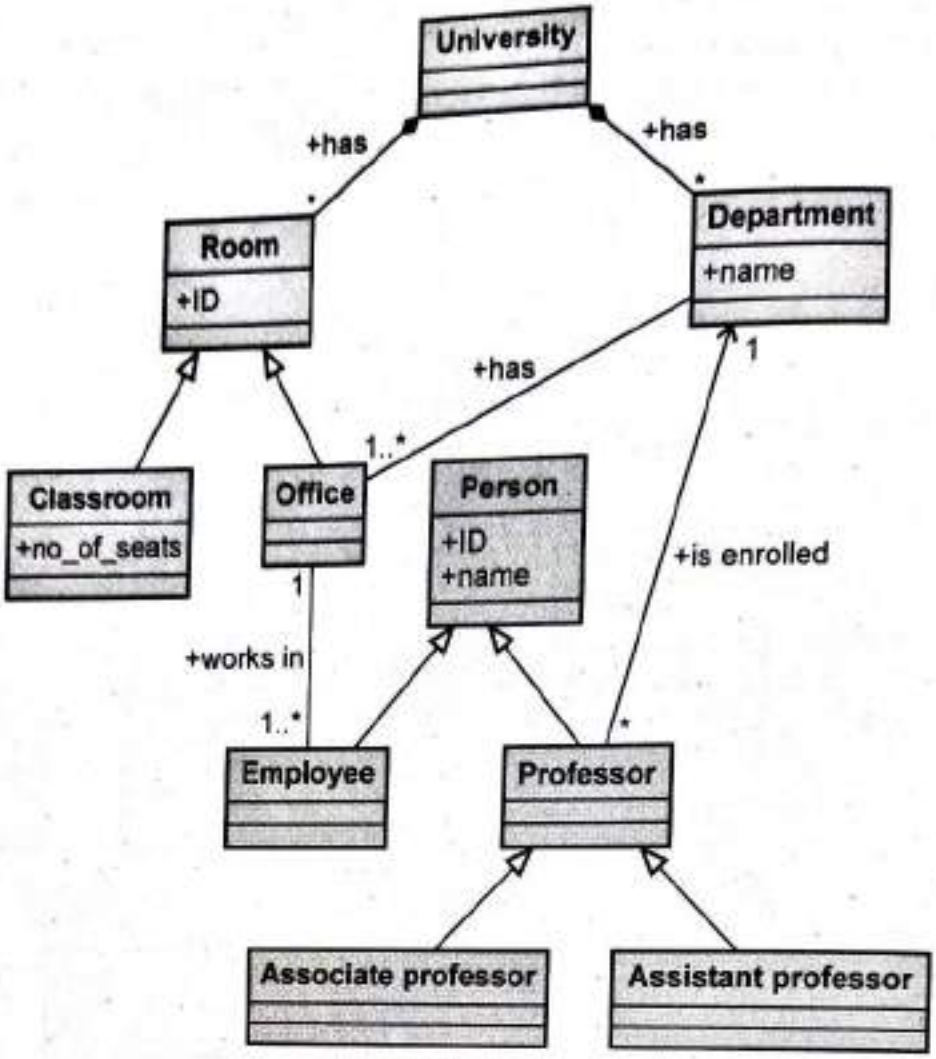
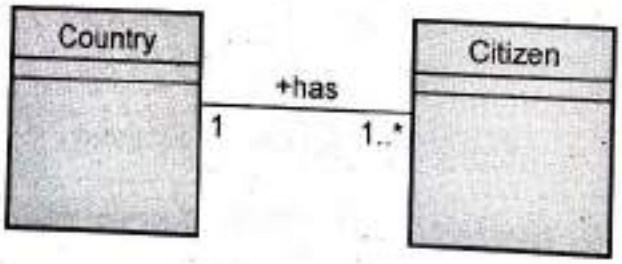


Fig. 2.12.22

Example 2.12.10 *A country has many citizens. One citizen among all can be identified by an Adhar card. Represent this as a qualified association. Two diagrams are expected. One without qualifier and the other with qualifier.*

Solution :



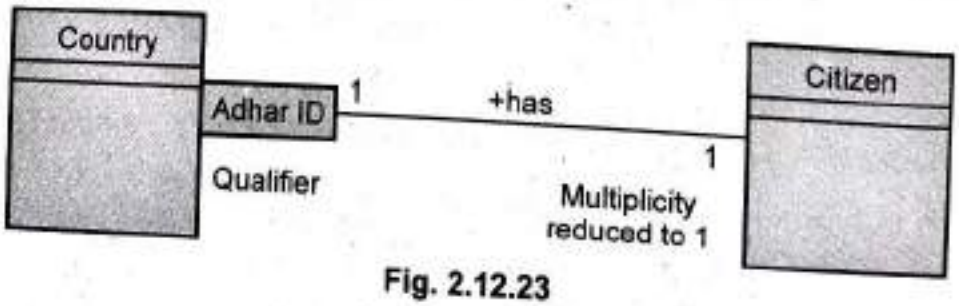
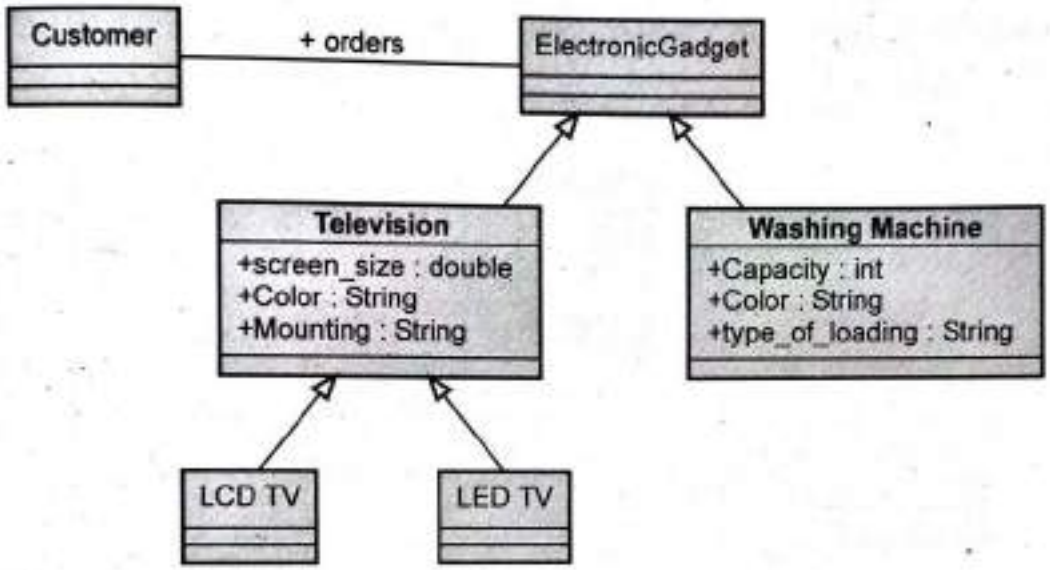


Fig. 2.12.23

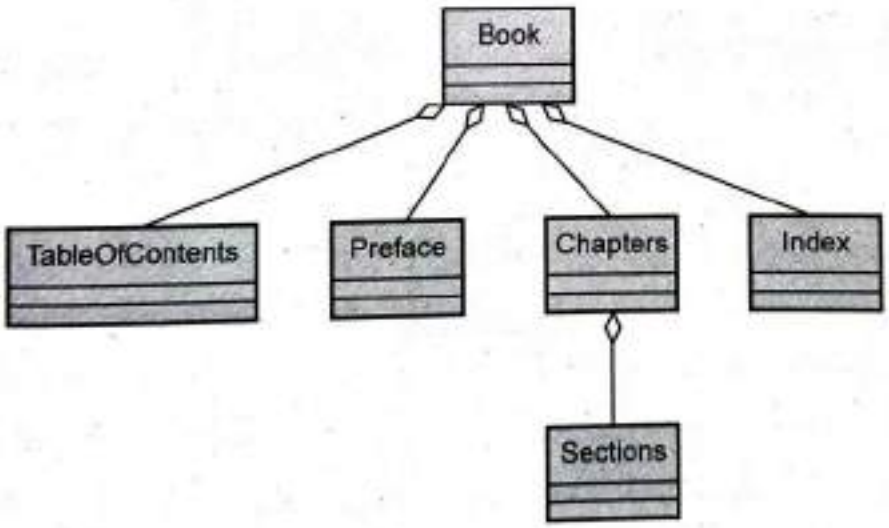
Example 2.12.11 *An electronic gadget shop has television and washing machines. Television has screen size inches, color, mounting(only wall or table mounts are possible). Televisions come in two types namely LCD and LED. A washing machine has a capacity in litres, color, type of loading(Top or front). A customer can order an electronic gadget. Draw a class diagram for this with attributes and relationships.*

Solution :



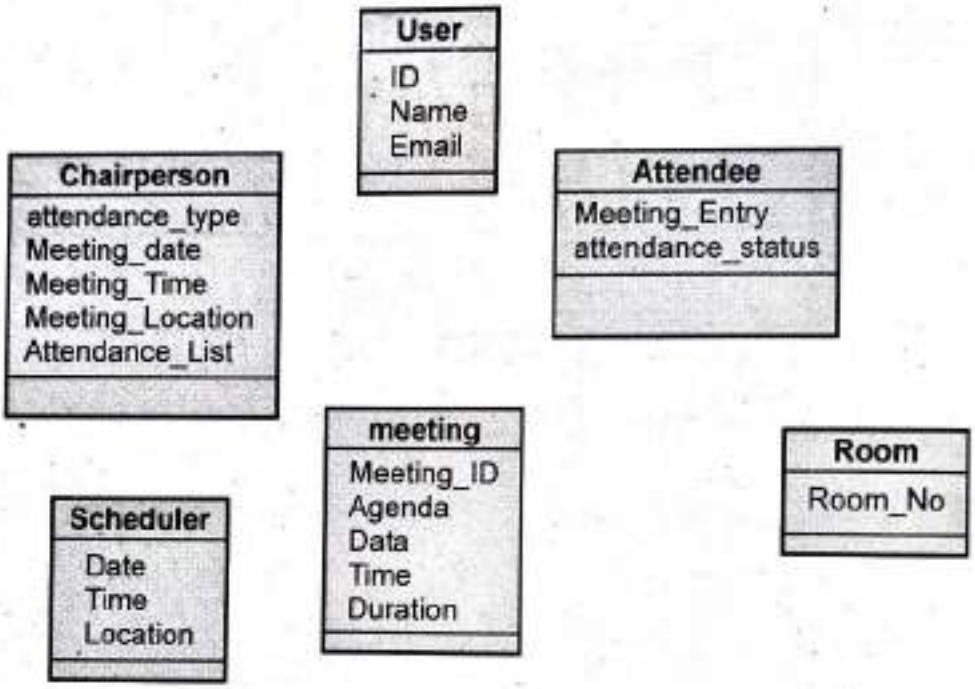
Example 2.12.12 *A book is composed of Table of Contents, Preface, many chapters and index. Chapters are composed of sections. Convert this using a class diagram.*

Solution :



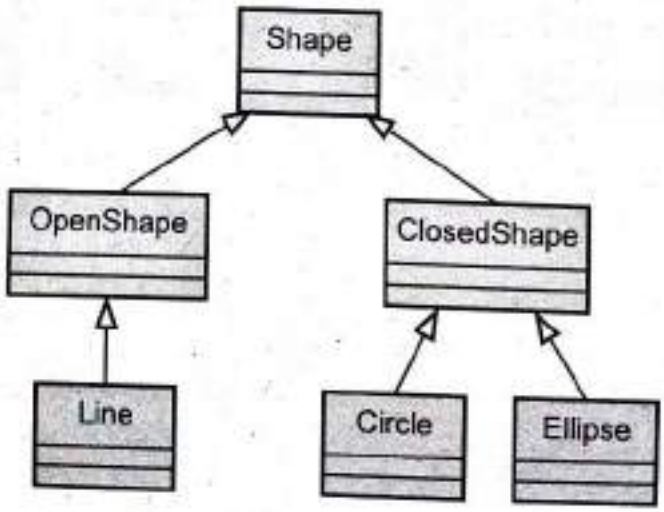
Example 2.12.13 *Analyze the following description and apply steps to find the candidate analysis classes and attributes from it. List all classes and all attributes separately. The scheduling software facilitates the meetings. When a user(chairperson) arranges a meeting, the software places a meeting entry in the schedule of each attendee. The chairperson uses the scheduler to reserve a room for the meeting, to identify attendees, and to find time on their schedules when everyone is available. The chairperson can indicate whether the attendance for each attendee is required or optional. The system tracks the attendance status for each attendee-whether an attendee has accepted or declined.*

Solution :



Example 2.12.14 *Draw a class diagram for generalization relationship using the keywords Shape, ClosedShape, OpenShape, Line, Circle, Ellipse.*

Solution :



Example 2.12.15 *Construct design for library information system which comprises and following notations (i) Aggregation (ii) Composition (iii) Association*

AU : Dec.-15, Marks 16

Solution :

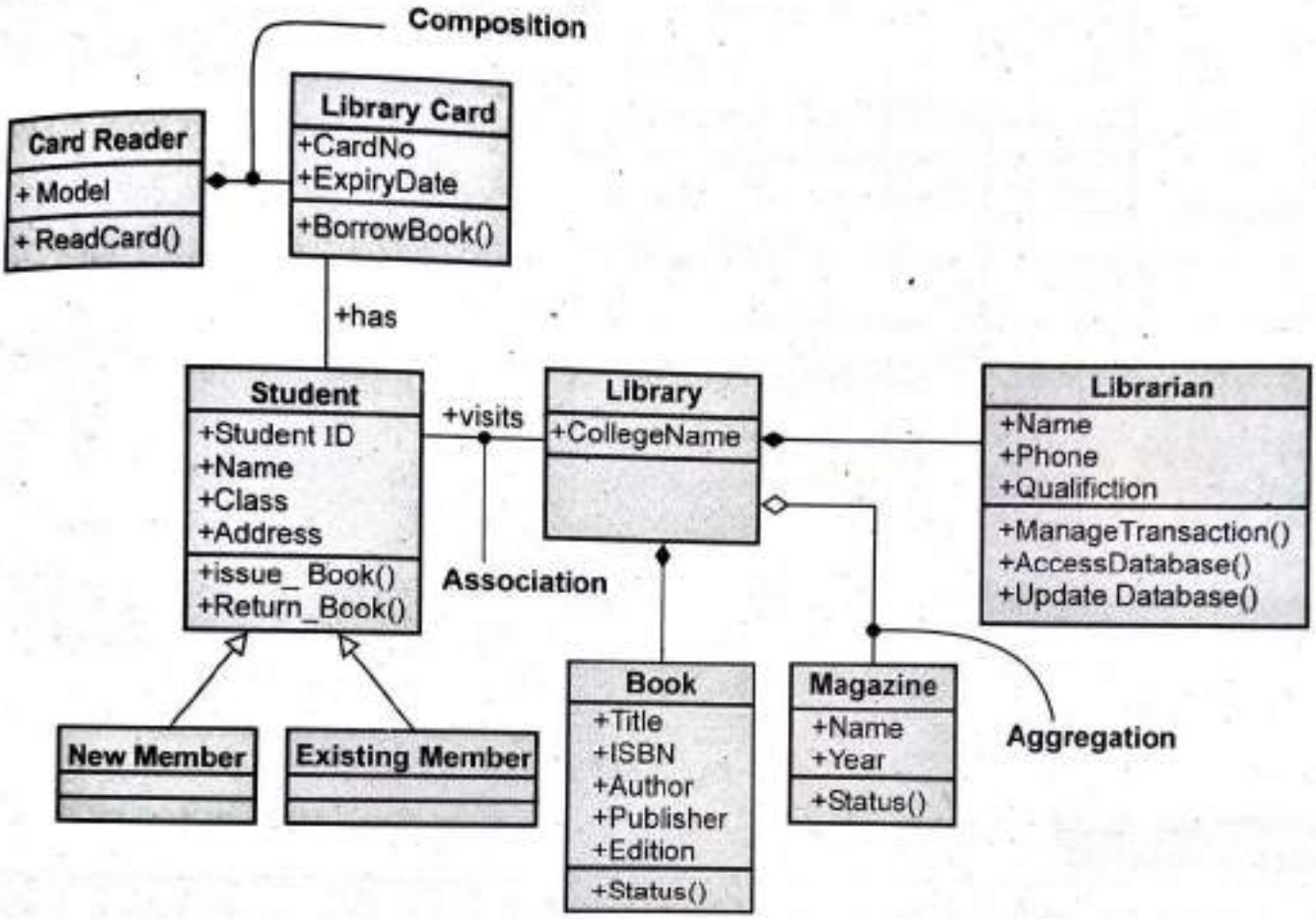


Fig. 2.12.24

Example 2.12.16 *Model a class diagram for "Banking System" State the functional requirements you are considering.*

AU : Dec.-17,18, Marks 15; Dec.-18, Marks 13

Solution : See Fig. 2.12.25 on next page.

Functional Requirements -

1) The valid customer must log in to the system using username and password.
2) The system should authenticate the user.
3) The system should allow the transfer the funds between authorised access.
4) The system should allow the customer to view the balance in his account.
5) An administrator is that person who is allowed for some editing for the internet banking system like add/cancel customer, check the transactions.

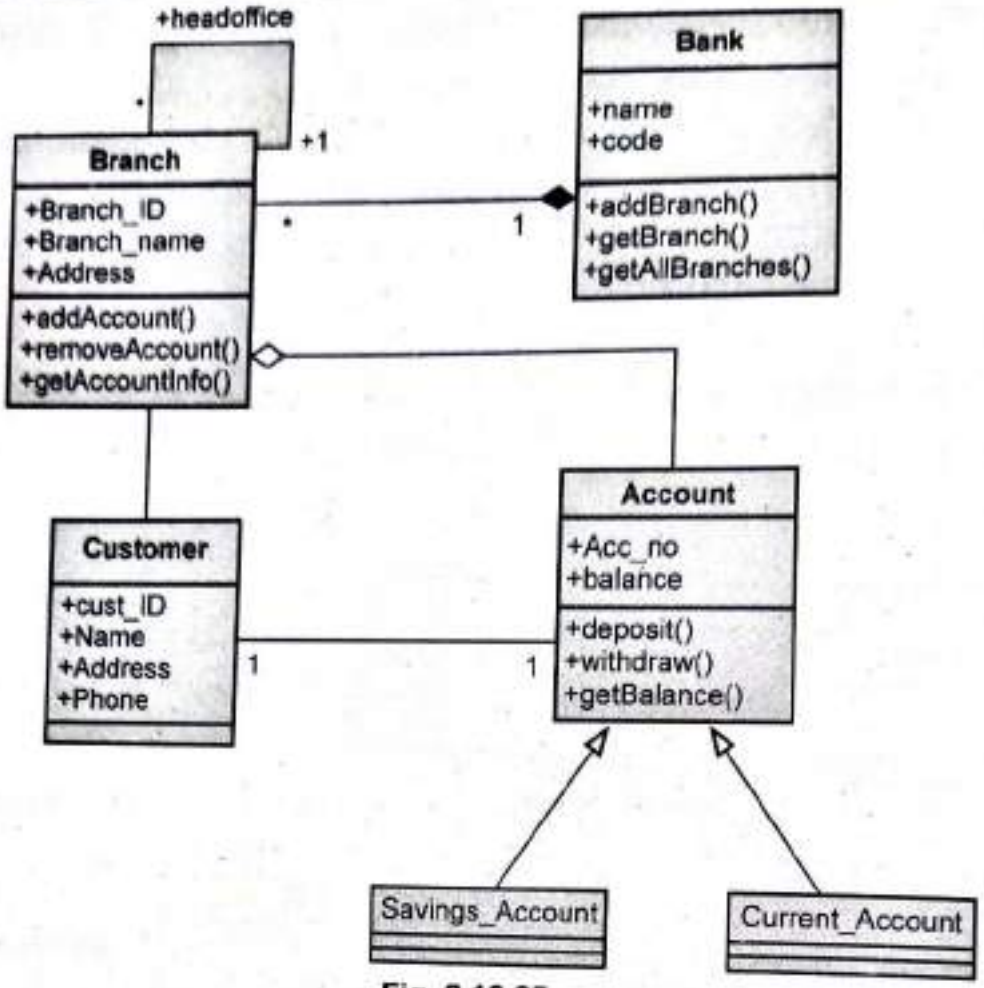


Fig. 2.12.25

Review Questions

1. *Describe the strategies used to identify the conceptual classes. Describe the steps to create a domain model used for representing the conceptual classes.*
   AU : May-11,12,18, Dec.-13, 14, Marks 16
2. *Explain the following with an example : Conceptual class diagram.* **AU : Dec.-11, Marks 8**
3. *Explain the guidelines for finding conceptual classes with neat diagrams.*
   AU : May-16, Marks 8, May-17 Marks 10
4. *Discuss in detail about the three strategies to find conceptual classes.* **AU : Dec.-16, Marks 8**
5. *Explain in detail about conceptual classes and description classes.* **May-19, Marks 5**
6. *What is multiplicity of an association ? Explain with an example the different types of multiplicities.* **AU : Dec.-17, Marks 7**
7. *Write a short note on the associations.* **AU : May-19, Marks 3**
8. *Define aggregation and composition* **AU : May-11, Marks 2**
9. *Explain association, aggregation and composition relationship in detail.*
   AU : Dec.-16, Marks 8
10. *Explain about Aggregation and Composition with examples.* **AU : May-17, Marks 6**
11. *Explain with an example aggregation and composition.* **AU : Dec.-17, Marks 6**
12. *Discuss about aggregation and composition.* **AU : May-18, Marks 7**
13. *Differentiate aggregation and composition.* **AU : May-19, Marks 8**

2.13 Interaction Diagrams

AU : May-11,14,18,19, Dec.-13,14,15,16,17,18, Marks 16

Interaction diagrams are created to represent interactive behavior of the elements of the system.

UML defines two types of interaction diagrams - **1. Sequence diagram** and **2. Communication or collaboration diagram.**

2.13.1 System Sequence Diagram

Definition : In UML, system sequence diagram is represented as the interactions made by the actors with the objects and operations initiated by them.

- The external input events are called the system events which are useful in analyzing the behavior of the system. System sequence diagram are useful UML notations for depicting the behavior of the system.
- The time based events or messages can also be represented by the sequence diagram.
- The sequence diagrams are useful tool to represent **what the system does and not how it does**. The sequence diagram is used as a tool to represent the system as a **black box.**
- Sequence diagrams are derived from the use cases and they represent one scenario.
- Sequence diagram is used to discover software architecture and interfaces.

Various elements of sequence diagram are -

Time line or Life line - The vertical line is called object's life line during the interaction.

Messages - The messages are represented by the line between the two objects. The messages will occur from top to bottom. Each message is labeled. This labeling can be an event that leads some action, self call, control information, condition checking and so on. It can be denoted as

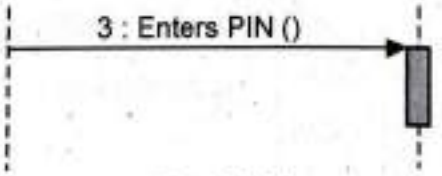


Fig. 2.13.1

Box - The boxes at the top of each time line generally represents the classes, objects or actors. Objects have labels in the standard UML format **name:ClassName,** where "**name**" is optional (objects that haven't been given a name on the diagram are called **anonymous objects**). Classes have labels in the format ClassName and actors have names in the format ActorName.

Fig. 2.13.2

Activation Box - The long thin boxes on the life line are called the activation boxes. These are also called as **method invocation boxes**. These boxes indicate that the processing is being done by the target object to fulfill a message. It is denoted as -

Fig. 2.13.3

X at the bottom - The cross at the bottom indicate that the particular object can be removed from the memory. It is represented as

Fig. 2.13.4

Conditional logic - The model can show the conditional logic by using the conditional messages.

The format of the sequence diagram will be like this -

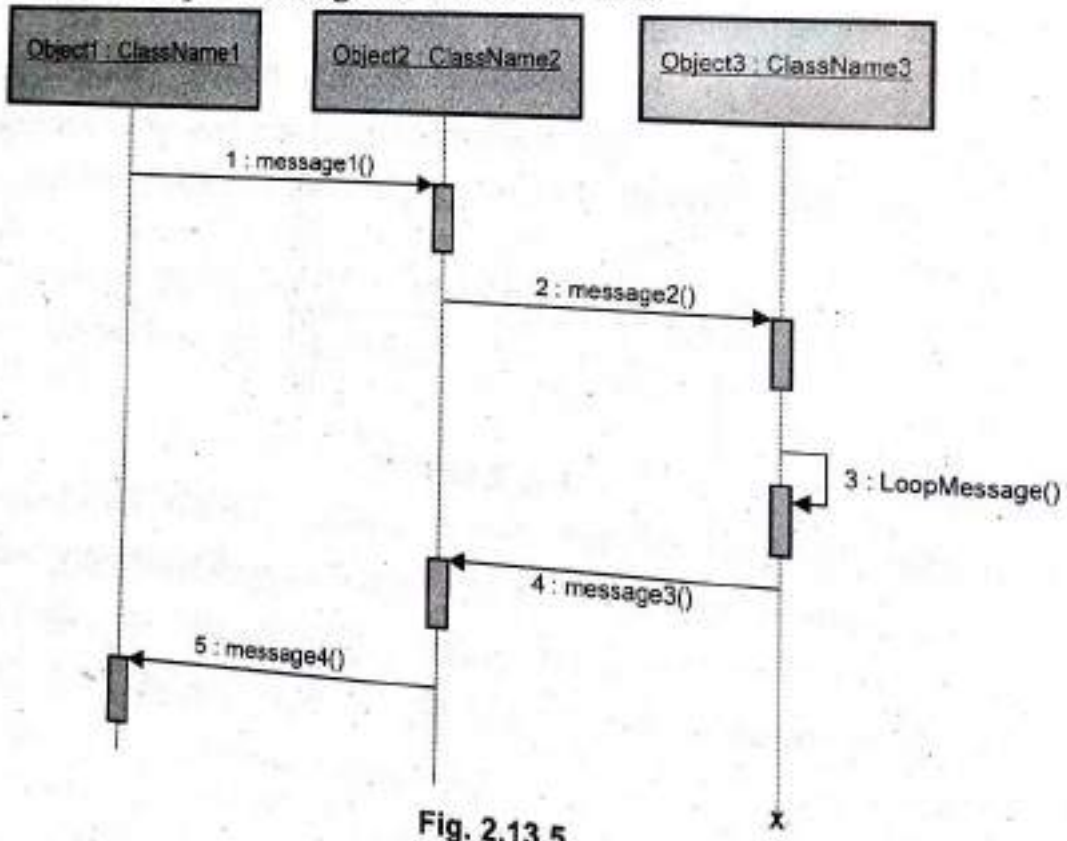


Fig. 2.13.5

2.13.2 Relationship between Sequence Diagram and Use Cases

- Sequence diagram is used to represent one scenario of the use case. Hence it can be derived by observing the use cases.

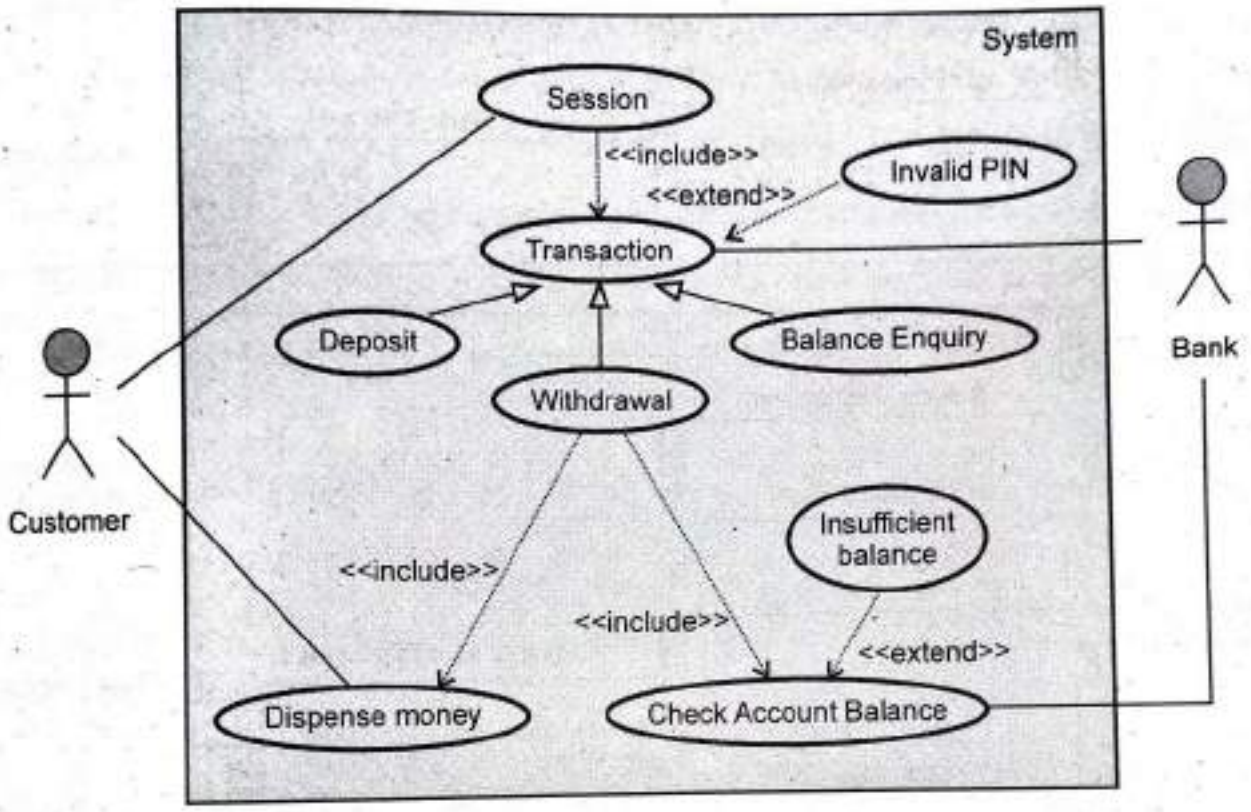


Fig. 2.13.6 Use case diagram for withdrawal of money

- Following is a use case diagram for the withdrawal of money from ATM system.
- Following is a system sequence diagram for **Withdrawal of money from ATM system.**
- One scenario of the use case can be represented using the system sequence diagram. The loops in the system can be shown using the **interaction frame.**

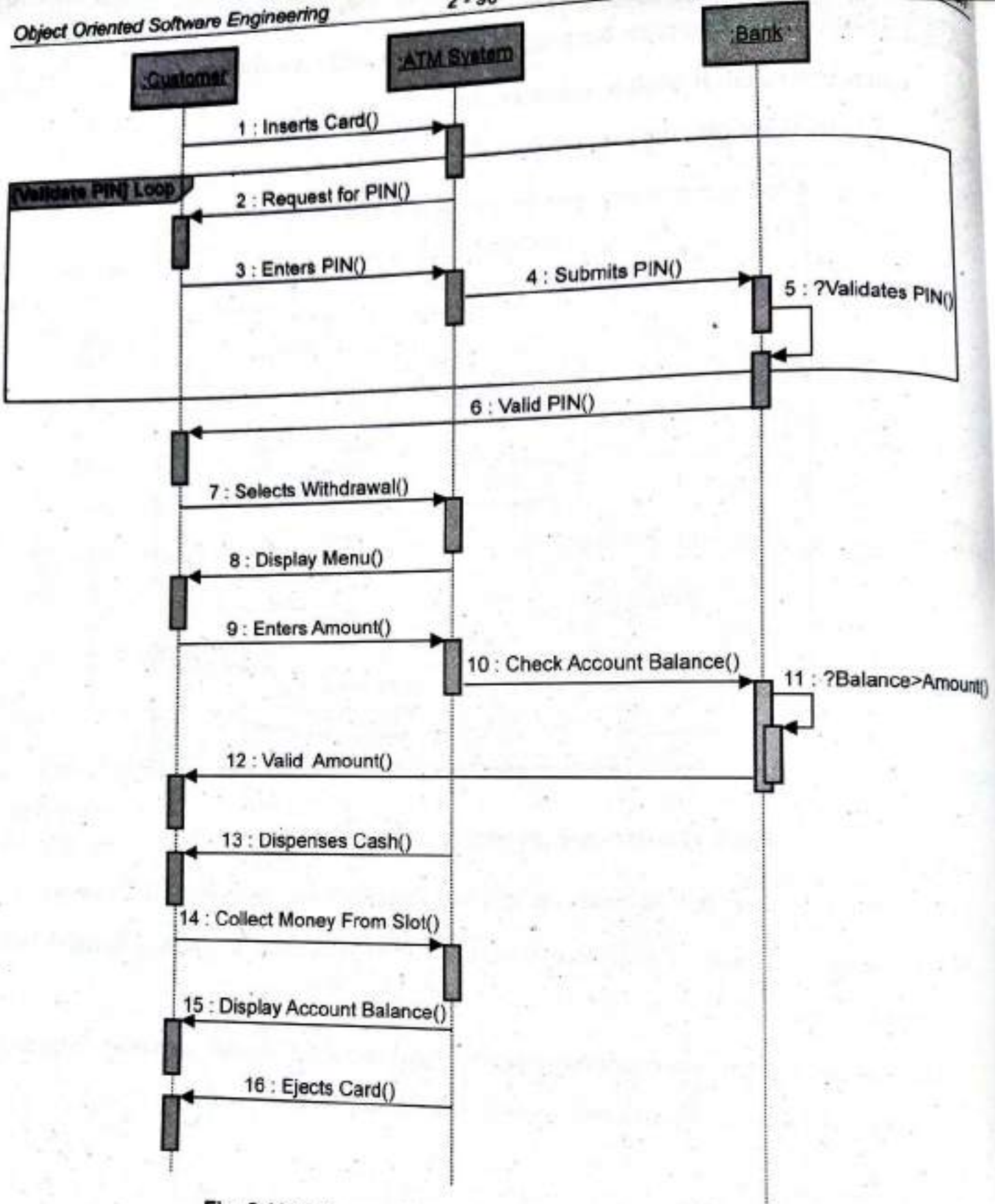


Fig. 2.13.7 Sequence diagram for withdrawal of money

Guideline

Following guideline is used while designing the SSD

1. The Sequence diagram generally represents only **one scenario of use case** model.

2. System events must be represented at the abstract level instead of representing in terms of physical input device.
3. The system events must be usually a verb. For example - display, enter, select, eject and so on.
4. The sequence diagram can represent the collaboration with other external systems. For example - For representing the scenario **withdrawal of money** the sequence diagram can be drawn for the objects **Customer** and the **ATM System.** The **Bank** can be external system in this model and the interaction with such an external system can also be represented using sequence diagram.

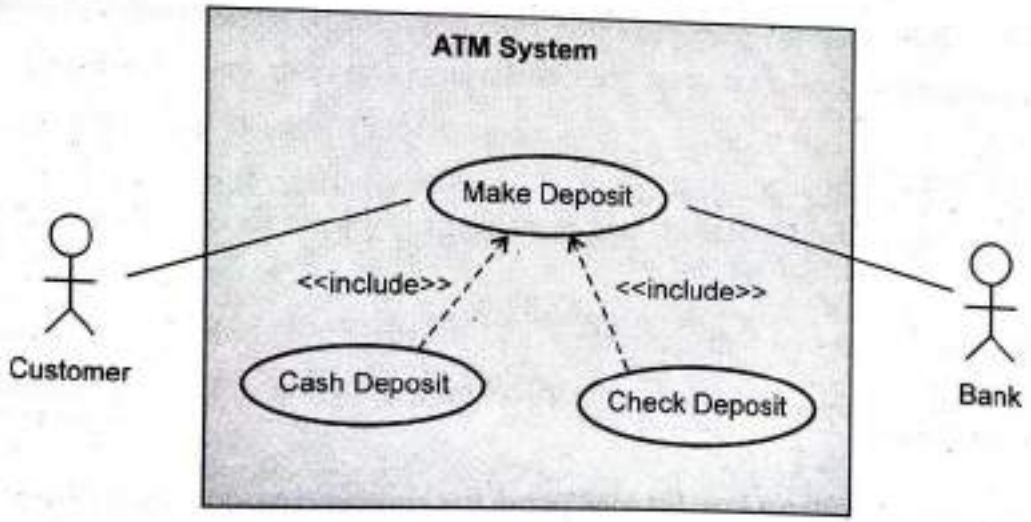
Example 2.13.1 *Consider the following use cases that play a role in a banking system :*

i) Deposit

ii) Withdraw (Minimum balance has to be checked)

Model sequence diagram for the above two cases **AU : Dec.-17, Marks 13**

Solution : i) Deposit



ii) Withdraw - Refer Fig. 2.13.6.

Sequence diagrams

i) Deposit - Refer Fig. 2.13.8.

ii) Withdraw - Refer Fig. 2.13.7.

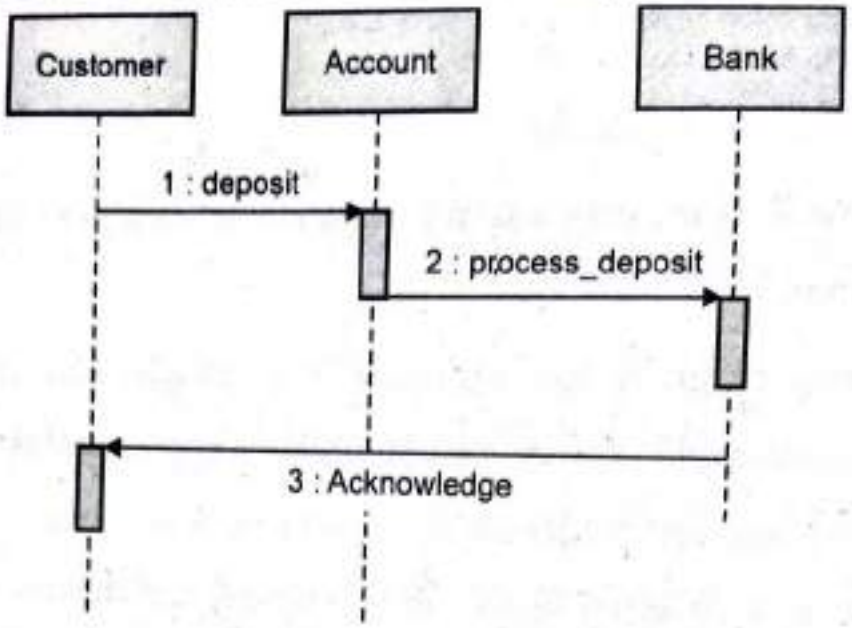


Fig. 2.13.8 Sequence diagram (Deposit)

2.13.3 Collaboration Diagram

- **Definition :** A collaboration diagram shows the objects and relationships involved in an interaction and the sequence of messages exchanged among the objects during the interaction.
- Collaboration diagram represent the structural organization of object.
- Both the sequence and collaboration diagrams are the **interaction diagram.**
- The sequence and collaboration are similar and can be transformed into each other.

2.13.3.1 Objects and Links

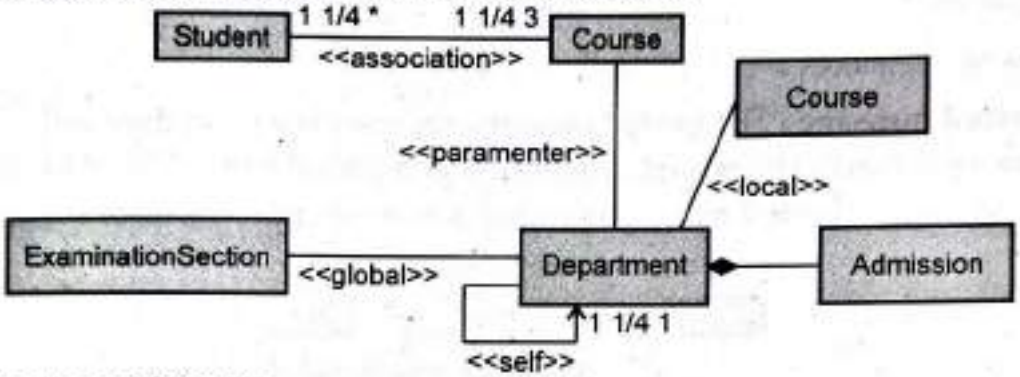
- For representing the structural organization of objects the collaboration diagram is used.
- The **communicating objects** are placed as vertices in graph. The **links** that connect these vertices are the arcs of this graph. These links have the role name to identify them.
- The links can be adorned using **messages** that are sent and received by the objects.
- This gives the **flow of control** in when the objects collaborate with each other.

A link specifies a path along which the message is sent to another object. The end of the link can be adorned by using following constraints : -

Adornment for end of links	Meaning
<<association>>	It specifies that corresponding object can be visible by the associating object.

<<global>>	It specifies that the corresponding object is visible in its enclosing scope.
<<local>>	It specifies that the corresponding object is visible in its as it has a local scope for invoking object.
<<self>>	It specifies that the object is visible due to its operations.
<<parameter>>	It specifies that the corresponding object is visible because it is parameter.

Following figure shows these adornments to the links

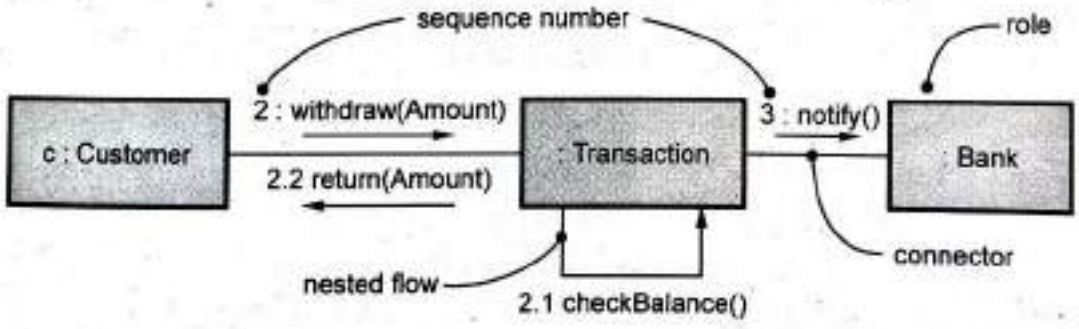


2.13.3.2 Messages and Stimulus

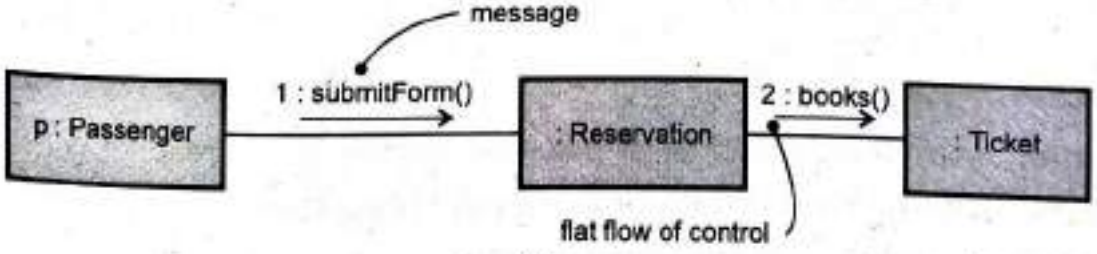
- There is a distinct flow of control within the system. For each control flow, the messages are ordered in some sequence. The sequence can be specified by following format

  Sequence_number : message

- Following is a communication diagram that shows the messages.



(a)



(b)

Fig. 2.13.9 Flat sequence

- There are two ways to specify the sequence of messages -
  - **Procedural sequence :** The procedural or nested flow of control is represented by a filled solid arrowhead. The nesting of some messages is represented by the sub-numbering.
  - **Flat sequence :** For representing the non procedural sequence, the flat sequence is used. It is represented by a stick arrowhead.
- The message sequence number is useful for distinguishing the flow of control.
- A stimulus represents a specific instance of sending the message, with particular arguments.
- **Types of messages :**
  - **Nested message :** The nested message represents a procedure call or other nested flow of control. The nested sequence is completed before the outer level sequence resumes. The nested message symbol is represented by a filled solid arrowhead.

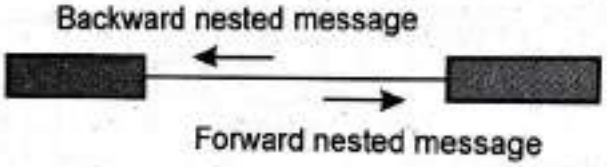


Fig. 2.13.10

  - **Flat message :** The flat message shows the progression to the next step in a sequence. The flat message symbol is represented by a stick arrowhead.

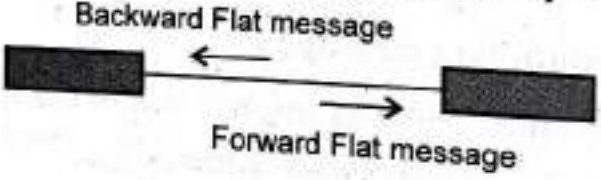


Fig. 2.13.11

  - **Asynchronous message :** The asynchronous message symbol shows an asynchronous message between two objects. The asynchronous message symbol is represented by a half stick arrowhead.

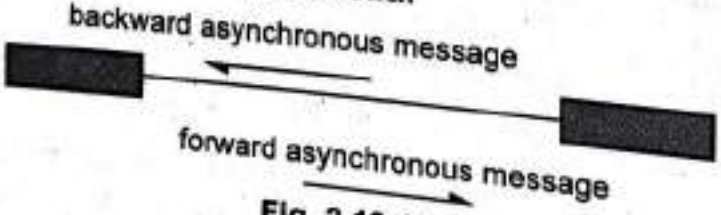


Fig. 2.13.12

2.13.3.3 Active Objects

Definition : An active object is an instance that owns a thread of control and can initiate control activity. For example, processes and tasks are active objects. Active objects can contain other symbols and links among them.

It can be represented as follows -

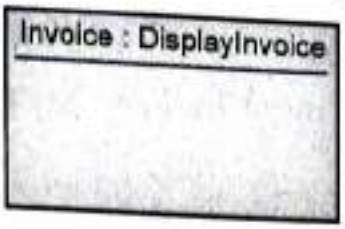


Fig. 2.13.13

The syntax of denoting the active object is

[name][:type]

where

name is the name of active object

type is a class or use case

2.13.3.4 Communication Diagram

There are two types of interaction diagrams - Sequence diagram and communication diagram. Both the diagrams are semantically equivalent. Hence one form of the interaction diagram can be converted to other without losing any information. But there are ways by which these two diagrams differ. For example in above given communication diagram, the two objects can be linked to other objects using {local} or {global} adornment but this is not possible in sequence diagram. Similarly, in sequence diagram at the end of the transaction you can destroy the object but this facility is not there in communication diagram.

Sequence diagram	Communication diagram
The sequence diagram models the lifeline of the objects. The lifeline denotes the existence of the object between its creation and destruction.	The communication diagram represents structural links among the objects during there interactions.
The particular object can be destroyed explicitly in sequence diagram.	There is no facility to destroy some object.
The linking of the objects with other object can not be represented as local or global.	The {local} or {global} scope of the objects with other objects can be represented in this diagram.

2.13.4 Strengths and Weakness of Sequence and Communication Diagrams

Different developers have different preferences for the interaction diagram. However the sequence diagram is preferred more because they posses greater notational power.

Strengths of sequence diagram

1. The call flow can be represented more systematically in sequence diagram than that of communication diagram.
2. The sequence diagram can be read from top to bottom and proved to be excellent for the documentation.
3. The reverse engineering can be easily carried out in sequence diagram.

Weakness of sequence diagram

1. Drawing sequence diagram is complicated.
2. It is more space consuming because if we wish to insert any new object it has to be inserted to the right.

Strength of communication diagram

1. The communication diagram can be easily drawn.
2. They are more space efficient because the boxes can be drawn vertically or horizontally and can be placed anywhere. Sketching of communication diagram is very simple.

Weakness of communication diagram

1. It is difficult to see the sequence of messages in communication diagram.
2. There are very few notational options in communication diagram.

2.13.5 Examples on Interaction Diagram

Example 2.13.2 *Draw a sequence diagram to show how a GUI interacts with other objects and also draw corresponding collaboration diagram.*

Solution :

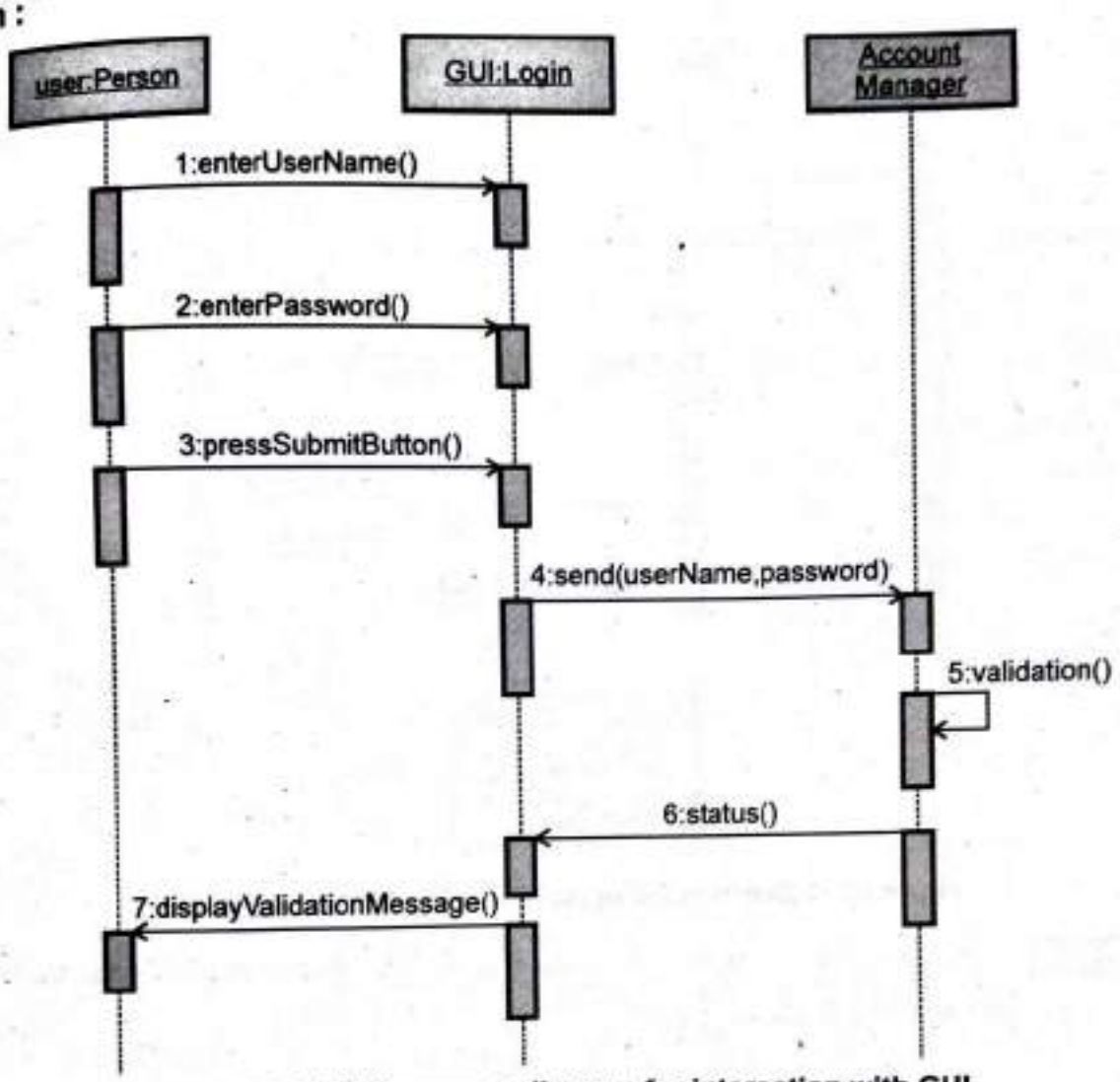


Fig. 2.13.14 (a) Sequence diagram for interaction with GUI

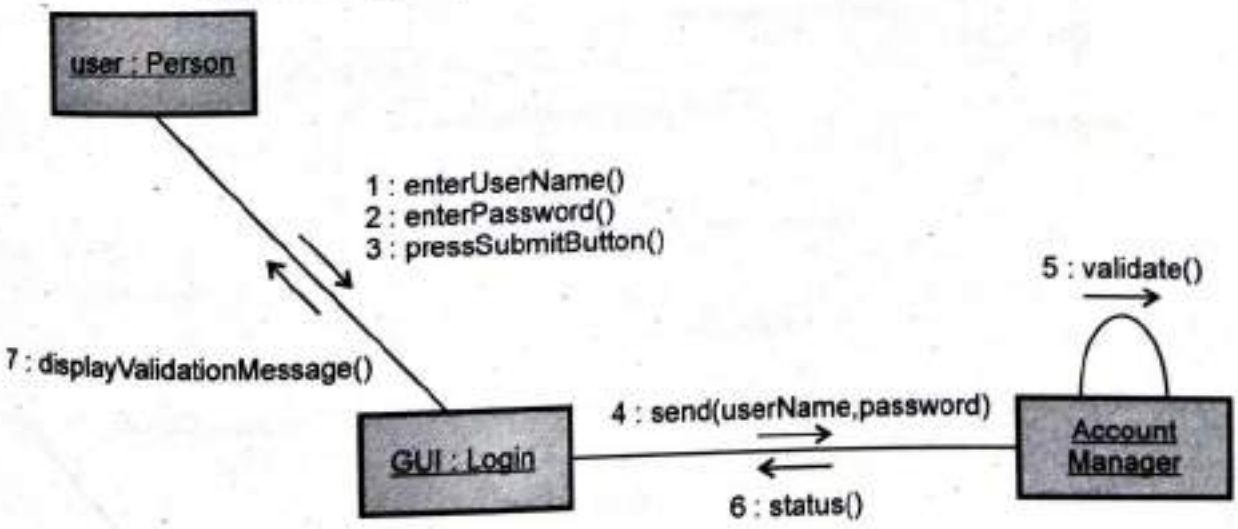


Fig. 2.13.14 (b) Communication diagram for interaction with GUI

Example 2.13.3 *Draw a sequence diagrams that specifies the flow of control involved in initiating a simple, two party phone call.*

Solution :

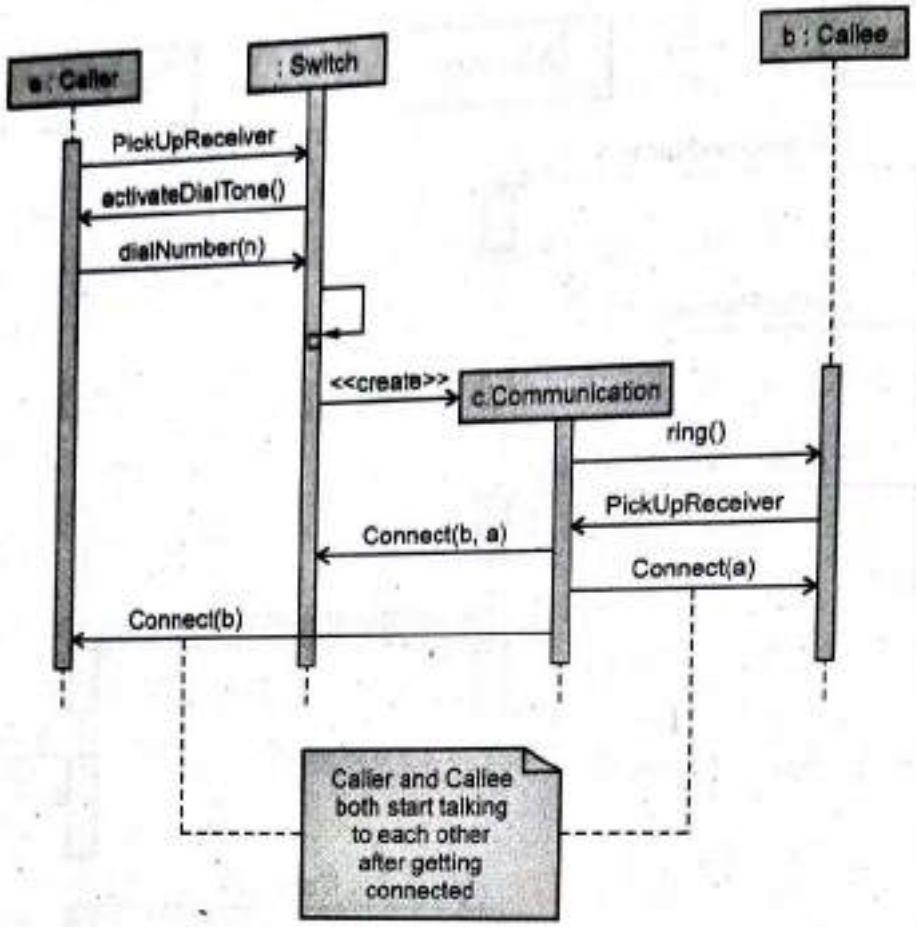


Fig. 2.13.15 Sequence diagram for two party phone call

Example 2.13.4 *Draw a collaboration diagram that specifies the flow of control involved in registering a new student at a school.*

Solution :

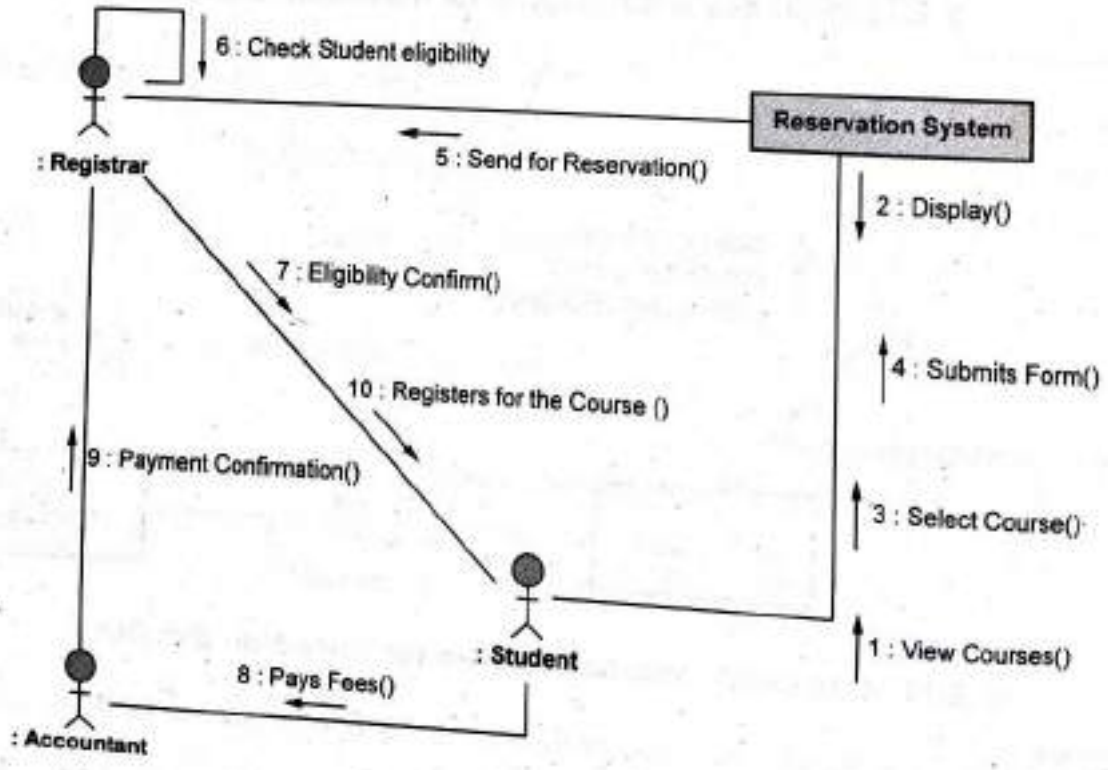


Fig. 2.13.16

Example 2.13.5 *Draw sequence diagram for login operation in ATM model.*

Solution : The login operation in ATM model can be performed by entering the valid ATM card and PIN. The sequence diagram for it is given below -

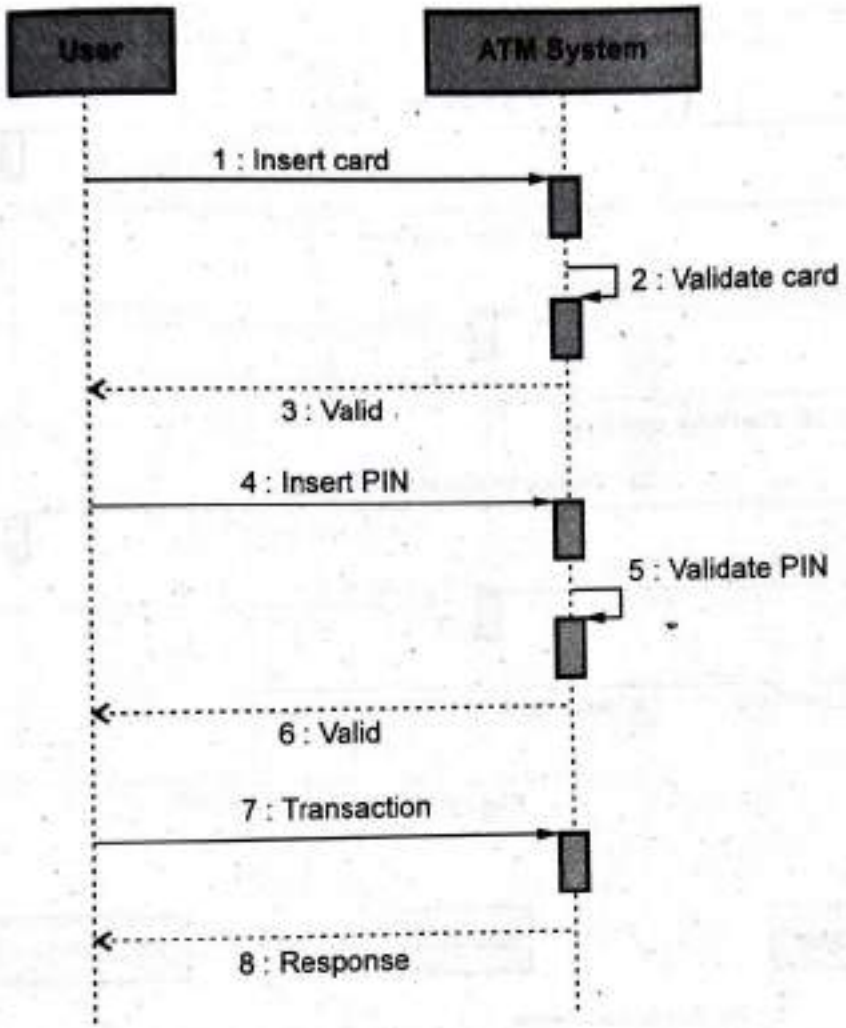


Fig. 2.13.17

Example 2.13.6 *Explain the two interaction diagrams for "issue of a book" and "renewal of a book" operations in a library.*

Solution : i) Sequence diagram for issue of a book

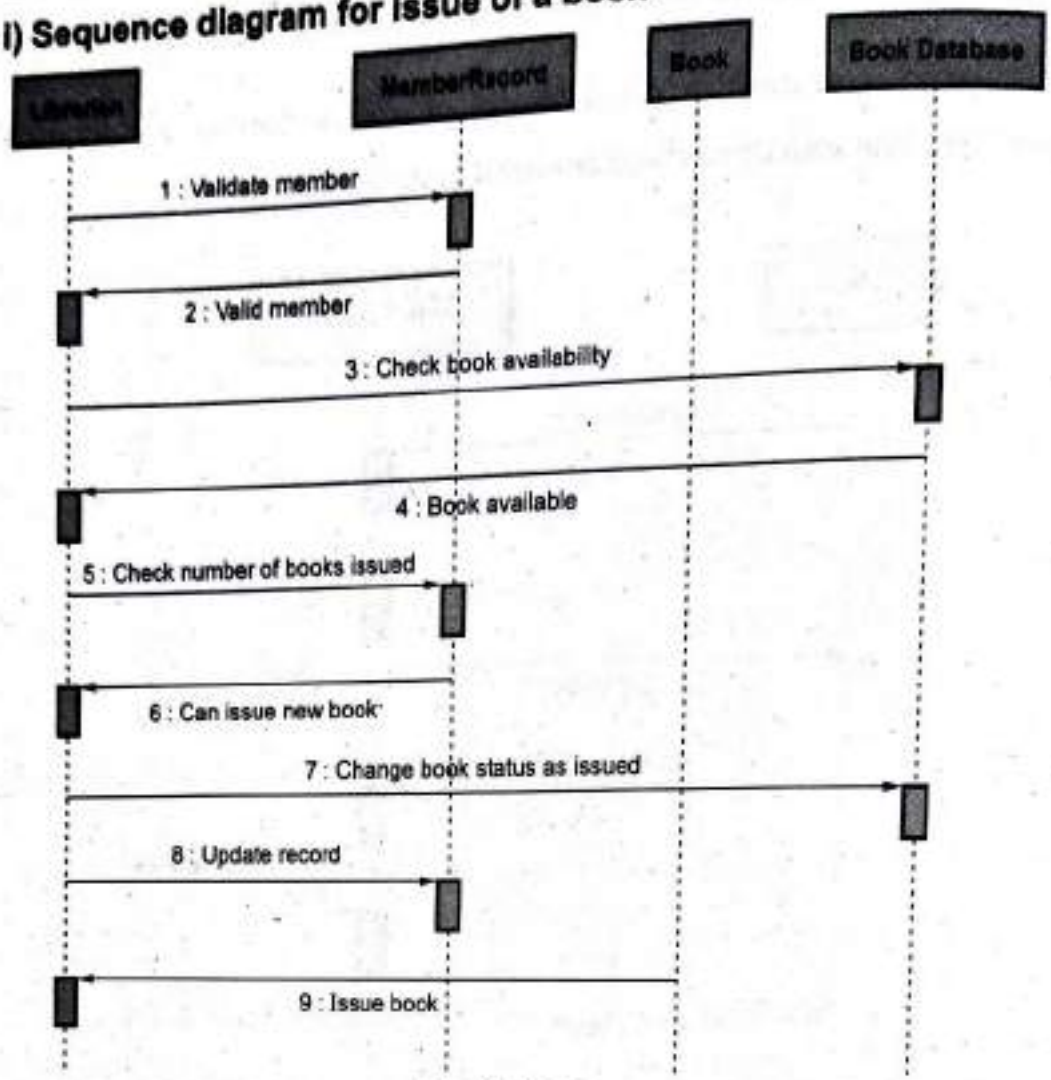


Fig. 2.13.18

ii) Renewal of a book

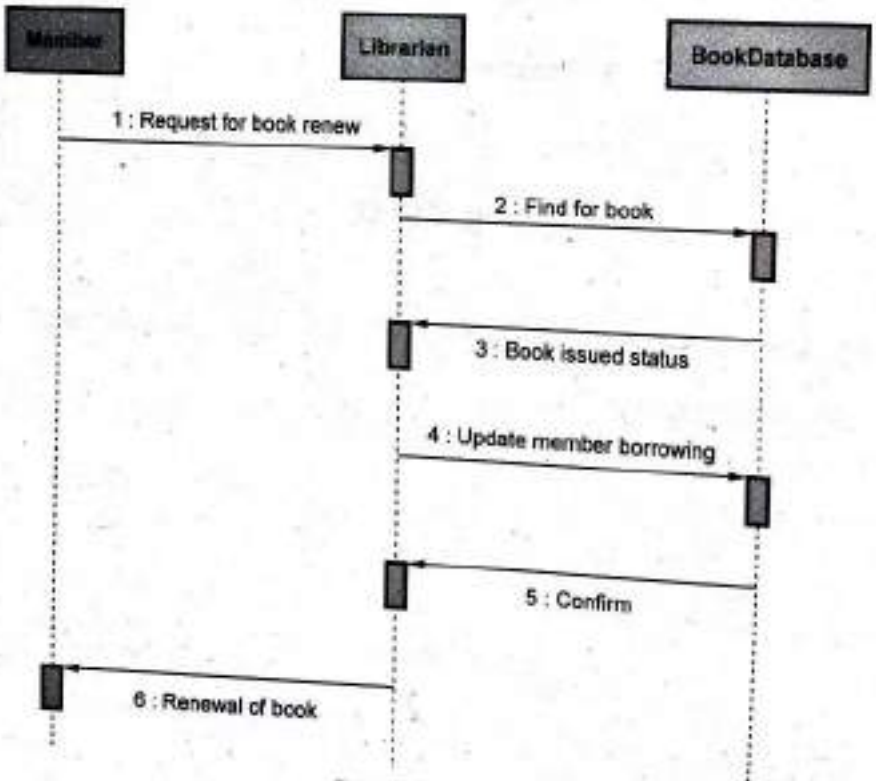


Fig. 2.13.19

Example 2.13.7 *Draw sequence and collaboration diagram for online course reservation system.*

Solution : i) Sequence diagram

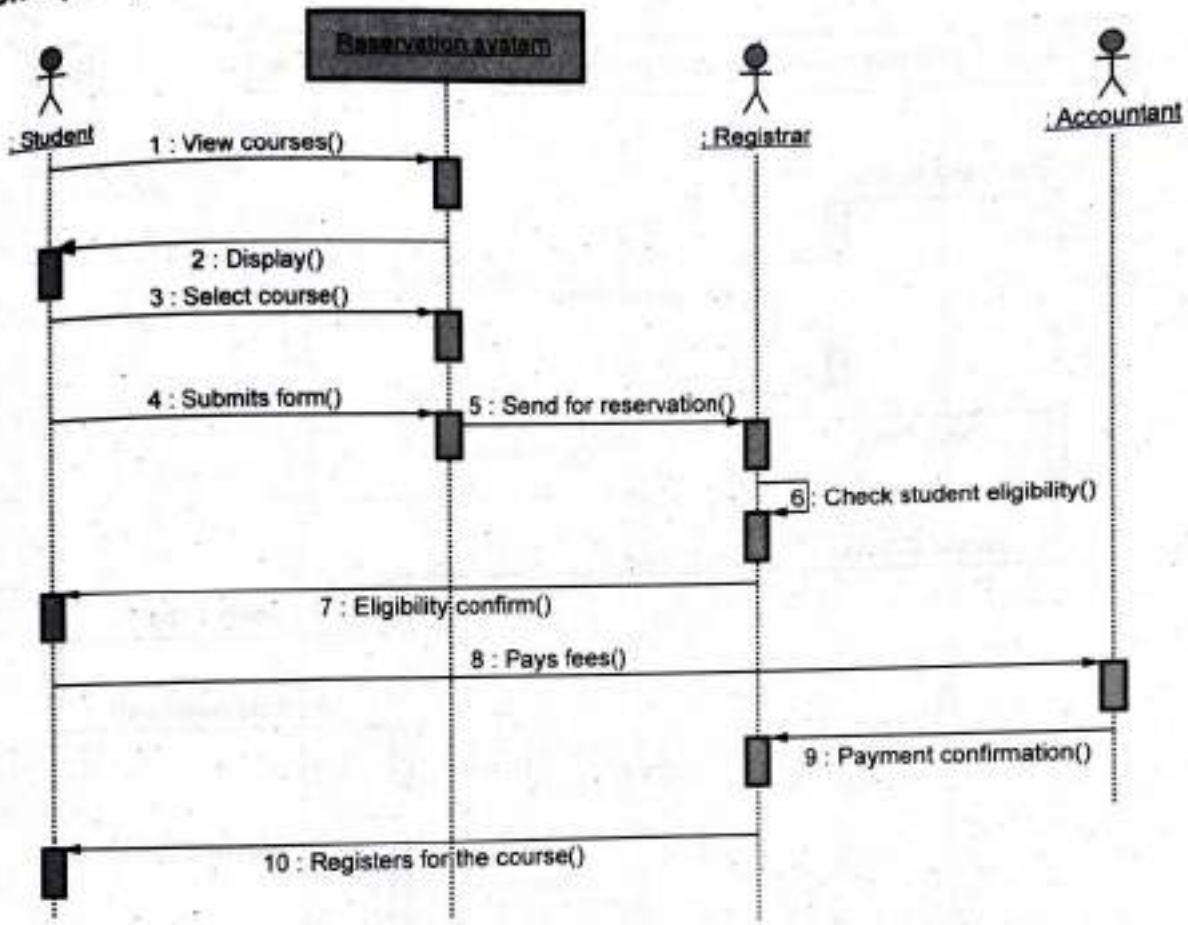


Fig. 2.13.20

Collaboration diagram

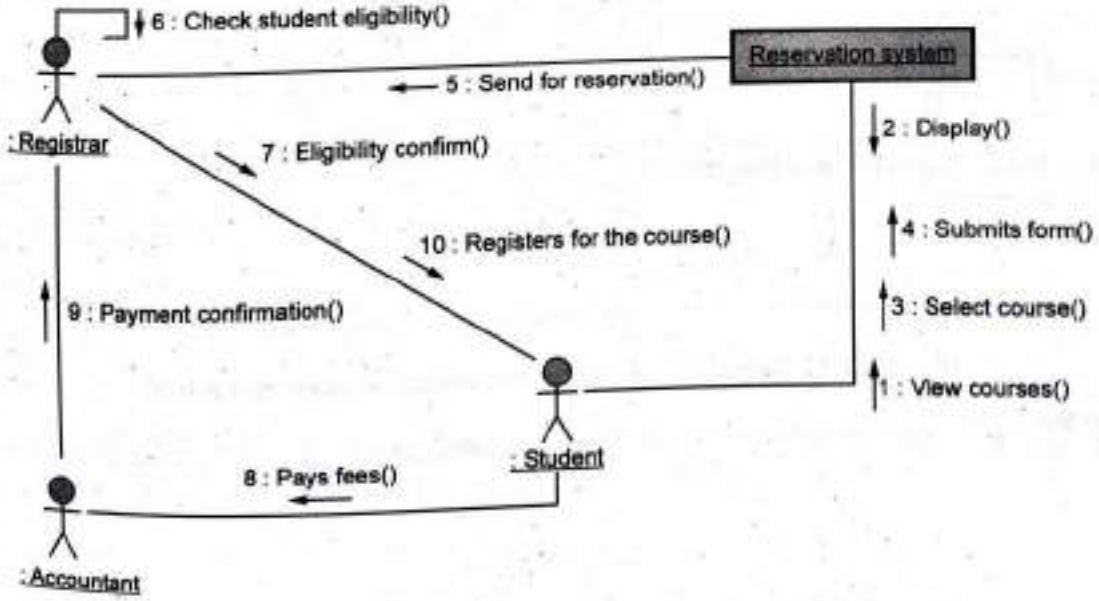


Fig. 2.13.21

Example 2.13.8 *Draw a sequence diagram for online shopping system.*

Solution :

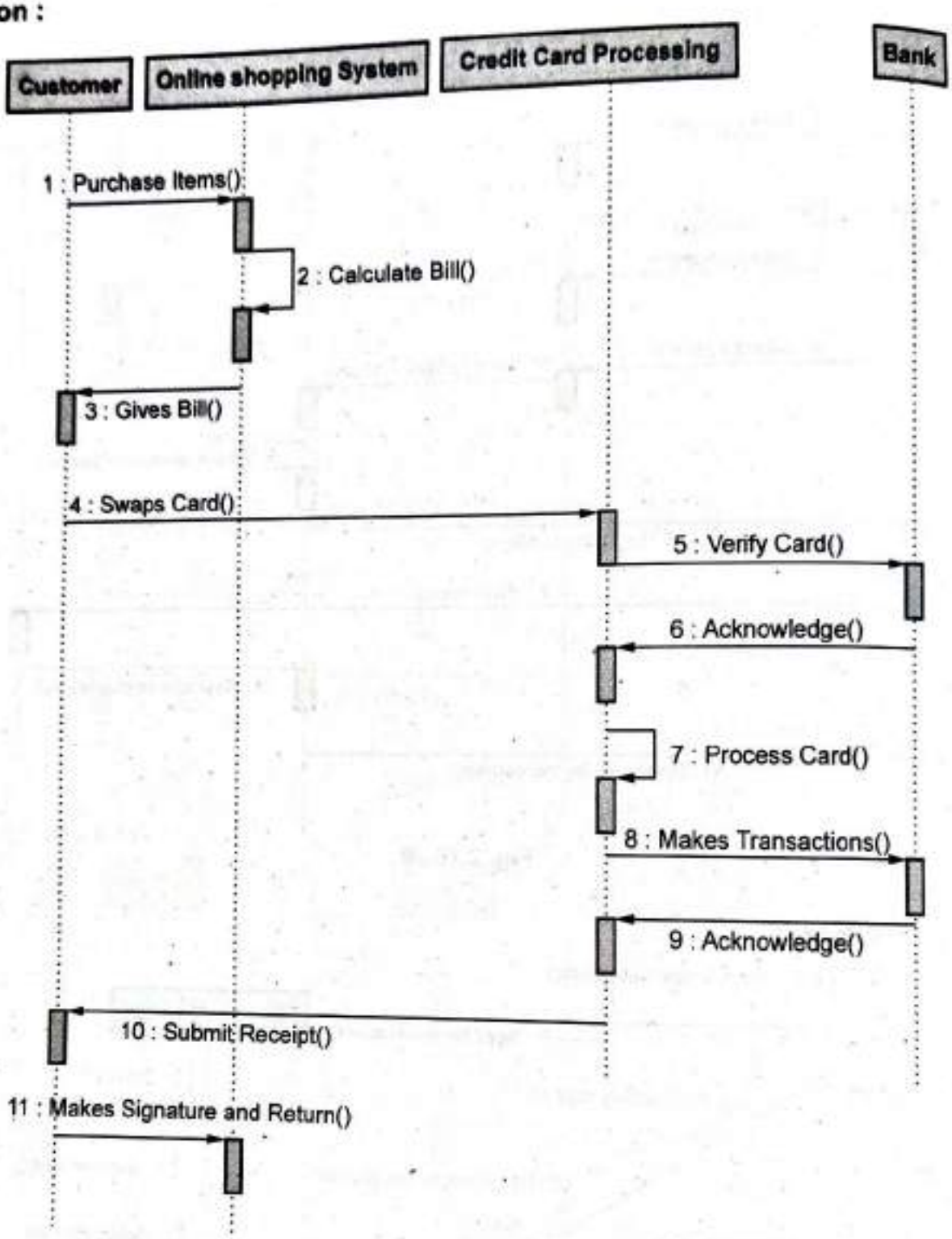


Fig. 2.13.22 Sequence diagram for online shopping system

Example 2.13.9 *Explain about interaction diagram notation for inventory management system.*

Solution :

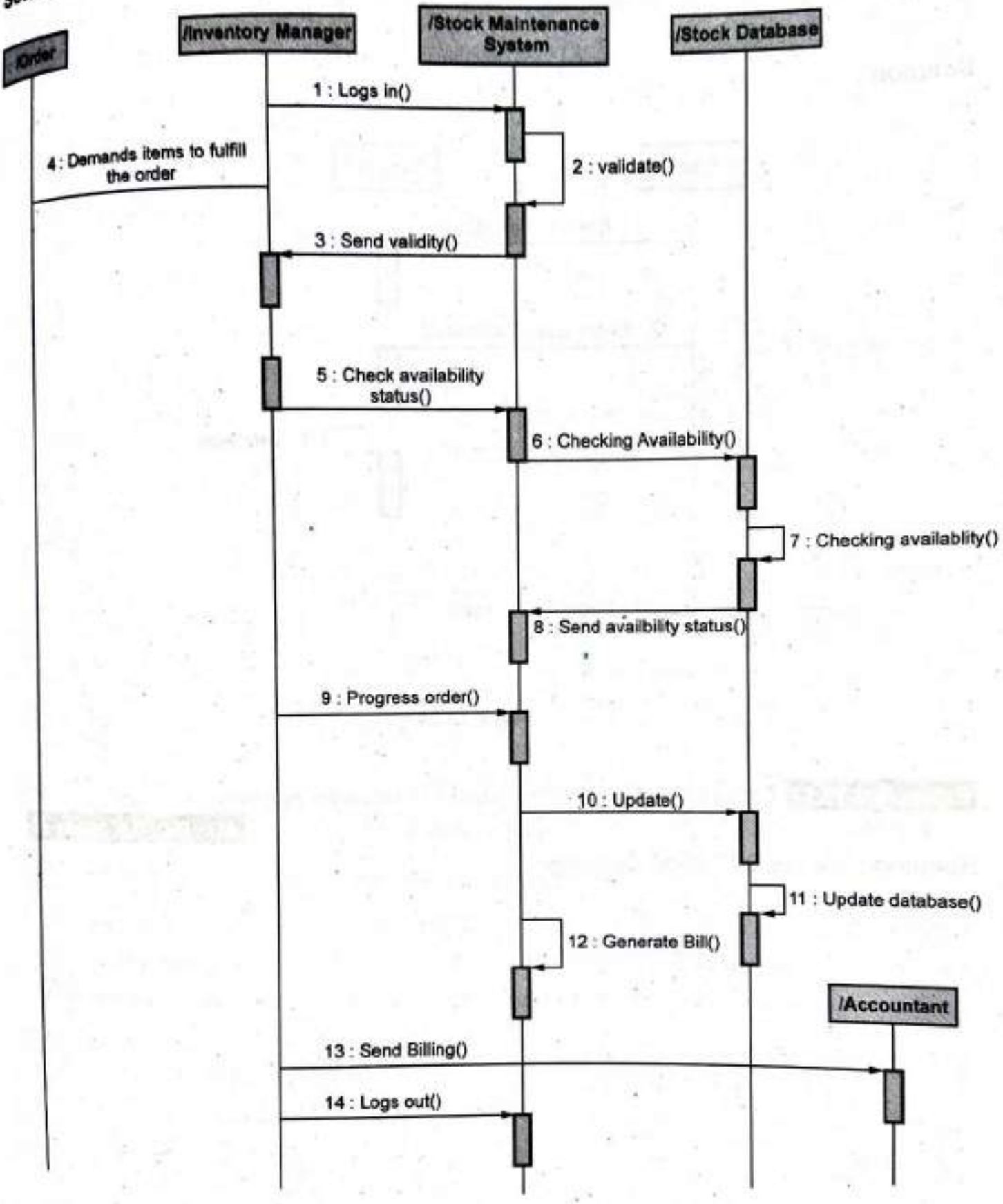


Fig. 2.13.23

Example 2.13.10 *The use case 'Login' accepts userid and password from the user. The system validates them and displays a message 'Log in is successful'. Draw a sequence diagram for this.*

Solution :

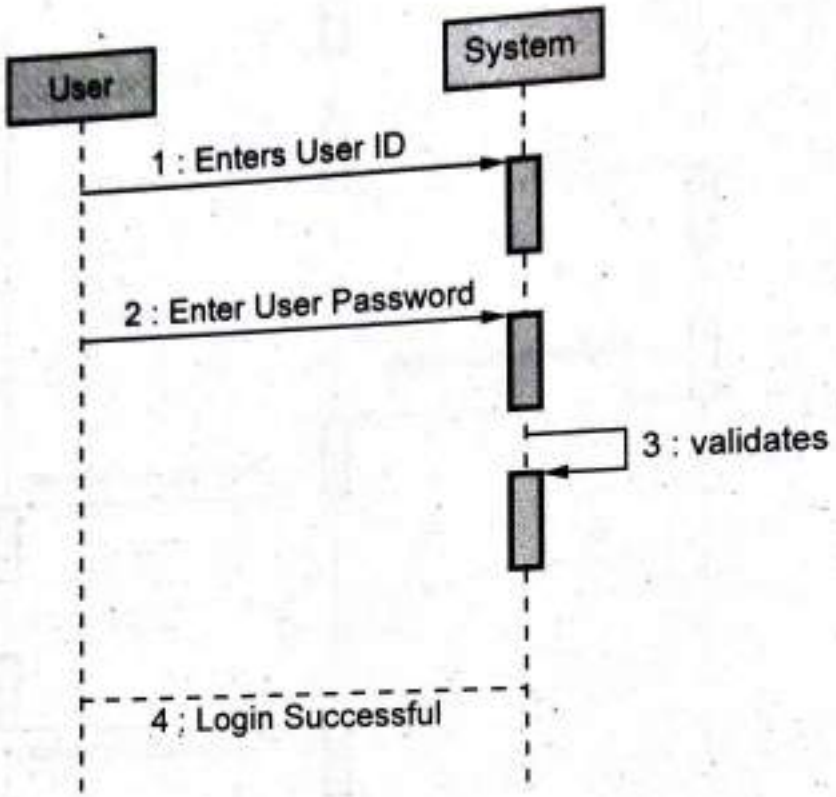


Fig. 2.13.24

Example 2.13.11 *Explain about Interaction diagram notation for inventory management system.* **AU : Dec.-15, Marks 16**

Solution : See Fig. 2.13.25 on next page

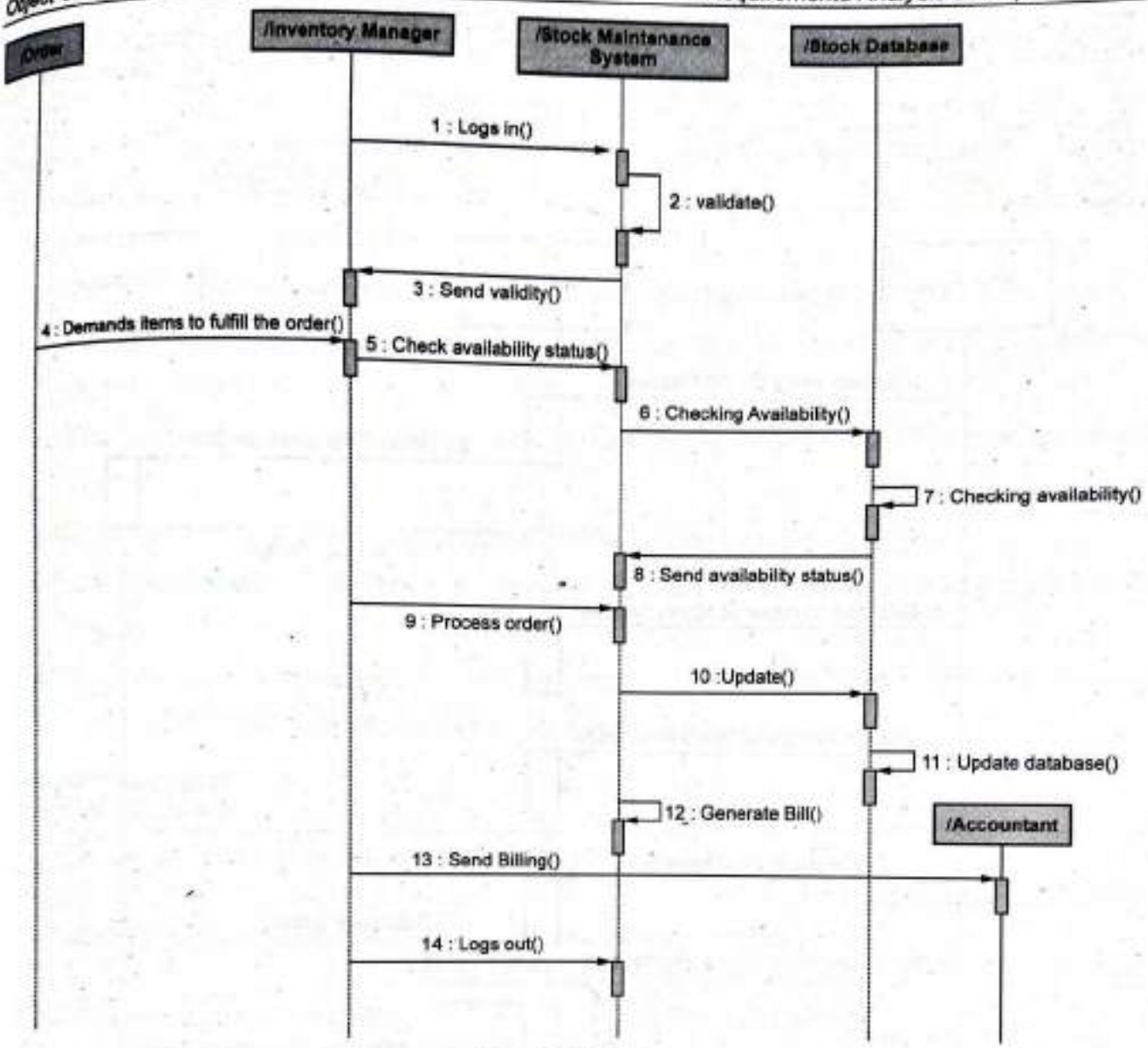


Fig. 2.13.25

Example 2.13.12 *Apply interactive modeling for a payroll system in UML.*

AU : Dec.-16, Marks 16

Solution : The interaction modeling can be done by designing the sequence diagram for the given problem statement. The sequence diagram for payroll system is as shown below

Sr. No.	Sequence diagram	Collaboration diagram
1.	The sequence diagram models the **lifeline of the objects**. The lifeline denotes the existence of the object between its creation and destruction	The collaboration diagram represents **structural links** among the objects during there interactions.
2.	The particular object can be destroyed explicitly in sequence diagram.	There is no facility to destroy some object.

3.	The linking of the objects with other object can not be represented as local or global.	The {local} or {global} scope of the objects with other objects can be represented in this diagram.
4.	Designing sequence diagram is very difficult.	It is difficult to see the sequence of message in collaboration diagram.

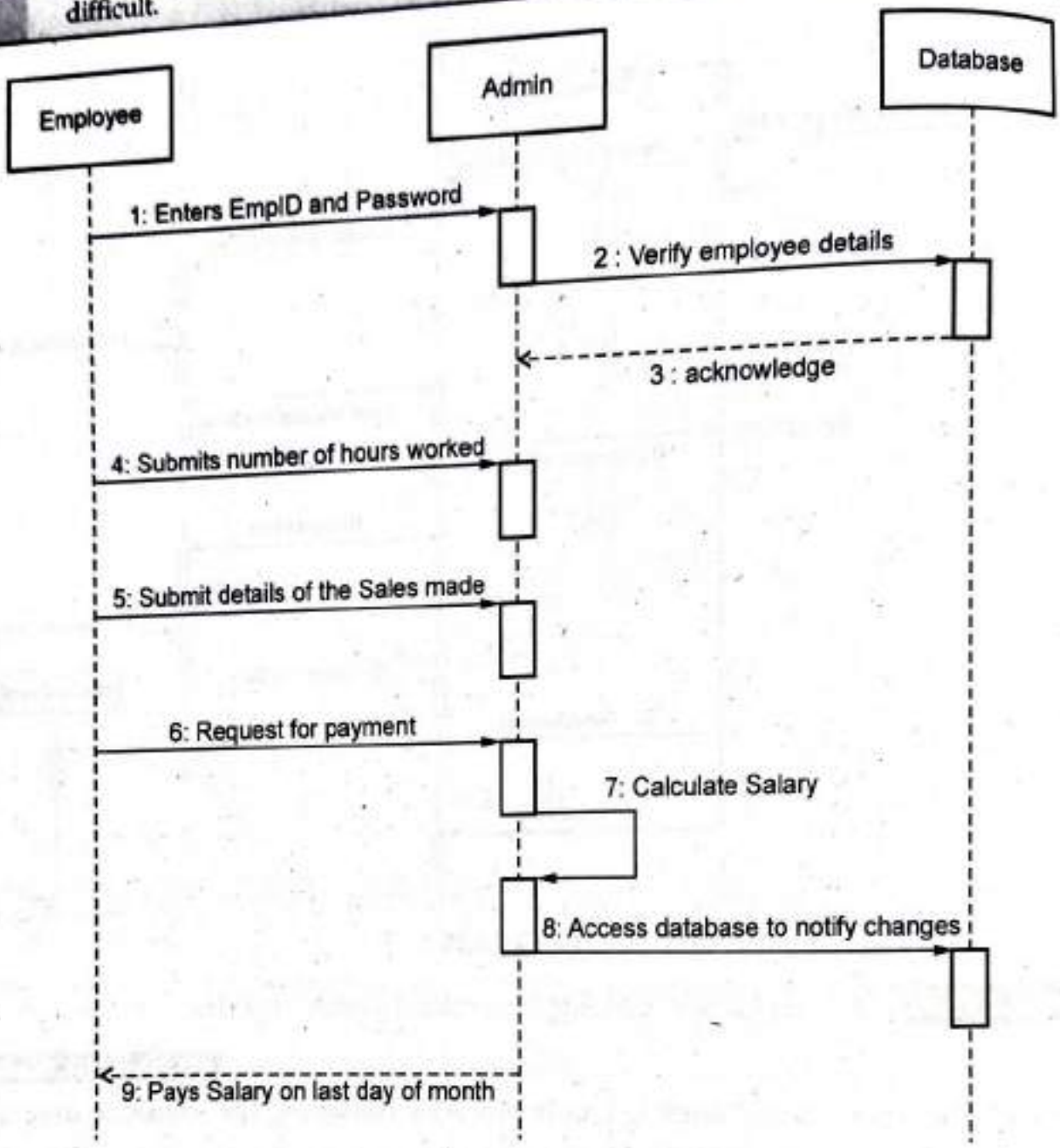


Fig. 2.13.26

2.13.6 When to use Sequence Diagram ?

The sequence diagram represents following things -

- One particular scenario of a use case
- The events that are generated by the external actors
- Sequence of those events
- Events among the interacting objects of the system.

2.13.7 When to use Communication Diagram ?

The interaction diagrams such as sequence diagram or communication diagrams are used to represent the dynamic aspect of the system. Following are the two ways by which the communication diagram is used -

1. To model flows of control by time ordering

- The communication diagram is used to model the flow of control by time ordering.
- The sequence of messages that take part in the interaction over the time is represented in the communication diagram.
- The iterations and branching are shown more effectively in communication diagram.

2. To model flows of control by organization

- The communication diagram us used to model flow of control by organization of objects.
- This diagram emphasizes on the structural relationship. The interaction among various instances (objects) is shown by the messages.

Review Questions

1. *Discuss the relationship between sequence diagram and class diagram.* **AU : Dec.-18, Marks 13**
2. *Illustrate with an example, the relationship between sequence diagram and use cases.* **AU : May-11,14, Dec.-13,14, Marks 16, May-18,19, Marks 13**
3. *What are system sequence diagrams ? What is the relationship between SSDs and Use cases ? Explain with an example.* **AU : Dec.-16, Marks 16**
4. *Mention the strengths and weakness of sequence diagram vs. communication diagram.*
5. *What is the use of system sequence diagram ?* **AU : May-11, Marks 2**

2.14 Activity Diagrams

AU : Dec.-13,18, Marks 16

- An activity diagram, is just similar to the flowchart in which flow of control from activity to activity is shown. The difference between traditional flow chart and activity diagram is that - in an activity diagram, the concurrency as well as branches of control is shown.
- Activity diagram focuses on dynamic aspect of the system. An activity is an ongoing structured execution of behavior.

2.14.1 Activity and Actions

Actions

- The executable and atomic computations are called as actions. For example - evaluation of some expression, sending of signal to some object, creation or destruction of some object - all these represent the actions.
- There are two ways of representing an action - 1) Simple action and 2) expression.
- The actions are represented by rounded box.

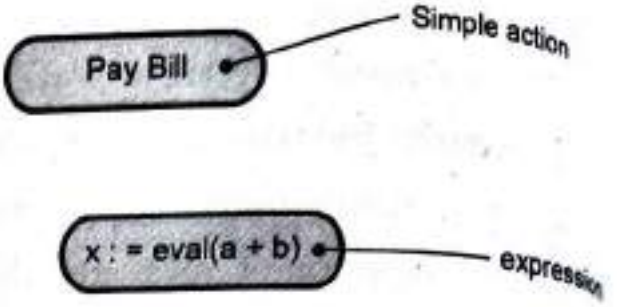


Fig. 2.14.1 Actions

Activity node

- Activity node is a nested grouping of actions or other nested activity node.
- Activity nodes are created for those actions that take time to complete.
- Actions is an activity node that cannot be further decomposed.
- It is represented as

Fig. 2.14.2 Activity nodes

2.14.2 Initial and Final Activity

- **Initial state** : In activity diagram, the initial state is represented by a solid circle. From this circle the transition to the first activity starts. Thus the initial state represents that all the activities start their execution from this point.
- **Termination** : In activity diagram, the termination is represented by **bull's eye** i.e. a solid circle surrounded by a hollow circle. This state represents the termination of all the activities at this point. Fig. 2.14.3 represents these states.

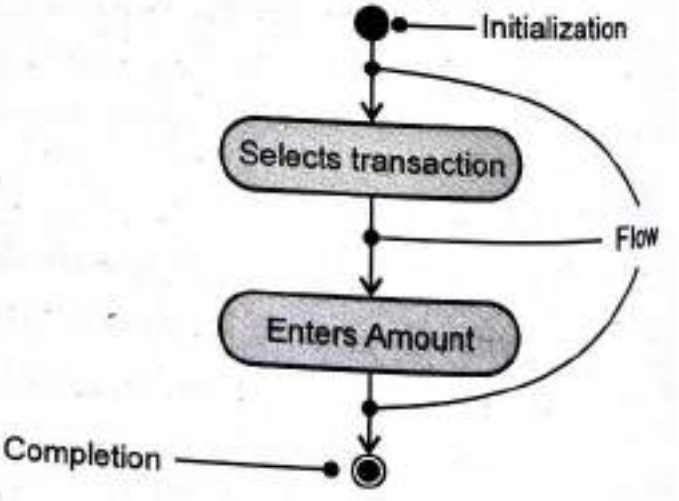


Fig. 2.14.3 Initiation and termination

2.14.3 Activity Edge

- In order to show the flow through an activity the actions are linked up using **activity class.**
- Activity edge can be **control edge** or **data flow edge** (object flow edge).
- Thus we can show activity edge as a line with an arrow pointing to the next action. For example :

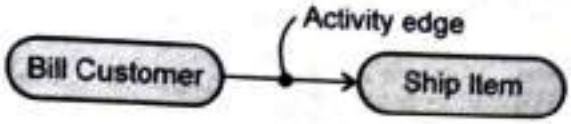


Fig. 2.14.4 Activity edge

2.14.3.1 Object Flows

When activity edge is represented for data only. For example,

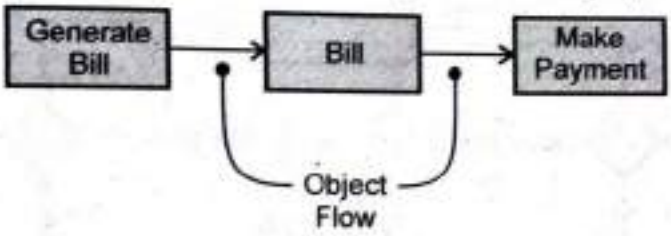


Fig. 2.14.5

2.14.4 Decision and Merge Points

A decision node or decision point is a control node which has one input edge and multiple outgoing edges.

When data arrives at the decision point, a single outgoing edge is selected from multiple outgoing edges and data is sent over that outgoing edge. The outgoing edge is selected based on the evaluation of guard condition. Guard condition is a boolean expression that tests some value visible to the activity.

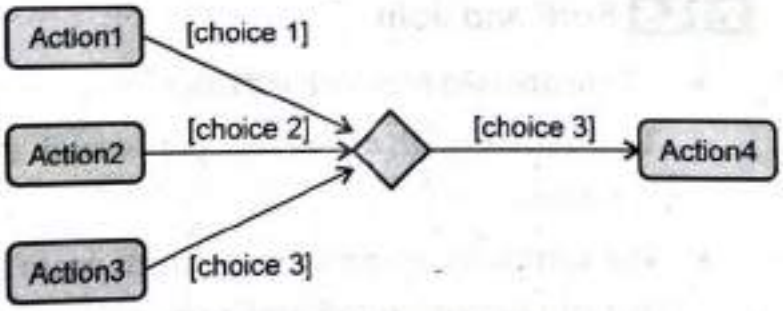


Fig. 2.14.6 Merge point

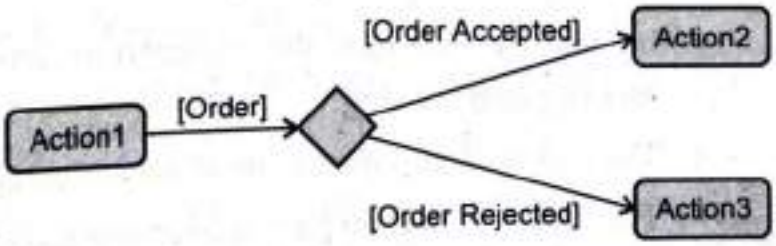


Fig. 2.14.7 Decision point

The merge node is opposite of decision node. It has multiple input flows and single output flow. The merge point simply takes any token along some incoming input flow and makes it available on the outgoing edge.

The notation for decision and merge point is diamond shaped.

Example

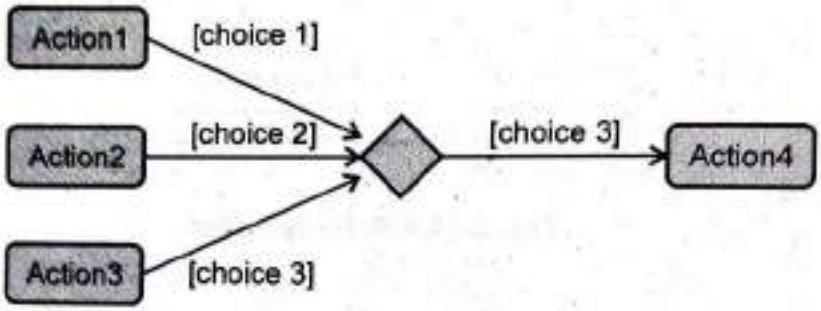


Fig. 2.14.8 Merge point

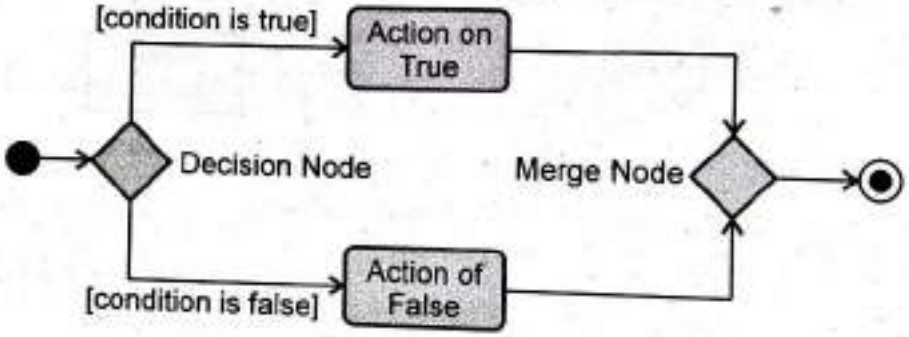


Fig. 2.14.9 Decision and merge point together

2.14.5 Fork and Join

- There are two types of activities - sequential activities and concurrent activities.
- The activities that can be performed one after the other are called sequential activities.
- The computer system can perform more than one activities at a time. The activities that can be performed at the same time are called concurrent activities.
- The concurrent activities can split in several activities or can join and merge in some action.
- In activity diagram the concurrent activities are represented with the help of **forking and joining**.
- The fork and join in activity diagram is represented by **synchronization bars**.
- The synchronization bars are represented by thick horizontal or vertical bars.
- The **fork** represents the separation of two control flows whereas **join** represents joining of two control flows.

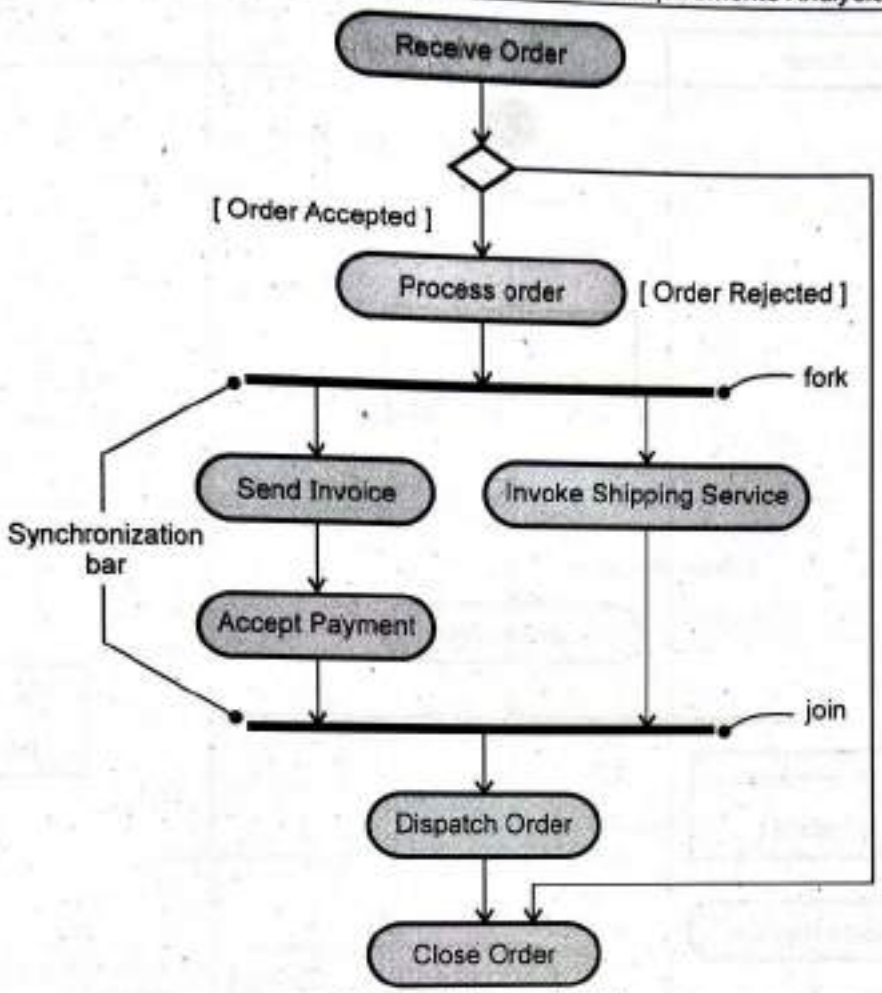


Fig. 2.14.10 Forking and joining

2.14.6 Activity Partitions or Swimlane

- Many times the activity diagram needs to partition in groups to represent the flow of business processes.
- Each group represents the particular business flow belonging to some category. Such a group is called **swimlane.**
- Graphically the swimlane is denoted by a vertical solid line.

2.14.7 Object Flow

- The objects may be involved in the activity diagram.
- These objects are associated with the control flow of the activity diagram. This is called **object flow.**
- The state of the object is specified for each of these objects.
- Following is a simple activity diagram in which the object flow is represented for the objects of the classes **Order** and **Invoice.**

(See Fig. 2.14.11 on page 2 - 120)

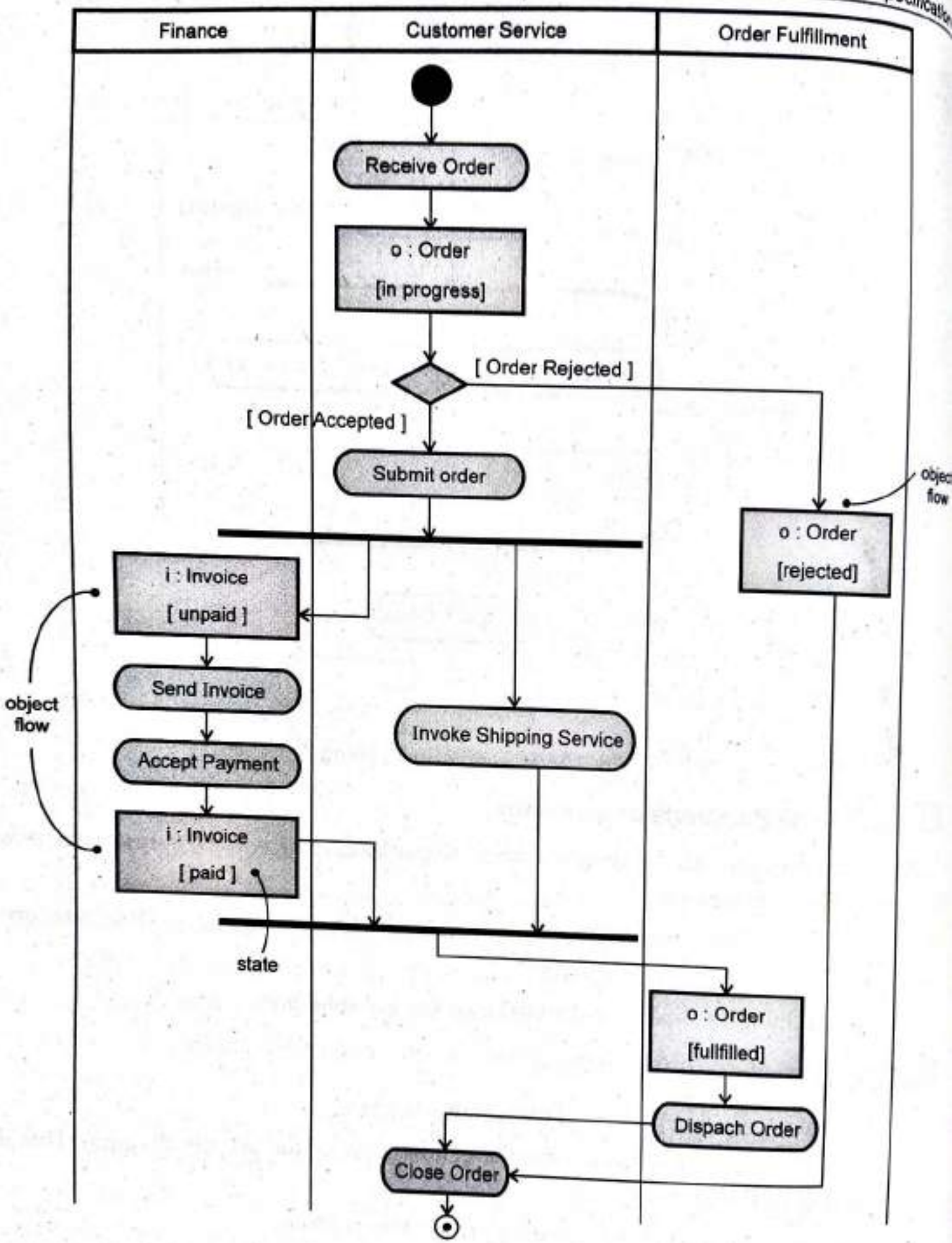


Fig. 2.14.11 Object flow

2.14.8 When to use Activity Diagrams?

- Activity diagrams are used to model the **dynamic aspects** of a system. That means the activity involved in classes, interface, components and nodes can be represented in the activity diagram.

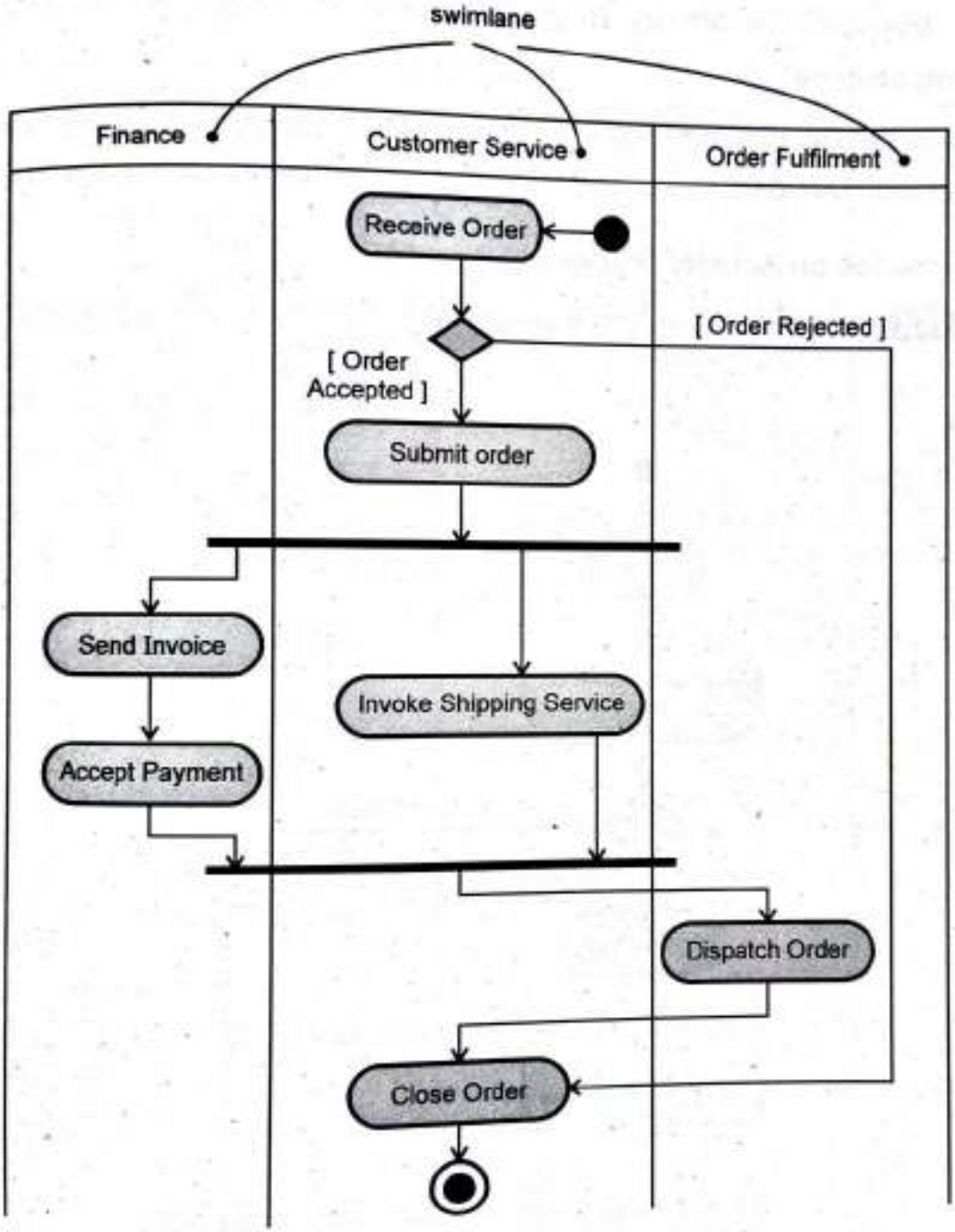


Fig. 2.14.12 Swimlane with an activity diagram

- The activity diagram can be used in two ways -

1. **To model the workflow**
   - The actors collaborate with the system and then various activities are performed.

- These activities depict the object flow.
- The activity diagrams can be used to model this work flow.

2. To model the operations

- In this case the activity diagram is used as a flowchart to model some computations.
- These activity diagrams include branch, fork and join. Such activity diagrams are normally useful for design of the operations having parameters and local objects.

2.14.9 Examples on Activity Diagram

Example 2.14.1 *Draw activity diagram for issue of book in library information system.*

Solution :

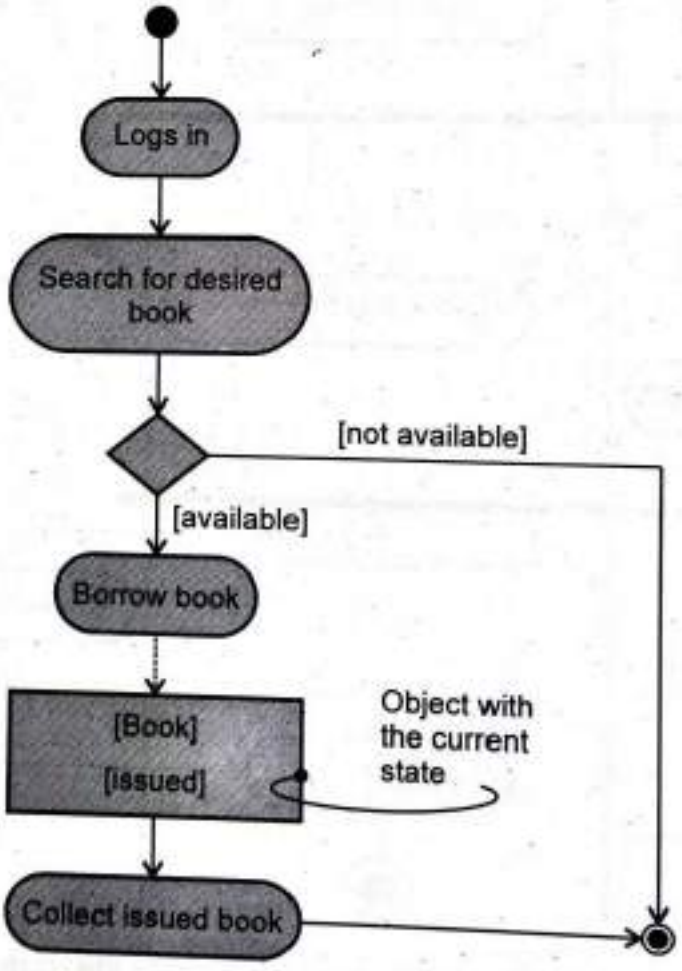


Fig. 2.14.13

Example 2.14.2 *Draw swimlane diagram for exam registration systems.*

Solution :

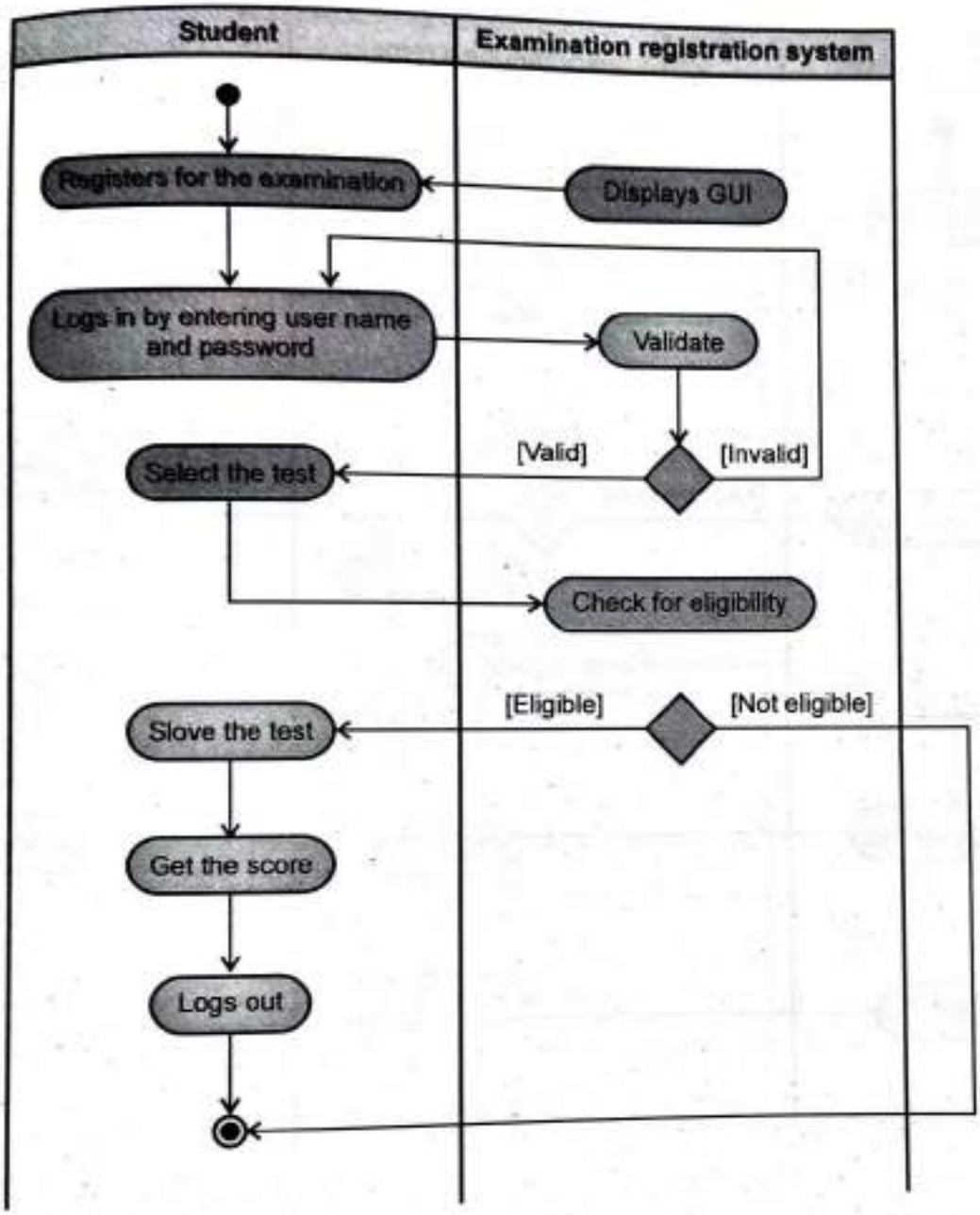


Fig. 2.14.14

Example 2.14.3 *Draw activity/swimlane diagram for stock maintenance systems.*

Solution :

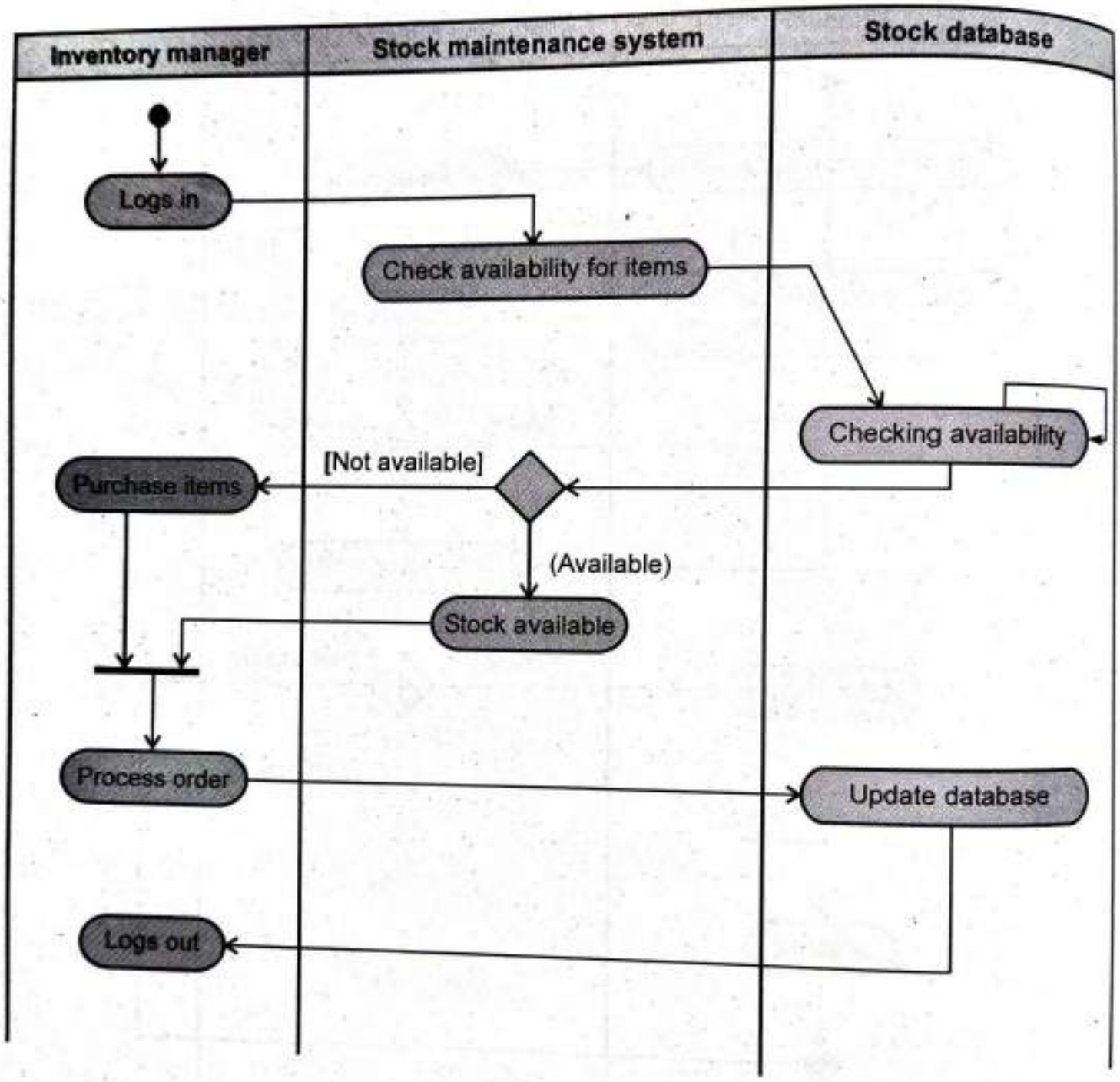


Fig. 2.14.15

Example 2.14.4 *Draw activity/swimlane diagram for credit card processing system.*

Solution :

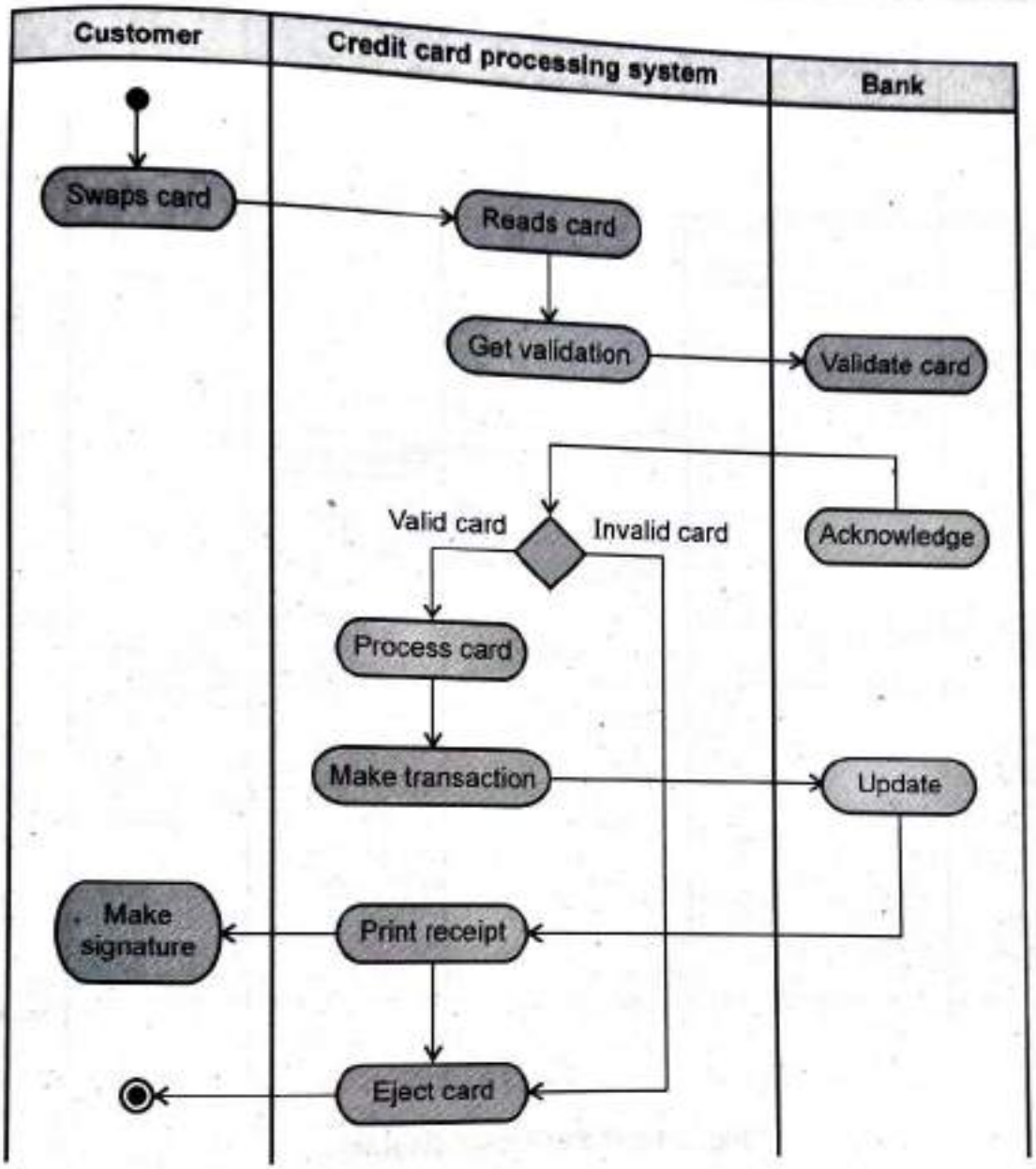


Fig. 2.14.16

Example 2.14.5 *A customer decides to upgrade her PC and purchase a DVD player. She begins by calling the sales department of the PC vendor and they tell her to talk to customer support. She then calls customer support and they put her on hold while talking to engineering. Finally the customer support tells the customer about several supported DVD options. The customer chooses a DVD and it is shipped by the mail department. The customer receives the DVD, installs it satisfactorily and then mails her payment to accounting.*

Construct an activity diagram for this process. Use swimlanes to show the various interactions.

Solution :

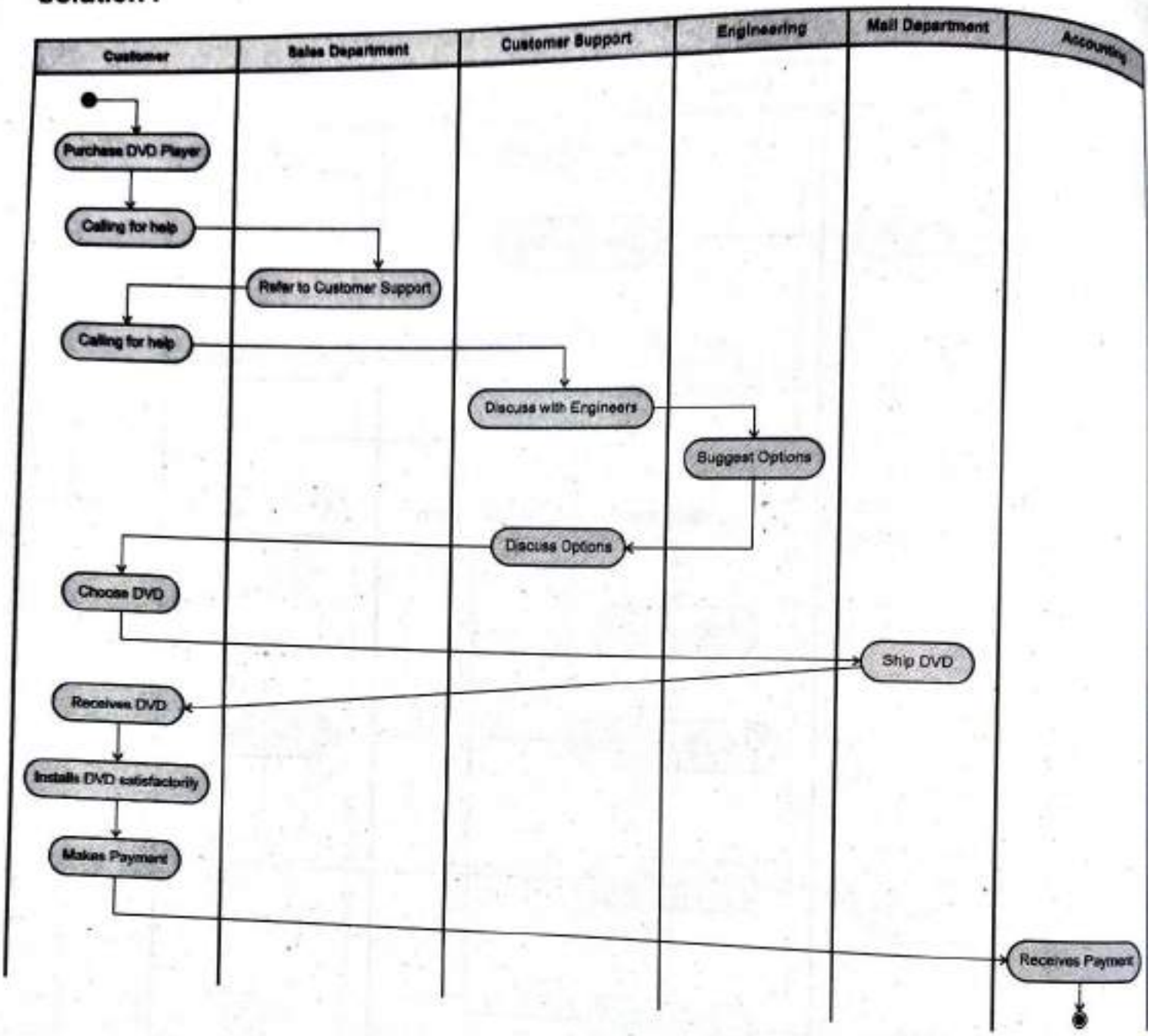


Fig. 2.14.17 Swimlane diagram

Example 2.14.6 *Prepare an activity diagram for awarding marks to regular students. If the student has attended 80 % classes, he is awarded minimum 5 marks. If the student has attended more than 80 % classes, he is awarded minimum 10 marks. The students who have completed assignments are given 10 marks. Those who have completed 50 % are given 5 marks and rests are given 0 marks.*

Solution :

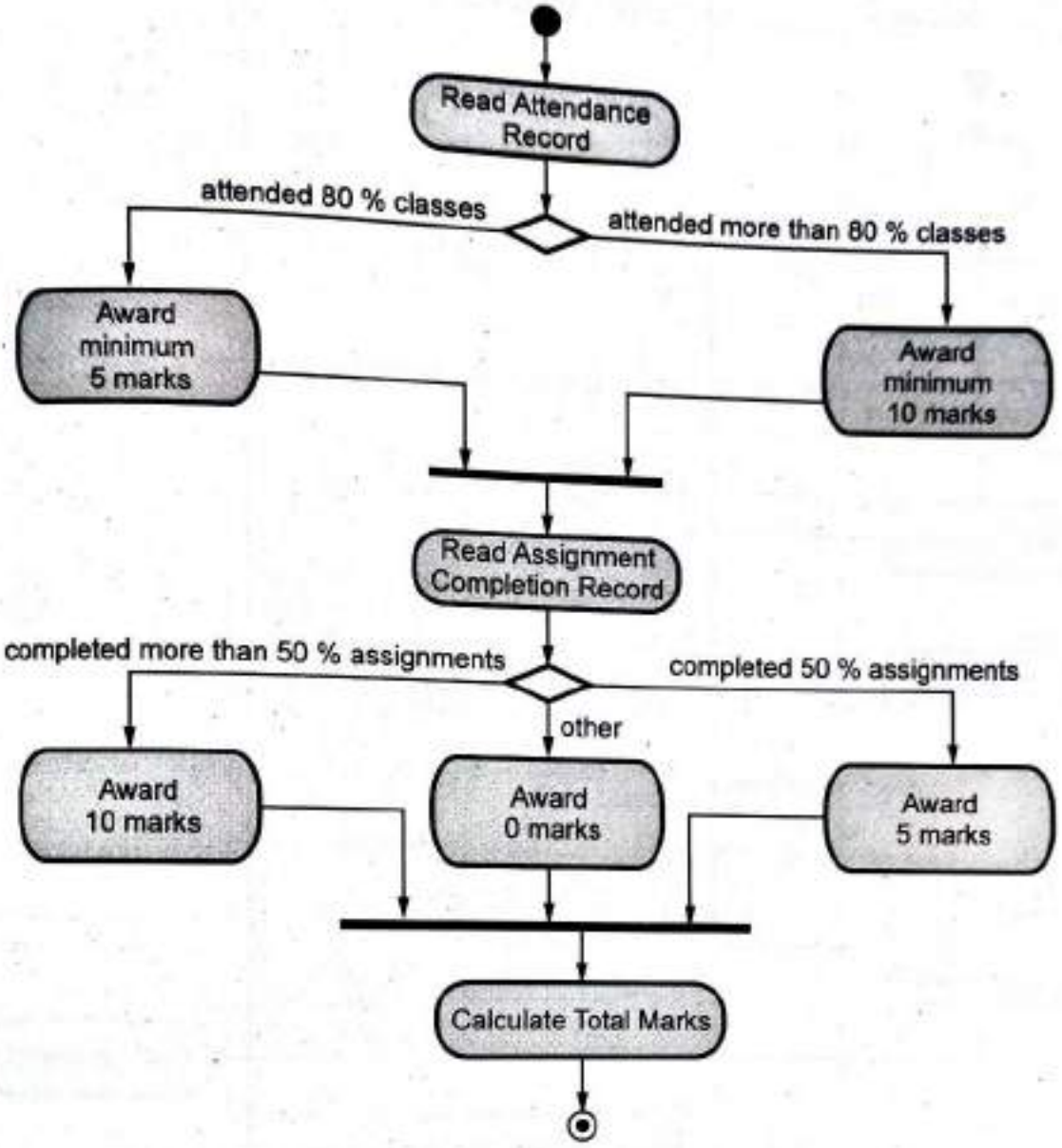


Fig. 2.14.18

Example 2.14.7 *A draw an activity diagram for a Passport Management System to get a new passport, an applicant has to apply on-line, get the appointment. He has to submit the documents in passport office on the date of appointment. In case of insufficient or incorrect documents, the applicant's has to reapply and get new appointment. After submission of documents, applicant's verification is done by the police. On successful verification, passport is issued to the applicant. If verification is unsucessful, applicant has to reapply for passport.*

Solution :

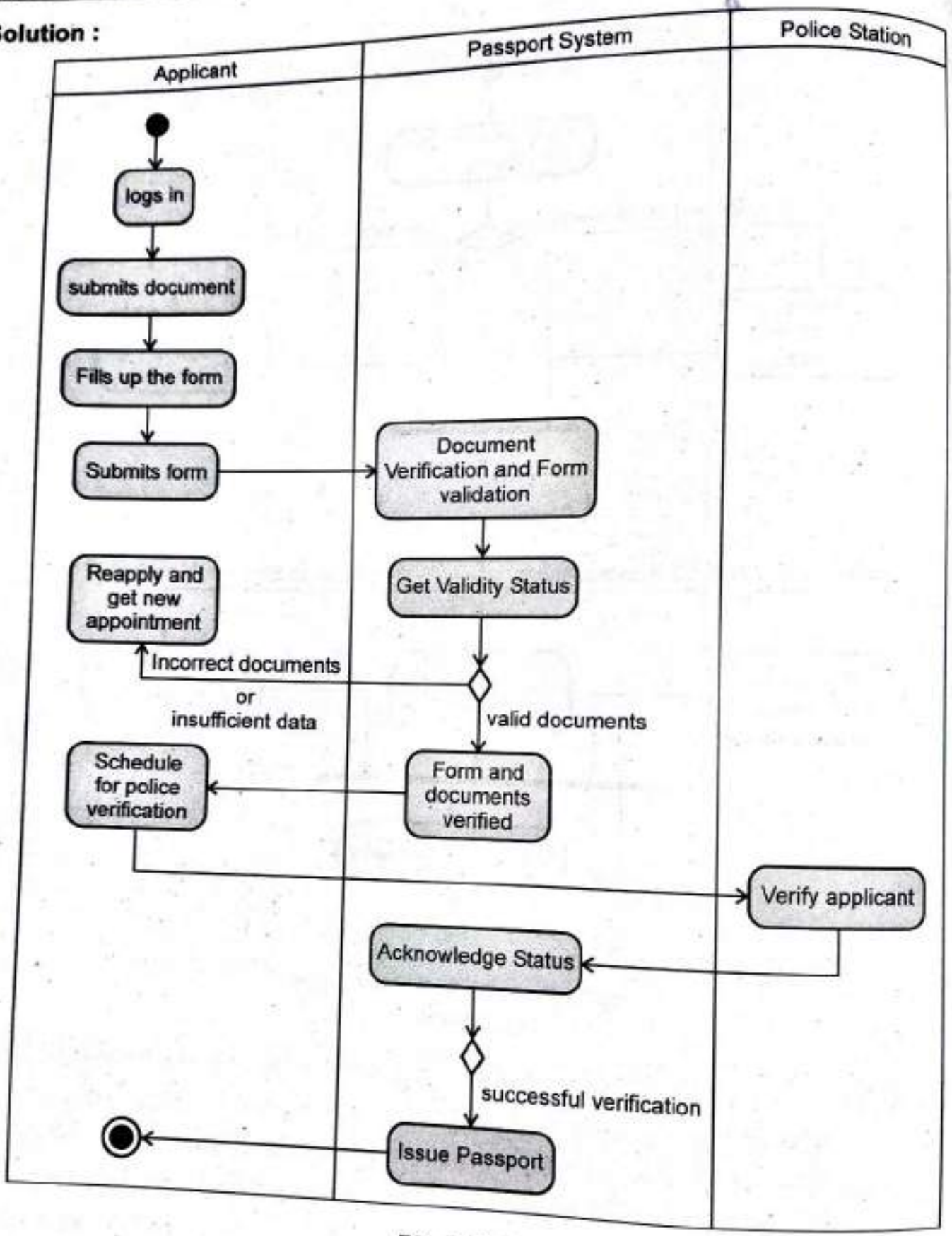


Fig. 2.14.19

Example 2.14.8 *Insurance system provides vehicle insurance to the owners. Initially the customer fill the form which has vehicle details and owner's details. This information is submitted to the agent. The agent sends the information to various insurance companies. The companies quote the insurance. The agent selects the best policy and gives it to the owner. Draw an activity diagram with swimlane and other applicable notations.*

Solution :

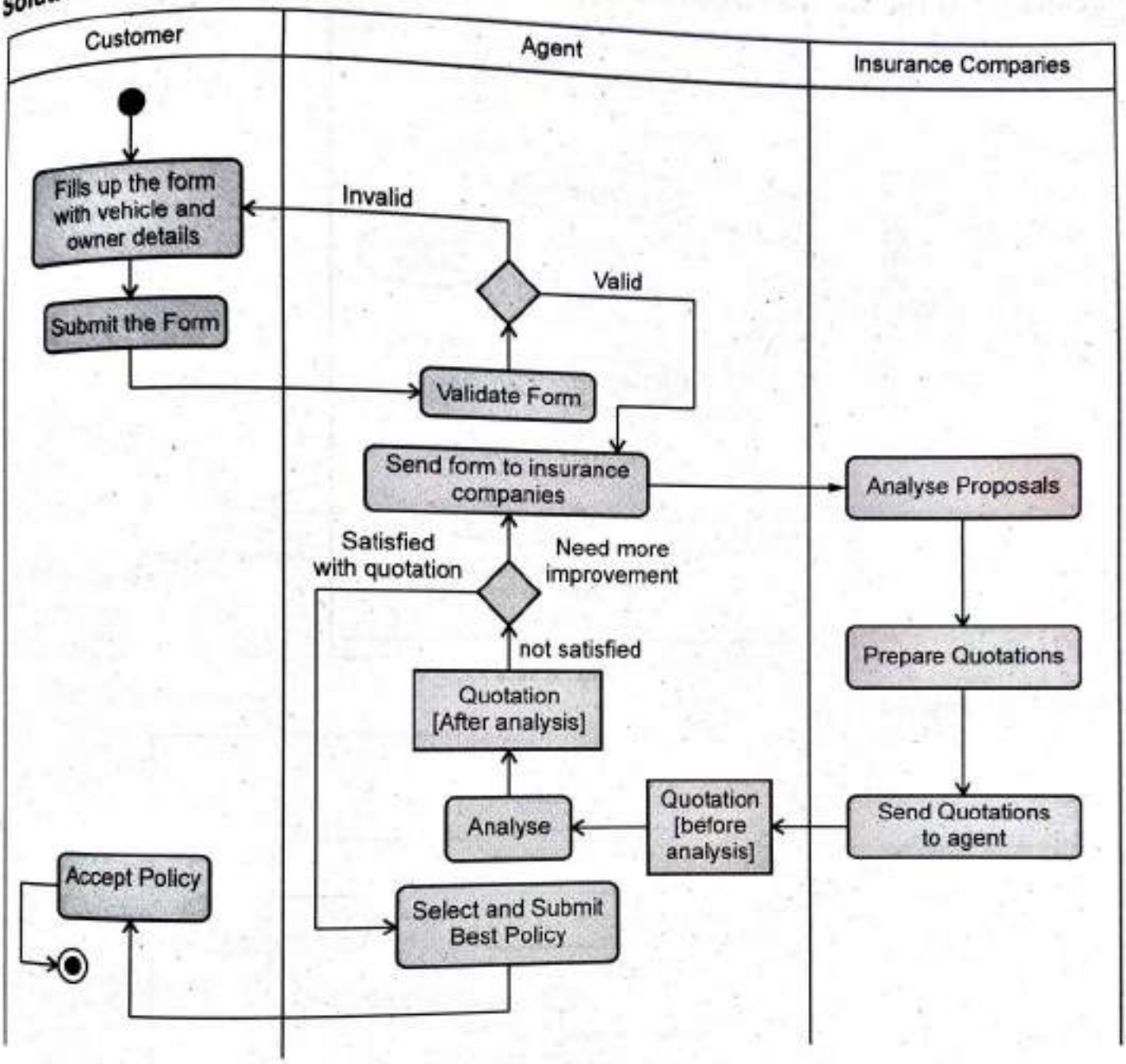


Fig. 2.14.20

Example 2.14.9 *A coffee vending machine dispenses coffee to customers. Customers order coffee by selecting a recipe from a set of recipes. Customers pay for the coffee using coins. Change is given back, if any, to the customers. The 'Service Assistant' loads ingredients (coffee. powder, milk, sugar, water, chocolate) into the coffee machine. The 'Service Assistant' adds a recipe by indicating the name of the coffee, the units of coffee powder, milk, sugar, water and chocolate to be added as well as the cost of the coffee. The Service Assistant can also edit and delete a recipe. i) Develop the use case diagram for the specification above. (Marks 6) ii) For any two scenarios draw an activity diagram and sequence diagram. (5+5 Marks)* **AU : Dec.-13. Marks 16**

Solution : **i)** The use case diagram is as given below -

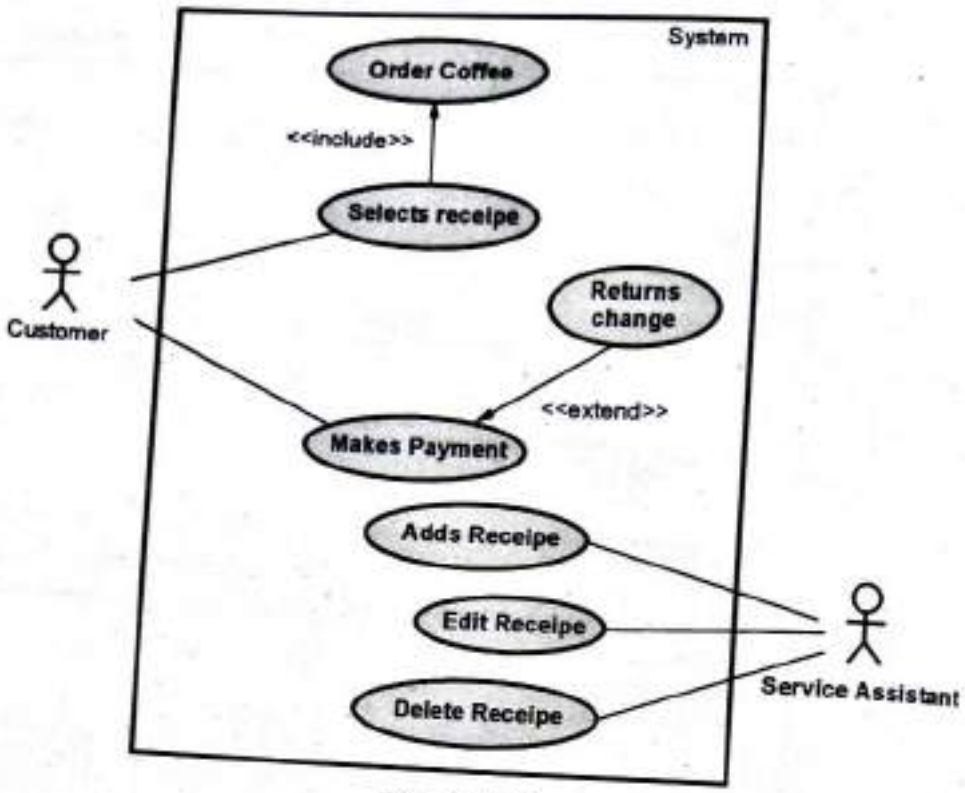


Fig. 2.14.21

ii) Activity diagram

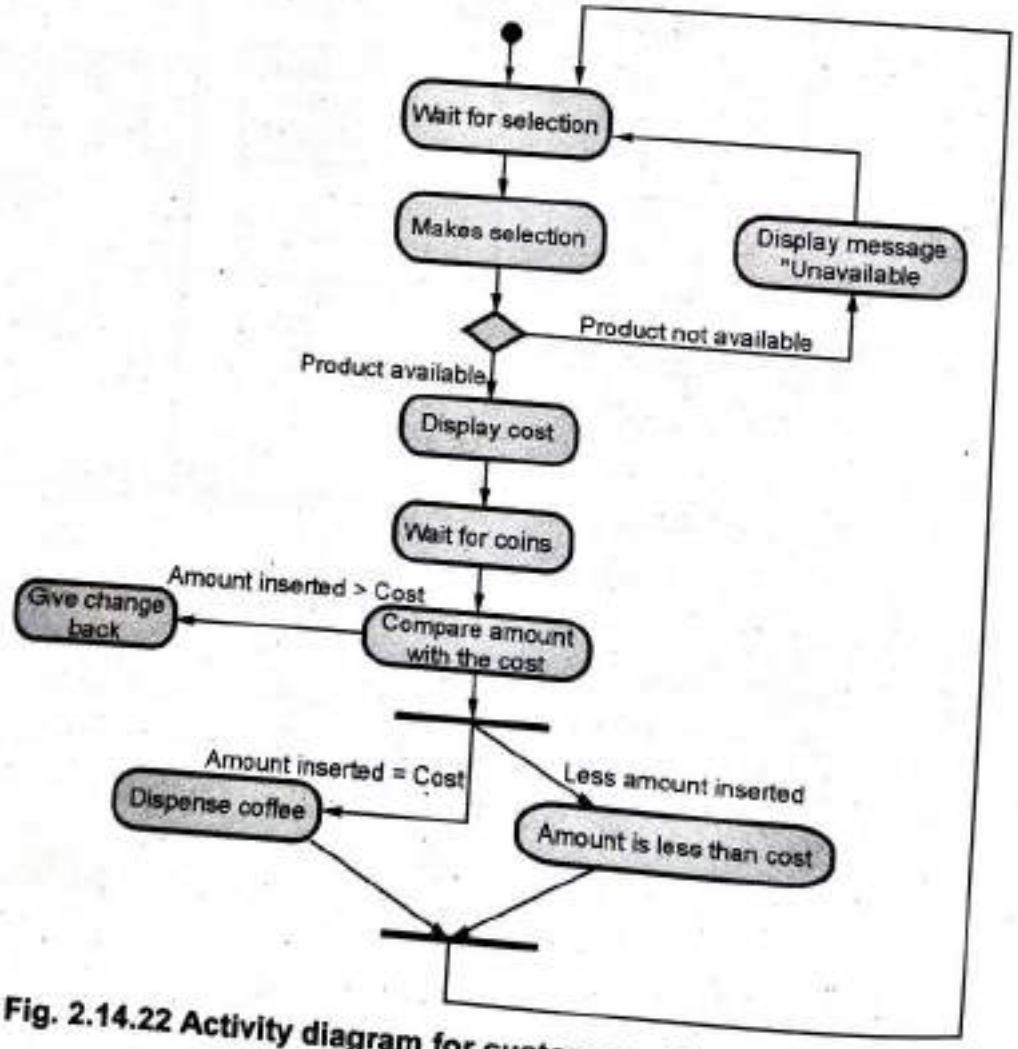
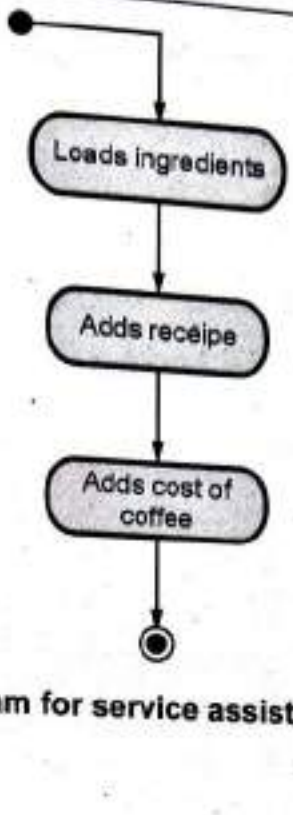


Fig. 2.14.22 Activity diagram for customer orders for the coffee



Fig, 2.14.23 Activity diagram for service assistant for adding the receipe

Sequence diagram :

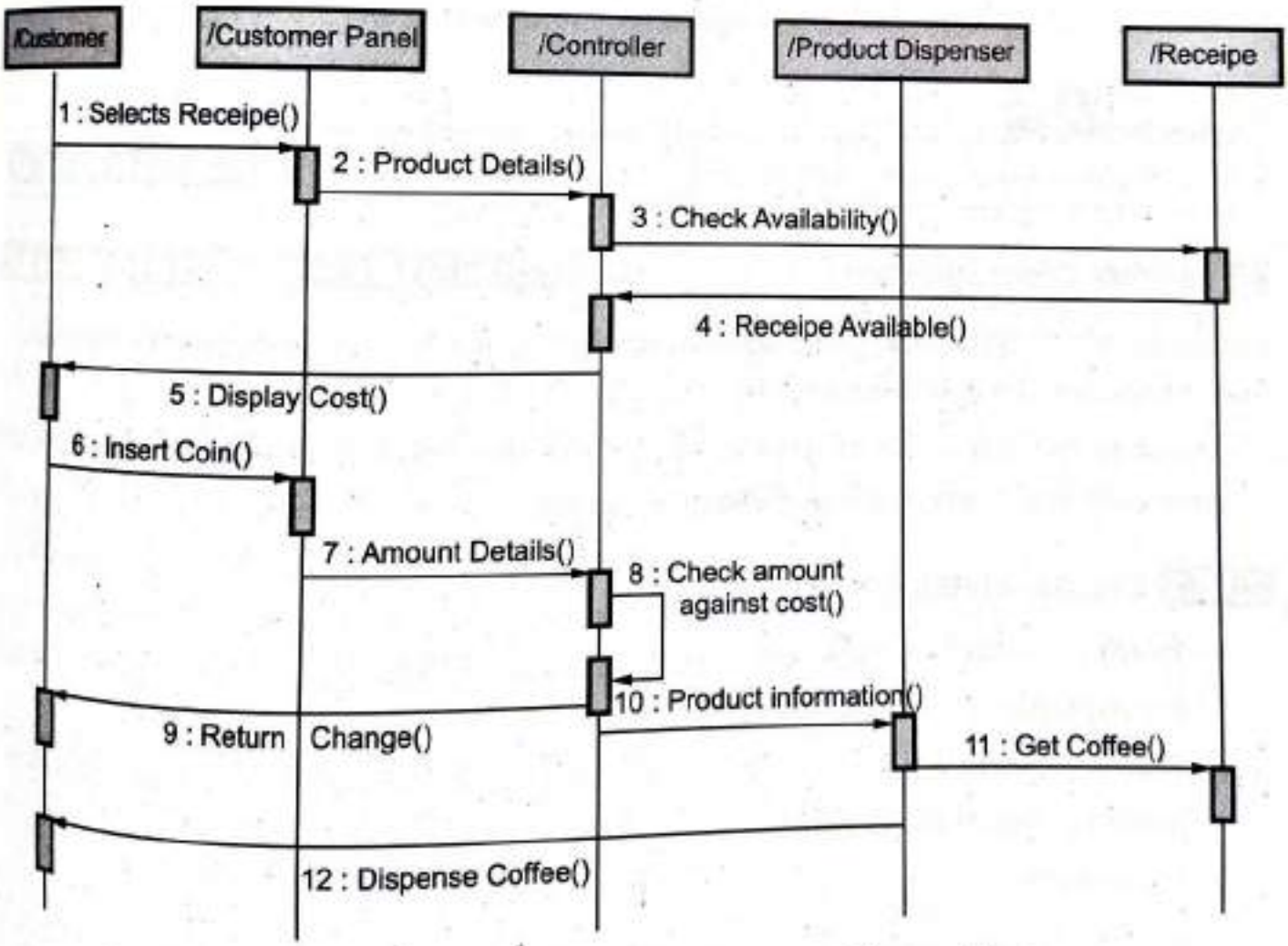


Fig. 2.14.24 Customer getting coffee from vending machine

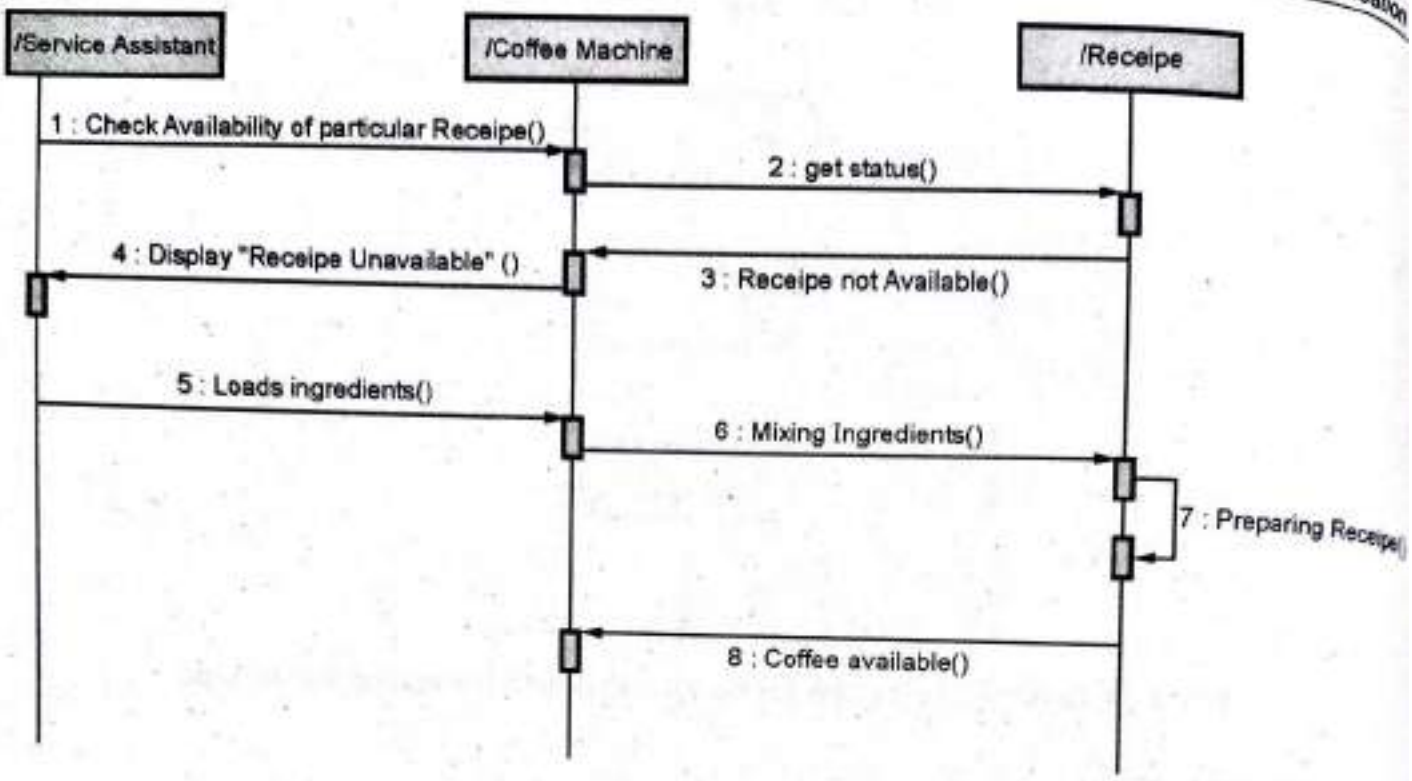


Fig. 2.14.25 Service assistant loads the ingredient and prepare coffee

Review Question

1. *With an example, Explain the need for activity diagram.* **AU : Dec.-18, Marks 13**

2.15 State Chart Diagrams

AU : Dec.-11,13,14, May-11,14,19, Marks 16

Purpose : UML **State machine diagram** represents the events and states of object and the behavior of object in reaction to an event.

The state chart diagram shows the life cycle of the object. Sometimes it is not possible to show each and every event in the state diagram.

2.15.1 Events, States and Transitions

- **Event :** An event is significant occurrence of something.

  For example : Customer inserts ATM card.

- **States :** A state is a condition of object at a particular moment of time. This time is mostly between the events.

  For example :

1. Waiting for user to enter PIN 2. Authenticating the User 3. Dispensing Cash

Note that these states represent that object is waiting for something or doing something.

- **Transitions** : The relationship between the two states is indicated using the transition. When some event occurs the object moves or transits from one state to another. This is represented by the transition.

For example : When the system operator "performs startup"(an event occurs) the transition from **Idle state** to **Active state** occurs.

Example :

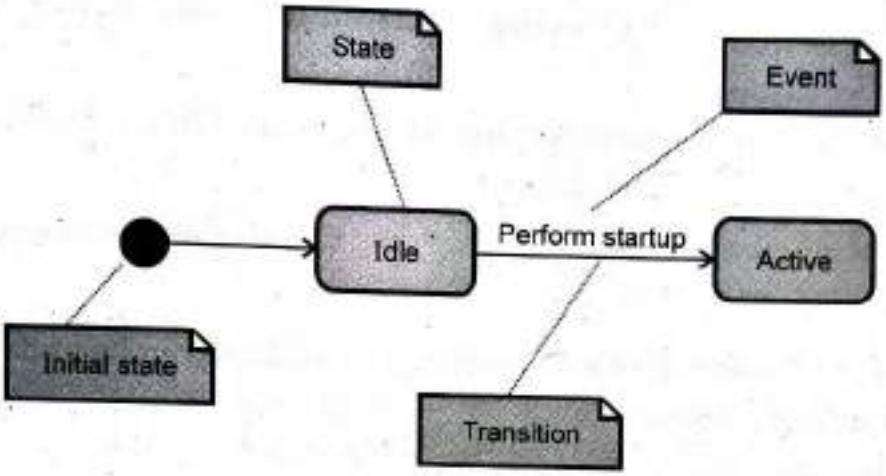


Fig. 2.15.1 State machine diagram for ATM system

2.15.2 Applying State Machine Diagram

If the object responds to the same way to an event then that object is considered as **state-independent** with respect to that event. For example - If an object receives the message "Customer finished the transaction" or "Customer pressed the Cancel button" the corresponding method will be **Ejecting Card**.

If objects respond differently to the events depending upon their states or mode then such objects are considered as **state-dependant** objects.

For example - For an "Insert Card" event the state will be "Reading Card" whereas "Enter PIN" event the state will be "Validating PIN"

Guideline about the state dependant and independent objects

1. The state machines containing state dependant objects with complex behaviour is considered better than the state independent objects.
2. The business information systems have few complex state-dependant classes. On the other hand the process control, device control and protocol control systems have many state dependant objects.

The state machine diagrams can be created for following two good reasons -

1. In order to model the complex reactive objects in response to events.
2. To model legal sequence of operations or language specifications.

A list of common objects used in state model diagram is -

1. Complex reactive objects

- The **physical devices** controlled by the objects can be represented as the sta dependant objects. For example CardReader, Microwave Oven, Washi Machine.
- The **transaction** or **business objects** such as Order, Invoice Manageme Purchase can be state dependant objects.
- The **Role mutator** are the objects that change their role. For example **Student**.

2. Protocols

- The communication protocols such as **TCP** or **UDP** can be understood using th State machine model.

3. Legal sequences

- **User Interface Controller** or **session** such as web applications in which modifie web pages can be displayed to the user based on the current session.
- **Navigation** can be represented using the state machine model. The navigatio represents the transition of one web page to another or transition of windows.
- For representing **Individual UI Window event handling** the state transitio diagram is an effective tool. The sequence of occurring events can be represente using the state dependant object.
- Use case operations can be represented using the state machine model. Fo example - Withdrawal of money, Validating user.

2.15.3 More Notations

During the transactions the event occurs and these events cause the change in stat These transition may represent the call to the method of some specific class. A transitio may be a condition guard which represents the Boolean test. When the condition result true then only the further state change occurs. For example in the following Fig. 2.15.2 th transition between the ***Reading Card*** and ***Requesting PIN*** states is shown. On readin card, it is necessary to validate that card. If the inserted card is valid then only the reques for the PIN is made. When the process of Reading card gets completed then the indicato is glowing.

The ***Glow indicator*** denotes the transition action and the ***Valid Card*** denotes **guard condition.**

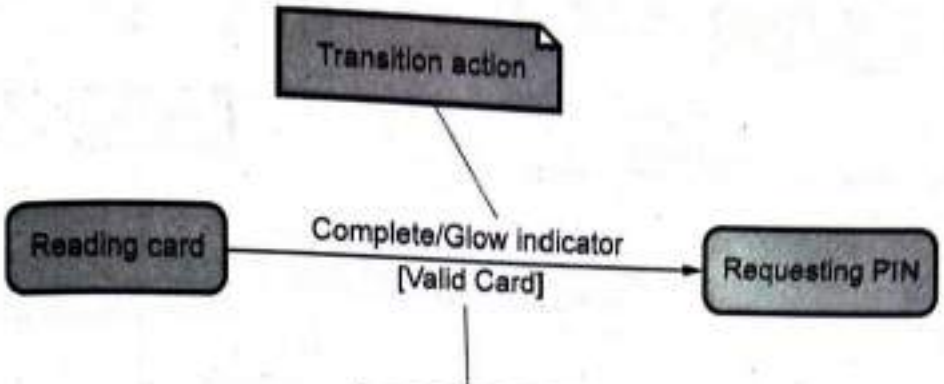


Fig. 2.15.2

Nested states

The nested states are the states that occur within the one super state. For example in the following state machine diagram, the **Active** state is a super state in which the sub states such as **Display, Reading Card Validating PIN** and so on are present. When the Perform shutdown event occurs then the system is restored to the **Idle** state from the **Active state.**

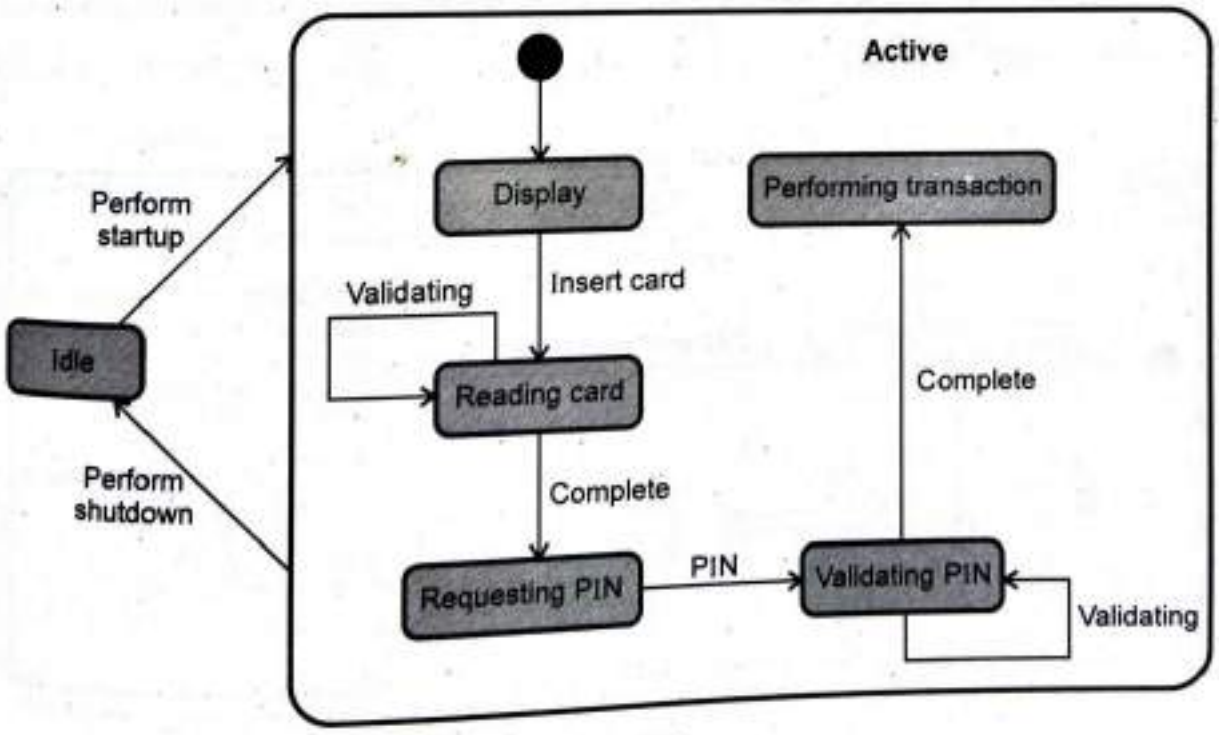


Fig. 2.15.3

Example 2.15.1 *Following is a simple state machine diagram representing one session with ATM System*

Solution :

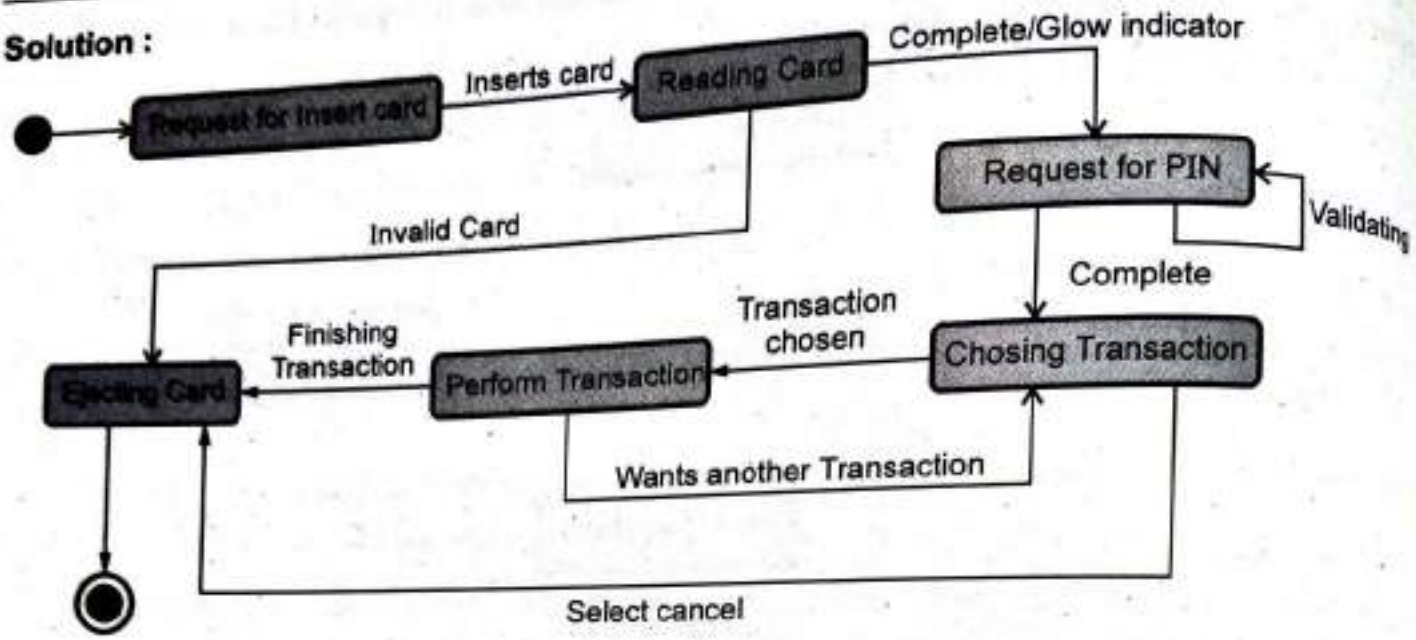


Fig. 2.15.4 State machine model

Example 2.15.2 *Draw state diagram for the control of a telephone answering machine. The machine detects an incoming call on the first ring and answers the call with a prerecorded announcement. When the announcement is complete, the machine records the caller's message. When the caller hands up, the machine hangs up and shuts off. Place the following in the diagram : call detected, answer call, play announcement, record message, caller hangs up, announcement complete.*

Solution :

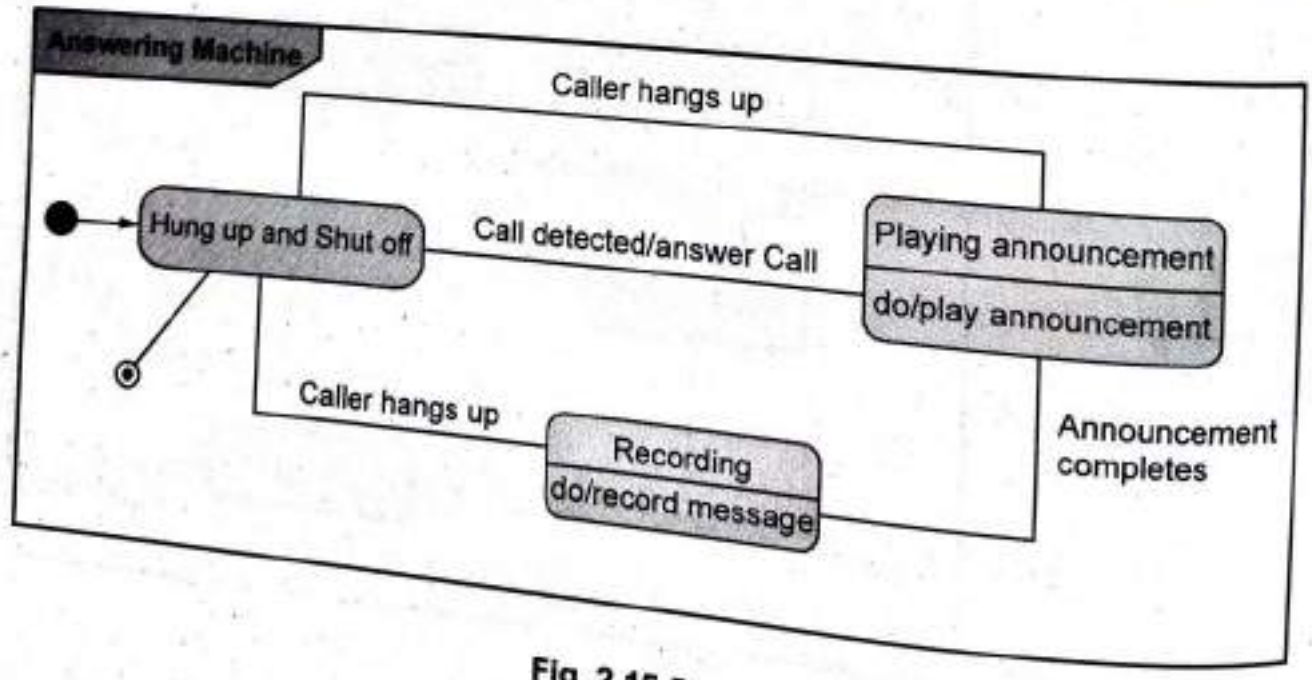


Fig. 2.15.5

Example 2.15.3 *Model a state transition diagram for the following scenario :*

Here is what happens in a microwave oven :

- *The oven is initially in an idle state with door open, where the light is turned on.*
- *When the door is closed it is now in idle but the light is turned off.*
- *If a button is pressed, then it moves to initial cooking stage, where the timer is set and lights are on and heating starts.*
- *At any moment the door may be opened, the cooking is interrupted, the timer is cleared and heating stops.*
- *Also while cooking, another button can be pushed and extended cooking state starts, where the timer gets more minutes. At any moment door can be opened here also.*
- *If the timer times out, then cooking is complete, heating stops, lights are off and it sounds a beep.*
- *When the door is open, again the oven is in idle state with the door open.*

AU : Dec.-17, Marks 13

Solution :

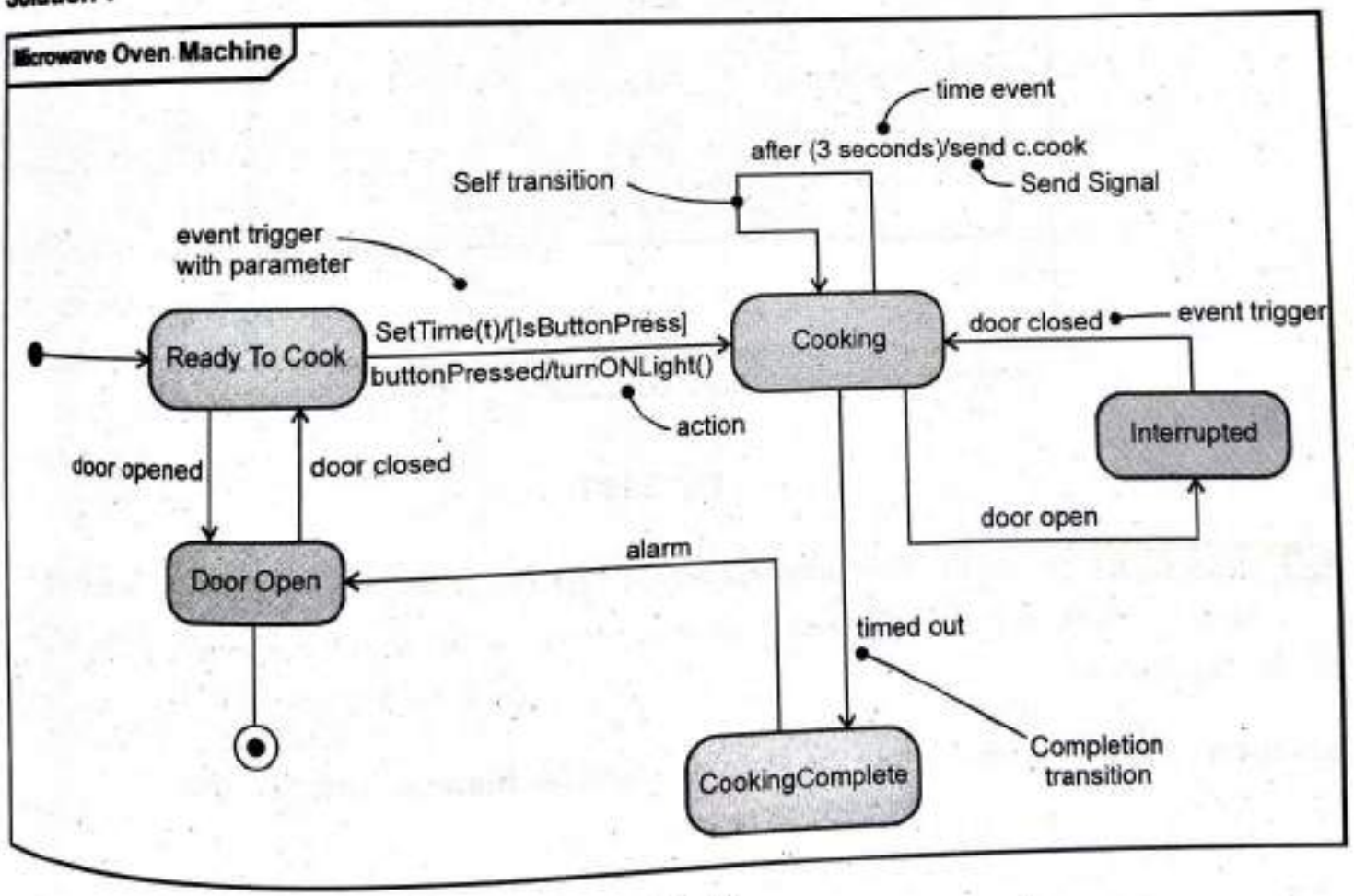


Fig. 2.15.6

Example 2.15.4 *Draw a state diagram for elevator.*

Solution :

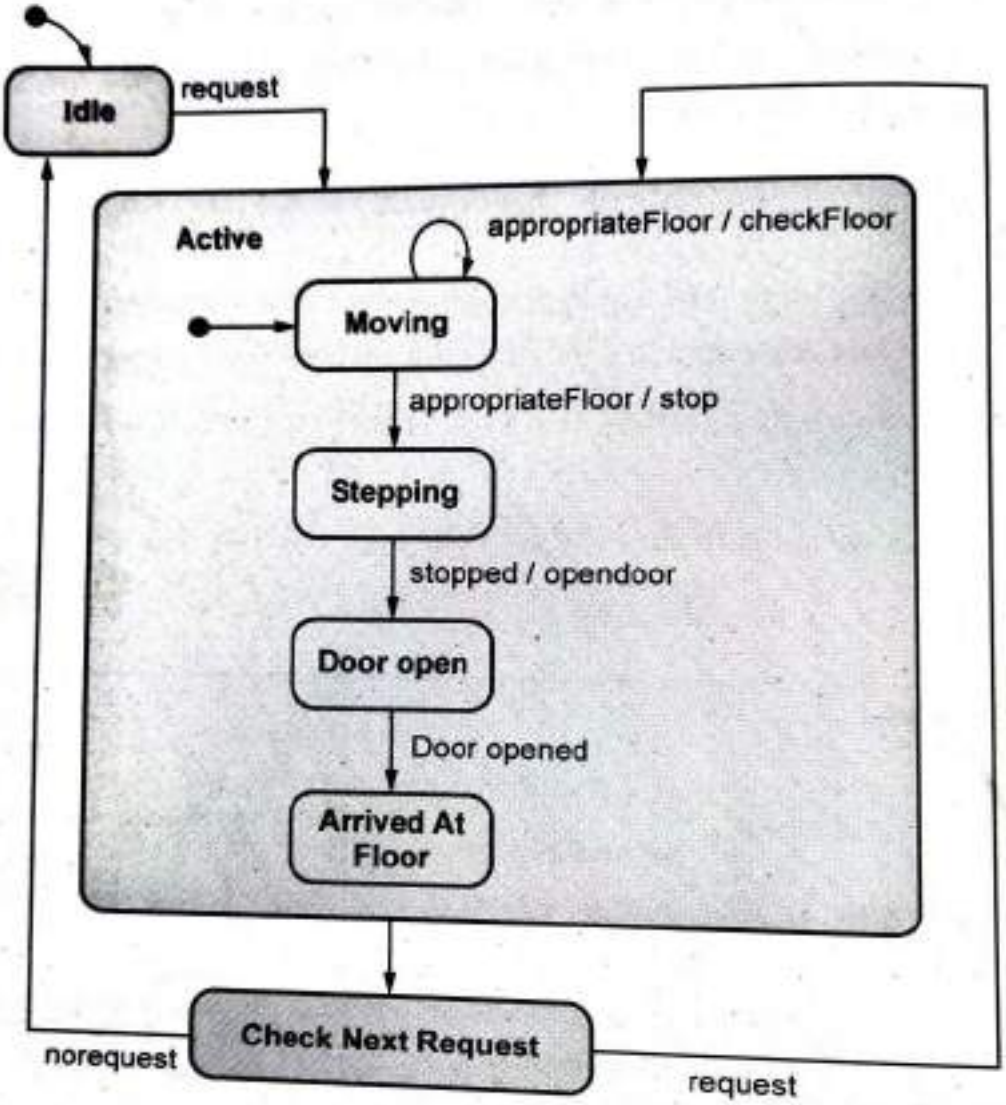


Fig. 2.15.7

Example 2.15.5 *Prepare an object oriented diagram for the controller in a home security system, which is responsible for monitoring various sensors around the perimeter of the house. Briefly explain.*

Solution : Fig. 2.15.8 represents the state diagram for home security system.

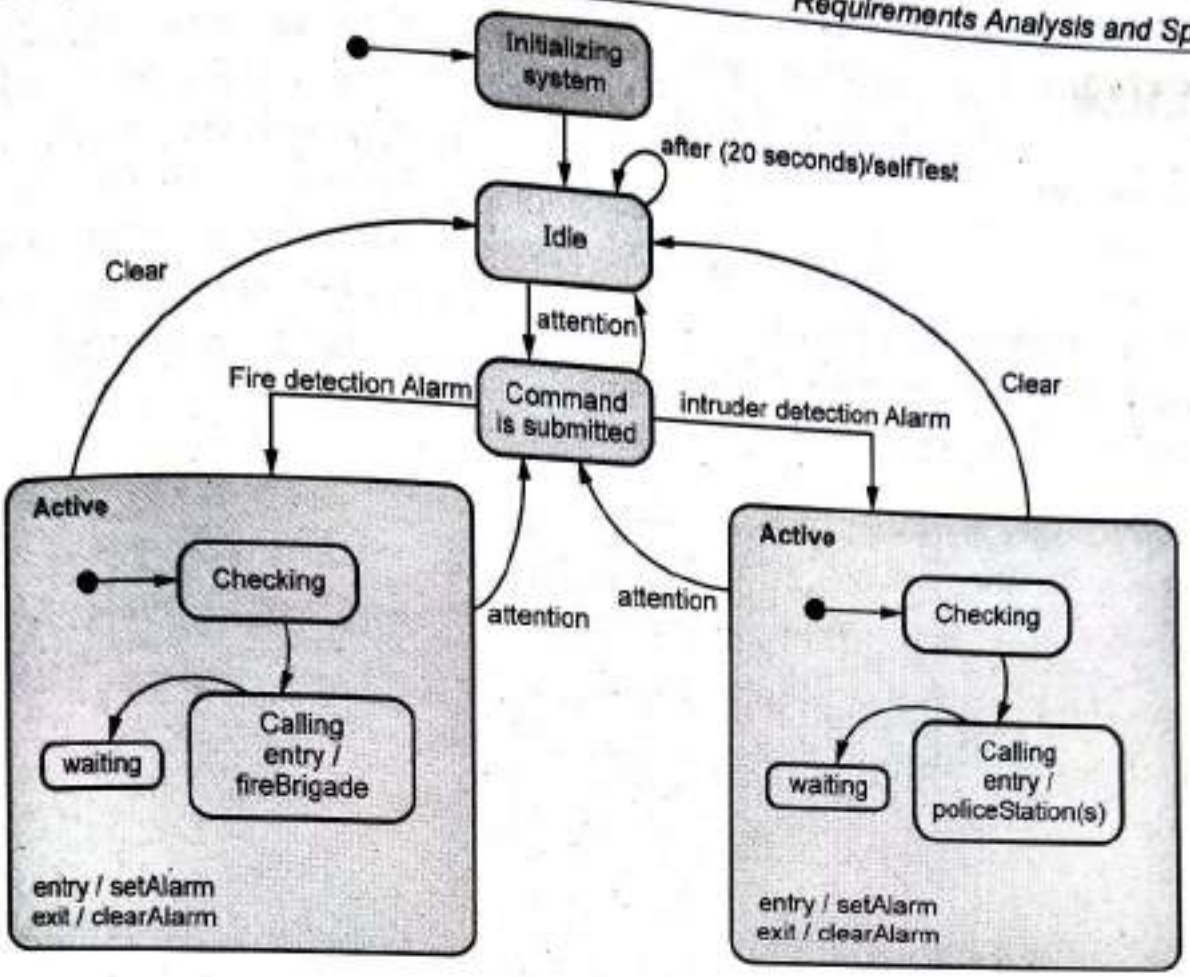


Fig. 2.15.8 Home security system

Example 2.15.6 *Draw a state diagram for a printer.*

Solution : The printer has two explicit states **on** and **off**. When it is **on** it undergoes through various substates such as **Idle, Working, Low tray, Empty tray** and **Full tray** states. When printer is in **off** state it is triggered to on state by pushing the button to on, similarly when printer is on then it can be brought to off state by pushing the button. (Refer Fig. 2.15.9)

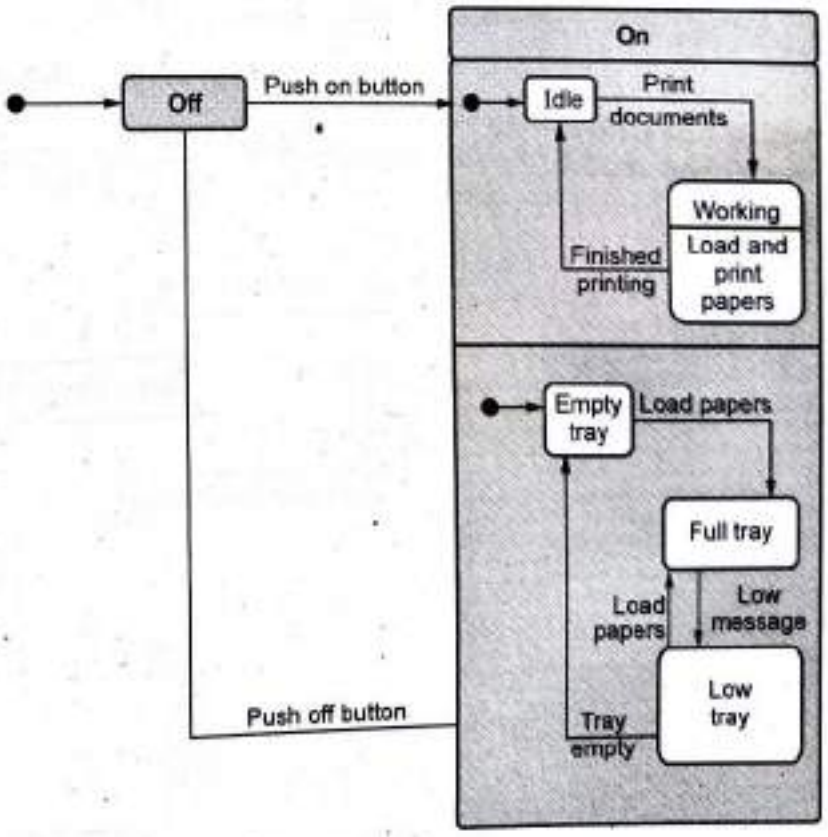


Fig. 2.15.9 State - machine for printer

Example 2.15.7 *A simple digital watch has a display and two buttons, button 'A' and 'B' to set it. The watch has two modes of display, 'display time' mode and 'set time' mode. In the display time mode, the watch displays the hours and minutes separated by a flashing colon. The set time mode has two sub modes, set hours and set minutes. The 'A' button selects modes. Each time it is pressed, the mode advances in a sequence : display time, set hours and set minutes. Within the sub modes, the B button advances the hours or minutes once each time it is pressed. Buttons must be released before they can generate another event. Prepare a state diagram of the watch.*

Solution :

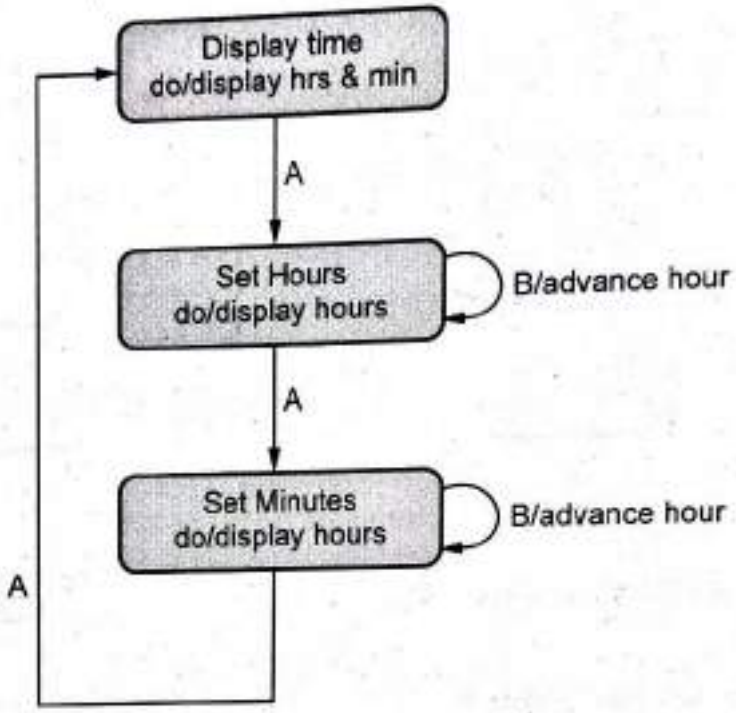


Fig. 2.15.10 Digital watch

Example 2.15.8 *Draw a state machine diagram for coffee vending machine.*

Solution :

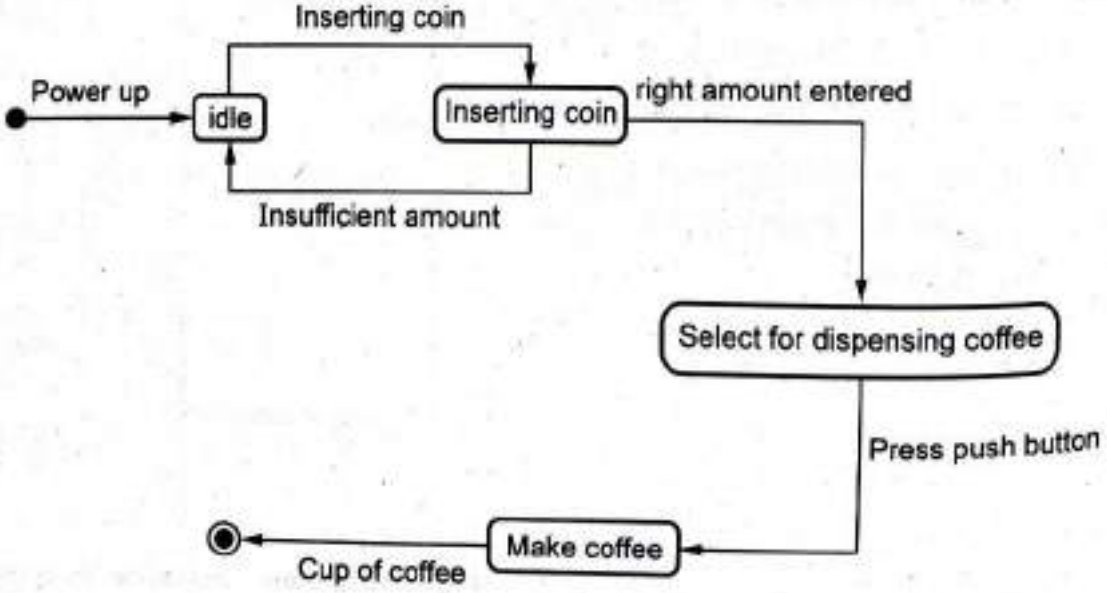


Fig. 2.15.11

2.15.4 When to use State Diagram ?

- State diagrams are used **to model the dynamic aspects** of a system. That means the event ordered behavior of any kind of object is represented using state diagram. The classes, interfaces, components and nodes can be represented in the state diagram.
- The state diagram can be used in context of the **system as a whole**. The state diagrams can also be attached to the use cases.
- The state diagrams can be used **to model reactive objects**. The reactive or event driven object is a kind of object whose behavior is characterized by the **responses to the events**. The reactive object is typically idle until it gets some event. When it receives some event that reactive object responds based on previous event.
- State diagram is used **to model various states of the objects**. These states can be the stable states or the transitive states transiting due to event triggering.
- The state diagrams are used **to model the actions** that occur on each state change.

Review Questions

1. *Write elaborately on UML state diagram and modeling.*
2. *Explain the state chart diagram with a suitable example. Also define its components and use.* **AU : Dec.-11, Marks 16**
3. *Explain UML state machine diagrams and modeling.* **AU : Dec.-13, May-14, Marks 16**
4. *What is the purpose of state chart diagram ? How to draw state chart diagram ? Explain.* **AU : Dec.-14, Marks 8**
5. *Write a short note on the state diagrams.* **AU : May-19, Marks 3**
6. *Give the meaning of event, state and transition.* **AU : May-11, Marks 2**
7. *State independent and state dependant objects.*
8. *How to model state dependant objects ?*

2.16 Functional Modelling

AU : May-16, Marks 8

- In function-oriented design, the design of software system is divided into various **interacting units.**
- Each unit then becomes a function. Thus system is designed from a **functional point of view.**
- This approach is more used in smaller systems but not preferred for larger systems.
- Various models that can be created in function-oriented design are -

1. Data flow models 2. Data dictionaries 3. Structure charts

Difference between Function oriented design and Object oriented design

Sr. No.	Function oriented design	Object oriented design
1.	The problem is decomposed into functions and procedures.	The problem is decomposed into real-world entities as objects.
2.	Functions can be reused and combined to create more complex functionality.	Objects are self-contained entities that contain data and behavior.
3.	In function oriented design, state information is centrally located and several functions on this centralized data are defined.	In object oriented design, state information is shared among the objects.
4.	Functional-oriented design can be difficult to keep track of all the functions and how they interact with each other.	Object-oriented designs are easier to manage as the design is decomposed into objects and modules.
5.	Function-oriented design is associated with procedural programming like C.	Object-oriented design is associated with object oriented programming like C++, Java.
6.	The function-oriented design does not support inheritance and polymorphism.	The object-oriented design supports inheritance and polymorphism.
7.	The function oriented approach is simple to design.	Using this design approach bugs can be easily located and corrected without affecting the other module.
8.	For example - In railway reservation system the major functions will be get_personal_details(),reserve(),cancel(), compute_fair()	For example - To design a railway reservation system various classes could be-Class Passenger, the attributes are : name, age, sex, source, destination and the Operations are : get_personal_data(), reserve(),cancel(), Class Ticket, the attributes are : Ticket_ID, Source, destination, date, fair_amount and operation is : compute_fair()

Review Question

1. *Discuss the differences between object-oriented and function-oriented design.*

AU : May-16, Marks 8

2.17 Data Flow Diagram

AU : May-08,10,15,17,18,19, Dec.-13,15,16,17, Marks 16

The structured system analysis is a technique in which the system requirements are converted into specifications and then into computer programs, hardware configurations and related manual procedures.

- The structured analysis is mapping of problem domain to flows and transformations.
- The system can be modeled using :
  - Entity relationship diagram are used to represent the data model.
  - Data flow diagram which is used to represent the functional model.
- Data Flow Diagram (DFD) represents the flow of information for any process or system.

2.17.1 Rules for Designing DFD

1. No process can have only outputs or only inputs. The process must have both outputs and inputs.

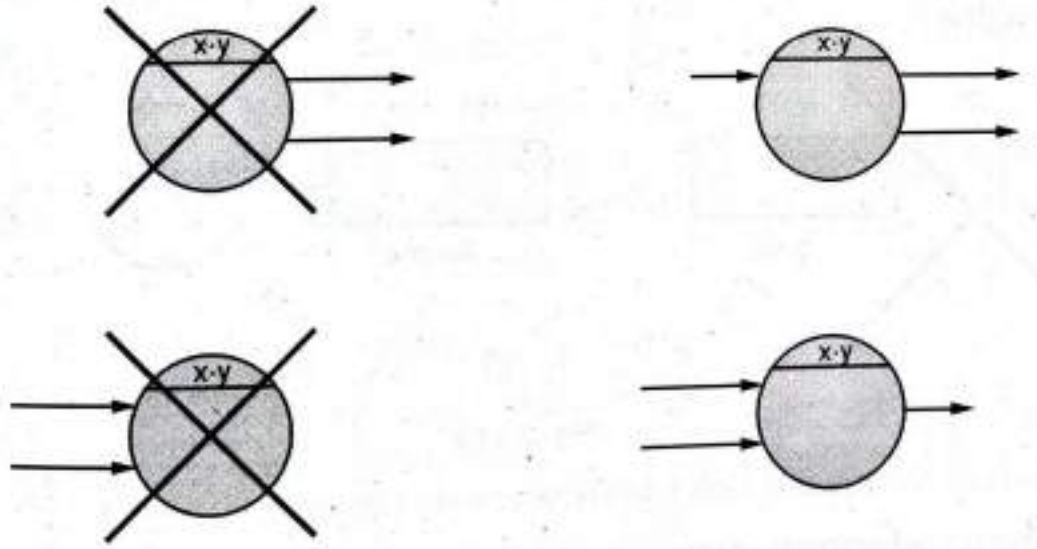


Fig. 2.17.1

2. The verb phrases in the problem description can be identified as processes in the system.
3. There should not be a direct flow between data stores and external entity. This flow should go through a process.

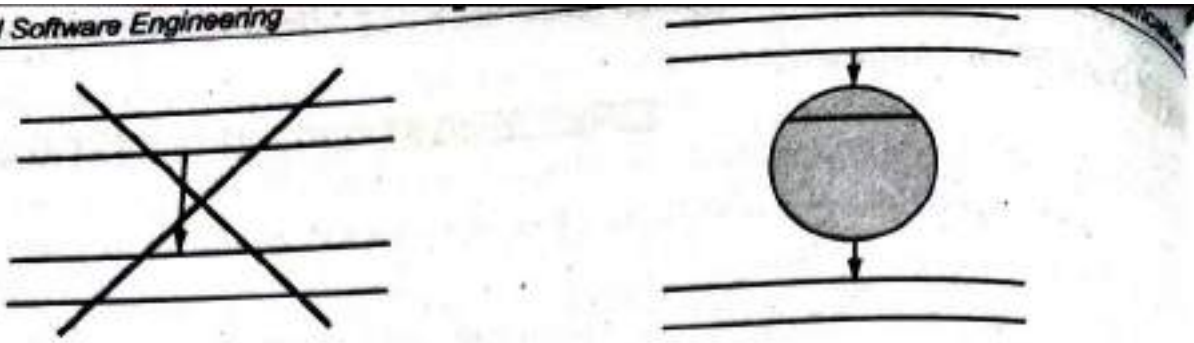


Fig. 2.17.2

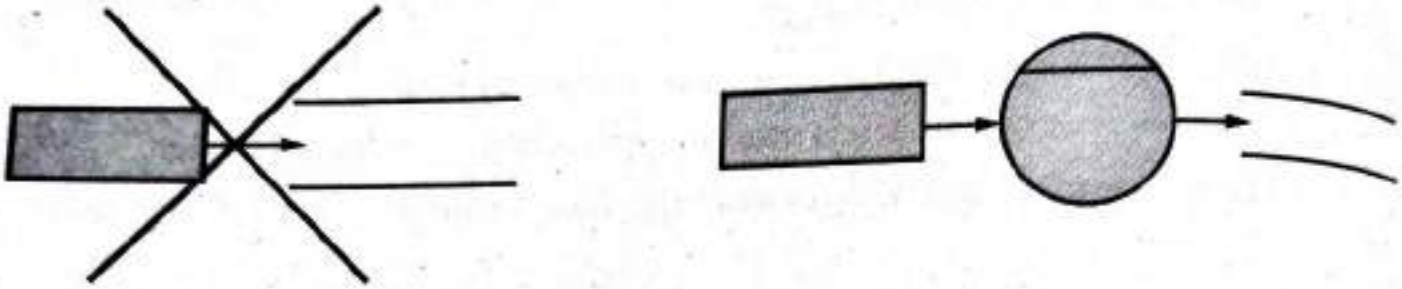


Fig. 2.17.3

4. Data store labels should be noun phrases from problem description.
5. No data should move directly between external entities. The data flow should go through a process.

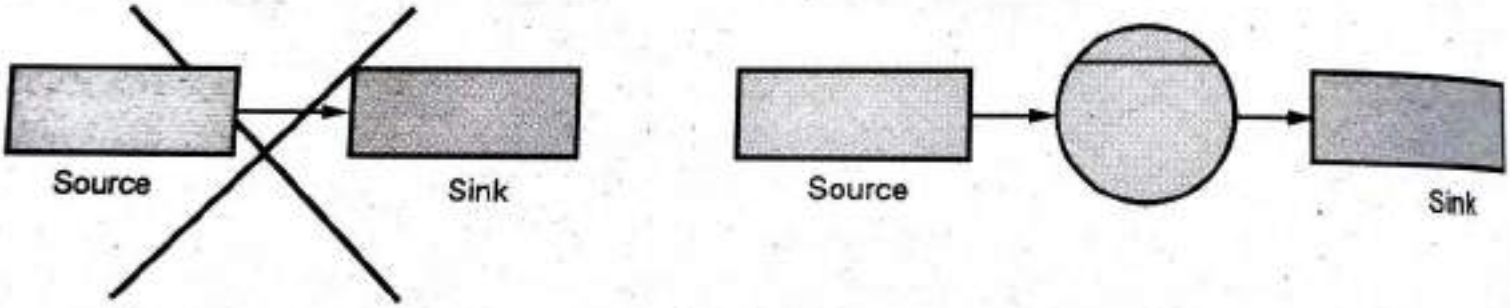


Fig. 2.17.4

6. Generally source and sink labels are noun phrases.
7. Interactions between external entities is outside scope of the system and therefore not represented in DFD.
8. Data flow from process to a data store is for updation/insertion/deletion.
9. Data flow from data store to process is for retrieving or using the information.
10. Data flow labels are noun phrases from problem description.

Example 2.17.1 *Design DFD for library management system for level 0 DFD and level 1 DFD.*

AU : Dec.-16, Marks 8

Solution : A **student** comes to a library for borrowing book. The student makes the book request by giving book title and author name. The student has to submit his library card to the library. Sometimes student may simply give topic and demand for a book (For example "just give me book on data structure"). The library information system maintains list of authors, list of titles, list of topics. This system also keeps record of topics on which books are available with the system. This system maintains information about shelf number on which books are located. Finally the list of demanded book should be displayed, on the console for ease of selection.

The first level of DFD can be

Solution :

In this DFD the whole system is represented with the help of input, processing and output. The input can be -

i) Student requests for a book hence **Book request.**

ii) To show identity of the student he/she has to submit his/her Library card, hence **Library card.** The processing unit can be globally given as

Library information system

The system will produce following outputs-

i) The demanded book will be given to student. Hence **Book** will be the output.

ii) The library information system should display **demanded book information** which can be used by customer while selecting the book.

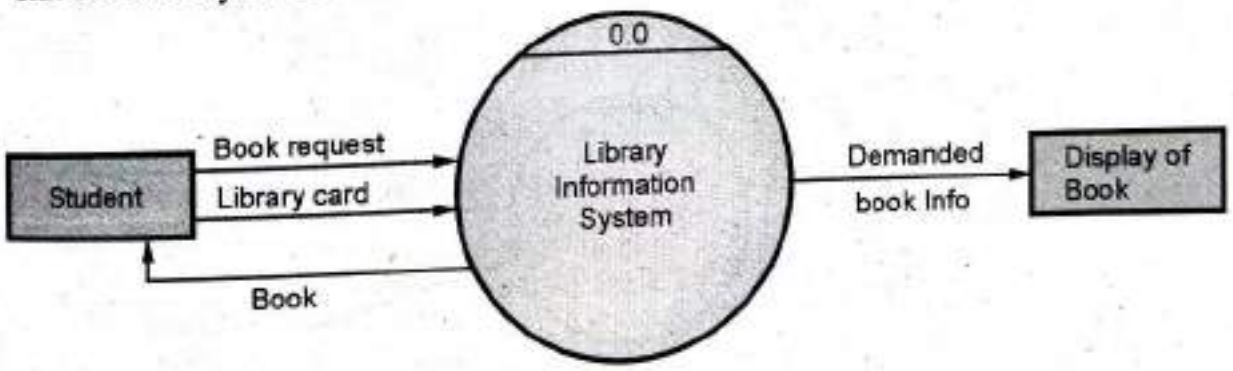


Fig. 2.17.5 Level 0 DFD (Context level DFD)

Level 1 DFD

In this level, the system is exposed with more processing details. The processes that need to carried out are -

i) Delivery of book ii) Search by topic

These processes require certain information such as List of Authors, List of Titles, List of Topics, the book selves from which books can be located. This kind of information is represented by **data store.**

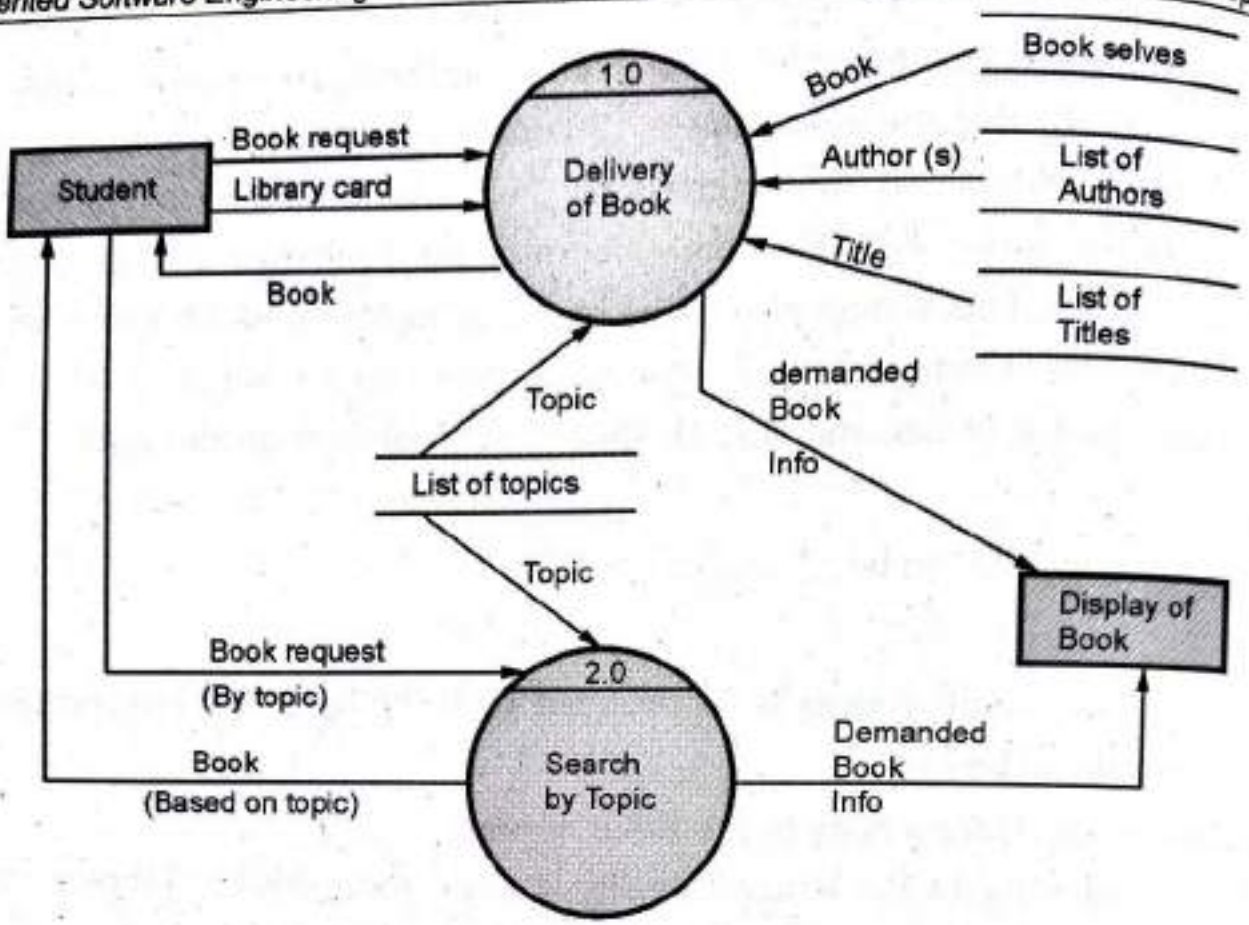


Fig. 2.17.6 Level 1 DFD

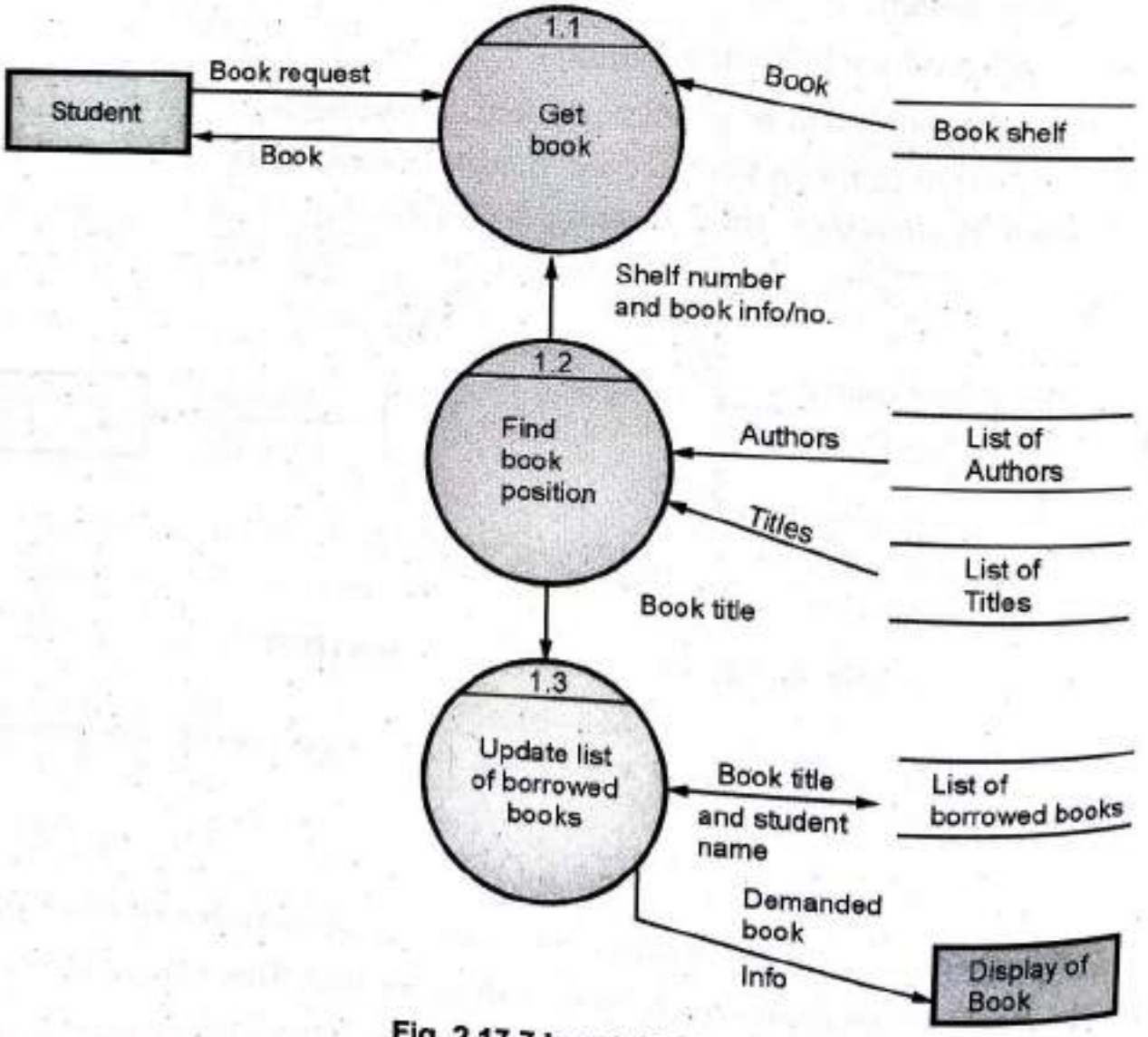


Fig. 2.17.7 Level 2 DFD

Example 2.17.2 *Draw use case and data flow diagrams for a restaurant system. The activities of the restaurant system are listed below. Receive the customer food orders, produce the customer ordered foods, serve the customer with their ordered foods, collect payment from customers, store customer payment details, order raw materials for food products, pay for raw materials and pay for labor.* **AU : Dec.-16, Marks 8, May-15, Marks 16**

Solution : A customer goes to a restaurant and orders for the food. The food order is noted down carefully and this order is sent to kitchen for preparing the required food.

This restaurant has to manage one housekeeping department which maintains **sold items** and **inventory data.** The daily information about sold items and inventory deplition amount is used to generate a **management report**. Finally this management report is given to **restaurant manager.**

In this level, the system is designed globally with input and output. The input to Food ordering system are -

1. As customer orders for the food. Hence food order is an input.

Level 0 DFD

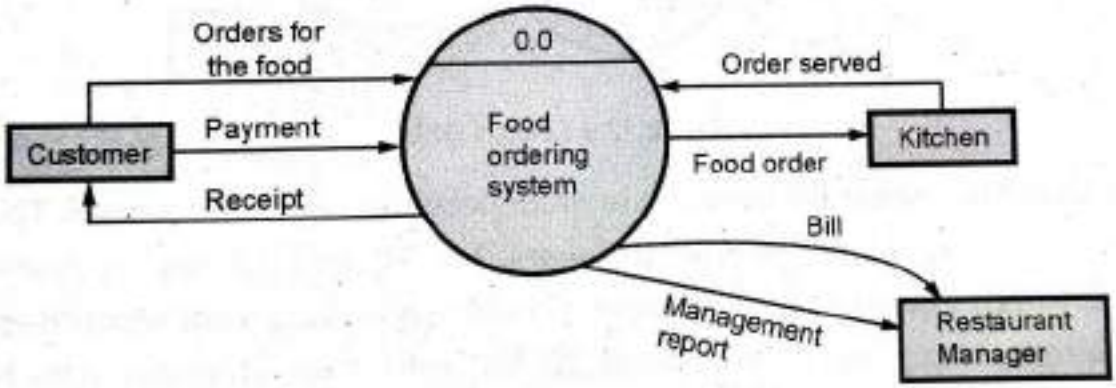


Fig. 2.17.8 Level 0 DFD (Context level DFD)

The output to food ordering system are -

1) Receipt.
2) The food order should be further given to kitchen for processing the order.
3) Bill and management report is given to restaurant manager.

Level 1 DFD

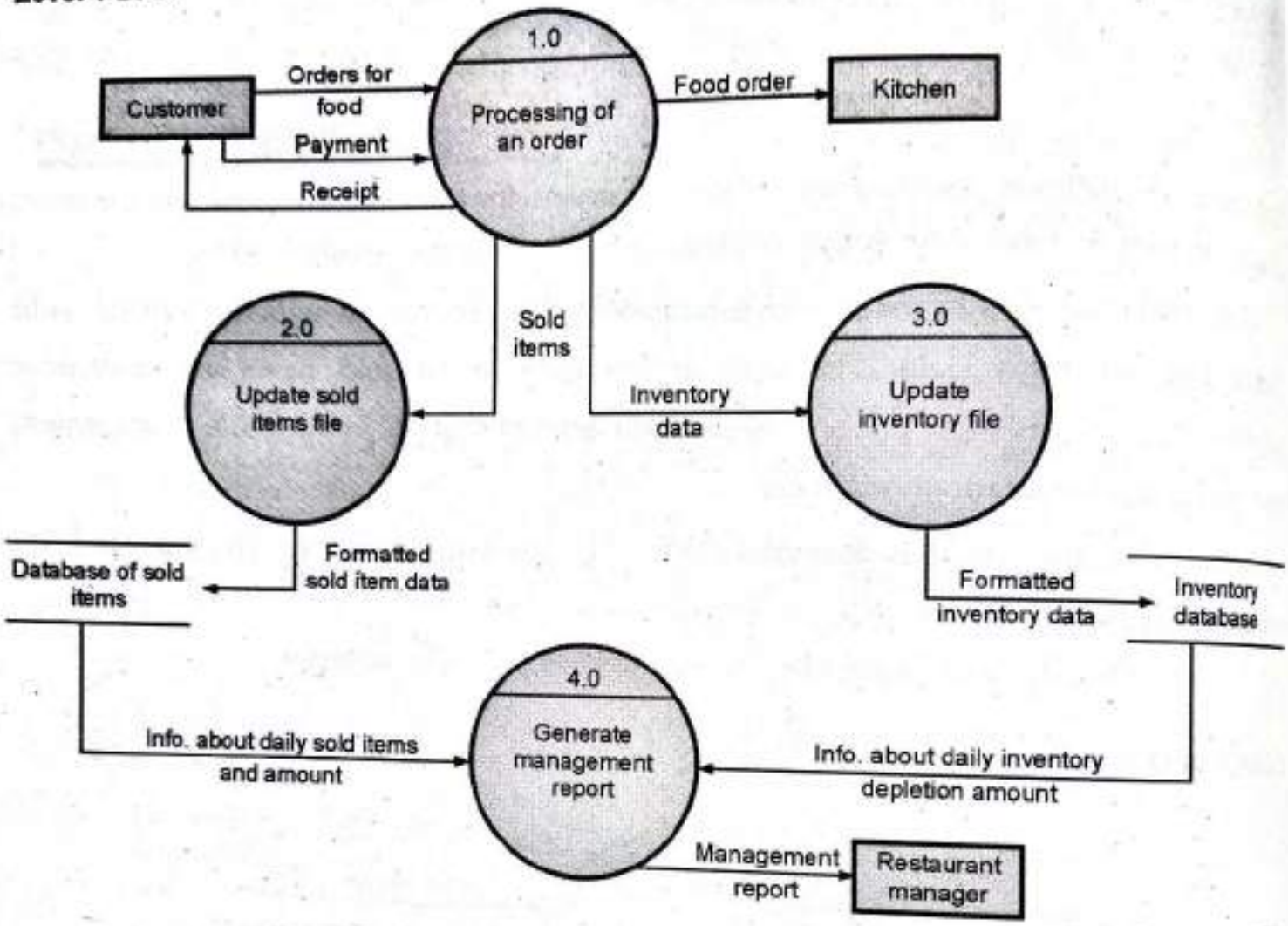


Fig. 2.17.9 Level 1 DFD

In this level, the bubble 0.0 is shown in more detail by various processes. The process 1.0 is for processing an order. And processes 2.0, 3.0 and 4.0 are for housekeeping activities involved in food ordering system. To create a management report there should be some information of daily sold items. At the same time inventory data has to be maintained for keep track of 'instock' items. Hence we have used two **data stores** in this DFD -

1. Database of sold items
2. Inventory database

Finally management report can be prepared using daily sold details and daily inventory deplition amount. This management report is given to restaurant manager.

In this DFD we will elabortate "processing of order".

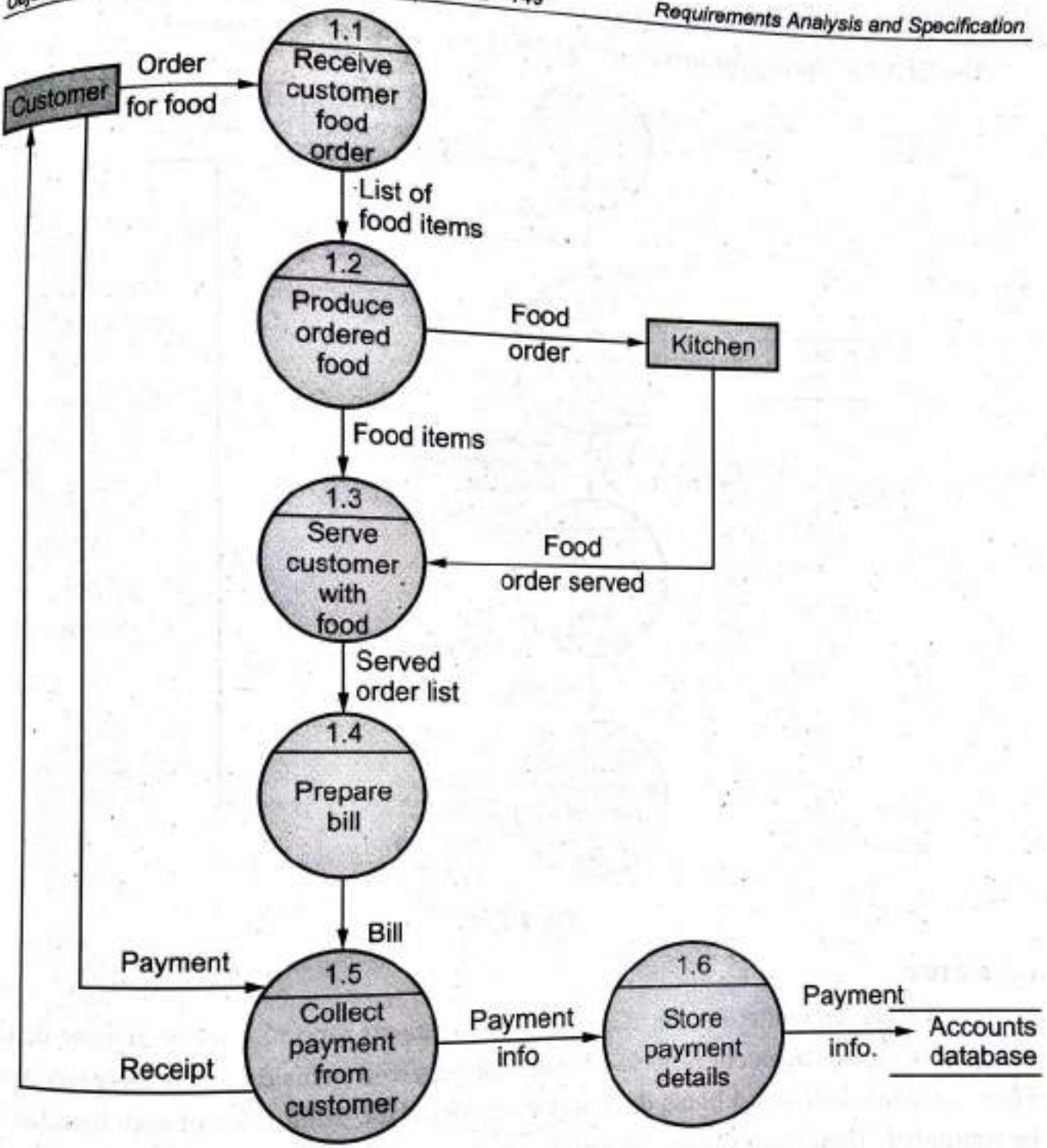

Customer
Order for food
1.1 Receive customer food order
List of food items
1.2 Produce ordered food
Food order
Kitchen
Food items
1.3 Serve customer with food
Food order served
Served order list
1.4 Prepare bill
Bill
Payment
1.5 Collect payment from customer
Receipt
Payment info
1.6 Store payment details
Payment info.
Accounts database

Fig. 2.17.10

The DFA for "update for inventory file " is shown in detail as follows -

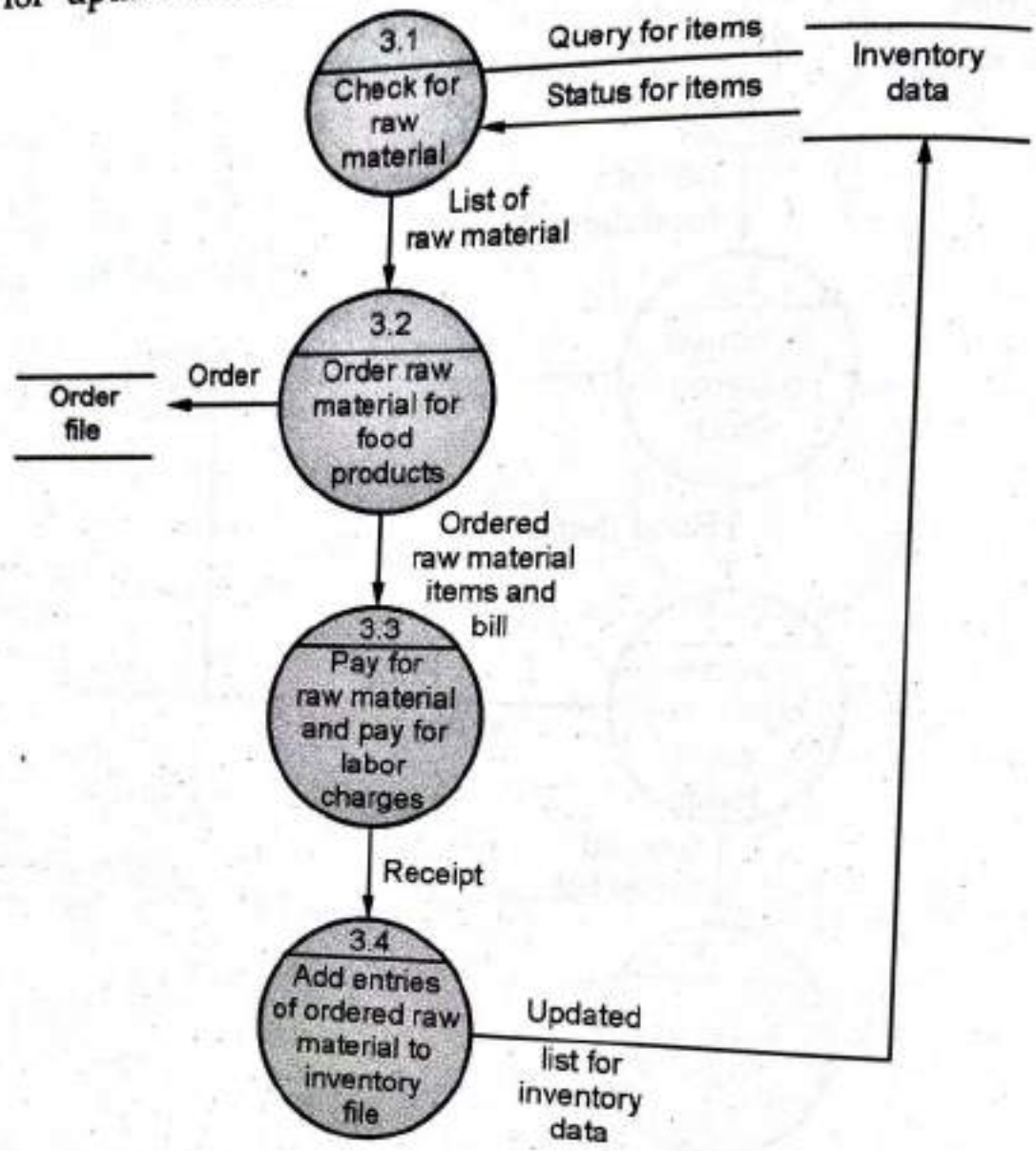


Fig. 2.17.11

Level 2 DFD

In this DFD we will elaborate "Generate management report" activity in more detail. For generating management report we have to access sold items data and inventory data. Then aggregate both solid items data and inventory data. Total price of each item has to be computed. Then from these calculation a management report has to be prepared and given to the restaurant manager. These details can be shown in this DFD.

(See Fig. 2.17.12 on next page.)

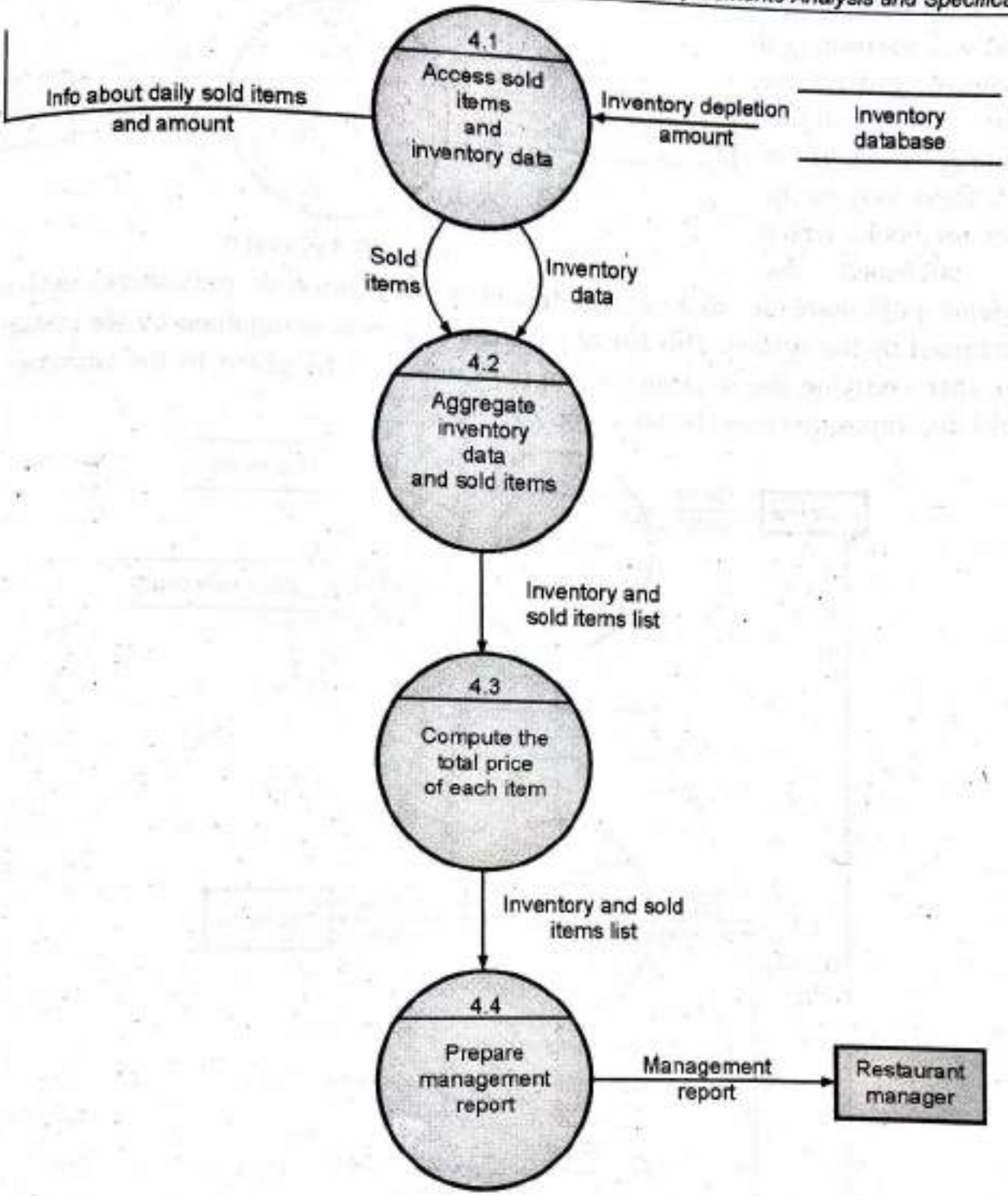


Fig. 2.17.12 Level 3 DFD

Example 2.17.3 *Prepare a DFD for a Books order processing system.*

Solution : The **customers** in this system are book sellers who do not make a stock of books. As the **orders** come the books are demanded from **publishers** directly.

As per the problem description it is clear that the input to this system is 'customer' who places an order and output will be purchase order given to publisher.

Now, we will do more detailing for order processing when an order is received by the system, it will be first verified using the books database. Credit rating will be decided by checking customer details (i.e. whether the customer is regular customer or whether he is

REVIEW AND RECOMMENDATION

new). For submitting a batch of orders we have to maintain pending orders list as well. There may be the order for books which are published by different publishers, in such a case database for different publishers need to be maintained by the system. This list of purchase orders is maintained by the system and then after verifying the shipment a shipping note will be given to the customer. The level 1 decomposition is as shown in Fig. 2.17.14.

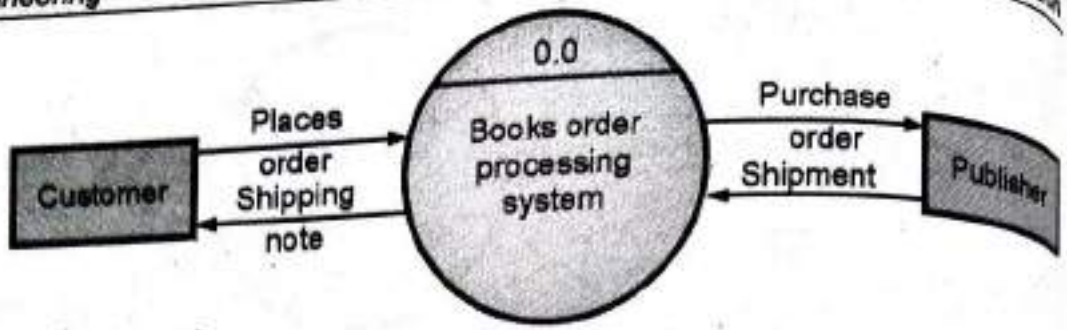
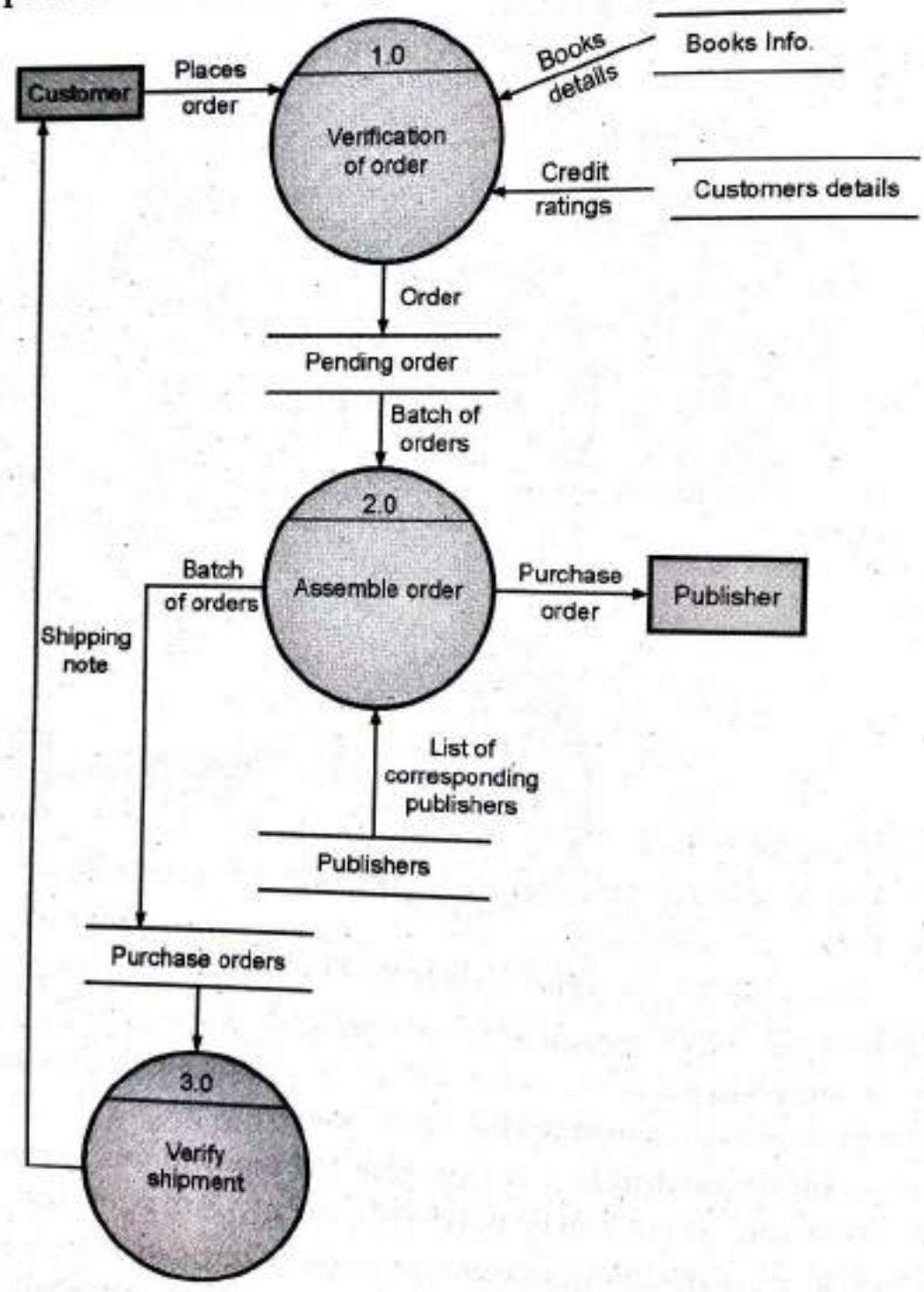


Fig. 2.17.13 Level 0



We can further decompose the system by elaborating the process **Assemble order.** In assemble order we

- First get details of total copies per title.
- Then prepare purchase order accordingly for corresponding publishers.
- Send these purchase orders to publishers.
- These purchase order details should be stored in the system. Let us draw level 2 DFD for these details.

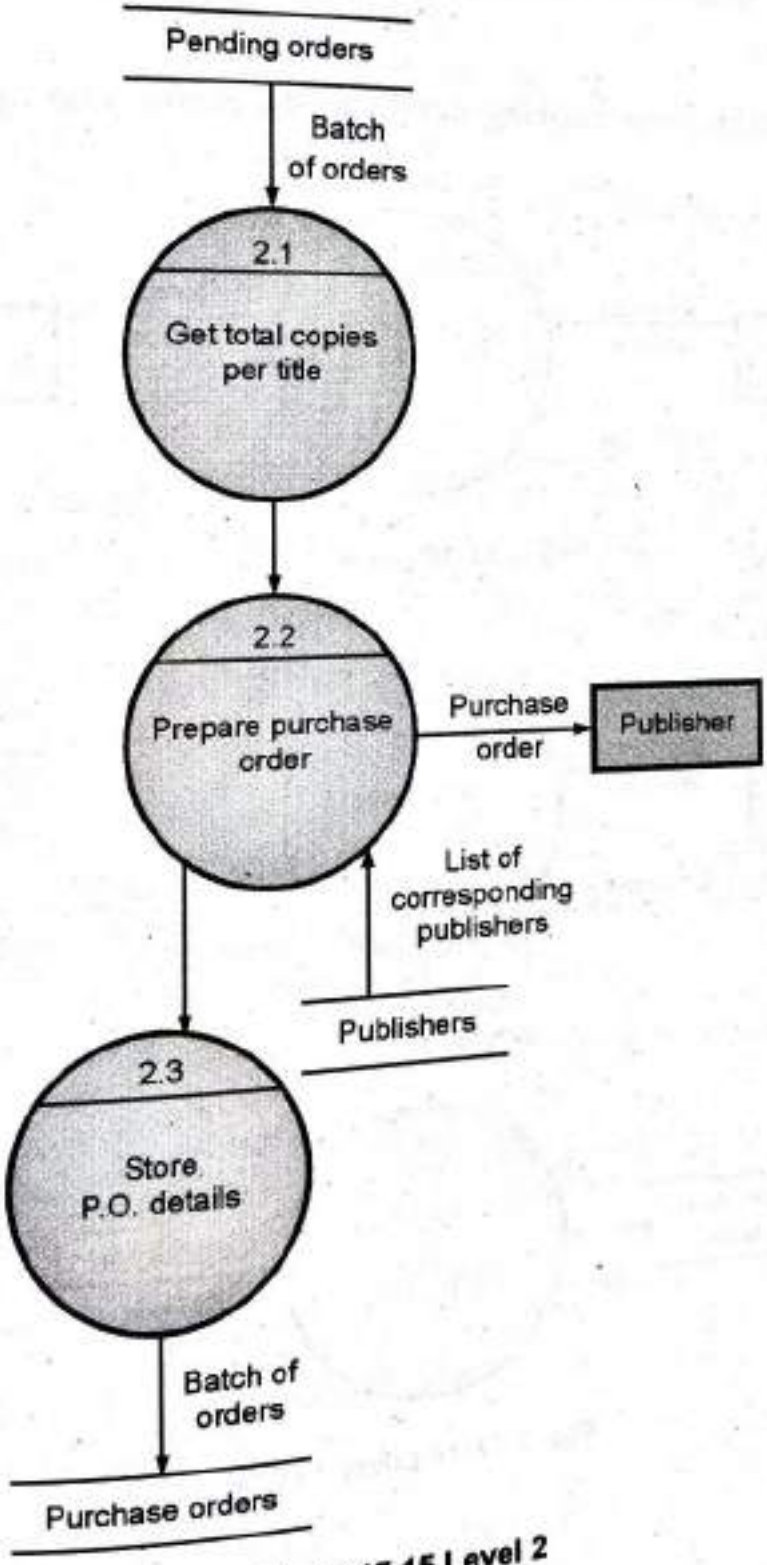


Fig. 2.17.15 Level 2

Example 2.17.4 *Design DFD for Hotel reservation system.*

Solution : In a hotel reservation system, a **customer** can make onlihe booking for a hotel, by specifying the accomodation requirements such as type of room (AC/Non AC/One bed/Two bed), total number of rooms, duration of stay. The system then **selects** a suitable hotel as per customer's requirements. If such a hotel is found then the **availability** of rooms in that hotel is **checked.** The **charges are calculated** for the selected requirement and these are acknowledged to the customer. If the customer is satisfactory about the selection made by the system then he **confirms** the reservation.

The system then gives these booking details to the corresponding **hotel**. Design DFD for this system.

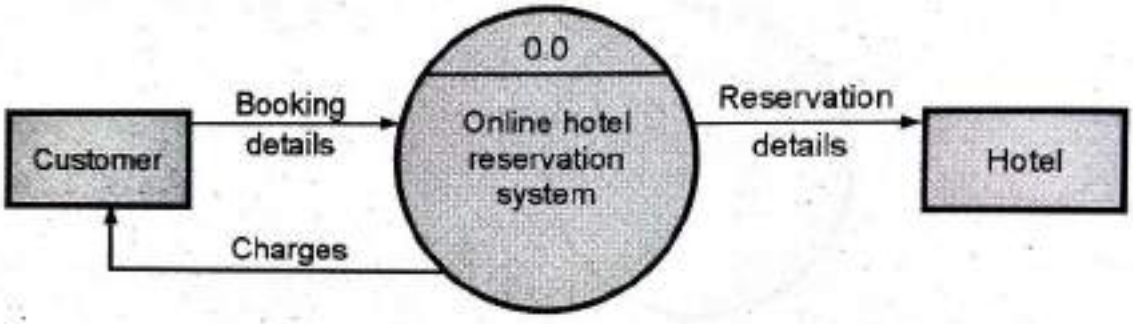


Fig. 2.17.16 Level 0 DFD

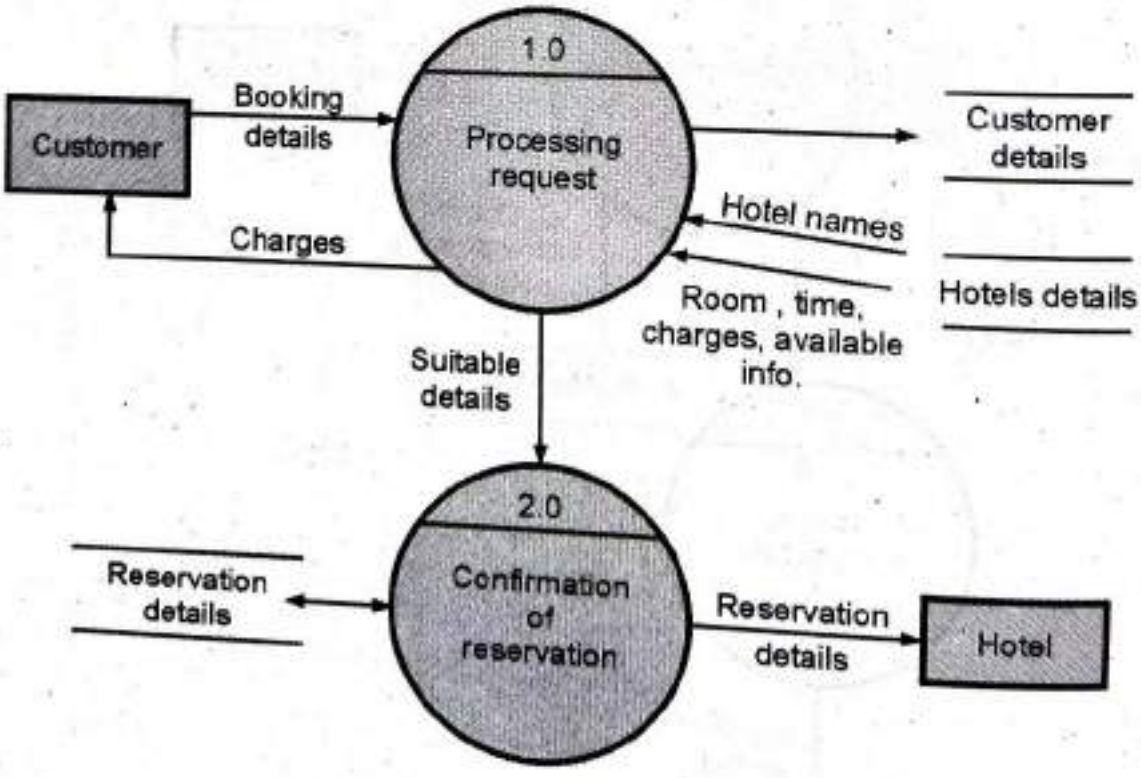


Fig. 2.17.17 Level 1 DFD

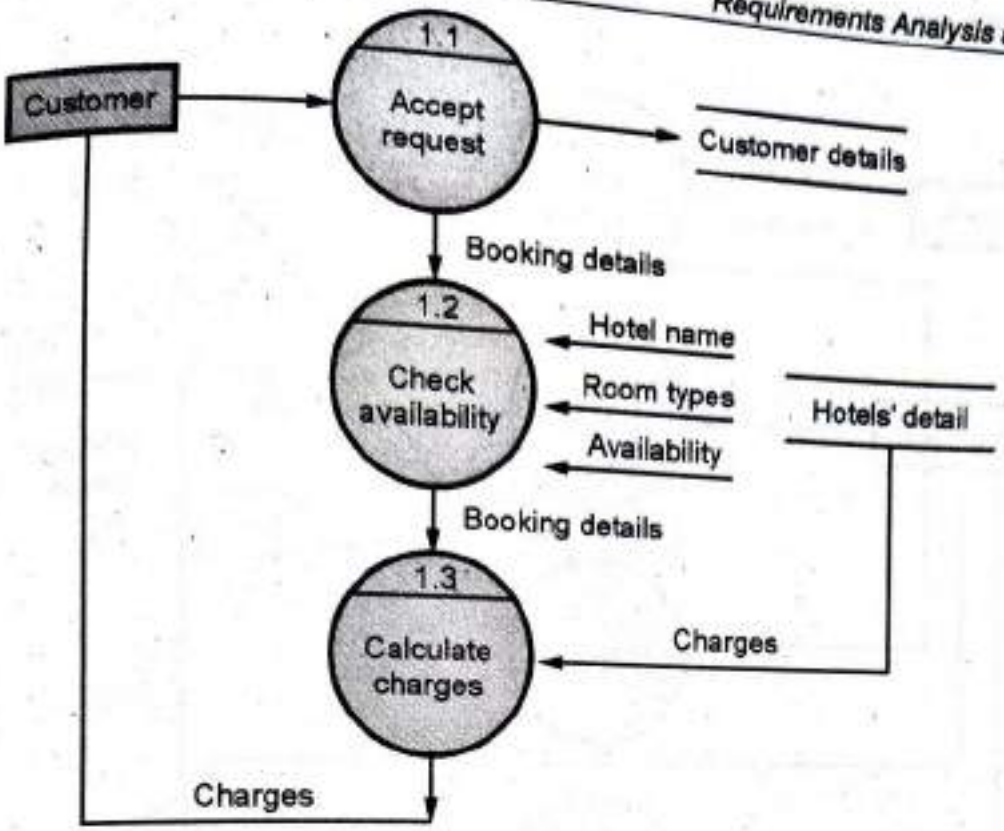


Fig. 2.17.18 Level 2 DFD

Example 2.17.5 *Tamil Nadu Electricity Board (TNEB) would like to automate its billing process. Customers apply for a connection-(domestic/commercial). EB staff take readings and update the system each customer is required to pay charges bi-monthly according to the rates set for the type of connection. Customers can choose to pay either by cash/card. A bill is generated on payment. Monthly reports are provided to the EB Manager.*

i) Give a name for the system. (Mark 1)
ii) Draw the Level - 0 DFD (Context Flow Diagram) (Marks 5)
iii) Draw the Level - 1 DFD. (Marks 10)

AU : Dec.-13. Marks 16

Solution : i) Electricity Bill Management System

ii)

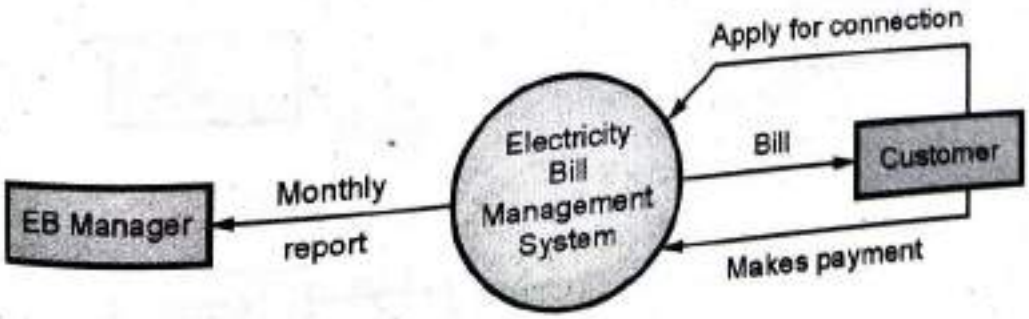


Fig. 2.17.19 Level 0 DFD

iii)

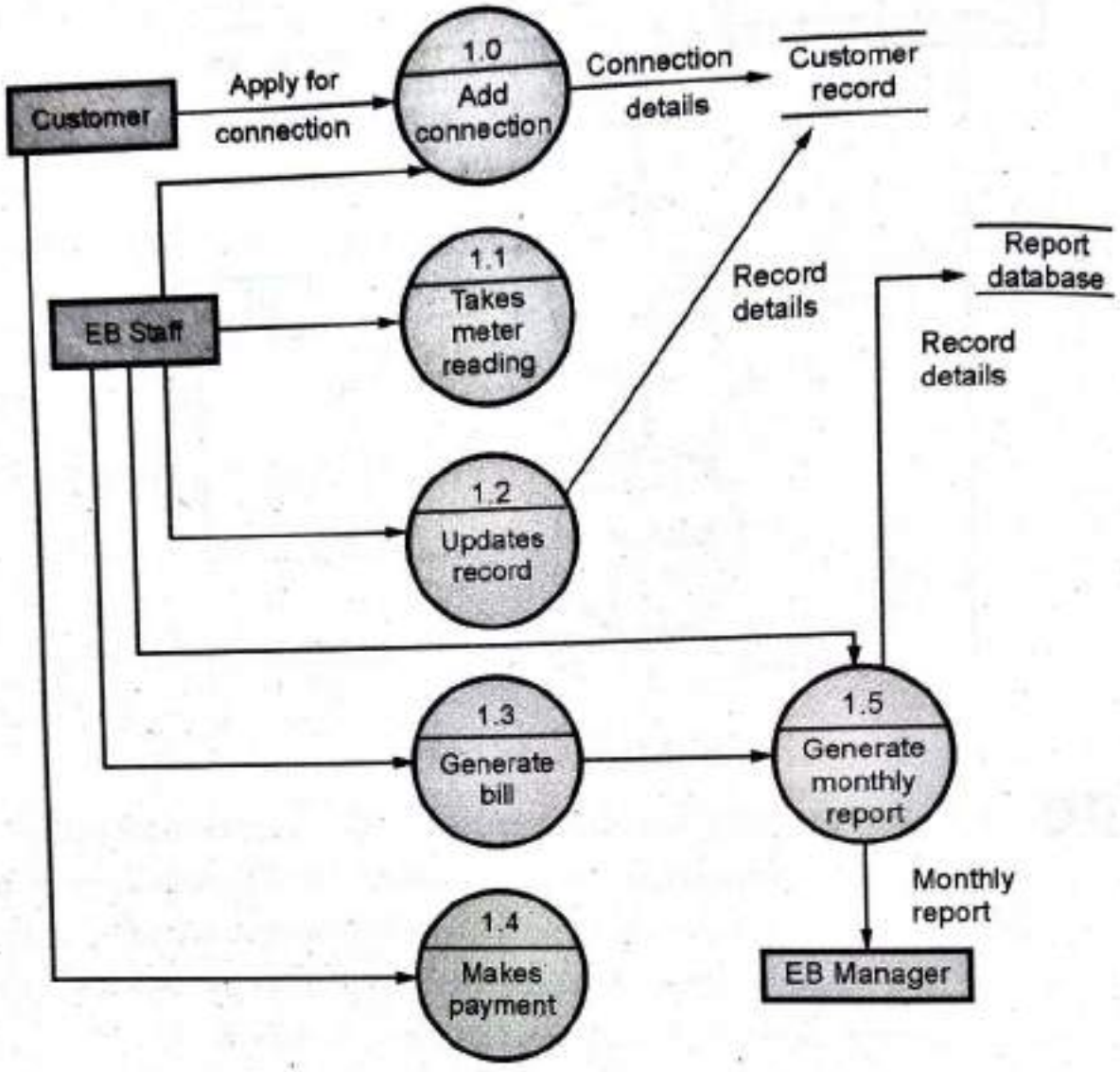


Fig. 2.17.20 Level 1 DFD

Example 2.17.6 *Develop a dataflow diagram for burglar alarm system along with entity relationship diagram.* **AU : May-10, Marks 8**

Solution : 1) DFD :

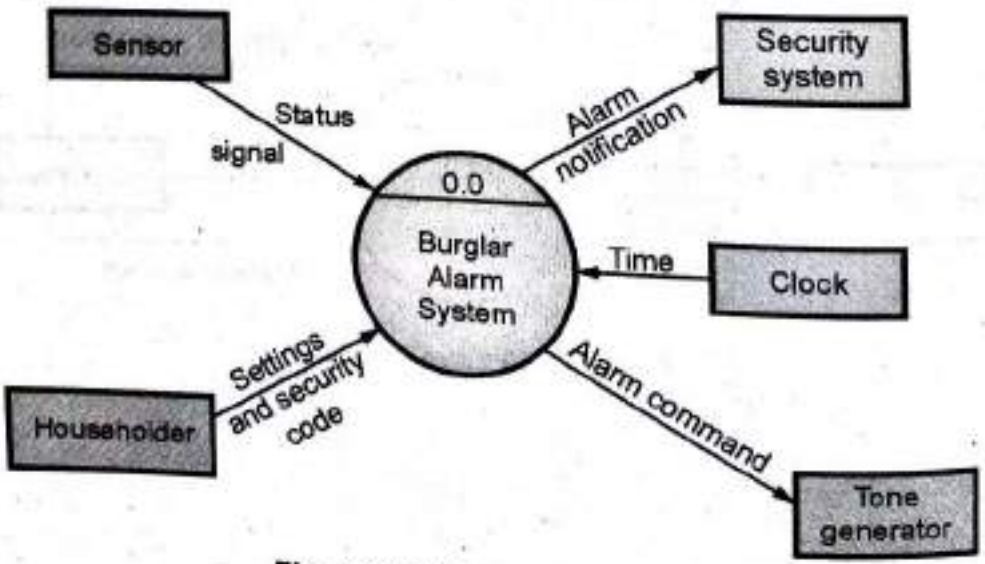


Fig. 2.17.21 Context diagram

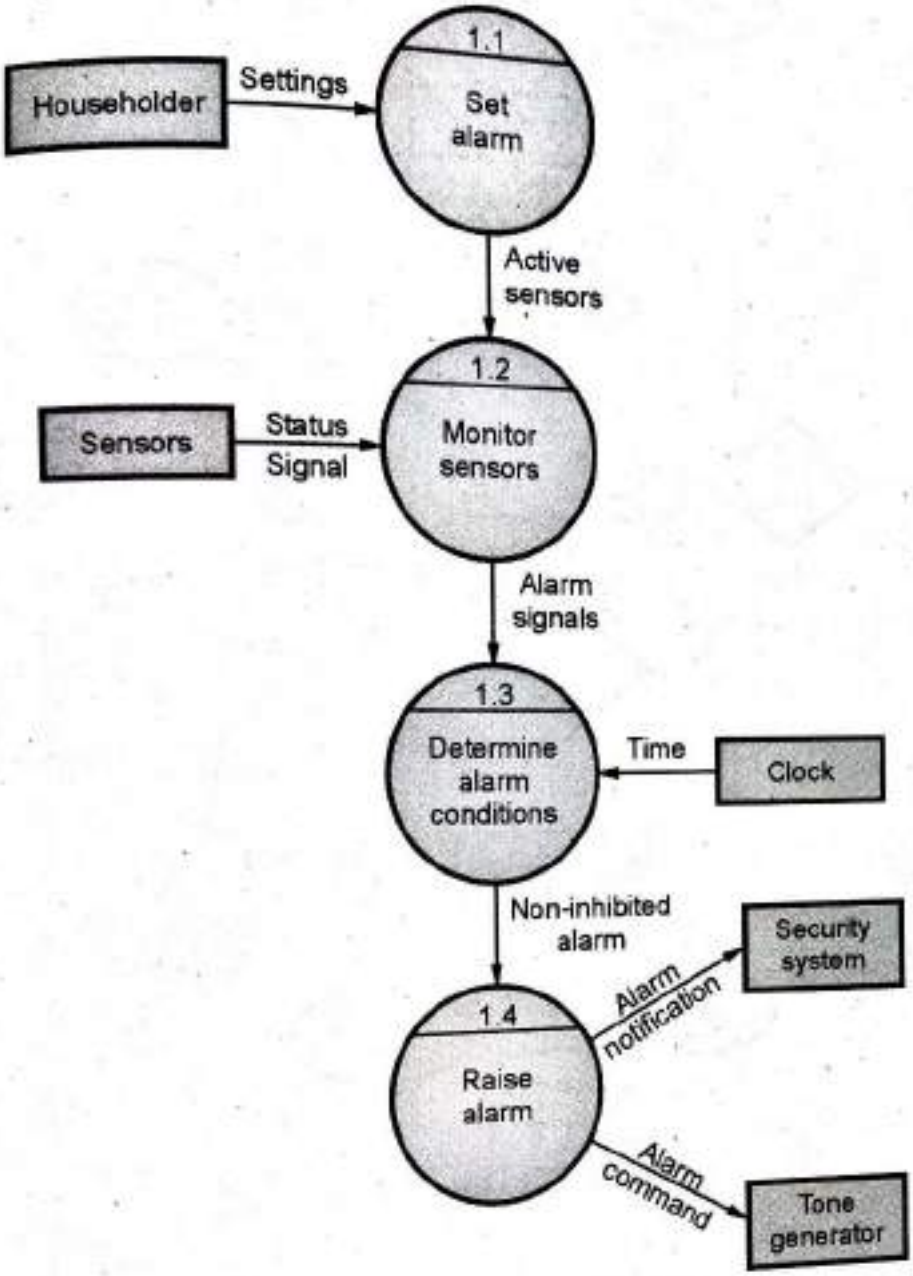


Fig. 2.17.22 Level 1 DFD

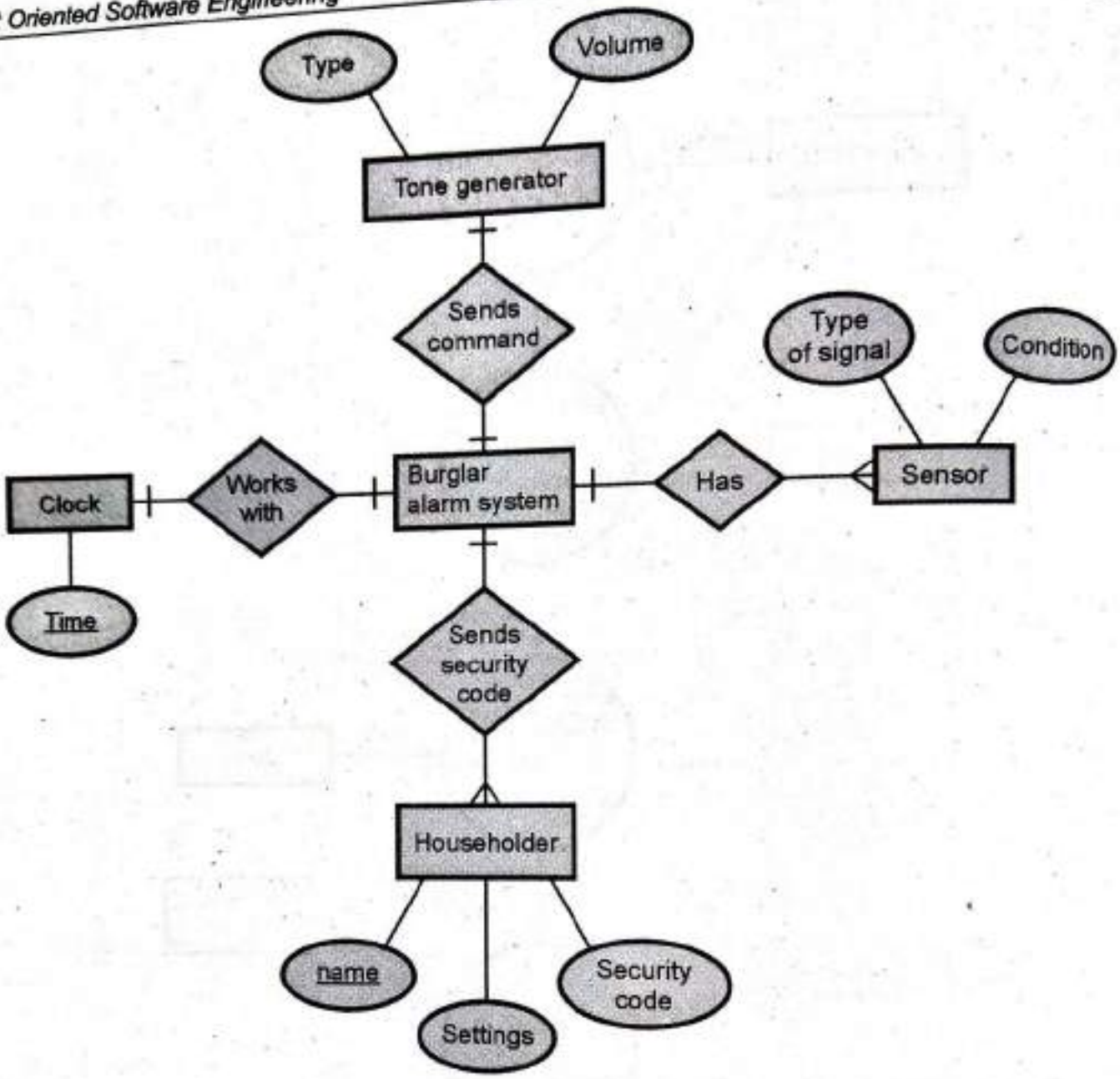


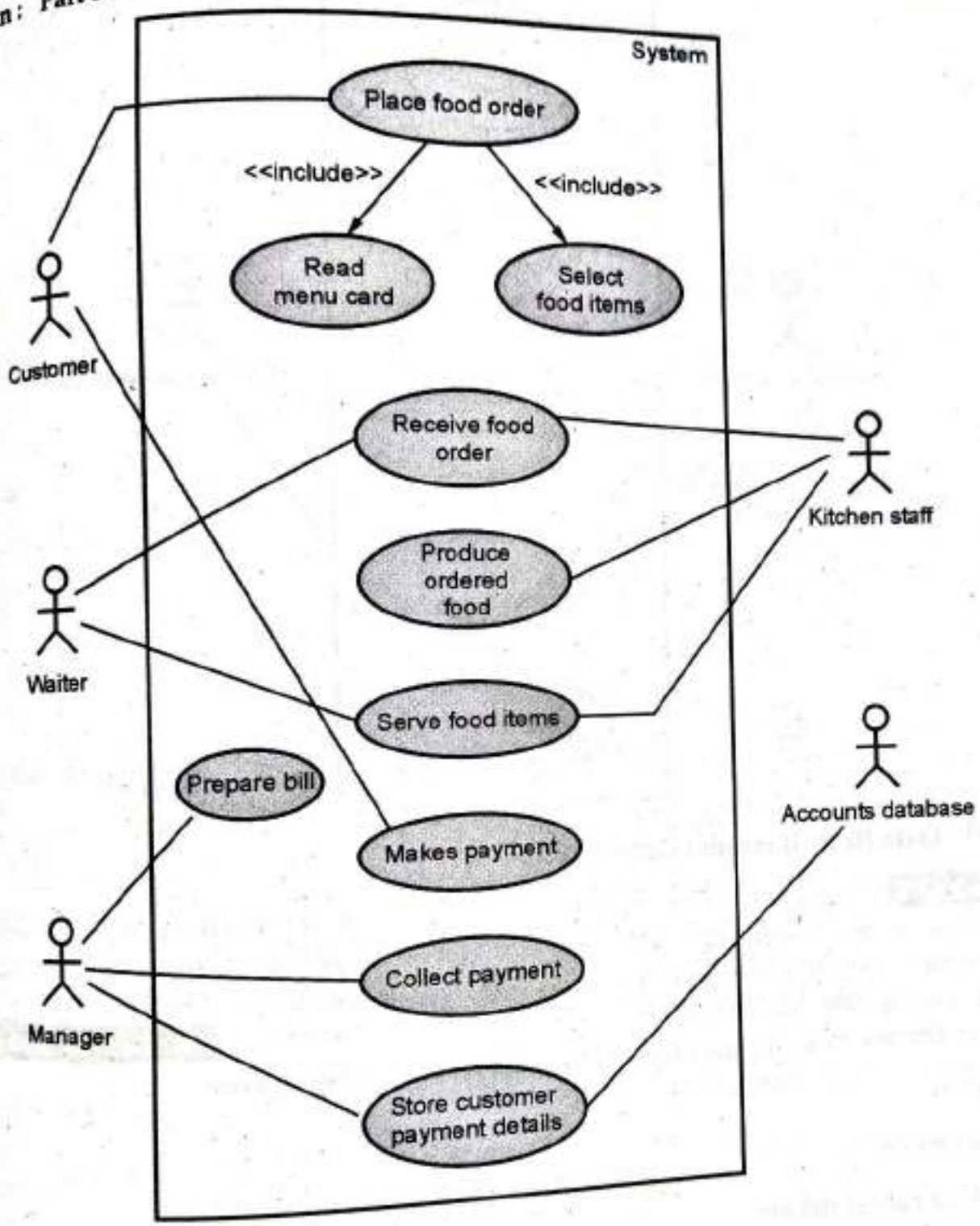
Fig. 2.17.23 ER diagram

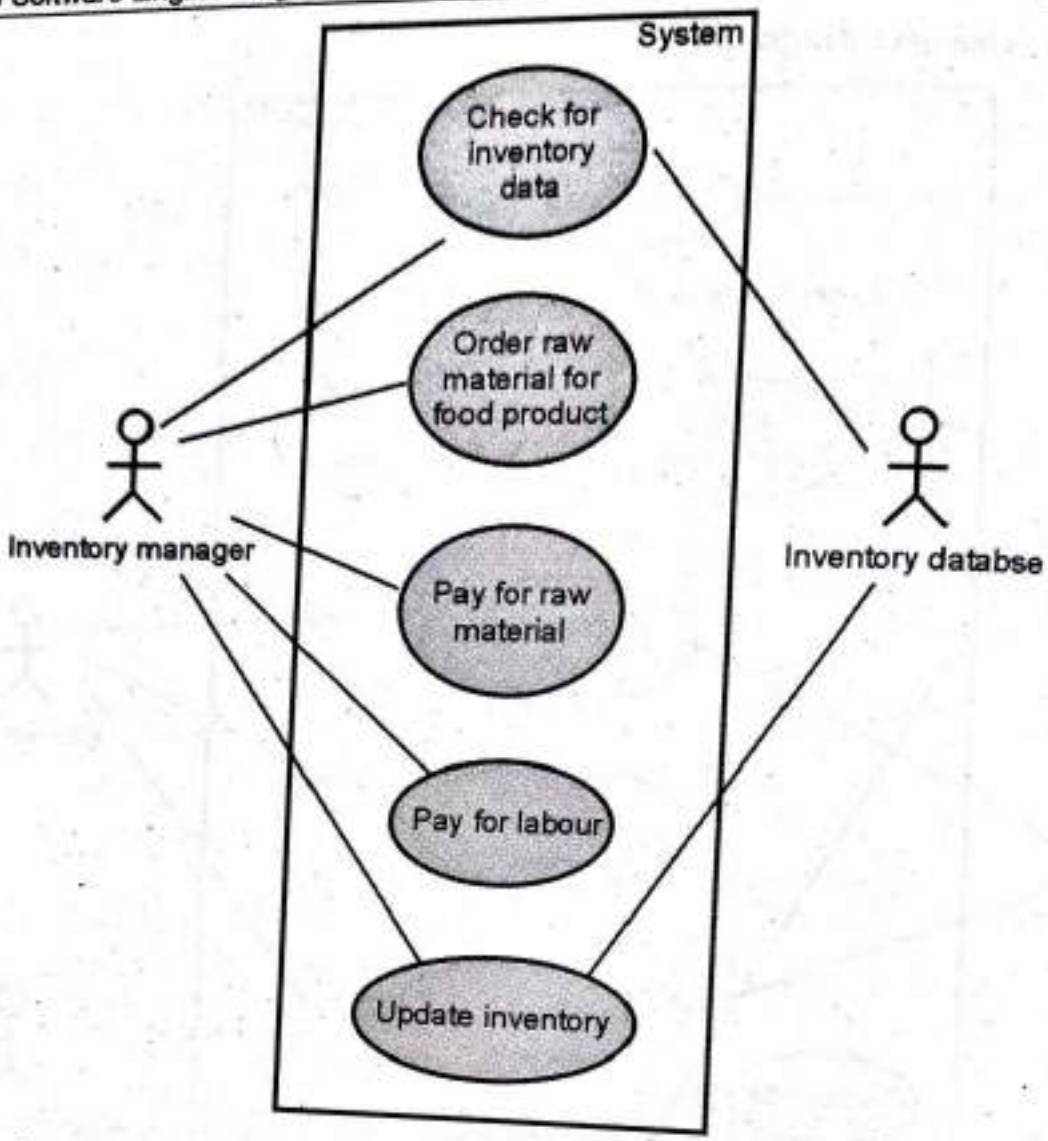
Example 2.17.7 *Draw the use case and data flow diagrams for a restaurant system. The activities of restaurant system are listed below -*

Receive the customer food orders, produce the customer ordered foods, serve the customer with their ordered foods, collect payment from customers, store customer payment details, order raw materials for food products, pay for raw materials and pay for labour.

AU : May-15, Marks 16

Solution : **Part I : Use case diagram :**





Part II : Data flow diagram : Refer example 2.17.2.

Example 2.17.8 *Consider an online railway reservation system, which allows the user to select route, book/cancel tickets using net banking/credit/Debit cards. The site also maintains the history of the passengers. For the above system, list and draw the use case scenario and model the above specification using data flow diagram.* **AU : Dec.-15, Marks 16**

Solution : The use cases for the given railway reservation system are -

1. Logs in
2. Select route
3. Book or cancel tickets
4. View status
5. Logs out

The use case diagram is as follows -

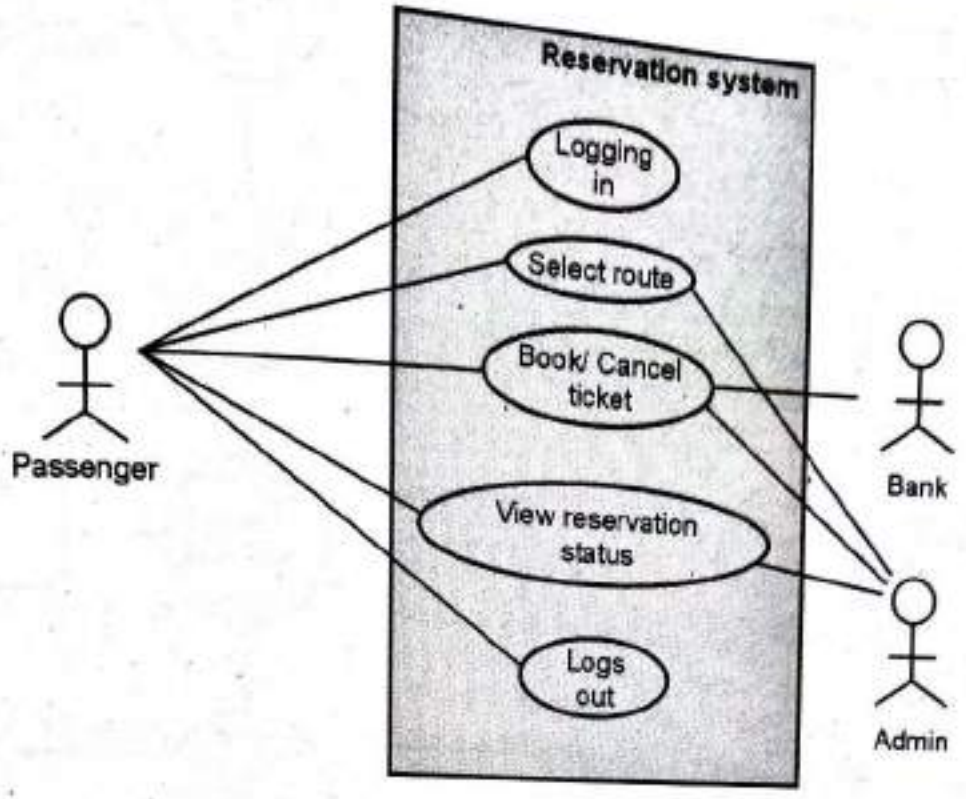


Fig. 2.17.24 Use case diagram

The level 0 and level 1 DFD can be as shown below –

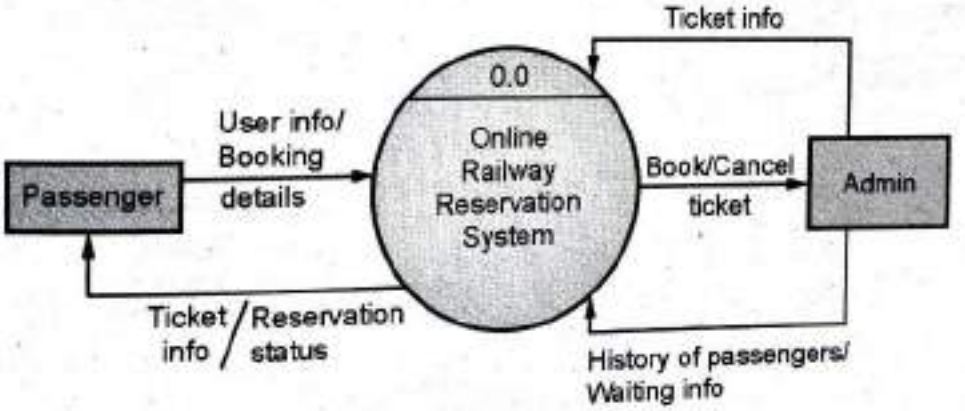


Fig. 2.17.25 Level 0 DFD

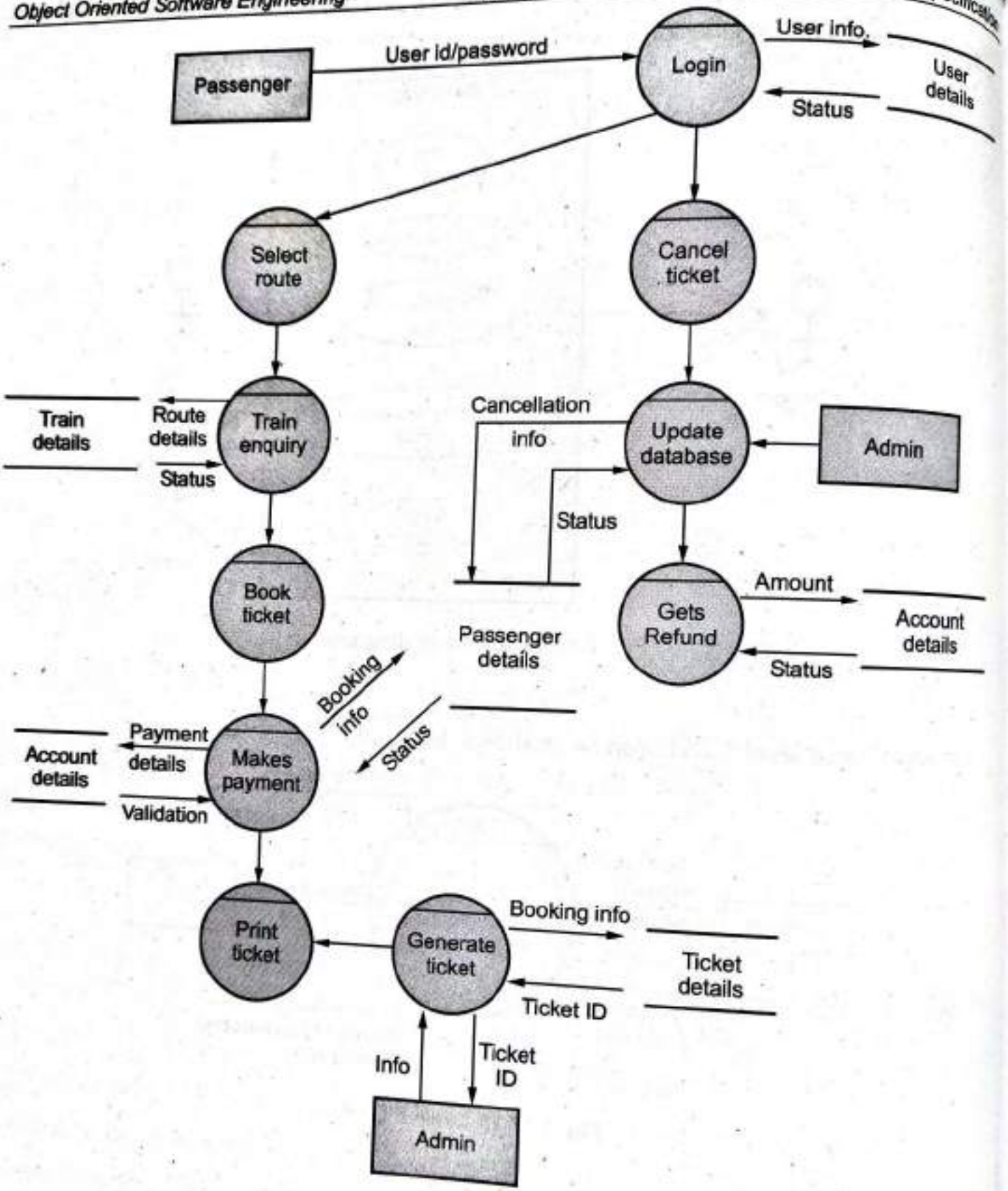


Fig. 2.17.26 Level 1 DFD

Example 2.17.9 *Consider an online book stores. It accepts individual/bulk orders, process payments, triggers delivery of the books. Some of the major features of the system include :*

- *Order books*
- *User friendly online shopping cart function*
- *Create, view, modify and delete books to be sold*
- *To store inventory and sales information in database*
- *To provide an efficient inventory system*
- *Register for book payment options*
- *Request book delivery*
- *Add a wish list*
- *Place request for books not available*
- *To be able to print invoices to members and print a set of summary reports*
- *Internet access.*

Analyse the system using the context diagram and level 1 DFD for the system. Explain the components of DFD.

AU : May-17, Marks 15

Solution :

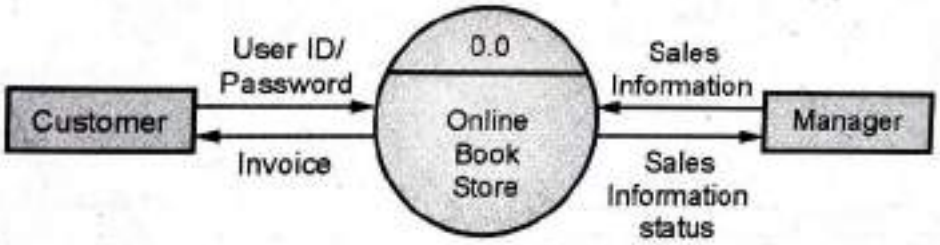


Fig. 2.17.27 Context level DFD

Refer Fig. 2.17.28 on next page.

Example 2.17.10 *Consider the process of ordering a pizza over the phone. Draw the use case diagram and also sketch the activity diagram representing each step of the process, from the moment you pick up the phone to the point where you start eating pizza. Include activities that others need to perform. Add exception handling to the activity diagram you developed. Consider at least two exceptions (e.g. delivery person wrote down wrong address, deliver person brings wrong pizza).*

AU : Dec.-17, Marks 13

Solution : Use case diagram

Refer Fig. 2.17.29 on page 2 - 165.

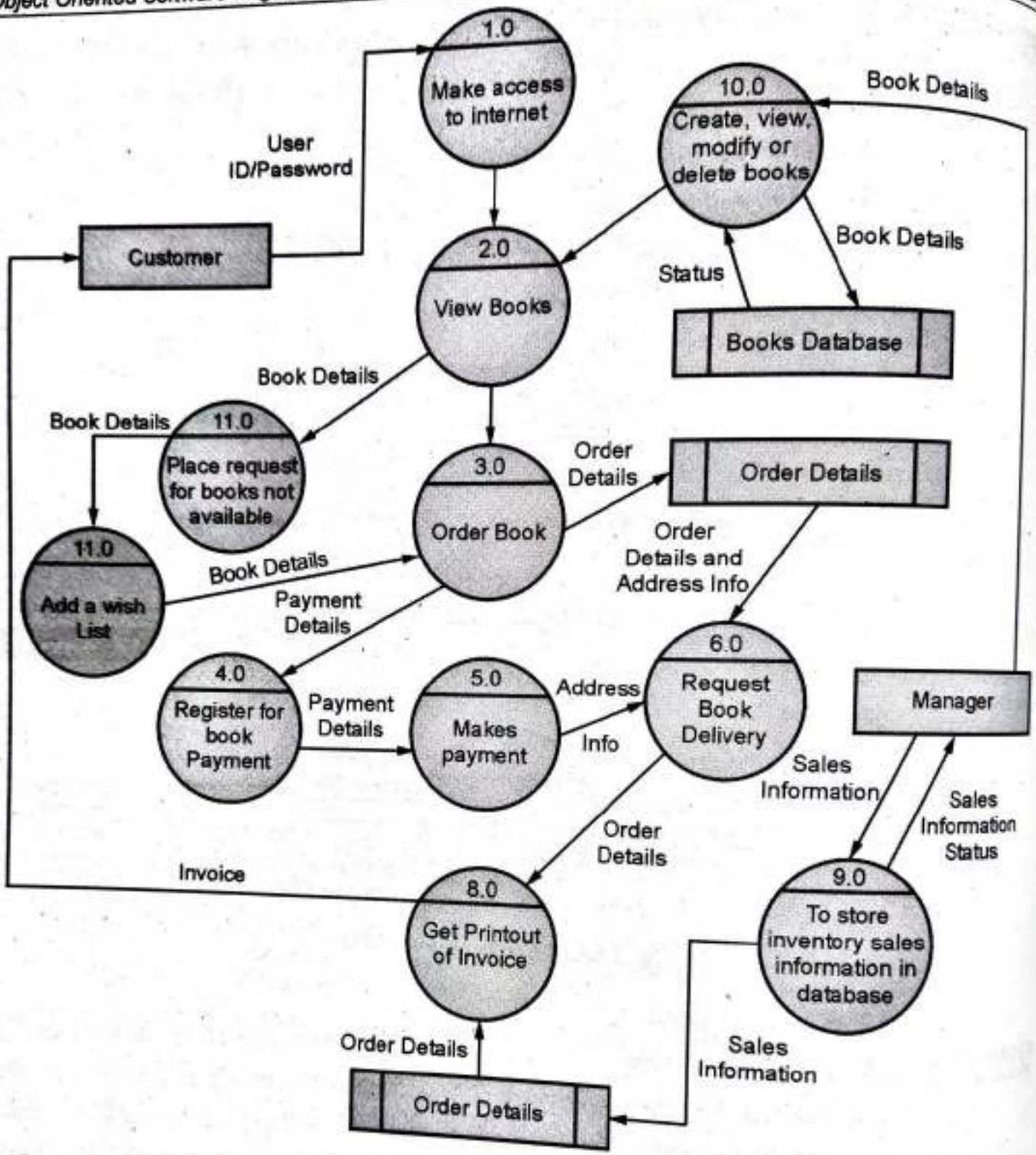


Fig. 2.17.28 Level 1 DFD

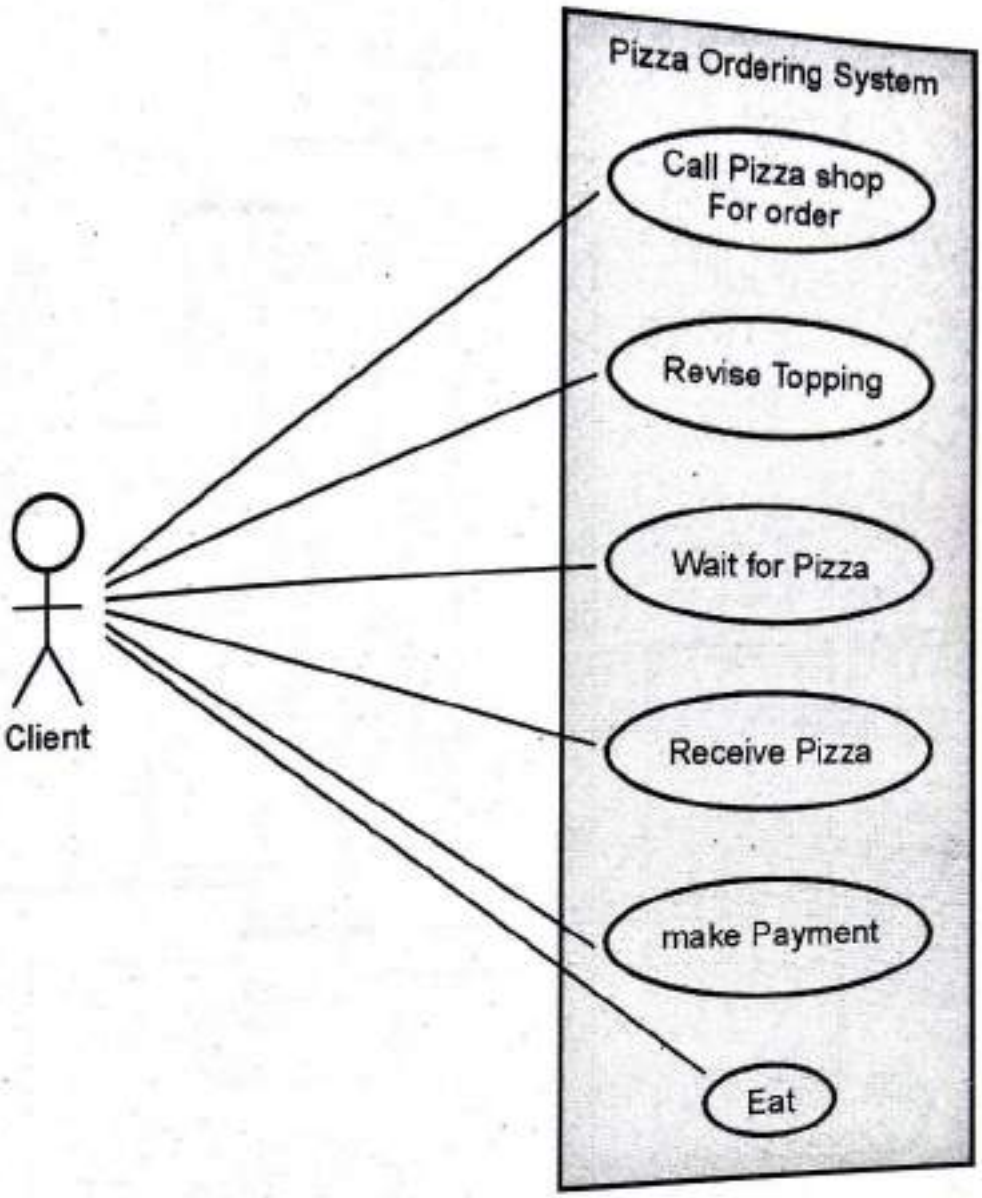


Fig. 2.17.29 Use case diagram for pizza ordering system

Activity diagram

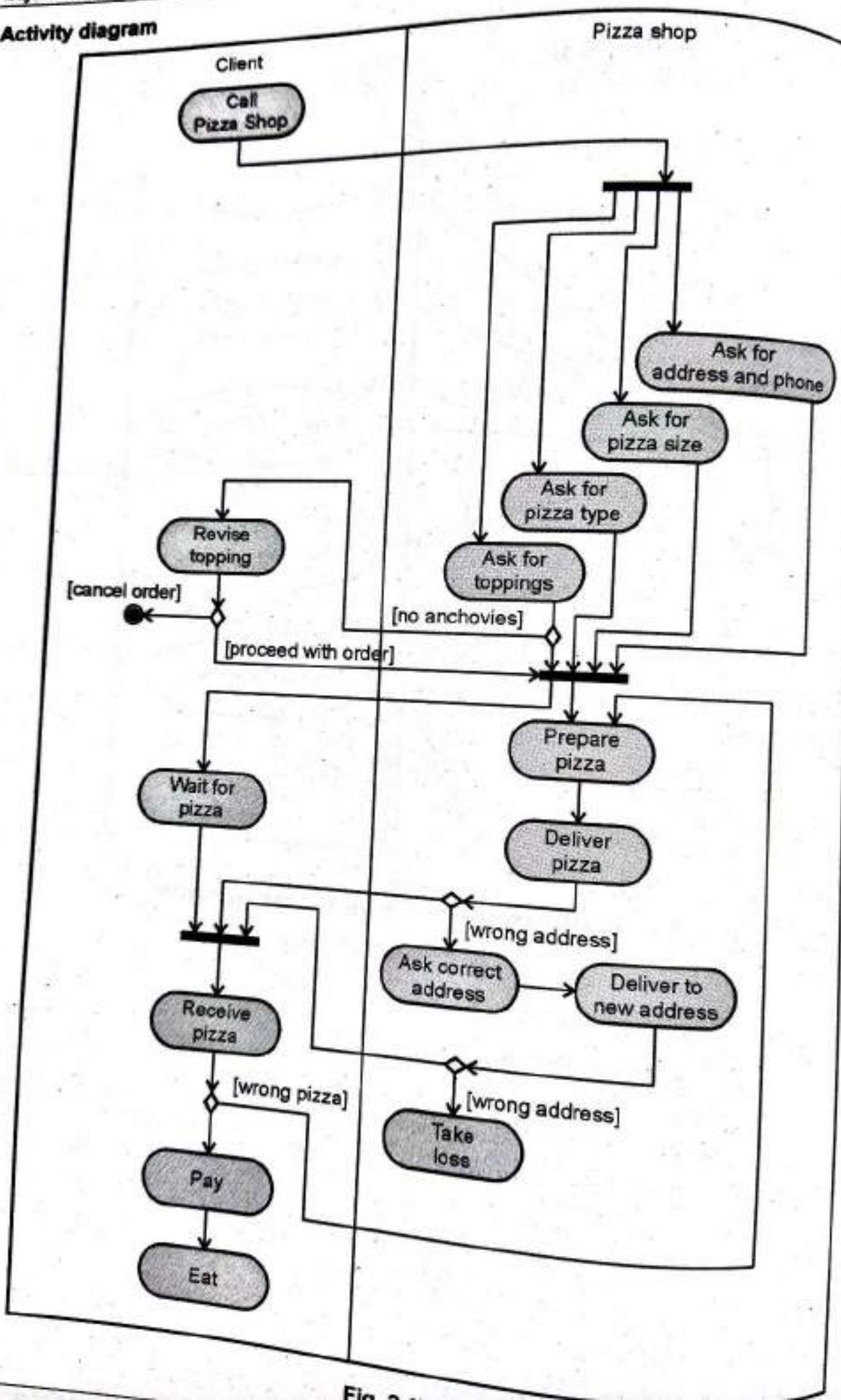


Fig.

Example 2.17.11 *What is the purpose of DFD ? What are the components of DFD ? Construct DFD for the following system : An on-line shopping system for XYZ provides many services and benefits to its members and staffs. Currently, XYZ staffs manually handle the purchasing information with the use of basic office software, such as Microsoft Offices Word and Excel. It may results in having mistakes easily and the process is very inconvenient. XYZ needs an online shopping system has five key features :*

i) to provide the user friendly online shopping cart function to members to replace hardcopy ordering form ;

ii) to store inventory and sales information in database to reduce the human mistakes, increase accuracy and enhance the flexibility of information processing ;

iii) to provide an efficient inventory system which can help the XYZ staffs to gain enough information to update the inventory ;

iv) to be able to print invoices to members and print a set of summary reports for XYZ's internal usage ;

v) to design the system that is easy to maintain and upgrade. **AU : May-18, Marks 15**

Solution : Level 0 DFD

Fig. 2.17.31

Level 1 DFD

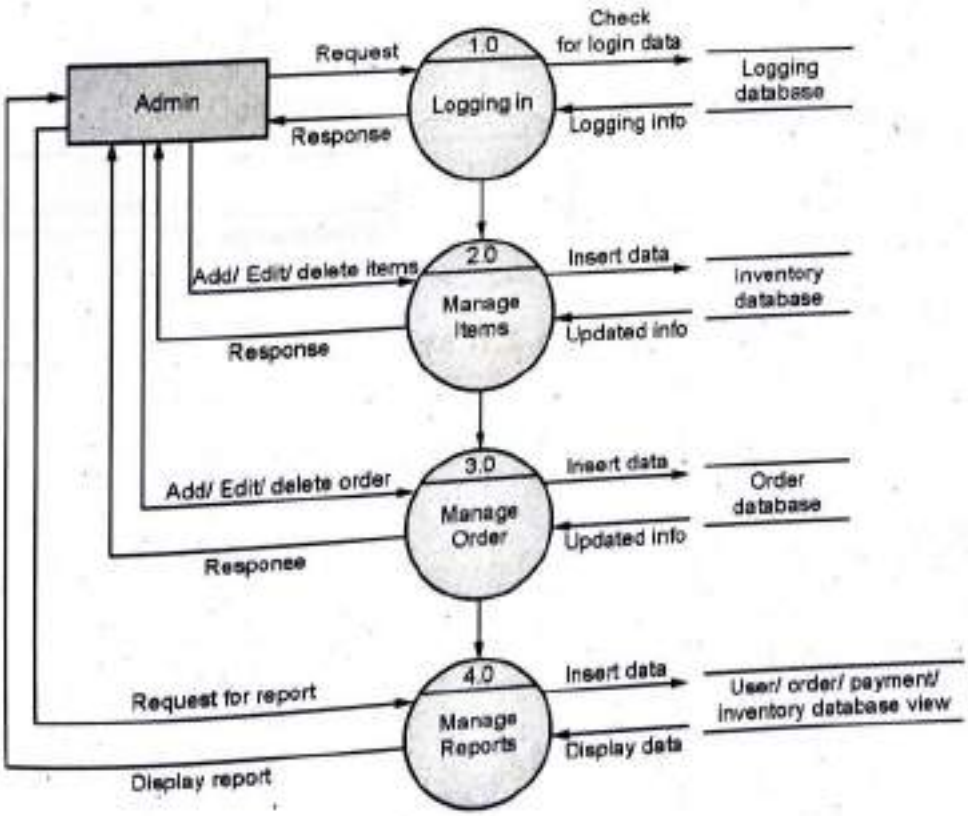


Fig. 2.17.32

Level 2 DFD : For manage order

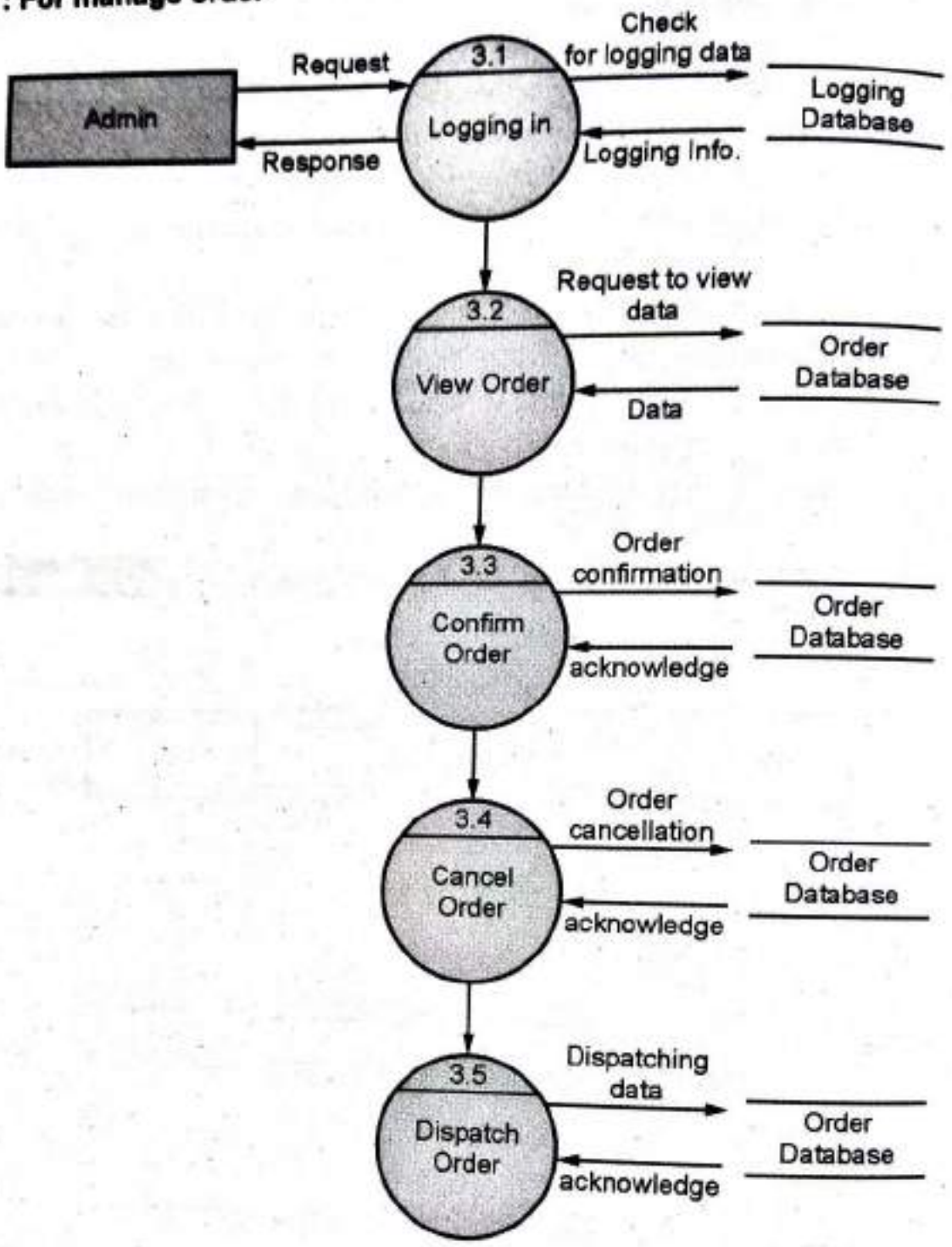


Fig. 2.17.33

Level 2 DFD : For manage items

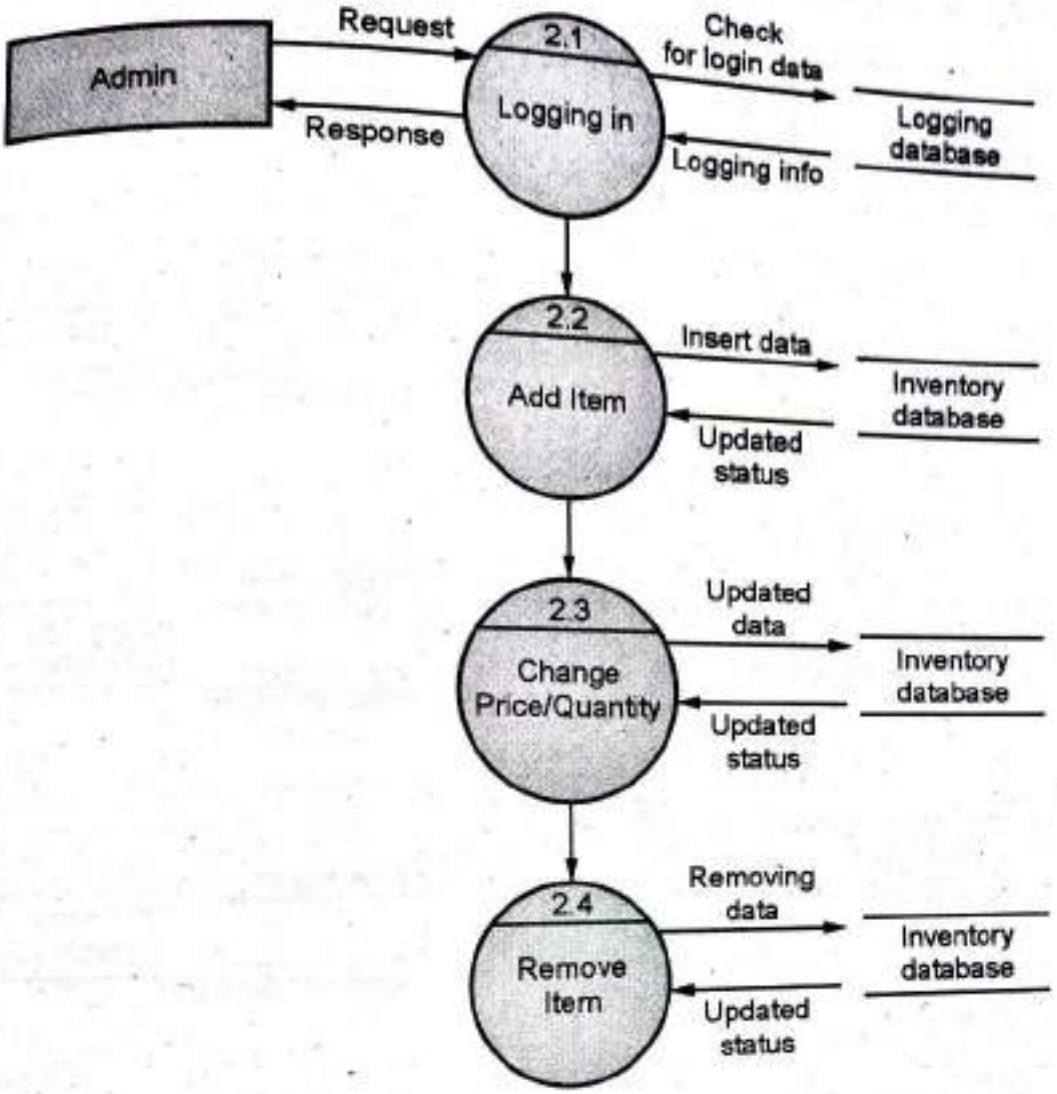


Fig. 2.17.34

Level 1 DFD : From user's perspective

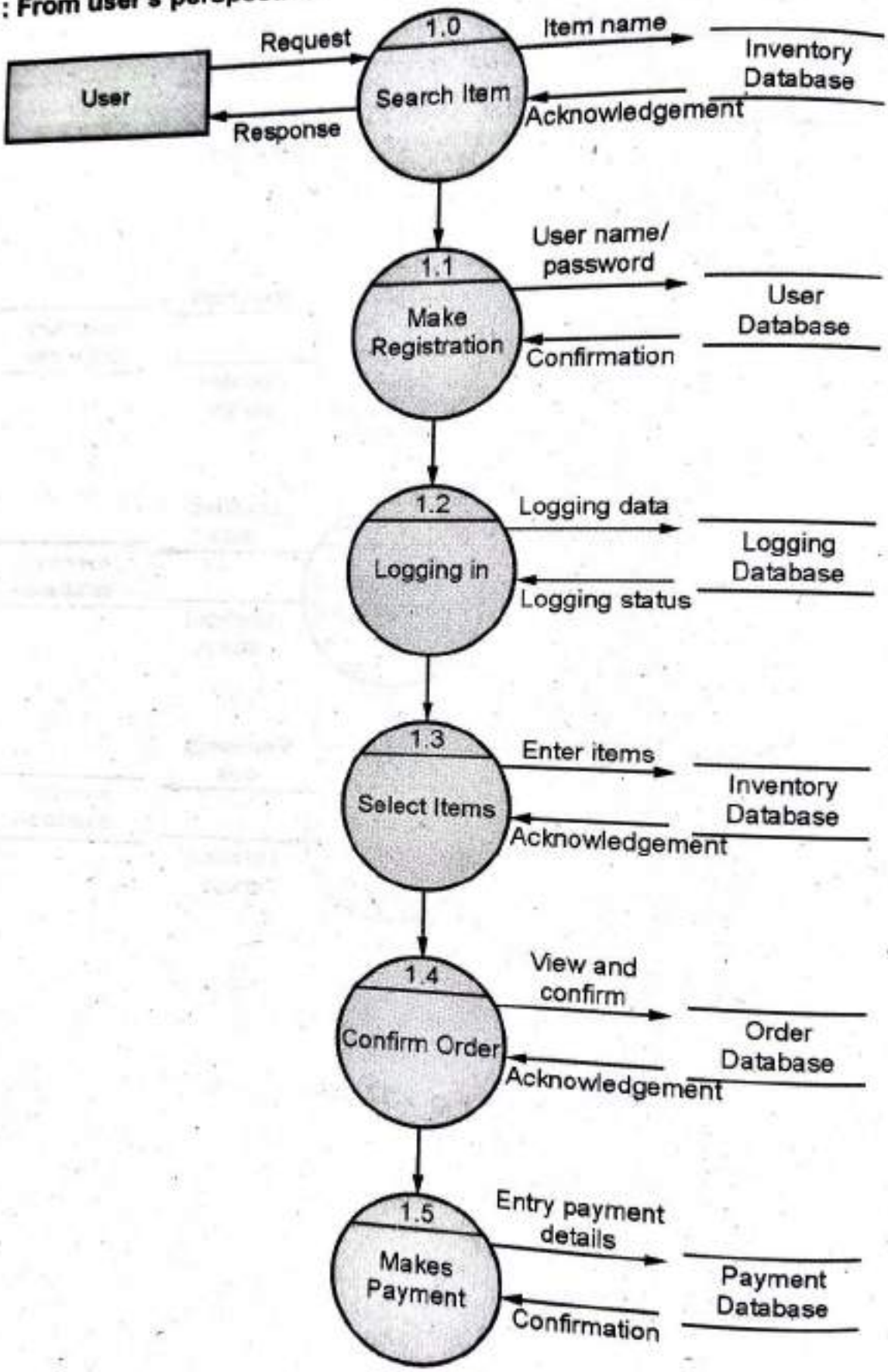


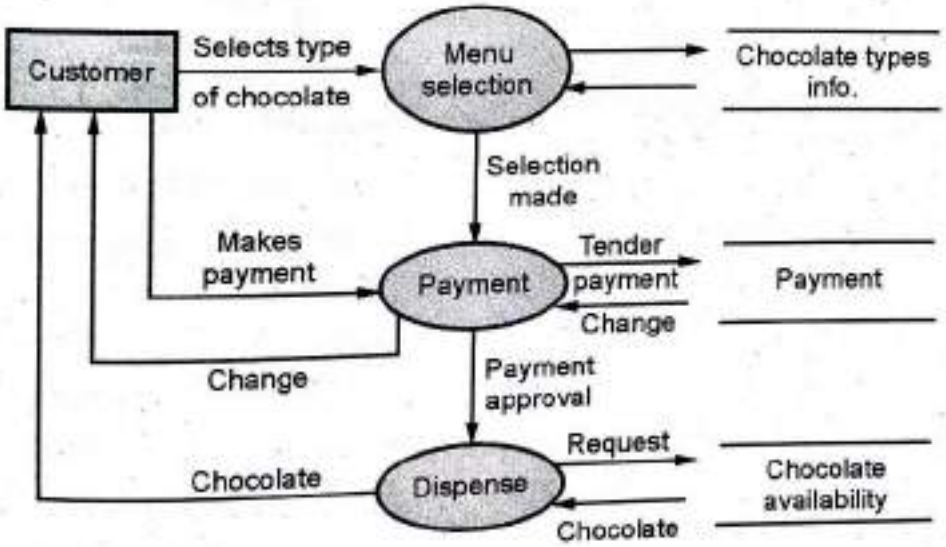
Fig. 2.17.35

Example 2.17.12 *i) Draw the Level 0 and Level 1 data flow diagram for the following system. ii) Identify entities in the system and draw a diagram showing the relationship between entities.*

The Chocolate Vending Machine (CVM) system requirements are as follows : The CVM dipenses chocolates : 1) Very large Chocolates (VC) at ₹15, 2) Large Chocolates (LC) at ₹10 and 3) a Small Chocolates (SC) at ₹5. The vending machine only. deals in coins. The CVM gives the proper change after the product selection is made. The CVM must check the amount deposited by the customer. The vending machine operates in the following Way. (A) The CVM remains idle until a customer or owner begins to interact with the machine. When a selection button is pressed the VCM indicates the required amount (₹15/-, ₹10/-, ₹5). (B) If the full amount needed has been deposited then dispense the proper chocolate and display : Thank You! (C) If an insufficient amount (possibly zero) has been deposited then display : remaining amount needed. (D) If an over amount has been deposited them dispense the proper candy and change arid display : Thank You! **AU : May-19, Marks 8 + 7**

Solution :

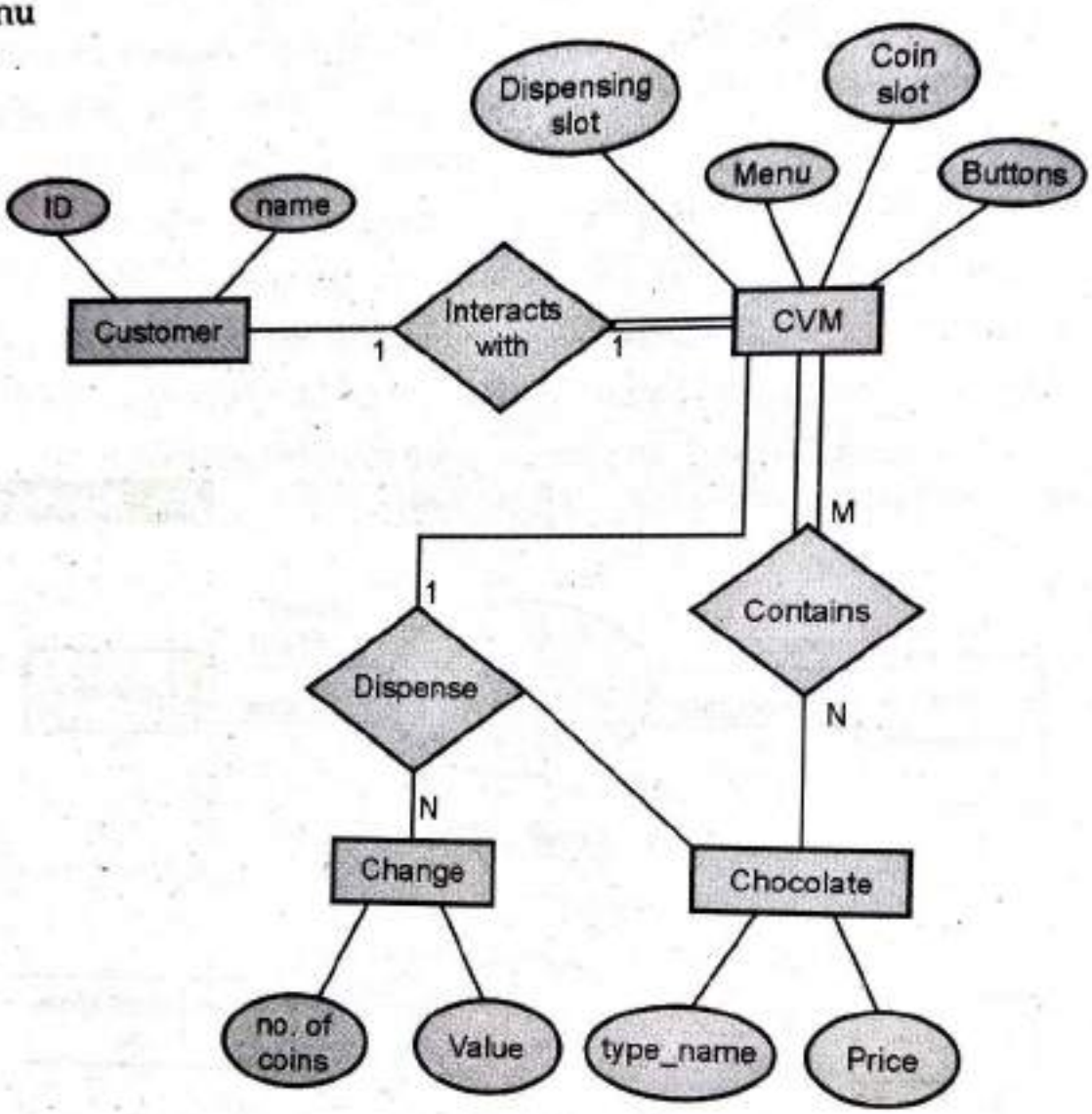
Level 0 DFD



Level 1 DFD

Entities in the system are :

1) Customer
2) CVM
3) Chocolate
4) Menu



2.17.3 Difference between Structured Analysis and Object Oriented Analysis

Structured analysis	Object oriented analysis
In structured analysis the process and data are separately treated.	In object oriented analysis the process(methods) and data are encapsulated in the form of object.
Various models that are getting developed in structured analysis are DFD, ER diagram and Data dictionary.	Various models that are getting developed in object oriented analysis are class diagram, use case diagram, activity diagram and deployment diagram.
Simpler to design.	Modular systems can be developed using this kind of analysis.
It is not suitable for large and complex projects.	For large and complex system object oriented approach is more suitable.

Review Question

1. *Describe primary differences between structured analysis and object oriented analysis.*

AU : May 08, Marks 6

2.18 Two Marks Questions with Answers

Q.1 Write distinct steps in requirement engineering process. AU : IT, May-06

Ans. : The distinct steps in requirement engineering process are

1. Requirements elicitation.
2. Requirements analysis.
3. Requirements validation.
4. Requirements management.

Q.2 Why SRS must be traceable? What is traceability requirement ? AU : IT, Dec.-05

Ans. : There may be chances of having new requirements during development of the software. Traceability is concerned with relationship between requirements their sources and the system design. Thus change in requirement is manageable if SRS is traceable.

The traceability requirement is the relationship between dependent requirements.

Q.3 What are non-functional requirements for a software ? AU : IT, May-03, May-05, Dec.-19

Ans. : The non-functional requirements define system properties and constraints. Various properties of a system can be reliability, response time, storage requirements, and constraints of the system can be input and output device capability, system representations.

Q.4 What is the outcome of feasibility study ? AU : IT, Dec.-03

Ans. : The feasibility study decides whether the proposed system is worthwhile or not. The outcome of feasibility study is the results obtained from following questions :

1. Whether system contributes to organizational objectives ?
2. Whether the system can be engineered? Is it within the budget ?
3. Whether the system can be integrated with other existing system ?

After performing the feasibility study a feasibility study report is prepared.

Q.5 Differentiate data flow diagram and state transition diagram. AU : IT, Dec.-06

Ans. :

Data flow diagram	State transition diagram
Data flow diagram is a graphical representation for representing the information flow and the transforms that are applied as data move from input to output.	State transition diagram is a graphical representation for representing the behaviour of a system by depicting its states and the events that cause the system to change state.
The data flow diagram is a collection of process, data store, flow of data (transitions) and external entity.	The state transition diagram is a collection of states and events.
The data flow diagrams are used to represent the system at any level of abstraction, and the increasing levels are used to expose more and more functionalities in the system.	The state transition diagrams are typically drawn at single level. They are intended to expose the overall behaviour of the system.

Q.6 Why it is so difficult to gain a clear understanding of what customer wants ?

AU : CSE, Dec.-07

OR Write a note on what are the difficulties in elicitation, requirement elicitation.

AU : May-17, Marks 5

OR Requirements analysis is unquestionably the most communication intensive step in the software engineering process. Why does the communication path frequently breaks down ?

AU : May-16, Marks 8

Ans. : Following are the reasons for : why it is difficult to understand customer wants -

1. Customer sometimes is unable to specify the **scope of the project.** Sometimes customers specify too many technical details and this may increase the confusion.
2. There is difficulty in **understanding the problem.** Sometimes customer could not decide what are their needs and wants. Sometimes they have got poor understanding of capabilities and limitations the existing computing environment.

   Sometimes customers find it difficult to communicate with the system engineer about their needs. Sometimes customers may have got some conflicting

requirements. This ultimately results in specifying ambiguous requirements.

3. As project progresses the needs or requirements of the customers changes. This creates a problem of volatility.

Q.7 Identify ambiguities and omissions in the functional requirements. What questions would you ask to clarify these functional requirements ? AU : IT, May-08

Ans. : Following are the questions that must be asked to clarify the functional requirements -

1. Is the requirement specification clear ? Can it be misinterpreted ?
2. What are the sources of requirements ? Is the final requirement specification as per the original source ?
3. Is there any requirement violation for domain constructs ?
4. What are the other related requirements of the current requirement ? And are they cross verified ?
5. For tracing the requirement is there any system model ?
6. Is the requirement testable ?
7. Has an index specification created for the requirements ?
8. Is the requirement satisfying the system objective.

Q.8 List out requirements engineering. AU : CSE, May-09

Ans. : Various activities in requirement engineering are -

1. Requirements elicitation 2. Requirements analysis
3. Requirements validation 4. Requirements management

Q.9 Define functional and non functional requirements. (Refer sections 2.2.1 and 2.2.2) AU : Dec.-10, May-14

Q.10 Give two examples of non functional requirements. AU : Dec.-14

Ans. : Consider the Library Management System for which the two non functional requirements can be -

1. The user who wishes to read the article on-line must be authenticated first.
2. The article must be displayed within 5 seconds.

Q.11 What is the need for feasibility analysis ? AU : May-15

Ans. : The feasibility study is required to decide whether or not the proposed system is worthwhile to implement. AU : Dec.-15

Q.12 Define feasibility study and list the types.

Ans. : Definition : A feasibility study is a study made to decide whether or not the proposed system is worthwhile.

Types : 1) Technical feasibility 2) Economic feasibility
3) Schedule feasibility 4) Operational feasibility

Q.13 List the characteristics of good SRS. AU : May-16, Dec.-20

Ans. : 1) SRS must be **correct.** 2) SRS must be **unambiguous.**
3) SRS must be **complete.** 4) SRS must be **consistent.**
5) SRS must be **traceable.**

Q.14 Classify the following as functional / non functional requirements for banking system.

a) Verifying back balance

b) Withdrawing money from bank

c) Completion of transactions in less than one second.

d) Extending the system by providing more tellers for customers. AU : Dec.-16

Ans. : a) Functional requirement b) Functional requirement
c) Non-functional requirement d) Non-functional requirement.

Q.15 What is the purpose of petrinet ? AU : May-17

OR Define a petrinet. AU : Dec.-15

Ans. : Petrinets are rigorously used to define the system. There are three types of components in petrinets. These are

i) Places : Represent possible states of the system.

ii) Transitions : Causes the change in the states.

iii) Arc : Connects a place with transition or transition with places.

Q.16 Differentiate between normal and exciting requirements. AU : May-17

Ans. : Nornal requirements : The requirements as per goals and objectives of the system are called normal requirements. For example - Handling mouse and keyboard events for any GUI based system.

Exciting requirements : When certain requirements are satisfied by the software beyond customer's expectations then such requirements are called exciting requirement. For example - In word processing software system if there are some exceptional page layout facilities then it falls in exciting requirements category.

Q.17 Draw a use case diagram for an online shopping which should provide provisions for registering, authenticating the customers and also for online payment through any payment gateway like Paypal. AU : Dec.-17

Ans. :

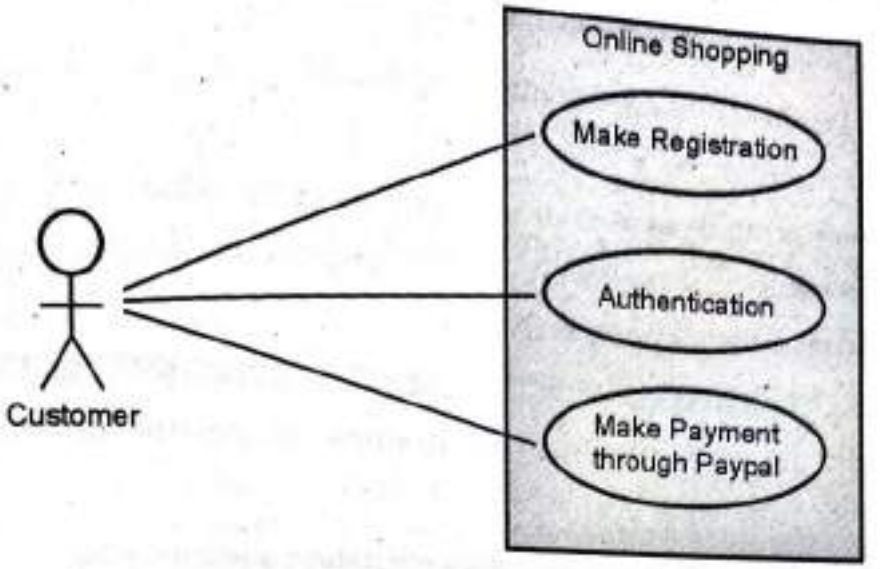


Fig. 2.18.1

Q.18 Define Quality Function Development (QDF). AU : Dec.-17

Ans. : Quality function deployment is a quality management technique which translates the customer needs and wants into technical requirements. This technique was introduced in Japan.

Q.19 Draw the context flow graph of a ATM automation system. AU : Dec.-17

Ans. :

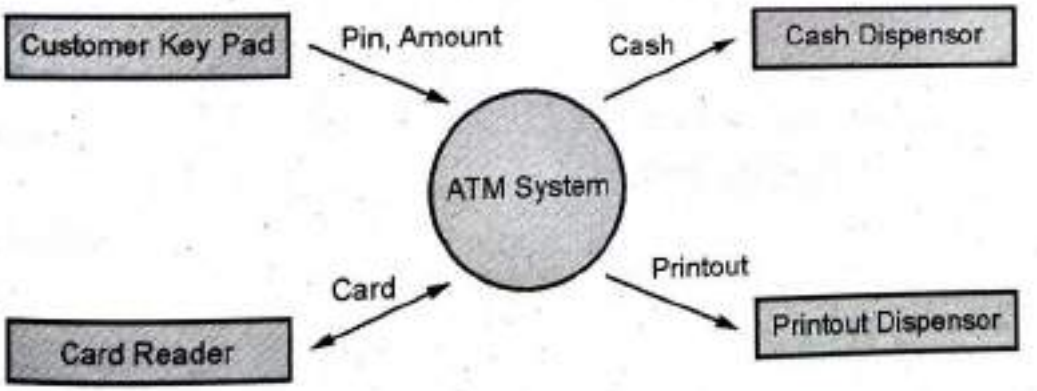


Fig. 2.18.2

Q.20 State two advantages of using Petri Nets AU : May-19

Ans. :

1) It specifies the precise definition of problem

2) It represents the graphical workflow.

Q.21 How does data flow diagram help in design of software system ? AU : May-19

Ans. : The data flow models are used to model the information and function domain. Refinement of DFD into greater levels help the analyst to perform functional decomposition.

Q.22 Differentiate : Functional and Non-functional requirements. (Refer section 2.2.3)

AU : May-19

Q.23 Name any two requirements of elicitation techniques.

AU : Dec.-20

Ans. :

1) **Use cases :** Developing use cases or user stories to describe specific interactions and scenarios involving the software. This technique helps capture functional requirements from a user's perspective.
2) **Questioning :** Continuous asking questions to and seeking clarification from stakeholders during requirements discussions to ensure understanding of the requirements

Q.24 Mention the steps involved in requirements elicitation and analysis.

AU : Dec 22

Ans. : The steps are as follows -

1) Requirement discovery
2) Requirements classification and discovery
3) Requirements prioritisation
4) Requirement documentation

❑❑❑

UNIT III

3 Software Design

Syllabus

Software design - Design process - Design concepts - Coupling - Cohesion - Functional independence - Design patterns - Model-view-controller - Publish-subscribe - Adapter - Command - Strategy - Observer - Proxy - Facade - Architectural styles - Layered - Client Server - Tiered - Pipe and filter- User interface design - Case Study.

Contents

3.1 Software Design
3.2 Design Process ..... **May-05,** ..... Marks 8
3.3 Design Concepts ..... **May-06,08,13,16,19,** ..... **Dec.-05,17,22,** ..... Marks 16
3.4 Design Patterns ..... **Dec.-15, May-19,** ..... Marks 8
3.5 Model-View-Controller
3.6 Publisher-Subscriber Pattern
3.7 Adapter ..... **May-11,12,13,14,** ..... **Dec.-11,12,13,14,** ..... Marks 16
3.8 Command
3.9 Strategy
3.10 Observer
3.11 Proxy
3.12 Façade
3.13 Role of Architecture
3.14 Architectural Styles
3.15 Layered Architectural Style
3.16 Client Server Style
3.17 Tiered Architecture
3.18 Pipe and Filter ..... **Dec.-20, May-22,** ..... Marks 13
3.19 User Interface Design ..... **May-16,13,22,** ..... **Dec.-08,09,15,16,17,20,22,** Marks 16
3.20 Two Marks Questions with Answers

3.1 Software Design

Software design is model of software which translates the requirements into finished software product in which the details about software data structures, architecture, interfaces and components that are necessary to implement the system are given.

3.1.1 Designing within the Context of Software Engineering

- Software design is a vital activity during the process of software development. It is carried out irrespective of any process model used.
- During software engineering process, the first task is to identify and analyze the requirements. Based on this analysis the software design is developed. This design serves as a basis for code generation and testing. Hence software design is an intermediate process which translates the analysis model into design model.
- The four types of elements of **analysis model** are scenario based elements, class based elements, behavioral elements, and flow oriented elements.
- During the scenario based analysis various models such as use case diagrams, activity diagrams or swimlane diagrams are created. During the class based analysis the class diagrams are created. During flow oriented analysis the Data flow diagrams are created. Behavioral analysis can be done using state diagrams and sequence diagrams.
- The **class based elements** of analysis model are used to create **class diagrams**.
- The **flow oriented elements** and **class based elements** are used to create **architectural design.**

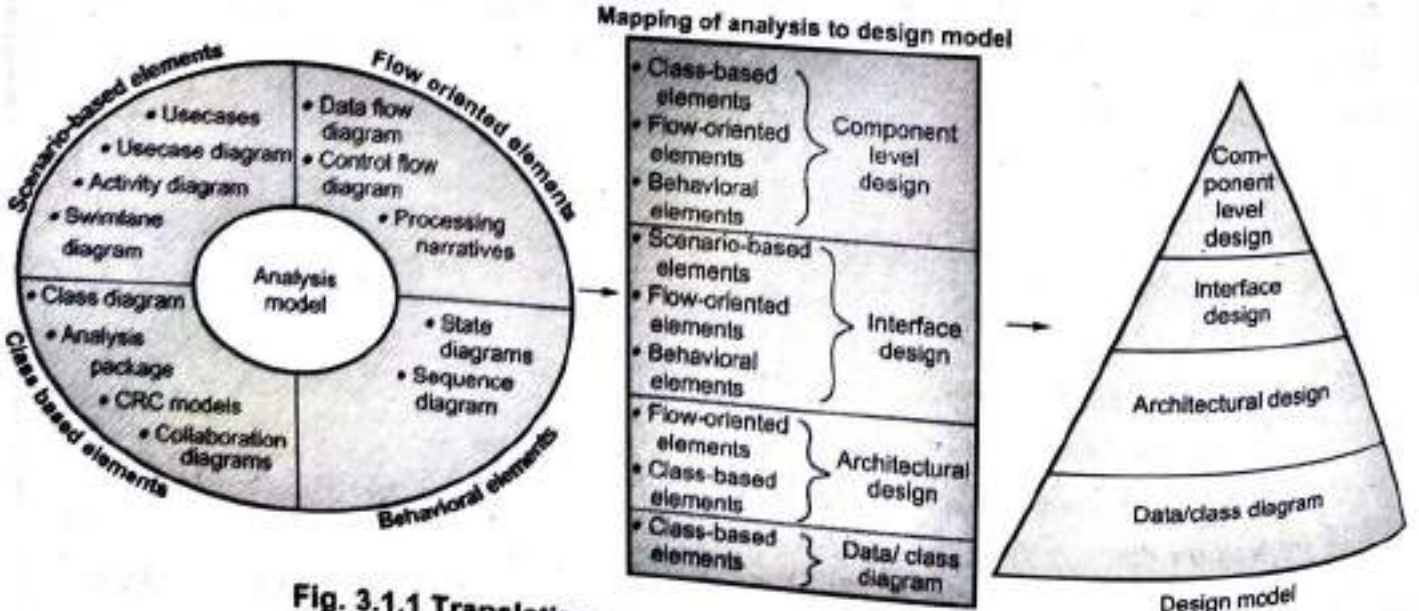


Fig. 3.1.1 Translating analysis model into the design model

- The **scenario based elements, flow oriented elements and behavioral elements** are used to create **Interface design**. The interface design describes how software communicates with systems.
- The **class based elements, flow oriented elements and behavioral elements** are used to create **component level design**. This design transforms the structural elements of software architecture into procedural description of software module. (Refer Fig. 3.1.1).
- Software design is very important for assessing the quality of software. Because design is the only way that we can accurately translate the user requirements into the finished software product.
- Without software design, it is difficult to test the software product. Not only this, but for making a small change in the software - it is the software design which helps us to make the necessary changes without disturbing other part of the software.

3.2 Design Process

AU : May-05, Marks 8

- Software design is an iterative process using which the user requirements can be translated into the software product.
- At the initial stage of the software is represented as an **abstract view**. During the sub-sequent iterations data, functional and behavioral requirements are traced in detail. That means at each iteration the refinement is made to obtain lower level details of the software product.

Characteristics of Good Design

1. The good design should **implement all the requirements** specified by the customer. Even if there are some implicit requirements of the software product then those requirements should also be implemented by the software design.
2. The design should be **simple** enough so that the software can be understood easily by the developers, testers and customers.
3. The design should be comprehensive. That means it should provide complete picture of the software.

Quality Guideline

The goal of any software design is to produce high quality software. In order to evaluate quality of software there should be some predefined rules or criteria that need to be used to assess the software product. Such criteria serve as characteristics for good design. The quality guidelines are as follows -

1. The design architecture should be created using following issues -
   - The design should be created using **architectural styles and patterns.**

- Each component of design should posses **good design characteristics.**
- The implementation of design should be **evolutionary**, so that testing can be performed at each phase of implementation.

2. In the design the data, architecture, interfaces and components should **be clearly represented.**
3. The design should be **modular.** That means the subsystems in the design should be logically partitioned.
4. The **data structure** should be appropriately chosen for the design of specific problem.
5. The **components** should be used in the design so that functional independency can be achieved in the design.
6. Using the information obtained in software **requirement analysis** the design should be created.
7. The **interfaces** in the design should be such that the complexity between the connected components of the system gets reduced. Similarly interface of the system with external interface should be simplified one.
8. Every design of the software system should **convey its meaning** appropriately and effectively.

Quality Attributes(FURPS Model)

The design quality attributes popularly known as **FURPS** (Functionality, Usability, **R**eliability, **P**erformance and **S**upportability) is a set of criteria developed by Hewlett and Packard. Following table represents meaning of each quality attribute

Quality attribute	Meaning
Functionality	Functionality can be checked by assessing the set of **features** and **capabilities** of the functions. The functions should be **general** and should not work only for particular set of inputs. Similarly the **security** aspect should be considered while designing the function.
Usability	The usability can be assessed by knowing the **usefulness** of the system.
Reliability	Reliability is a measure of frequency and severity of **failure**. Repeatability refers to the consistency and repeatability of the measures. The Mean Time To Failure (**MTTF**) is a metric that is widely used to measure the product's performance and reliability.

Performance	It is a measure that represents the response of the system. Measuring the performance means measuring the processing speed, memory usage, response time and efficiency.
Supportability	It is also called maintainability. It is the ability to adopt the enhancement or changes made in the software. It also means the ability to withstand in a given environment.

Review Question

1. *Discuss in detail about the design process in software development process.* **AU : May-05, Marks 8**

3.3 Design Concepts

AU : May-06,08,16,19, Dec.-05,17,22, Marks 16

The software design concept provides a framework for implementing the right software.

Following are certain issues that are considered while designing the software -

- Abstraction
- Modularity
- Architecture
- Refinement
- Pattern
- Information hiding
- Functional independence
- Refactoring
- Design classes

3.3.1 Abstraction

The abstraction means an ability to cope up with the complexity. Software design occurs at different levels of abstraction. At each stage of software design process levels of abstractions should be applied to refine the software solution. At the **higher level** of abstraction, the solution should be stated in **broad terms** and in the **lower level** more **detailed description** of the solution is given.

While moving through different levels of abstraction the procedural abstraction and data abstraction are created.

The **procedural abstraction** gives the named sequence of instructions in the specific function. That means the functionality of procedure is mentioned by its implementation details are hidden. For example : Search the record is a procedural abstraction in which implementation details are hidden(i.e. Enter the name, compare each name of the record

against the entered one, if a match is found then declare success !! Other wise declare 'name not found')

In **data abstraction** the collection of data objects is represented. For example for the procedure search the data abstraction will be record. The record consists of various attributes such as record ID, name, address and designation.

3.3.2 Modularity

- The software is divided into separately named and addressable components that called as **modules.**
- Monolithic software is hard to grasp for the software engineer, hence it has now become a trend to divide the software into number of products. But there is a co-relation between the number of modules and overall cost of the software product. Following argument supports this idea -

"Suppose there are two problems A and B with varying complexity. If the complexity of problem A is greater than the complexity of the problem B then obviously the efforts required for solving the problem A is greater than that of problem B. That also means the time required by the problem A to get solved is more than that of problem B."

The overall complexity of two problems when they are combined is greater than the sum of complexity of the problems when considered individually. This leads to **divide and conquer strategy** (according to divide and conquer strategy the problem is divided into smaller subproblems and then the solution to these subproblems is obtained). Thus dividing the software problem into manageable number of pieces leads to the concept of modularity. It is possible to conclude that if we subdivide the software indefinitely then effort required to develop each software component will become very small. But this conclusion is invalid because the total number of modules get increased the efforts

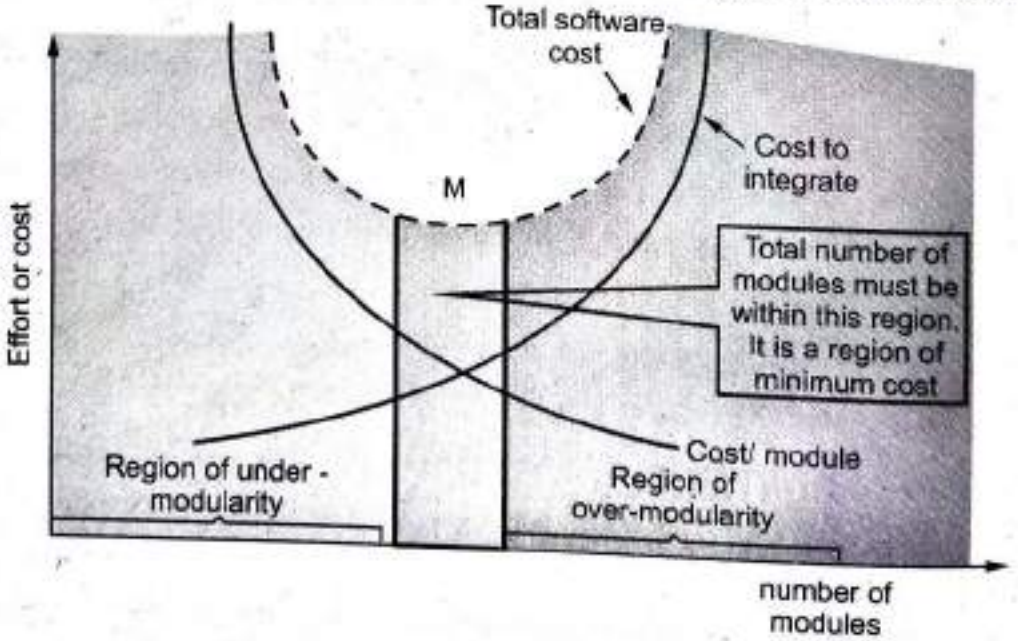


Fig. 3.3.1 Modularity and software cost

required for developing each module also gets increased. That means the cost associated with each effort gets increased. The effort(cost)required for integration these modules will also get increased. The total cost required to develop such software product is shown in Fig. 3.3.1.

The Fig. 3.3.1 provides useful guideline for the modularity and that is - **overmodularity or the undermodularity while developing the software product must be avoided.** We should modularize the software but the modularity must remain near the region denoted by **M**

- Modularization should be such that the development can be planned easily, software increments can be defined and delivered, changes can be more easily accommodated and long term maintenance can be carried out effectively.
- **Meyer** defines five criteria that enable us to evaluate a design method with respect to its ability to define an effective modular system :
- **Modular decomposability :** A design method provides a systematic mechanism for decomposing the problem into sub-problems. This reduces the complexity of the problem and the modularity can be achieved.
- **Modular composability :** A design method enables existing design components to be assembled into a new system.
- **Modular understandability :** A module can be understood as a standalone unit. It will be easier to build and easier to change.
- **Modular continuity :** Small changes to the system requirements result in changes to individual modules, rather than system-wide changes.
- **Modular protection :** An aberrant condition occurs within a module and its effects are constrained within the module.
- **Importance :**
- Due to modularity, each task forms **a separate, distinct program module.** At implementation time each module and its **inputs and outputs are well-defined;** there is no confusion in the intended interface with other system modules.
- The testing of modules and their integrity with other modules can be tested independently.
- Some modules may be defined by standard procedures which are used and reused in different programs or parts of the same program.
- Large projects containing modules become easier to monitor and control.

3.3.3 Architecture

Architecture means representation of **overall structure** of an integrated system. In architecture various components interact and the data of the structure is used by various components. These components are called **system elements**. Architecture provides the **basic framework** for the software system so that important framework activities can be conducted in systematic manner.

In architectural design various system models can be used and these are

Model	Functioning
Structural model	Overall architecture of the system can be represented using this model
Framework model	This model shows the architectural framework and corresponding applicability.
Dynamic model	This model shows the reflection of changes on the system due to external events.
Process model	The sequence of processes and their functioning is represented in this model
Functional model	The functional hierarchy occurring in the system is represented by this model.

3.3.4 Refinement

- Refinement is actually a process of elaboration.
- Stepwise refinement is a top-down design strategy proposed by Niklaus WIRTH.
- The architecture of a program is developed by successively refining levels of procedural detail.
- The process of program refinement is analogous to the process of refinement and partitioning that is used during requirements analysis.
- Abstraction and refinement are complementary concepts. The major difference is that - in the abstraction low-level details are suppressed. Refinement helps the designer to elaborate low-level details.

3.3.5 Pattern

According to **Brad Appleton** the design pattern can be defined as - It is a named nugget(something valuable) of insight which conveys the essence of proven solution to a recurring problem within a certain context.

In other words, design pattern acts as a design solution for a particular problem occurring in specific domain. Using design pattern designer can determine whether-

- Pattern can be **reusable**.

- Pattern can be used for **current work.**
- Pattern can be used to **solve similar kind of problem** with different functionality.

3.3.6 Information Hiding

Information hiding is one of the important property of effective modular design. The term information hiding means the modules are designed in such a way that information contained in one module cannot be accessible to the other module (the module which does not require this information). Due to information hiding only limited amount of information can be passed to other module or to any local data structure used by other module.

The **advantage** of information hiding is basically in testing and maintenance. Due to information hiding some data and procedures of one module can be hidden from another module. This ultimately **avoids** introduction of **errors** module from one module to another. Similarly one can make **changes** in the desired module without affecting the other module.

3.3.7 Functional Independence

- The functional independence can be achieved by developing the functional modules with single-minded approach.
- By using functional independence functions may be compartmentalized and interfaces are simplified.
- Independent modules are easier to maintain with reduced error propagation.
- Functional independence is a key to good design and design is the key to software quality.
- The major benefit of functional independence is in achieving effective modularity.
- The functional independence is assessed using two qualitative criteria - **Cohesion** and **coupling.**

3.3.7.1 Cohesion

- With the help of cohesion the information hiding can be done.
- A cohesive module performs only "one task" in software procedure with little interaction with other modules. In other words cohesive module performs only one thing.
- Different types of cohesion are :
  1. **Coincidentally cohesive** - The modules in which the set of tasks are related with each other loosely then such modules are called coincidentally cohesive.

2. **Logically cohesive** - A module that performs the tasks that are logically related with each other is called logically cohesive.
3. **Temporal cohesion** - The module in which the tasks need to be executed in some specific time span is called temporal cohesive.
4. **Procedural cohesion** - When processing elements of a module are related with one another and must be executed in some specific order then such module is called procedural cohesive.
5. **Communicational cohesion** - When the processing elements of a module share the data then such module is communicational cohesive.

- The goal is to achieve high cohesion for modules in the system.

3.3.7.2 Coupling

- Coupling effectively represents how the modules can be "connected" with other module or with the outside world.
- Coupling is a measure of interconnection among modules in a program structure.
- Coupling depends on the interface complexity between modules.
- The goal is to strive for lowest possible coupling among modules in software design.
- The property of good coupling is that it should reduce or avoid change impact and ripple effects. It should also reduce the cost in program changes, testing and maintenance.
- Various types of coupling are :

i) **Data coupling** - The data coupling is possible by parameter passing or data interaction.

ii) **Control coupling** - The modules share related control data in control coupling.

iii) **Common coupling** - In common coupling common data or a global data is shared among the modules.

iv) **Content coupling** - Content coupling occurs when one module makes use of data or control information maintained in another module.

Sr. No.	Coupling	Cohesion
1.	Coupling represents how the **modules are connected** with other modules or with the outside world.	In cohesion, the cohesive module performs **only one thing.**
2.	With coupling **interface complexity** is decided.	With cohesion, **data hiding** can be done.

3.	The goal of coupling is to achieve **lowest coupling.**	The goal of cohesion is to achieve **high cohesion.**
4.	Various types of couplings are - Data coupling, Control coupling, Common coupling and Content coupling.	Various types of cohesion are - Coincidental cohesion, Logical cohesion, Temporal cohesion, Procedural cohesion and Communicational cohesion.

3.3.8 Refactoring

Refactoring is necessary for simplifying the design without changing the function or behaviour. **Fowler** has defined refactoring as "the process of changing a software system in such a way that the external behaviour of the design do not get changed, however the internal structure gets improved".

Benefits of refactoring are -

- The **redundancy** can be achieved.
- **Inefficient algorithms** can be eliminated or can be replaced by efficient one.
- Poorly constructed or **inaccurate data structures** can be removed or replaced.
- Other **design failures** can be rectified.

The decision of refactoring particular component is taken by the designer of the software system.

3.3.9 Design Classes

Design classes are defined as the classes that describe some elements of problem domain, focus on various aspects of problem from user's point of view.

The **goal** of design classes is :

1. To refine the analysis classes by providing the detail design, so that further implementation can be done easily.
2. To create new set of classes for implementing the infrastructure of the software.

There are five different types of design classes

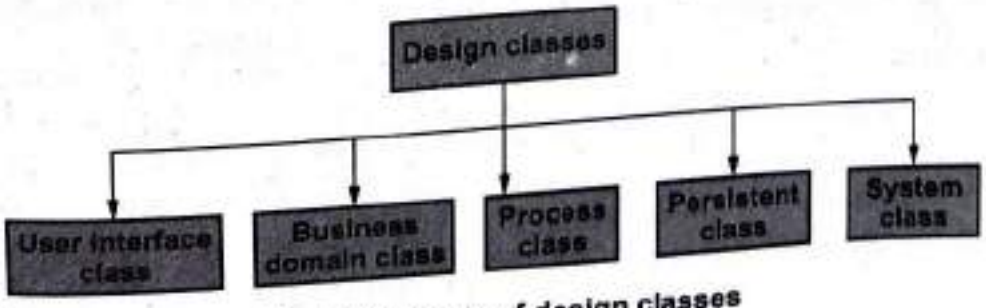


Fig. 3.3.2 Types of design classes

1. User Interface Class

The user interface class defines all the abstractions that are necessary for Human Computer Interface(HCI). The user interface classes is basically a **visual representation** of the HCI.

2. Business Domain Class

Business domain classes are the refinement of analysis classes. These classes identify the **attributes and services** that are needed to implement some elements of **business domain.**

3. Process Class

Process class implement lower level business abstractions used by the business domain.

4. Persistent Class

These classes represent the data store such as databases which will be retained as it is even after the execution of the software.

5. System Class

These classes are responsible for software management and control functions that are used for system operation.

Each class must be **well formed design class**. Following are some **characteristics** of **well formed design classes -**

- **Complete and Efficient**

A design class must be properly **encapsulated** with corresponding attributes and methods. Design class must contain all those methods that are sufficient to achieve the intent of the class.

- **Primitiveness**

Methods associated with one particular class must perform unique service and the class should not provide another way to implement the same service.

- **High Cohesion**

A cohesive designed class must posses small, focused set of responsibilities and these responsibilities must be associated with all the attributes of that class.

- **Low Coupling**

Design classes must be collaborated with manageable number of classes. If one design class is collaborated with all other design classes then the implementation, testing and maintenance of the system becomes complicated. The **law of Demeter** suggests that the one particular design class should send messages to the methods of neighbouring design classes only.

Review Questions

1. *Describe decomposition levels of abstraction and modularity concepts in software design.* **AU : Dec.-05, Marks 16**
2. *Explain about the various design concepts considered during design.* **AU : May-06, Marks 12 , Dec.-17, Marks 13**
3. *Describe the concepts of cohesion and coupling. State difference between cohesion and coupling with a suitable example.* **AU : May-08,13, Dec.-22, Marks 13**
4. *What is modularity ? State its importance and explain coupling and cohesion.* **AU : May-16, Marks 8**
5. *List and explain any five fundamental software design concepts.* **AU : May-19, Marks 13**

3.4 Design Patterns

AU : Dec.-15, May-19, Marks 8

Definition of design pattern : Design pattern is general reusable solution to commonly occurring problem within a given context in software design.

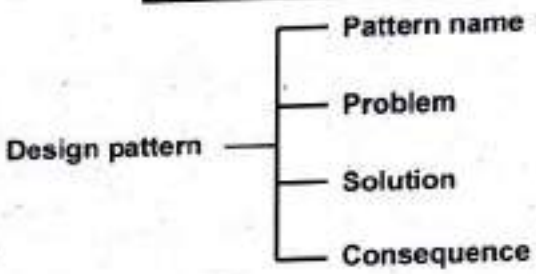


Fig. 3.4.1

In general the pattern has **four essential elements -**

Pattern name : The pattern name is used as handle to describe the design pattern. The naming of the pattern increases the design vocabulary. Due to the name of the pattern it becomes easy to think about the design and to communicate the design with the others. Finding the appropriate name for the design pattern is one of the hardest and tricky part.

Problem : The problem describes when to apply the pattern. It explains the problem and its context. It might describe various things such as algorithm for the problem, the class or the object structure of the problem or it can list down the conditions that must be followed before applying the pattern.

Solution : The solution describes the elements of the design, their relationships, responsibilities and collaborations. The solution never describes the concrete design or the implementation because the pattern is like templates and it can be applied to various problems in varied situations.

Consequence : The consequences describe the results and trade-offs of applying pattern. The consequences for software often concerned with space and time trade-offs. The listing of consequences helps in evaluation of the design pattern.

Generally the design pattern is divided into three categories - creational pattern, structural pattern and behavioral pattern.

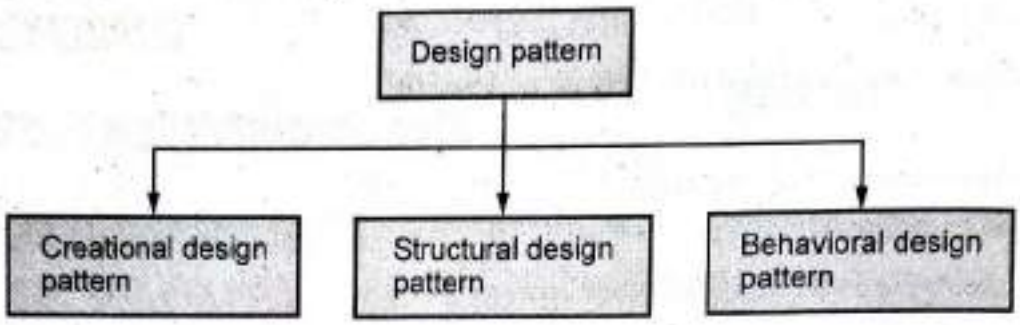


Fig. 3.4.2 Categories of design pattern

Example 3.4.1 *State the role and patterns while developing system design.*

AU : Dec.-15, Marks 8

Solution :

Sr.No	Pattern	Role
1.	Creator	The role of creator pattern is to find creator that needs to be connected to the created object in any event.
2.	Information expert	This pattern assigns the responsibilities to various objects that are participating in the system design.
3.	Singleton	This pattern ensures that a class has only one instance and provides global point of access to it.
4.	Factory method	It is responsible for creating the instances of several derived classes.
5.	Bridge	The role of this pattern is to separate out the interface from implementation without using inheritance.
6.	Adapter	The role of this pattern is to allow the interface of existing class to be used from another interface.
7.	Observer	The role of this pattern is to define the link between objects so that when one object's state changes, all dependent objects are updated automatically.
8.	Strategy	The role of this pattern is to define the family of algorithms, encapsulate these algorithms within in class and make them interchangeable.

3.4.1 Creational Pattern

Creational design pattern are the design pattern that are used for **object creation** mechanism. This type of pattern is used in the situation when basic form of object creation could result in design problems or increase complexity of a code base.

The creational patterns show how to make the software design flexible.

There are five well known design patterns that are part of creational pattern. These are -

1. **Abstract factory pattern :** This pattern is for creating the instances of several families of classes.
2. **Builder pattern :** This pattern separates the object construction from its representation.
3. **Factory method pattern :** This pattern is for creating the instances of several derived classes.
4. **Prototype pattern :** It specifies the kinds of objects to create using a prototypical instance and create new objects by copying this prototype.
5. **Singleton pattern :** It ensures that a class has only one instance and provides global point of access to it.

3.4.2 Structural Pattern

- In software engineering, structural design patterns are design patterns that ease the design by identifying a simple way **to realize relationships between entities.**
- The structural design pattern serves as a blueprint for how different classes and objects are combined to form larger structures.
- Structural patterns are just similar to data structures.
- There are various design patterns that are the part of structural design pattern -

1. **Bridge :** This pattern separates the object interface from its implementation.
2. **Adapter :** This pattern matches the interface of different classes.
3. **Composite :** This pattern is based on the tree structure of simple and composite objects.
4. **Decorator :** In this pattern the responsibilities on the object are dynamic.
5. **Façade :** In this pattern the single class represents the entire subsystem.
6. **Flyweight :** This pattern makes use of fine-grained instance for efficient sharing.
7. **Private class data :** In this pattern there is restricted access to the class data.
8. **Proxy :** In this pattern one object represents another object.

3.4.3 Behavioural Pattern

- The behavioral design patterns are design patterns that identify common communication patterns between objects and realize these patterns. By doing so, these patterns increase flexibility in carrying out this communication.
- A behavioral pattern explains how objects interact.
- It describes how different objects and classes send messages to each other to make things happen and how the various tasks are divided among different objects.
- There are various design patterns that are the part of behavioral design pattern -

1. **Chain of responsibility** : The chain of responsibility pattern is a design pattern that defines a linked list of handlers, each of which is able to process requests.
2. **Command** : The command pattern is a design pattern that enables all of the information for a request to be contained within a single object.
3. **Interpreter** : This pattern specifies how to evaluate sentences in a language. This pattern is useful when developing domain-specific languages or notations.
4. **Iterator** : This pattern is designed to sequentially access the elements of a collection.
5. **Mediator** : In this pattern the communication between objects is encapsulated with a mediator object. Instead of classes communicating directly, classes send messages to the mediator and the mediator sends these messages to the other classes.
6. **Null object** : This pattern is designed to act as a default value of an object.
7. **Memento** : This pattern captures and restores an object's internal state.
8. **Observer** : The observer pattern is a design pattern that defines a link between objects so that when one object's state changes, all dependent objects are updated automatically.
9. **State** : In this pattern it alters the object's behavior when its state changes.
10. **Strategy** : In this pattern, it encapsulates an algorithm inside a class.
11. **Template method** : This pattern defers the exact steps of an algorithm to a subclass.
12. **Visitor** : This class defines new operation to class without changes.

Review Question

1. *Write a short note on the structural patterns.* AU : May-19, Marks 4

3.5 Model-View-Controller

Problem : The user interface often gets changed when we extend the functionality of an application. Sometimes a customer may call for a specific user interface adaption. Different users may place conflicting requirements on the user interface. Building a system with required flexibility is expensive and error-prone if user interface is tightly interwoven with core functional logic.

Solution

The Model View Controller is also known as MVC architecture. It divides the whole application logic into three sections **model, view** and **controller**. This design pattern separates the user interface from core functional logic.

Structure

The working is as follows -

1. Suppose a user is requesting a web page from the server, then server will send this request to a specific **controller**.
2. The **controller** then asks to get the data from the **model**.
3. The **model** interacts with the database and sends the data to the controller on validation.
4. The **controller** then sends this data to the **view** in order to render it or to make a presentation of that data.
5. The **view** sends the proper presentation of data to the **controller**.
6. The **controller** then sends this presentation to the client who requested the web page. This presentation is considered a response to the client's request.

The following figure illustrates the above work.

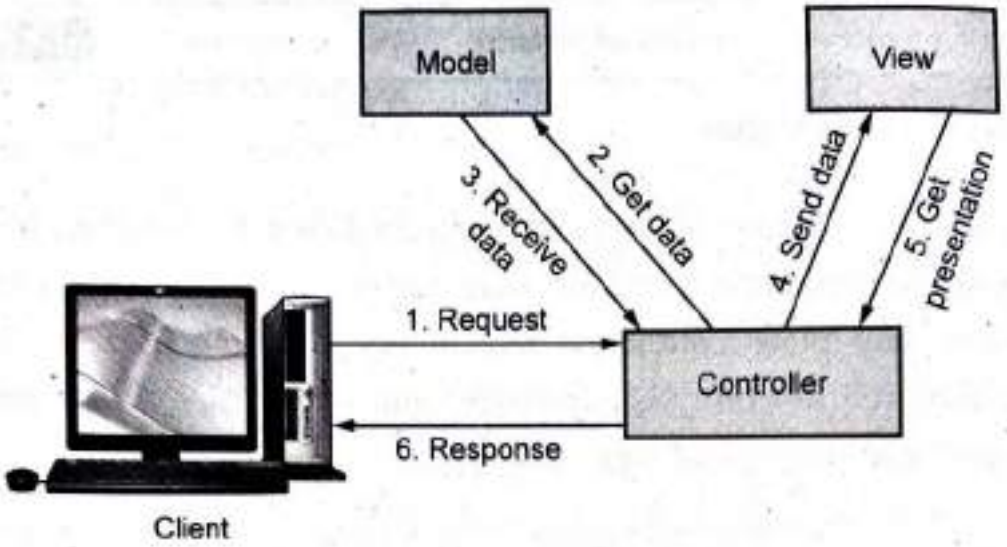


Fig. 3.5.1 Model View Controller architecture

➢ **Participating classes :**

The purpose of each component of Model-View-Controller architecture is described as follows -

Model

1) It handles the business logic or data logic.
2) It interacts with the database to get the requested data.
3) It performs the validation of that data.
4) It then sends the valid data to the controller.

View

1) It prepares the presentation of the data received from the controller.
2) It then sends the presentation to the controller.

Controller

1) It handles the request flow from **client**, to **model** and from **model** to the **view**. Again from **view** back to the client.
2) Sends the response to the **client**.

Known uses :

The MVC pattern is used mostly in web development applications.

The best-known example of the use of the Model-View-Controller pattern is the user-interface framework in the Smalltalk environment.

➢ **Consequences :**

Advantages

1) It is easy to make modifications in the application as the business logic is separated from the presentation logic.
2) The development of applications becomes fast.
3) It is easy for the developers to collaborate and work together.
4) Development of application becomes fast.

Disadvantages

1) It is a complex architectural pattern and difficult to learn and implement.
2) For developing simple applications MVC is not a suitable design pattern as it adversely affects the performance of such applications.

3.6 Publisher-Subscriber Pattern

Problem : A commonly arising situation is that data changes at one place and there exists many components depending on this data. For example in some GUI based application, if some internal data gets changed, then multiple views based on this data get affected.

Solution

- To solve this problem there must be some dedicated component who can inform the changes in the data to all the dependent components. Hence, in the publisher subscriber pattern, one dedicated component takes the role of the **publisher**(also called as subject). All the components dependent on changes in the publisher are called its **subscribers**(also called as observer).
- The publisher maintains a registry of currently subscribed components. Whenever a component wants to become a subscriber, it makes use of **subscribe interface** offered by publisher.
- Whenever the publisher changes the state, it sends the notification to all its subscribers.
- The subscribers then retrieve the changed data.
- Following are the freedoms used in publisher-subscriber pattern -
  - The publisher can decide which internal state changes it will notify to its subscribers.
  - An object can be subscriber to many publishers.
  - An object can take both the roles - it can be publisher or it can be a subscriber.
- Based on publisher-subscriber pattern the **push and pull models** are derived.

- In the push model, the publisher sends all the changed data to its subscribers.
- In the pull model, the publisher only sends minimal information about the change to its subscribers. It is then subscribers' responsibility to get the required data from the publisher whenever needed.
- The push model has rigid dynamic behavior whereas the pull model is flexible.

Structure :

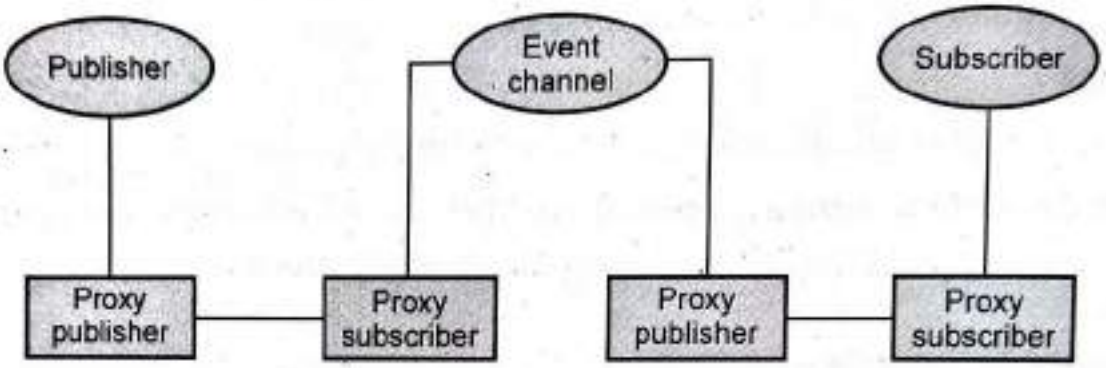


Fig. 3.6.1 Publisher-Subscriber pattern

Known uses :

Event messaging : Publisher-Subscriber pattern is widely used in delivery logistics. As we shop online more frequently for a wider variety of goods, package delivery has become commonplace. Logistics companies need to use delivery resources more efficiently. To optimize delivery, dispatching systems need up-to-date information on where their drivers are. Pub/Sub-event messaging helps logistics companies do this.

Advantages :

1) There is a support for distributed system.

2) Extensibility

Disadvantages :

1) Less efficient

2) Complex pattern to implement

3.7 Adapter

AU : May-11,12,13,14, Dec.-11,12,13,14, Marks 16

Sometime we come across the situation where two components are working well functionally but they can not work together because of some reason. In such a situation we can use something like adapter to interface two components.

In object oriented design the adapter pattern is adapting between classes and objects. Like any adapter in the real world where it is used to be an interface, a bridge between two objects. In real world we have adapters for power supplies, If you can not plug in the

power cord in your laptop you can use an adapter. You plug the power cord in the adapter and the adapter in to laptop slot.

Problem : How to resolve incompatible interfaces. How to provide a stable interface to the components having different interfaces ?

Solution

Convert the original interface of component into another interface through intermediate adapter object.

This pattern is also known as **wrapper.** This is a **GoF** pattern.

There are two types of adapters -

1. Object adapters 2. Class adapters

Object adapters :

In this type of adapter pattern the adapter contains the instance of a class it wraps. When the **Client** wants some task to get fulfilled it makes a request to the **Target.** The **Adapter** inherits the **Target** and at the same time holds the instance of **Adaptee.** Hence a specific requests of **Adaptee** can be invoked by the **Apdapter**. Thus it is the **Adapter** who hides the **Adaptees's** interface from the client. It is illustrated in the Fig. 3.7.1(a).

Class adapters :

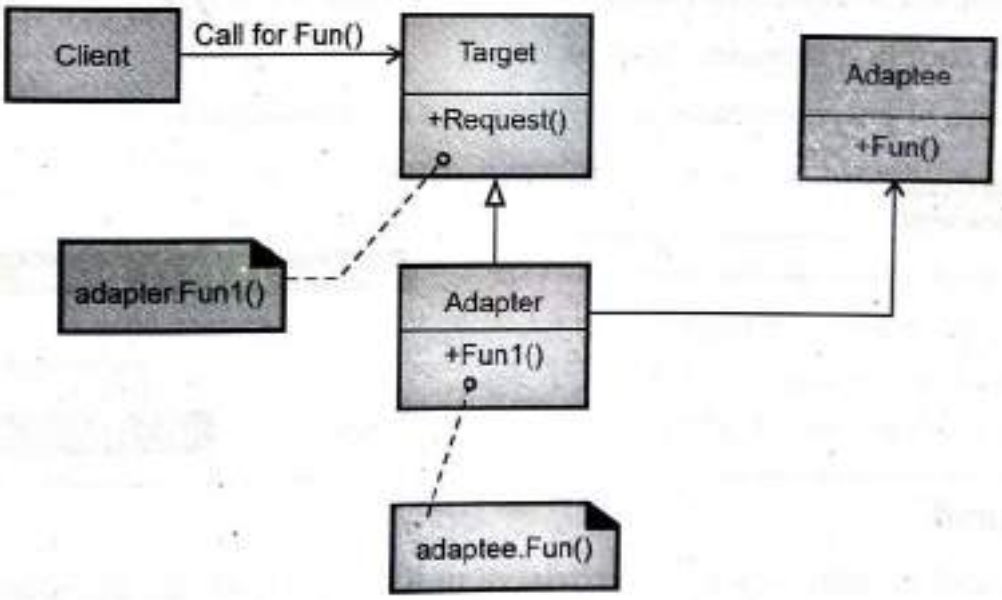


Fig. 3.7.1 (a) Object adapter pattern

In this type of adapter pattern the **Adapter** uses multiple polymorphic interfaces of **Adpatee.** Hence the number of requests of **Adaptee1,Adaptee2,..., AdapteeN** can be invoked by the **Apdapter.** It is illustrated in the Fig. 3.7.1(b).

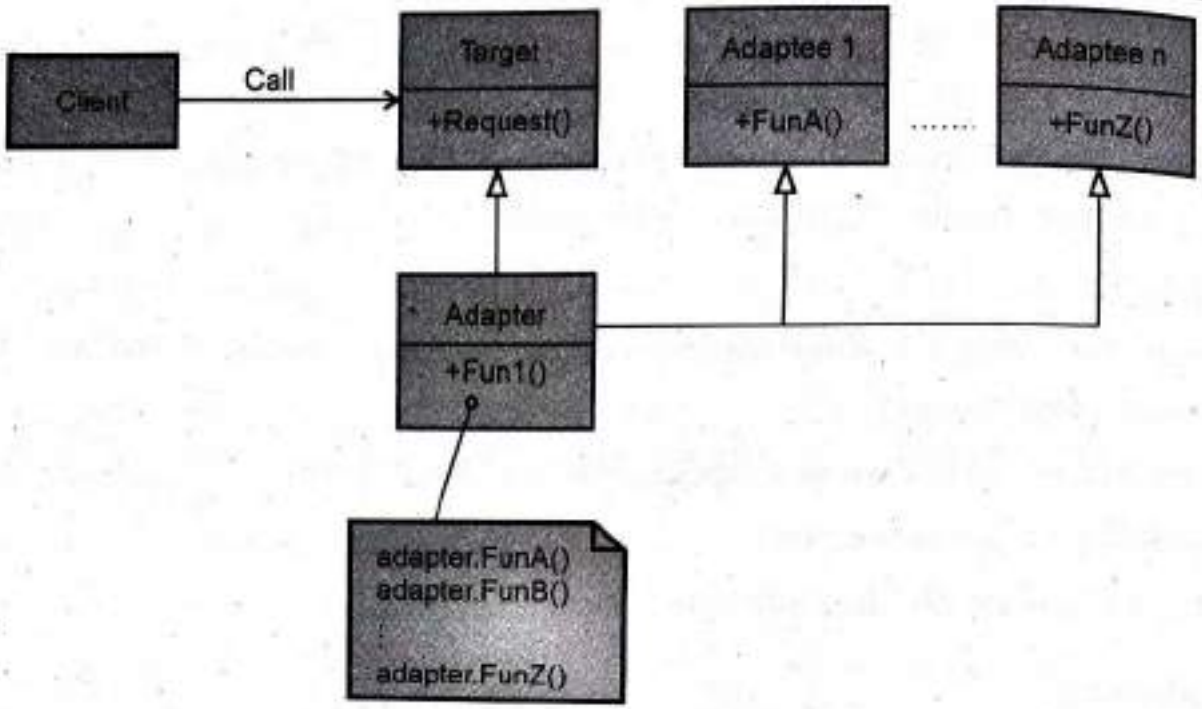


Fig. 3.7.1 (b) Class adapter pattern

Example

The typical examples of adapter pattern are WindowsAdapter in Swing API or JDBC-ODBC API.

Benefits

1. Two compatible objects or classes can be interfaced using adapter pattern.
2. Reduces coupling between the objects.
3. Due to use of polymorphism and indirection only essential behavior is focused.

Review Questions

1. *Write a short note on adapter pattern.* **AU : May-11,13,14, Dec.-13,14, Marks 4**
2. *What are the two types of adapter pattern ?*
3. *Write short note on adapter.*
4. *What is design pattern ? Explain the GoF design patterns.* **AU : May-12, Marks 4** **AU : Dec.-11,12 Marks 16**

3.8 Command

- Command pattern is one of the behavioural design pattern. This design pattern is used to implement loose coupling in a request-response model.

Intent

The command pattern converts a request or an operation into a standalone object that contains all the information about the request.

Also Known As

Action, Transaction

Motivation

There is an encapsulation of the operational request of a service into an entity called command so that the same service can be used to trigger different operations. Optionally the command can also be used for **undo, logging or queueing** operations.

Participants

Command_Interface :

Declares an interface for executing an operation.

ConcreteClass (Command_name...) :

- Defines a binding between a Receiver object and an action.
- Implements Execute by invoking the corresponding operation(s) on Receiver.

Client (Application) :

Creates a **ConcreteClass** object and sets its receiver.

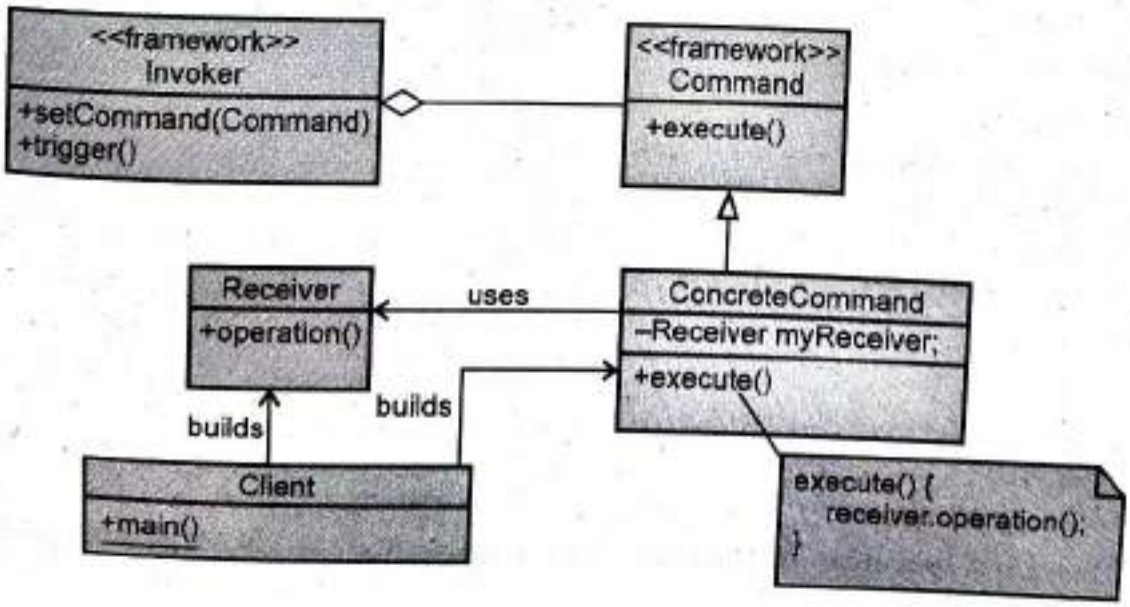
Invoker (command) :

Asks the command to carry out the request.

Receiver (Receiver_item) :

Knows how to perform the operations associated with carrying out a request. Any class may serve as a Receiver.

Structure



Implementation

```
class Invoker {
   private Command myCommand;
   public void setCommand(Command aCommand){
      myCommand = aCommand; //actual command
   }
   public void trigger(){
      myCommand.execute();
   }
}

absract class Command {
   abstract void execute();
}

class ConcreteCommand : Command {
   private Receiver myReceiver;
   public ConcreteCommand(Receiver aReceiver) {
      myReceiver = aReceiver;
   }

   public void execute() {
      myReceiver.operation();
   }
}

class Receiver {
   public operation() { // operation here }

}
class Client {
  public static void Main(String[] args) {
    Command c =
      new ConcreteCommand(new Receiver));
    Invoker i = new Invoker()
    i.setCommand(c);
    i.trigger()
  }
}
```

Example

Consider an application in which we want to perform some operations on a text file. Let us call it as **FileOperation** application. These operations can be **Read File** operation or

Write File operation using Read or Write commands. To implement **Read** and **Write** commands we need a text file. Hence the text file is called as **Receiver** over here. The **ReadFileOperation** and **WriteFileOperation** are the **concrete classes** that will invoke the commands **Read** and **Write** respectively. These methods are placed in the **Receiver**. Hence Receiver is something that will actually execute the commands. There is an **Executor** or an **Invoker** which will invoke the commands. The **Invoker** will have the **list of commands** that will be executing currently. This facility of Invoker is also useful if we want to perform undo operations. The **Client** is a command application. The **Client** calls the **Invoker** which in turn executes the desired command.

To Summarize,

Command : Read and Write operations

Receiver : File on which the operations are to be performed.

Invoker : Contains the list of commands that are currently executing

Client : Will be using the Invoker to call the commands

The implementation for the above example is as follows -

```
//Command Interface
public interface FileOperation {
   String execute();
}
//Concrete Class1
public class ReadOperation implements FileOperation {
   private Text_File myfile;
   //Constructor
   public ReadOperation(Text_File myfile) {
      this.myfile = myfile;
   }
   @Override
   public String execute(){
      return myfile.Read();
   }
}
//Concrete Class2
public class WriteOperation implements FileOperation {
   private Text_File myfile;
   //Constructor
   public WriteOperation(Text_File myfile) {
      this.myfile = myfile;
   }
   @Override
```

```
    public String execute(){
        return myfile.Write();
    }
}
//Receiver
public class Text_File {
    private String file_name;
    //Constructor
    public Text_File(String file_name) {
        this.file_name = file_name;
    }
    //Actual Code for Read Command
    public String Read() {
        return "Reading a file: "+ file_name;
    }
    //Actual Code for Write Command
    public String Write() {
        return "Writing to a file: "+ file_name;
    }
}
//Invoker or Executor
public class MyFileOperationInvoker {
    //creating list of operations
    private final list<FileOperation> List = new ArrayList<>();
    public String executeCommand(FileOperation myOperation) {
        //Adding the currently invoked command to the ArrayList
        List.add(myOperation);
        return myOperation.execute();
    }
}
//Client
class Application_Client {
    public static void main(String args[]) {
        //client is invoking the invoker or Executor
        MyFileOperationInvoker myExecutor = new MyFileOperationInvoker();
        //Using Executor object the command for execution is called via Concrete class constructor
        //by passing the file name(Note that file name acts as a receiver)
        myExecutor.executeCommand(new ReadOperation(new Text_File("test1.txt")));
        myExecutor.executeCommand(new WriteOperation (new Text_File("test1.txt")));
    }
}
```

➢ Merits and Demerits

Merits

1) New commands can be easily added to the existing application.
2) It is easy to perform redo and undo operations while using the commands.
3) It decouples the classes that invoke the operation from the object that knows how to execute the operation

Demerit

1) Using command design pattern may requires more effort on implementation, since each command requires a concrete command class, which will increase the number of classes significantly.

3.9 Strategy

Intent

- There are common situations when classes differ only in their behavior. For this case, is a good idea to isolate the algorithms in separate classes in order to have the ability to select different algorithms at runtime.
- This is a kind of design pattern in which a set of similar algorithms to be defined and encapsulated in their own classes.
- Thus this pattern is intended to define a family of algorithms, encapsulate each of them and make them interchangeable.
- Strategy is like **Template Method**.

Structure and Implementation

Following is an UML diagram which represents how to implement strategy design pattern.

Client class makes use of interchangeable algorithms. It maintains the reference to strategy object.

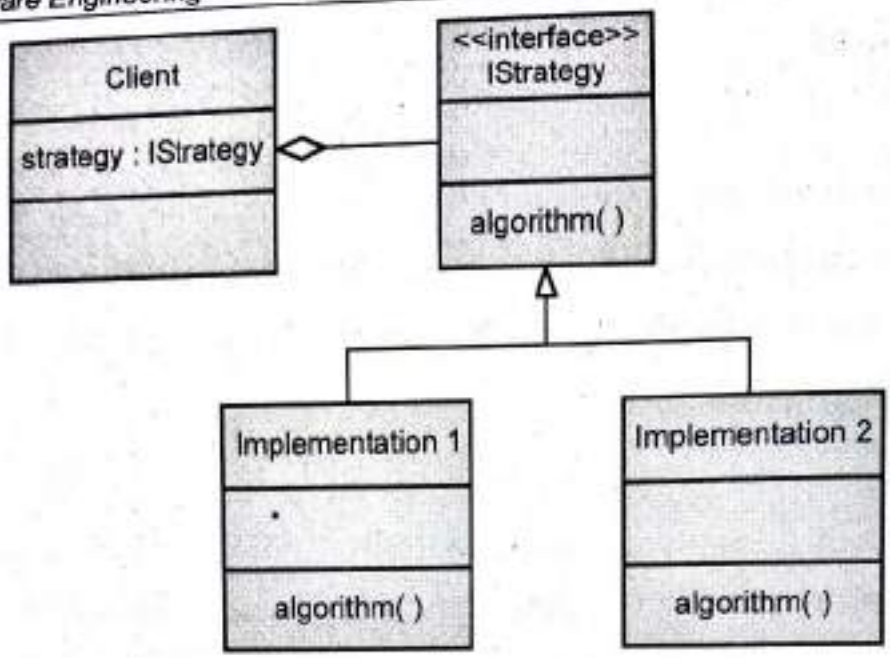


Fig. 3.9.1 Representation of strategy

IStrategy declares an interface common to all supported algorithms. The client uses this interface to call the algorithm defined by a **Implementation1**, **Implementation2** and so on.

Implementation1, **Implementation2** represent instances that provide different algorithms that can be used by the client.

The skeleton code for this can be as given below

```
static class Program
{
   public static void main(String arg[])
   {
     //invoking the algorithm() of Implementation1
     //invoking the algorithm() of Implementation2
   }
}

public class Client
{
     //declaring strategy object
}

public abstract class IStrategy
{
   public void algorithm();
}

public class Implementation1 extends IStrategy
{
   public  void algorithm()
   {
   }
```

```
}
public class Implementation1 extends IStrategy
{
public  void algorithm()
   {
   }
}
```

Example :

In online shopping system modes of payment is a typical example of strategy pattern. The online payment mode can be card payment, cash payment and so on.

➢ Merits and Demerits

Merits

1. If one needs to have several different behaviours that the object to perform, it is much simpler to keep track of them if each behaviour is a separate class. One can add, remove or change any of the behaviours very easily.
2. This strategy brings the flexibility in the code.
3. When you have several objects that are basically the same and differ only in their behaviour, it is a good idea to make use of the strategy pattern.

Demerits

1. The application must be aware of all the strategies to select the right one in right situation.
2. The memory requirement for implementing this pattern is more as **Client** has to maintain the object of **Strategy** base class, in order to choose appropriate algorithm each time.

Example 3.9.1 *For the description given below, draw the class diagram and identify the roles for a strategy design pattern. Mention the roles identified for each class and its relevant behaviour in the class diagram. Grand health club offers a scheme for membership of the health club. The options available for registering are 'yoga' and 'aerobics'. The monthly charges for aerobics membership are 2000.00. The monthly charges for yoga membership are 1000.00. The members can avail a single option out of the two options. If a person books for three months, he gets 20 % discount. If he books for six months, he gets 25 percent discount. If he books for nine months, he gets 35 % discount. If he books for one year, he gets 50 % discount.*

Solution :

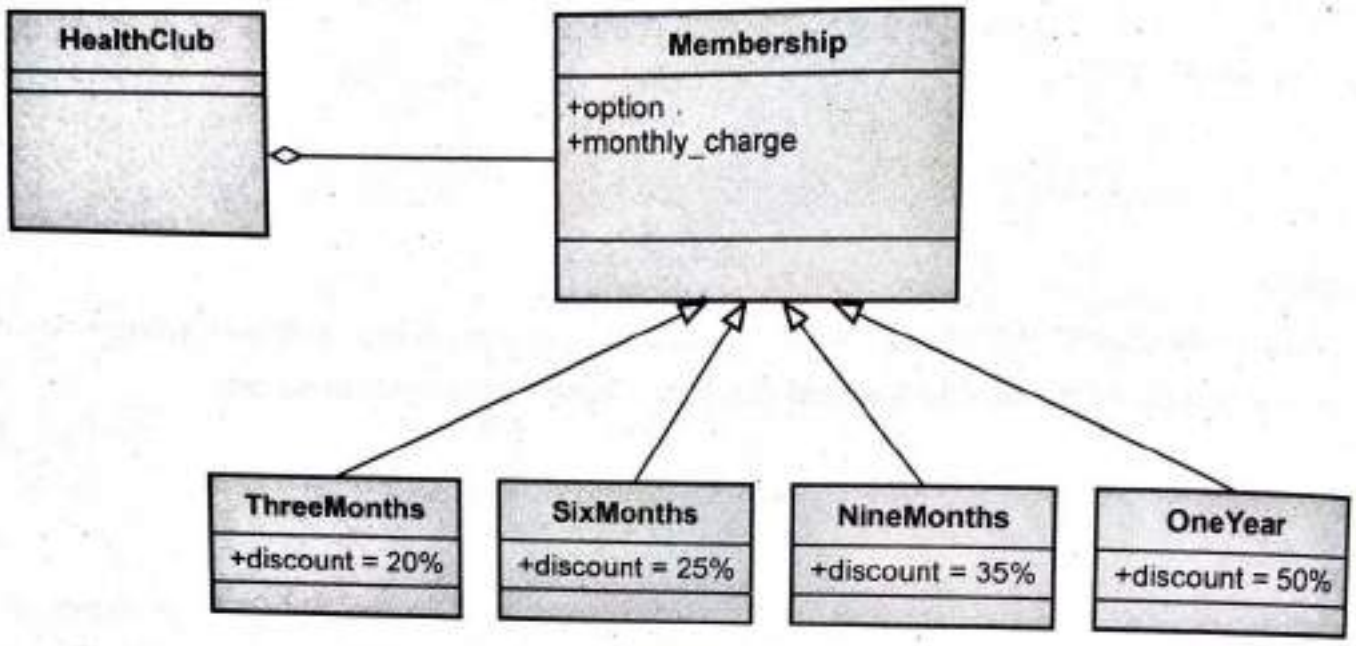


Fig. 3.9.2

Example 3.9.2 *Apply strategy design pattern to the following and draw the class diagram. A company has many employees. Each employee has a name and a performance index in the range of 1 to 5. When the index is 2, the increment is 10 percent of the previous year salary. ,3 the increment is 15 percent of the previous year salary, 4 the increment is 20 percent of the previous year salary and when 5 the increment is 25 percent of the previous year salary. Indicate the roll of each class in the class diagram.*

Solution :

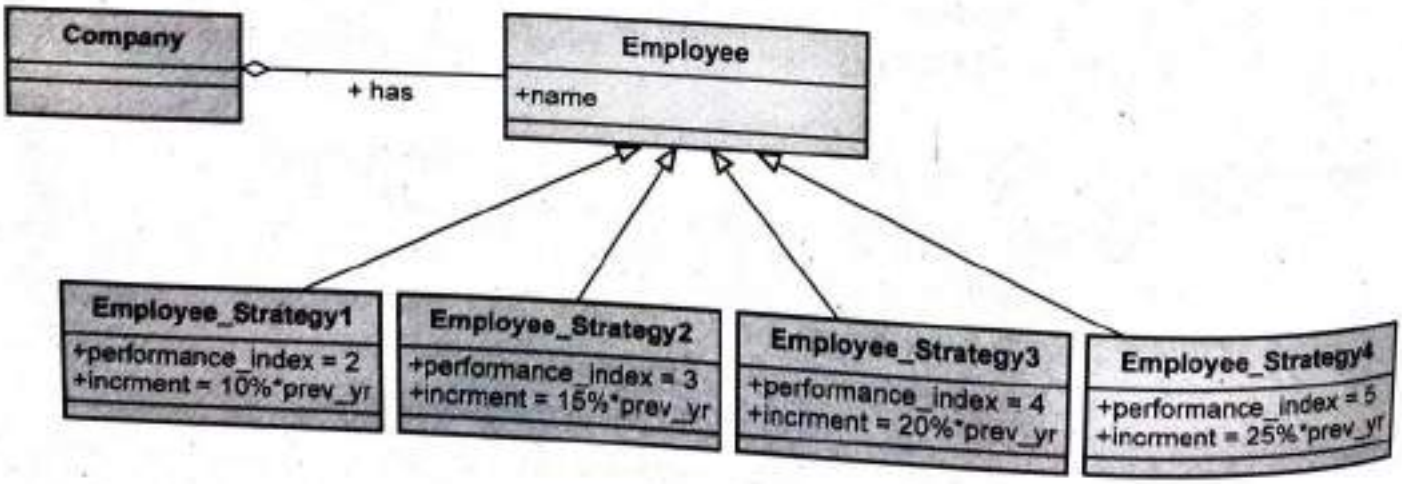


Fig. 3.9.3

3.10 Observer

Intent

The observer pattern is a design pattern that defines a link between objects so that when one object's state changes, all dependent objects are updated automatically. This pattern allows communication between objects in a loosely coupled manner.

Structure and Implementation

Following is an UML diagram which represents how to implement observer design pattern.

Observable interface or abstract class defining the operations for attaching and de-attaching observers to the client. It is also called as **subject**.

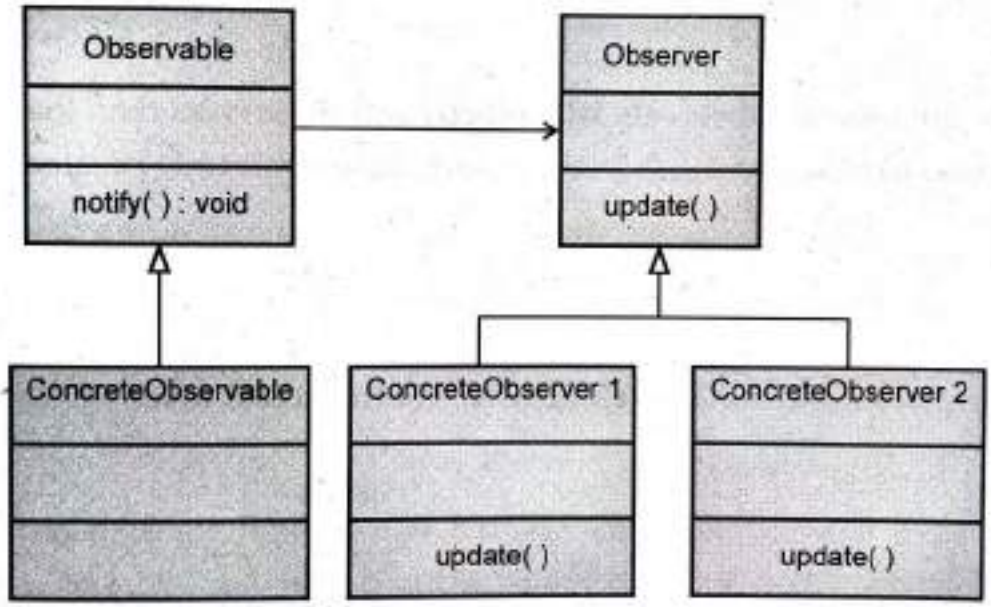


Fig. 3.10.1 Representation of observer

ConcreteObservable is a concrete Observable class. It maintain the state of the object and when a change in the state occurs it notifies the attached **Observer**.

Observer is the interface or abstract class defining the operations to be used to notify this object.

ConcreteObserver1, ConcreteObserver2 are concrete Observer implementations.

The flow of execution is as follows -

The main function instantiate the **ConcreteObservable** object. Then it instantiate and attaches the concrete observers to it using the methods defined in the **Observable** interface. Each time the on the change in state, it notifies all the attached **ConcreteObservers** using the methods defined in the **Observer** interface.

➢ Merits and Demerits

Merits

1. This pattern is useful when there is a change of state in one object that must be reflected in another object.
2. The new observers can be added easily with the minimal changes in the entire code.
3. It allows to send the data to many other objects in very efficient manner.

Demerits

When there are several **Observabels(subjects)** and **Observers** then there are multiple relations that need to be maintained. Due to which the overall code becomes complex.

3.11 Proxy

Intent

Provides the placeholder for another object to control access to it.

Also Known as

Surrogate

Motivation

- In this pattern the class represents the functionality of another pattern.
- This type of pattern belongs to structural design pattern.
- The intention of this pattern is to provide a client for an object to control access to it. Using extra level of indirection controlled and intelligent access can be provided.
- The proxy object gives the client that actual request is handled by the proxy.

Structure

Following UML diagram represents how to implement proxy design pattern.

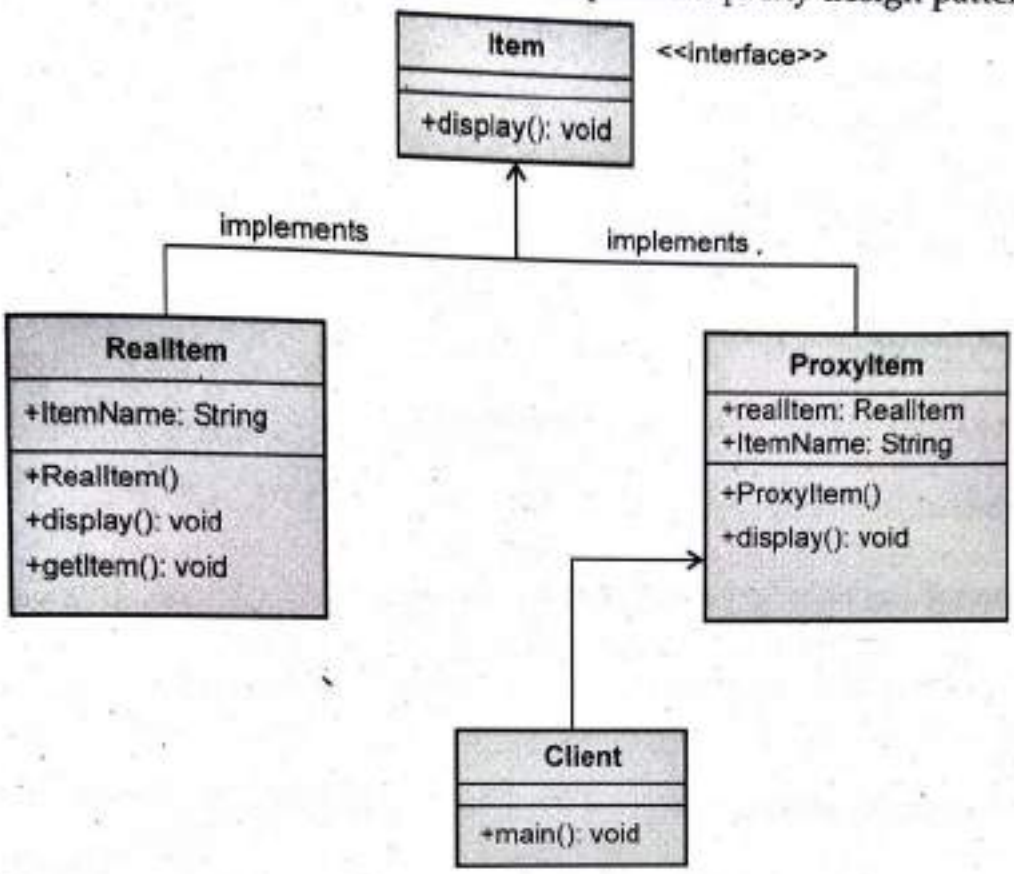


Fig. 3.11.1 Representation of proxy

Implementation

When client needs some item, it will actually use the ProxyItem to get an objet for Item. The **Item** is an interface which is implemented by the concrete classes **RealItem** and **ProxyItem.** The **ProxyItem** is a proxy class which uses the instance of **RealItem.**

The skeleton code for the above representation is as given below

The interface is created as follows

Item.java

```
public interface Item
{
    void display();
}
```

The concrete classes **RealItem** and **ProxyItem** are as follows -

RealItem.java

```
public class RealItem implements Item
{
   private String ItemName;
   public RealItem(String ItemName)
   {
     this.ItemName = ItemName;
     getItem(ItemName);
   }
   public void display()
   {
     System.out.println("Displaying Item : " +ItemName);
   }
   private void getItem(String ItemName)
   {
     System.out.println("Item: " +ItemName+' is obtained");
   }
}
```

ProxyItem.java

```
public class ProxyItem implements Item
{
   private RealItem myItem;
   private String ItemName;
   public ProxyItem(String ItemName)
   {
     this.ItemName = ItemName;
   }
   public void display()
   {
     if(myItem == null)
     {
       myItem = new RealItem(ItemName);
     }
     myItem.display();
   }
}
```

Client.java

```
public class Client
{
  public static void main(String[] args) {
  Item item = new ProxyItem("MyFavorite Item");
//item will be obtained using getItem first and then will be displayed
```

```
item.display();
}
}
```

Application

The **payment** made by the customer by **cheque** or by bank **demand draft** is proxy for the actual cash present in the account. A **cheque or DD can be used in place of cash** for making purchases and ultimately controls access to cash in the issuer's account.

➢ Merits and Demerits

Merits

1. Security is the most primary advantage of this design pattern. The main object remains protected from the client access by providing the proxy instance.
2. The duplication of the object can be avoided, this saves the system's memory space and also increases the performance of application.

Demerits

The conflicts in the behavior of the system may occur if one client directly accesses the real object and another client accesses the proxy object.

Review Question

1. *Explain the proxy designer pattern with suitable example.*

3.12 Façade

The Façade design pattern is a structural design pattern. It is evolved from the French word façade which means "frontage" or "face".

It defines the high-level interface that makes the subsystem easier to use.

Intent

- Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.
- Wrap a complicated subsystem with a simpler interface.

Motivation

To simplify the complex system architecture by unifying it with different interfaces, the Façade object is used. The Façade shields the complexity of the system from the client.

Structure

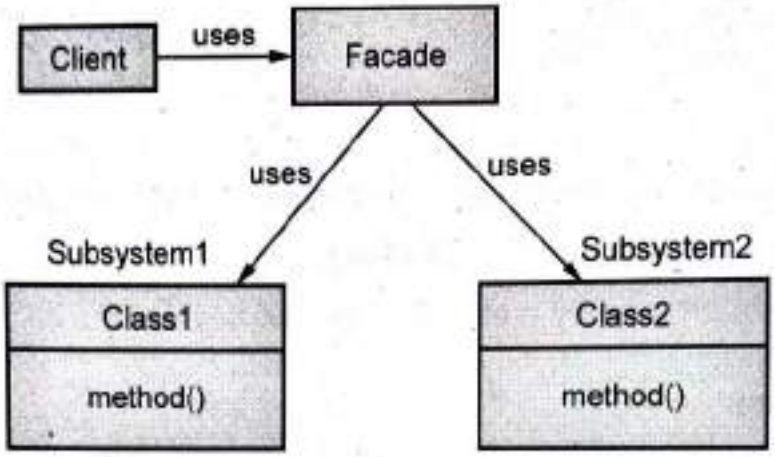


Fig. 3.12.1

Example

Consider a scenario where a customer has some issues with the product he/she has purchased. So the customer calls makes a call on customer care number and speaks with a customer service representative. The customer care service person acts as a Facade, providing an interface to the order fulfilment. The underlying system may include a team of service engineers. The corresponding service engineer of that region is then assigned a duty to look after the complaint raised by the customer and solve the complaint. The order of execution will be -

1) Customer calls the **ProductServiceSystem**
2) The customer care service person then hands it over to a subsystem that assigns the appropriate service engineer.
3) The service engineer visits the customer and solves the issue raised by the customer.

Implementation

```
public class ProductServiceSystem(){
   public void assignServiceEngineer(){

   }
}
public class ServiceEngineer() {
   public void getRequest(){

   }
   public void fulfillRequest(){

   }
}
```

```
//Facade
public class CustomerCareServicePerson(){
  private ProductServiceSystem service;
  private ServiceEngineer engineer;
  service.assignServiceEngineer();
  engineer.getRequest();
  engineer.fulfillRequest();
}
```

➢ Merits and Demerits

Merits

1) It hides much of the complexity and makes the subsystem easy to use.
2) It helps to have the principle of loose coupling, so changes can be made to the system without affecting the clients.

Demerits

1) It incorporates the additional level of indirection.
2) There is high degree of dependence on façade interface.
3) Clients cannot access underlying classes and certain functionalities might be unavailable to clients.

Review Question

1. *Explain the intent, motivation, structure, implementation, merits and demerits of Facade design pattern.*

3.13 Role of Architecture

- Architecture is a design of a system which gives a high level view of the parts of the systems and focus on the relationship among these subsystems.
- There is no unique structure of the system which can be described by the architecture. In fact there are various types of structures. Hence the architecture can be formally defined as

Definition : Software architecture of the system is a structure or structures of the system which consists of software elements and the externally visible properties of those elements and relationships among them.

- According to this definition the elements that have some important properties are considered as the main elements. The behaviour of these elements is a part of architecture.
- The uses of the software architecture descriptions are -

1. Understanding and communication

- The software architecture description is for communicating the architecture of the system to its users, developers and to the architects. This description helps these stakeholders(users, developers and architects) to understand the micro properties of the system and how the system satisfies the functional and quality requirements.
- The architecture description of proposed system will describe how the system will be composed and what the parts of it are. This will help in understanding the proposed system.
- The architecture description is also useful for understanding the existing system.

2. Reuse

- Software architecture helps in reusing the software. The software reuse help in increasing the productivity and decreasing the cost of software.
- Software engineers are working for developing such software components that will be useful for building any desired software system. Then architectures are chosen in such a manner that these components will be reused effectively.
- If there are similar types of products then architectures suggest the components that can be reused in similar products.

3. Construction and evolution

- For construction of the system, the **partitioning** is done.
- Architecture partitions the system into parts. Suitable partitioning in the architecture helps the project to build the system part by part. Then each part can be independently developed and built by separate group of developers. Thus architecture guides the development process.
- After construction of the complete system sometimes **new features** need to be **added** or some **changes** must be done in the existing system.
- It is the architecture which helps in deciding where to add the new features with minimum complexity and effort.
- If some changes need to be incorporated in the system then architecture decides which part of the system will get affected by the changes. The changes should be made carefully without disturbing the overall functionality of the system.

4. Analysis

- Before building the actual system some important properties of the system can be determined. The software engineers use the models to analyze the design of a product for its cost, reliability and performance. Architecture helps in determining such models.
- The properties of the system that has to be build can be predicted if the system is developed from the architecture. Thus architecture description helps in analyzing the system before its actual development.

Review Question

1. *What is software architecture ? Also explain the uses of software architecture description.*

3.14 Architectural Styles

- Architectural style is a **pattern** for creating the system architecture for the given problem. Most of the large systems are **heterogeneous** and do not follow a single architectural style.
- These styles can **provide ideas** for creating an architectural view for the given problem.
- Styles can be combined to form **richer views**.
- Many systems use multiple styles and different parts of a system may use different styles.
- Various architectural styles are -
  - Layered architectural style
  - Client Server style
  - Tiered architecture
  - Pipe and Filter style

3.15 Layered Architectural Style

- The layered architecture is composed of different layers. Each layer is intended to perform specific operations so machine instruction set can be generated. Various components in each layer perform specific operations.

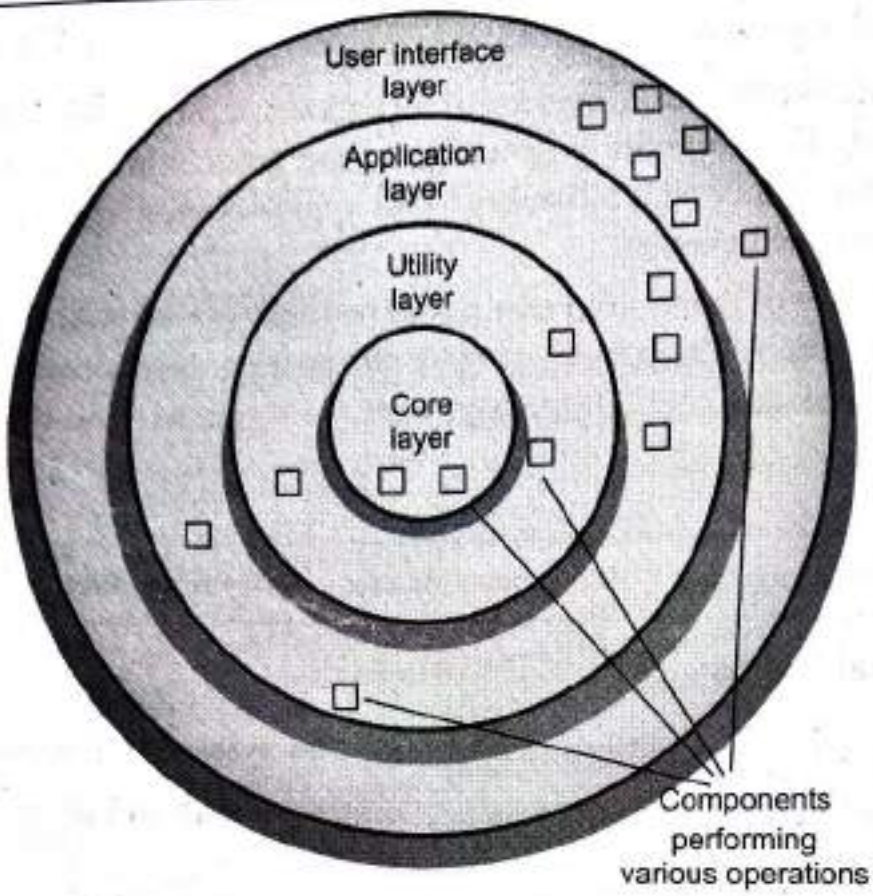


Fig. 3.15.1 Layer architecture of components

- The outer layer is responsible for performing the user interface operations while the components in the inner layer perform operating system interfaces.
- The components in intermediate layer perform utility services and application software functions.

3.16 Client Server Style

- In this style, there are two component types **client** and **server**.
- The client can communicate to the server but not to another client.
- The communication between the client and server is initiated by the client. The client sends the request to the server and for service and the server supports.
- The server receives any request from the client on a specific port. Then it builds the requested service and then the result is returned to the client.
- There is one type of connector used in this style and that is **request and reply** type. The task of this connector is to connect the client with the server. This type of connector is **asymmetric**. The client end of the connector makes the request and the server end of connector the response is sent. The communication is synchronous in nature. That means client makes a request and wait for the response and it gets blocked until the request gets a replay.

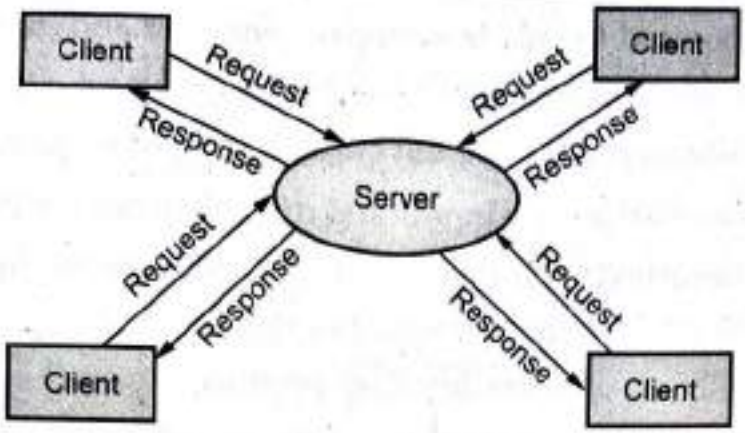


Fig. 3.16.1 Client Server architecture

Review Question

1. *Explain the client server architecture.*

3.17 Tiered Architecture

- The general form of client server architecture is **n-tier** structure. The client sends a request to the server and in order to serve the request the server further sends this request to some another server.
- The most commonly used structure is **3-tier** architecture. In this architecture the client makes a request to the server and receives the response from the server. This is a **presentation tier**. Then the server processes this request and computes the result. This is a **middle tier** which is called **business logic tier**. The third tier is the **database tier** in which the data resides. The business tier interacts with the database tier for accessing the required data.

  Refer Fig. 3.17.1.

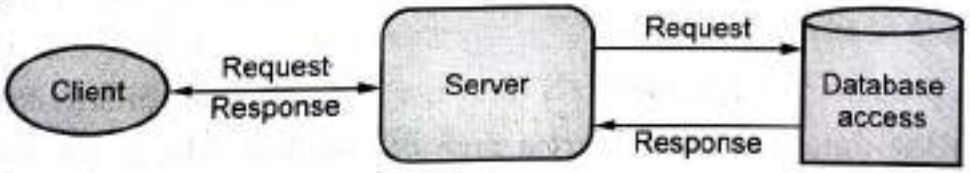


Fig. 3.17.1 Client Server architecture

For the client server communication usually the read write connector is used but many client-server systems today support the TCP ports as the connectors. For the Web applications HTTP protocol is used to support the connector.

3.18 Pipe and Filter

AU : Dec.-20, May-22, Marks 13

- In this architecture series of transformations are applied to produce the output data.
- There is only one type of component used in this architectural style and that is

Filter which is connected to only one type of connector and that is **Pipe**. The data is transformed from one filter to another using the pipes.

- The filters work independently without bothering the other filter.
- A filter may have more than one inputs and more than one outputs.
- Filters are only concerned with the data coming to them as input through pipe. A filter need not know the identity of the filter that sent the input data or the identity of the filter that will consume the data they produce.
- Pipe does not change the data which travel through them. These are **unidirectional** entities which transfers the data received at one end to another.

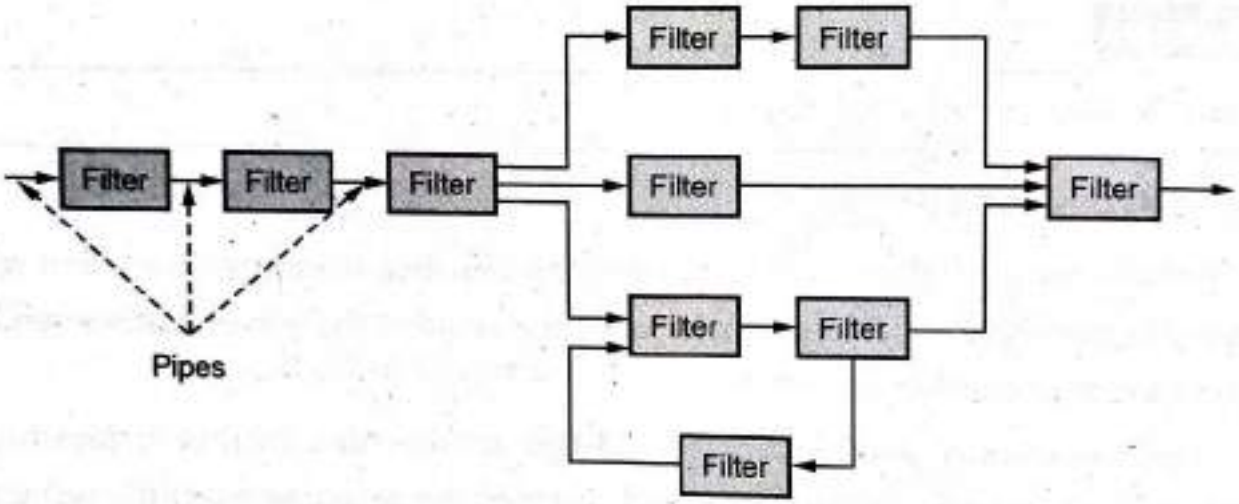


Fig. 3.18.1 Pipes and Filters

- There are some constraints on this architectural style imposes -
  1. The filters should work without the knowledge of producer (the filter who sends the data) or consumer (the filter who receives the data).
  2. A pipe is a **two-way connector**. It must connect an output port of a filter to an input port of another filter.
  3. In pure pipe and filter structure a filter has independent thread of control which process the data and to support a pipe mechanism which transforms the data the **synchronization** needed.
- Pipe and filter architectural style is well suited for **data processing and transformation**.
- In **text processing applications** and **Signal processing applications** this style proves to be well suited. Also it is useful in the applications that typically perform encoding, error correction and other transformations on the data.

Review Questions

1. *Explain pipe and filter architectural pattern.*
2. *What are the different types of architectural styles exist for software and explain any software architecture in detail ?* **AU : Dec.-20, Marks 13**
3. *Bring out the importance of architectural design. Explain any two software architectural styles with suitable example.* **AU : May-22, Marks 13**

3.19 User Interface Design

AU : May-16,13,22, Dec.-08,09,17,15,16, Marks 16

Any computer based system requires two things: its **computational ability** and **functionality**. The computer based systems are typically used by casual users. Hence the purpose of user interface design is to have effective communication medium between the computerized system and the user. This kind of interface design is necessary because of many reasons such as : software is difficult to use, the use of software forces the user to make mistakes due to lack of understandings about the system.

While designing for user interface the very first step is to identify **user, tasks** and **environmental techniques**. Based on user tasks, **user scenarios** are prepared. Such user scenarios help to design user interface objects and corresponding actions. This set of objects and actions help in deciding the **screen layout**. Once such a layout has been prepared, appropriate icons, screen texts and menu items can be placed at respective positions.

Advantages :

Following are the advantages of having user interface system.

1. The user interface systems provide fast and intuitive interactions to the user. The new user can easily operate the system.
2. As in user interface design, menus are provided, it avoids typing mistakes and very little typing is required.
3. The user interface helps in simple data entry.
4. It provides the flexibility in switching from one task to another.

All the above mentioned advantages of user interface design helps in building the flexible software product. And therefore the sale for software product can be increased.

3.19.1 Golden Rules

Thao Mandel has proposed three golden rules for user interface design -

1. Place the user in control
2. Reduce the user's memory load
3. Make the interface consistent.

Let us discuss each rule in detail -

3.19.1.1 Place the User in Control

While analysing any requirement during requirement analysis the user often demands for the system which will satisfy user requirements and help him to get the things done. That means the user always wants to control the computerized system. Following are the design principles that allow user to control the system :

- **Define the interaction modes in such a way that user will be restricted from doing the unnecessary actions.**

  Interaction mode means the current state in which user is working and being in such a mode user is supposed to do the related tasks only if the user has to perform unnecessary actions at such time then the GUI becomes frustrating.

- **The interaction should be flexible.**

  The user interaction should be flexible. For example in the Microsoft power point slide show the slide transition is possible using mouse clicks as well as using keyboard. Such flexibility allows any user to operate the system as per his comfort.

- **Provide the facility of 'undo' or 'interruption' in user interaction.**

  This is the feature which allows the user to correct himself whenever necessary without loosing his previous work. For example : In the software like 'Paint' one can draw some objects and perform 'redo' or 'undo' actions for performing his desired drawing.

- **Allow user to customize the interaction.**

  It is observed that in while handling user interface certain actions need to be done repeatedly. It saves the time if these actions are collected in a Macro.

- **Hide technical details from the user.**

  This feature is essential for a casual user. User should not be aware of system commands, operating system functions or file management functions. For example while printing some document instead of giving the command for printing if user clicks on the icon indicating print then it becomes convenient for a casual user to take the print out.

- **The objects appearing on the screen should be interactive with the user.**

  This also means that user should be in position to adjust the object appearing on the screen. For example in 'Paint' one should be able to stretch the oval or edit the text written in the object.

3.19.1.2 Reduce the User's Memory Load

If the user interface is good then user has to remember very less. In fact the design should be such that the system remembers more for the user and ultimately it assists the user to handle the computer based systems. Following are the principles suggested by Mandel to reduce the memory load of the user.

1. **Do not force the user to have short term memory.**

   When user is involved in complex tasks then he has to remember more. The user interface should be such that the user need not have to remember past actions and results. And this can be achieved by providing proper visual interface.

2. **Establish meaningful defaults.**

   Meaningful default options should be available to the user. For example in the Microsoft word the user can set the default font size as per his choice. Not only this, there should be some reset option available to the user in order to get back the original values.

3. **Use intuitive shortcuts.**

   For quick handling of the system such short cuts are required in the user interface. For example control+s key is for saving the file or control+o is for opening the file. Hence use of meaningful mnemonics is necessary while designing an interface.

4. **The visual layout of the interface should be realistic.**

   When certain aspect/feature of the system needs to be highlighted then use of proper visual layout helps a casual user to handle the system with ease. For example in an online purchase system if pictures of Master card/Visa card are given then it guides the user to understand the payment mode of online purchasing.

5. **Disclose the information gradually.**

   There should not be bombarding of information on user. It should be presented to the user in a systematic manner. For instance one can organize the information in hierarchical manner and narrate it to the user. At the top level of such hierarchical structure the information should be in abstract form and at the bottom level more detailed information can be given.

3.19.1.3 Make the Interface Consistent

The user interface should be consistent. This consistency can be maintained at three levels such as,

- The visual information (all screen layouts) should be consistent throughout and it should be as per the design standards.

- There should be limited set of input holding the non conflicting information.
- The information flow transiting from one task to another should be consistent.

Mandel has suggested following principles for consistent interface design.

1. **Allow user to direct the current task into meaningful manner.**

   This principle suggests that create a user interface with proper indicators on it so that user can understand his current task and how to proceed for new task.

2. **Maintain consistency across family of product.**

   If an application come in a packaged manner then every product of that application family should posses the consistent user interface. For example Microsoft Office family has several application products such as MS WORD, MS Power Point, MS Access and so on. The interface of each of these product is consistent (of course with necessary changes according to its working!). That means there is a title bar, menu bar having menus such as File, Edit, View, Insert, Format, Tools, Table, Window, Help on every interface. Then tool bars and then design layouts are placed on the interface.

3. **If certain standards are maintained in previous model of application do not change it until and unless it is necessary.**

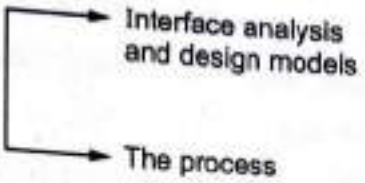
   Certain sequence of operation becomes a standard for the user, then do not change these standards because user becomes habitual with such practices. For instance control + s is for saving the file then it has become a standard rule now, if you assign different short cut key for saving then it becomes annoying to the user.

3.19.2 User Interface Analysis and Design

User interface analysis and design can be done with the help of following steps.

- Create different **models** for system functions.
- In order to perform these functions identify the human-computer interface **tasks**.
- Prepare all **interface designs** by solving various design issues.
- Apply modern tools and techniques to **prototype** the design.
- **Implement** design model.
- **Evaluate** the design from end user to bring quality in it.

These steps can be broadly categorized in two classes.
Let us discuss each of them in detail –



3.19.2.1 Interface Analysis and Design Model

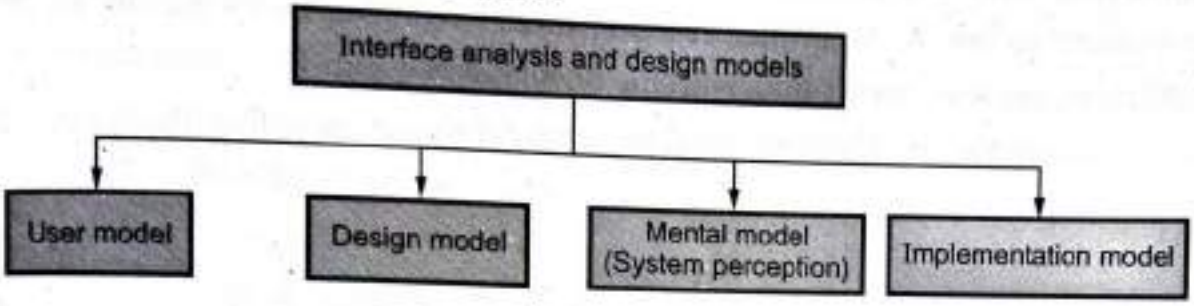


Fig. 3.19.1 Analysis and design model

Typically there are four types of models that can be prepared in interface analysis and design. Let us understand each of them

- User model - To design any user interface it is a must to understand the user who is using the system. Hence in this model the profile of user is prepared by considering age, sex, educational, economical and cultural background. The software engineer prepares user model. Globally there are three kinds of users

User	Description
Novice	The user with very little knowledge of the computer who simply knows semantics (simply the wants) of the system and does not know the implementation of such system.
Knowledgeable and intermittent	The user having knowledge about the semantics of the system as well as having little knowledge of syntactic of the system.
Knowledgeable and frequent	The user with good semantic as well as syntactic knowledge of the system.

- Design model - It consists of data, architectural, interface and procedural representation of the software. While preparing this model the requirement specification must be used properly to set the system constraints. Software engineer prepares the design model of the interface.

- Mental model (system perception) - The user model is the representation of what **user thinks** about the system? Basically any user interface design is heavily dependant upon the description obtained from the user about his wants and needs. If the user is knowledgeable then more complete description of the system can be obtained than that of novice user.
- Implementation model - The implementation model generates the **look and feel** of interface. This model describes the system's semantic and syntax. It is very necessary to match the implementation model with the user's mental model then only user can feel comfortable with the developed system.

Finally the interface designer has to resolve any differences within these models. The supreme principle that has to be followed in interface analysis and design method is that know the user and know the task!

3.19.2.2 The Process

The user interface analysis and design process can be implemented using iterative spiral model. It consists of four framework activities.

1. Environment analysis and modelling
2. Interface design
3. Implementation
4. Interface validation

As shown in the below figure each of these tasks can be performed more than once. At each pass around the system more requirements can be elaborated and detailed design can be performed.

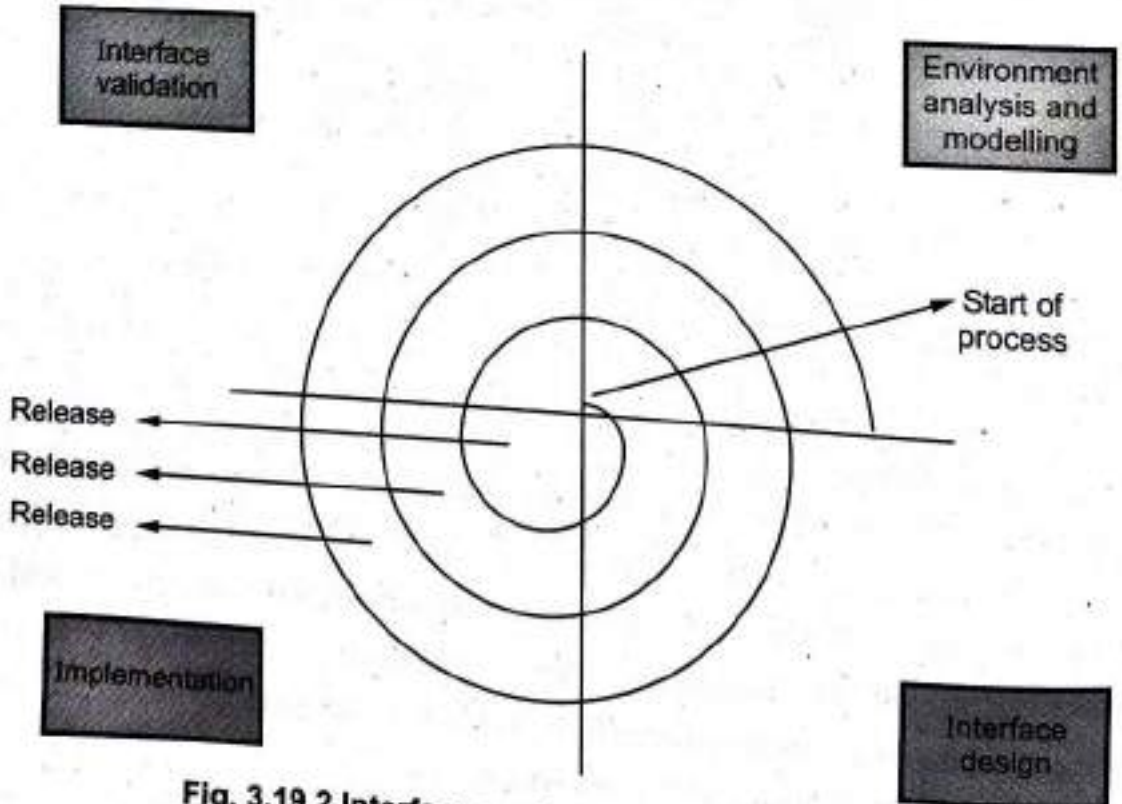


Fig. 3.19.2 Interface analysis and design process

Environment analysis and modelling - In this phase three major factors are focused i.e. user, task and environment. First of all the user profile is analysed to understand the user and to elicit the requirements, then the tasks that are required to carry out desired functionality are identified and analysed. The analysis is made on user environment which involves the physical work environment. Finally analysis model is created for the interface. This model serves as a basis for the design of user interface.

Interface design - The interface design is a phase in which all the interface objects and corresponding actions of each task are defined.

Implementation - The implementation phase involves creation of prototype using which the interface scenarios can be evaluated. To accomplish the construction of user interface some automated tools can be used

Validation - The goal of validation is to validate the interface for its correct performance. During validation it is tested whether all the user requirements get satisfied or not. The purpose of validation is also to check whether the interface is easy to learn and easy to use.

3.19.3 Interface Analysis

- Before preceding for interface design it is necessary to understand the problem.
- Understanding the problem means understanding -
  1. The people or user who actually interacts with the system.
  2. The task that are performed by the end user for interacting the system
  3. The contents of the interface that will be displayed to the user.
  4. The environment in which the task will be conducted.

3.19.3.1 User Analysis

- Following are the ways by which one can learn **what the user wants from the user interface -**
  - **User interviews :** This is the most effective technique in which some representatives from software team meet the end user to better understand the user needs, motivations and many other issues. Meetings are conducted for this purpose.
  - **Sales input :** Sales people interact with the users regularly and collect the information. Based on this information the users are categorized in particular groups and thereby their requirements are better understood.
  - **Marketing input :** Market analysis is made in order to understand the usage of software.

- **Support input :** Support staff keeps a regular interaction with the user interaction for knowing the certain things like the likings and dis-likes of the users, which features are easy to use and so on.

- Following is a set of **questions** that will help the **interface designer** better **understand the users** of a system -

1. Are user trained professionals, technicians, or worker ?
2. Are the users capable of learning from written material ? OR they require any classroom training ?
3. What is the formal education of the user ?
4. Are users aware of using keyboard ?
5. What is the age group of user ?
6. Will user be represented dominantly by one gender ?
7. Does the user work in office hour or do they work until the job is done ?
8. What kind of compensation will be given to the user for the work they perform ?
9. Will the user make use of the software frequently or occasionally ?
10. What is the primary spoken language among the users ?
11. What will happen if the user makes a mistake in handling the system ?
12. Are the expert users addressed by the system ?
13. Do users want to know the technology used behind the interface ?

- The answers to these questions help in understanding the end-user.

3.19.3.2 Task Analysis and Modelling

- In task analysis following questions are answered

1. In specific situation what work the user should perform ?
2. When user performs the work what are the tasks and sub tasks that should be performed ?
3. When user performs the work, what are all those problem domain objects that user will manipulate ?
4. What is the hierarchy of the task ?
5. What is the workflow for accomplishing the task ?

- In order to answer these questions following techniques are used -

1. **Use-cases :** Use cases describe the manner in which the actor interacts with the system. Use cases always show how the end user performs specific work related task. Use cases are mostly written in informal way i.e. in paragraphs.
2. **Task elaboration :** For task elaboration the functional decomposition or the stepwise refinement of the processing task is done. The task analysis for interface design adopts **elaborative approach.** During the task elaboration, identification and classification of tasks is performed.
3. **Object elaboration :** The software engineers examine the use case and the other information obtained from user and extract the information about the physical objects. These objects are further classified into classes. The attributes of each class are defined and evaluation of actions will identify the list of operations.
4. **Workflow analysis :** When there are many users who are simultaneously using the system then for understanding the flow of the series of tasks conducted the work flow analysis technique is applied. Workflow processes can be easily modelled using swimlane diagram. In swimlane diagram, the modelling is done for the classes who are responsible for carrying out the activities.

See Fig. 3.19.3 on next page.

As shown in Fig. 3.19.3 the reservation system makes use of some important classes such as Booking service, Passenger service and Finance service. Following are the set of activities -

- Passenger makes enquiry about reservation.
- He has to fill up the form. The passenger will fill the up the information such as source and destination city, date and time of travel, number of passengers and type of reservation.
- If the desired seats are available then he gets the confirmation for the reservation otherwise he will be put in the waiting list.
- If the confirmed reservation is made, the passenger pays for the ticket.
- The finance service will receive the payment.
- The booking service will issue the tickets.

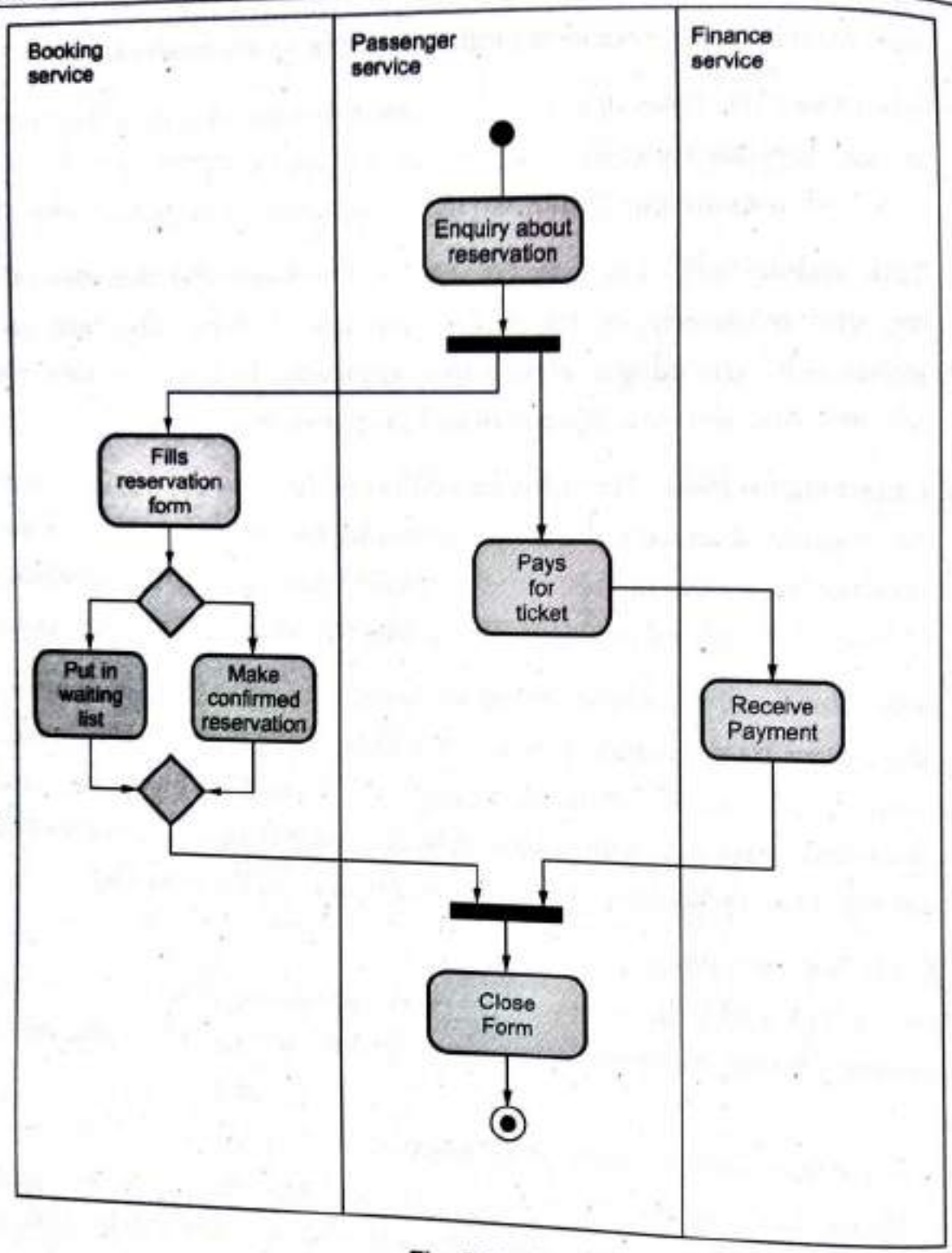


Fig. 3.19.3

5. **Hierarchical representation :** In workflow analysis **process elaboration** occurs. After establishing the workflow analysis, the task hierarchy for each user type should be specified. In this hierarchical representation, there is a stepwise representation of each task identified for each user. For example :

Fills reservation form

- **Provide the personal information**
  - **i. Specify name(s)**

ii. Specify age

iii. Desired source city

iv. Desired destination city

v. Date and time

Reservation details

i. Name(s)

ii. Age

iii. Source city

iv. Destination city

v. Date and time of train

vi. Seat numbers

vii. Type of reservation

viii. Name of the train

3.19.3.3 Analysis of Display Content

- The contents that get displayed as an user interface are called the **display contents.**
- The display contents can be spreadsheets, 3D models, pictures or images, a graph or specialised information such as audio or video files.
- During this analysis step, **the format and aesthetics of the display contents** are considered.
- For determining the format and the aesthetics of the contents following questions can be asked or answered.
  a. Is it possible for the interface to display various types of data in a consistent manner on geographic location ?
  b. Is it possible to customize the screen location with respect to the contents ?
  c. If the large report is partitioned properly for the sake of understanding ?
  d. Is the summary of large amount of data available directly ?
  e. Will the graphical output be displayed properly so that it will fit to the display device ?
  f. Is there a use of sophisticated color combination?
  g. Are the errors and warning messages presented to the user in interactive manner ?

- Answers to these questions will help in gathering the requirements for the presentation of the contents.

3.19.3.4 Analysis of Work Environment

- The analysis of work environment is very important.
- In some applications the user interface for the computer based systems is placed in very user friendly environment. In such a situation, there may be interactive presentation of the information, proper lighting, good display height, easy keyboard access, proper sitting arrangement and so on. In such work environment using the application becomes an enjoyable activity.
- But there are some workplaces in which there may be insufficient light, lot of noise and distractions, unavailability of keyboard or mouse interfacing and so on. Using the application in such environment becomes difficult and frustrating.
- In addition to these physical factors, consideration of work place culture is extremely important. These factors are raised by following questions -
  - Will more than one person access the same information before providing the input ?
  - Will the system interaction be measured in terms of some measure. For instance : Time required for processing of data.
  - Will the system provide some kind of support to the user for handling it ?
- These issues must be solved before the completion of interface design phase.

3.19.4 Interface Design

After interface analysis all the tasks and corresponding actions are identified. Interface design is an iterative process in which each design process occurs more than once. Each time the design step gets elaborated in more detail. Following are the commonly used interface design steps -

1. During the interface analysis step **define** interface **objects** and corresponding **actions** or operations.
2. **Define** the major **events** in the interface. These events depict the user actions. Finally **model** these events.
3. **Analyse** how the **interface** will look like from user's point of view.
4. **Identify** how the **user understands** the interface with the information provided along with it.

While designing the interface software engineer communicates the user and according to his thinking about the interface, software engineer draws the sketches. Get it approved from the user and then work on defining objects and corresponding actions. While designing the interface the designer has to follow -

- Golden rules
- Model the interface
- Analyse the working environment

Let us now discuss with some example how to apply these design steps -

3.19.4.1 Application of Interface Design Steps

As we know the first step in any interface design is to **identify** all the necessary **objects and corresponding actions**. The use cases are used for this purpose. That is, the use case description is parsed and the nouns and verbs from this description is identified. The nouns correspond to objects and verbs correspond to actions. Thus a list of objects and actions is prepared.

There are three types of objects

1. *Target object* - The target object is an object in which some object can be merged.
2. *Source object* - The source object is an object which can be dragged and dropped to some other object.
3. *Application object* - The object which represents the application specific data.

After identifying all the necessary objects and actions the screen layout can be prepared. The creation of screen layout includes placing of useful icons, descriptive text, menus and windows. This layout should resemble the real world description of the application.

For example : Online student registration

Some part of problem description is given below for online student registration system.

"A **student** will be typical user of this system. This user has to fill up an online **registration form**. In this form he has to submit the **student information** and then he has to select the **courses** which he/she wants to adopt. Then the **Time Table** of the corresponding course will be displayed to the user. There should be a facility that student can get a print of time table. Then student will be given some **Student ID**, he then has to set the **password** for this ID. If a **teacher** wants to find the particular student teacher will put the student ID and can get the complete information of him."

In above problem description the boldface words indicate the objects and the underline words represent the verbs using this information we can design the user interface as shown below

"A **student** will be typical user of this system. This user has to fill up an online **registration form**. In this form he has to submit the **student information** and then he has to select the **courses** which he/she wants to adopt."

Then the **Time Table** of the corresponding course will be displayed to the user. There should be a facility that student can get a print of time table.

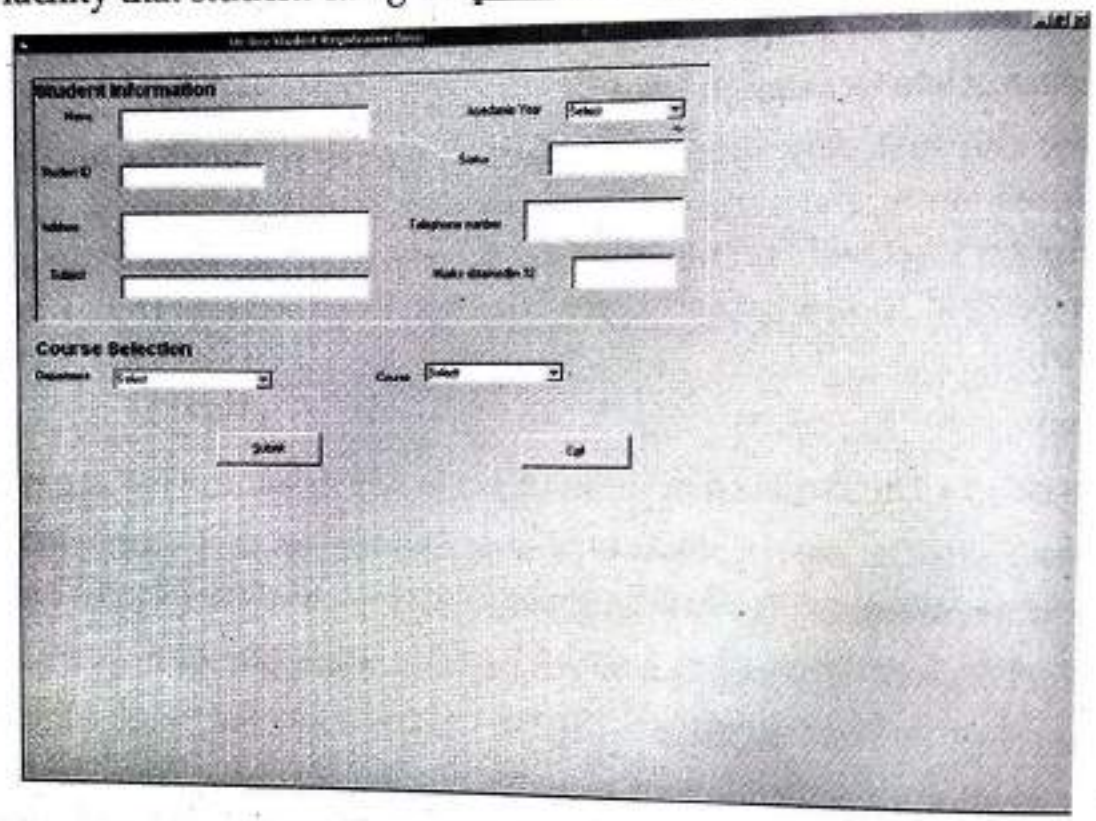


Fig. 3.19.4

If a **teacher** wants to find the particular student teacher will put the student ID and can get the complete information of him.

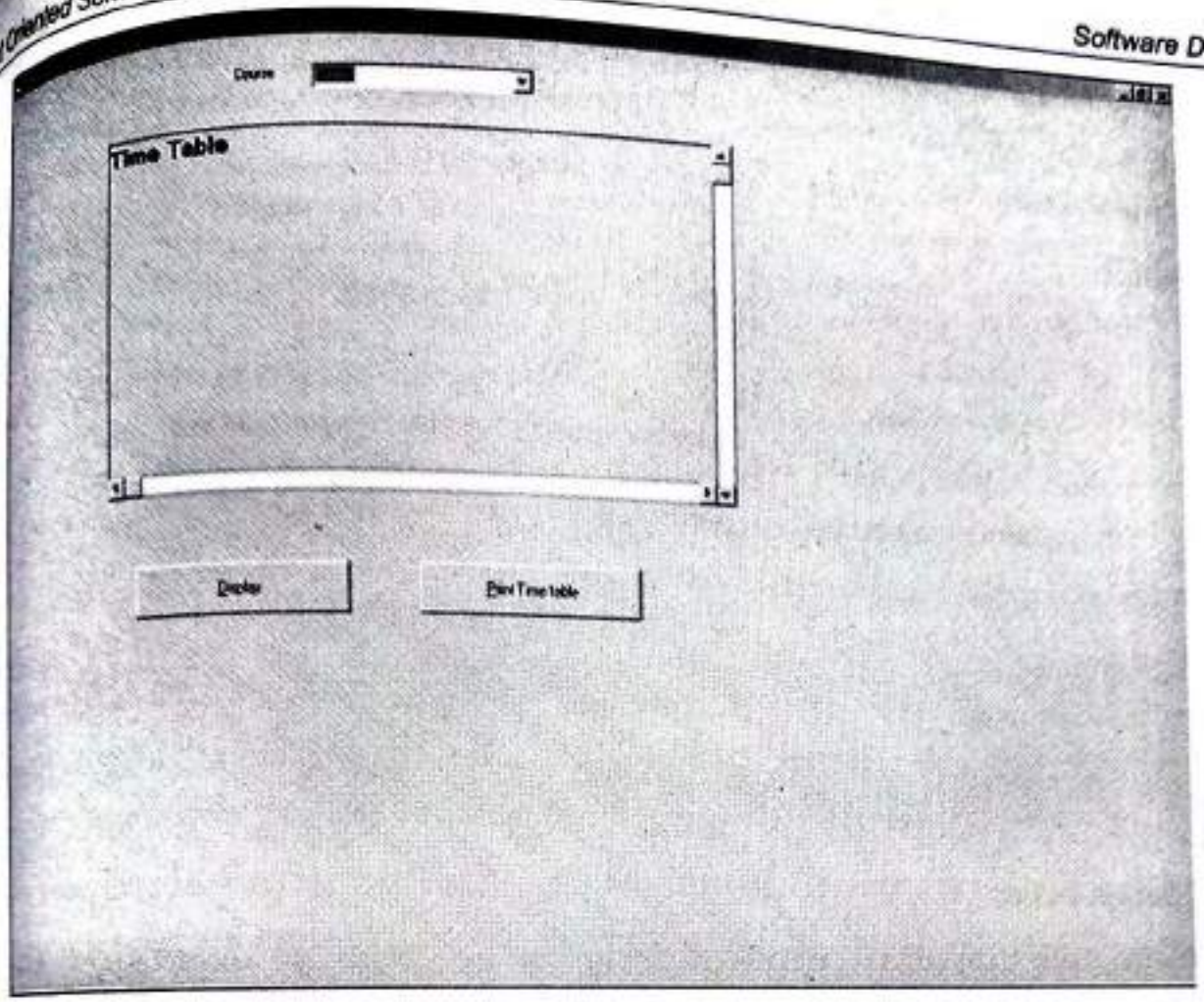


Fig. 3.19.5

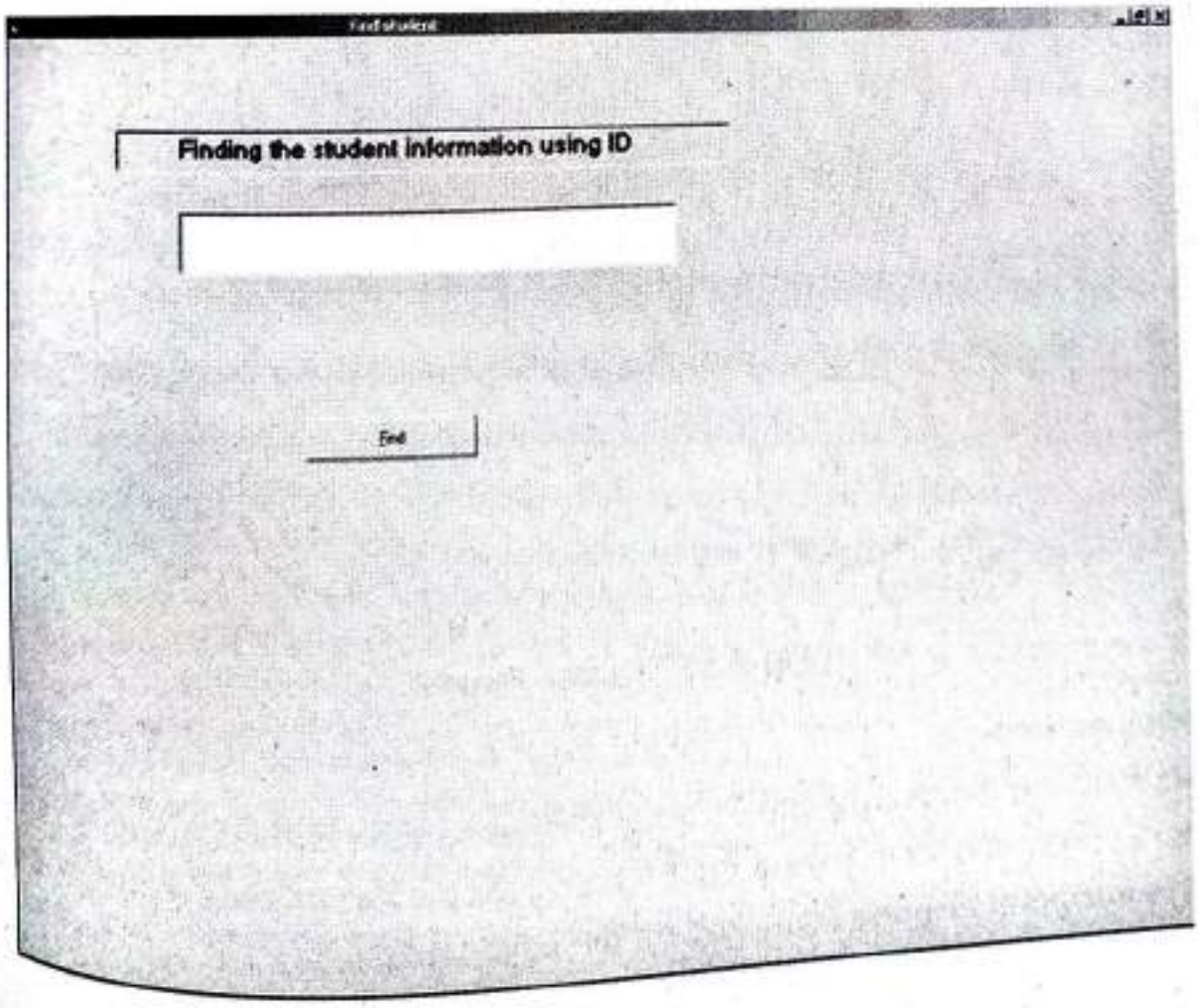


Fig. 3.19.6

3.19.4.2 User Interface Design Pattern

Today it is a common practice to include some user interface design patterns. These design patterns serve as a solution to specific design problem.

For example

Pattern	Meaning
Progress indicator	We found this type of pattern in any installation program. By this pattern user gets a feel that some process is progressing continuously behind the screen.
Wizard	This pattern serves as a guideline for completion of task through various windows.
Shopping cart	It is typically used in on-line purchasing system. It provides the list of items that can be purchased.

3.19.4.3 Design Issues

There are four important interface design issues that are depicted by following Fig. 3.19.7.

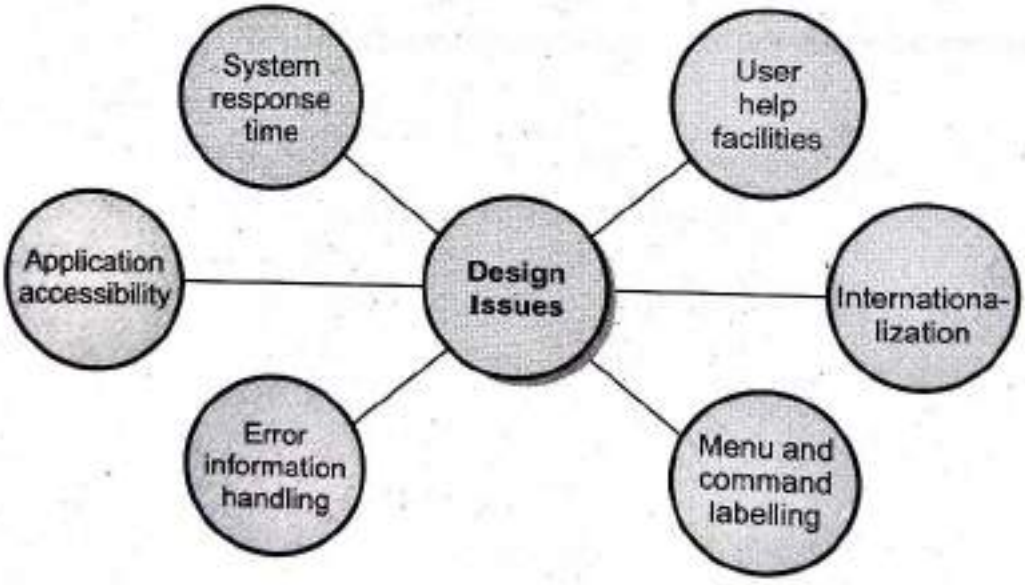


Fig. 3.19.7 Design issues

System Response Time

Any user hate the delayed response time. Response time is the amount of time taken by the system to respond from the point at which user performs some control action (such as clicks on some point or press some key on the keyboard).

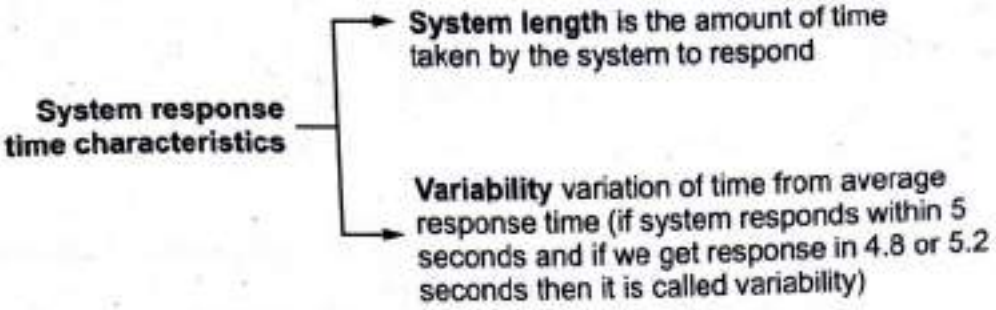


Fig. 3.19.8

Help facilities : This is the most essential criteria for any user interface this makes the system more interactive. The help can be **on-line help** or it can be in the form of **user manual**. Even some software provide the help in the **assistance** form i.e. the question may be asked by the user and the answer given by the system will serve as a help to the user.

Error handling : Errors and warning cause the frustrations to the user. Sometimes the error messages are in very vague form. For example : Application

ABC has encountered error 1033. Now such type of error messages do not direct the user about what went wrong. Following are the characteristics for presenting errors.

1. The error message should be in a language which user can understand.
   For example : Remote server not responding.
2. There should be some useful message along with error in order to recover from the error. For example : Press enter to exit
3. There should be some audible or visual cue along with the error message. For example: beep sound or highlighting back ground of the text.
4. There should not be any negative impact of error on the user. For example : File XYZ.SYS has been deleted such error will cause frustration on the user.
5. The wording of the error should be carefully used and it should not blame user. (because nobody likes to get criticized!!)

Thus error messages should be so effective that they should bring qualitative interaction between the user and the system.

Menu and command labelling : There are typically two ways by which the user handles the system i.e. **keyboard** and **mouse**. The system can be handled using commands by the power user(i.e. knowledgeable and frequent users) whereas any ordinary user makes use of GUI and he prefers not to use any command as such. There are number of design issues for typed command and menu labels.

Command related design issues	Menu label related design issues
In which form the command will be ?	Will there be any menu for corresponding command ?
Will the commands be difficult to learn and remember ?	Are the menus self explanatory ?
What to do if the command is forgotten ?	Is there any consistency between menu and submenus
Is there any abbreviation for the command ? Or can they be customized ?	

Application accessibility :

- In modern era, Computer application are used by everybody and every where. Hence software engineers must develop a mechanism by which the most frequently required applications must be availed easily.
- The software is most important entity for the users who are physically challenged. This can be used by them for moral, legal and business reasons.
- Accessibility guidelines are used for while developing the applications. These guidelines are mostly used by the web applications. The guideline assist in designing the interfaces used within the application.
- The accessibility guideline also provides the guideline for assistive technology that addresses the needs of those visual, hearing, speech and mobility impairments.

Internationalization

- There are situations in which the user interface is developed for localised purpose and for local language. If the same interface is required in other side of the country then existing interface is used with more or less modifications.
- The real challenge is to develop **globalised** software. That means the user interfaces should be designed to accommodate a generic core functionality. This functionality can be used by any user belonging to any country.
- Localized user interface is useful to only localised market.
- There are many internationalization guidelines that are available for software developers.
- These guidelines address the design and implementation issues.
- The **Unicode standard** has been developed to handle the challenges of managing the natural languages with lots of characters and symbols.

Review Questions

1. *List the activities of user interface design process.* AU : May-13, Dec.-08, Marks 8
2. *Discuss about user interface design of a software with an example and neat sketch.* AU : Dec.-15, Marks 16, Dec.-17, Marks 13
3. *Describe the golden rules for interface design.* AU : Dec.-16, Marks 8
4. *Write short note -User-interface design.* AU : May-16, Marks 4
5. *Explain the core activities involved in user interface design process with necessary block diagram.* AU : Dec.-20, Marks 13
6. *Describe the golden rules for designing user interface design.* AU : May 22, Marks 7
7. *Explain the three golden rules of Theo Mandel in user interface design.* AU : Dec.-22, Marks 13

3.20 Two Marks Questions with Answers

Q.1 State the guidelines for modular design. AU : IT, Dec.-03

Ans. : The guideline for modular design can be given as :

1. Achieve the functional independence.
2. High level of cohesion should be achieved.
3. There should be minimum number of coupling.

Q.2 How do you evaluate user interface ? AU : IT, May-03

Ans. : The user interface can be evaluated by assessing the usability of interface and checking that it meets the user requirements.

- After preparing the preliminary design a first level prototype is created. This prototype is evaluated by the user.
- Then a formal technical review is conducted.
- The feedback of both the above mentioned reviews is given to designer to make appropriate design modifications.
- This evaluation cycle is continued no further modifications are suggested in the interface design.

Q.3 Brief the importance of user interface. AU : IT, Dec.-03

Ans. :

1. An inexperienced user can use the system easily with the user interface.
2. The user can switch from one task to another very easily. He can also interact with many applications simultaneously. The application information remains visible in its own window.
3. Fast and full screen interaction is possible with user.

Q.4 What is the work product of software design process and who does it ? AU : IT, May-04

Ans. : The work product of software design is the design specification. This specification consists of design models that describe data, architecture, interfaces and components. Software engineers can do it but while designing the complex systems specialized system engineers are required.

Q.5 Define the term software architecture. AU : IT, May-04; CSE, May-08

Ans. : The software architecture is the hierarchical structure of software components and their interactions. In software architecture the software model is designed. The structure of that model is partitioned horizontally or vertically.

Q.6 What are the various models produced by the software design process ? AU : IT, May-06

Ans. : Various models produced during design process are -

1. Data design model used to transform the information domain model of analysis phase into the data structures.
2. The architectural design model is used to represent the relationship between major structural elements with the help of some "design patterns."
3. The interface design model describes how software interacts within itself.
4. In the component-level design model the structural elements of software architecture into procedural description of software components.

Q.7 What are the quality parameters considered for effective modular design ?

OR

State different criteria's applied to evaluae an effective modular system.

AU : CSE, IT, May-06

Ans. : Various parameters considered for effective modular design are -

1. Functional independence - By using functional independence functions may be compartmentalized and interfaces are simplified. Independent modules are easier to maintain with reduced error propagation.
2. Cohesion - A cohesive module performs only "one task" in software procedure with little interaction with other modules. In other words cohesive module performs only one thing.
3. Coupling - Coupling effectively represents how the modules can be "connected" with other module or with the outside world. Coupling is a measure of interconnection among modules in a program structure.

Q.8 In what way abstraction differs from refinement ? AU : CSE, May-06

Ans. : Abstraction and refinement are complementary concepts. The major difference is that - in the abstraction low-level details are suppressed.

Refinement helps the designer to elaborate low-level details.

Q.9 List out atleast four design principles of a good design. AU : IT, Dec.-06 , May-09,18

Ans. :

1. The design process should not suffer from "tunnel vision".
2. The design should be traceable to analysis model.
3. The design should exhibit the uniformity and integrity.
4. Design is not coding and coding is not design.

Q.10 "Modularity is the single attribute of the software that allows a program to be intellectually manageable" - How this is true ? AU : IT, Dec.-06

Ans. : This is quoted by Glenford Mayers. Consider that there is a large program composed of single module. Such a program cannot be grasped by reader. The control paths, span of reference, number of variables and overall complexity of such a program is beyond understanding. On the other hand if the program is divided into certain number of modules then overall complexity of the program gets reduced. The error detection and handling can be done effectively. Also changes made during testing and maintenance become manageable. Hence it is true that "Modularity is the single attribute of the software that allows a program to be intellectually manageable".

Q.11 What are the types of coupling ? (Refer section 3.3.7.2) AU : IT, May-07

Q.12 Name the three levels of abstraction, which are in practice for the design. (Refer section 3.3.1) AU : IT, May-07

Q.13 What are the steps involved in design stage of a software ? (Refer section 3.2) AU : CSE, May-07

Q.14 What are various supporting documents to be prepared for the software ? Explain. AU : CSE, May-07

Ans. :

Stage	Software document	Purpose of the document
System feasibility study	Feasibility report	To check the feasibility of the software development is checked in this stage.
Requirement analysis and project planning	• Software Requirement Specification(SRS) • Project plan (it includes RMMM plan also)	• The requirement gathering and analysis is made in this document. The purpose of this document is to identify scope of the project, effort estimation and determine the project schedule.
Design	Design document	The detailed system design is given in this document.

Code	Programs	Algorithms using suitable programming languages are implemented.
Testing	Test plan, test report	This document typically contains various test cases and test suits.
Installation	Installation manual/user guide	This document contain the specific system requirements that are must for installing the software system being developed. User guide/manual helps the end user to operate the system.

Q.15 What is an architectural style ? (Refer section 3.14) AU : CSE, Dec.-07

Q.16 What are the types of interface design ? AU : CSE, May-08

Ans. : Following are the types of interface design.

1. User interface.
2. External interface to other systems, networks and devices.
3. Internal interfaces between various design components.

Q.17 Why modularity is important in software projects ? AU : IT, Dec.-07

Ans. : Modularity is important in software projects because of following reasons - Modularity reduces complexity and helps in easier implementation. The parallel development of different parts of the system is possible due to modular design. Changes made during testing and maintenance become manageable and they do not affect the other modules.

Q.18 Why it is necessary to design the system architecture before specifications are completed ? AU : IT, May-08

Ans. : The system architecture defines the role of hardware, software, people, database procedures and other system elements. Designing of system architecture will help the developer to understand the system as a whole. Hence system architecture must be built before specifications are completed.

Q.19 How can we evaluate a design method to determine if it will lead to effective modularity ? AU : CSE, Dec.-08

Ans. : **Meyer** has defined five criteria for evaluating design method to determine if it will lead to effective modularity.

Refer section 3.3.2 for these criteria.

Q.20 If a module has logical cohesion what kind of coupling is this module likely to have with others. AU : CSE, May-09, May-16

Ans. : When a module that performs a tasks that are logically related with each other is called logically cohesive. For such module content coupling can be suitable for coupling with other modules. The content coupling is a kind of coupling when one module makes use of data or control information maintained in other module.

Q.21 What is system design ? AU : IT, May-09

Ans. : System design is process of translating customer's requirements into a software product layout. In this process a system design model is created.

Q.22 Define data abstraction. AU : May-13

Ans. : Data abstraction is a kind of representation of data in which the implementation details are hidden.

Q.23 What is software architecture ? AU : May 14

Ans. : Software architecture is a structure of systems which consists of various components, externally visible properties of these components and inter-relationships among these components.

Q.24 Define : Modularity. AU : Dec.-14

Ans. : The modularity is an approach used during the designing of the software system. In this approach, the software is divided into separately named and addressable components called **modules.** Due to modularity, the program becomes manageable.

Q.25 A system must be loosely coupled and highly cohesive. Justify. AU : Dec.-14

Ans. : Loose coupling is an approach to interconnecting the components in a system or network so that those components, are less dependant upon each other. High cohesiveness indicate that module perform only one task. Ideally the components in the system must be least dependent on each other and every component must perform only one task at particular instance. Hence it is said that the system must be loosely coupled and highly cohesive.

Q.26 List down the steps to be followed for user Interface Design. AU : May-15

Ans. : User interface analysis and design can be done with the help of following steps.

1. Create different models for system functions.
2. In order to perform these functions identify the human-computer interface tasks.
3. Prepare all interface designs by solving various design issues.
4. Apply modern tools and techniques to prototype the design.
5. Implement design model.
6. Evaluate the design from end user to bring quality in it.

Q.27 What are the golden rules for an interface design ? AU : Dec.-15

Ans. : There are three golden rules for user interface design -

1) Place the user in control.

2) Reduce the user's memory load.

3) Make the interface consistent.

Q.28 How can refactoring be made more effective ? AU : May-16

Ans. : After careful analysis of design (or code) separate out the components that can be refactored. Then refactor them and finally integrate and test them.

Q.29 What architectural styles are preferred for the following systems ? Why ?

a) Networking b) Web based systems c) Banking system AU : Dec.-16

Ans. : a) Pipe and filter can be used for networking system. The pipe and filter pattern has set of components called filters, connected by pipes which transmit data from one component to next.

b) The layered architecture can be used for web based systems. In this architecture

- **At outer layer :** There are component service user interface operations.
- **At inner layer :** Components perform operating system interfacing
- **At intermediate layer :** There are utility services and application software functions.

c) The object oriented architecture can be used for banking system.

In this architecture, each object encapsulates data and operations. The objects are identified and corresponding operations are performed.

Q.30 What UI design patterns are used for the following ?

a) Page layout b) Tables

c) Navigation through menus and web pages

d) Shopping cart

Ans. : AU : Dec.-16, May-17,18

(a)	Page layout	Card stack
(b)	Tables	Sorted tables
(c)	Navigation through menus and pages	Edit-in place
(d)	Shopping cart	Shopping cart which provides list of items selected for purchase

Q.31 List out the various types of cohesion and coupling.

Ans. : Different types of cohesion are - AU : Dec.-20

1) Coincidentally cohesive
2) Logically cohesive
3) Temporal cohesion
4) Procedural cohesion
5) Communicational cohesion

Different types of coupling are -

1) Data coupling
2) Control coupling
3) Common coupling
4) Content coupling

Q.32 Define cohesion and coupling. AU : May-22

Ans. : Cohesion : A cohesive module performs only "one task" in software procedure with little interaction with other modules. In other words cohesive module performs only one thing.

Coupling : Coupling effectively represents how the modules can be "connected" with other module or with the outside world. Coupling is a measure of interconnection among modules in a program structure.

Q.33 What is software design ? AU : Dec.-22

Ans. : Software design is model of software which translates the requirements into finished software product in which the details about software data structures, architecture, interfaces and components that are necessary to implement the system are given.

❑❑❑

UNIT IV

4 Software Testing and Maintenance

Syllabus

Testing - Unit testing - Black box testing - White box testing - Integration and System testing - Regression testing - Debugging - Program analysis - Symbolic execution - Model Checking- Case Study.

Contents

4.1 Introduction to Testing **May-07,16,**
..... **Dec.-04,05,08,10,**..... Marks 8
4.2 Internal and External Views of Testing
4.3 Black Box Testing **May-07,12,15,16,17,19,22,**
..... **Dec.-03,04,11,13,15,16,20,** Marks 16
4.4 White Box Testing **May-04,07,08,17,18,19,**
..... **Dec.-11,13,14,16,17,** ..... Marks 10
4.5 Testing Strategies ..... **May-05, 06, 22,** ..... Marks 16
4.6 Unit Testing ..... **Dec.-06,08,15, May-03,09,**.. Marks 16
4.7 Integration Testing **Dec.-06,08,15,19, May-03,04,05,09,**
..... **May-13,14,15,17,18,** ..... Marks 16
..... **May-03,04,05, Dec.-14,** ..... Marks 16
4.8 System Testing ..... **May-05,16,** ..... Marks 16
4.9 Validation Testing ..... **Dec.-10,14, May-15,18,** ..... Marks 4
4.10 Debugging Techniques
4.11 Program Analysis
4.12 Symbolic Execution
4.13 Model Checking
4.14 Two Marks Questions with Answers

4.1 Introduction to Testing

AU : May-07,16,Dec.-04,05,08,10, Marks 8

Definition : Software testing *is a critical element of software quality assurance and represents the ultimate review of specification, design and coding.*

The purpose of software testing is to ensure whether the software functions appear to be working according to specifications and performance requirements.

4.1.1 Testing Objectives

According to **Glen Myers** the testing objectives are,

1. Testing is a process of executing a program with the intend of finding an error.
2. A good test case is one that has high probability of finding an undiscovered error.
3. A successful test is one that uncovers an as-yet undiscovered error.

The major testing objective is to design tests that systematically uncover types of errors with minimum time and effort.

4.1.2 Testing Principles

Every software engineer must apply following testing principles while performing the software testing.

1. All tests should be traceable to customer requirements.
2. Tests should be planned long before testing begins.
3. The Pareto principle can be applied to software testing - 80 % of all errors uncovered during testing will likely be traceable to 20 % of all program modules.
4. Testing should begin "in the small" and progress toward testing "in the large".
5. Exhaustive testing is not possible.
6. To be most effective, testing should be conducted by an independent third party.

4.1.3 Why Testing is Important ?

- Generally, testing is a process that requires more efforts than any other software engineering activity.
- Testing is a set of activities that can be planned in advance and conducted systematically.
- If it is conducted haphazardly, then only time will be wasted and more even worse errors may get introduced.
- This may lead to have many undetected errors in the system being developed. Hence performing testing by adopting systematic strategies is very much essential in during development of software.

Example 4.1.1 *Why does software testing need extensive planning ? Explain.* **AU : May-16, Marks 8**

Solution : Following are the issues that explain why the testing need extensive planning.

1. Testing is a process of executing the program with the intent of finding errors. Although it is an expensive activity, yet it is essential to perform testing rigorously especially in systems where human safety is involved.
2. Testing requires the developers to find errors from their software. But it is very difficult for developers to point out errors from their own design or code. Hence many organizations have made distinction between development and testing phase by making different people responsible for each phase.
3. The testing is performed on program's response to every possible input. That means, we should test for all valid and invalid inputs. Although practically testing each and every input can not be tested, the important cases of valid and invalid inputs must be tested.
4. Another aspect of testing is for execution of all possible paths of program. A program path can be traced through the code from the start of the program to program termination. Although it is just difficult to test each and every path of the program, it is expected to test those areas where probability of getting a fault is maximum.

Thus organizations should develop strategies and policies for choosing effective testing techniques which requires extensive planning.

Example 4.1.2 *Distinguish among Error/Fault/Failure.* **AU : CSE, Dec.-04, Marks 6**

OR

Distinguish between software fault and failure. **AU : CSE, Dec.-08, Marks 2**

Solution : Error : It is a state that can lead to a system behaviour that is unexpected by the system user.

Fault : It is a characteristic of a software system that can lead to system error.

Failure : It is an event that occurs at some point in time when the system does not deliver a service as per user's expectations.

Example 4.1.3 *Distinguish between defects and errors.* **AU : Dec.-05, Marks 6**

Solution : Error is a state that can lead to a system behaviour that is unexpected by the system user. The software team performs the formal technical reviews to test the software developed. In this review errors are identified and corrected.

Any errors that remain uncovered and are found in later tasks are called defects. Error removal is software development activity and defect removal is a software quality assurance activity.

Example 4.1.4 *When do you stop testing ? Justify your answer with two illustrations.*

AU : CSE, May-07, Marks 8

Solution : Testing is a complex activity in the software systems. It is said that testing is an endless process and complete testing not possible for almost all the projects. But there are some common factors that are required to decode when to stop testing.

1. When the testing cost is increasing and if it is more than the project cost then it is enough to test.
2. If the project deadline and testing deadline is already crossed.
3. After completion of critical or key test cases one can stop testing.
4. If the project is meeting functional coverage, code coverage or satisfying the client requirements at some point.
5. When high priority bugs are resolved and defect rates fall below certain specified level.
6. When project progresses through alpha and beta testing.

For example :

Case 1 :

In the development of Windows operating system the Internet Explorer is a famous web browser that we use. There are many security patches that need to be applied. Some patches are already introduced with newer versions of operating systems. But still this product specially shows how testing is an **endless process**.

Case 2 :

For sorting a list of elements typical test cases can be -

Test data	Test case name	Expected result
2, 1, 3	Testing unsorted list.	1, 2, 3
2, 3, 2	Testing when equal elements are present.	2, 2, 3
	Testing empty list.	Print message "List is empty".
−4, −5, 0	Testing with negative numbers.	−5, −4, 0

This much testing is sufficient because it satisfies the major requirement of the function. Thus unlike case 1 we can **stop testing** here.

Example 4.1.5 *What are the characteristics of good tester ?*

AU : Dec.-10, Marks 2

Solution : Following are the characteristics of good tester -

1. The tester must be able to understand the software. He should be in a position to find out high probability errors.

2. The tester must not conduct two different tests for the same purpose.
3. The tester must be able to write simple test cases.
4. The tester should conduct the tests which should have highest likelihood of uncovering errors.

4.2 Internal and External Views of Testing

There are two views of the testing. The internal view and external view. The internal view is also known as white box testing and the external view is also known as black box testing.

1. Black box testing

The black box testing is used to demonstrate that the **software functions** are operational. As the name suggests in black box testing it is tested whether the **input** is accepted properly and **output** is correctly produced.

The major focus of black box testing is on **functions, operations, external interfaces, external data** and **information.**

2. White box testing

In white box testing the **procedural details** are closely examined. In this testing the internals of software are tested to make sure that they operate according to specifications and designs. Thus major focus of white box testing is on **internal structures, logic paths, control flows, data flows, internal data structures, conditions, loops,** etc.

4.3 Black Box Testing

AU : May-07,12,15,16,17,19,22, Dec.-03,04,11,13,15,16,20. Marks 16

- The black box testing is also called as **behavioural testing.**
- Black box testing methods focus on the functional requirements of the software. Test sets are derived that fully exercise all functional requirements.
- The black box testing is not an alternative to white box testing and it uncovers different class of errors than white box testing.

Why to perform black box testing ?

Black box testing uncovers following types of errors.

1. Incorrect or missing functions
2. Interface errors
3. Errors in data structures
4. Performance errors
5. Initialization or termination errors

4.3.1 Equivalence Partitioning

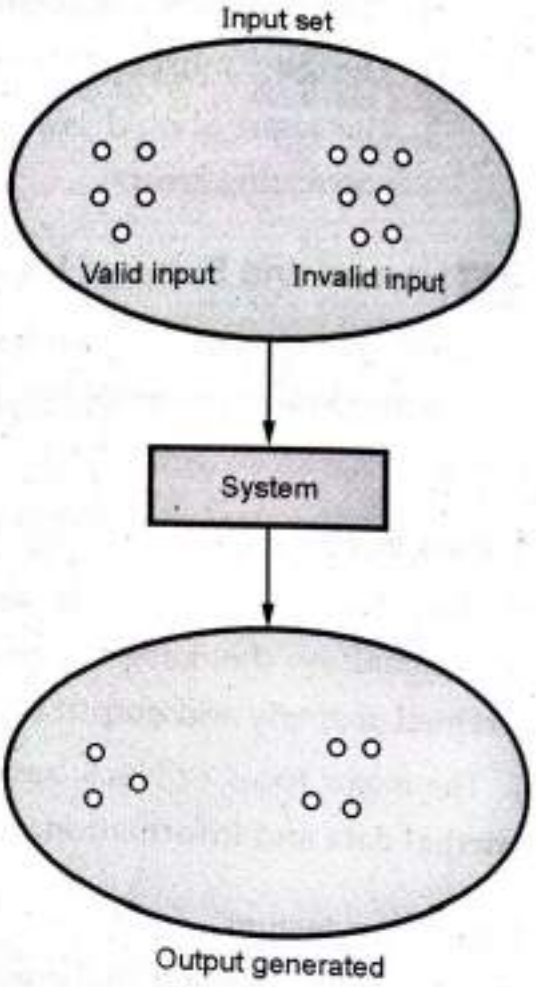


Fig. 4.3.1

- It is a black box technique that divides the input domain into classes of data. From this data test cases can be derived.
- An ideal test case uncovers a class of errors that might require many arbitrary test cases to be executed before a general error is observed.
- In equivalence partitioning the equivalence classes are evaluated for given input condition. Equivalence class represents a set of valid or invalid states for input conditions.
- Equivalence class guidelines can be as given below :
  - If input condition specifies a range, one valid and two invalid equivalence classes are defined.
  - If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
  - If an input condition specifies a member of a set, one valid and one invalid equivalence class is defined.
  - If an input condition is Boolean, one valid and one invalid equivalence class is defined.

For example :

Area code : Input condition, Boolean - The area code may or may not be present.

Input condition, range - Value defined between 200 and 700.

Password : Input condition, Boolean - A password may or may not be present.

Input condition, value - Seven character string.

Command : Input condition, set - Containing commands noted before.

Example 4.3.1 *Describe black box testing. Design the black-box test suit for the following program. The program computes the intersection point of two straight lines and displays the result. It reads two integer pairs (m_1, c_1) and (m_2, c_2) defining the two straight lines of the form $y' = mx + c$.*

AU : May-17, Marks 5

Solution : Black box testing : Refer section 4.3.

The equivalence classes for given program are as follows :

1) Parallel lines if $m_1 = m_2$, $c_1 \neq c_2$

2) Intersecting lines if $m_1 \neq m_2$

3) Coincident lines if $m1 = m_2$, $c_1 = c_2$

The representative values can be selected from each equivalence class.

Thus the test suit obtained can be

(5, 5) (5, 7)

(7, 10) (8, 12)

(15, 15) (15, 15)

4.3.2 Boundary Value Analysis (BVA)

- Boundary value analysis is done to check boundary conditions.
- A boundary value analysis is a testing technique in which the elements at the edge of the domain are selected and tested.
- Using boundary value analysis, instead of focusing on input conditions only, the test cases from output domain are also derived.
- Boundary value analysis is a test case design technique that complements equivalence partitioning technique.
- Guidelines for boundary value analysis technique are,

1. If the input condition specified the range bounded by values x and y, then test cases should be designed with values x and y. Also test cases should be with the values above and below x and y.
2. If input condition specifies the number of values then the test cases should be designed with minimum and maximum values as well as with the values that are just above and below the maximum and minimum should be tested.
3. If the output condition specified the range bounded by values x and y, then test cases should be designed with values x and y. Also test cases should be with the values above and below x and y.
4. If output condition specifies the number of values then the test cases should be designed with minimum and maximum values as well as with the values that are just above and below the maximum and minimum should be tested.
5. If the internal program data structures specify such boundaries then the test cases must be designed such that the values at the boundaries of data structure can be tested.

For example :

Integer D with input condition [– 2, 10],

Test values : – 2, 10, 11, – 1, 0

If input condition specifies a number values, test cases should developed to exercise the minimum and maximum numbers. Values just above and below this min and max should be tested.

Enumerate data E with input condition : {2, 7, 100, 102}

Test values : 2, 102, – 1, 200, 7

Example 4.3.2 *Find the boundary value test cases for the following :*
If x is less than level 1 go to 100 else 200.
If y is greater than level 2 go to 300 else 400.

AU : May-07, Marks 8

Solution :

Sr. No.	Test case name	Test data	Expected result
1.	Testing lower boundary of x	If x = level 0 If x = level 1	go to 100 go to 200
2.	Testing upper boundary of x	If x = level 2 or more	go to 200
3.	Testing lower boundary of y	If y = level 1 or level 0 If y = level 2	go to 400 go to 400
4.	Testing upper boundary of y	If y = level 3 or more	go to 300

Example 4.3.3 *Design a black box testing for an under water submarine.*

AU : Dec.-11, Marks 8

Solution : Inside submarine there are containers called **ballast tanks.** If ballast tanks are full of air then, the submarine will float otherwise, if water is pumped into the ballast tank then submarine will sink.

To make submarine work underwater it is necessary to fill up ballast tank with water. The submarine will float, sink or rise by adjusting water and air level in ballast tank. The **rudder** of submarine is turned left or right.

Sr. No.	Test case	Expected result
1.	The ballast tank is filled with air above threshold level.	It will float on the surface of water.
2.	The ballast tank is filled with water with water above threshold.	It will sink under water by letting some water out.
3.	Fin is centered.	Submarine can be moved forward.

4.	Fin to right side.	Direction changes.
5.	Fin to left side.	Direction gets changed.
6.	Creating adequate amount of oxygen.	The water is separated out as and O by releasing oxygen.

Example 4.3.4 *A program specs state the following for an input field : The program shall accept an input value of 4-digit integer equal or greater than 2000 and less than or equal 8000. Determine the test cases using.*

i) Equivalence class partitioning. ii) Boundary value analysis. **AU : Dec.-15, Marks 16**

Solution : i) Equivalence class partitioning

We can have set of test cases which are based on input domain

I_1 = { Integer < 2000 }

I_2 = { Integer = 2000 }

I_3 = { Integer > 2000 and < 8000 }

I_4 = { Integer = 8000 }

I_5 = { Integer > 8000 }

The test cases are

Test case	Integer	Expected output
1	1900	Invalid input
2	2000	Valid input
3	5000	Valid input
4	8000	Valid input
5	9000	Invalid input

ii) Boundary Value Analysis :

The boundary values can be,

i) Less than 2000

ii) Greater than and / or equal to 2000

iii) Less than and / or equal to 8000

iv) Greater than 8000.

Test case	Integer	Expected output
1	1000	Invalid input
2	2000	Valid input
3	4000	Valid input
4	8000	Valid input
5	9000	Invalid input

Example 4.3.5 *Consider a program for determining the previous date. Its input is a triple of day, month and year with the values in the range 1 month 12, 1 day 31, 1990 year 2014. The possible outputs would be previous date or invalid input date. Design the boundary value test cases.* **AU : May-16, Marks 8**

Solution : This program takes current date as input and returns previous date as its output.

There are three variables for the program – month, day and year. Hence there would be $4n + 1 = (4 \times 3) + 1 = 13$ test cases that can be designed. The boundary value test cases are.

Test Case	Month	Day	Year	Expected output
1	5	16	1990	15 May 1990
2	5	16	1992	15 May 1992
3	10	1	1995	30 September 1995
4	4	15	1985	Invalid date
5	7	2	2001	1 July 2001
6	3	11	2003	10 March 2003
7	3	29	2005	28 March 2005
8	11	12	2007	11 November 2007
9	11	15	2008	14 November 2008
10	1	15	2008	14 January 2008
11	6	17	2010	16 June 2010
12	8	16	2014	15 August 2014
13	8	16	2015	Invalid date

Example 4.3.6 *Given the requirements for an Automated Teller Machine (ATM) system (see below), design the following :*

i) Use case diagram. ii) Activity diagram detailing each use case.

iii) List test cases for any one functionality from your use case diagram.

The ATM will service one customer at a time. A customer will be required to insert an ATM card and enter a Personal Identification Number (PIN) - both of which will be sent to the bank for validation as part of each transaction. The customer will then be able to perform one or more transactions.

The ATM must be able to provide the following services to the customer :

A customer must be able to make a cash withdrawal from any suitable account linked to the card, in multiples of $20.00. Approval must be obtained from the bank before cash is dispensed.

A customer must be able to make a deposit to any account linked to the card, consisting of cash and/or checks in an envelope. The customer will enter the amount of the deposit into the ATM, subject to manual verification when the envelope is removed from the machine by an operator. Approval must be obtained from the bank before physically accepting the envelope.

A customer must be able to make a transfer of money between any two accounts linked to the card.

A customer must be able to make a balance inquiry of any account linked to the card.

AU : May-19, Marks 15

Solution : i) Use case diagram

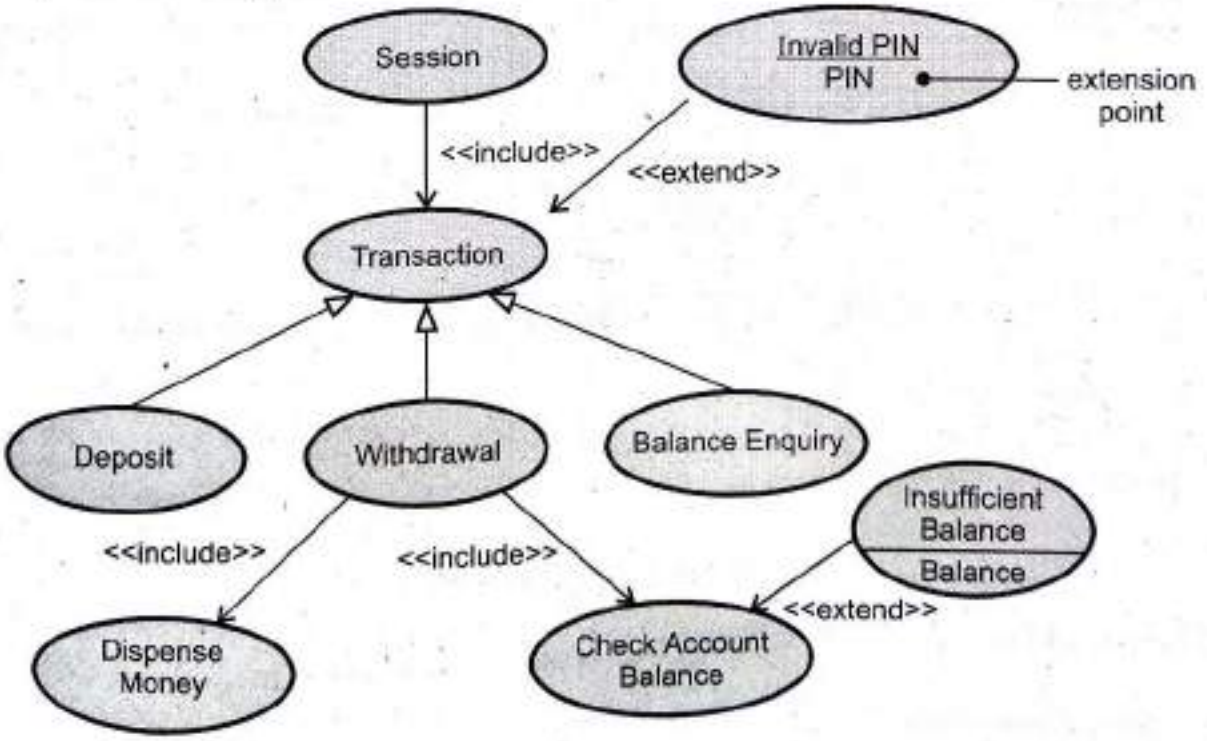


Fig. 4.3.2 Withdrawal of money from ATM

ii) Activity diagram

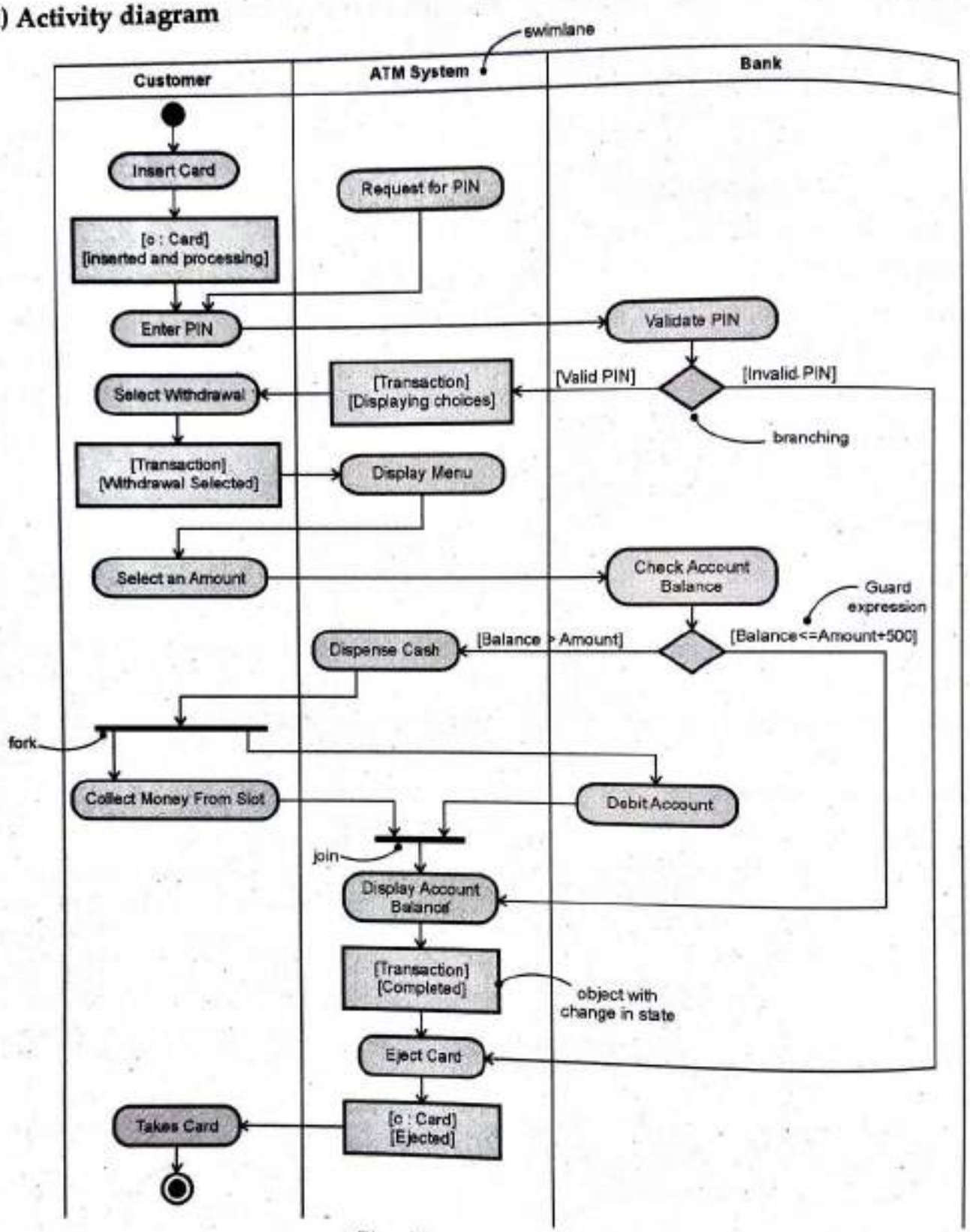


Fig. 4.3.3 Activity model

iii) The list of test cases is as follows -

Cash withdrawal operation

1) Machine does not accept the card as card is expired.
2) User enters wrong PIN.

3) Verify that user is presented with different account type options like - saving, current etc.
4) Verify that user is only allowed to enter amount in multiples of denominations as per the specifications.
5) Verify that user is prompted to enter the amount again in case amount entered is not as per the specification and proper message should be displayed for the same.
6) Verify that user cannot withdraw more amount than the balance.

Money transfer from one account to another

1) Verify that the correct selection of the choice of transfer money from Menu.
2) Verify that the sufficient amount is present in the source account.
3) Verify that the correct account number is mentioned for transfer of amount'.
4) Verify that the correct amount is mentioned for transfer.
5) Verify that other account is linked to the card.

Review Questions

1. *What is black box testing ? Explain the different types of black box testing strategies. Explain by considering suitable examples.* **AU : Dec.-16, Marks 16**
2. *How do you test boundary conditions ?* **AU : May-12, Dec.-03, Marks 2, Dec.-04, Marks 8**
3. *What is boundary value analysis ? Explain the technique specifying rules and its usage with the help of an example.* **AU : Dec.-13, Marks 7**
4. *Describe the various black box and white box testing techniques. Use suitable examples for your explanation.* **AU : May-15, Marks 16**
5. *Explain equivalence partitioning technique with suitable example.* **AU : Dec.-20, Marks 13**
6. *With suitable example, explain boundary value analysis.* **AU : May-22, Marks 6**

4.4 White Box Testing

AU : May-04,07,08,15,17,18,19, Dec.-07,11,13,14,16,17, Marks 10

- The white box testing is also called as structural testing.
- In white box testing derivation of test cases is according to program structure. Hence knowledge of the program is used to identify additional test cases.
- Objective of white box testing is to exercise all program statements.

4.4.1 Condition Testing

- To test the logical conditions in the program module the condition testing is used. This condition can be a Boolean condition or a relational expression.
- The condition is incorrect in following situations.
  i) Boolean operator is incorrect, missing or extra.
  ii) Boolean variable is incorrect.

iii) Boolean parenthesis may be missing, incorrect or extra.

iv) Error in relational operator.

v) Error in arithmetic expression.

- The *condition testing* focuses on each testing condition in the program.
- The *branch testing* is a condition testing strategy in which for a compound condition each and every true or false branches are tested.
- The *domain testing* is a testing strategy in which relational expression can be tested using three or four tests.

4.4.2 Loop Testing

Loop testing is a white box testing technique which is used to test the loop constructs. Basically there are four types of loops.

1. Simple loops :

The tests can be performed for n number of classes.

if,

i) n = 0 that means skip the loop completely.

ii) n = 1 that means one pass through the loop is tested.

iii) n = 2 that means two passes through the loop is tested.

iv) n = m that means testing is done when there are m passes where m < n.

v) Perform the testing when number of passes are n – 1, n, n + 1.

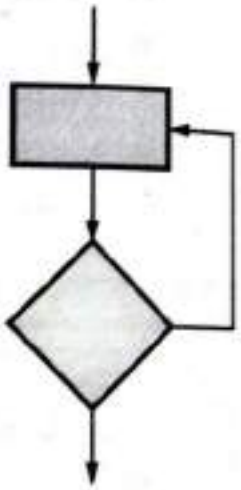


Fig. 4.4.1 Simple loop

2. Nested loops :

The nested loop can be tested as follows.

i) Testing begins from the innermost loop first. At the same time set all the other loops to minimum values.

ii) The simple loop test for innermost loop is done.

iii) Conduct the loop testing for the next loop by keeping the outer loops at minimum values and other nested loops at some specified value.

iv) This testing process is continued until all the loops have been tested.

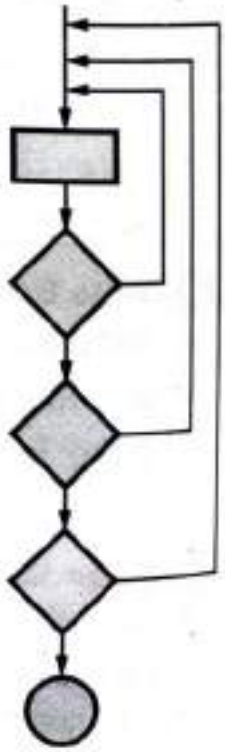


Fig. 4.4.2 Nested loops

3. Concatenated loops :

The concatenated loops can be tested in the same manner as simple loop tests.

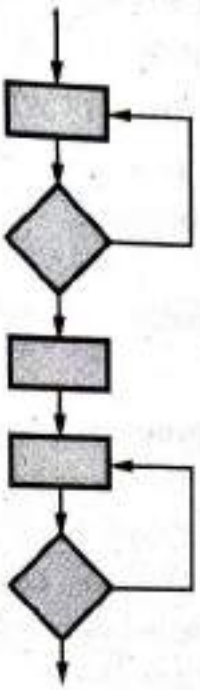


Fig. 4.4.3 Concatenated loops

4. Unstructured loops :

The testing cannot be effectively conducted for unstructured loops. Hence these types of loops needs to be redesigned.

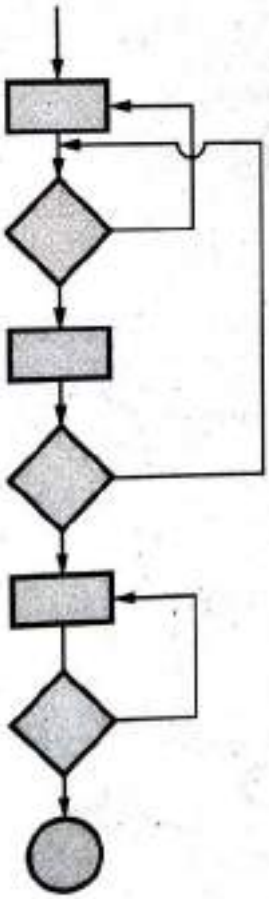


Fig. 4.4.4 Unstructured loops

4.4.3 Basis Path Testing

Path testing is a structural testing strategy. This method is intended to exercise every independent execution path of a program atleast once.

Following are the steps that are carried out while performing path testing.

Step 1 : Design the flow graph for the program or a component.

Step 2 : Calculate the cyclomatic complexity.

Step 3 : Select a basis set of path.

Step 4 : Generate test cases for these paths.

Let us discuss each in detail.

Step 1 : Design the flow graph for the program or a component.

Flow graph is a graphical representation of logical control flow of the program. Such a graph consists of circle called a ***flow graph node*** which basically represents one or more procedural statements and arrow called as ***edges*** or ***links*** which basically represent control flow. In this flow graph the areas bounded by nodes and edges are called ***regions***. Various notations used in flow graph are

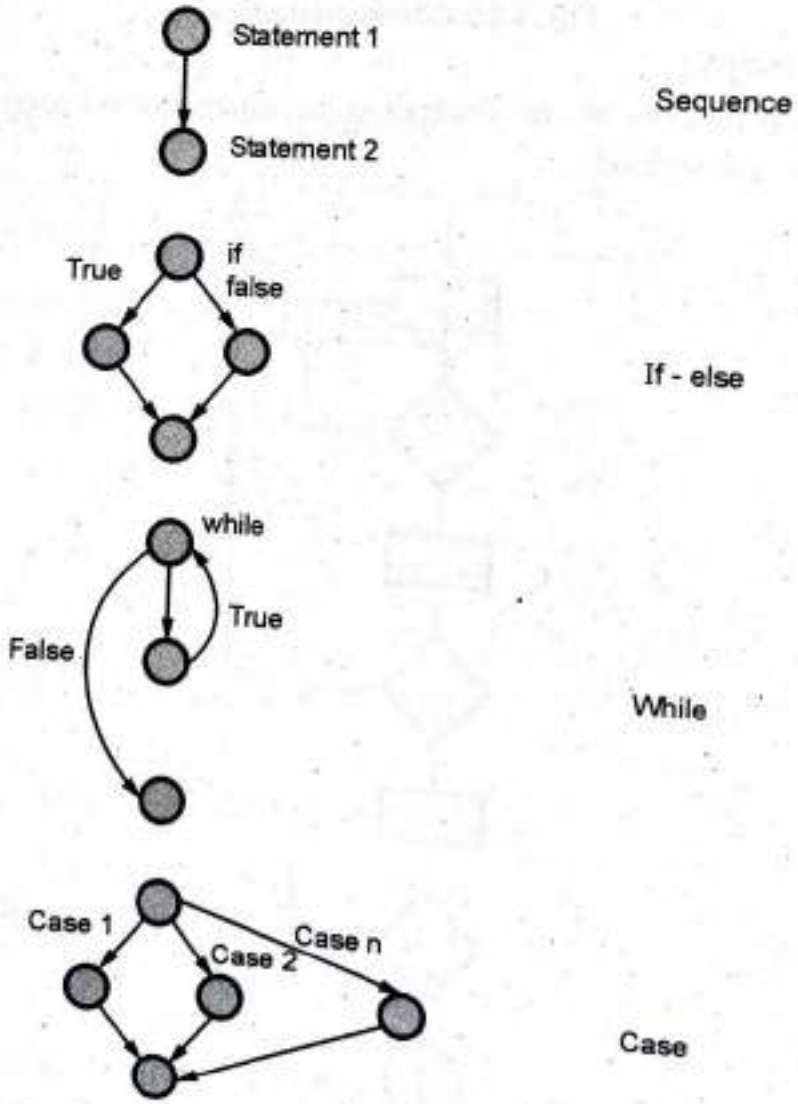


Fig. 4.4.5

For example : Following program is for searching a number using binary search method. Draw a flow graph for the same.

```
void search (int key, int n, int a [ ])
 {
  int mid;
1) int bottom = 0;
2) int top = n – 1;
3) while (bottom <=top)
4) { mid = (top + bottom) / 2;
5) if (a [mid] = = key)
  {
6) printf ("Element is present");
7) return;
  } // end of if
  else
  {
8) if (a [mid] < key)
9) bottom = mid + 1;
  else
10)    top = mid – 1;
  } // end of else
  } // end of while
11)   } // end of search
```

The flow graph will be

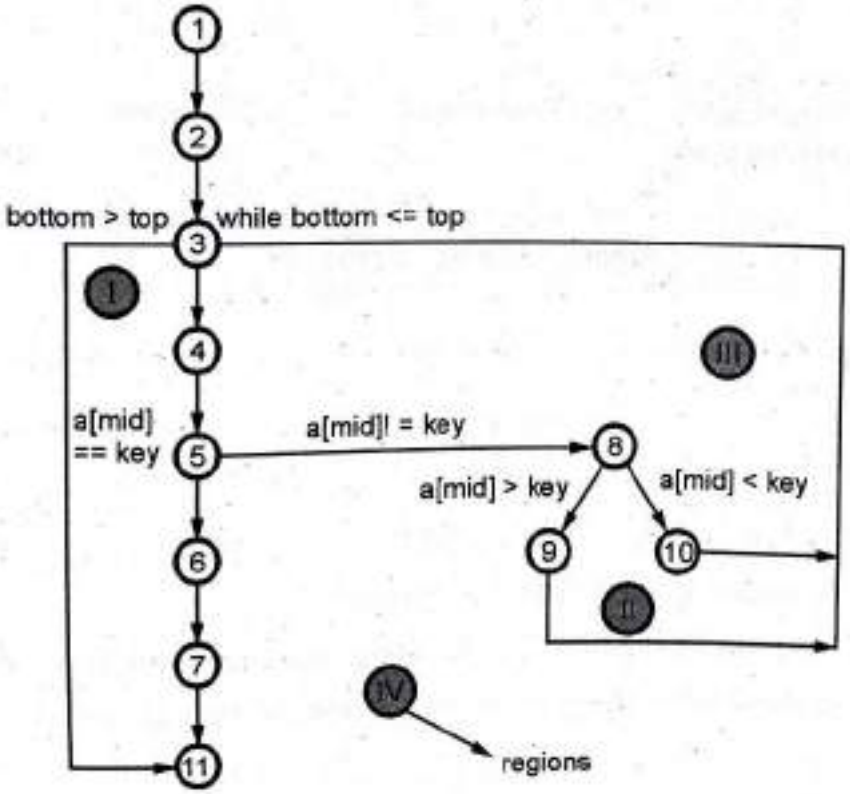


Fig. 4.4.6

Step 2 : Calculate the cyclomatic complexity.

The cyclomatic complexity can be computed by three ways.

1) Cyclomatic complexity = Total number of regions in the flow graph = **4** (note that in above flow graph regions are given by shaded roman letters).

2) Cyclomatic complexity = E – N + 2 = 13 edges – 11 nodes + 2

$$= 2 + 2 = 4$$

3) Cyclomatic complexity = P + 1 = 3 + 1 = **4**. There are 3 predicate (decision making) nodes : Nodes 3, 5 and 8.

Step 3 : Select a basis set of path.

The basis paths are

Path 1 : 1, 2, 3, 4, 5, 6, 7, 11

Path 2 : 1, 2, 3, 11

Path 3 : 1, 2, 3, 4, 5, 8, 9, 3 ...

Path 4 : 1, 2, 3, 4, 5, 8, 10, 3 ...

Step 4 : Generate test cases for these paths.

After computing cyclomatic complexity and finding independent basis paths, the test cases has to be executed for these paths. The format for test case is -

Preconditions :

Test case id	Test case name	Test case description	Test steps			Test case status (Pass/Fail)	Test priority	Defect severity
			Step	Expected	Actual			
1								
2								
n								

The test case for binary search can be written as -

Precondition : There should be list of elements arranged in ascending order. The element to be searched from the list, its value should be entered and will be stored in variable *'key'*.

Test case id	Test case name	Test case description	Test steps			Test case status	Test status (P / F)	Test priority	Defect severity
			Steps	Expected	Actual				
1.	Validating the list boundary	Checking the bottom and top values for the list of elements	Set bottom = 0 top = n – 1 check if bottom < = top by while loop. This condition defines the length of the list from which the key is searched.	Initially bottom < = top will be true. But during iterations list's length will be reduced and if entire list gets scanned at one point (bottom > = top) will be reached then return to main		Design			
2.	Checking list element with key	Checking if middle element of array is equal to key value	Set mid = (top + bottom) / 2 Then compare if a[mid] is equal to key.	If a[mid] = key value then print message "Element is present" and return to main.		Design			
		If a[mid] is < key value if a[mid] is > key value	Set bottom = mid + 1 and then goto "while" set top = mid +1 and go back to "while" loop	The right sublist will be searched. The left sublist will be searched for key element					

Example 4.4.1 *Write a program for sorting of n numbers. Draw the flowchart, flowgraph, find out the cyclomatic complexity.* **AU : Dec.-11, Marks 16**

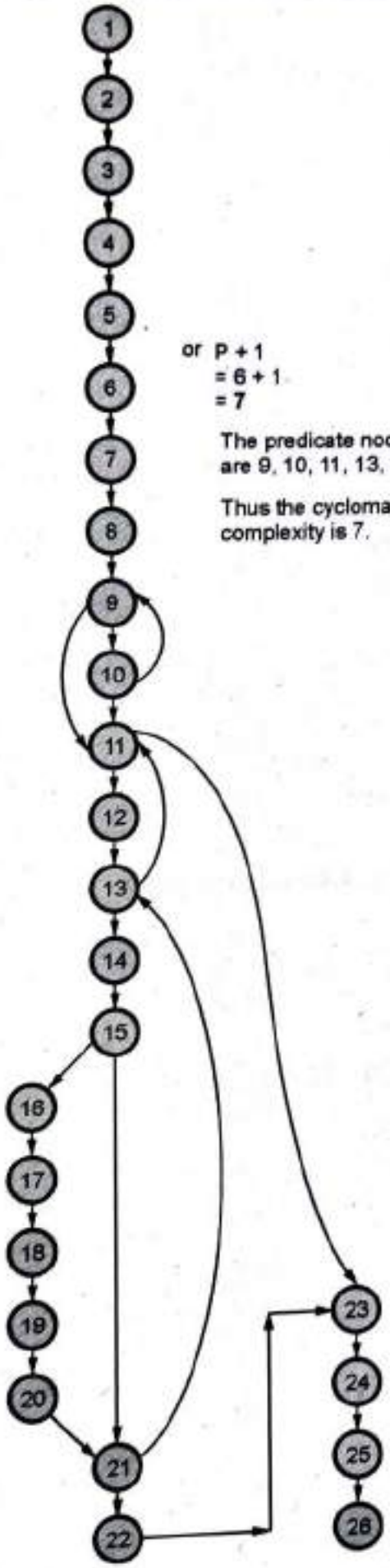
Solution :

```
#include<stdio.h>
#include<conio.h>
int n;
1.  void main( )
2.  {
3.    int i,A[10];
4.    int j,temp;
5.    printf("\n\t\t Bubble Sort\n");
6.    printf("\n How many elements are there?");
7.    scanf("%d",&n);
8.    printf("\n Enter the elements\n");
9.    for(i=0;i<n;i++)
10.     scanf("%d",&A[i]);
11.   for(i=0;i<=n-2;i++)

12.   {
13.     for(j=0;j<=n-2-i;j++)
14.     {
15.     if(A[j]>A[j+1])
16.     {
17.      temp=A[j];
18.      A[j]=A[j+1];
19.      A[j+1]=temp;
20.     }
21.     }
22.   }
23.   printf("\n The sorted List is ...\n");
24.   for(i=0;i<n;i++)
25.     printf(" %d",A[i]);
26.   }
```

The flow graph for above program is as shown in Fig. 4.4.7. (See Fig. 4.4.7 on next page)

Example 4.4.2 *Given a set of numbers 'n', the function. Find Prime(a[ ],n) prints a number - if it is a prime number. Draw a control flow graph, calculate the cyclomatic complexity and enumerate all paths. State how many test case-s are needed to adequately cover the code in terms of branches, decisions and statement ? Develop the necessary test cases using sample values for 'a' and 'n'.* **AU : Dec.-13, Marks 16**



or P + 1
= 6 + 1
= 7

The predicate nodes P are 9, 10, 11, 13, 15, 21.

Thus the cyclomatic complexity is 7.

Fig. 4.4.7

Solution :

```
Void Find Prime (int a[], int n)
{
 1. i=0
 2. while (i < n)
    {
 3.   flag = 0
 4.   j = 2
 5.   while (j < a[i])
      {
 6.    rem = a[i]% j;
 7.    if (rem = = 0)
       {
 8.        flag = 1 ;
 9.        break ;
       }
10.        j++;
11.  } //end of while
12.  if (flag = = 0)
13.       Print f (" %d", a[i]);
14.  i++;
15. }//end of while
}//end of function
```

The flow graph will be - (See Fig. 4.4.8 on next page)

Number of edges E = 20

Number of nodes N = 15

Cyclomatic complexity = E – N + 2

= 20 – 15 + 2

= 7

Basis set of path :

Path 1 : 1, 2, 3, 4, 5, ... 15

Path 2 : 1, 2, 15

Path 3 : 1, 2, 3, 4, 5, 6, 7, 10, ... 15

Path 4 : 1, 2, 3, 4, 5, 11, 12, 13, ... 15

Path 5 : 1, 2, 3, ... 12, 14, 15

Path 6 : 1, 2, 3, 4, 5, 11, 12, 14, 15

Path 7 : 1, 2, 3, 4, 5, 11, 12, 13, 14, 15.

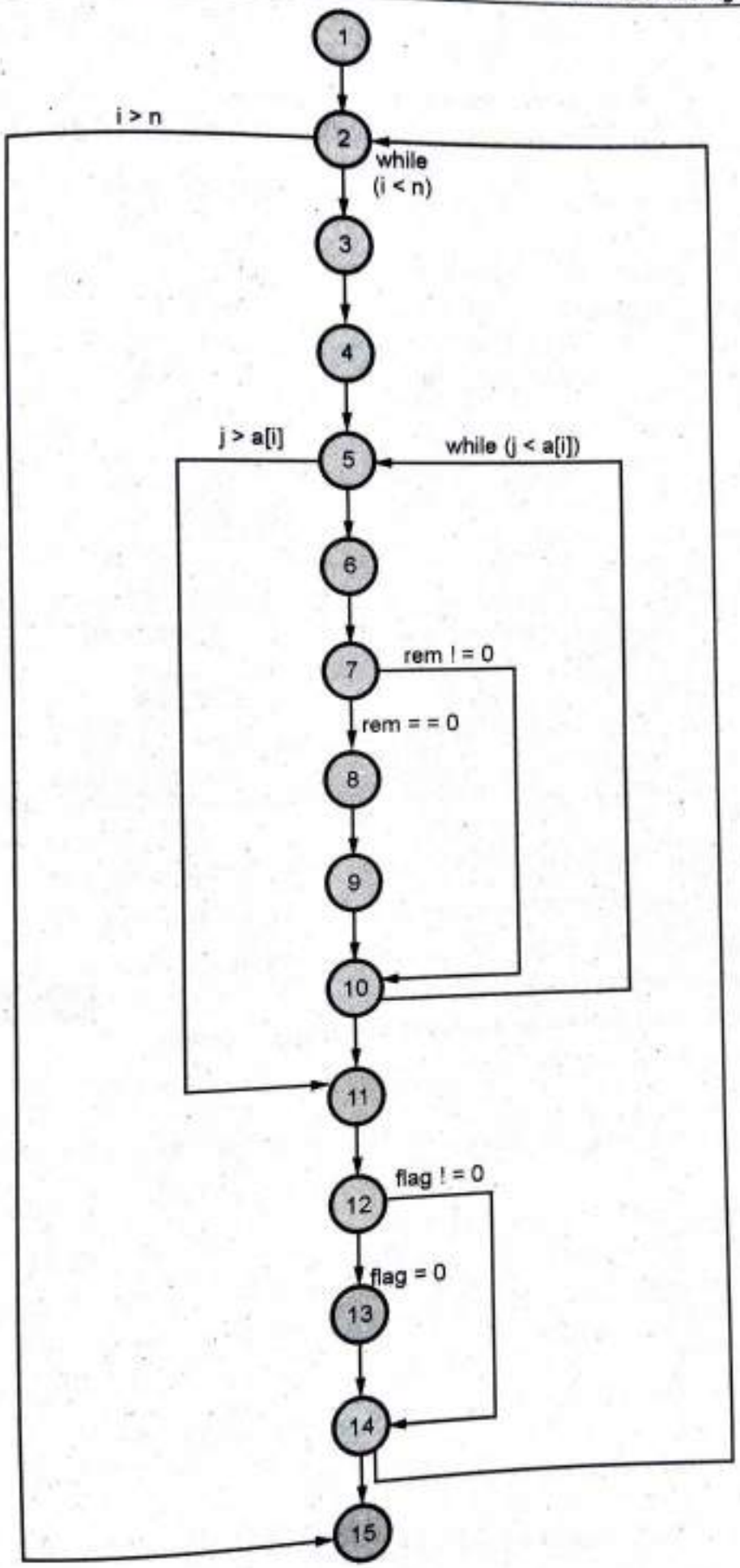

1
i > n
2
while
(i < n)
3
4
j > a[i]
5
while (j < a[i])
6
7
rem ! = 0
rem = = 0
8
9
10
11
12
flag ! = 0
flag = 0
13
14
15

Fig. 4.4.8

Test case :

Pre condition : i) a[ ] stores number to be tested.
ii) j denotes any number within a range to test divisibility.

Test case id	Test case name	Description	Step	Test case status (P/F)

1.	Divisibility by other number	remainder is zero when number in a[ ] is divisible by other than 1 or itself	rem = a[i] % j
			where j < n
			if (rem = = 0)
			set flag = 1
			If (flag ! = 0)
			then
			"Number is not Prime"
				P

2.	Divisibility by 1 or self	remainder is not zero when number in a [ ] is not divisible by a number other than 1 or self	rem = a[i]%j
			where j<n
			if (rem ! = 0)
			set flag = 0
			if (flag = 0)
			then
			"Number is Prime"
				P

Example 4.4.3 *Given a set of 'n' numbers, write an algorithm that finds whether the given number is positive, negative, zero, even or odd. Finally, the total number in each category is also printed. Draw the flow graph and enumerate paths for testing. Determine the number of independent paths using cyclomatic complexity.* **AU : Dec.-14, Marks 8**

Solution : Step 1 : We will first write an algorithm using 'C' code.

```
void main()
{
  1. int a[ ] = {10, – 4, 1, 7, 8, 6, 4 };
  2 int i;
  3. int nz, np, nn, ne, no;
  4. nz = np = nn = ne = no = 0;
  5. for(i = 0; i < n; i++)
     {
  6.   if (a[i] = = 0)
  7.   nz++; /* number of zeros */
  8.   else if(a|i]>0)
  9.   np++; /* number of positive numbers */
  10.  else
  11.  nn++;* number of negative numbers */
     }
```

```
12. for(i = 0; i<n; i++)
    {
13. if(a[i]%2 = = 0)
14.     ne++;
15. else
16.     no++;
    }
17. printf ("Number of zeros %d", nz);
18. printf ("Number of positive %d", np);
19. printf ("Number of negatives %d", nn);
20. printf ("Number of even %d", ne);
21. printf ("Number of odd %d", no);
}
```

Step 2 : We will draw the flow graph from above code. It is as follows -

(See Fig. 4.4.9 on next page.)

Step 3 : Some of the independent paths for testing are -

Path I : 1, 2, 3, 4, 5, 6, 7, 5, 12, 13, 15, 16, 12, 17, 18, 19, 20, 21.

Path II : 1, 2, 3, 4, 5, 6, 8, 9, 5, 12, 13, 15, 16, 12, 17, 18, 19, 20, 21.

Path III : 1, 2, 3, 4, 5, 6, 8, 10, 11, 5, 12, 13, 15, 16, 12, 17, 18, 19, 20, 21.

Path IV : 1, 2, 3, 4, 5, 12, 17, 18, 19, 20, 21.

Step 4 : The cyclomatic complexity can be computed as follows -

Method 1 : Number of edges (E) = 25

Number of vertices (N) = 21

Cyclomatic complexity = E – N + 2

= 25 – 21 + 2

= 6

Method 2 : The total number of regions in Fig. 4.4.9 = 6.

Hence cyclomatic complexity = 6.

Note that the area outside the graph is also treated as one region. Hence VI is marked as a region, along with all other closed loops.

Method 3 : Number of predicate nodes (P) = 5

Because node 5, 6, 8, 12, 13 are predicate nodes

∴ Cyclomatic complexity = P + 1

= 5 + 1

= 6

Thus the cyclomatic complexity is **6.**

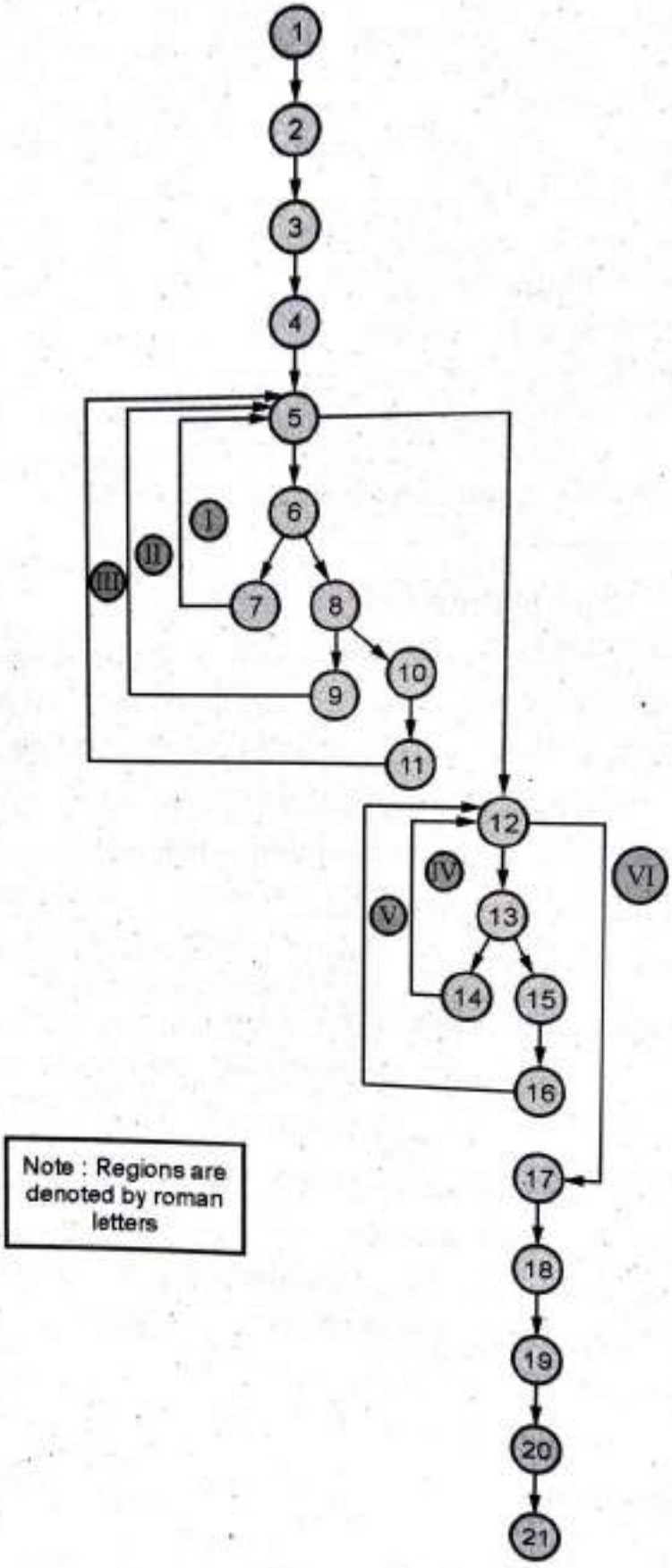

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
I
II
III
IV
V
VI
Note : Regions are denoted by roman letters

Fig. 4.4.9

Example 4.4.4 *Consider the following program segment.*

```
/* num is the number the function searches in a presorted integer array arr*/
int bin_search (int num)
{
int min, max; min = 0; max = 100;
while (min! = max)
if (arr[min + max]/2) > num
max = (min + max) /2;
else if (arr[(min + max)/2]
min = (min + max)/2;
else return ((min + max)/2);
}
return (- 1);
}
```

i) Draw the control flow graph for this program segment.

ii) Define cyclomatic complexity.

iii) Determine the cyclomatic complexity for this program. Show the intermediate steps in your computation. Writing only the final result is not sufficient.

AU : Dec.-17, Marks (2+2+9)

Solution : i) We will number the program statements as follows

```
int bin_search (int num)
{
1) int min, max; min = 0; max = 100;
2) while (min! = max)
3) if (arr[min + max]/2) > num
4) max = (min + max) /2;
5) else if (arr[(min + max)/2]
6) min = (min + max)/2;
7) else return ((min + max)/2);
8) }
9) return (- 1);
}
```

The Control Flow Graph(CFG) is as follows - (Refer Fig. 4.4.10)

ii) Cyclomatic complexity : The cyclomatic complexity is defined as the number of independent paths in the basis set of programs that provides the upper bound for the number of tests that must be conducted to ensure that all the statements have been executed atleast once.

iii) Cyclomatic complexity = E – N + 2 = 14 – 9 + 2 = 7

Total number of regions = 7

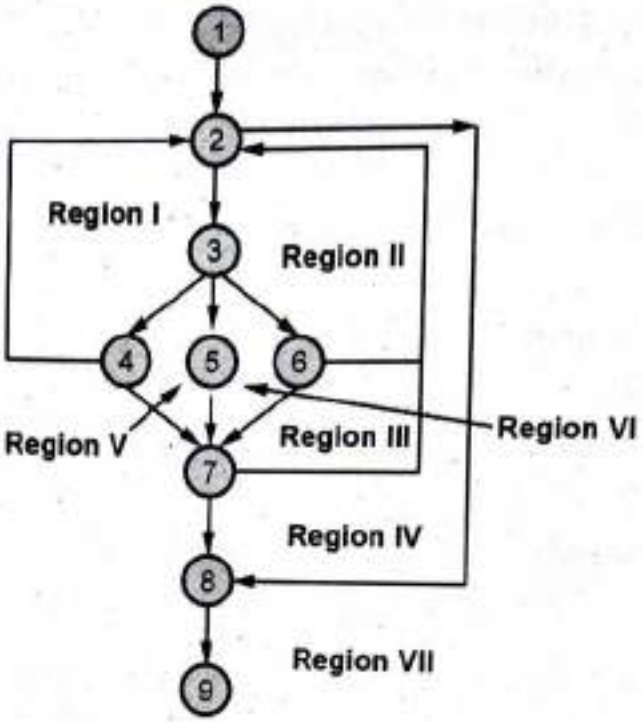


Fig. 4.4.10

Example 4.4.5 *Design test case for the following program :*

```
GCD(X,Y); if X = Y, then X
else if X>Y, then GCD(X–Y,Y)
else GCD(X,Y–X)
end if
end if
```

AU : May-07, Marks 8

Solution : The typical format of writing the test is as given below.

Precondition : If any precondition is required then mention it.

Test case id	Test case name	Test case description	Testing steps			Test case status (Pass/ Fail)	Test priority	Test severity
			Step	Expected result	Actual result			

Precondition : The two non-zero numbers should be entered for obtaining the GCD value.

Test case id	Test case name	Test case description	Testing steps			Test case status	Test priority	Test severity
			Step	Expected result	Actual result	(Pass/ Fail)		
1.	X = Y	If the value of both the numbers is same.	Enter two numbers such that X = Y.	The GCD is value of X.				
2.	X > Y	If first number is greater than the second one.	Enter two numbers such that value of X is > Y.	Call recursive routine by setting X = X – Y and Y as it is.				
3.	X<Y	If first number is less than the second one.	Enter two numbers such that value of X < Y.	Call recursive routine by setting X as it is and Y = Y – X.				

Example 4.4.6 *What are the attributes of a "good" test ? Explain the test case design.*

AU : CSE, May-08, Marks 10

Solution : Attributes of "good" test :

1. Finding high probability errors should be the goal of testing. For that purpose it is necessary to understand the software. The developer must have a judgement of where the software might fail.
2. Good test is not redundant. That means two different tests must not be conducted for the same purpose.
3. A good test is said to be "best of breed". The test should have highest likelihood of uncovering errors.
4. A good test must not be too simple not too complex.

Test case design : Refer section 4.4.3.

Example 4.4.7 *Consider the pseudo code for simple subtraction given below*

1) Program 'Simple substraction'
2) Input (x, y)
3) Output (x)
4) Output (y)
5) If x > y then D0
6) x – y = z
7) Else y – x = z
8) EndIf
9) Output (z)
10) Output "End Program"
Perform basis path testing and generate test cases.

AU : Dec.-16, Marks 10, May-17, Marks 6; May-18, Marks 9

Solution : Step 1 : We will first draw the flow graph for given pseudo code. This flow graph is as shown in Fig. 4.4.11.

Step 2 : We will compute cyclomatic complexity using following formula = e – V + 2

where e is total number of edges, V is total number of vertices

$\therefore \quad 9 - 9 + 2 = 2$

Now we will find two independent paths for basis path testing.

Step 3 :

Path 1 = 2 – 3 – 4 – 5 – 6 – 8 – 9 – 10

Path 2 = 2 – 3 – 4 – 5 – 7 – 8 – 9 – 10

Step 4 : The test cases for these paths are as given below

Independent path	x	y	Expected Result (z)
Path 1 2 – 3 – 4 – 5 – 6 – 8 – 9 – 10	10	5	5 End Program
Path 2 2 – 3 – 4 – 5 – 7 – 8 – 9 – 10	5	10	5 End Program
Path 2 2 – 3 – 4 – 5 – 7 – 8 – 9 – 10	5	5	0 End Program

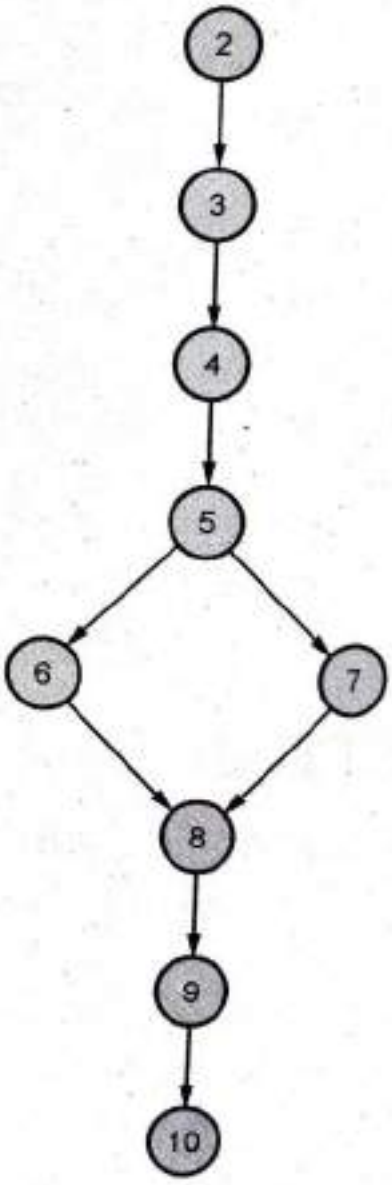


Fig. 4.4.11

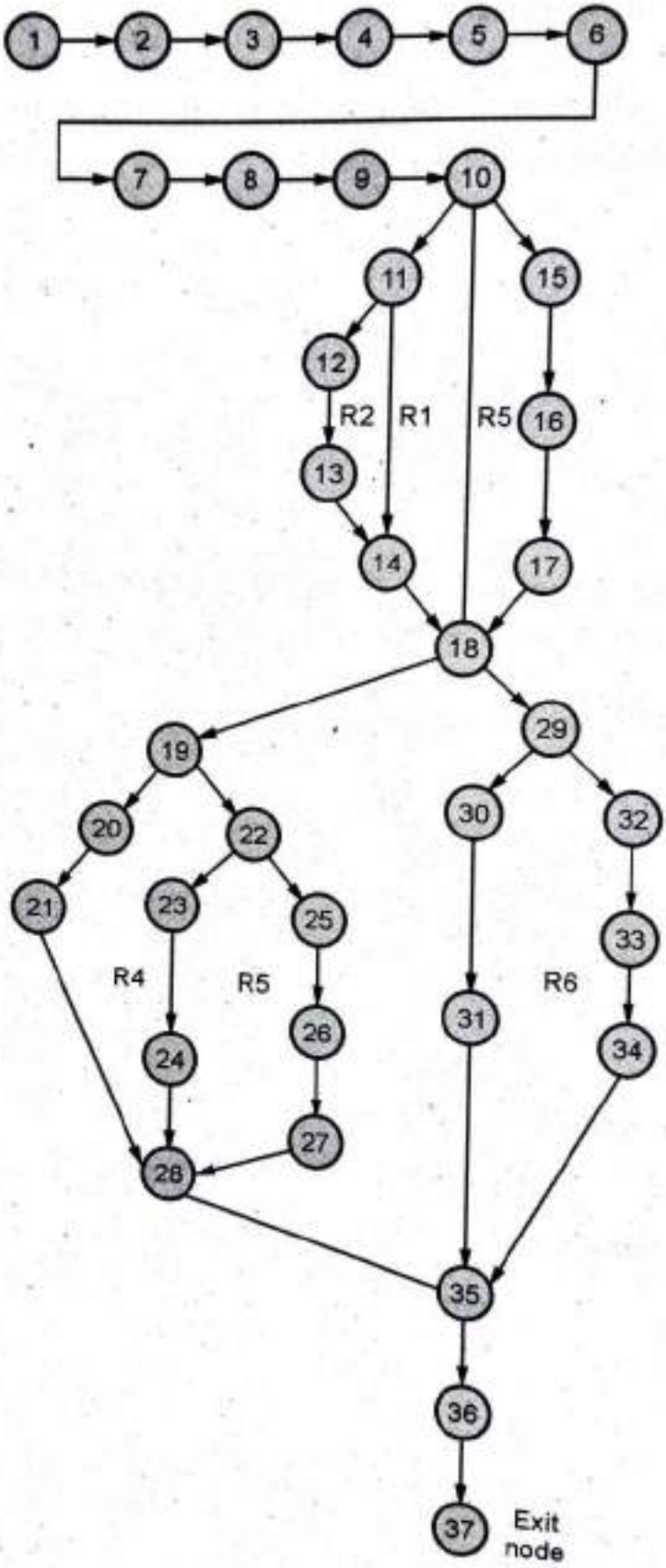
Example 4.4.8 *Write a procedure for the following : Given three sides of a triangle, return the type of triangle i.e. equilateral, isosceles and scalene triangle. Draw the control flow graph and calculate cyclomatic complexity to calculate the minimum number of paths. Enumerate the paths to be tested.*

AU : May-19, Marks 9

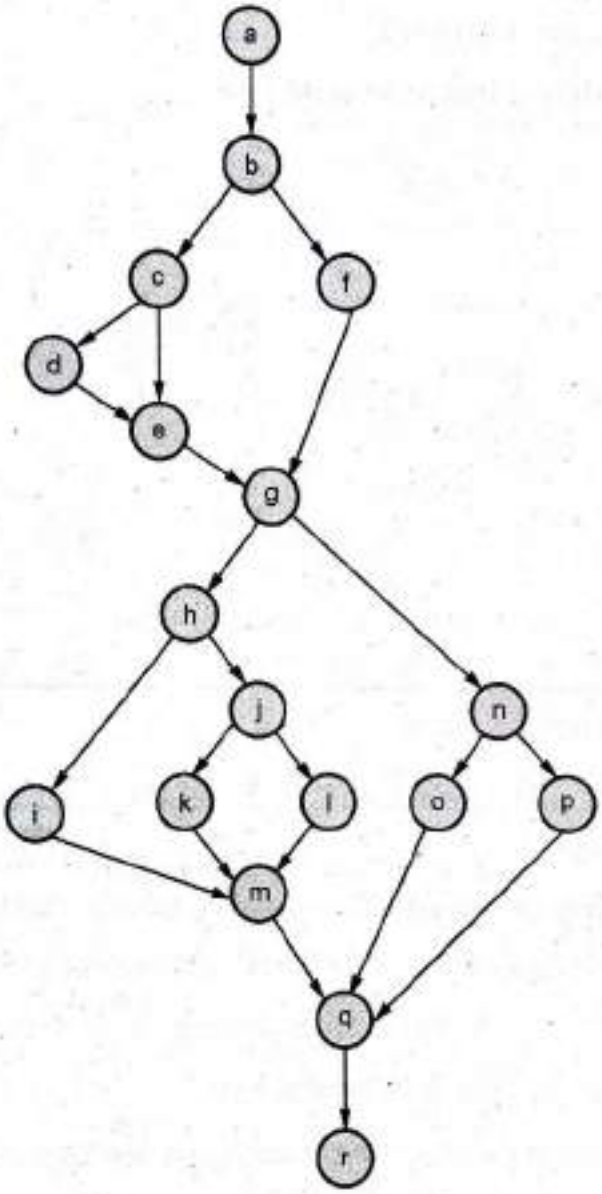
Solution : Step 1 : We will write the procedure for finding the type of triangle. Each important step is numbered so that we can draw the flow graph in the next step

```
(1) int main ( )
(2) {
(3) int a, b, c, status = 0;
(4) printf ("nt Enter side a :");
(5) scanf ("%d", & a);
(6) printf("nt Enter side b :");
(7) scanf ("%d", & b);
(8) printf ("nt Enter side c:");
(9) scanf ("'%d", &c);
(10) if ((a > 0) && (a < - 100) && (b > 0) && (b < . 100) && (c > 0) && (c < =100)) {
(11) if ((a + b) > c) && ((c + a) > b) && ((b + c) > a)) {
(12) status = 1;
(13) }
(14) }
(15) else {
(16) status = -1;
(17) }
(18) if (status = = 1) {
(19) if ((a = =b) && (b = =c)) {
(20) printf ("Triangle is equilateral");
(21) }
(22) else if ((a = =b) I I (b = = c) I I (c = = a)) {
(23) print ("Triangle is isosceles");
(24) }
(25) else {
(26) printf("Triangle is scalene");
(27) }
(28) }
(29) else if (status = = 0) {
(30) printf ("Not a triangle");
(31) }
(32) else
(33) printf ("Invalid input range");
(34) }
(35) getch ( );
(36) return -1;
(37) }
```

Step 2 : Now the flow graph is designed as follows -



Step 3 : The Decision to Decision path(DD Path) is drawn from the above flow graph. In this graph we only concentrate on decision nodes, the sequential nodes are combined together into one node. The DD path graph is as shown below -



Step 3 : Computing the cyclomatic complexity as follows -

Method 1 : Cyclomatic complexity = E – N + 2 where E is number of edges and N are number of nodes

= 23 – 18 + 2

= 7

Method 2 : Cyclomatic complexity = P + 1 where P is a predicate node

= 6 + 1 where b, c, g, h, j and n are predicate nodes

= 7

Method 3 : Cyclomatic complexity = Number of regions +1

= 6 + 1

= 7

Thus the cyclomatic complexity is 7

Step 4 : The independent paths to be tested are -

(1) abfgnpqr

(2) abfgnoqr

(3) abcegnpqr

(4) abcdegnoqr

(5) abfghimqr

(6) abfghjkmqr

(7) abfghjlmqr

Review Questions

1. *What is white box testing ? Explain.* **AU : May-17, Marks 7**
2. *Explain how the various types of loops are tested.* **AU : Dec.-17, Marks 9**

4.5 Testing Strategies

AU : May-05, 06, 22, Marks 16

- We begin by **'testing-in-the-small'** and move toward **'testing-in-the-large'**.
- Various testing strategies for conventional software are
  1. Unit testing
  2. Integration testing
  3. Validation testing
  4. System testing

1. **Unit testing** - In this type of testing techniques are applied to detect the errors from each software component individually.
2. **Integration testing** - It focuses on issues associated with verification and program construction as components begin interacting with one another.
3. **Validation testing** - It provides assurance that the software validation criteria (established during requirements analysis) meets all functional, behavioural and performance requirements.
4. **System testing** - In system testing all system elements forming the system is tested as a whole.

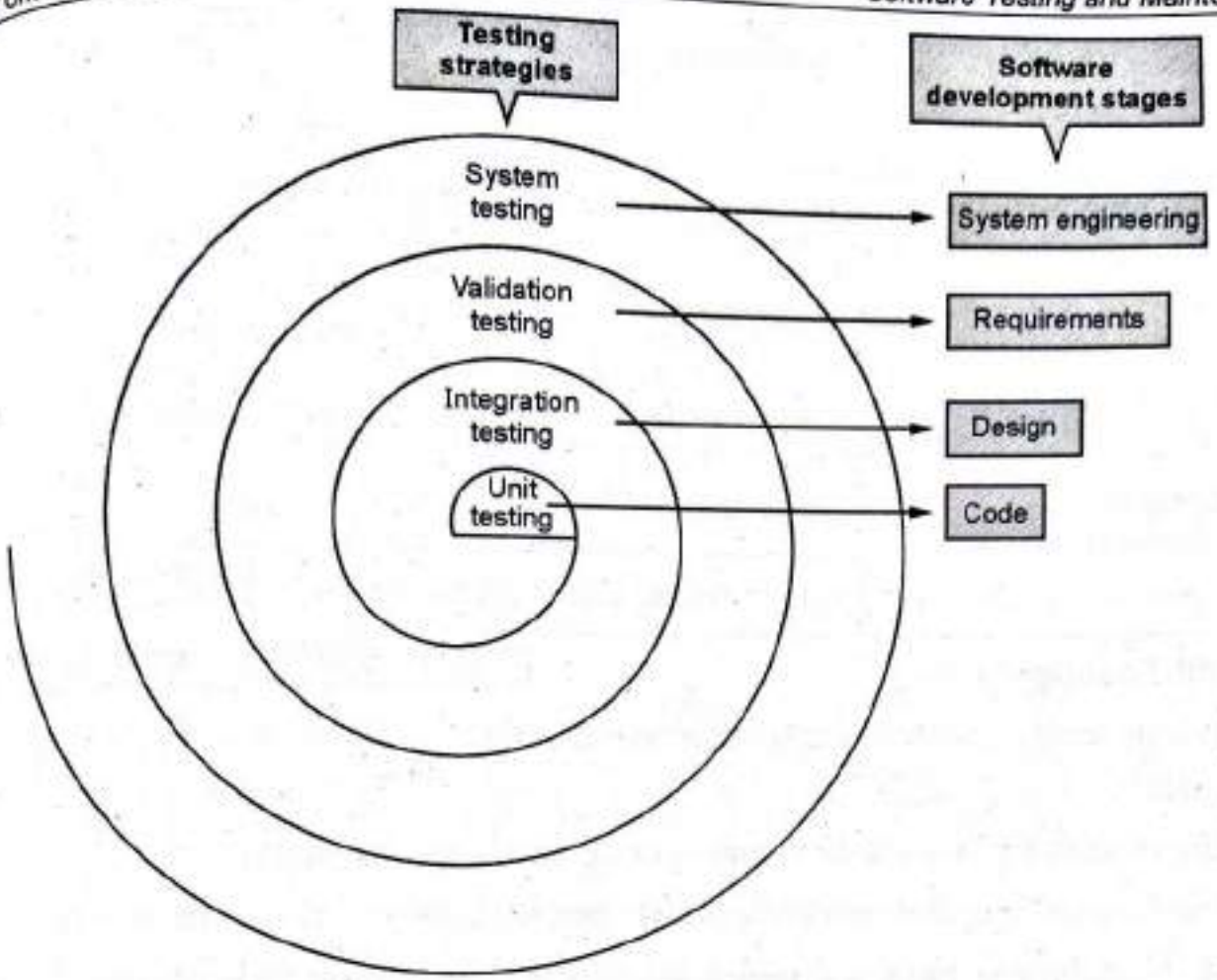


Fig. 4.5.1 Testing strategy

Difference between black box testing and White box testing

Sr. No.	Black box testing	White box testing
1.	Black box testing is the software testing method which is used to test the software **without knowing the internal structure** of code or program.	White box testing is the software testing method in which **internal structure** is being **known** to tester who is going to test the software.
2.	This type of testing is **carried out by testers.**	Generally, this type of testing is carried out **by software developers.**
3.	**Implementation knowledge is not required** to carry out black box testing.	**Implementation knowledge is required** to carry out white box testing.
4.	Programming knowledge is not required to carry out black box testing.	Programming knowledge is required to carry out white box testing.
5.	Testing is applicable on **higher levels of testing** like system testing, acceptance testing.	Testing is applicable on **lower level of testing** like unit testing, integration testing.

6.	Black box testing means functional test or **external testing.**	White box testing means structural test or **interior testing.**
7.	Black box testing can be started based on **requirement specifications documents.**	White box testing can be started based on **detail design documents.**
8.	The functional testing, behavior testing, **close box testing** is carried out under black box testing.	The Structural testing, logic testing, path testing, loop testing, code coverage testing, open box testing is carried out under white box testing.

Review Question

1. *Explain the various levels of software testing with suitable examples.* **AU : May-22, Marks 7**

4.6 Unit Testing

AU : Dec.-06,08,15, May-03,09, Marks 16

- In unit testing the individual components are tested independently to ensure their quality.
- The focus is to uncover the errors in design and implementation.
- The various tests that are conducted during the unit test are described as below.
  1. Module interfaces are tested for proper information flow in and out of the program.
  2. Local data are examined to ensure that integrity is maintained.
  3. Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.
  4. All the basis (independent) paths are tested for ensuring that all statements in the module have been executed only once.
  5. All error handling paths should be tested.

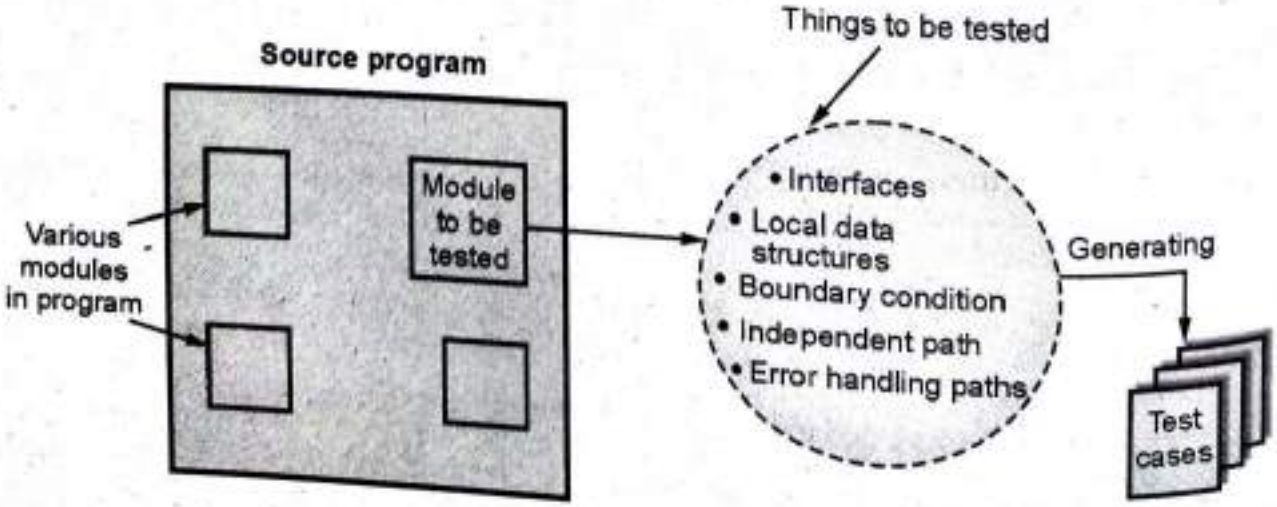


Fig. 4.6.1 Unit testing

6. Drivers and stub software need to be developed to test incomplete software. The "driver" is a program that accepts the test data and prints the relevant results. And the "stub" is a subprogram that uses the module interfaces and performs the minimal data manipulation if required. This is illustrated by following Fig. 4.6.2.

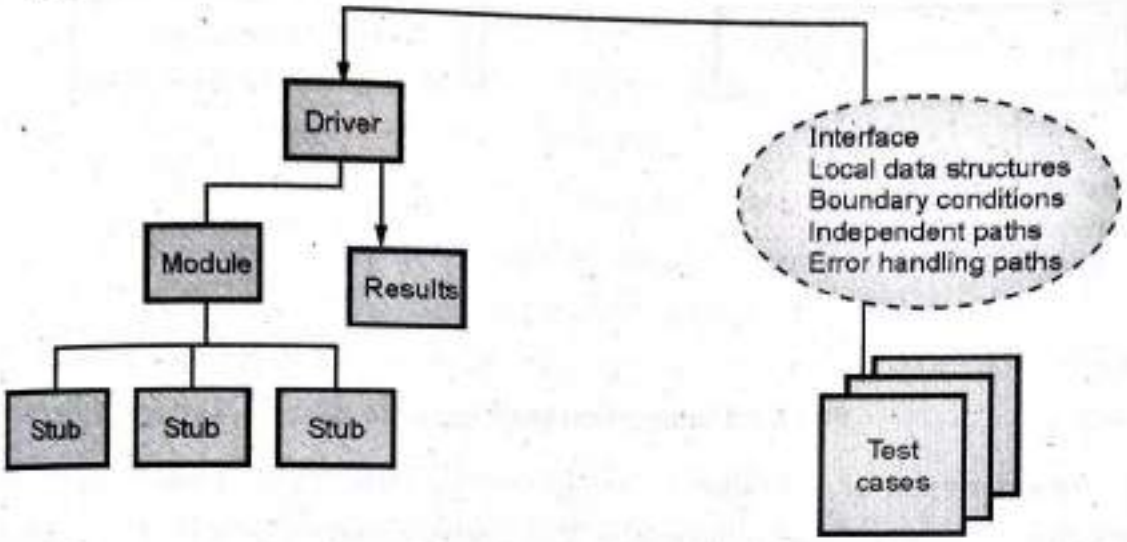


Fig. 4.6.2 Unit testing environment

7. The unit testing is simplified when a component with high cohesion (with one function) is designed. In such a design the number of test cases are less and one can easily predict or uncover errors.

4.7 Integration Testing

AU : Dec.-06,08,15,19, May-03,04,05,09,13,14,15,17,18, Marks 16

- A group of dependent components are tested together to ensure their quality of their integration unit.
- The objective is to take unit tested components and build a program structure that has been dictated by software design.
- The focus of integration testing is to uncover errors in :
  - Design and construction of software architecture.
  - Integrated functions or operations at subsystem level.
  - Interfaces and interactions between them.
  - Resource integration and/or environment integration.
- The integration testing can be carried out using two approaches.

1. The non-incremental integration 2. Incremental integration.

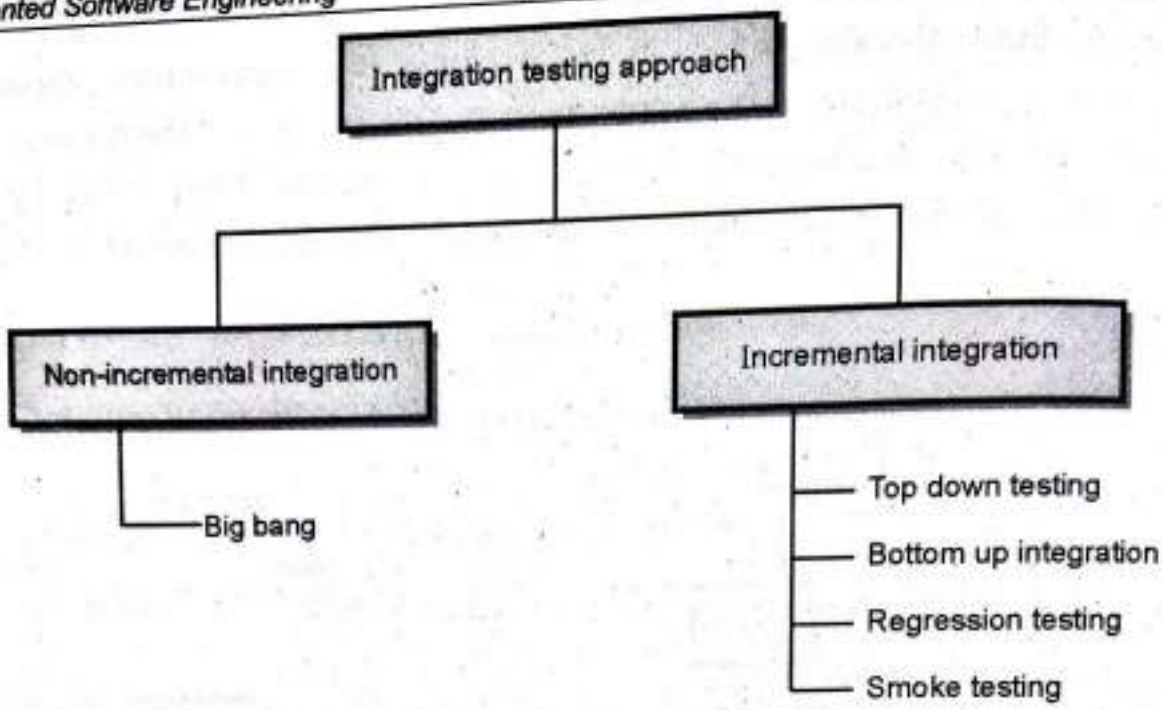


Fig. 4.7.1 Integration testing approach

- The non-incremental integration is given by the "**big bang**" approach. All components are combined in advance. The entire program is tested as a whole. And chaos usually results. A set of errors is tested as a whole. Correction is difficult because isolation of causes is complicated by the size of the entire program. Once these errors are corrected new ones appear. This process continues infinitely.

Advantage of big-bang : This approach is simple.

Disadvantages :

1. It is hard to debug.
2. It is not easy to isolate errors while testing.
3. In this approach it is not easy to validate test results.
4. After performing testing, it is impossible to form an integrated system.

- An incremental construction strategy includes,

  Top down integration

  Bottom up integration

  Regression testing

  Smoke testing.

The outcome of integration testing is

- Errors in design and construction of software architecture.
- Errors from integrated functions or operations at sub-system level.
- Errors from interfaces and interactions between them.
- Errors in resource integration and environment integration.

- The system test is a series of tests conducted to fully exercise the computer based system.

4.7.1 Top Down Integration Testing

- Top down testing is an incremental approach in which modules are integrated by moving down through the control structure.
- Modules subordinate to the main control module are incorporated into the system in either a depth first or breadth first manner.
- Integration process can be performed using following steps.
  1. The main control module is used as a test driver and the stubs are substituted for all modules directly subordinate to the main control module.
  2. Subordinate stubs are replaced one at a time with actual modules using either depth first or breadth first method.
  3. Tests are conducted as each module is integrated.
  4. On completion of each set of tests, another stub is replaced with the real module.
  5. Regression testing is conducted to prevent the introduction of new errors.

For example :

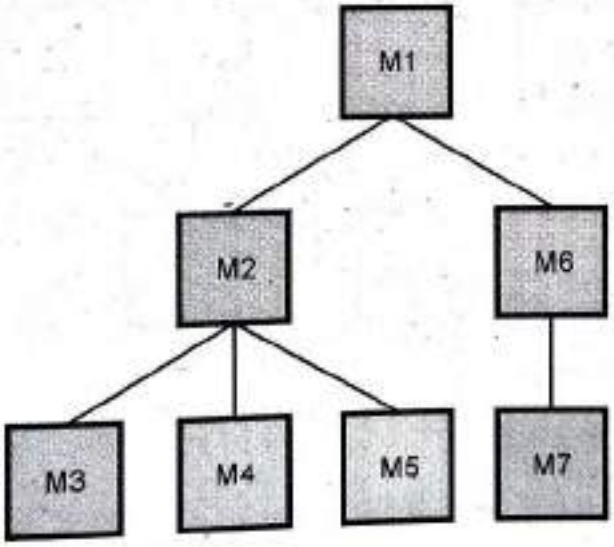


Fig. 4.7.2 Program structure

In top down integration if the depth first approach is adopted then we will start integration from module M1 then we will integrate M2 then M3, M4, M5, M6 and then M7.

If breadth first approach is adopted then we will integrate module M1 first then M2, M6. Then we will integrate module M3, M4, M5 and finally M7.

4.7.2 Bottom Up Integration Testing

In bottom up integration the modules at the lowest levels are integrated at first, then integration is done by moving upward through the control structure.

The bottom up integration process can be carried out using following steps.

1. Low-level modules are combined into clusters that perform a specific software subfunction.
2. A driver program is written to co-ordinate test case input and output.
3. The whole cluster is tested.
4. Drivers are removed and clusters are combined moving upward in the program structure.

For example :

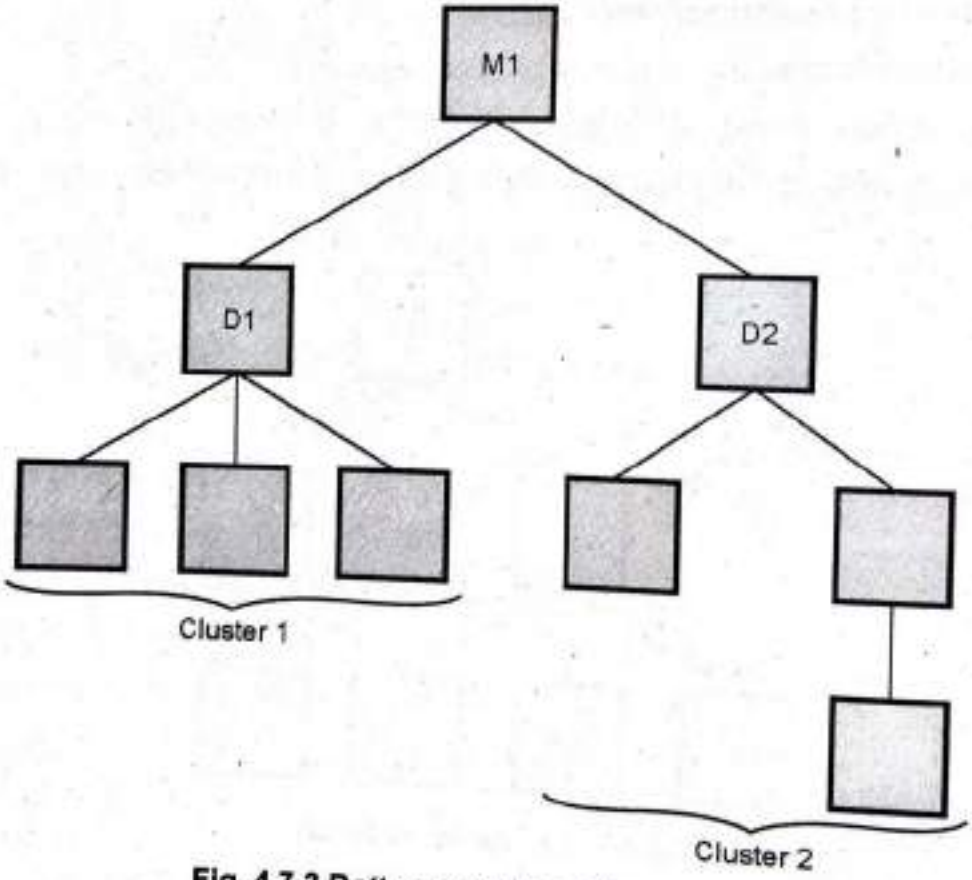


Fig. 4.7.3 Bottom up integration testing

First components are collected together to form cluster 1 and cluster 2. Then each cluster is tested using a driver program. The clusters subordinate the driver module. After testing the driver is removed and clusters are directly interfaced to the modules.

Difference between top down and bottom up integration testing :

Sr. No.	Top-down Integration	Bottom-up Integration
1.	The major controls or decisions are verified at early stage itself.	After integrating all the components at the bottom level, major contols or decisions can be verified.
2.	Testing can be performed from the early stage.	Individual components can be tested at higher level, but after integration of low level components into cluster testing is required.
3.	In top-down testing, testing stubs are created.	In botom-up testing, test drivers are required.
4.	Working system can be available at early stage.	Working system is available only after integrating all the components.

4.7.3 Regression Testing

- Regression testing is used to check for defects propagated to other modules by changes made to existing program. Thus regression testing is used to reduce the side effects of the changes.
- There are three different classes of test cases involved in regression testing -
  - Representative sample of existing test cases is used to exercise all software functions.
  - Additional test cases focusing software functions likely to be affected by the change.
  - Tests cases that focus on the changed software components.
- After product had been deployed, regression testing would be necessary because after a change has been made to the product an error that can be discovered and it should be corrected. Similarly for deployed product addition of new feature may be requested and implemented. For that reason regression testing is essential.

4.7.4 Smoke Testing

- The smoke testing is a kind of integration testing technique used for time critical projects wherein the project needs to be assessed on frequent basis.
- Following activities need to be carried out in smoke testing -
  1. Software components already translated into code are integrated into a "build". The "build" can be data files, libraries, reusable modules or program components.
  2. A series of tests are designed to expose errors from build so that the "build" performs its functioning correctly.

3. The "build" is integrated with the other builds and the entire product is smoke tested daily.

Smoke testing benefits

1. Integration risk is minimized.
2. The quality of the end product is improved.
3. Error diagnosis and correction are simplified.
4. Assessment of progress is easy.

Review Questions

1. *What is meant by integration testing and system testing ? Explain. Discuss on their outcomes.* AU : May-03, 04, 05, Marks 16
2. *What is integration testing ? Discuss any one method in detail.* AU : May-17, Marks 8
3. *Explain unit testing and integration testing process with an example.* AU : Dec.-15, Marks 16
4. *Write short note on - Regression testing.* AU : May-13, Marks 8, May-18, Marks 6
5. *Elaborate path testing and regression testing with an example.* AU : Dec.-19, Marks 13
6. *Discuss about the integration techniques with suitable example.* AU : Dec.-22, Marks 7

4.8 System Testing

AU : May-03,04,05, Dec.-14, Marks 16

The system test is a series of tests conducted for fully the computer based system.

Various types of system tests are

1. Recovery testing
2. Security testing
3. Stress testing
4. Performance testing

The main focus of such testing is to test

- System functions and performance.
- System reliability and recoverability (recovery test).
- System installation (installation test).
- System behaviour in the special conditions (stress test).
- System user operations (acceptance test/alpha test).
- Hardware and software integration and collaboration.
- Integration of external software and the system.

4.8.1 Recovery Testing

- Recovery testing is intended to check the system's ability to recover from failures.
- In this type of testing the software is forced to fail and then it is verified whether the system recovers properly or not.
- For automated recovery then reinitialization, checkpoint mechanisms, data recovery and restart are verified.

4.8.2 Security Testing

- Security testing verifies that system protection mechanism prevent improper penetration or data alteration.
- It also verifies that protection mechanisms built into the system prevent intrusion such as unauthorized internal or external access or willful damage.
- System design goal is to make the penetration attempt more costly than the value of the information that will be obtained.

4.8.3 Stress Testing

- Determines breakpoint of a system to establish maximum service level.
- In stress testing the system is executed in a manner that demands resources in abnormal quantity, frequency or volume.
- A variation of stress testing is a technique called sensitivity testing.
- The sensitive testing is a testing in which it is tried to uncover data from a large class of valid data that may cause instability or improper processing.

4.8.4 Performance Testing

- Performance testing evaluates the run time performance of the software, especially real time software.
- In performance testing resource utilization such as CPU load, throughput, response time, memory usage can be measured.
- For big systems (e.g. banking systems) involving many users connecting to servers (e.g. using internet) performance testing is very difficult.
- Beta testing is useful for performance testing.

Review Question

1. *What is system testing ? Discuss types of system tests.* **AU : Dec.-14, Marks 12**

4.9 Validation Testing

AU : May-05, 16, Marks 16

- The integrated software is tested based on requirements to ensure that the desired product is obtained.
- In validation testing the main focus is to uncover errors in
  - System input/output
  - System functions and information data
  - System interfaces with external parts
  - User interfaces
  - System behaviour and performance.
- Software validation can be performed through a series of **black box tests.**
- After performing the validation tests there exists two conditions.
  1. The function or performance characteristics are **according** to the **specifications** and are accepted.
  2. The requirement specifications are derived and the deficiency list is created. The **deficiencies** then can be **resolved** by establishing the proper communication with the customer.
- Finally in validation testing a review is taken to ensure that all the elements of software configuration are developed as per requirements. This review is called configuration review or audit.

4.9.1 Acceptance Testing

The acceptance testing is a kind of testing conducted to ensure that the software works correctly in the user work environment.

The acceptance testing can be conducted over a period of weeks or months.

The types of acceptance testing are

1. **Alpha test** - The alpha testing is a testing in which the version of complete software is tested by the customer under the supervision of developer. This testing is performed at developer's site. The software is used in natural setting in presence of developer. This test is conducted in controlled environment.

Advantages

1. The users of the system get systematic training to use the new system.
2. The acceptance of the system can be immediately known to the developers.

3. The primary problems in the identified and fixed in immediately before delivery of the product to the customer.
4. Any mis-understanding about the scope of the project or in requirements given by the customer can be cleared during this type of testing.

Disadvantages

1. In depth functionality cannot be tested as software is still under development stage. Sometimes developers and testers are dissatisfied with the results of alpha testing.
2. Alpha testing is conducted in lab environment or testing environment which may not be similar to real-life environment.
3. Data used during this testing may not be similar to the business data or actual data. Hence false image of working module can be created and actual problems remain hidden.

2. **Beta test** - The beta testing is a testing in which the version of software is tested by the customer without the developer being present. This testing is performed at customer's site. As there is no presence of developer during testing, it is not controlled by developer. The end user records the problems and report them to developer. The developer then makes appropriate modification.

Advantages :

1. Beta testing improves the product quality via customer feedback.
2. Any training requirement for use of the product can be identified during beta testing.
3. The testing is done in actual work environment without the interference of developer. Hence addition in requirements, changes in the requirements can be identified.
4. The security loopholes can be identified by the customer itself.
5. Beta testing allows the organization to test post-launched infrastructure.
6. Due to customer feedback product failure risk can be reduced.
7. Usability of the application can be identified and verified by the customer.

Disadvantages :

1. Finding right beta user and maintaining his participation can be real challenge.
2. Tests can not be management because beta testing is conducted in actual working environment and not in the organization.

Review Question

1. *Compare and contrast alpha and beta testing.* AU : May-16, Marks 8

4.10 Debugging Techniques

AU : Dec.-10,14, May-15, 18, Marks 4

Debugging is a process of removal of a defect. It occurs as a consequence of successful testing.

Debugging process starts with execution of test cases. The actual test results are compared with the expected results. The debugging process attempts to find the lack of correspondence between actual and expected results. The suspected causes are identified and additional tests or regression tests are performed to make the system to work as per requirement.

Common approaches in debugging are :

Brute force method - The memory dumps and run-time traces are examined and program with write statements is loaded to obtain clues to error causes.

In this method "Let computer find the error" approach is used.

This is the least efficient method of debugging.

Backtracking method - This method is applicable to small programs.

In this method, the source code is examined by looking backwards from symptom to potential causes of errors.

Cause elimination method - This method uses binary partitioning to reduce the number of locations where errors can exist.

Why debugging is so difficult ?

Following are some reasons that reveal why debugging is so difficult -

1. The symptoms of bug may be present at some part(module) of the program and its effect might be seen in some other module of the program. Hence tracing out the location of symptom becomes difficult.
2. Symptoms may be caused by software developers during the development process. Such symptoms are difficult to trace out.
3. The symptom may appear due to timing problems instead of processing problem.
4. If the developer corrects some error then the symptom may disappear temporarily.
5. The symptom can appear if some in-accuracies in the program are simply rounded off.

6. In real time systems, it is not possible to accurately reproduce the input conditions and this may lead to symptoms of bugs.

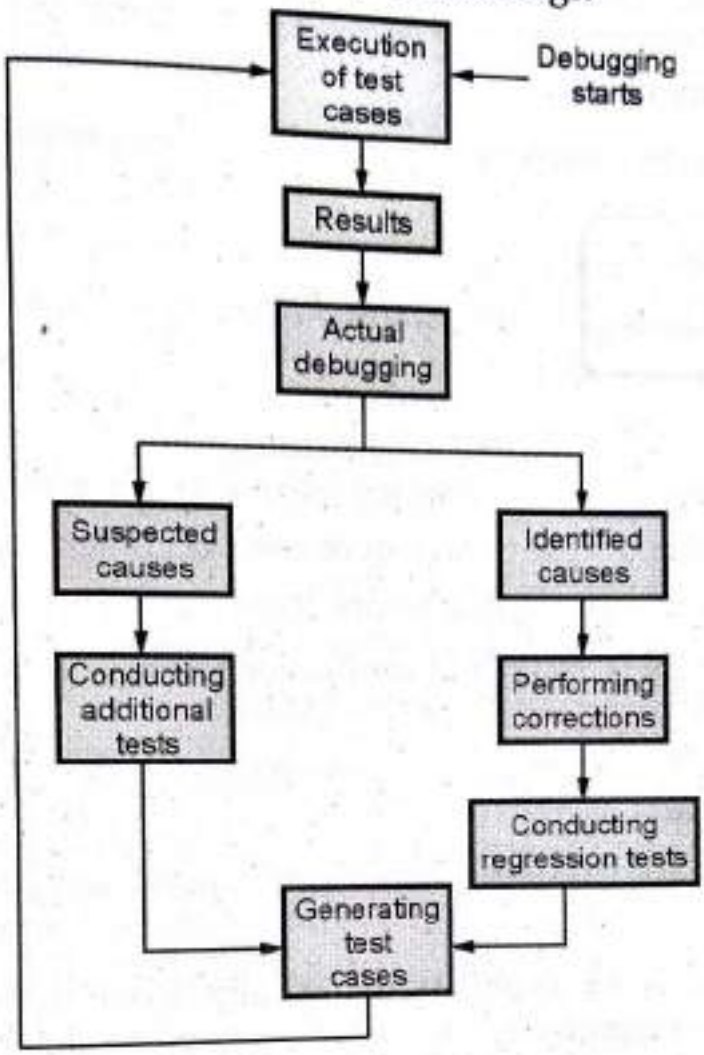


Fig. 4.10.1 Debugging process

7. The symptom may appear periodically. Such things normally occur in embedded systems in which hardware and software is coupled.

8. In distributed systems number of tasks are running on several distinct processors which may lead to symptoms.

4.10.1 Testing Vs. Debugging

Sr. No.	Testing	Debugging
1.	Testing is a process in which the bug is identified.	Debugging is the process in which the bug or error is corrected by the programmer.
2.	In testing process, it is identified where the bug occurs.	In debugging the root cause of error is identified.
3.	Testing starts with the execution results from the test cases.	Debugging starts after the testing process.

Review Question

1. *Write short note on - Debugging.* **AU : May-18, Marks 6**

4.11 Program Analysis

There are two ways by which the program analysis is carried out -

1) Static analysis
2) Dynamic analysis

➢ 1) Static analysis

- **Definition :** Static testing is a testing technique in which software is tested without executing the code. As the code, requirement documents, and design documents are tested manually in order to find errors, it is called static.
- This kind of testing is also called verification testing.

Static analysis testing techniques :

- **Informal reviews :**
  - In this technique, the team of reviewers just checks the documents and gives comments.
  - The purpose is to maintain the quality from the initial stage. It is non-documented in nature.
- **Formal reviews :**
  - It is well structured and documented and follows six main steps : Planning, kick-off, preparation, review meeting, rework follow-up.

  Technical reviews :

  The team of technical experts will review the software for technical specifications. The purpose is to pin out the difference between the required specification and product design and then correct the flaws. It focuses on technical documents such as test strategy, test plan and requirement specification documents.
- **Walk-through :**
  - The author explains the software to the team and teammates can raise questions if they have any. It is headed by the author and review comments are noted down.
- **Inspection process :**
  - The meeting is headed by a trained moderator. A formal review is done, a record is maintained for all the errors and the authors are informed to make rectifications on the given feedbacks.

- **Static code review :**
  - Code is reviewed without execution, it is checked for syntax, coding standards and code optimization. It is also referred to as white box testing.

Advantages :

1) It is a fast and easy technique used to fix errors.
2) It helps in identifying flaws in code.
3) With the help of automated tools it becomes very easy and convenient to scan and review the software.
4) With static testing it is possible to find errors at an early stage of the development life cycle.

Disadvantages :

1) It takes a lot of time to conduct the testing procedure if done manually.
2) Automated tools work for a restricted set of programming languages.
3) The automated tools simply scan the code and can not test the code deeply.

2) Dynamic analysis

Definition : Dynamic testing is a process by which code is executed to check how software will perform in a runtime environment. As this type of testing is conducted during code execution it is called dynamic. It is also called **validation testing**.

Dynamic testing techniques

- **Unit testing :** As the name suggests individual units or modules are tested. The source code is tested by the developers.
- **Integration testing :** Individual modules are clubbed and tested by the developers. It is performed in order to ensure that modules are working in the right manner and will continue to perform flawlessly even after integration.
- **System testing :** It is performed on a complete system to ensure that the application is designed according to the requirement specification document.

Advantages

1) It identifies weak areas in a runtime.
2) It helps in performing detailed analysis of code.
3) It can be applied with any application.

Disadvantages

1) It is not easy to find a trained software tester to perform dynamic testing.
2) It becomes costly to fix errors in dynamic testing.

Difference between static and dynamic analysis :

Sr. No.	Static analysis	Dynamic analysis
1.	Static analysis is a testing process that is done at the early stage of the development life cycle.	Dynamic analysis is a testing process done later stage of the development life cycle.
2.	It involves a walkthrough and code review.	It involves functional and non-functional testing.
3.	This testing is a verification process.	This testing is a validation process.
4.	This type of testing is done before the execution of code.	This type of testing is done during code execution.
5.	It is about prevention.	It is about a cure.
6.	It is cost-effective.	It is costly.

4.12 Symbolic Execution

- Symbolic execution is a testing technique in which we use symbols to see what is the output of the code without actually running the code. This helps to find and fix the errors in the software without doing a real test.
- Symbolic execution is a technique for testing software by symbolically evaluating the program's control flow and data flow. It is a type of dynamic analysis, which means that it tests the program by executing it with symbolic input values instead of concrete input values.
- Symbolic execution involves analyzing the code without running it with actual values.
- For example -

```
function divide(int a, int b)
{
    if(b!=0)
        result = a/b;
    else
        result = -1;
    return result
}
```

In above code,

The **result** is a symbolic variable we can analyze the code's behavior in both cases without having specific values for 'a' and 'b' at this stage. This helps us understand how the code will react to various scenarios.

- Symbolic execution is often used in more complex programs to find the major bugs.
- This technique also helps us to understand how the code responds to different inputs without running it with actual data.
- It is an important technique for debugging and testing software to identify possible problems and ensure robustness.

Review Question

1. *Write a short note on - Symbolic execution.*

4.13 Model Checking

Model checking is a technique for verifying that a software system satisfies the desired properties. In this technique, a model of the system is created and then using a model checker we exhaustively explore all possible states of the model to check the properties of the system.

Model checking can be used in both software testing and debugging. In software testing, it can be used to verify that the system meets its requirements. In software debugging, it can be used to identify the root cause of a bug.

Once the model and the desired properties have been specified, a model checker can be used to verify that the model satisfies the properties.

For example - Consider a ATM system in which we have to design a function for withdrawal of money say **withdraw()**. This function allows the user to withdraw money from his/her account. In order to implement this function, following requirements need to be satisfied -

1. The user must have a positive balance in their account in order to withdraw money.
2. The user cannot withdraw more money than they have in their account.
3. The user's balance must be updated after each withdrawal.

Before actually executing the code, we can use model checking technique to verify that the withdraw() function meets these requirements. In this technique, we first create a model of the **withdraw()** function. We must specify the above requirements as **properties** of that model. The model checker would explore all possible states of the function to

verify that the function satisfies all these properties. For instance - If there is a negative balance, then there must not be a path to 'withdraw amount' state. Similarly if the balance amount greater than the withdrawal amount then only there should be a path to 'withdraw amount' state. If the model checker finds a state in which the function does not update the user's balance after a withdrawal then it should report a bug. By identifying this issue in our model, we can locate the corresponding code in our software and correct the logic to prevent this scenario from happening.

Model checking is particularly valuable for complex systems, where manually testing all possible combinations and paths would be impractical.

4.14 Two Marks Questions with Answers

Q.1 State the guidelines for debugging. AU : Dec.-04, 05, May-03

Ans. : Following are some debugging guideline suggested by Myers

1. Debugging can be effectively carried out by mental analysis.
2. If an error cannot be located by yourself, then describe the problem to someone else.
3. Use debugging tool only if no alternative is present.
4. Avoid experimentation.
5. If one bug occurs in one area of code then there are chances of occurrence of more bugs.
6. Fix the error and not its symptoms.
7. For correcting the existing program, if a new code is added to it then test it rigorously than the original program.
8. There are chances of introducing new errors on correction of older ones.
9. The process error repair must be conducted in parallel during design phase.
10. Change the source code and not the object code/machine code.

Q.2 What are static and dynamic software testing ? AU : Dec.-05

Ans. : The verification activities fall into the category of static testing. During static testing, you have a checklist to check whether the work you are doing is going as per the set standards of the organization.

The dynamic testing is a method in which the actual testing is done to uncover all possible errors. Various testing strategies such as unit testing, integration testing, validation testing, system testing can be applied while performing dynamic testing.

Q.3 What is stress testing ? AU : May-03

Ans. : Stress testing is a testing for a system which is executed in a manner that demands resources in abnormal quantity, frequency or volume.

A variation of stress testing is a technique called sensitivity testing.

Q.4 What is partial integration testing ? AU : Dec.-03

Ans. : Partial integration testing verifies correct data movement from one external system to another.

For instance : If there is one system that brings data from single external system and then sends the data to several other external systems. Then partial test of this system would be to test whether the data can be sent to one of the external systems.

Q.5 Why testing is important with respect to software ? AU : May-04

Ans. : The purpose of software testing is to ensure whether the software functions appear to be working according to specifications and performance requirements. This ultimately helps in improving the overall quality of the software.

Q.6 How regression and stress tests are performed ? AU : Dec.-04

Ans. : Regression testing is used to check for defects propagated to other modules by changes made to existing program. Thus regression testing is used to reduce the side effects of the changes.

Stress testing determines breakpoint of a system to establish maximum service level. In stress testing the system is executed in a manner that demands resources in abnormal quantity, frequency or volume.

Q.7 Define cyclomatic complexity. AU : May-06, 14

Ans. : Cyclomatic complexity is kind of software metric that gives the quantitative measure of logical complexity of a program. In basis path testing method the cyclomatic complexity defines the number of independent paths of the program structure.

Q.8 Write the types of system tests. AU : May-06

Ans. : Various types of system tests are

1. Recovery testing
2. Security testing
3. Stress testing
4. Performance testing

Q.9 Write short note on equivalence partitioning. AU : May-06

Ans. : In equivalence partitioning the equivalence classes are evaluated for given input condition. Equivalence class represents a set of valid or invalid states for input conditions.

It is a black-box technique that divides the input domain into classes of data. From this data test cases can be derived.

Q.10 What is white box testing and what is the difficulty while exercising it ? AU : May-05

Ans. : In white box testing the procedural details are closely examined. In this testing the internals of software are tested to make sure that they operate according to specifications and designs. Thus major focus of white box testing is on internal structures, logic paths, control flows, data flow, internal data structures, conditions, loops, etc.

This kind of testing will be exhaustive testing. This type of testing will keep the processor continuously busy. And software developer will find it difficult to manage the schedule of the project with white box testing. The white box testing is impossible for large system.

Q.11 What is behavioural testing ? AU : May-06

Ans. : The black box testing is also called as behavioural testing.

Black box testing methods focus on the functional requirements of the software. Tests sets are derived that fully exercise all functional requirements.

Q.12 What is cyclomatic complexity? How to calculate this ? AU : Dec.-06

Ans. : The cyclomatic complexity defines the number of independent paths in the basis set of the program that provides the upper bound for the number of tests that must be conducted to ensure that all the statements have been executed atleast once.

The cyclomatic complexity can be computed by one of the following ways.

1. The number of regions of the flow graph correspond to the cyclomatic complexity.
2. Cyclomatic complexity, V(G), for a flow graph, G, is defined as :

$$V(G) = E - N + 2,$$

E = Number of flow graph edges

N = Number of flow graph nodes

3. $$V(G) = P + 1$$

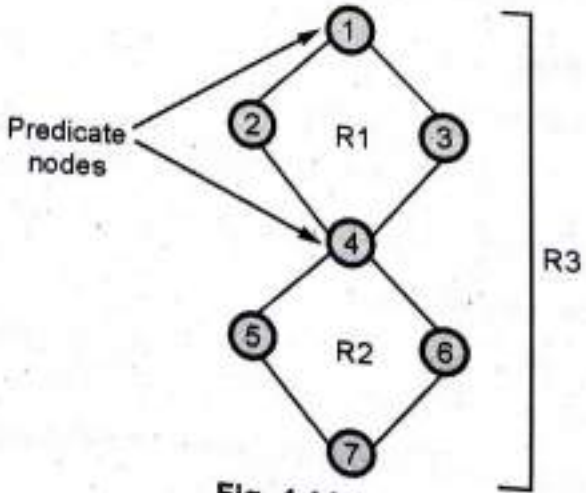


Fig. 4.14.1

where P is the number of predicate nodes contained in the flow graph G.

Consider the flow graph as shown below.

The cyclomatic complexity can be computed as -

1. The flow graph has 3 regions.
2. V(G) = 8 edges – 7 nodes + 2 = 3
3. V(G) = 2 predicate nodes + 1 = 3

Hence cyclomatic complexity is 3.

Q.13 Write the steps involved in testing real time systems. AU : May-07

Ans. : Following are the steps involved in testing real time systems.

1. **Task testing** : This is the first step in testing real time systems. In this step, each task is independently tested. Task testing uncovers the errors in logic and function. But it does not find errors from behaviour.
2. **Behavioural testing** : With the help of automated tool the behaviour of the real time system is simulated. Then it becomes possible to examine the behaviour of the system as a series of external events.
3. **Inter task testing** : After finding and removing the errors from individual task and behaviour of the system the next step is to obtain errors from asynchronous tasks that may communicate with one another. Such type of testing is called inter task testing.
4. **System testing** : When all the required hardware and software is integrated then a full range of testing is carried out in order to uncover errors at hardware and software interface. As real time system is based on interrupts therefore testing the Boolean events is necessary.

Q.14 What is the objective of unit testing ? AU : Dec.-06

Ans. : The objective of unit testing is to test individual components independently to ensure their quality. Thus the focus is to uncover the errors in design and implementation.

Q.15 Assume a program for computing the roots of a quadratic equation. List out the test cases using equivalence partitioning method. AU : May-07

Ans. : To write the test cases for roots of quadratic equations we will first write a program for computing the roots for quadratic equation -

```
void Roots_of_QE (float a, float b, float c)
{
  float d;
  d = (b * b) - (4 * a * c);
  if (d < 0)
  {
    printf ("Roots are complex");
    r1 = (-b + sqrt (d)) / (2 * a);
```

```
    r2 = (– b – sqrt (d)) / (2 * a);
    printf ("%f %f ", r1, r2);
   }
   else if (d = = 0)
   {
    printf (" Single real root");
    r1 = – b / 2 * a;
    printf ("%f ", r1);
   }
   else
   {
    printf (" Two real roots");
    r1 = (– b + sqrt (d)) / (2 * a);
    r2 = (– b – sqrt (d)) / (2 * a);
    printf (" %f %f ", r1, r2);
   }
}
```

Now the test cases can be written as follows.

Precondition : The quadratic equation $ax^2 + bx + c$ is to be entered. This will set a, b and c.

Test case id	Test case name	Test case description	Test steps			Test case status (P/F)
			Steps	Expected	Actual	
1.	Check for complex roots	For a quadratic equation, complex roots exist if $d = b^2 - 4ac$ is < 0.	Compute d = (b * b) – (4 * a * c) check if (d < 0)	If (d < 0) the message "Roots are complex" will be displayed. And then roots r1 and r2 will be printed.	For equation : $x^2 + x + 2 = 0$ $r1 = \frac{-1+\sqrt{-7}}{2}$ $r2 = \frac{-1-\sqrt{-7}}{2}$ Then solve (x – r1) * (x – r2)	Pass
2.	Check for single real root	For a quadratic equation, single real root exists if $d = b^2 - 4ac$ is equal to 0.	Compute d = (b * b) – (4 * a * c) check if (d = = 0)	If (d is equal to 0) then the message "Single real root" will be displayed and only one root r1 will be printed.	For equation : $9x^2 - 24x + 16 = 0$ r1 = 4/3 Then solve $(x - r1)^2 = 0$ we will get $9x^2 - 24x + 16 = 0$	Pass
3.	Check for two real rocts	For a quadratic equation, two real roots exist if $d = b^2 - 4ac$	Compute d = (b * b) – (4 * a * c) check if (d > 0)	If (d > 0) then the message "Two real roots" will be	For equation : $x^2 + 2x - 2 = 0$ $r1 = -1 + \sqrt{3}$ $r2 = -1 - \sqrt{3}$	Pass

	is greater than 0.	printed.	Then solve $(x - r1) * (x - r2)$ and we will get $x^2 + 2x - 2$

Q.16 What are the steps for top-down integration ? AU : Dec.-07

Ans. : Following are the steps in top-down integration.

1. The main control module is used as a test driver and stubs are substituted for all the modules directly subordinate to the main control module.
2. Subordinate stubs are replaced one at a time with actual modules using either depth first or breadth search method.
3. Tests are conducted as each module is integrated.
4. On completion of each set of tests, another stub is replaced with the real module.
5. Regression testing is conducted to prevent the introduction of new errors.

Q.17 Mention any two characteristics of software testing. AU : May-08

OR Write down generic characteristics of software testing. AU : May-13

Ans. : Following are the characteristics of software testing :

1. Testing should be conducted by the developer as well as by some independent group.
2. The testing must begin at the component level and must work outwards towards the integration testing.
3. Different testing must be appropriate at appropriate times.

Q.18 Which is called as a glass box testing ? What is the objective of this ? (Refer section 4.4) AU : Dec.-07

Q.19 List out the data structure errors identified during the unit testing. AU : Dec.-07

Ans. : Various data structure errors that can be identified during the unit testing are -

1. Incorrect arithmetic precedence.
2. Mixed mode operations.
3. Precision inaccuracy.
4. Comparison of different data types.

Q.20 Calculate the cyclomatic complexity for the following program. Explain your approach.

```
int temp
(a>b) temp a
else temp b
```

```
if (c>temp)
temp=c
return temp
```

AU : May-08

Ans. : The flow graph for given program is -

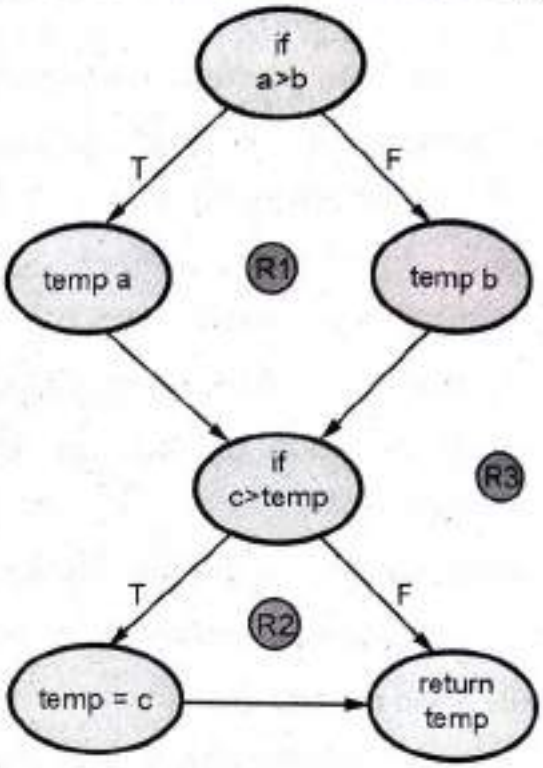


Fig. 4.14.2

Cyclomatic complexity

$= E - N + 2$

$= 7 - 6 + 2 = 3$

Cyclomatic complexity

$= P + 1$

$= 2 + 1 = 3$

Cyclomatic complexity

= Regions encountered

= 3

Hence cyclomatic complexity of given program is **3.**

Q.21 What is a critical module and why should we identify it ? AU : Dec.-08

Ans. : The module which possess following characteristics is called critical module.

1. The critical module addresses several software requirements.
2. This module has definite performance requirements.
3. It has high level of control.
4. This module is complex and error prone.

Because of above mentioned characteristics, it is necessary to identify and test such module in order to make the system errorfree and as per user requirements.

Q.22 What is meant by smoke testing ? *(Refer section 4.7.4)* AU : Dec.-08

Q.23 What is the difference between alpha testing and beta testing ? AU : May-09

Sr. No.	Alpha testing	Beta testing
1.	This testing is done by a developer or by a customer under the supervision of developer in company's premises.	This testing is done by the customer without any interference of developer and is done at customer's place.

2.	Sometime full product is not tested using alpha testing and only core functionalities are tested.	The complete product is tested under this testing. Such product is usually given as free trial version.

Q.24 Will the exhaustive testing guarantee that the program is 100 % correct ? AU : May-09, 16

Ans. : No the exhaustive testing will not guarantee that the program is 100 % correct. This is because there are several parameters that affect the correctness of the program. For instance -

User acceptance - It is not guaranteed that the program is strictly as per users' expectation and will work as per users' design.

Performance testing - It is necessary to test whether the program performs to a level that was satisfactory. Similarly the performance should not get degraded if volume load gets increased.

Integration testing - This testing needs to be done to check whether the program works properly if it gets integrated with some other software component.

Besides these testing there are many other types of testing that needs to be carried out on the program, and there are chances of a program getting failed in one or more testing.

Q.25 What is meant by integration testing ? *(Refer section 4.7)* AU : May-09

Q.26 State any four software testing principles. *(Refer section 4.1.2)* AU : Dec.-09

Q.27 What is an unit ? AU : May-10

Ans. : Unit is a smallest testable part in the application. In procedural programming, unit may be an individual program, function or procedure. While in object oriented programming, the smallest unit is a method, which may belong to a base class, abstract class or derived class.Ideally independent cases can be written for each unit(function) of a source program.

Q.28 In unit testing of a module, it is found for a set of test data, at maximum 90 % of the code alone were tested with the probability of success 0.9. What is the reliability of the module ? AU : Dec.-11

Ans. : The reliability of the module is at the most 0.81. For the given set of test data the 90 % of the code is tested and probability of success is given as 0.9 i.e. 90 %.

Hence Reliability = (90/100)*0.9 = 0.81

Q.29 With a simple example establish that 100 % testing is impossible.

Ans. : It is not possible to test all inputs or all permutations and combinations of all inputs. It is not even possible to test all paths of even a moderate system. For example - Suppose we have to test a function that handles verification of user password. Each user on each

computer must have a password. Each password might be atleast 6 to 8 characters long. It might be in capital letters, alphanumeric or in lower case. Then it is just impossible to consider various permutations and combinations for password verification.

Q.30 Distinguish between stress and load testing. AU : May-12

Ans. : Load testing is conducted to measure system performance based on volume of users. On the other hand the stress testing is conducted for measuring the breakpoint of a system when extreme load is applied. In load testing the expected load is applied to measure the performance of the system whereas in stress testing the extreme load is applied.

Q.31 What is a big-bang approach ? AU : Dec.-12

Ans. : Big-bang approach is used in non incremental integration testing. In this approach of integration testing, all the components are combined in advance and then entire program is tested as a whole.

Q.32 How are the software testing results related to the reliability of the software ?

AU : Dec.-12

Ans. : During the software testing the program is executed with the intention of finding as much errors as possible. The test cases are designed that are intended to discover yet-undiscovered errors. Thus after testing, majority of serious errors are removed from the system and it is turned into the model that satisfied user requirements.

Q.33 What are the levels at which the testing is done ? AU : Dec.-13

Ans. : Testing can be done in the two levels as given below -

1. **Component level testing** : In the component level testing the individual component is tested.
2. **System level testing** : In system testing, the testing of group of components integrated to create a system or subsystem is done.

Q.34 What are the classes of loops that can be tested ? AU : May-14

Ans. : Various classes of loop testing are -

1. Simple loop
2. Nested loop
3. Concatenated loop
4. Unstructured loop.

Q.35 What is validation testing ? AU : Dec.-14

Ans. : Validation testing is a testing which is meant to validate the requirements as defined in the Software Requirements Specification (SRS). It ensures that the application as developed matches with the requirements defined and is fit for intended use.

Q.36 What is the need for regression testing ? AU : May-15

Ans. : Regression testing is required to test the defects that get propagated from one module to another when changes are made to existing program. Thus regression testing is used to reduce the side effects of changes.

Q.37 How will you test simple loop ? AU : Dec.-15

Ans. : The simple tests can be performed for n number of classes.

i) If n = 0 then skip the loop completely.

ii) If n = 1 then one pass through the loop is tested.

iii) If n = m then testing should be done for m passes where m < n.

iv) Perform the testing when number of passes are n – 1, n, n + 1.

Q.38 Why does software fail after it has passed from acceptance testing ? AU : May-16

Ans. : During acceptance testing, the random input is used for testing. This may lead to the situation that some input values that may cause failure go unhandled. The practical problem with acceptance testing is that it is time consuming. Hence in order to keep testing cost low, there are restricted number of test cases.

Hence there are high chances of software failure even after passing the acceptance testing.

Q.39 What is validation testing ? AU : Dec.-14

Ans. : Validation testing is a testing which is meant to validate the requirements as defined in the Software Requirements Specification(SRS). It ensures that the application as developed matches with the requirements defined and is fit for intended use.

Q.40 What is the need for regression testing ? AU : May-15

Ans. : Regression testing is required to test the defects that get propagated from one module to another when changes are made to existing program. Thus regression testing is used to reduce the side effects of changes.

Q.41 What is smoke testing ? AU : May-17

Ans. : The smoke testing is a kind of integration testing technique used for time critical projects where in the projects need to be assessed on frequent basis.

Q.42 Mention the purpose of stub and driver used for testing. AU : Dec.-17

Ans : Driver : The "driver" is a program that accepts the test data and prints the relevant results.

Stub : The "stub" is a subprogram that uses the module interfaces and performs the minimal data manipulation if required.

Q.43 What are the testing principles the software engineer must apply while performing the software testing ? AU : May-18

Ans. :

1) All tests must be traceable to customer requirements.
2) Tests should be planned long before testing begins.
3) Testing should begin in small and progress towards testing in large.
4) Exhaustive testing is not possible.

5) Testing should be done independent third party.

Q.44 Identify the type of maintenance for each of the following :

a) Correcting the software faults

b) Adapting the change in environment. AU : May-18

Ans. : a) Corrective maintenance **b)** Adaptive maintenance

Q.45 List the levels of testing. AU : May-19

Ans. : 1) Unit testing 2) Integration testing 3) Validation testing 4) System testing.

Q.46 What is test case ? AU : Dec.-19

Ans. : Test case is a document or a specification of inputs, execution conditions, testing procedure and expected results while executing every single test.

Q.47 Outline the need for system testing. AU : Dec.-19

Ans. : The system testing is needed for testing all the system elements so that the system as a whole can be tested.

Q.48 How black box testing is different from white box testing ? *(Refer section 4.4)* AU : Dec.-20

Q.49 Define regression testing. AU : May-22, Dec.-22

Ans. : Regression testing is used to check for defects propagated to other modules by changes made to an existing program. This testing is used to reduce the side effects of the changes.

❑❑❑

UNIT V

5 Project Management

Syllabus

Software Project Management- Software Configuration Management - Project Scheduling- DevOps : Motivation-Cloud as a platform-Operations - Deployment Pipeline : Overall Architecture Building and Testing-Deployment - Tools - Case Study.

Contents

5.1 Software Project Management

5.2 Management Spectrum

5.3 People

5.4 Product

5.5 Process

5.6 Project

5.7 The W5HH Principle

5.8 Project Estimation **May-05,06,07,09,16,17,18,19,22,**
..... **Dec.-07,10,14,15,16,19,** ..... Marks 16

5.9 Make Buy Decision ..... **May-16,** ..... Marks 8

5.10 COCOMO I Model **May-07,08,14,15,17,22,**
..... **Dec.-05,13,22,** ..... Marks 16

5.11 COCOMO II Model **Dec.-07,10,11,14,15,16,19,**
..... **May-07,16,17,18,** ..... Marks 16

5.12 Software Configuration Management

5.13 Project Scheduling **May-05,15,16,17,22,**
..... **Dec.-06,07,15,16,** ..... Marks 16

5.14 DevOps

5.15 Cloud as a Platform

5.16 Two Marks Questions with Answers

5.1 Software Project Management

- Management is an essential activity for the computer based systems and product. The term project management involves various activities such as planning, monitoring, control of people, process and various events that occur during the development of software.
- Building the software system is a complex activity and many people get involved in this activity for relatively long time. Hence, it is necessary to manage the project. The project management is carried out with the help of **4 P's i.e. people, product, process and project.**
- Hence, we will start our discussion by focusing on these elements.

5.2 Management Spectrum

Effective software project management focuses four P's i.e. **people, product, process and project**. The successful project management is done with the help of these four factors where the order of these elements is not arbitrary. Project manager has to motivate the **communication between stakeholders**. He should also prepare a **project plan** for the success of the product.

5.2.1 The People

- **People factor** is an important issue in software industry. There is a strong need for motivated and highly skilled people for developing the software product. The Software Engineering Institute (SEI) has developed the People Management Capability Maturity Model (PM-CMM).
- By using PM-CMM model software organizations become capable for undertaking complex applications which ultimately attracts or motivates the talented people.
- Following are some key practice areas for software people -
  - Recruitment
  - Selection
  - Performance management
  - Training compensation
  - Career development
  - Organization and work design
  - Culture development.

5.2.2 The Product

- Before planning the project three important tasks need to be done -
  1) Product objectives and scope must be established.
  2) Alternative solutions should be considered.
  3) and management constraints must be identified.
- The software developer and customer must communicate with each other in order to define the objectives and scope of the product.
- This is done as the first step in requirement gathering and analysis.
- The scope of the project identifies primary data, functions and behaviour of the product.
- After establishing the objectives and scope of the product the alternative solutions are considered.
- Finally, the constraints imposed by - delivery deadline or budgetary restrictions, personal availability can be identified.

5.2.3 The Process

- The software process provides the framework from which the software development plan can be established.
- There are various framework activities that needs to be carried out during the software development process. These activities can be of varying size and complexities.
- Different task sets-tasks, milestones, work products and quality assurance points enable framework activities to adapt the software requirements and certain characteristics of software project.
- Finally, umbrella activities such as Software Quality Assurance (SQA) and Software Configuration Management (SCM) are conducted. These umbrella activities depend upon the framework activities.

5.2.4 The Project

To manage the complexity of the project, we conduct planned and controlled software projects. In order to successfully manage a software project, we must understand what can go wrong and how to do it right. There are some indicators that imply poor management of the projects.

Review Question

1. *Explain the role of people, product and process in project management.*

5.3 People

People is the most important element for the success of software project. People participate in the project with different roles and responsibilities. Let us discuss these -

5.3.1 Stakeholder

Stakeholders mean persons who will be affected by the system. Following are the categories of the stakeholders -

1. Senior Manager

- These are the persons who define the business issues and the decisions taken by have significant influence on the success of the project.

2. Project Manager

- The project manager performs various tasks such as planning, motivating, organizing and controlling the software practitioners who are involved in software building.

3. Practitioners

- The practitioners are the entities having sufficient technical skills required for engineering the product or application.

4. Customers

- These stakeholders specify the requirements of the project and are interested only in outcome of the project.

5. End-users

- The end-users interact with the software product once get released.
- The team should be organized in such a way that skills and abilities of each person can be utilized at the most and this job is done by Team leader.

5.3.2 Team Leaders

Project management is an activity influenced by people. Following are the qualities that the team leader must posses -

- **Motivation**

  This ability is necessary to encourage technical people to do their work with the fullest of their ability.

- **Problem Solving**

  During project management, the project manager must diagnose the technical and organizational issues in such a manner that systematic and effective solution can be obtained. He/She must take lessons from the previous projects and must remain flexible to change the direction of the if initial problem solving methods get failed.

- **Organization**

  This is an ability to arrange the existing processes so that intial concept can be translated into the final working product.

- **Innovation**

  Innovation or ideas mean encouraging people to create and to feel creative.

- **Managerial Identity**

  The effective project manager must take the charge of complete project and must have full control over the project development process.

- **Achievement**

  The productivity of the project team can be optimized by awarding the team members for their good work and by not punishing them on the controlled risk.

- **Influence and Team Building**

  An effective project manager should be able to "read" the people in his team, he should understand their needs and must remain in controlled in stressful situations.

5.3.3 Software Team

- The **best** team structure depends on three things - i) Management style of organization ii) Number of people involved in the project and their skill levels and iii) Overall problem difficulty.

- **Mantei** described **seven factors** that should be considered for **planning** the **structure of software engineering teams -**
  - The **difficulty** of the problem to be solved.
  - The **size** of the project (may be considered in terms of LOC or in Function points).
  - The **time** that the team will stay together.
  - The required **quality and reliability** of the system to be built.
  - The **rigidity or flexibility** of the delivery date.
  - The degree of **communication** required for the project.

Constantine suggested four **organizational paradigms** for software engineering team -

- **Closed Paradigm**

  It represents the **traditional hierarchy** of team. This team work well for producing software based on **past efforts** but it **fails** to work for **innovative ideas**.

- **Random Paradigm**

  This is a **loosely structured** and depends upon individual initiative of team members. This performs well when innovation or technological breakthrough is required but performs poorly when the orderly performance is required.

- **Open Paradigm**

  The open paradigm attempts to structure a team in such a manner that some controls are achieved using **closed paradigm** but innovation or technological break through is achieved using the **random paradigm**.

- **Synchronous Paradigm**

  It structures the team using natural compartmentalization of the problem. This team work on the piece of problem by establishing proper communication among them.

5.3.4 Agile Team

- Agile philosophy encourages -
  - Customer satisfaction.
  - Early incremental delivery of the software.
  - Small but highly motivated project team.
  - Informal methods.
  - Development simplicity.

Hence a small motivated team is called **agile team.**

- Agile philosophy stresses on **individual competency** coupled with **group collaboration** as critical success factor for the team.
- Agile teams are **self-organizing** teams. There is no **fixed Constantine team paradigm** that can be applied to the agile, instead of that the agile team can use elements of random, open, closed and synchronous paradigms.
- Many agile **models automate** the project management and technical decisions required for project accomplishment.
- Agile team is allowed to select its **own approach** for software development. The only condition is that the business requirements and organizational standards must get satisfied during software development.
- As the project proceeds the **individual competency** might be **improved** in such a way that it will be beneficial to the project. To improve the individual competency the agile team conducts the **daily meeting**, for co-ordinating and synchronizing the work. The **information** obtained during this meeting is used to adapt the useful **approach** for software development.

5.3.5 Co-ordination and Communication Issues

Following are some characteristics of modern software -

- **Scale**

  The scale of development efforts may be very large which leads to complexity and confusion in co-ordinating the team members.

- **Uncertainty**

  Continuous changes occur in the project, due to which uncertainty is common in modern software.

- **Interoperability**

  Interoperability is the key characteristic of many systems. New software must be able to communicate with existing software.

To deal with above mentioned characteristics software engineering team must establish effective methods for co-ordinating people. This co-ordination is possible only by means of formal and informal communication among the team members.

Review Questions

1. *What are the categories of stakeholders ? What are the characteristics of effective project manager ?*
2. *List the four P's of software project management spectrum. Explain how "the people" factor contributes towards the success of software project.*
3. *What are the categories of stakeholders ? What are the characteristics of effective project manager ?*

5.4 Product

A software project manager has to examine the product in order to obtain the quantitative estimates and organizational plan. By examining the product the scope of the product can be decided.

5.4.1 Software Scope

The first step in software project management is to determine the scope of the software project. Following questions need to be answered for determining the scope of the project -

1. Context

a) How does the software built fit into the larger system and business context ?

b) What are the constraints imposed on the context of the project ?

2. Information Objectives

a) What are the visible data objects that get produced as an output of software ?

b) What are the data objects that are required for the input of the software ?

3. Function and performance

a) For transforming the input data to an output what are those required functions ?

b) What are special performance characteristics ?

The software scope must be **unambiguous and understandable** at the management and technical levels. A statement of scope must be **bounded**. **Constraints and limitations** are noted and **mitigating factors** are described.

5.4.2 Problem Decomposition

- Problem decomposition means partitioning or elaborating the problems.

- During the project scope no attempt is made to decompose the problem fully. The decomposition that is normally applied in based on two major areas -
  1. **Functionalities** that must be delivered.
  2. The **process** that will be used to deliver it.
- Normally, "**Divide and Conquer**" strategy is applied to partition the product into smaller pieces which can be managed easily as project planning begins.
- The software functions described in the scope are evaluated and refined to provide more details before the beginning of the estimation.
- As both cost and schedule are functionally oriented, some degree of decomposition is often useful.
- As the statement of scope evolves the first level of partitioning naturally occurs.

5.5 Process

- There are some framework activities that characterize the software processes. Such activities are applied to all software projects.
- The Project Manager must decide which Process Model is most appropriate for -
  1. The Customers and the People who are involved in project.
  2. The characteristics of Product.
  3. The Project Environment in which the Software works.
- When a Process Model is selected, the Software Team then defines a Preliminary **Project Plan** based on Process Framework activities. After establishing the preliminary Project Plan, Process Decomposition begins.

5.5.1 Melding the Product and the Process

- At the beginning of planning process there is the melding of Product and the Process. Each function to be engineered by the software team must go through the set of **Framework Activities** that has been defined for a Software organization.
- Assume that following are some **framework activities** -
  - Communication
  - Planning
  - Modeling
  - Construction and Release
  - Deployment
  - Customer Evaluation

- When team members work on the product functions they will apply these farmework activities to it. A matrix as shown in Fig. 5.5.1 is created.

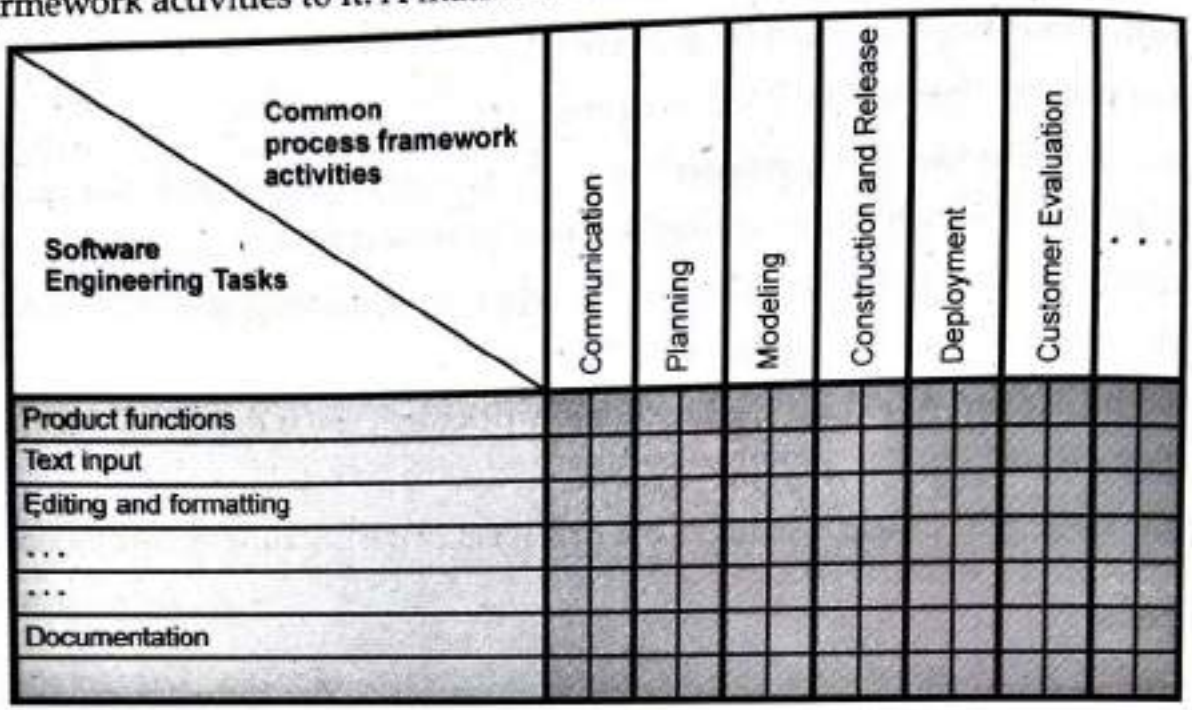


Fig. 5.5.1 Matrix for framework activities

- On the left hand side the major product functions are described and framework activities are listed on the top. Then the software engineering work tasks would be entered in the row.
- Project Manager estimates the resource requirements for each matrix cell with Start and End Dates for each Task and Work Products to be produced (delivered) for each task.

5.5.2 Process Decomposition

- The software development team has a flexibility for choosing the software process model which is best suitable for the project. After decision of process model the software engineering tasks are decided.
- When a project manager raises a question "How do we accomplish the framework activities?" then project decomposition commences. For example - a small project might require following work tasks for performing the communication activity -
  1. A list of **clarification issues** is to be prepared.
  2. For addressing the clarification issues conduct **meetings** with customers.
  3. The developer and customer together must prepare **statement of scope**.
  4. **Review** the statement of scope.
  5. Perform **modifications** in the statement of scope if required.

- Similarly for the large projects the work task list can be prepared which may contain some additional tasks.

5.6 Project

- For a successful software project, it is necessary to understand the mistakes in the project and how to correct them. John Reel defined ten symptoms to indicate why the software projects fail -
  1. Software developers do not understand the customer's need.
  2. The scope of the project is not defined properly.
  3. Change management is done poorly.
  4. Business needs change very often.
  5. The technological changes are quite often.
  6. Users are not co-operative.
  7. Unrealistic deadlines are set.
  8. Projects do not get sponsored or Sponsorship gets lost.
  9. Lack of skilled people in the project.
  10. Project managers do not adopt the best practices for the projects.
- John Reel suggested following five points to overcome the problems in the software project -
  1. **Start on right foot** - The software developer must understand the problem, managers should set the realistic goals and customers should keep the realistic expectations from the project. The right team must be chosen to complete the project. Such a team must be given autonomy and authority. The technology needed by such team should be provided. All these factors avoid lot of problems of the software project.
  2. **Maintain momentum** - Many projects are very good initially but later on their quality decreases. Therefore project managers should provide incentives for absolute minimum income. Quality must be insisted at every stage of development.
  3. **Track progress** - Regular progress of the project must be tracked. The work products at every stages of development must be analysed.
  4. **Make smart decisions** - Simple decisions must be made by software project managers as well as team members. Whenever possible make use of the off-the shelf components. Try to identify the obvious risks and allocate more time than you think for solving complex risks.

5. **Conduct analysis** - After completion of every project try to find out what went wrong and how to correct them. Evaluate the plans and actual schedule. Get the feedback about the project from the team members and customers.

5.7 The W5HH Principle

Bary Bohem suggested an approach for addressing - project objectives, deciding milestones and schedules, roles and responsibilities, project management, technical approaches and required resources by WWWWWHH principle. The W5HH principle is nothing but the series of questions in the form of why, what, when, who, how and so on. Answers to these questions lead to definition of key project characteristics. The answers of these questions are also useful in building the resultant project plan. Following are those W5HH questions -

- **Why is the system being developed ?**

  Answer to this question assesses the validity of business reasons for the software work. It also justifies the expenditure of people, time and money.

- **What will be done ?**

  Answer to this question will help the software team member to identify the project tasks and milestones.

- **When will be done ?**

  The answer to this question will help to prepare the project schedule with identified project tasks and milestones.

- **Who is responsible for a function ?**

  By answering this question, the roles and responsibilities required to develop the system can be defined.

- **Where are they organisationally located ?**

  All the roles can not be defined within the software team itself. There are customers, users and other stakeholders holding some responsibilities.

- **How to do the job technically with proper management ?**

  Answer to this question will help to establish the technical strategy about the project.

- **How much of each resource required ?**

  This answer will be useful for deriving the project estimate or cost of the project.

Review Question

1. *Explain W5HHH principle.*

5.8 Project Estimation

AU : May-05,06,07,09,16,17,18,19,22, Dec.-07,10,14,15,16,19, Marks 16

Software project estimation is a form of problem solving. Many times the problem to be solved is too complex in software engineering. Hence for solving such problems, we decompose the given problem into a set of smaller problems.

The decomposition can be done using two approached decomposition of problem or decomposition of process. Estimation uses one or both forms of decomposition (partitioning).

5.8.1 Software Sizing

- Following are certain issues based on which accuracy of software project estimate is predicated -
  1. The degree to which planner has properly estimated the size of the product to be built.
  2. The ability to translate the size estimate into human-effort, calendar time and money.
  3. The degree to which the project plan reflects the abilities of software team.
  4. The stability of product requirements and the environment that supports the software engineering effort.
- Sizing represents the project planner's first major challenge. In the context of project planning, size refers to a quantifiable outcome of the software project.
- The sizing can be estimated using two approaches - **a direct approach** in which lines of code is considered and **an indirect approach** in which computation of function point is done.
- Putnam and Myers suggested four different approaches for sizing the problem -

1. **Fuzzy logic sizing**

In this approach planner must identify -

- The **type** of application.
- Establish its **magnitude on a qualitative scale** and then refine the magnitude within the original range.

- Planner should also have **access to historical database** of the project so that estimates can be composed to actual experience.

2. Function point sizing

Planner develops estimates of the information domain.

3. Standard component sizing

There are various **standard components** used in software. These components are subsystems, modules, screens, reports, interactive, programs, batch program, files, LOC and Object-level instruction.

The project planner estimates the number of time these standard components are used. He then uses historical project data to determine the delivered size per standard component.

4. Change sizing

This approach is used when existing software has to be modified as per the requirement of the project. The size of the software is then estimated by the number and type of reuse, addition of code, change made in the code, deletion of code.

- The result of each sizing approaches must be combined statistically to create **three-point estimate** which is also known as **expected-value estimate.**

5.8.2 Problem based Estimation

- The problem based estimation is conducted using LOC based estimation, FP based estimation, process based estimation and use cased based estimation.
- LOC and FP based data are used in two ways during software estimation -
  1. These are useful to estimate the **size** each element of software.
  2. The baseline metrics are collected from past project and LOC and FP data is used in conjunction with estimation variable to develop cost and effort values for the project. (LOC) and (FP) estimation are different estimation techniques. Yet, both have number of characteristics in common.
- With a bounded statement of software scope a project planning process begins and by using the statement of scope the software problem is decomposed into the functions that can be estimated individually.
- (LOC) or (FP) is then estimated for each function.
- **Baseline productively metrics** are then applied to the appropriate estimation variable and cost or effort for the function is derived.

- **Function estimates** are combined to obtain an overall estimate for the entire project.
- Using historical data the project planner **expected value** by considering following variables -
  1. Optimistic
  2. Most likely
  3. Pessimistic

For example, following formula

$$S = [S_{opt} + 4 * S_m + S_{pess}]/6$$

considers for "most likely" estimate where S is the estimation size variable, represents the optimistic estimate, represents the most likely estimate and represents the pessimistic estimate values.

5.8.3 LOC based Estimation

- Size oriented measure is derived by considering the size of software that has been produced.
- The organization builds a simple record of size measure for the software projects. It is built on past experiences of organizations.
- It is a direct measure of software

Project	LOC	Effort	Cost($)	Doc.(pgs.)	Errors	Defects	People
ABC	10,000	20	170	400	100	12	4
PQR	20,000	60	300	1000	129	32	6
XYZ	35,000	65	522	1290	280	87	7
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 5.8.1 Size measure

- A simple set of size measure that can be developed is as given below :
  - Size = Kilo Lines of Code (KLOC)
  - Effort = Person/month
  - Productivity = KLOC/person-month

- Quality = Number of faults/KLOC
- Cost = $/KLOC
- Documentation = Pages of documentation/KLOC

- The size measure is based on the lines of code computation. The lines of code is defined as one line of text in a source file.
- While counting the lines of code the simplest standard is :
  - Don't count blank lines.
  - Don't count comments.
  - Count everything else.
- The size oriented measure is not universally accepted method.

Advantages

1. Artifact of software development which is easily counted.
2. Many existing methods use LOC as a key input.
3. A large body of literature and data based on LOC already exists.

Disadvantages

1. This measure is dependent upon the programming language.
2. This method is well designed but shorter program may get suffered.
3. It does not accommodate non procedural languages.
4. In early stage of development it is difficult to estimate LOC.

5.8.4 Example of LOC based Estimation

Example

Consider an ABC project with some important modules such as,

1. User interface and control facilities
2. 2D graphics analysis
3. 3D graphics analysis
4. Database management
5. Computer graphics display facility
6. Peripheral control function
7. Design analysis models

Estimate the project in based on LOC.

Solution :

For estimating the given application we consider each module as separate function and corresponding lines of code can be estimated in the following table as,

Function	Estimated LOC
User Interface and Control Facilities(UICF)	2500
2D graphics analysis(2DGA)	5600
3D Geometric Analysis function(3DGA)	6450
Database Management(DBM)	3100
Computer Graphics Display Facility(CGDF)	4740
Peripheral Control Function(PCF)	2250
Design Analysis Modules (DAM)	7980
Total estimation in LOC	**32620**

- Expected LOC for 3D Geometric analysis function based on three point estimation is -
  - Optimistic estimation 4700
  - Most likely estimation 6000
  - Pessimistic estimation 10000

$$S = [S_{opt} + (4 * S_m) + S_{pess}]/6$$

Expected value = $[4700 + (4 * 6000) + 10000] /6 \rightarrow 6450$

- A review of historical data indicates -
  1. Average productivity is 500 LOC per month
  2. Average labor cost is $6000 per month

Then cost for lines of code can be estimated as

cost/LOC = (6000/500) = $12

By considering total estimated LOC as 32620

- Total estimated project cost = (32620*12) = $391440
- Total estimated project effort = (32620/500) = 65 Person-months

Example 5.8.1 *Consider 7 functions with their estimated lines of code given below*

Function	LOC
Funct1	2340
Funct2	5380
Funct3	6800
Funct4	3350
Funct5	4950
Funct6	2140
Funct7	8400

Average productivity based on historical data is 620 LOC/pm and Labour rate is ₹ 8000 per month. Find the total estimated project cost and effort. **AU : Dec.-16, Marks 8**

Solution : We will first compute total estimation in LOC.

Function	LOC
Funct1	2340
Funct2	5380
Funct3	6800
Funct4	3350
Funct5	4950
Funct6	2140
Funct7	8400
Total estimation (LOC)	33360

- A review of historic data indicates

1) Average productivity is 620 LOC/per month

2) Average labour cost is ₹ 8000 per month

Then cost for lines of code can be estimated as

Cost/LOC = 8000/620 = ₹ 13 approx

By considering total estimated LOC as 33360

- Total estimated project cost = (33360 *13)

= ₹ 433680

- Total estimated project effort = (33360/620)
  = 54 person-months

Example 5.8.2 *If Team A found 342 errors prior to release of software and Team B found 182 errors. What additional measures and metrics needed to find out if teams have removed the errors effectively ? Explain.* **AU : May-18, Marks 4**

Solution : The additional measures and metrics needed to find out if team have removed errors effectively or not are -

1. Errors per KLOC (thousand lines of code)
2. Defects per KLOC
3. $ per LOC
4. Pages of documentation per KLOC

The LOC based metric can be chosen to understand the errors.

5.8.5 Function Oriented Metrics

- The function point model is based on functionality of the delivered application.
- These are generally independent of the programming language used.
- This method is developed by Albrecht in 1979 for IBM.
- Function points are derived using :
  1. Countable measures of the software requirements domain.
  2. Assessments of the software complexity.

How to calculate function point ?

- The data for following information domain characteristics are collected :
  1. Number of user inputs - Each user input which provides distinct application data to the software is counted.
  2. Number of user outputs - Each user output that provides application data to the user is counted, e.g. screens, reports, error messages.
  3. Number of user inquiries - An on-line input that results in the generation of some immediate software response in the form of an output.
  4. Number of files - Each logical master file, i.e. a logical grouping of data that may be part of a database or a separate file.
  5. Number of external interfaces - All machine-readable interfaces that are used to transmit information to another system are counted.
- The organization needs to develop criteria which determine whether a particular entry is simple, average or complex.

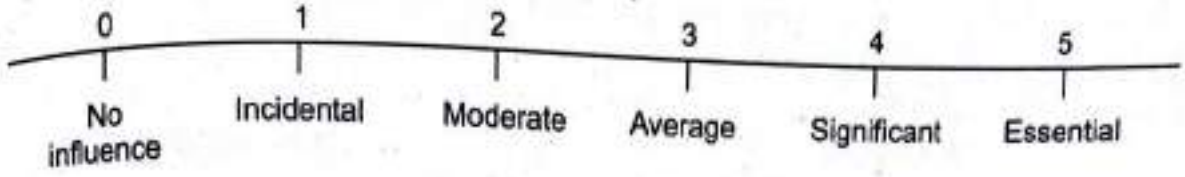
- The weighting factors should be determined by observations or by experiments.

Domain characteristics	Count		Weighting factor			Count
			Simple	Average	Complex	
Number of user input		X	3	4	6	
Number of user output		X	4	5	7	
Number of user inquiries		X	3	4	6	
Number of files		X	7	10	15	
Number of external interfaces		X	5	7	10	
Count Total						

- The count table can be computed with the help of above given table.
- Now the software complexity can be computed by answering following questions. These are **complexity adjustment values.**

1. Does the system need reliable backup and recovery ?
2. Are data communications required ?
3. Are there distributed processing functions ?
4. Is performance of the system critical ?
5. Will the system run in an existing, heavily utilized operational environment ?
6. Does the system require on-line data entry ?
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations ?
8. Are the master files updated on-line ?
9. Are the inputs, outputs, files or inquiries complex ?
10. Is the internal processing complex ?
11. Is the code which is designed being reusable ?
12. Are conversion and installation included in the design ?
13. Is the system designed for multiple installations in different organizations ?
14. Is the application designed to facilitate change and ease of use by the user ?

- Rate each of the above factors according to the following scale :
- **Function Points (FP)** = Count total $\times$ $(0.65 + (0.01 \times \text{Sum}(F_i))$



- Once the functional point is calculated then we can compute various measures as follows
  - Productivity = FP/person-month
  - Quality = Number of faults/FP
  - Cost = $/FP
  - Documentation = Pages of documentation/FP.

Advantages

1. This method is independent of programming languages.
2. It is based on the data which can be obtained in early stage of project.

Disadvantages

1. This method is more suitable for business systems and can be developed for that domain.
2. Many aspects of this method are not validated.
3. The functional point has no significant meaning. It is just a numerical value.

5.8.6 Example of FP based Estimation

FP focuses on information domain values rather than software functions. Thus we create a function point calculation table for ABC project.

INFO DOMAIN VALUE	Opt.	most likely	pessimistic	esti. value	weight factor	FP
NO.OF INPUTS	25	28	32	28.1	4	112
NO. OF OUTPUTS	14	17	20	17	5	85
NO. OF INQURIES	17	23	30	23.1	5	116
NO. OF FILES	5	5	7	5.33	10	53
NO. OF EXTERNAL INTERFACES	2	2	3	2	7	15
COUNT TOTAL						381

- For this example we assume average complexity weighting factor.

- Each of the complexity weighting factor is estimated and the complexity adjustment factor is computed using the complexity factor table below.

(Based on the 14 questions)

Sr. No.	FACTOR	VALUE (Fi).
1.	Back-up and recovery ?	4
2.	Data communication ?	2
3.	Distributed processing ?	0
4.	Performance critical ?	4
5.	Existing operational environment ?	3
6.	On-line data entry ?	4
7.	Input transactions over multiple screens ?	5
8.	Online updates ?	3
9.	Information domain values complex ?	5
10	Internal processing complex ?	5
11.	Code designed for reuse ?	4
12.	Conversion / installation in design ?	3
13.	Multiple installations ?	5
14.	Application designed for change ?	5

$\sum(F_i) \rightarrow$ **52**

The estimated number of adjusted FP is derived using the following formula : -

FP ESTIMATED = (FP COUNT TOTAL * [COMPLEXITY ADJUSTMENT FACTOR]

FP ESTIMATED = COUNT TOTAL* $[0.65 + (0.01 * ? \sum (F_i))]$

Complexity adjustment factor = [0.65 + (0.01 * 52)] = 1.17

- **FP ESTIMATED = (381 * 1.17) = 446** (Function point count adjusted with complexity adjustment factor)
- A review of historical data indicates -
  1. Average productivity is 6.5 FP/Person month
  2. Average labor cost is $6000 per month
- Calculations for cost per function point, total estimated project cost and total effort
  1. The cost per function point = (6000 / 6.5) = $923
  2. Total estimated project cost = (446 * 923) = $411658
  3. Total estimated effort = (446 / 6.5) = 69 Person-month.

Example 5.8.3 *Study of requirement specification for ABC project has produced following results : Need for 7 inputs, 10 outputs, 6 inquiries, 17 files and 4 external interfaces. Input and external interface function point attributes are of average complexity and all other function points attributes are of low complexity.*
Determine adjusted function points assuming complexity adjustment value is 32.

Solution : Given that :

7 inputs

10 Outputs

6 inquiries

17 files

4 external interfaces

Average complexity for inputs and external interfaces. Low complexity for remaining parameters.

Adjusted function point value $\sum(F_i) = 32$.

Let us calculate count total value.

Measurement parameters	Count		Weighting factor			
		×	Simple	Average	Complex	
Number of user inputs	7	×		4		28
Number of user outputs	10	×	4			40
Number of user inquiries	6	×	3			18
Number of files	17	×	7			119
Number of external interfaces	4	×		7		28
Count total						233

$$\text{Function point} = \text{Count total} \times [0.65 + 0.01 \times \Sigma(F_i)]$$

$$= 233 \times [0.65 + 0.01 \times 32]$$

$$= 233 \times [0.65 + 0.32]$$

$$= 233 \times 0.97$$

$$FP = 226.01$$

Hence adjusted function point is **226.01**

Example 5.8.4 *An application has the following : 10 low external inputs, 8 high external outputs, 13 low internal logical files, 17 high external interface files, 11 average external inquires and complexity adjustment factor of 1.10. What are unadjusted and adjusted function point counts ?* **AU : May-16, Marks 4**

Solution : The unadjusted function points are calculated using predefined weights for each function type which is as given below.

Functional Units	Weighting Factors		
	Low	Average	High
External Input	3	4	6
External Output	4	5	7
External Inquiries	3	4	6
Internal Logical Files	7	10	15
External Interface File	5	7	10

The Unadjusted Function Point (UFP)

$$= \Sigma \text{ Functional units} \times \text{Weighting factor.}$$

$$= ((10 \times 3) + (8 \times 7) + (13 \times 7) + (17 \times 10) + (11 \times 4))$$

$$= (30 + 56 + 91 + 170 + 44)$$

$$UFP = 391$$

Adjusted Function Point (FP)

$$= UFP \times \text{Complexity adjustment factor}$$

$$= 391 \times 1.10$$

$$\mathbf{FP = 430.10}$$

Example 5.8.5 *Consider the following function point components and their complexity. If the total degree of influence is 52, find the estimated function points.*

Function type	*Estimated count*	*Complexity*
ELF	*2*	*7*
ILF	*4*	*10*
EQ	*22*	*4*
EO	*16*	*5*
EI	*24*	*4*

AU : Dec.-16. Marks 8

Solution : We will first compute **count total**

Function type	Estimated count	Weight	FP count
ELF	2	7	2 × 7 = 14
ILF	4	10	4 × 10 = 40
EQ	22	4	22 × 4 = 88
EO	16	5	16 × 5 = 80
EI	24	4	24 × 4 = 96
Count Total (VFP)			318

We will compute complexity

$$\text{Adjustment factor (CAF)} = \text{Degree of influence} \times 0.01 + 0.65$$

$$= 52 \times 0.01 + 0.65$$

$$\text{CAF} = 1.17$$

$$\text{FP} = \text{UFP} \times \text{CAF}$$

$$= 318 \times 1.17$$

$$\text{FP} = 372$$

Example 5.8.6 *An application has following : 10 Low External Inputs, 12 High External Outputs, 20 LOW Internal Logical Files, 15 High External Interface Files, 12 Average External Inquiries and a value adjustment factor of 1.10. What is the unadjusted and adjusted function point-count ?*

AU : May-17. Marks 5

Solution : The unadjusted function points are calculated using predefined weights for each function type which is as given below

Functional Units	Weighting Factors		
	Low	Average	High
External Input	3	4	6
External Output	4	5	7
External Inquiries	3	4	6
Internal Logical Files	7	10	15
External Interface Files	5	7	10

The Unadjusted Function Point (UFP)

$$= \sum \text{Funtional units} \times \text{Weighting Factor}$$

$$= (10 \times 3) + (12 \times 7) + (20 \times 7) + (15 \times 10) + (12 \times 4)$$

$$= (30 + 84 + 140 + 150 + 48)$$

$$\text{UFP} = 452$$

Adjusted Function Point (FP)

$$= \text{UFP} \times \text{Complexity adjustment factor}$$

$$= 452 \times 1.10$$

$$\text{FP} = 497.2$$

Example 5.8.7 *Compute the function point FP for a payroll program that reads a file of employees and file of information for the current month and prints cheques for all the employees. The program is capable of handling an interactive command to print an individually requested cheque immediately.* **AU : May-09, Marks 16**

Solution : From given data -

1. Reading a file of employee and file of information for current month = 2 input operations.
2. Print cheques of employee = 1 output
3. There are two files : File of employee and file of information for current month.
4. Interactive command means = User enquiries.

We assume complexity adjustment factor and weighting factors are average.

	Count		Average weighting factor	
Number of input	2	×	4	8
Number of user output	1	×	5	5
Number of user inquiries	1	×	4	4
Number of files	2	×	10	20
Count total			= 37	

$$\text{Function point} = \text{Count total} \times \left[0.65 + \left[0.01 \times \sum_{i=1}^{14} F_i\right]\right]$$

$$= 37 \times (0.65 + (0.01 \times (14 \times 3)) = 37 \times (0.65 + 0.42)$$

$$= 37 \times (1.07)$$

Function point = 39.59

Example 5.8.8 *Compute function point value for a project with following information domain characteristics :*

No. of external inputs - 30

No. of external outputs - 52

No. of external inquiries - 22

No. of log files - 12

No. of external interface files - 2

Assume complexity adjustment values for above are average (4, 5, 4, 10, 7 respectively).

AU : Dec.-14, Marks 6

Solution : Funcion Point (FP) = Count total $\times \left[0.65 + \left[0.01 \times \sum_{i-1}^{14} F_i\right]\right]$

Now we will compute count total

$$\text{Count total} = \sum (\text{Information domain characteristic} \times \text{Complexity adjustment values})$$

$$= [(30 \times 4) + (52 \times 5) + (22 \times 4) + (12 \times 10) + (2 \times 7)]$$

$$= 120 + 260 + 88 + 120 + 14$$

Count total = 602

$$FP = 602 \times \left[0.65 + \left[0.01 \times \sum_{i-1}^{14} F_i\right]\right]$$

$$= 602 \times [0.65 + (0.01 \times (14 \times 3))]$$

$$= 602 \times (0.65 + 0.42)$$

$$= 602 \times 1.07$$

$$= 644.14$$

∴ **Function point = 644.14**

Review Questions

1. *How is functional point analysis methodology is applied in estimation of software size ? Explain. Why FPA methodology is better than LOC methodology ?* **AU : May-17, Marks 8**
2. *Discuss about the metrics for small organizations.* **AU : Dec.-15, Marks 6**
3. *Explain the function point approach to establish the size of a project.* **AU : Dec.-10, 14, Marks 10**
4. *Discuss the process of function point analysis. Explain function point analysis with sample cases for components of different complexity.* **AU : May-18, Marks 13**
5. *List the features of LOC and FP based estimation models. Compare the two models and list the advantages of one over other.* **AU : May-19, Marks 13**
6. *Outline the steps in function point analysis with an example.* **AU : Dec.-19, Marks 15**
7. *Brief out the importance of LOC and FP cost estimation.* **AU : May-22, Marks 7**

5.9 Make Buy Decision

AU : May-16, Marks 8

Software engineering managers are often faced with a make-buy decision to acquire a computer software. Normally following are the options that are used to acquire the software.

1. Purchase or buy the software.
2. Reuse existing partially built components to construct the system.
3. Build the system from scratch.
4. Contract the software development to an outside vendor.

The decision of acquisition of software is critically based on the cost. A tree structure is built to analyze the costs of software which can be acquired using any of the above given ways.

For example - Consider the make-buy decision tree for system S.

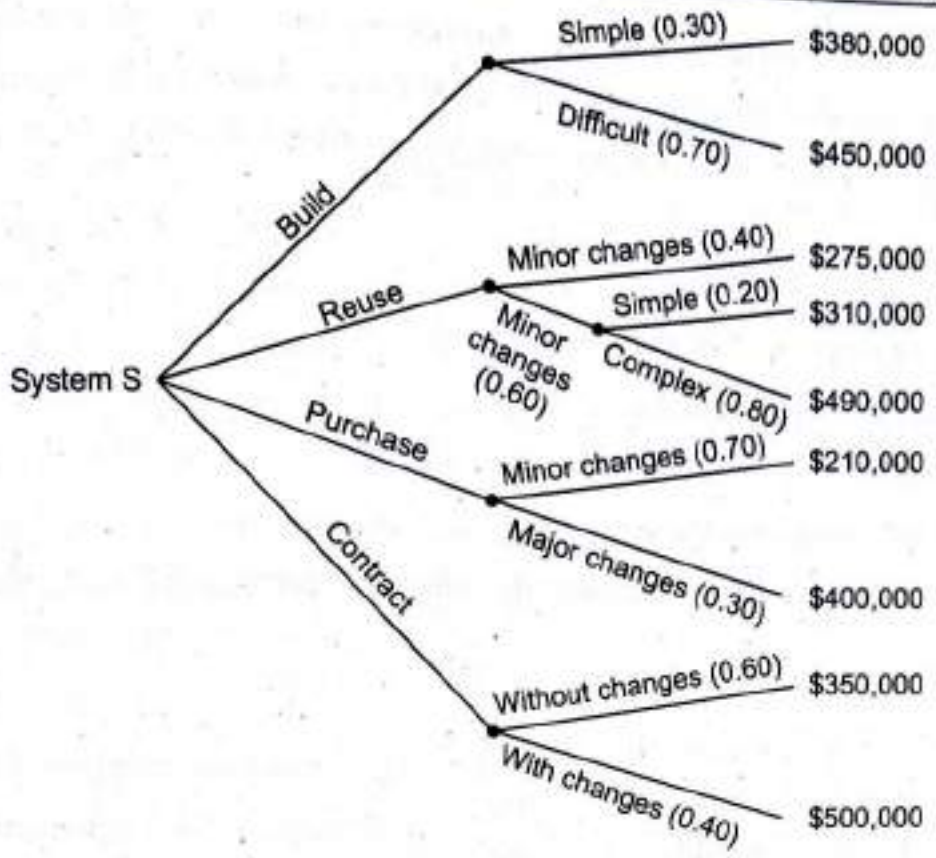


Fig. 5.9.1

Expected cost can be computed for each branch using following formula.

$$\text{Expected cost} = \Sigma \text{ Path Probability of decision tree path} \times \text{Estimated path cost}$$

For example for the branch **system S → Reuse** can be computed as :

$$\text{Expected cost}_{\text{Reuse}} = 0.40(\$275\text{ K}) + [0.60\ (0.20\ (\$310\text{ K}) + 0.80\ (\$490\text{ K}))]$$
$$= \$110\text{ K} + [0.60\ (\$\ 62\text{ K} + \$\ 392\text{ K})]$$
$$= \$110\text{ K} + [0.60\ (\$\ 454\text{ K})]$$
$$= \$110\text{ K} + \$272.4\text{ K}$$
$$= \$382\text{ K}$$

Thus the expected cost at each node can be computed. It is summerised as given below -

Node	Expected cost
Build	$429 K
Reuse	$382 K
Purchase	$267 K
Contract	$410 K

From this we can conclude that by purchasing the software we select for lowest expected cost option. But simply cost should not be a criterion to acquire the software. During decision making process for software acquisition following factors should also be considered.

1. Availability of reliable software.
2. Experience of developer or vendor or contractor.
3. Conformance to requirements.
4. Local politics.
5. Likelihood of changes in the software.

These are some criteria which can heavily affect the decision of make-buy of software.

5.9.1 Outsourcing

Outsourcing is a process in which software engineering activities are contracted to a third party who does the work at lowest cost with high quality.

In strategic level, the significant portion of software work can be contracted to third party.

In Tactical level, a project manager determines whether part or all of the software can be accomplished with good quality by contracting it.

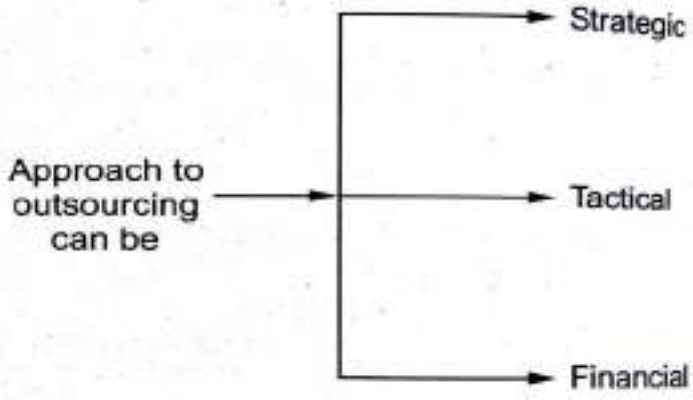


Fig. 5.9.2 Approaches for outsourcing

In financial level, the cost is the prime factor in the decision of outsourcing.

Benefits of outsourcing

1. Cost savings - If a software is outsourced then people and resource utilization can be reduced. And thereby the cost of the project can be saved effectively.

2. Accelerated development - Since some part of software gets developed simultaneously by a third party, the overall development process gets accelerated.

Drawbacks of outsourcing

1. A software company loses some control over the software as it is developed by third person.

2. The trend of outsourcing will be continued in software industry in order to survive in competitive world.

Review Question

1. *Write short note - Make/Buy decision.* **AU : May-16, Marks 8**

5.10 COCOMO I Model

AU : May-07,08,14,15,17,22, Dec.-05,13,22, Marks 16

COCOMO is one of the most widely used software estimation models in the world. This model is developed in 1981 by **Barry Boehm** to give an estimate of the number of man-months it will take to develop a software product. COCOMO predicts the efforts and schedule of a software product based on **size** of the software. COCOMO stands for "COnstructive **CO**st **MO**del".

COCOMO has three different models that reflect the complexity -

- Basic model
- Intermediate model
- Detailed model.

Similarly there are three classes of software projects.

1) Organic mode : In this mode, relatively small, simple software projects with a small team are handled. Such a team should have good application experience to less rigid requirements.

2) Semi-detached projects : In this class an intermediate projects in which teams with mixed experience level are handled. Such projects may have mix of rigid and less than rigid requirements.

3) Embedded projects : In this class, projects with tight hardware, software and operational constraints are handled.

Let us understand each model in detail.

1) Basic model : The basic COCOMO model estimates the software development effort using only **Lines of Code**. Various equations in this model are -

$$E = a_b(KLOC)^{b_b}$$

$$D = c_b(E)^{d_b}$$

$$P = E/D$$

Where E is the **effort** applied in person-months.

D is the development time in chronological months.

KLOC means kilo line of code for the project.

P is total number of persons required to accomplish the project.

The coefficients a_b, b_b, c_b, d_b for three modes are as given below.

Software projects	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Table 5.10.1

Merits of basic COCOMO model

Basic COCOMO model is good for quick, early, rough order of magnitude estimates of software project.

Limitations of basic model

1. The accuracy of this model is limited because it does not consider certain factors for cost estimation of software. These factors are hardware constraints, personal quality, and experience, modern tachniques and tools.
2. The estimates of COCOMO model are within a factor of 1.3 only 29 % of the time and within the factor of 2 only 60 % of time.

Example

Consider a software project using semi-detached mode with 30,000 lines of code. We will obtain estimation for this project as follows -

i) Effort estimation

$$E = a_b(KLOC)^{b_b}$$

i.e. $E = 3.0\,(30)^{1.12}$ where lines of code = 30000 = 30 KLOC

$E = 135$ person-month

ii) Duration estimation

$$D = C_b(E)^{d_b}$$

$$= 2.5(135)^{0.35}$$

$D = 14$ months

iii) Persons estimation

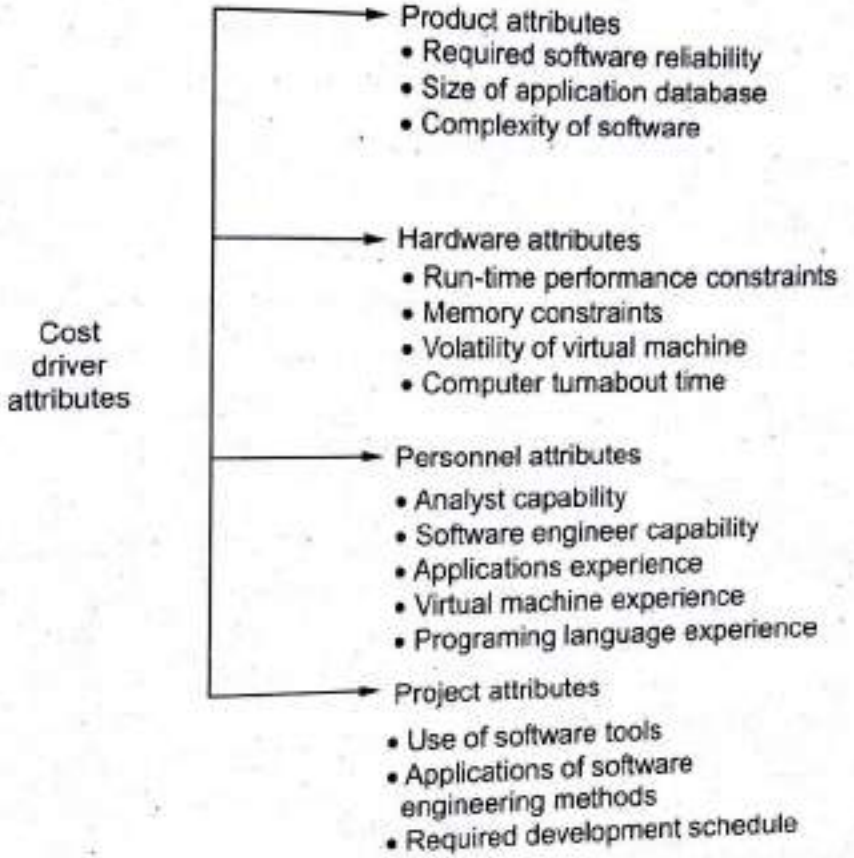
$$P = E/D$$

$$= 135/14$$

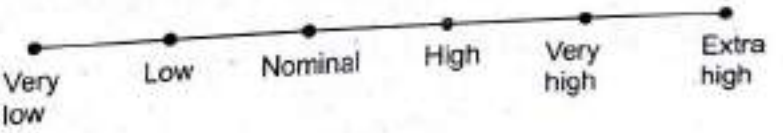
$P = 10$ persons approximately

2) Intermediate model

This is an extension of Basic COCOMO model. This estimation model makes use of set of "Cost driver attributes" to compute the cost of software.



Now these 15 attributes get a 6-point scale ranging from "very low" to "extra high". These ratings can be viewed as



The effort multipliers for each cost driver attribute is as given in following table. The product of all effort multipliers result in "Effort Adjustment Factor" (EAF).

Cost drivers	Ratings					
	Very low	Low	Nominal	High	Very high	Extra high
Product attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Size of application database		0.94	1.00	1.08	1.16	

Complexity of software	0.70	0.85	1.00	1.15	1.30	1.65
Hardware attributes						
Run-time performance constraints			1.00	1.11	1.30	1.66
Memory constraints			1.00	1.06	1.21	1.56
Volatility of virtual machine		0.87	1.00	1.15	1.30	
Computer turnabout time		0.87	1.00	1.07	1.15	
Personnel attributes						
Analyst capability	1.46	1.19	1.00	0.86	0.71	
Software engineer capability	1.42	1.17	1.00	0.86	0.70	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project attributes						
Use of software tools	1.24	1.10	1.00	0.91	0.82	
Applications of software engineering methods	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

Table 5.10.2

The formula for effort calculation can be -

$$E = a_i (KLOC)^{b_i} \cdot EAF \text{ person-months}$$

The values for a_i and b_i for various class of software projects are -

Software project	a_i	b_i
Organic	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.20

Table 5.10.3

The duration and person estimate is same as in basic COCOMO model. i.e.

$$D = c_b(E)^{d_b} \text{ months}$$

$$P = E/D \text{ persons}$$

i.e. use values of c_b and d_b coefficients that are in Table 5.10.1

Merits of intermediate model

1. This model can be applied to almost entire software product for easy and rough cost estimation during early stage.
2. It can also be applied at the software product component level for obtaining more accurate cost estimation.

Limitations of intermediate model

1. The estimation is within 20 % of actual 68 % of the time.
2. The effort multipliers are not dependent on phases.
3. A product with many components is difficult to estimate.

Example

Consider a project having 30,000 lines of code which is an **embedded** software with critical area hence reliability is high. The estimation can be

$$E = a_i(KLOC)^{b_i} \cdot EAF$$

As reliability is high, EAF = 1.15 (product attribute)

$$\left.\begin{matrix} a_i = 2.8 \\ b_i = 1.20 \end{matrix}\right\} \text{ for embedded software}$$

$\therefore$ $E = 2.8(30)^{1.20} * 1.15$

$= 191$ person-month

$D = c_b(E)^{d_b} = 2.5(191)^{0.32}$

$= 13$ months approximately

$P = E/D$

$= 191/13$

$P = 15$ persons approximately

3) Detailed COCOMO model

The detailed model uses the same equations for estimation as the Intermediate Model. But detailed model can estimate the effort (E), duration (D) and persons (P) of each of development phases, subsystems, modules.

The experimentation with different development strategies is allowed in this model.

Four phases used in detailed COCOMO model are -

1. Requirements Planning and Product Design (RPD)
2. Detailed Design (DD)
3. Code and Unit Test (CUT)
4. Integrate and Test (IT)

The effort multipliers for detailed COCOMO are

Phases	Very low	Low	Nominal	High	Very high
RPD	1.80	0.85	1.00	0.75	0.55
DD	1.35	0.85	1.00	0.90	0.75
CUT	1.35	0.85	1.00	0.90	0.75
IT	1.50	1.20	1.00	0.85	0.70

Using these detailed cost drivers, an estimate is determined for each phase of the lifecycle.

Review Questions

1. *Describe in detail COCOMO model for software cost estimation. Illustrate considering a suitable example.* **AU : May-17, Marks 13**
2. *Discuss about the COCOMO models (Basic, intermediate and detailed) for cost estimation.* **AU : May-14, 15, Marks 16**
3. *Explain how effort and cost estimation are determined using COCOMO model ?* **AU : Dec.-20, Marks 13**
4. *Discuss about the basic COCOMO model with advantages and limitations.* **AU : Dec.-22, Marks 8**
5. *Explain COCOMO model for software estimation.* **AU : May-22, Marks 7**

5.11 COCOMO II Model

AU : Dec.-07,10,11,14,15,16,19, May-07,16,17,18, Marks 16

COCOMO II is applied for modern software development practices addressed for the projects in 1990's and 2000's.

The sub-models of COCOMO II model are -

1. Application composition model

- For estimating the efforts required for the prototyping projects and the projects in which the existing software components are used application-composition model is introduced.
- The estimation in this model is based on the number of application points. The application points are similar to the object points.

- This estimation is based on the level of difficulty of object points. Boehm has suggested the object point productivity in the following manner.

Developers experience and capability	Very low	Low	Nominal	High	Very high
CASE maturity	Very low	Low	Nominal	High	Very high
Productivity (NOP/Month)	4	7	13	25	50

- Effort computation in application-composition model can be done as follows -

$$PM = (NAP^{(1-\%reuse/100)}) / PROD$$

where

PM means effort required in terms of person-months.

NAP means number of application points required.

% reuse indicates the amount of reused components in the project. These reusable components can be screens, reports or the modules used in previous projects.

PROD is the object point productivity. These values are given in the above table.

2. An early design model

- This model is used in the early stage of the project development. That is after gathering the user requirements and before the project development actually starts, this model is used. Hence approximate cost estimation can be made in this model.
- The estimation can be made based on the functional points.
- In early stage of development different ways of implementing user requirements can be estimated.
- The effort estimation (in terms of person month) in this model can be made using the following formula :

$$\text{Effort} = A \times \text{size}^B \times M$$

where

Boehm has proposed the value of coefficient **A = 2.94**.

Size should be in terms of Kilo source lines of code i.e. KSLOC.

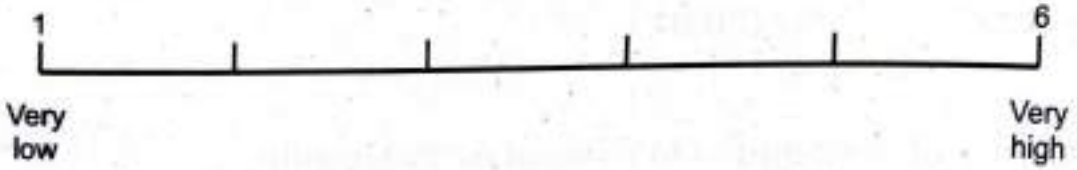
The lines of code can be computed with the help of function point.

The value of **B** is varying from 1.1 to 1.24 and depends upon the project.

M is based on the characteristics such as

1. Product reliability and complexity (RCPX)
2. Reuse required (RUSE)
3. Platform difficulty (PDIF)
4. Personnel capability (PERS)
5. Personnel experience (PREX)
6. Schedule (SCED)
7. support facilities (FCIL)

These characteristics values can be computed on the following scale -



- Hence the effort estimation can be given as

$$PM = 2.94 \times size^{B} \times M$$

$$M = RUSE \times PDIF \times PERS \times PREX \times SCED \times FCIL$$

3. A reuse model

- This model considers the systems that have significant amount of code which is reused from the earlier software systems. The estimation made in reuse model is nothing but the efforts required to integrated the reused models into the new systems.
- There are two types of reusable codes : Black box code and white box code. The black box code is a kind of code which is simply integrated with the new system without modifying it. The white box code is a kind of code that has to be modified to some extent before integrating it with the new system, and then only it can work correctly.
- There is third category of code which is used in reuse model and it is the code which can be generated automatically. In this form of reuse the standard templates are integrated in the generator. To these generators, the system model is given as input from which some additional information about the system is taken and the code can be generated using the templates.
- The efforts required for automatically the generated code is,

$$PM = (ASLOC \times AT/100)/ATPROD$$

where

AT is percentage of automatically generated code.

ATPROD is the productivity of engineers in estimating such code

- Sometimes in the reuse model some white box code is used along with the newly developed code. The size estimate of newly developed code is equivalent to the reused code. Following formula is used to calculate the effort in such a case -
- $ESLOC = ASLOC \times (1-AT/100) \times AAM$

where

ESLOC means equivalent number of lines of new source code.

ASLOC means the source lines of code in the component that has to be adapted.

AAM is adaptation Adjustment multiplier. This factor is used to take into account the efforts required to reuse the code.

4. Post architecture model

- This model is a detailed model used to compute the efforts. The basic formula used in this model is

$$\text{Effort} = \mathbf{A} \times \mathbf{Size}^{\mathbf{B}} \times \mathbf{M}$$

- In this model efforts should be estimated more accurately. In the above formula **A** is the amount of code. This code size estimate is made with the help of three components -
  1. The estimate about **new** lines of code that is added in the program.
  2. Equivalent number of source lines of code (ESLOC) used in **reuse model**.
  3. Due to changes in requirements the lines of code get modified. The estimate of amount of **code** being **modified**.

The exponent term **B** has three possible values that are related to the levels of project complexity. The values of B are continuous rather than discrete. It depends upon the five scale factors. These scale factors vary from very low to extra high (i.e. from 5 to 0).

- These factors are -

Scale factor for component B	Description
Precedentedness	This factor is for previous experience of organisation. Very low means no previous experience and high means the organisation knows the application domain.
Development flexibility	Flexibility in development process. Very low means the typical process is used. Extra high means client is responsible for defining the process goals.

Architecture/risk resolution	Amount of risk that is allowed to carry out. Very low means little risk analysis is permitted and extra high means high risk analysis is made.
Team cohesion	Represents the working environment of the team. Very low cohesion means poor communication or interaction between the team members and extra high means there is no communication problem and team can work in a good spirit.
Process maturity	This factor affects the process maturity of the organisation. This value can be computed using Capacity Maturity Model (CMM) questionnaire, for computing the estimates CMM maturity level can be subtracted from 5.

- Add up all these rating and then whatever value you get, divide it by 100. Then add the resultant value to 1.01 to get the exponent value.
- This model makes use of 17 cost attributes instead of seven. These attributes are used to adjust initial estimate.

Cost attribute	Type of attribute	Purpose
RELY	Product	System reliability that is required.
CPLX	Product	Complexity of system modules
DATA	Product	Size of the data used from database
DOCU	Product	Some amount of documentation used
RUSE	Product	Percentage of reusable components
TIME	Computer	Amount of time required for execution
PVOL	Computer	Volatility of development platform.
STOR	Computer	Memory constraint
ACAP	Personnel	Project analyst's capability to analyse the project.
PCAP	Personnel	Programmer capability
PCON	Personnel	Personnel continuity
PEXP	Personnel	Programmer's experience in project domain.
LTEX	Personnel	Experience of languages and tools that are used.
AEXP	Personal	Analyst's experience in project domain.

TOOL	Project	Use of software tools
SCED	Project	Project schedule compression.
SITE	Project	Quality of inter-site and multi-site working.

Example 5.11.1 *Describe in detail COCOMO model for software cost estimation. Use it to estimate the effort required to build software for a simple ATM that produces 12 screens, 10 reports and has 80 software components. Assume average complexity and average developer maturity. Use application composition model with object points.* **AU : Dec.-16, Marks 16**

Solution : The number of screens, reports and components are weighted according the table as given below

Object type	Complexity Weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
Software Components			10

As there is average complexity and average developer maturity, we will compute object point as

Object point = $12 * 2 + 10 * 5 + 80 * 10$

Object point = 874

Example 5.11.2 *Using COCOMO, estimate time required for the following :*

1) A semi-detached model of software project of 2000 lines.

2) An embedded model of software of 30,000 lines.

3) An organic model of software of one lakh lines.

4) An organic model of software of 10 lakh lines. **AU : May-07. Marks 8**

Solution : To estimate time using basic model of COCOMO following formula can be used.

$$E = a_b(KLOC)^{b_b}$$

where E is the effort in person-month.

$$D = c_b(E)^{d_b}$$

where D is development time in chronological months.

$$P = E/D$$

System	a_b	b_b	c_b	d_b
Organic system	2.4	1.05	2.5	0.38
Semidetached system	3.0	1.12	2.5	0.35
Embedded system	3.6	1.20	2.5	0.32

Given that, System = Semidetached

es of code = 2000 lines = 2 KLOC

$$E = a_b(KLOC)^{b_b}$$

$$E = 3.0(2)^{1.15}$$

$$E = 6.65 \text{ person-month}$$

$$D = c_b(E)^{d_b}$$

$$D = 4.8 \text{ months}$$

$$P = E/D$$

$$P = 1.3 \approx 1 \text{ person}$$

nus 1 person can handle this project within approximately **5 months.**

ven that, System = Embedded

nes of code = 30,000 lines = 30 KLOC

$$E = a_b(KLOC)^{b_b}$$

$$= 3.6(30)^{1.20}$$

$$E \approx 213 \text{ person - month}$$

$$D = c_b(E)^{d_b}$$

$$= 2.5(213)^{0.32}$$

$$D \approx 14 \text{ months}$$

$$P = E/D$$

$$\approx 213/14$$

$$\approx 15 \text{ persons.}$$

That means 15 persons can complete this project within approximately **14 months.**

Given that,system = Organic

Lines of code = 1 lakh = 100 KLOC

$$E = a_b(KLOC)^{b_b}$$

$$\therefore \quad = 2.4\,(100)^{1.05}$$

$$E \approx 302 \text{ person-month}$$

$$D = c_b(E)^{d_b} = 2.5(302)^{0.38}$$

$$\approx 21 \text{ months}$$

$$P = E/D$$

$$\therefore \quad \approx 302/21$$

$$\approx 14 \text{ persons.}$$

That means this project can be completed within 21 months by 14 persons, approximately,

4) Given that, System = Organic

Lines of code = 10 lakh = 1000 KLOC

$$\therefore \quad E = a_b(KLOC)^{b_b} = 2.4(1000)^{1.05}$$

$$\approx 3390 \text{ person - month}$$

$$D = c_b(E)^{d_b}$$

$$\approx 2.5\,(3390)^{0.38}$$

$$\approx 55 \text{ months}$$

$$\therefore \quad P = E/D$$

$$= 3390/55$$

$$\approx 61 \text{ persons}$$

This project can be completed within 55 months by 61 people approximately.

Example 5.11.3 *Calculate the effort and duration using the above details for basic COCOMO model.*

Given,

Number of user inputs = 15

Number of user outputs = 3

Number of external interfaces = 11

1 function point = 20 LOC (as fourth generation language is used).

Values of constant used in basic COCOMO model. a = 2.4, b = 1.05, c = 2.5, d = 0.38.

AU : Dec.-11, Marks 8

Solution : We will first calculate FP from given data. Assume all complexity adjustment factors and weighting factors as average.

$$\therefore \quad UFP = 15 \times 4 + 3 \times 5 + 11 \times 7 = 152$$

$$CAF = 0.65 + 0.01 \times \left(\sum F_i\right) = 0.65 + 0.01 \times (14 \times 3) = 1.07$$

$$FP = UFP \times CAF = 152 \times 1.07 = 163$$

$$1FP = 20 \text{ LOC}$$

$$\therefore \quad 163 = 3260 \text{ LOC} = 3.26 \text{ KLOC.}$$

The basic COCOMO equation take the form :

$$E = a_b(KLOC)^{b_b}$$

$$D = C_b(KLOC)^{d_b}$$

For organic mode

$$E = 2.4\,(3.26)^{1.05} \qquad E = 8.28 \text{ PM}$$

$$D = 2.5\,(3.26)^{0.38} \qquad D = 3.9 \text{ M}$$

Review Questions

1. *Discuss about COCOMO II model for software estimation.* **AU : Dec.-15, Marks 10**
2. *Write short notes - COCOMO II.* **AU : May-16, Marks 8**
3. *Describe in detail COCOMO model for software cost estimation. Illustrate considering suitable example.* **AU : May-17, Marks 16, May-18, Marks 9**
4. *Elaborate the cost estimation COCOMO II cost estimation model.* **AU : Dec.-19, Marks 13**

5.12 Software Configuration Management

Definition : Software configuration management is a set of activities carried out for identifying, organizing and controlling changes throughout the lifecycle of computer software.

- During the development of software change must be managed and controlled in order to improve quality and reduce error.
- Hence Software Configuration Management is a quality assurance activity that is applied throughout the software process.
- The **origins of changes** that are requested for software are -
  1. New business or market positions cause the changes in the requirements. Due to which the changes need to occur.
  2. New stakeholders may require some changes in the existing requirements.

3. Due to business growth or project extension, it is essential to makes changes in the project.
4. Sometimes due to schedule or budge constraints, changes are must in project.

- The software configuration management is concerned with managing evolving software systems.
- Software configuration management is a set of tracking and control activities that begin when a software development project begins and terminates when the software is taken out of operation.

5.12.1 SCM Basics

5.12.1.1 Goals in Change Management

- During software development process, various roles and tasks are involved.
- The goal of project manager is to ensure that the project is getting completed within given schedule and budget. Hence project managers continuously track the progress of the project.
- The configuration managers ensures that the changes in the projects is appropriately taking place and the testing is conducted thoroughly after the modification in the project.
- The goal of software engineer is to work on the assigned module to produce efficient code and error free code.
- Finally the customer uses the product, analyze it and may request for the change or find out the bugs.

5.12.1.2 Elements of Configuration Management System

- Susan Dart identified four important elements for software configuration management system. These elements are -

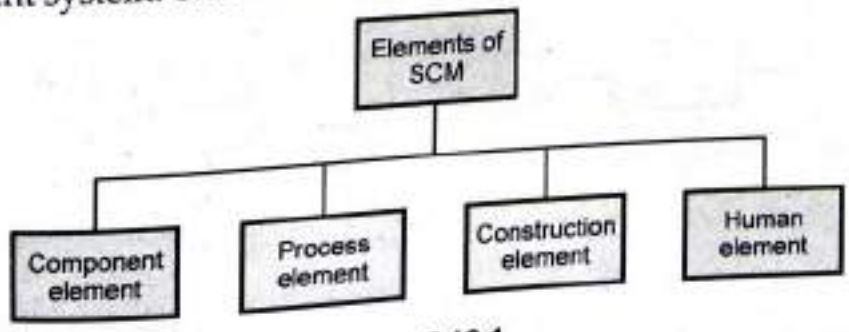


Fig. 5.12.1

1. **Component Elements :** It consists of collection of tools that are used for file management system. For example - databases.

2. **Process Elements :** It consists of actions and tasks used during change management and use of software.
3. **Construction Elements :** It is a collection of tools that automate the construction of software.
4. **Human Elements :** It consists of set of tools that are used by software team to implement software configuration management.

5.12.1.3 Baselines

- The IEEE (IEEE Std. No. 610.12-1990) defines a baseline as :
  - A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.
- A baseline is a milestone in the development of software that is marked by the delivery of one or more software configuration items and the approval of them is obtained through formal technical review.
- The baseline is the shared project database. It is an SCM task to maintain the integrity of the set of artifacts.

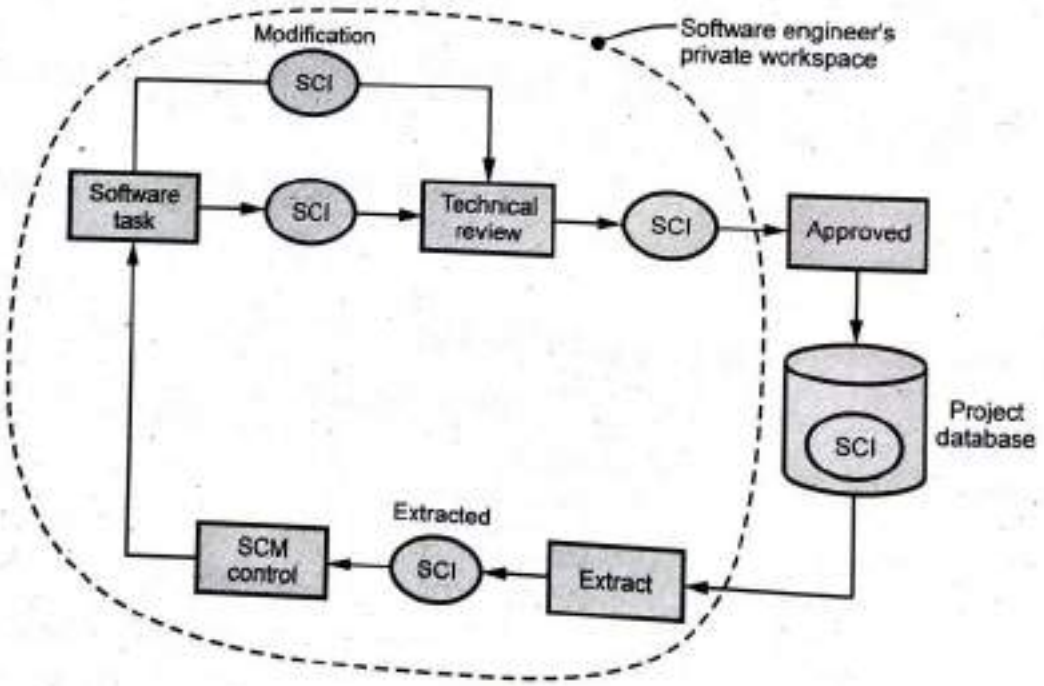


Fig. 5.12.2 Software baseline

- The elements of a design model have been first documented and reviewed. From this design model errors are identified and corrected. Once all parts of the model have been reviewed, corrected and then approved, the design model becomes a baseline.

- Fig. 5.12.2 represents the common software baseline.
  - **Step 1 :** The Software engineering task produce one or more SCIs.
  - **Step 2 :** These SCIs are then reviewed and approved.
  - **Step 3 :** These are then placed in project database or project library.
  - **Step 4 :** When software engineer wants to make changes to baselined SCI, it is copied to software engineers private workspace.
  - **Step 5 :** The extracted SCI is modified only if SCM controls are followed.

5.12.1.4 Software Configuration Items

- Software configuration item is basically the information that is created as a part of software engineering process.
- Examples of Software Configuration Items are
  - Computer programs
    - Source programs
    - Executable programs
  - Documents describing the programs
    - Technical manual
    - Users manual
  - Data
    - Program components or functions
    - External data
    - File structure
- For each type of item, there may be a large number of different individual items produced. For instance there may be many documents for a software specification such as project plan, quality plan, test plan, design documents, programs, test reports, review reports.
- These SCI or items will be produced during the project, stored, retrieved, changed, stored again and so on.
- Each configuration item must have a unique name and a description or specification which distinguishes it from other items of the same type.

5.12.2 SCM Repository

- Software Configuration Items (SCI) are maintained in project repository or project library.

- The software repository is basically a database that acts as a center for both accumulation and storage for software engineering information.
- The software engineer interacts with repository using tools that are integrated within it.

5.12.2.1 Role of Project Repository

- Software repository is a collection of information accessed by software engineers to make appropriate changes in it.
- This repository is handled using all the modern database management functions.
- It must maintain important properties such as data integrity, sharing and integration.
- The repository must maintain uniform structure and format for software engineering work products.
- To achieve these capabilities, the repository is defined in terms of **meta-model**. The meta model determines -
  i) How information is stored in repository ?
  ii) How data can be accessed by using the tool ?
  iii) How the security and integrity of the data is maintained ?
  iv) How the existing model can be extended to accommodate new requirements ?

5.12.2.2 Features

The Software Configuration Management can be performed by interacting with repository. The repository must have toolset that provides support for following features -

1. **Versioning :** As project progresses various versions of individual work products may get created. The repository must be able to maintain all these versions and permit the developer to go back to previous versions during testing and debugging.
2. **Dependency Tracking and Change Management :** The data elements stored in the repository are related to each other. The repository must have an ability to keep track of these relationship and to maintain data integrity.
3. **Requirements Tracing :** If the constructed components, its designed is tracked, then particular requirement can be traced out. The repository must have this ability to trace the requirement from constructed components.

4. **Configuration Management :** The repository must be able to keep track of series of configurations representing project milestones and production releases.
5. **Audit Trails :** The audit trail.

5.12.3 SCM Process

The primary objectives of Software Configuration Management process (SCM) are -

1. **Configuration Identification :** Identify the items that define the software configuration.
2. **Change Control :** Manage changes to one or more items.
3. **Version Control :** Facilitate to create different versions of the application.
4. **Configuration Authentication :** To ensure that the quality of the software is maintained as the configuration evolves over the time.

The SCM process must be developed in such a way that the software team must answer the following set of questions -

1. How does the software team identify the software configuration items ?
2. How does the software team control the changes in the software before and after delivering it to the customer ?
3. How does the software team manage the versions of the programs in the software package ?
4. How does team get ensured that the changes are made properly ?
5. Who is responsible for approving the changes in the software ?

The answers to these questions lead the definition of five tasks of SCM and those are - Identification, change control, version control and configuration audit and status reporting.

5.12.3.1 Identification of Objects in Software Configuration

The software configuration items must be separately named and identified as object.

- These objects must be arranged using **object oriented approach**.
- There are two categories of objects - **basic objects** and **aggregate objects**.
- The **basic object** is unit of information created during requirements analysis, design, coding or testing. For example basic object can be part of **source code**.
- Aggregate object is a collection of basic objects and other aggregate objects. For example SRS or data model can be aggregate object.

- Each object can be uniquely identified because it has got -
  1. **Name :** The name of the object is nothing but the collection of characters (string) or some text. It is unique.
  2. **Description :** For describing the object, the object description can be given. This description contains document, program or some other description such as project identifier or version information.
  3. **List of resources :** The resources are the entities that are used for accessing, referencing and processing of objects. The data types and functionalities can serve as a resource.
  4. **Realization or identification :** It is pointer to object.
- The configuration object identification can also consider relationships that exist between the named objects.
- If a change is made to one configuration object it is possible to determine which other configuration objects in the repository are affected by the change.
- Basically object evolve throughout the software process. During the process of object identification the evolution of objects along with its process must be identified.
- Major modifications in the object must be noted.

5.12.3.2 Change Control

- Changes in any software projects are vital. Sometimes, introducing small changes in the system may lead to big problems in product.
- Similarly, introducing some changes may enhance the capabilities of the system.
- According to **James Bach** too little changes may create some problems and too big changes may create another problems.
- For a large software engineering project, uncontrolled change creates lot of chaos. For managing such changes, human procedures or automated tools can be used.
- The change control process is shown by following Fig. 5.12.3.
(See Fig. 5.12.3 on next page)
  - **Step 1 :** First of all there arises a need for the change.
  - **Step 2 :** The change request is then submitted by the user.
  - **Step 3 :** Developers evaluate this request to assess technical merits, potential side effects and overall impact on system functions and cost of the project.

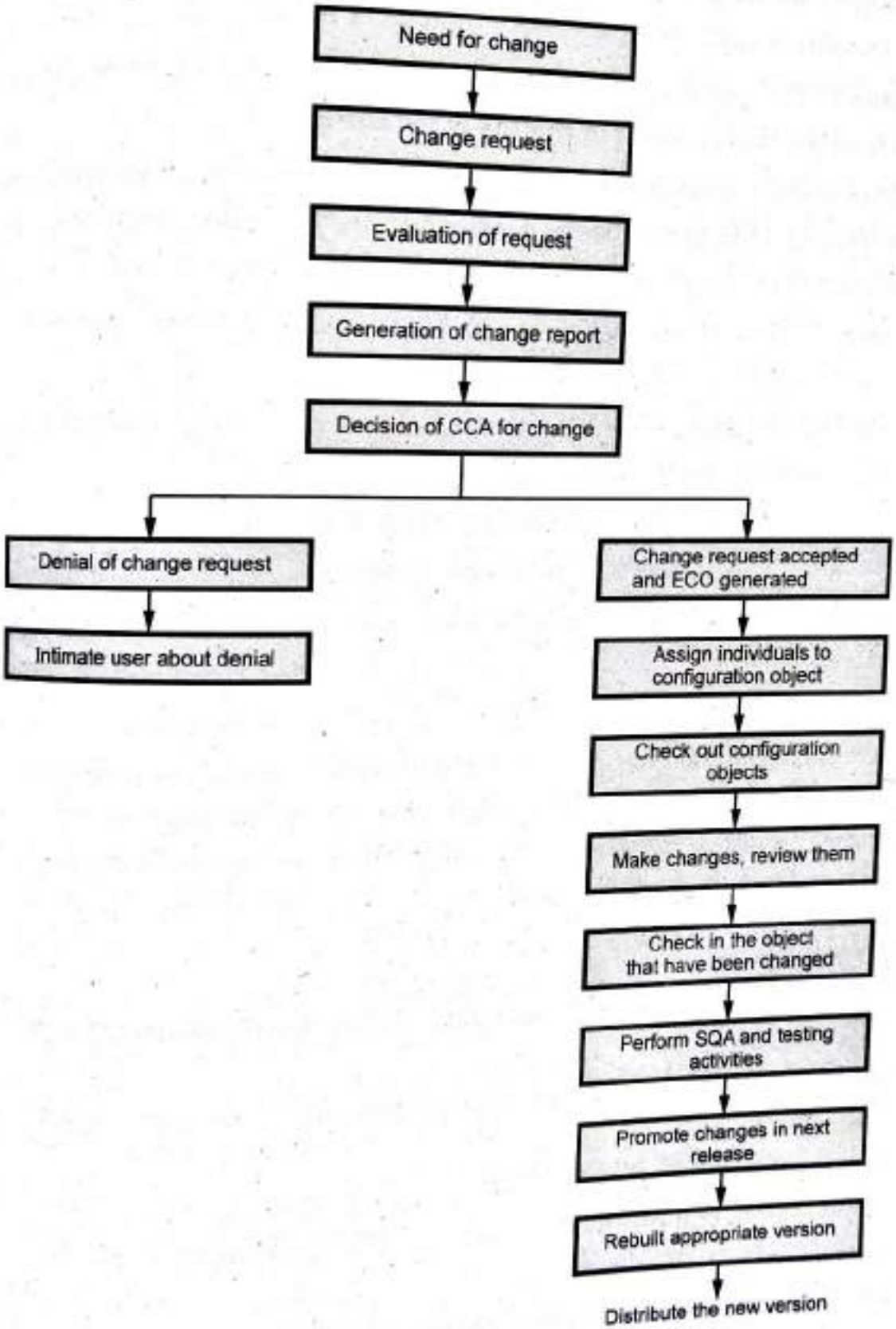


Fig. 5.12.3 Change control process

- **Step 4 :** A change report is then generated and presented to the Change Control Authority (CCA).
- **Step 5 :** The change control authority is a person or a group of people who makes a final decision on status or priority of the change.
- **Step 6 :** An Engineering Change Order (ECO) is generated when the change gets approved. In ECO the change is described, the restrictions and criteria for review and audit are mentioned.
- **Step 7 :** The object that needs to be changed is checked out of the project database.
- **Step 8 :** The changes are then made on the corresponding object and appropriate SQA activities are then applied.
- **Step 9 :** The changed object is then checked in to the database and appropriate version control is made to create new version.

The checked in and checked out mechanisms require two important elements -

- Access control
- Synchronization control

The **access control** mechanism gives the authority to the software engineer to access and modify the specific configuring object. The **synchronization control** mechanism allows to make parallel changes or the changes made by two different people without overwriting each other's work.

5.12.3.3 Version Control

- Version is an instance of a system which is functionally distinct in some way from other system instances.
- Version control works to help manage different versions of configuration items during the development process.
- The configuration management allows a user to specify the alternative configurations of the software system by selecting appropriate version.
- Certain attributes are associated with each software version. These attributes are useful in identifying the version. For example : The attribute can be 'date', 'creator', 'customer', 'status'.
- In practice the version needs an associated name for easy reference.

- Different versions of a system can be shown by an evolution graph as shown in Fig. 5.12.4.
- Each version of software system is a collection of software configuration items.

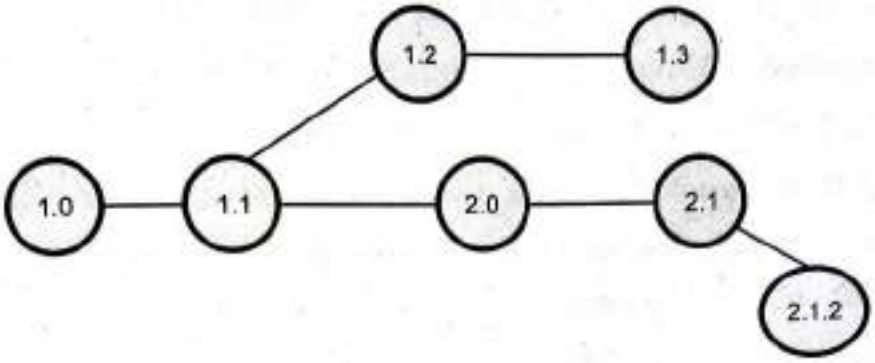


Fig. 5.12.4 Version numbering in evolution graph

5.12.3.4 Configuration Audit

- In order to ensure that the change has been properly implemented or not two activities are carried out.
  1. Formal Technical Review (FTR)
  2. Software Configuration Audit
- In Formal Technical Review, the correctness of configuration object is identified and corrected. It is conducted by technical reviewer.
- The software configuration audit assess the configuration object for the characteristics that are not reviewed in formal technical review. It is conducted by software quality assurance group.
- Following are some primary questions that are asked during configuration audit -
  1. Whether FTR is conducted to assess the technical correctness ?
  2. Whether or not the change specified by ECO has been made ?
  3. If additional changes need to be made or not ?
  4. Whether the software engineering standards are properly followed ?
  5. Do the attributes of configuration objects reflect the change ?
  6. Whether all the SCI are updated properly ?
  7. Whether the SCM process (object identification, change and version control, configuration audit and status reporting) are properly followed ?
- The above questions can be asked as a part of formal technical review.

5.12.3.5 Status Reporting

- The status reporting focuses on communication of changes to all people in an organization that involve with changes.
- During status reporting following type of questions were asked.
  1. **What happened ?** : What are the changes that are required ?
  2. **Who did it ?** : Who will be handling these changes ?
  3. **When did it happen ?** : The time at which these changes are arised.
  4. **What else will be affected ?** : The objects or part of the software that might be reflected due to these changes.

Review Questions

1. *What is configuration management repository ? Discuss role and features of SCM repository.*
2. *Which are the layers of SCM process ? Explain each in detail.*

5.13 Project Scheduling

AU : May-05,15,16,17,22, Dec.-06,07,15,16, Marks 16

- While scheduling the project, the manager has to estimate the time and resource of the project. All the activities in the project must be arranged in **coherent sequence**.
- The schedules must be continually updated because some uncertain problems may occur during the project life cycle.
- For new projects initial estimates can be made optimistically.
- During the project scheduling the total work is separated into various **small activities**. And time required for each activity must be determined by the project manager.
- For efficient performance some activities are conducted **in parallel**.

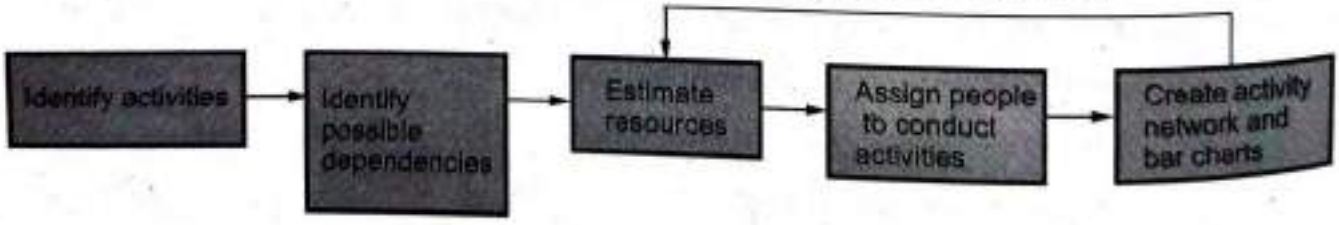


Fig. 5.13.1 Project scheduling process

- The project manager should be aware of the fact that : Every stage of the project may not be problem-free. Some of the typical problems in project development stage are :
  i) People may leave or remain absent.

ii) Hardware may get failed.

iii) Software resource may not be available.

- To accomplish the project within given schedule the required resources must be available when needed. Various resources required for the project are -

i) Human effort.

ii) Sufficient disk space on server.

iii) Specialized hardware.

iv) Software technology.

v) Travel allowance required by the project staff.

- Project schedules are represented as the set of chart in which the work-breakdown structure and dependencies within various activities is represented.

5.13.1 Relationship between People and Effort

- People work on the software project doing various activities such as requirements gathering, design, analysis, coding and testing.
- There is a common myth among the software managers that by adding more people in the project, the deadline can be achieved. But this is not true - as by adding more people in the project, first we need to train them for the tools and technologies that are getting used in the project. And only those people can teach the new people who are already working. Thus during teaching or training the time will be simply wasted and there won't be the progress in the project.
- Once the effort required to develop a software has been determined, it is necessary to determine the people or staff requirement for the project.
- **Putnam** first studied the on how much staffing is required for the projects. He extended the work of **Norden** who had earlier investigated the staffing pattern.
- The **Putnam-Norden-Rayleigh Curve(PNR Curve)** represents the relationship between effort applied and delivery time for the software project.
- Following equation shows the relationship of project effort as a function of project delivery time.

$$E_a = m \left(t_d^4 / t_d^4\right)$$

Where

E_a denotes the effort in person months

t_d denotes the nominal delivery time for the schedule

t_a denotes the actual delivery time desired.

- Let t_o will be the delivery time at which least effort is expended.
- As we move to left of t_o and accelerate delivery the curve rises non linearly.
- The curve rises sharply left to t_d indicating that project delivery time can not be compressed beyond 0.75 t_d.
- Beyond this the failure region is shown and failure risk becomes high.
- **Software equation :** The software equation can be derived from the PNR curve.
- It represents the non linear relationship between **time to complete the project** and **human effort** applied to the project. It can be denoted as

$$L = P \times E^{1/3} t^{4/3}$$

that is

$$E = L^3 / P^3 t^4$$

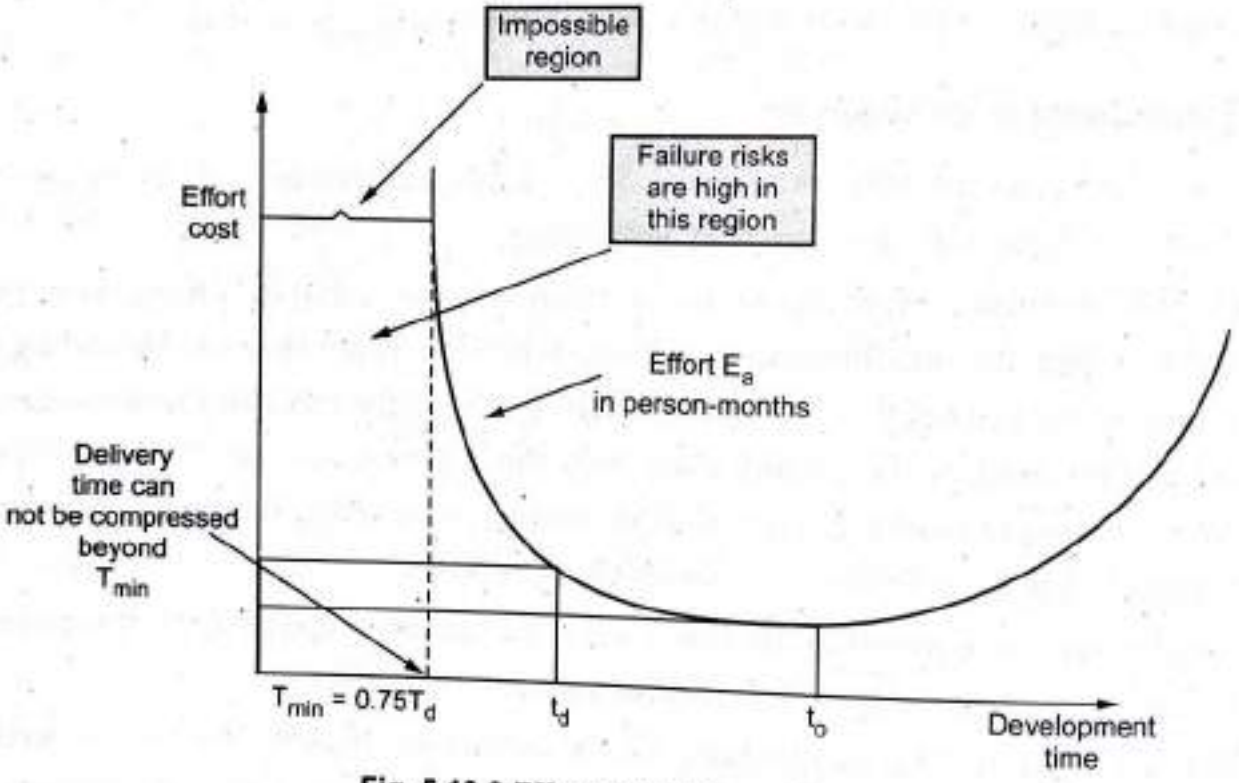


Fig. 5.13.2 Effort and delivery time

That also implies that by extending the end date six months, we can reduce the number of people.

5.13.2 Task Sets

- **Definition of task set :** The task set is a collection of software engineering work tasks, milestones, and work products that must be accomplished to complete particular project.
- Every process model consists of various tasks sets. Using these tasks sets the software team define, develop or ultimately support computer software.
- There is no single task that is appropriate for all the projects but for developing

large, complex projects the set of tasks are required. Hence every effective software process should define a collection of task sets depending upon the type of the project.

- Using tasks sets the high quality software can be developed and any unnecessary work can be avoided during software development.
- The number of tasks sets will vary depending upon the type of the project. Various **types of projects** are enlisted below -
  1. **Concept development project** : These are the projects in which new business ideas or the applications based on new technologies are to be developed.
  2. **New application development project** : These projects are developed for satisfying a specific customer need.
  3. **Application upgradation project** : These are kind of projects in which existing software application needs a major change. This change can be for performance improvement, or modifications within the modules and interfaces.
  4. **Application maintenance project** : These are kind of projects that correct, adapt or extend the existing software applications.
  5. **Reengineering projects** : These are the projects in which the legacy systems are rebuilt partly or completely.
- Various **factors that influence** the tasks sets are -
  1. Size of project
  2. Project development staff
  3. Number of user of that project
  4. Application longetivity
  5. Complexity of application
  6. Performance constraints
  7. Use of technologies
- **Task set example** : Consider the concept development type of the project. Various tasks sets in this type of project are -
  - **Defining scope** : This task is for defining the scope, goal or objective of the project.
  - **Planning** : It includes the estimate for schedule, cost and people for completing the desired concept.
  - **Evaluation of technology risks** : It evaluates the risk associated with the

technology used in the project.

- **Concept implementation** : It includes the concept representation in the same manner as expected by the end user.

Example 5.13.1 *How to compute a task set selector for the project ? Explain with suitable illustration.* **AU : IT, Dec.-07, Marks 8**

Solution : Before computing the task set selector value let us list out the 'adaption criteria' used for software process in software project.

1. Size of project.
2. Total number of skilled users.
3. Complexity and difficult level of mission.
4. How long the application will exist ?
5. Requirements stability.
6. Customer-developer communication.
7. Project staff.
8. Maturity of technology used.
9. Performance constraints.
10. Re-engineering factor.
11. Embedded and non embedded characteristics.

Now following are the steps that must be followed for computing task set selector.

Step 1 : Prepare a table in which all the above mentioned adaption criteria must be entered. Assign appropriate grade to each adaption criteria. These grades range from 1 to 5.

Step 2 : Then assign corresponding weighting factor to each adaption criteria. This weighting factor ranges from 0.8 to 1.2.

Step 3 : Then there are some entry point multiplier based on -

i) Concept development ii) New development

iii) Enhanced application iv) Maintenance project

v) Re-engineering project

These values can be 0 or 1 indicating relevance of adaption criteria.

Step 4 : Computer product value for each production criteria using -

Grade × Weighting factor × Entry point multiplier

Step 5 : Take average of all 'product' values. This will indicate task set selector.

For example

Adaption criteria	Grade	Weighting factor	Entry point multiplier					Product
			Conc. dev.	New dev.	Enha.	Maint.	Re-engg.	
Size of project	3	1.1		1				3.3
Total number of skilled users	3	1.1		1				3.3
Complexity of mission	4	1.2		1				4.8
How long application will exist	3	0.9		1				2.7
Requirement stability	2	1.1		1				2.2
Customer-developer comm.	2	1.0		1				2.0
Project staff	3	1.2		1				3.6
Maturity of technology	2	1.0		0				2.0
Performance constraint	3	0.8		1				2.4
Reengineering factor	0	1.1		0				0
Embedded and non-embedded characteristics	2	1.1		1				2.2
								2.6

Task set selector value

5.13.3 Task Network

- The task is a small unit of work.
- The task network or an activity network is a graphical representation, with :

  Nodes corresponding to activities.

  Tasks or activities are linked if there is a dependency between them.

  The task network for the product development is as shown in Fig. 5.13.3.

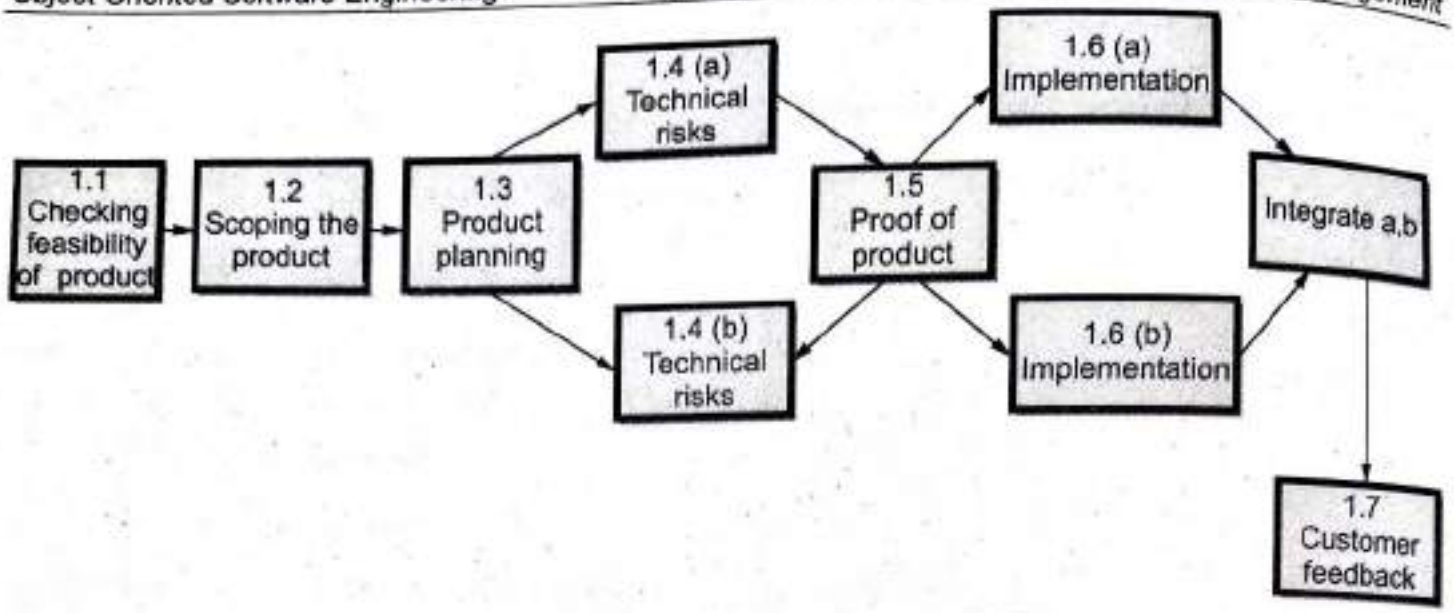


Fig. 5.13.3 Task network

- The task network definition helps project manager to understand the project work breakdown structure.
- The project manager should be aware of interdependencies among various tasks. It should be aware of all those tasks which lie on the critical path.

5.13.4 Time Line Chart

- In software project scheduling the timeline chart is created. The purpose of timeline chart is to emphasize the scope of individual task. Hence set of tasks are given as input to the time line chart.
- The time line chart is also called as Gant chart.
- The time line chart can be developed for entire project or it can be developed for individual functions.
- In time line chart
  1) All the tasks are listed at the leftmost column.
  2) The horizontal bars indicate the time required by the corresponding task.
  3) When multiple horizontal bars occur at the same time on the calendar, then that means concurrency can be applied for performing the tasks.
  4) The diamonds indicate the milestones.
- In most of the projects, after generation of time line chart the project tables are prepared. In project tables all the tasks are listed along with actual start and end dates and related information.

Example

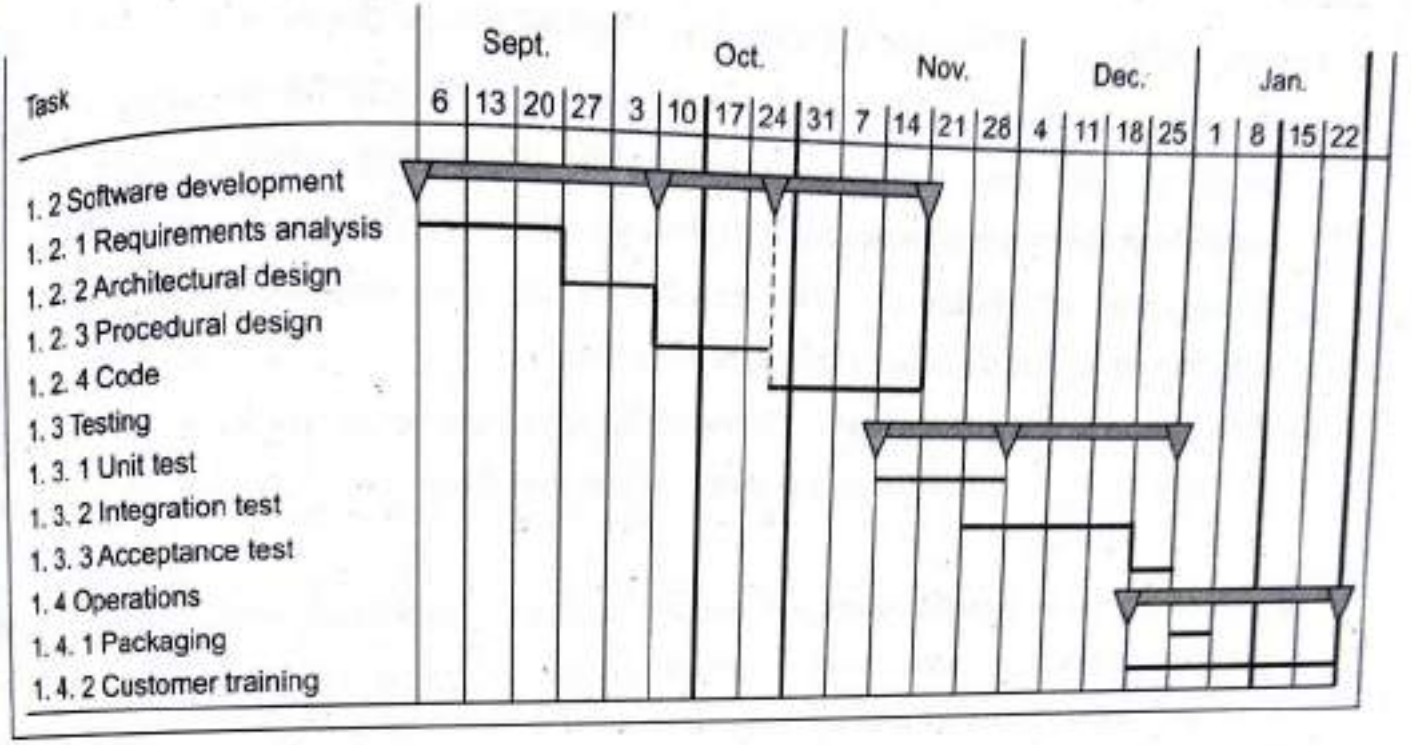


Fig. 5.13.4 Time line chart

Project table

Tasks	Planned start	Actual start	Planned end	Actual end	Effort assignment
Requirement analysis	6th Sept'05	6th Sept'05	20th Sept'05	22nd Sept'05	Jayashree, Padma, Lucky
Architectural design	27th Sept'05	27th Sept'05	3rd Oct'05	7th Oct'05	Trupti, Varsha
Procedural design	10th Oct'05	12th Oct'05	24th Oct'05	25th Oct'05	Varsha, Sachin, Devendra
⋮	⋮	⋮	⋮	⋮	⋮
Customer training	1st Jan'06	4th Jan'06	22nd Jan'06	25th Jan'06	Smita, Yogita

Table 5.13.1 Project table

5.13.5 Tracking Schedule

Project schedule is the most important factor for software project manager. It is the duty of project manager to decide the project schedule and track the schedule.

Tracking the schedule means determine the tasks and milestones in the project as it proceeds.

Following are the ways by which tracking of schedule can be done -

1. **Periodic meeting** - The regular periodic meetings should be conducted. In these meeting each team member should report the progress of the project. Also various problems encountered in project should be openly discussed.
2. **Evaluation of reviews** - The results of all the reviews during software engineering process should be evaluated.
3. **Milestone accomplishment** - To track the schedule of the project it is necessary to determine whether formal project milestones have been accomplished on the scheduled date.
4. **Meeting with practitioners** - Conduct informal meetings with practioners to obtain subjective assessment of progress to date. Also the problems arising in project should be discussed.
5. **Earned value analysis** - The project can be assessed quantitatively using earned value analysis.

Thus for tracking the schedule of the project the project manager should be an experienced person. In fact project manager is the only responsible person who is controlling the software project. When some problems occur in the project then addition resources may be demanded, skilled and experienced staff may be employed or project schedule can be redefined. For handling the severe deadlines, project manager uses a technique of **time boxing.** In this technique each it is understood that the complete product cannot be delivered on given time. Part by part i.e. in the series of increments the product can be delivered to the customer. The project manager uses time box technique means he is associating each task with a box. That means each task is put in a "time box" and within that time frame each task must be completed. When the current task reaches to boundary of its time box, then the next task must be started (even if current task is remaining incomplete). Some researchers had argued upon - leaving the task incomplete when current task reaches to the boundry but for this argument the counterpart is that even if the task is remaining incomplete it reaches to almost completion stage and remaining part of it can be completed in the next successive increment.

5.13.6 Earned Value Analysis

- The Earned Value Analysis (EVA) takes into consideration the project context for planned and actual expenditure.

- This analysis is made to find out project scope, schedule and resource characteristics.
- The EVA acts as a measure for software project progress.
- Various measures are determined during EVA. These measures are -

1) **Planned Value (PV) :** It denotes the planned cost of the work. The planned value is developed by first determining all of the work, that must be accomplished for successful project result.
2) **Actual Cost (AC) :** It represents the actual amount that the business has to expend on the project.

$$AC = \sum \text{Efforts expended on work task have been completed by time t}$$

3) **Earned Value (EV) :** It is project manager's estimate of the amount of originally budgeted work completed.
4) **Budget At Completion (BAC) :** It represents total budget for the project.
5) **Schedule variance (SV) :** It indicates the status of the schedule. It represents whether work is ahead or behind the plan. If SV is negative the project is behind schedule, if it is positive then project is ahead of schedule. If SV is equal to zero, then project is on schedule.
6) **Cost Variance (CV) :** It is the difference between earned value and actual cost.

   If CV = 0 then project is on budget.

   If CV < 0 then project is over budget

   If CV > 0 then project is under budget
7) **Schedule Performance Index (SPI) :** It is a measure of schedule efficiency on a project. If SPI = 1.0, project is on schedule.

   If SPI is greater than 1 then project is ahead of the schedule.

   IF SPI is less than 1 then project is getting delayed and it is behind the schedule.
8) **Cost Performance Index (CPI) :** It is a measure of cost efficiency on a project. The value 1.0 represents that project is within the given budget.

   If CPI < 1.0 then that means project is over budgeted.

   If CPI > 1.0 then that means project is under budgeted and we require most cost to accomplish the project.

Formula used during (EVA) :

PV = Planned Completion (%) * Budget At Completion (BAC)

EV = Actual Completion (%) * BAC

$$SV = EV - PV$$

$$CV = EV - AC$$

$$SPI = EV/PV$$

$$CPI = EV/AC$$

Example 5.13.2 *Mr. Koushan is the project manager on a project to build a new cricket stadium in Mumbai, India. After six months of work; the project is 27 % complete. At the start of the project, Koushan estimated that it would cost $ 50,000.000, What is the Earned value ?*

AU : Dec.-15, Marks 2

Solution : The formula for Earned value is -

$$\text{Earned value} = \% \text{ of work} \times \text{Budget}$$

$$= 27\,\% \times 50000$$

$$= \frac{27}{100} \times 50000 = 13,500$$

∴ The earned value is **$ 13,500.**

Example 5.13.3 *Suppose you have a budget cost of a project as ₹9,00,000. The project is to be completed in 9 months. After a month, you have completed 10 percent of the project at a total expense of ₹ 1,00,000. The planned completion should have been 15 percent. You need to determine whether the project is on-time and on-budget ? Use Earned Value analysis approach and interpret.*

AU : Dec.-16, Marks 8

Solution : The abbreviations that we will use are as follows

EV = Earned Value

PV = Planned Value

BAC = Budget At Completion

AC = Actual Cost

Following formula will be used.

PV = Planned Completion(%) * BAC

EV = Actual Completion(%) * BAC

CPI = EV/AC

SPI = EV/PV

Given that :

BAC = ₹ 9,00,000

AC = ₹ 1,00,000

1) Planned Value (PV) = Planned Completion (%) * BAC

= 15 % * 900000

$$= \frac{15}{100} * 900000$$

∴ **PV = ₹ 135000**

2) Earned Value (EV) = Actual Completion (%) * BAC

= 10 % * 900000

$$= \frac{10}{100} * 900000$$

∴ **EV = ₹ 90000**

3) Cost Performance Index (CPI) $= \frac{EV}{AC}$

$$= \frac{900000}{100000}$$

CPI = 0.90

As CPI < 1 means project is **over budget.**

4) Schedule Performance Index (SPI) $= \frac{EV}{PV}$

$$= \frac{900000}{135000}$$

SPI = 0.67

As SPI < 1 means project is **behind schedule.**

This indicates project is in real trouble.

Example 5.13.4 *Suppose you are managing a software development project. The project is expected to be completed 8 months at a cost ₹10000 per month. After 2 months, you realize that project is 30 percent completed at a cost of ₹40,000. You need to determine whether the project is on-time and on-budget after 2 months.*

Solution : Given that

- Budget at completion (BAC) = 10,000 * 8

  = ₹ 80,000
- Actual Cost (AC) = ₹ 40,000
- Planned Completion = 2/8 = 25 %

- Actual Completion = 30 %

We will compute Planned Value (PV) and Earned Value (EV)

i) Planned Value (PV) = Planned Completion (%) * BAC

$$= 25\% * 80{,}000$$

$$= \frac{25}{100} * 80{,}000$$

PV = ₹ 20,000

ii) Earned Value (EV) = Actual Completion (%) * BAC

$$= 30\% * 80{,}000$$

$$= \frac{30}{100} * 80{,}000$$

EV = ₹ 24,000

Now we will compute the Cost Performance Index (CPI) and Schedule Performance Index (SPI).

iii) Cost Performance Index (CPI) $= \frac{EV}{AC}$

$$= \frac{24{,}000}{40{,}000}$$

CPI = 0.6

iv) Schedule Performance Index (SPI) $= \frac{EV}{PV}$

$$= \frac{24{,}000}{20{,}000}$$

SPI = 1.2

The CPI < 1, then that means the project is over budget. For every ₹ 1 spent we are getting 60 paisa worth performance.

The SPI > 1, then that means the project is ahead of the schedule. That means project can be delivered before the given schedule.

Example 5.13.5 *Given the following project plan of tables Table 5.13.2 and Table 5.13.3 :*

Table 5.13.2

ID	Task	Immediate predecessor (*)	Expected duration (days)	Budget ($)
A	Meet with client		5	500
B	Write SW	A	20	10000
C	Debug SW	B	5	1500
D	Prepare draft manual	B	5	1000
E	Meet with clients	D	5	1000
F	Test SW	C, E	20	2000
G	Make modifications	F	10	8000
H	Finalized manual	G	10	5000
I	Advertise	C, E	20	8000

(*) all dependencies are assumed to be FS-Finish to Start and the following progress status :

Table 5.13.3

ID	Task	Status	Actual start (days)	Actual duration (days)	Actual costs ($)
A	Meet with client	100 %			1500
B	Write SW	100 %	+ 5 days	+ 10 days	9000
C	Debug SW	100 %	+ 5 days	+ 5 days	2500
D	Prepare draft manual	100 %	As per other delays		1000
E	Meet with clients	100 %	As per other delays		1000
F	Test SW	100 %	As per other delays		750
G	Make modifications	0 %	As per other delays		0
H	Finalize manual	0 %	As per other delays		0
I	Advertise	10 %	+ 5 on top of other delays		1000

Perform an analysis of the project status at week 13, using EVA. Use the CPI and SPI to determine project efficiency. Explain the process involved. **AU : May-17, Marks 6**

Solution : To perform analysis of the project we will compute PV, AC and EV.

- **PV** is the sum of planned cost. It is computed by determining for each reporting period, the cost associated with each activity and by summing and cumulating them over time. The cumulative cost can be computed by summing planned expenditure week by week.

		W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	Total
A	Meet with client	500													500
B	Write SW		2500	2500	2500	2500									10000
C	Debug SW						1500								1500
D	Prepare draft manual						1000								1000
E	Meet with clients							1000							1000
F	Test SW								500	500	500	500			2000
G	Make modifications												4000	4000	8000
H	Finalize manual														0
I	Advertise								2000	2000	2000	2000			8000
	Total	500	2500	2500	2500	2500	2500	1000	2500	2500	2500	2500	4000	4000	
	Planned Value	500	3000	5500	8000	10500	13000	14000	16500	19000	21500	24000	28000	32000	

Note that each week is of 5 days.

∴ PV = 32000

- AC is the sum of the actual cost. The summing of actual cost is as shown below.

		W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	T.tal
A	Meet with client	1500													500
B	Write SW			1500	1500	1500	1500	1500	1500						9000
C	Debug SW									1250	1250				2500
D	Prepare dragt manual									1000					1000
E	Meet with clients										1000				1000
F	Test SW											250	250	250	750
G	Make modifications														0
H	Finalize manual														0
Z	Advertise												500	500	1000
	Total	1500	0	1500	1500	1500	1500	1500	1500	2250	2550	250	750	750	
	Planned Value	1500	1500	3000	4500	6000	7500	9000	10500	12750	15000	15250	16000	16750	

∴ AC = 16750

- EV is the earned value which is sum of the planned cost on actual schedule. The actual result is as shown below

		W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	Total
A	Meet with client	500													500
B	Write SW			5000	0	0	0	0	5000						10000
C	Debug SW									750	750				1500

D	Prepare dragt manual									1000					1000
E	Meet with clients										1000				1000
F	Test SW											1000	0	0	1000
G	Make modifications														0
H	Finalize manual														0
I	Advertise												4000	0	4000
	Total	500	0	5000	0	0	0	0	5000	1750	1750	1000	4000	0	
	Planned Value	500	500	5500	5500	5500	5500	5500	10500	12250	14000	15000	19000	19000	

$\therefore$ EV = 19000

- From above data, at W13 the following things are observed

i) PV > AC indicates that project is under budget.

ii) EV < PV means project is late.

- $$CPI = \frac{EV}{AC} = \frac{19000}{16750}$$
$$\approx 1.13$$

The CPI is cost performance index which is > 1. It indicates that project is within the given budget.

- $$SPI = \frac{EV}{PV} = \frac{19000}{32000}$$
$$= 0.6$$

The SPI is schedule performance which is < 1. It indicates that the project is late.

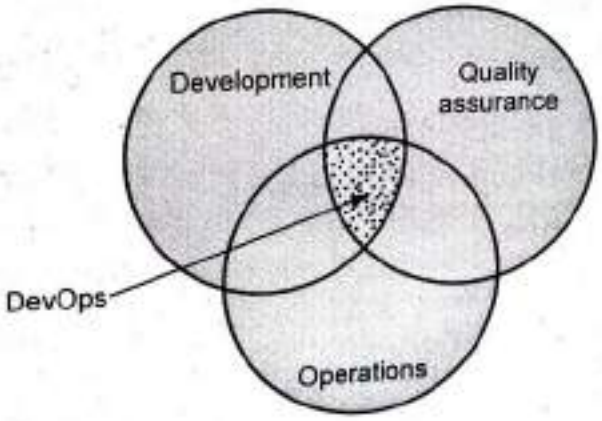
Review Questions

1. *How to track the schedule ? Give an illustration.* AU : Dec.-06,07, Marks 8
2. *Write short notes on the following.*
   i) Project scheduling.
   ii) Project timeline chart and task network.
3. *Discuss Putnam resources allocation model. Derive the time and effort equations.* AU : May-15, Marks (8+8)
4. *Discuss about any one project scheduling technique.* AU : May-16, Marks 12 AU : May-22, Marks 6

5.14 DevOps

Definition : DevOps is a practice in which development and operation engineers participate together in the entire lifecycle activities of system development from design and implementation to product support.

- The term DevOps is derived from "Software **DEV**elopment " and "information technology **O**Peration**S**".
- DevOps promotes a set of processes and methods from the three department **Development**, IT **operations** and **Quality assurance** that communicate and collaborate together for development of software system.



5.14.1 Why DevOps ?

- DevOps enhances the organization's performance, improves the productivity and efficiency of development and operations teams.
- Bringing the two teams together centralizes the responsibility on the entire team and not specific individuals working.
- DevOps is more than just a tool or a process change. It inherently requires an organizational culture shift.

- This cultural change is especially difficult, because of the conflicting nature of departmental roles :
  1. **Operations** - seeks organizational stability;
  2. **Developers** - seek change;
  3. **Testers** - seek risk reduction.
- Adoption of DevOps is driven by various factors. These factors are -
  1. Demand for an increased rate of production releases - from application and business unit stakeholders.
  2. Increased usage of data center automation and configuration management tools.
  3. Use of agile and other development processes and methods.
  4. Increased focus on test automation and continuous integration methods.
  5. Wide availability of virtualized and cloud infrastructure.

5.14.2 Motivation

The Goals of DevOps are as follows -

1. To make simple processes increasing programmable and dynamic.
2. Fast delivery of product.
3. Lower failure rate of new releases.
4. Shortened lead time between fixes.
5. Faster mean time to recovery.
6. Increases net profit of organization.
7. To standardize the development environment.
8. To reduce work in progress.
9. To reduce operating expenses.
10. To set up the automated environment.

5.14.3 Benefits

Various benefits of DevOps are -

➢ **Technical Benefits**

1. Continuous software delivery is possible.
2. There is less complexity in managing the project.

- The problems in the project gets resolved faster.

Cultural benefits

1. The productivity of teams get increased.
2. There is higher employee engagement.
3. There arise greater professional development opportunities.

Business benefits

1. The faster delivery of the product is possible.
2. The operating environment becomes stable.
3. Communication and collaboration are improved among the team members and customers.
4. More time is available for innovation rather than fixing and maintaining.

5.14.4 Agility and DevOps

- Basically Agile and DevOps are similar. But there lies some differences.
- DevOps brings more flexibility than Agile. With Continuous Integration (CI) and Continuous Delivery (CD) , the release of software products is made often and it is ensured that these releases actually work and meet the customer needs.
- Thus in DevOps there are an increased number of releases.
- One goal of DevOps is to establish an environment where releasing more reliable applications, faster and more frequently, can occur. This actually brings the **continuous delivery approach**.
- DevOps is not a separate concept but a mere extension of Agile to include operations as well to collaborate different agile teams together and work as ONE team with an objective to deliver software fully to the customer.

Sr.No.	Agile	DevOps
1.	The idea in Agile is to develop software in small iterations and be thus able to adapt to the changing customer needs.	DevOps is to deliver technology to business units in a timely fashion and ensures the technology runs without interruption or disruption.
2.	It adopts a rapid development approach.	This is not a rapid development approach.
3.	The focus of agile development is merely on software development and release.	The focus of DevOps is not only on software development and its release but on it safest deployment in the working environment.

4.	In agile development, every team member has a skill of design, development and coding. Any available team member should be able to do what's required for progress.	DevOps, on the other hand, assumes there will be development teams and operational teams, the two will be separate. These teams can communicate between themselves on a frequent and regular basis.
5.	Communication in agile development is informal and in the form of daily meetings.	In DevOps communication, specifications, and documents are involved and it is formal. It does not occur on a daily basis.
6.	The team is small in nature.	Large team sizes and multiple teams are required in DevOps.
7.	Agile is about software development.	DevOps is about software development and management.
8.	Documentation is not very important in Agile development.	Documentation is very important in DevOps.
9.	Agile development teams may choose to use certain automation tools. However, there are no specific tools required for an agile team.	DevOps absolutely depends on automated development to make everything happen smoothly and reliably. Certain tools are an integral part of DevOps.
10.	Agile is less flexible.	DevOps is more flexible.
11.	Agile has limited scope.	DevOps has a broader scope.

5.14.5 Deployment Pipeline

- A **Pipeline** is a set of automated processes that allow developers and DevOps professionals to reliably and efficiently compile, build and deploy their code to their production computing platform.
- Various components of a pipeline are -
  1) Build Automation
  2) Test Automation
  3) Deploy Automation
- Deployment pipeline is an important concept and practice in DevOps that involves automating and Streaming the process of delivering the software application from development to production.

- There are two important concepts used in the Deployment pipeline -
  1. Continuous Integration(CI) and
  2. Continuous Delivery(CD)/Continuous Deployment.

1. Continuous Integration :

- The continuous integration in DevOps is a practice where developers regularly merge their code changes into a central repository or a database after which automated **builds** and **tests** are run.
- Continuous Integration (CI) is the practice of **automating the integration of code changes** from multiple developers or testers into a single software project.
- Automated tools are used to assert the new code's correctness before integration.
- Continuous integration serves as a prerequisite for the testing, deployment and release stages of continuous delivery.
- The main benefit of performing continuous integration regularly and testing each integration is that we can detect errors more quickly and locate them easily.

2. Continuous Delivery or Continuous Deployment :

- Continuous Delivery means a developer's changes to an application are automatically bug-tested and uploaded to a repository.
- Continuous Deployment (CD) refers to the final stage in the pipeline that refers to the automatic releasing of any developer changes from the repository to the production.

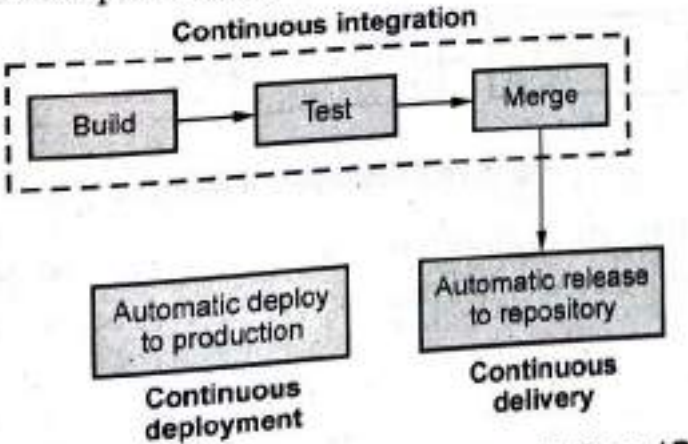


Fig. 5.14.1 Continuous Integration (CI) and Continuous Delivery / Deployment (CD)

- The key components of the deployment pipeline are -
  - **Source Code Management :** This is the first step of the deployment pipeline in which source code is stored in a version control system like Git and GitHub. Developers commit the changes (upload final changes) to this repository and pipeline is triggered when there are new commits.
  - **Build :** In this process, the code is built into executable entities. During the build process, the code is compiled, dependencies are packaged and binaries are created.
  - **Automated Testing :** The pipeline runs a suite of automated tests in order to test the software system. It includes unit testing, integration testing and performance testing.
  - **Deployment :** Once the code passes testing it is deployed to a staging or testing environment that closely resembles the production environment. This allows for further testing and validation in a controlled setting.
  - **User Acceptance Testing (UAT) :** In this stage, the software is tested by a group of users or stakeholders to ensure it meets their expectations.
  - **Final Deployment :** If all tests and checks are successful, the software is deployed to the production environment. This can be done manually, automatically or with a combination of both.
  - **Monitoring and Feedback :** The application is continuously monitored, if any issues are raised then these are fixed immediately. Feedback from the production environment is used to inform future development and improvements.
  - **Documentation :** Finally the comprehensive documents and reports are prepared.

Fig. 5.14.2 Deployment pipeline

Deployment pipelines are typically designed as automated services. This approach reduces the manual errors, accelerates the delivery process of the software system.

5.14.6 Overall Architecture

- DevOps is a practice of operations and development.

There are different phases of DevOps architecture -

1) **Plan :** In this phase, all the requirements of the project are gathered. The schedule and cost of the project is estimated approximately.
2) **Code :** In this phase the code is written as per the requirements. Entire project is divided into smaller units. Each unit can be coded as a module.
3) **Build :** In this phase, the building of all the units is done using tools such as Maven, Gradle to submit the code to a common code source.

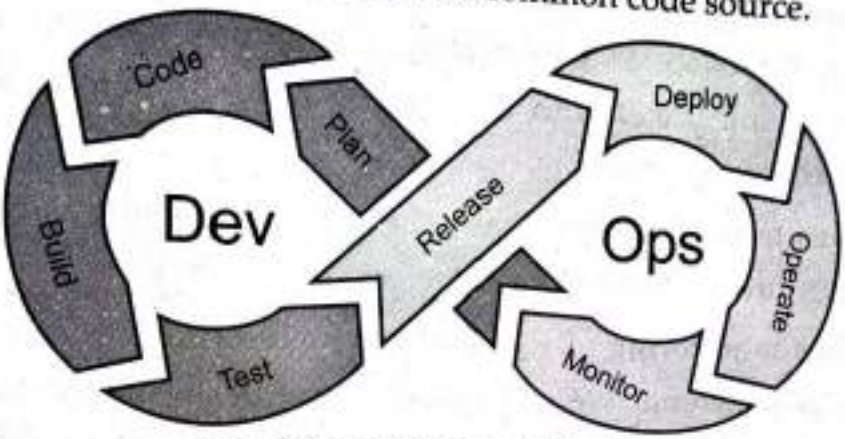


Fig. 5.14.3 DevOps architecture

4) **Test :** At this stage, all the units are tested to find if there exists any bug in the code. The testing can be done using tools like Selenium, JUnit, PYtest. Some important testing techniques such as acceptability testing, safety testing, integration checking, performance testing are carried out.
5) **Integrate** : In this phase, a new feature is added to the existing code and testing is performed. Continuous Development is achieved only because of continuous integration and testing.
6) **Deploy :** In this stage, the code is deployed in the client's environment. Some of the examples of the tools used for Deployment are AWS, Docker.
7) **Operate :** At this stage, the version can be utilized by the users. Operations are performed on the code if required. Some of the examples of the tools used are Kubernetes, open shift.
8) **Monitor :** At this stage, the monitoring of the version at the client's workplace is done. During this phase, developers collect data, monitor each function and spot

errors like low memory or server connection are broken. The DevOps workflow is observed at this level depending on data gathered from consumer behavior, application efficiency and other sources. Some of the examples of the tools used are Nagios, elastic stack for monitoring.

5.14.7 DevOps Lifecycle

The DevOps lifecycle phases are as follows -

1) **Continuous development :** In this phase, the planning and coding of software is done. Version control mechanism is used during this phase.
2) **Continuous integration :** In this phase, developers are required to commit changes in the source code frequently. The code supporting new functionality is continuously integrated with the existing code. Therefore, there is continuous development of software.
3) **Continuous testing :** In this phase, the software is continuously tested for bugs. Many times automation testing is preferred.
4) **Continuous monitoring :** By continuous monitoring, we can get notified before anything goes wrong. We can gather many performance measures, including CPU and memory utilization, network traffic, application response times, error rates and others.
5) **Continuous feedback :** In this DevOps stage, the software automatically sends out information about performance and issues experienced by the end-user. It's also an opportunity for customers to share their experiences and provide feedback.
6) **Continuous deployment :** In this phase, the code is deployed to the production servers. Also, it is essential to ensure that the code is correctly used on all the servers. The deployment process takes place continuously in this DevOps life cycle phase.
7) **Continuous operations :** It is the last phase which involves automating the application's release and all these updates that help you keep cycles short and give developers more time to focus on developing.

5.14.8 Tools

Various tools used in DevOps are as follows -

1. **Nagios :** It is a monitoring solution that gives new features and a modern user experience.

2. **ELK Stack :** This tool is used for collecting logs from all services, applications, networks, tools, servers and more in an environment into a single, centralized location for processing and analysis.
3. **Docker :** It eases configuration management and control issues.
4. **Jenkins :** Jenkins is a top tool for DevOps engineers who want to monitor executions of repeated jobs.
5. **Puppet :** It handles configuration management and software while making rapid, repeatable changes in it.
6. **Ansible :** Ansible is a configuration management tool or DevOps tool that is similar to Puppet and Chef.
7. **God :** It is a monitoring tool used in DevOps.
8. **Monit :** Monit has everything DevOps engineers need for system monitoring and error recovery.
9. **Consul.io :** This tool is used for service discovery and configuration management activities.
10. **Loggly :** This tool is used for log management in DevOps.

Review Questions

1. *What is DevOps ? What are the benefits of DevOps ?*
2. *Compare agility with DevOps.*
3. *Explain in detail the deployment pipeline.*
4. *Explain the overall architecture of DevOps.*
5. *Explain DevOps lifecycle.*
6. *What are different tools used in DevOps ?*

5.15 Cloud as a Platform

- "Cloud as a platform" is a concept that refers to using cloud computing infrastructure and services as the foundation for developing, deploying and running applications and services.
- Cloud computing is a term that refers to storing and accessing data over the internet. It doesn't store any data on the hard disk of our personal computer. In cloud computing, we can access data from a remote server.
- It includes services for storage, databases, analytics, networking, mobile, development tools and enterprise applications.

- AWS manages and maintains hardware and infrastructure, saving organizations and individuals the cost and complexity of purchasing and running resources on site. These resources may be accessed for free or on a **pay-per-use** basis.
- Popularly used cloud platforms are - **Amazon Web Services (AWS), Microsoft Azure** and **Google Cloud Platform (GCP).**

➢ **Benefits of using Cloud as a Platform**

1) Flexibility :

- The cloud platform always allows to use the operating system, programming languages and web application platforms that user is comfortable with.
- Flexibility means that migrating legacy applications to the cloud should be easy.
- Instead of re-writing the applications to adopt new technologies, we just need to move the applications to the cloud and tap into advanced computing capabilities.

2) Cost Effective :

- Instead of purchasing and creating our own expensive servers, we can use cloud services where we need to pay only for the tools and services that we use.
- Cloud platform offers a pay-as-you-go pricing method, which means that we only pay for the services that are needed and have been used for a period of time.

3) Scalable and Elastic :

- Cloud platform is scalable because its Auto Scaling service automatically increases the capacity of constrained resources as per requirements so that the application is always available.
- Elasticity is one of the advantages. If we use fewer resources and don't need the rest of them, then cloud platform itself shrinks the resources to fit our requirements. In short, upsizing and downsizing of resources is possible with AWS.

4) Secure :

- Cloud computing maintains confidentiality, integrity and availability of the user's data.
- Each service provided by the cloud is secure.
- Personal and business data can be encrypted to maintain data privacy.

5) **High Performance :**

- High-performance computing is the ability to process massive amounts of data at high speed.
- Cloud computing offers a high-performance computing service so that the companies need not worry about the speed.

5.15.1 Operations

- It provides different services of the cloud such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and packaged Software as a Service (SaaS).
  - **IaaS :** The infrastructure as a service means delivering computing infrastructure on demand. Under this service, the user purchases the cloud infrastructure including servers, networks, operating systems and storage using virtualization technology. These services are highly scalable. IaaS is used by network architects. Examples of cloud services : AWS and Microsoft Azure.
  - **PaaS :** The Platform as a Service means it is a service where a third-party provider provides both hardware and software tools to the clients. It provides elastic scaling of your application which allows developers to build applications and services over the internet and the deployment models include public, private and hybrid. PaaS is used by developers. Examples of cloud services - are Facebook and Google Search Engine.
  - **SaaS :** Software as a Service model that hosts software to make it available to clients. It is used by the end users. Examples of Cloud services : Google Apps.

Difference between IaaS, PaaS and SaaS

Sr. No.	IaaS	PaaS	SaaS
1.	IaaS is an acronym for Infrastructure as a Service.	PaaS is an acronym for Platform as a Service.	SaaS is an acronym for Software as a Service.
2.	IaaS allows suppliers to share their physical servers, virtual machines or virtual storage to the users or customers.	It provides an environment to allow users to create their own web applications. PaaS provides a runtime environment for applications and the development of applications.	It allows the use of software applications as a service to end users. It allows users to access specific applications online.

3.	It is most expensive.	It has mid-level costs.	It is the cheapest.
4.	The network architects primarily use the IaaS.	Developers mainly make use of PaaS.	End-users generally use SaaS.
5.	Services available are - Virtual machines, operating systems, network, storage, backup services.	PaaS provides all the facilities required to build and develop web applications.	SaaS model provides services to deploy applications online.
6.	For example - AWS EC2.	For example - Microsoft Azure.	For example - Salesforce, DropBox.

Review Questions

1. *What is cloud computing ? Explain the benefits of using cloud computing.*
2. *Differentiate between various services offered by cloud computing platform.*

5.16 Two Marks Questions with Answers

Q.1 Write short notes on empirical estimation models. AU : IT, May-04, 05

Ans. : The empirical estimation model uses empirically (experimentally) derived formulae to predict effort as a function of lines of code or function point. In empirical estimation model, the empirical values of LOC of FP are put. The empirical model supports a limited number of software project. Typical estimation model is derived using regression analysis on data collected from past software projects.

Examples of such models calculate effort as,

1. Walston-Felix model

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

2. Bohem simple model

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

Q.2 What is the standardization for the software metrics ? AU : IT, Dec.-04

Ans. : The standardization of software metrics indicates the attributes of effective software metrics. The software metrics should follow following standards.

1. Simple and computable - The software metrics should be easy to learn and derive.
2. Empirically and intuitively persuasive - The software metrics should satisfy the intuitive notions about the product attribute.
3. Consistent and objective - The software metrics should give the unambiguous results.

4. Consistent in its use of units and dimensions - The units and dimensions used in the software metrics should be consistent.
5. Programming language independent - The software metrics should be based on analysis model, design model and structure of the program. But it should be independent of syntax or semantics of the program.
6. Effective mechanism for high quality feedback - The software metric should provide information to software engineer whether the software product being developed is of quality or not.

Q.3 What are project indicators and how do they help a project manager ?

AU : CSE, May-05, 06

Ans. : Project Indicators mean combination of metrics that provides insight into the software process, project or product. An indicator is a tool that helps you and your organization know how far your project is from achieving your goals and whether you are headed in the right direction.

Q.4 Distinguish between direct and indirect measures of metrics. AU : CSE, Dec.-04

Ans. : Direct metrics is directly measurable attribute. For example lines of code, execution speed, size of memory, some defects.

Indirect metrics - Indirect metrics are the aspects that are not immediately measurable. For example functionality, reliability, maintainability.

Q.5 Differentiate between size oriented and function oriented metrics.

AU : IT, Dec.-06, May-13

Ans. :

Sr. No.	Size oriented metrics (LOC based)	Function oriented metrics
1.	Size oriented software metrics is by considering the size of the software that has been produced.	Function oriented metrics use a measure of functionality delivered by the software.
2.	For a size oriented metric the software organization maintains simple records in tabular form. The typical table entries are : Project name, LOC, Effort, Pages of documents, errors, defects, total number of people working on project.	Most widely used function oriented metric is the Function Point (FP) computation of the function point is based on characteristics of software's information domain and complexity.

Q.6 Define basic equation for the effort estimation models. AU : IT, May-07

Ans. : The basic equation for the effort estimation models are -

$$E = A + B \times (eV)^C$$

Where E = Effort in person-month

A, B and C = are empirically derived constants.

eV = Estimation variable either LOC or FP based method.

Q.7 List down few process and product metrics. AU : CSE, Dec.-06

Ans. : Product metrics are :

1. Size metric - This metric is used for measuring the size of the software.
   a) LOC based meritc - This metric makes use of lines of code to measure the software.
   b) FP based metric - This metric makes use of functional points to measure the software.
2. Complexity metric - A software module can be described by a control flow graph.
   a) Cyclomatic complexity.
   b) McCabe complexity metric.
3. Quality metric -
   a) Defects
   b) Reliability metric
   c) Maintainability metric

- Process metrics are based on following factors :
  a) Application of methods and tools.
  b) Use of standards for the system.
  c) Effectiveness of management of system.
  d) Performance of development of system.

Q.8 Define software measure. AU : CSE, May-08

Ans. : Software measure is a numeric value for a attribute of a software product or process. There are two types of software measures-direct measure and indirect measure. The direct measure refers to immediately measurable attributes. For example lines of code. The indirect measure refers to the aspects that are not immediately quantifiable or measurable. For example functionality of the program.

Q.9 How to measure the function point (FP) ? *(Refer section 5.8.5)* AU : IT, Dec.-07

Q.10 List out the different approaches to size of the software. AU : CSE, Dec.-08

Ans. : The two approaches used for estimating the size of the software are -

1. LOC i.e. computing the lines of code.
2. FP i.e. computing function point of the program.

Q.11 Mention difference between organic mode and embedded mode in cocomo model. AU : IT, May-09

Sr. No.	Organic mode	Embedded mode
1.	The small size projects are handled using organic mode.	The large size projects are handled using embedded mode.
2.	The experienced developers develop such projects.	The embedded mode projects are handled by little previous experienced people.
3.	Deadline of the projects under organic mode is not tight.	The embedded mode projects are carried out with tight deadline.
4.	Example - Payroll system	Example - Whether forecasting system.

Q.12 Name the metrics for specifying non-functional requirements. AU : Dec.-11

Ans. : Reliability, response time, storage capacity, usability are the metrics for functional requirements.

Q.13 An organic software occupies 15,000 LOC. How many programmers are needed to complete ? AU : Dec.-12

Ans. : Given that :

System = Organic

Lines of Code = 15KLOC

$$E = ab(KLOC)^{b_b}$$

$$= 2.4(15)^{1.05}$$

$$= 41 \text{ persons-month}$$

$$D = c_b(E)d_b$$

$$= 2.5(41)^{0.38}$$

$$= 10 \text{ months}$$

$$P = 41/10$$

$$P = 4 \text{ persons.}$$

Approximately 4 programmers are required to complete the work.

Q.14 What is scheduling ?

Ans. : Scheduling is an activity that distributes estimated effort across the planned project duration. These efforts are related to software engineering tasks.

AU : May-14

Q.15 What is error tracking ?

Ans. : Error tracking is a process of finding out and correcting the errors that may occur during the software development process at various stages such as software design, coding or documenting.

Q.16 Differentiate between size oriented and function oriented metrics

Ans. :

Sr. No.	Size oriented metrics	Function oriented metrics
1.	Size oriented software metrics is by considering the size of the software that has been produced.	Function oriented metrics use a measure of functionality delivered by the software.
2.	For a size oriented metric the software organization maintains simple records in tabular form. The typical table entries are : Project name, LOC, Effort, Pages of documents, errors, defects, total number of people working on project.	Most widely used function oriented metric is the Function Point (FP) computation of the function point is based on characteristics of software's information domain and complexity.

Q.17 State the advantages and disadvantages in LOC based cost estimation AU : May-15

Ans. : Advantages

1. Artifact of software development can be easily counted.
2. Many existing methods use LOC as a key input.
3. A large body of literature and data based on LOC already exists.

Disadvantages

1. This measure is dependent upon the programming language.
2. This method is well designed but shorter program may get suffered.
3. It does not accommodate non procedural languages.
4. In early stage of development it is difficult to estimate LOC.

Q.18 State the importance of scheduling activity in project management. AU : May-15

Ans. : Scheduling is carried out to assess progress of a software project. In order to build a complex system, many software engineering tasks occur in parallel. The result of work performed during one task may have a certain effect on work to be conducted in another task.

These interdependencies are very difficult to understand without a schedule.

Q.19 What are the issues in measuring the software size using LOC as metric ?

AU : Dec.-15

Ans. : 1) This measure is based on lines of code computation.
2) The LOC is not universally accepted metric.
3) This measure is extremely dependent upon the programming language.
4) This measure does not accommodate non procedural languages.

Q.20 List a few process and project metrics. AU : May-16

Ans. : 1) Size oriented metrics 2) Function oriented metrics
3) Object oriented metrics 4) Software quality metrics

Q.21 Differentiate between size oriented and function oriented metrics.

Ans. :

Sr. No.	Size oriented metrics	Function oriented metrics
1.	Size oriented software metrics is by considering the size of the software that has been produced.	Function oriented metrics use a measure of functionality delivered by the software.
2.	For a size oriented metric the software organization maintains simple records in tabular form. The typical table entries are : Project name, LOC, Effort, Pages of documents, errors, defects, total number of people working on project.	Most widely used function oriented metric is the Function Point (FP) computation of the function point is based on characteristics of software's information domain and complexity.

Q.22 How is productivity and cost are related to function points ? AU : Dec.-16

Ans. : Using Function Point (FP) one can compute both productivity and overall cost of the project.

Productivity = FP/persons-months

Cost = Rupees/FP

Q.23 What are the different types of productivity estimation measures ? AU : May-17

Ans. : i) LOC based Estimation is used for productivity estimation as Productivity = KLOC/person-month.

ii) Function Point based estimation can be used for productivity estimation as

Productivity = FP/person-month.

Q.24 List out the principles of project scheduling. AU : Dec.-17

Ans. : 1) Compartmentalize : Project must be compartmentalized into a number of manageable activities and tasks.

2) Interdependency : The interdependency of each compartmentalized activity or task must be determined.

3) Time allocation : Each task to be scheduled must be allocated some number of work units.

4) Effort validation : The project manager must ensure that no more than the allocated number of people have been scheduled at any given time.

5) Defined responsibilities : Every task that is scheduled should be assigned to a specific team member. AU : May-18

Q.25 What is EVA ?

Ans. : EVA stands for Earned Value Analysis. This analysis is made to find out project scope, schedule and resource characteristics. It acts as a measure for software project progress.

Q.26 List two advantages of COCOMO model. AU : May-19

Ans. : The two advantages of COCOMO model are - 1) It predicts the efforts and schedule of software product based on size of the software.

2) It can work on historical data and is more predictable and accurate.

Q.27 Bring the importance between COCOMO I and II. AU : May-22

Ans. : COCOMO I model is useful in the waterfall models of software development cycle. COCOMO II is useful in non-sequential rapid development and reuse models of software.

The size of software is stated in terms of Lines of Code in case of COCOMO I model. The size of software is stated in terms of object points, function points and lines of code in COCOMO II model.

Q.28 State any two project scheduling techniques. AU : May-22

Ans. :

1) Taks network 2) Tracking schedule 3) Earned value analysis.

Q.29 If team A found 342 errors prior to the release of software and Team B found 182 errors. What additional measures and metrics are needed to find out if the teams have removed the errors effectively ? AU : Dec.-22

Ans. : Following are some additional measures and metrics that are needed to find out if the teams have removed the errors effectively or not -

1) **Defect Density :** Calculate the defect density for each team, which is the number of defects found per unit of code, here the unit of code can be per thousand Lines of Code (LOC). This metric can help you compare the effectiveness of error removal relative to the size of the software.
2) **Defect Removal Efficiency (DRE) :** DRE is a measure of how effectively teams are removing defects. It's calculated as (Total defects found prior to release - Total defects found after release) / Total defects found prior to release. A higher DRE indicates better defect removal effectiveness.