

SUBJECT CODE : CS8494

Strictly as per Revised Syllabus of
ANNA UNIVERSITY
Choice Based Credit System (CBCS)
Semester - V (IT)
Semester - IV (CSE)

SOFTWARE ENGINEERING

Anuradha A. Puntambekar

M.E. (Computer)

Formerly Assistant Professor in
P.E.S. Modern College of Engineering,
Pune.



SOFTWARE ENGINEERING

Subject Code : CS8494

Semester - V (Information Technology)

Semester - IV (Computer Science & Engineering)

First Edition : January 2019

Reprint : June 2019

Second Revised Edition : January 2020

Third Revised Edition : June 2020

© Copyright with Author

All publishing rights (printed and ebook version) reserved with Technical Publications. No part of this book should be reproduced in any form, Electronic, Mechanical, Photocopy or any information storage and retrieval system without prior permission in writing, from Technical Publications, Pune.

Published by :



Amit Residency, Office No.1, 412, Shaniwar Peth,

Pune - 411030, M.S. INDIA, Ph.: +91-020-24495496/97

Email : sales@technicalpublications.org Website : www.technicalpublications.org

Printer :

Yogiraj Printers & Binders

Sr.No. 10/1A,

Ghule Industrial Estate, Nanded Village Road,

Tal. - Haveli, Dist. - Pune - 411041.

ISBN 978-93-332-2105-4



9 789333 221054

AU 17

PREFACE

The importance of **Software Engineering** is well known in various engineering fields. Overwhelming response to my books on various subjects inspired me to write this book. The book is structured to cover the key aspects of the subject **Software Engineering**.

The book uses plain, lucid language to explain fundamentals of this subject. The book provides logical method of explaining various complicated concepts and stepwise methods to explain the important topics. Each chapter is well supported with necessary illustrations, practical examples and solved problems. All the chapters in the book are arranged in a proper sequence that permits each topic to build upon earlier studies. All care has been taken to make students comfortable in understanding the basic concepts of the subject.

Representative questions have been added at the end of each section to help the students in picking important points from that section.

The book not only covers the entire scope of the subject but explains the philosophy of the subject. This makes the understanding of this subject more clear and makes it more interesting. The book will be very useful not only to the students but also to the subject teachers. The students have to omit nothing and possibly have to cover nothing more.

I wish to express my profound thanks to all those who helped in making this book a reality. Much needed moral support and encouragement is provided on numerous occasions by my whole family. I wish to thank the **Publisher** and the entire team of **Technical Publications** who have taken immense pain to get this book in time with quality printing.

Any suggestion for the improvement of the book will be acknowledged and well appreciated.

Author
A. A. Puntambekar

Dedicated to God.

SYLLABUS

Software Engineering - CS8494

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. **(Chapter - 1)**

UNIT - II Requirements Analysis and Specification

Software Requirements: Functional and Non-Functional, User requirements, System requirements, Software Requirements Document - Requirement Engineering Process: Feasibility Studies, Requirements elicitation and analysis, requirements validation, requirements management-Classical analysis: Structured system analysis, Petri Nets - Data Dictionary. **(Chapter - 2)**

UNIT - III Software Design

Design process - Design concepts - Design model - Design heuristic - Architectural design - Architectural styles, Architectural design, Architectural mapping using data flow - User interface design : Interface analysis, Interface design - Component level design : Designing class based components, Traditional components. **(Chapter - 3)**

UNIT - IV Testing and Maintenance

Software testing fundamentals - Internal and external views of Testing - White box testing - Basis path testing - control structure testing-black box testing- Regression Testing -Unit Testing - Integration Testing - Validation Testing - System Testing And Debugging - Software Implementation Techniques: Coding practices-Refactoring-Maintenance and Reengineering-BPR model-Reengineering process model-Reverse and Forward Engineering. **(Chapter - 4)**

UNIT - V Project Management

Software Project Management : Estimation - LOC, FP Based Estimation, Make/Buy Decision COCOMO I & II Model - Project Scheduling - Scheduling, Earned Value Analysis Planning - Project Plan, Planning Process, RFP Risk Management - Identification, Projection - Risk Management - Risk Identification - RMMM Plan - CASE TOOLS **(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 - 34
1.10.1 XP Values	1 - 35
1.10.2 Process.....	1 - 35
1.10.3 Industrial XP.....	1 - 38
Two Marks Questions with Answers	1 - 39

Unit - II

Chapter - 2 Requirements Analysis and Specification (2 - 1) to (2 - 98)

2.1 Software Requirements	2 - 2
2.2 Functional and Non Functional Requirements.....	2 - 3
2.2.1 Functional Requirements	2 - 3
2.2.1.1 Problems Associated with Requirements	2 - 3
2.2.2 Non Functional Requirements.....	2 - 4
2.2.2.1 Types of Non Functional Requirements	2 - 4
2.2.2.2 Domain Requirements.....	2 - 6
2.2.3 Difference between Functional and Non Functional Requirements.....	2 - 6
2.3 User Requirements.....	2 - 10
2.3.1 Guidelines for Writing User Requirements	2 - 11
2.4 System Requirements	2 - 11
2.4.1 Structured Language Specification.....	2 - 12
2.5 Software Requirements Document.....	2 - 14
2.5.1 Characteristics of SRS	2 - 17
2.5.2 Example of SRS	2 - 18
2.6 Requirement Engineering Process	2 - 32
2.7 Feasibility Studies	2 - 35
2.8 Requirements Elicitation and Analysis	2 - 36
2.8.1 Stakeholders.....	2 - 36

2.8.2 Requirement Elicitation and Analysis Process.....	2 - 37
2.8.3 Requirement Discovery	2 - 38
2.9 Requirement Validation	2 - 49
2.10 Requirement Management.....	2 - 51
2.10.1 Enduring and Volatile Requirements	2 - 51
2.10.2 Requirements Management Planning.....	2 - 52
2.10.3 Requirement Change Management	2 - 53
2.11 Structured System Analysis	2 - 54
2.11.1 Designing Data Flow Diagrams.....	2 - 54
2.11.2 Rules for Designing DFD	2 - 55
2.11.3 Difference between Structured Analysis and Object Oriented Analysis.....	2 - 85
2.12 Petri Nets.....	2 - 85
2.13 Data Dictionary.....	2 - 89
Two Marks Questions with Answers.....	2 - 91

Unit - III

Chapter - 3	Software Design	(3 - 1) to (3 - 76)
3.1 Introduction.....		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 Model.....	3 - 13
3.4.1 Data Design Element	3 - 14
3.4.2 Architectural Design Element.....	3 - 14
3.4.3 Interface Design Elements.....	3 - 14
3.4.4 Component Level Design Elements.....	3 - 15
3.4.5 Deployment Level Design Elements	3 - 15
3.5 Design Heuristic.....	3 - 16
3.6 Introduction to Architectural Design	3 - 17
3.7 Software Architecture	3 - 17
3.7.1 Structural Partitioning	3 - 18
3.8 Data Design	3 - 20
3.9 Architectural Style	3 - 21
3.9.1 Architectural Styles	3 - 21
3.9.1.1 Data Centered Architectures.....	3 - 22
3.9.1.2 Data Flow Architectures	3 - 23
3.9.1.3 Call and Return Architecture	3 - 23
3.9.1.4 Object Oriented Architecture	3 - 24
3.9.1.5 Layered Architecture	3 - 25
3.9.2 Architectural Patterns	3 - 25
3.10 Architectural Design	3 - 26
3.10.1 Representing System in Context	3 - 27
3.10.2 Defining Archetypes	3 - 28
3.10.3 Refining Architecture into Components.....	3 - 29
3.10.4 Defining Instantiations of the System	3 - 29
3.11 Architectural Mapping using Data Flow	3 - 30
3.11.1 Transform Mapping.....	3 - 32
3.11.2 Transaction Mapping.....	3 - 37
3.12 Concept of User Interface Design	3 - 41

3.12.1 Golden Rules	3 - 42
3.12.1.1 Place the User in Control	3 - 42
3.12.1.2 Reduce the User's Memory Load.....	3 - 43
3.12.1.3 Make the Interface Consistent	3 - 44
3.12.2 User Interface Analysis and Design	3 - 45
3.12.2.1 Interface Analysis and Design Model	3 - 45
3.12.2.2 The Process.....	3 - 46
3.12.3 Interface Analysis	3 - 48
3.12.3.1 User Analysis.....	3 - 48
3.12.3.2 Task Analysis and Modelling.....	3 - 49
3.12.3.3 Analysis of Display Content	3 - 51
3.12.3.4 Analysis of Work Environment	3 - 52
3.12.4 Interface Design	3 - 53
3.12.4.1 Application of Interface Design Steps.....	3 - 53
3.12.4.2 User Interface Design Pattern	3 - 56
3.12.4.3 Design Issues	3 - 56
3.13 Component Level Design.....	3 - 59
3.13.1 Designing Class based Components	3 - 59
3.13.1.1 Basic Design Principle	3 - 59
3.13.1.2 Component Level Design Guideline.....	3 - 60
3.13.1.3 Cohesion	3 - 60
3.13.1.4 Coupling.....	3 - 61
3.13.2 Traditional Components.....	3 - 61
3.13.2.1 Structured Programming	3 - 62
Two Marks Questions with Answers	3 - 67

Unit - IV

Chapter - 4	Testing and Maintenance	(4 - 1) to (4 - 76)
--------------------	--------------------------------	----------------------------

4.1 Definition of 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 White Box Testing.....	4 - 5
4.3.1 Condition Testing	4 - 5
4.3.2 Loop Testing	4 - 6
4.3.3 Basis Path Testing	4 - 8
4.4 Black Box Testing.....	4 - 26
4.4.1 Equivalence Partitioning.....	4 - 27
4.4.2 Boundary Value Analysis (BVA)	4 - 28
4.5 Comparison between Black Box Testing and White Box Testing.....	4 - 35
4.6 Testing Strategy.....	4 - 36
4.7 Unit Testing	4 - 37
4.8 Integration Testing	4 - 38
4.8.1 Top Down Integration Testing.....	4 - 40
4.8.2 Bottom Up Integration Testing.....	4 - 41
4.8.3 Regression Testing.....	4 - 42
4.8.4 Smoke Testing	4 - 42
4.9 Validation Testing.....	4 - 43
4.9.1 Acceptance Testing.....	4 - 44
4.10 System Testing.....	4 - 45
4.10.1 Recovery Testing	4 - 46
4.10.2 Security Testing	4 - 46
4.10.3 Stress Testing.....	4 - 46
4.10.4 Performance Testing	4 - 46
4.11 Debugging.....	4 - 47
4.11.1 Testing Vs. Debugging	4 - 48
4.12 Software Implementation Techniques.....	4 - 49
4.12.1 Coding Practices	4 - 49
4.12.2 Coding Standards.....	4 - 52
4.12.3 Refactoring	4 - 53

4.13 Maintenance and Reengineering	4 - 54
4.13.1 Need for Maintenance	4 - 55
4.13.2 Types of Software Maintenance.....	4 - 55
4.13.3 Software Maintenance Process.....	4 - 56
4.13.4 Issues in Software Maintenance	4 - 58
4.14 Business Process Reengineering	4 - 58
4.14.1 BPR Model	4 - 59
4.15 Reengineering Process Model.....	4 - 60
4.16 Reverse and Forward Engineering	4 - 62
Two Marks Questions with Answers	4 - 65

Unit - V

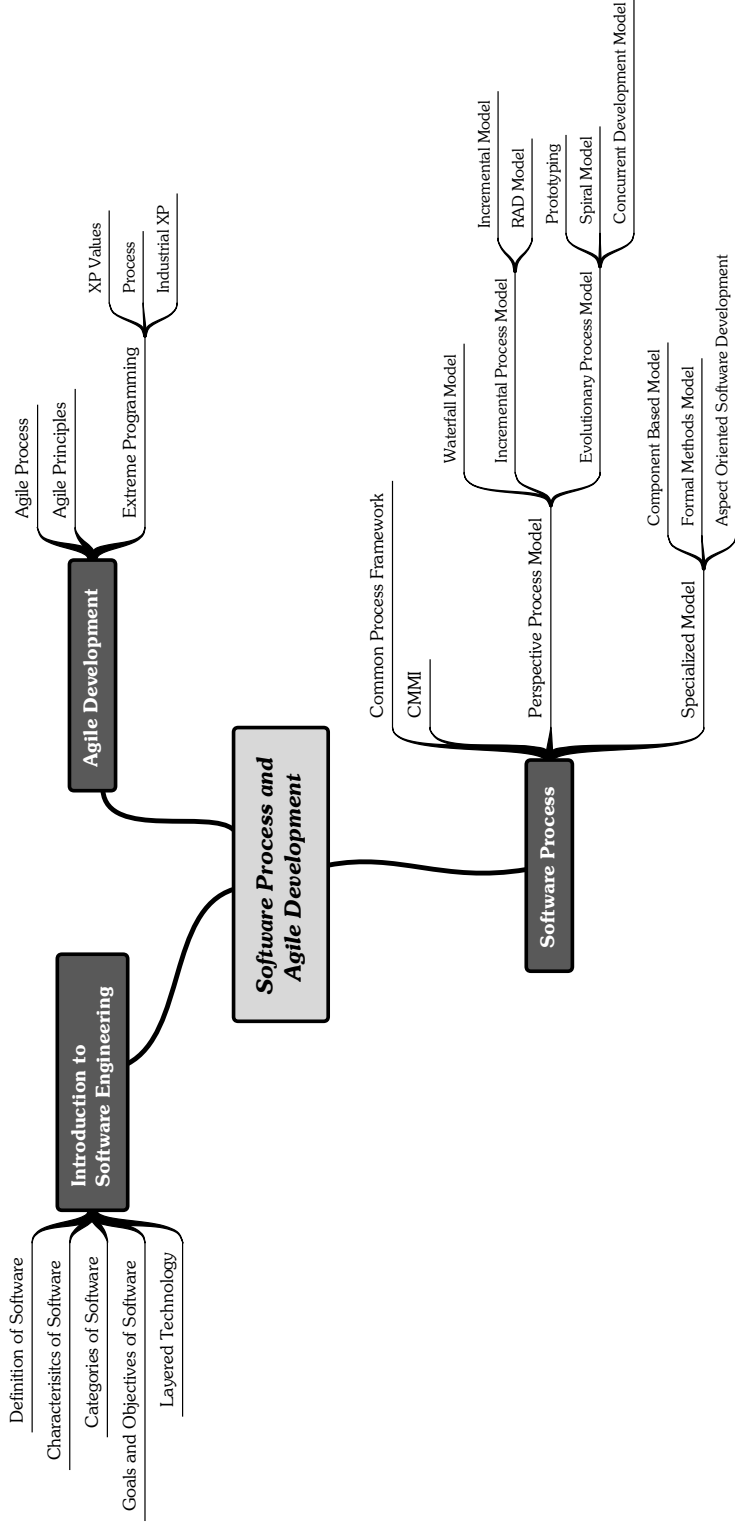
Chapter - 5	Project Management	(5 - 1) to (5 - 78)
--------------------	---------------------------	----------------------------

5.1 Software Project Management	5 - 2
5.2 Estimation.....	5 - 2
5.2.1 Software Sizing	5 - 2
5.2.2 Problem based Estimation	5 - 3
5.2.3 LOC based Estimation.....	5 - 4
5.2.4 Example of LOC based Estimation	5 - 5
5.2.5 Function Oriented Metrics	5 - 8
5.2.6 Example of FP based Estimation	5 - 10
5.3 Make Buy Decision	5 - 17
5.3.1 Outsourcing	5 - 19
5.4 COCOMO I Model	5 - 20
5.5 COCOMO II Model	5 - 25
5.6 Project Scheduling.....	5 - 33
5.6.1 Relationship between People and Effort	5 - 34
5.6.2 Task Sets	5 - 35
5.6.3 Task Network.....	5 - 38
5.6.4 Time Line Chart	5 - 39

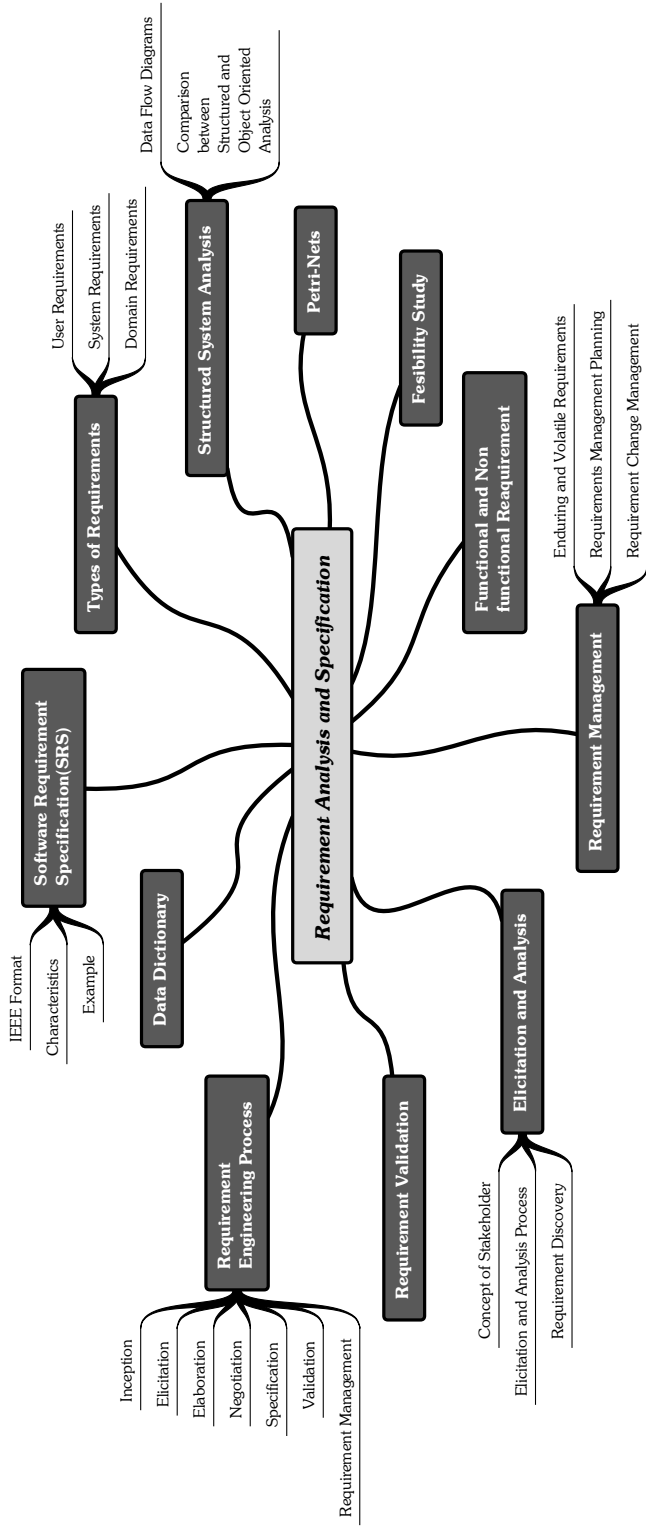
5.6.5 Tracking Schedule.....	5 - 40
5.6.6 Earned Value Analysis	5 - 41
5.7 Project Plan	5 - 50
5.8 Planning Process.....	5 - 50
5.9 Risk Management.....	5 - 51
5.9.1 Software Risks	5 - 52
5.9.2 Risk Identification.....	5 - 54
5.9.2.1 Risk Components and Drivers.....	5 - 55
5.9.2.2 How to Asses Overall Project Risk ?	5 - 56
5.9.3 Risk Projection.....	5 - 56
5.9.3.1 Building Risk Table.....	5 - 57
5.9.3.2 Assessing Risk Impact	5 - 58
5.9.4 RMMM	5 - 59
5.9.5 RMMM Plan.....	5 - 60
5.10 CASE Tools	5 - 62
5.10.1 Building Blocks of CASE	5 - 62
5.10.1.1 Taxonomy.....	5 - 63
5.10.2 Workbenches.....	5 - 66
5.10.3 Environments	5 - 67
5.10.4 Components of CASE.....	5 - 68
Two Marks Questions with Answers	5 - 69

Solved AU Question Papers	(S - 1) to (S - 6)
May-2019	(S - 1) to (S - 4)
Dec.-2019.....	(S - 5) to (S - 6)

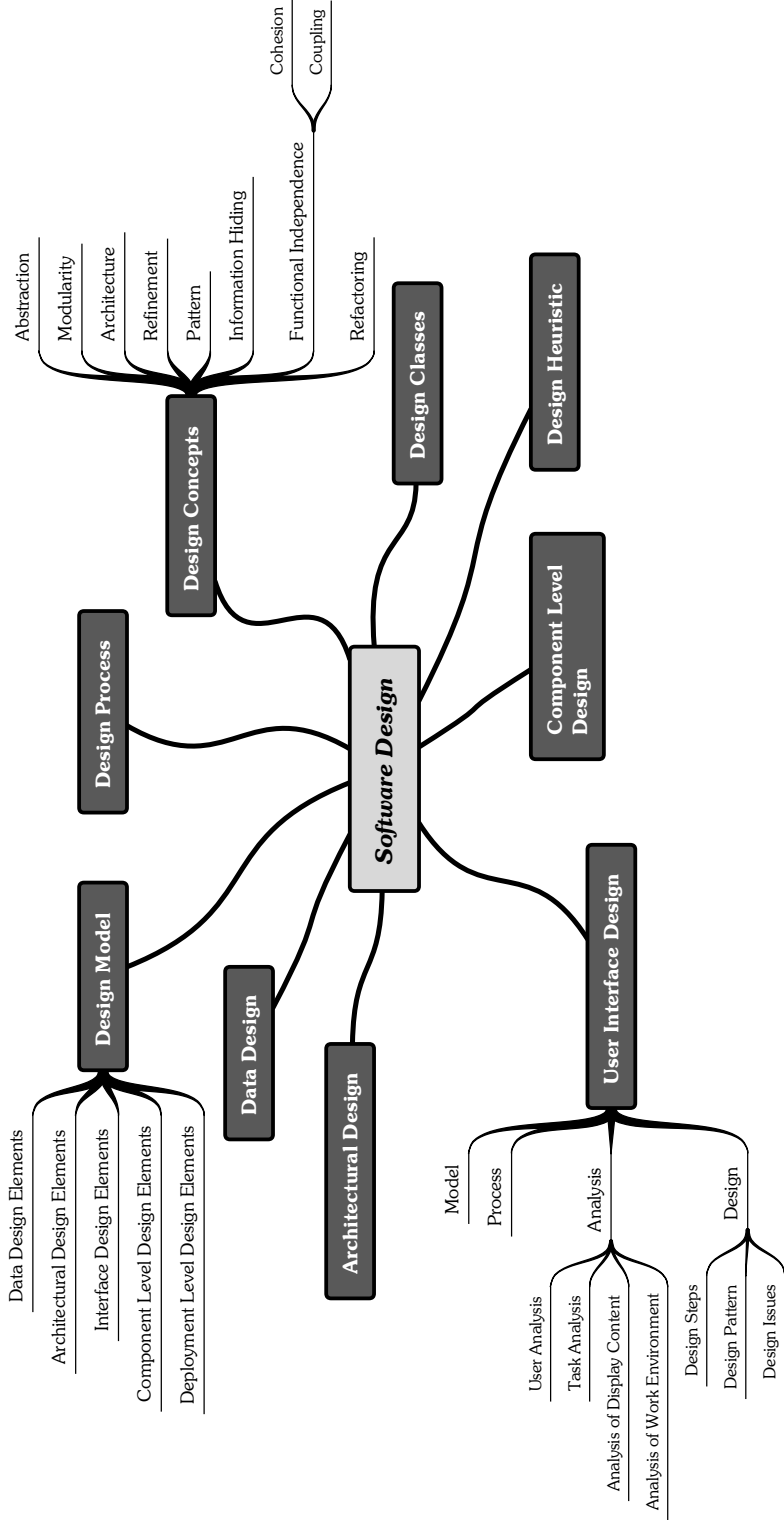
Mind Map : Chapter 1 : Software Process and Agile Development



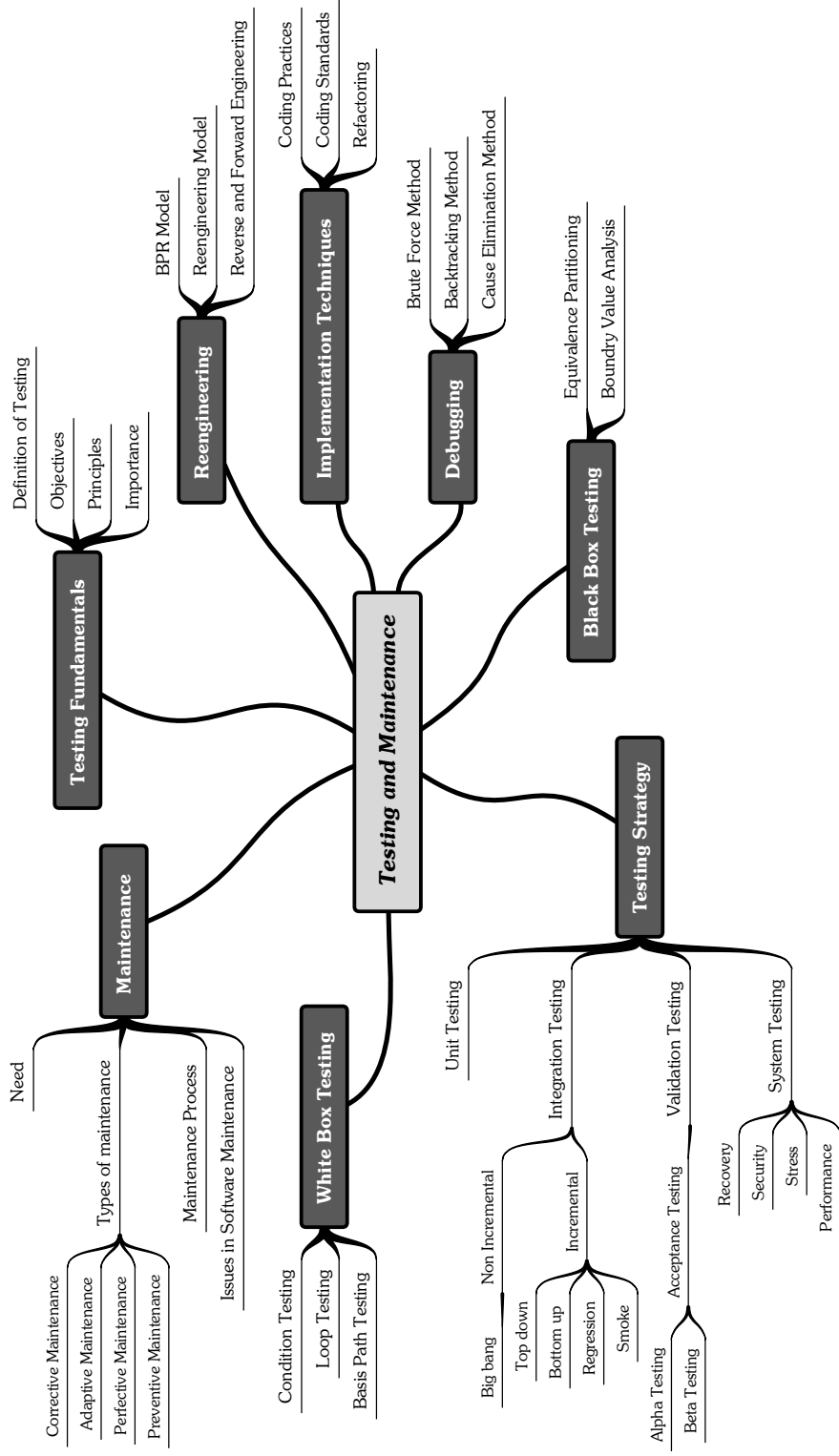
Mind Map : Chapter 2 : Requirement Analysis and Specification



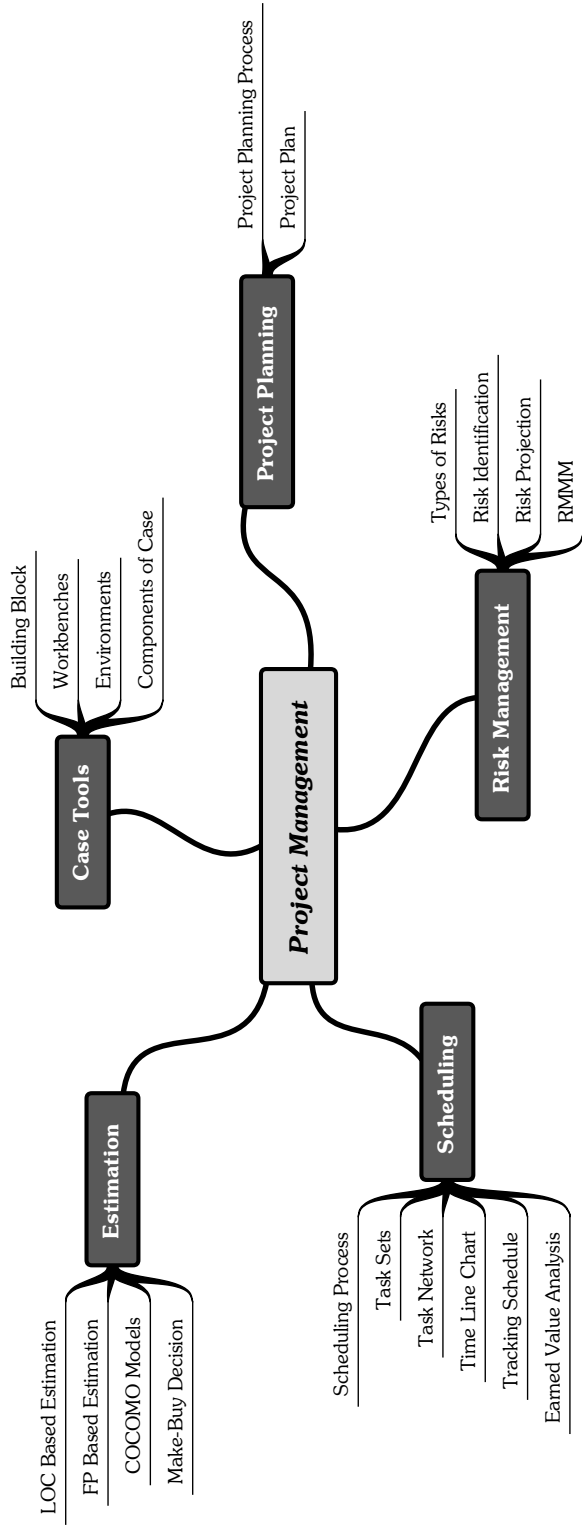
Mind Map : Chapter 3 : Software Design



Mind Map : Chapter 4 : Testing and Maintenance



Mind Map : Chapter 5 : Project Management



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		
1.5 Software Process	Dec.-10,17,	Marks 7
1.6 Prescriptive Process Models	May-05,06,09,15,16,17,18,	
	Dec.-06,09,11,16,17,19	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,	Marks 8
Two Marks Questions with Answers		
Long Answered Questions		

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 wear 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 :

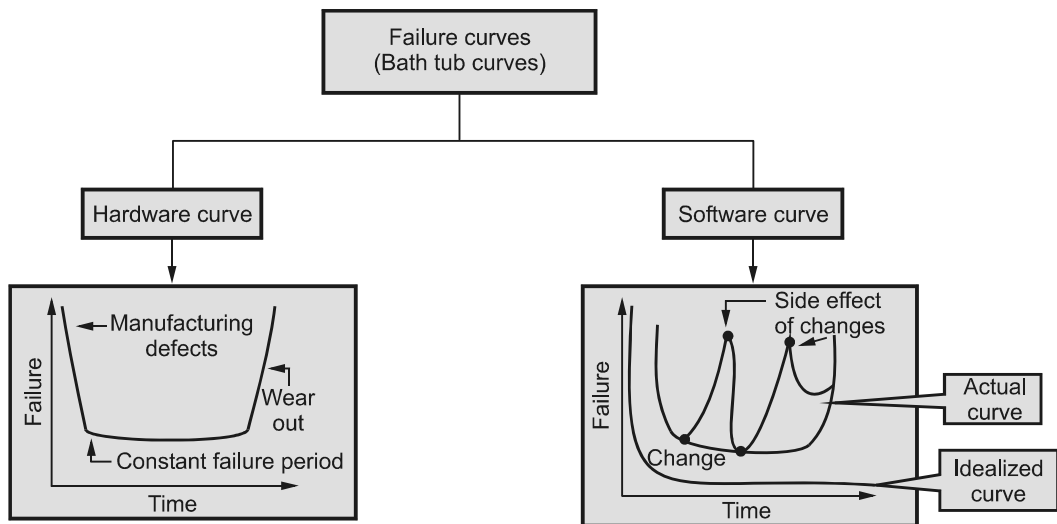


Fig. 1.1.1 Failure curves for hardware and software

- 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 automotive 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?*
2. *Write a note on the unique characters of a software.*

AU : Dec.-17, Marks 4

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

- 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.
- 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.

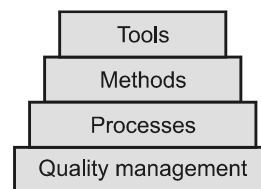


Fig. 1.4.1 Layered technology

1.5 Software Process

AU : Dec.-10,17, 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.

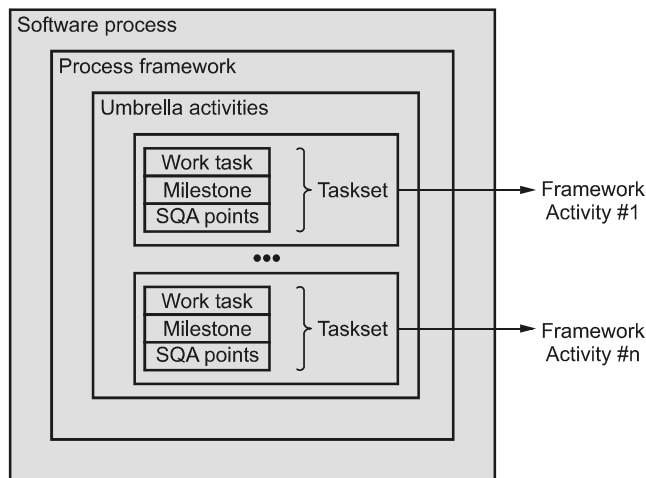


Fig. 1.5.1 Software process framework

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 Question

1. Explain the CMMI model to access the organization level.

AU : Dec.-17, Marks 7

1.6 Prescriptive Process Models

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

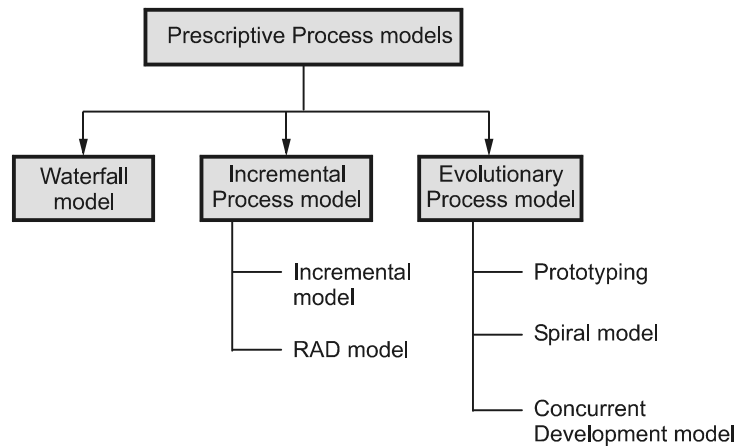


Fig. 1.6.1 Prescriptive process model

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

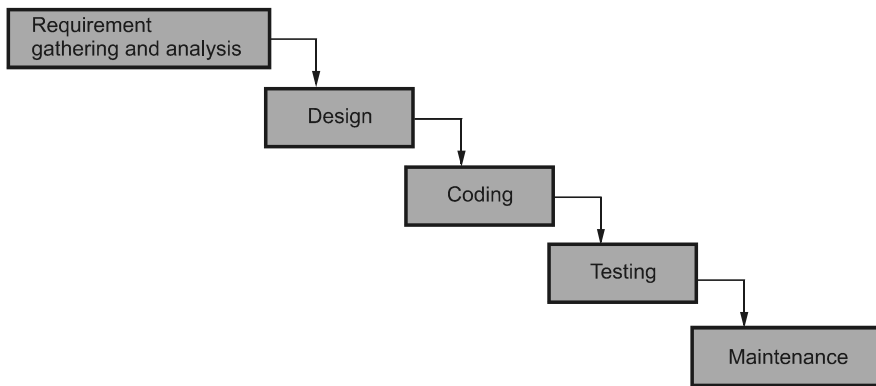


Fig. 1.6.2 Waterfall model

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.

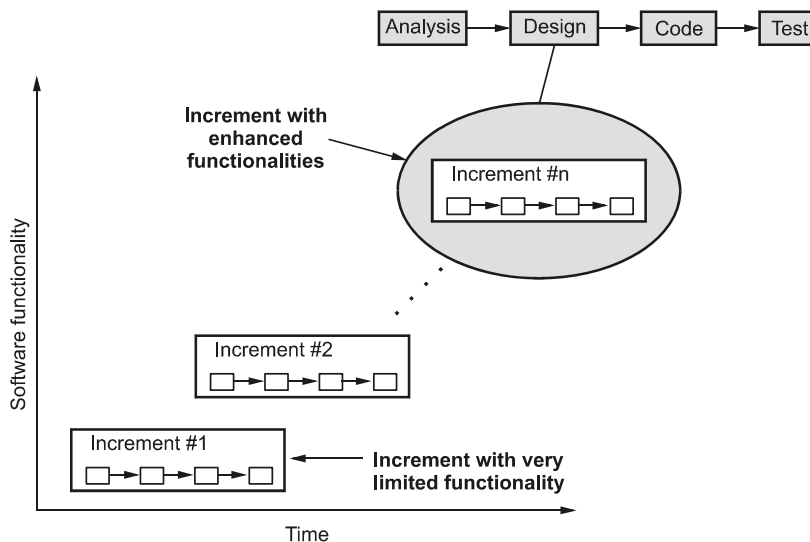


Fig. 1.6.3 The incremental model

- 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 parallelly.
- 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.

3. This model uses multiple teams on scalable 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.

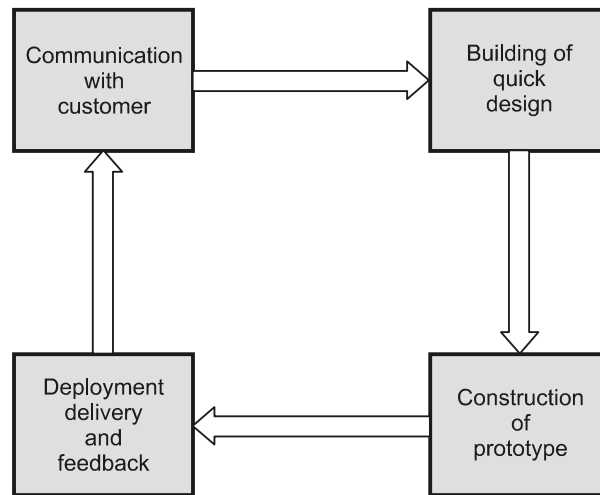


Fig. 1.6.5 Prototyping

- 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 almost always involve Human-machine 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 spiral 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

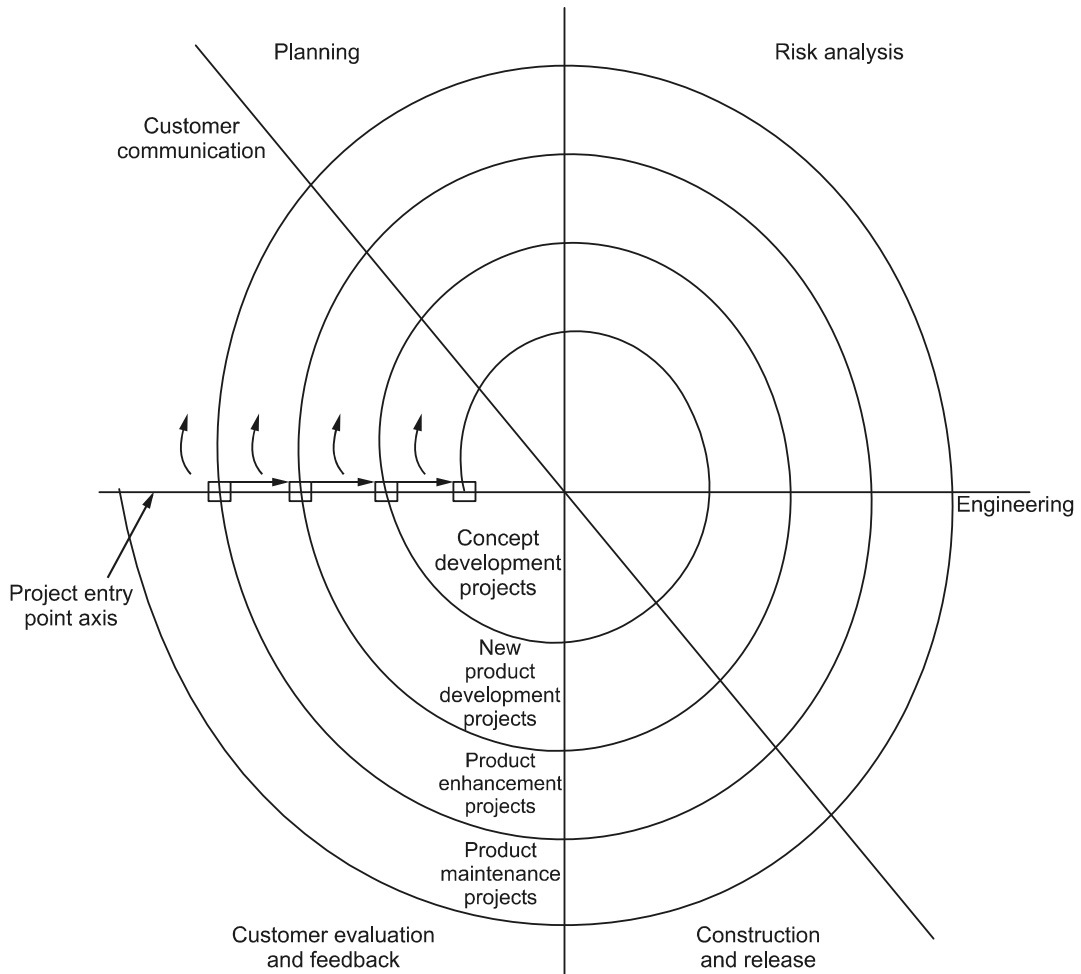


Fig. 1.6.6 Spiral model

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 :
 - Customer communication** - In this region, it is suggested to establish customer communication.
 - Planning** - All planning activities are carried out in order to define resources time line and other project related activities.
 - Risk analysis** - The tasks required to calculate technical and management risks are carried out.
 - 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 projects.
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.

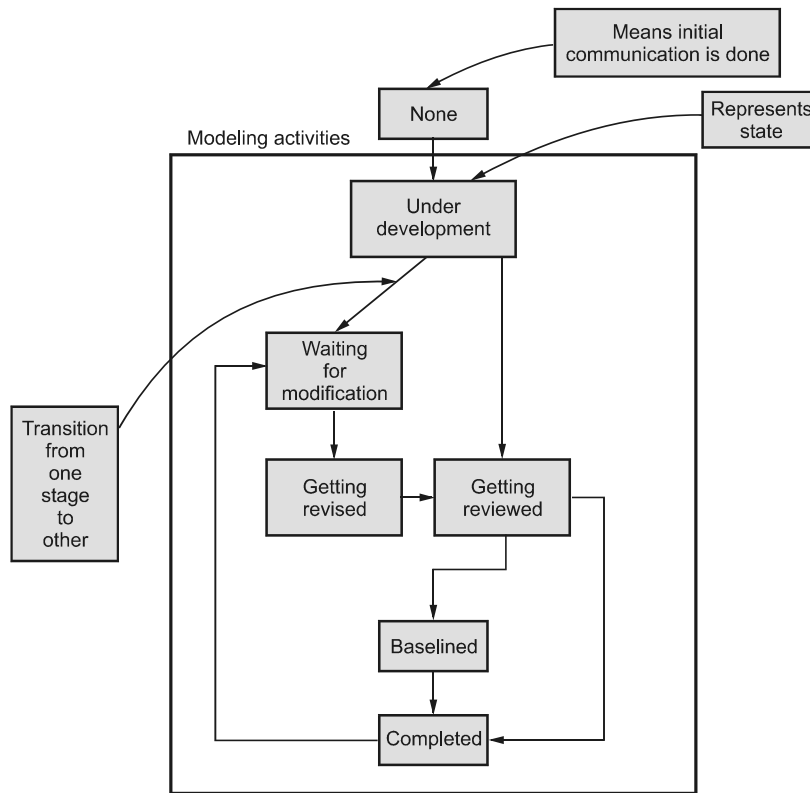


Fig. 1.6.7 Concurrent development model

- 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.
2. Discuss the prototyping model. What is the effect of designing a prototype on the overall cost of the software project ?
3. Which process model is best suited for risk management ? Discuss in detail with an example. Give the advantages and disadvantages of the model.
4. What is the significance of the spiral model when compared with other models.

AU : May-15, Marks 8

AU : May-16, Marks 8

AU : Dec.-16, Marks 16

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

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.

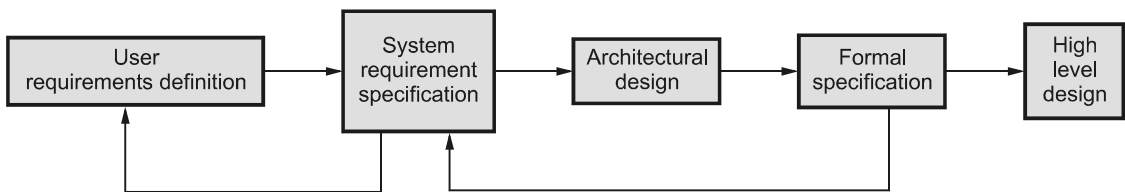


Fig. 1.7.1 Activities for formal method model

- The formal methods model offers **defect-free** software. However there are some **drawbacks** of this model which resist 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 inconsistency 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 :

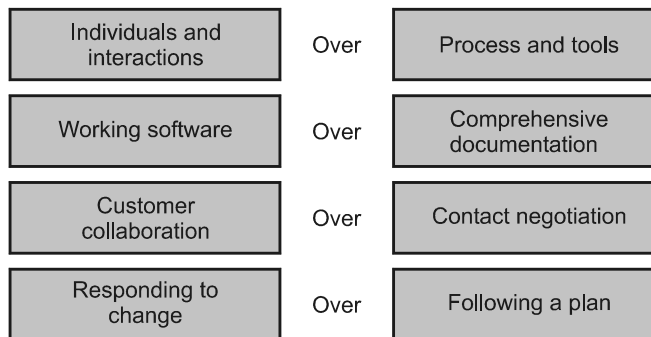
Waterfall model	RAD model	Spiral model	Formal methods model
Requirements must be clearly understood and defined at the beginning only.	Requirements must be clearly understood and defined at the beginning only.	The requirements analysis and gathering can be done in iterations because requirements get 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 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.

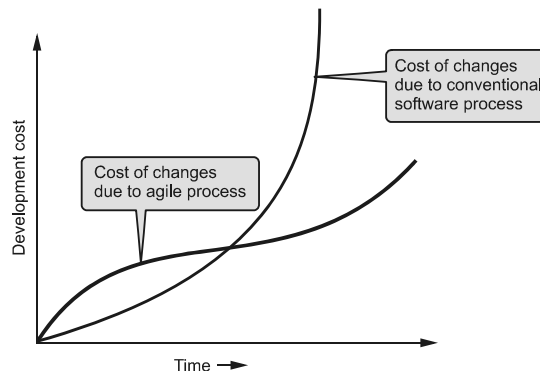


Fig. 1.9.1 Influence of software development approach on agile process

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 throughout 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 adjust their behavior accordingly.

Review Questions

1. List the principles of agile software development.
2. Define Agility. List any five principles of agility.
3. What is agility ? Elaborate the agile principles.

AU : Dec-16, Marks 4

AU : May-19, Marks 5

AU : Dec.-19, Marks 13

1.10 Extreme Programming

AU : May-19, Marks 8

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 future 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 requires 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 prioritizes 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 prioritizes 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.

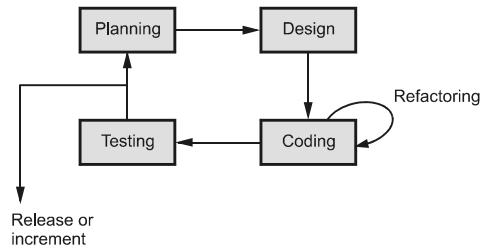


Fig. 1.10.1 XP process

- 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
<p>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.</p> <p>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.</p> <p>The customer can deposit the desired amount in his account. The deposited amount must be added to the available balance.</p>

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 Question

1. Explain the phases in extreme programming process.

AU : May-19, Marks 8**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 wear 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

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

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

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.

AU : IT, May-07

Ans. :

	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 ?**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 wear 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 accommodate 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 accommodating 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.



Notes

2

Requirements Analysis and Specification

Syllabus

Software requirements : Functional and non-functional, User requirements, System requirements, Software requirements document - Requirement engineering process : Feasibility studies, Requirements elicitation and analysis, Requirements validation, Requirements management-Classical analysis : Structured system analysis, Petri nets - Data dictionary.

Contents

2.1 Software Requirements	
2.2 Functional and Non Functional Requirements	Dec.-13,17, May-13, Marks 11
2.3 User Requirements	
2.4 System Requirements	May-16, Marks 4
2.5 Software Requirements Document.....	Dec.-11,19
.....	May-13,14,16,18,19, Marks 16
2.6 Requirement Engineering Process.....	May-15, 17, Dec.-15,19 Marks 16
2.7 Feasibility Studies	May-17,Dec.-17, Marks 8
2.8 Requirements Elicitation and Analysis	Dec.-13,16,
.....	May-08,17,18,19, Marks 16
2.9 Requirement Validation	
2.10 Requirement Management	May-16, Marks 12
2.11 Structured System Analysis	Dec.-13,15,16,17, 19
.....	May-08,10,15,17,18,19 Marks 16
2.12 Petri Nets	Dec.-19..... Marks 13
2.13 Data Dictionary	Dec.-06, 08, 09, Marks 16

2.1 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

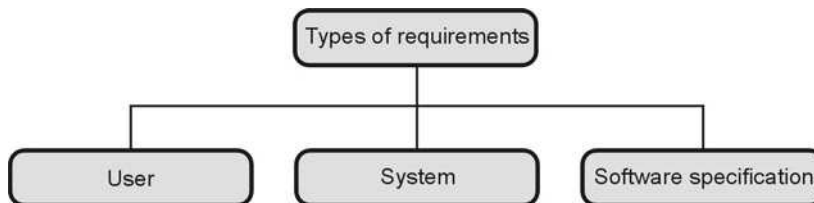


Fig. 2.1.1 Types of requirements

• 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, 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.

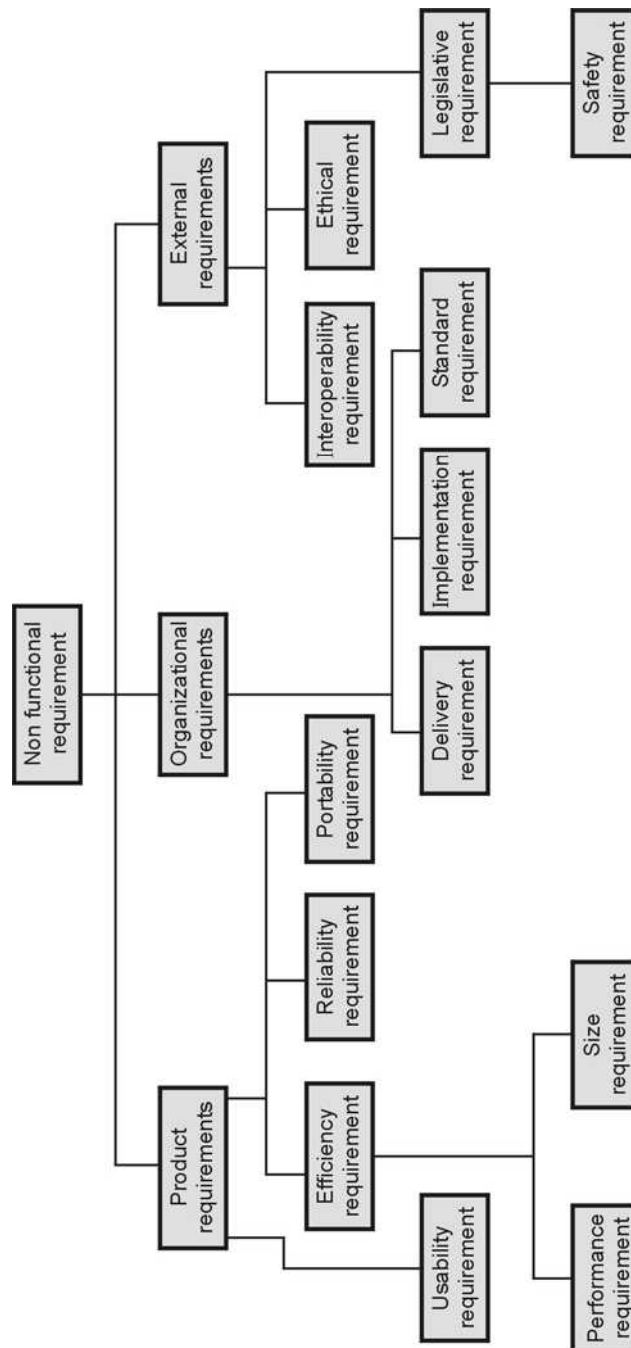


Fig. 2.2.1 Types of non functional requirement

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 wishes 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 the updated information in correct order.
2. **Availability** : Any information must be quickly available from computer to the authorized user.
3. **Security** : The application must be password protected. This feature is essential to avoid any unauthorized access to the application.
4. **Maintainability** : The application should be maintained in such a manner that if any change in requirement occurs then it should be easily incorporated in an individual module.
5. **Extensibility** : The application should be maintained in such a manner that if any new requirement occurs then it should be easily incorporated in an individual 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 code 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.

Review Question

1. Explain the metrics used for specifying non functional requirements.

AU : May-13, Marks 8

2.3 User Requirements

- The user requirements should describe functional and non functional requirements in such a way that they are understandable by system users who don't have detailed technical knowledge.
- User requirements are defined using natural language, tables and diagrams because these are the representations that can be understood by all users.

Various **problems** that can arise **in the requirement specifications** when requirements are given in natural language -

Lack of clarity

Sometimes requirements are given in ambiguous manner. It is expected that text should help in clear and precise understanding of the requirements.

Requirements confusion

There may be confusion in functional requirements and non functional requirements, system goals and design information.

Requirements mixture

There may be a chance of specifying several requirements together as a single requirement.

2.3.1 Guidelines for Writing User Requirements

- Prepare a standard format and use it for all requirements.
- Apply consistency in the language. Use
 - 'shall' for mandatory requirements
 - and 'should' for desirable requirements.
- The text which is mentioning the key requirements should be highlighted.
- Avoid the use of computer jargon (computer terminologies). It should be written in simple language.

For example

Consider a spell checking and correcting system of a word processor. The user requirements can be given in natural language as

1. The system should possess a traditional **word dictionary and user supplied dictionary**. It shall provide a user-activated facility which checks the spelling of words in the document against spellings in the system dictionary and user-supplied dictionaries.
2. When a word is found in the document which is not given in the dictionary, then the system should suggest 10 alternative words. These alternative words should be based on a match between the word found and corresponding words in the dictionaries.
3. When a word is found in the document which is not in any dictionary, the system should propose following options to user :
 1. Ignore the corresponding instance of the word and go to next sentence.
 2. Ignore all instances of the word.
 3. Replace the word with a suggested word from the dictionary.
 4. Edit the word with user-supplied text.
 5. Ignore this instance and add the word to a specified dictionary.

2.4 System Requirements

AU : May-16, Marks 4

- System requirements are more detailed specifications of system functions, services and constraints than user requirements.

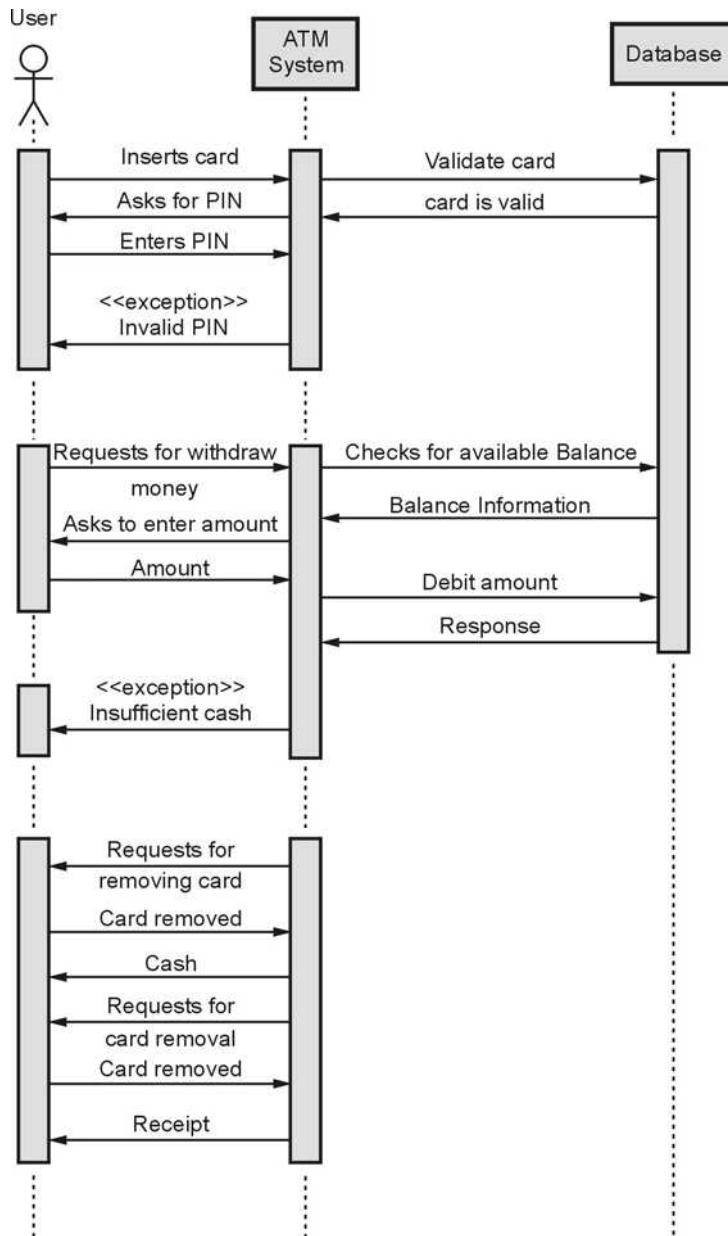
- They are intended to be a basis for designing the system.
- They may be incorporated into the system contract.
- The system requirements can be expressed using system models.
- The requirements specify what the system does and design specifies how it does.
- System requirement should simply describe the external behavior of the system and its operational constraints. They should not be concerned with how the system should be designed or implemented.
- For a complex software system design it is necessary to give all the requirements in detail.
- Usually, natural language is used to write system requirements specification and user requirements.

2.4.1 Structured Language Specification

- One of the method of writing requirements is by using structured natural language.
- All the requirements should be written in a **standard way** while using structured language specification.
- The **advantage** of specifying requirements using this method is that requirement become **understandable** and **expressive**.
- The only necessary thing while writing requirements using natural language is that some degree of **uniformity must be maintained**.
- Extra information can be added when the requirements are written using natural language. This information can be represented using tables or graphical models.
- One way of using graphical model is use of **sequence diagram**.
- The sequence diagram represents the sequence of actions that user performs while interacting the system.
- Example : Following is a sequence diagram for withdrawal of cash from ATM. (See Fig. 2.4.1 on next page)

Why requirement and design are inseparable ?

- A system architecture may be designed to structure the requirements.
- The system may inter-operate with other systems and that may generate design requirements.
- The use of a specific design may be a domain requirement.

**Fig. 2.4.1 Sequence diagram of ATM withdrawal**

Example 2.4.1 *Differentiate between user and system requirements.***AU : May-16, Marks 4****Solution :**

User Requirement	System Requirement
The focus of this types of requirements is on the problem domain.	The focus of this type of requirements is on solution domain.
These requirements describe what effects need to be achieved.	These requirements describe what software the software must do.
User requirement tell what application should do to satisfy users needs.	System requirements tell a system should have to be able to run program.
The user defines his/her requirements for the system. Hence it is not necessary to incorporate all the user requirements in the system contract.	These requirements must be incorporated into system contract.

2.5 Software Requirements Document

AU : Dec.-11, 19, May-13,14,16,18,19, 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 be 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.

2.5.1 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.

2.5.2 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.....	2
3.2 Technical Issues.....	2
4. Interface Requirements.....	2
4.1 GUI	3
4.2 Hardware Interface	3
4.3 Software Interface.....	3
5. Performance Requirements.....	4
6. Design Constraints.....	4
7. Other Non-functional Attributes.....	4
7.1 Security.....	5

7.2 Reliability	5
7.3 Availability	5
7.4 Maintainability	5
7.5 Reusability	5
8. Operational Scenarios	5
9. Preliminary Schedule	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 be 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.

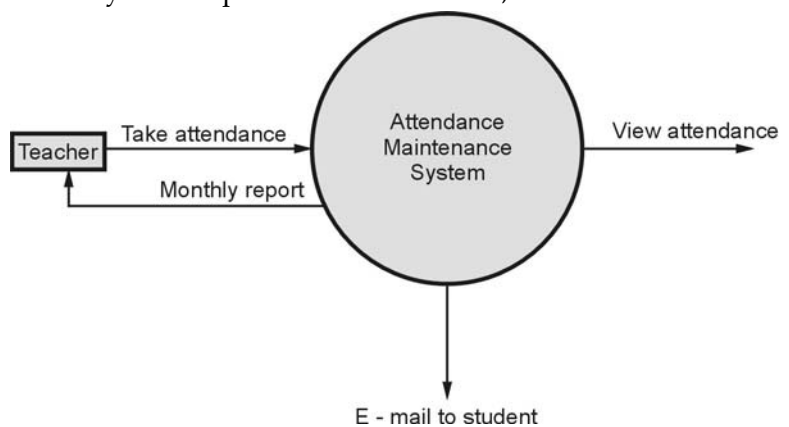


Fig. 2.5.1

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

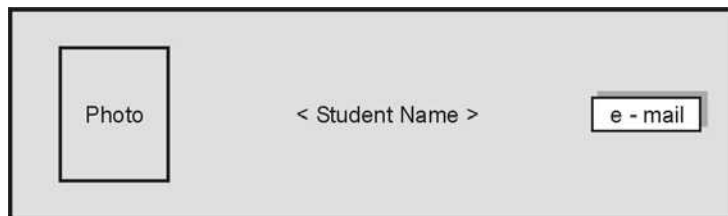


Fig. 2.5.2

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 Maintainability

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 2.5.1 *An independent trunk company wants to track and record its drivers driving habits. For this purpose the company has rented 800 phone 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 database.
- 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 2.5.2 *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 of 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

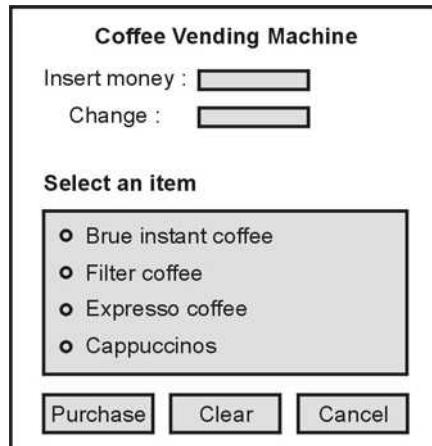
A coffee vending machine serves coffee to customers. A customer can choose a type of coffee among a list of options, supply the amount required and get served. Each coffee is prepared by adding units of hot water, coffee powder, milk and sugar. The recipe for each coffee is stored.

3.2 Technical Issues

The system should be implemented in Java

4. Interface Requirements

4.1 GUI



Coffee Vending Machine

Insert money :

Change :

Select an item

- ☐ Brue instant coffee
- ☐ Filter coffee
- ☐ Expresso coffee
- ☐ Cappuccinos

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 Maintainability

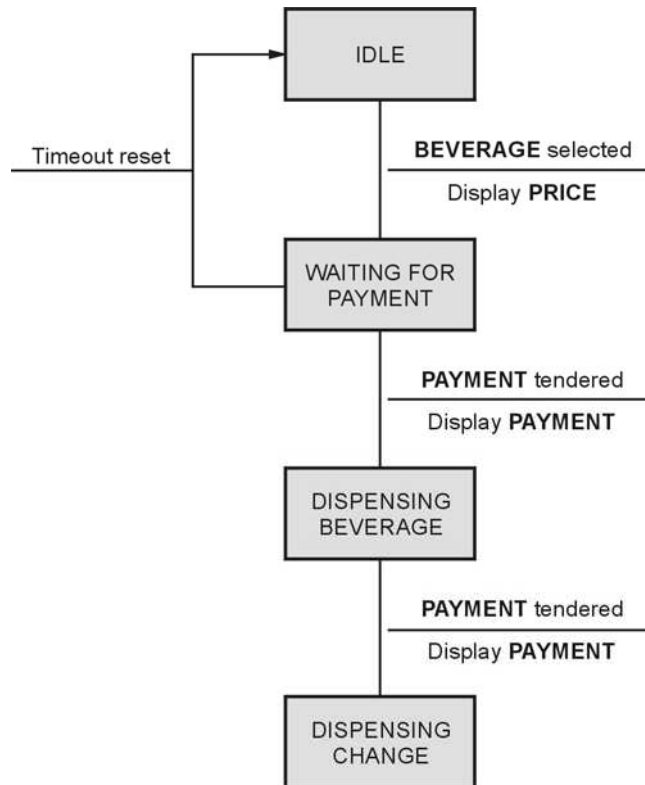
There should be a facility to clear the choice that is been made.

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 of 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 2.5.3 *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. The 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.

1.3 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 various 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 or 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 or 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

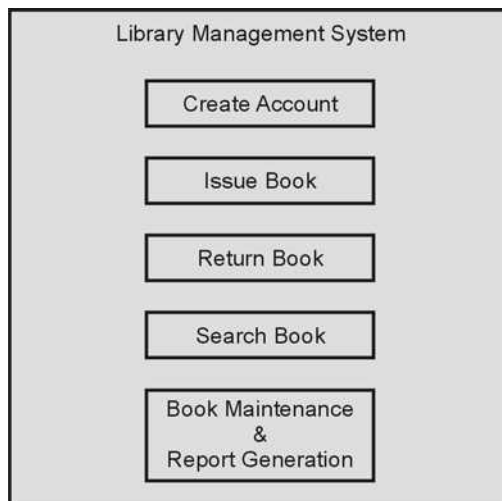
4.1 GUI

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



The diagram shows a login screen for the Library Management System. It features a title bar at the top. Below it, the word 'Login' is centered. There are two input fields: 'Username' with the text 'Admin' and 'Password' with four dots. Below these fields are two buttons: 'Sign in' and 'Sign up'. At the bottom, there is a link that says 'Forgot Password?'.

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



The diagram shows the main menu for the Library Management System. It features a title bar at the top. Below it, there are five buttons arranged vertically: 'Create Account', 'Issue Book', 'Return Book', 'Search Book', and 'Book Maintenance & Report Generation'.

GUI 2 : 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, first name, phone number, class and so on.

GUI 3 : If the user selects for the Search then the user will be allowed to search for the 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 within 3 seconds and response must be given.

6. Design Constraints

The system must be designed as a standalone system and must be run on window based system. The system can be developed in Java or Visual Basic. The database can be 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. The password can be changed by the system administrator/Librarian. The Student should be allowed to access the book database for searching of book. He/She is not allowed to 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**

2.6 Requirement Engineering Process

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

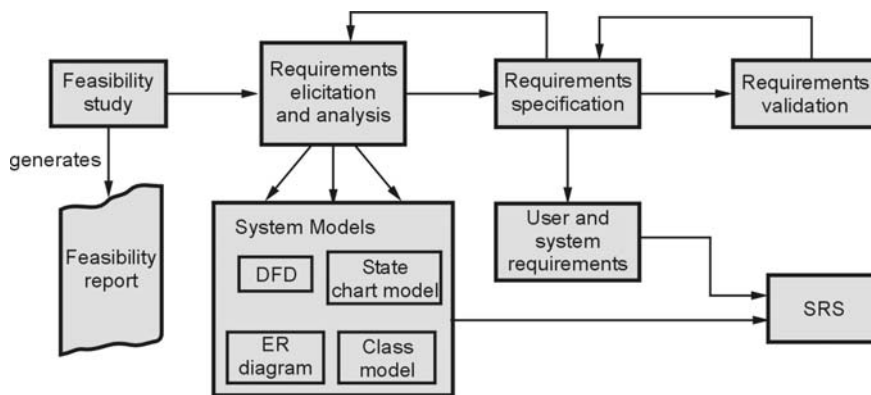


Fig. 2.6.1 Requirement engineering process

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

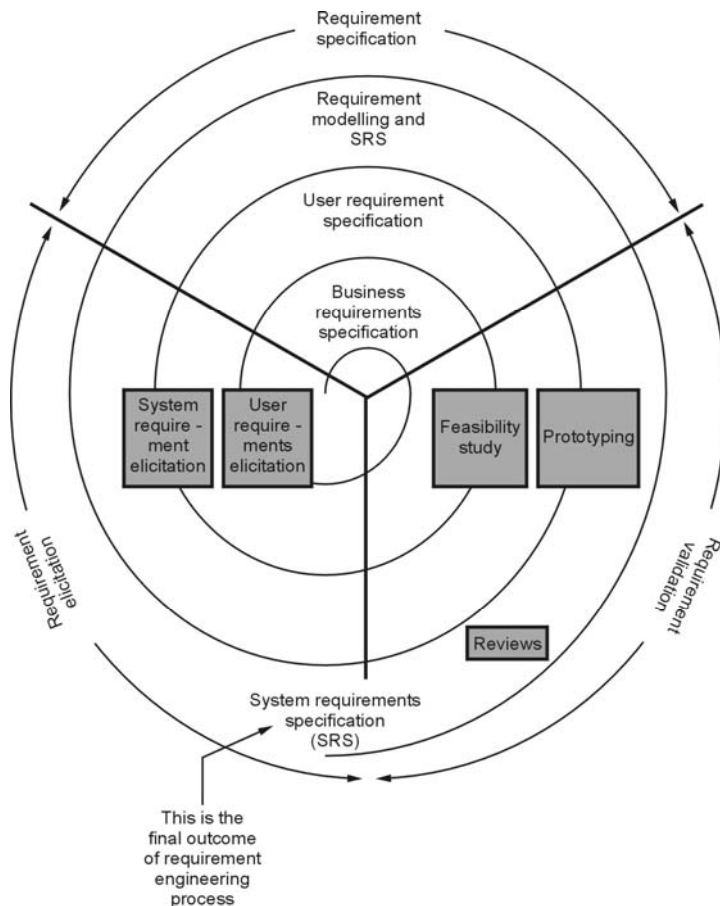


Fig. 2.6.2 Spiral model of requirements engineering process

- 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. 2.6.2.

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

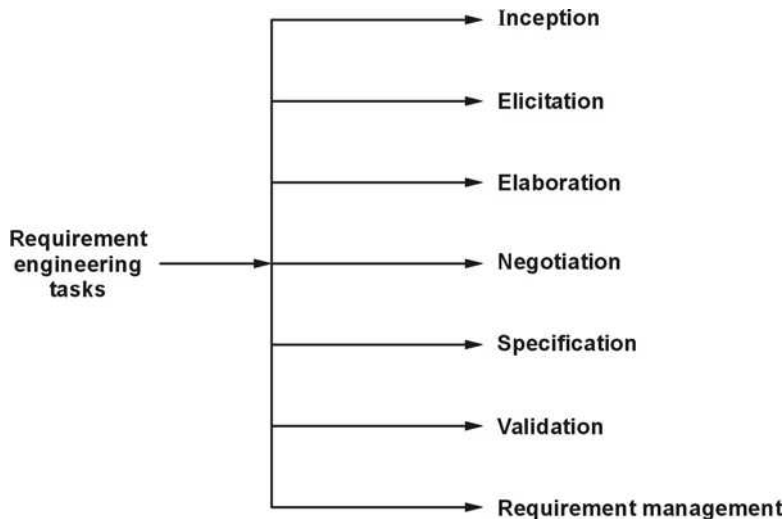


Fig. 2.6.3 Requirement engineering tasks

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 2.6.1 *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

2.7 Feasibility Studies

AU : May-17, 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

2.8 Requirements Elicitation and Analysis

AU : Dec.-13,16, 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.

2.8.1 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.

2.8.2 Requirement Elicitation and Analysis Process

The spiral model as given below depicts the requirement elicitation and analysis process.

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 is resolved by requirement engineers.
- 4. Requirement documentation :** This is the specification of all the requirements. And an important requirement document is created.

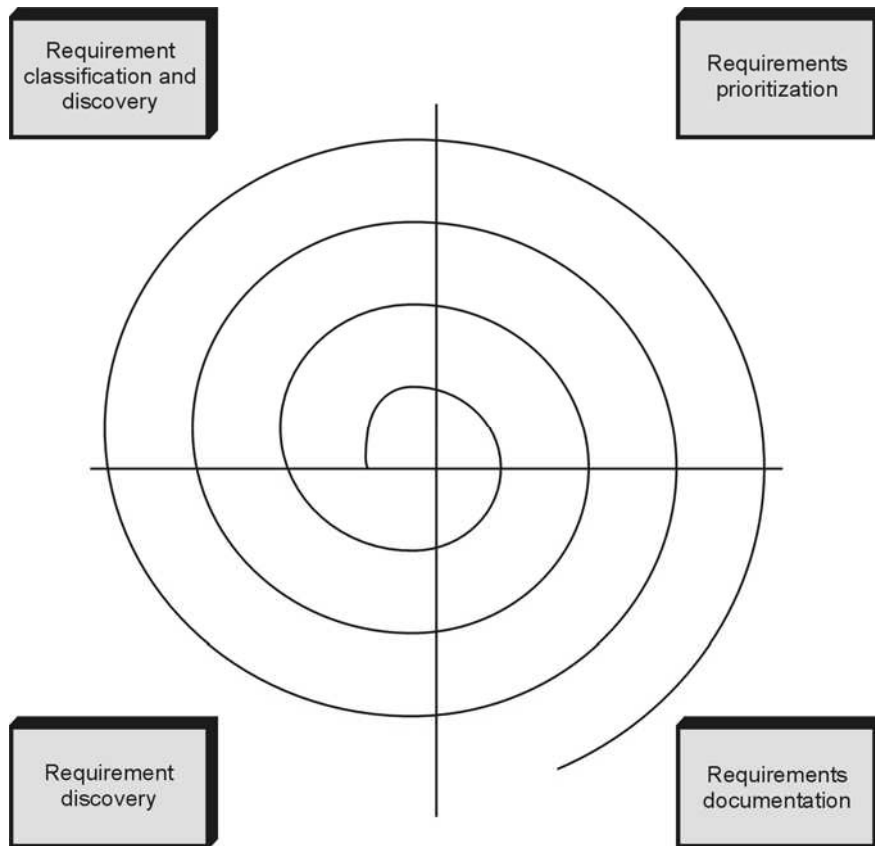


Fig. 2.8.1 Requirements elicitation and analysis process

2.8.3 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.



Fig. 2.8.2 Types of interview

There are two types of interviews. (Refer Fig. 2.8.2.)

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	<ol style="list-style-type: none"> 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	<ol style="list-style-type: none"> 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. 2.8.3

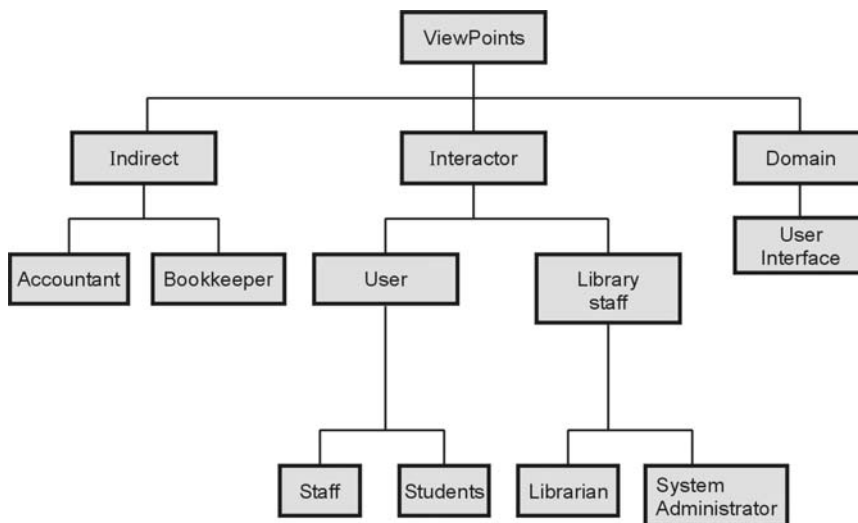


Fig. 2.8.3 Viewpoints for library system

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 requirements of the system can be identified. The typical notations used in the use cases are as shown in Fig. 2.8.4.

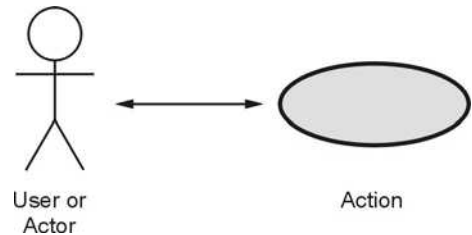


Fig. 2.8.4 Use-case notation

The use case for ATM system is as shown below.

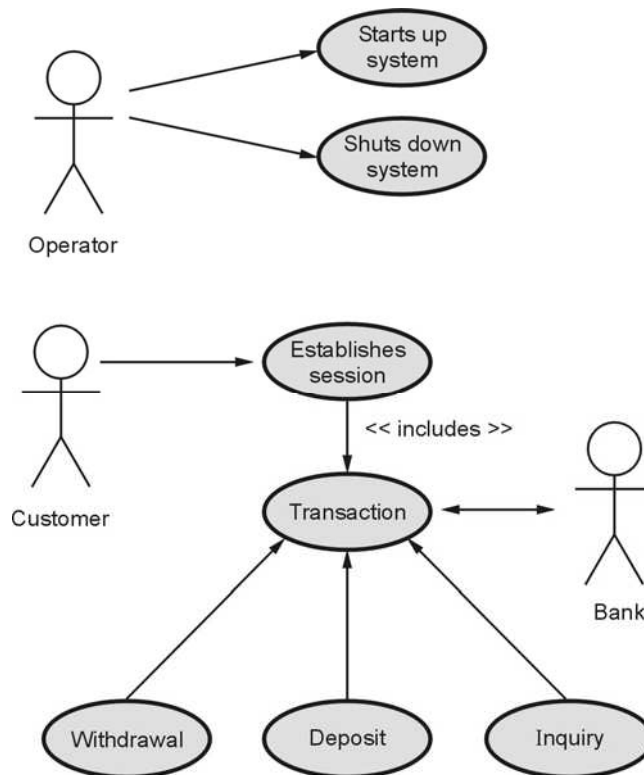


Fig. 2.8.5 Use cases for ATM system

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 connection 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 the 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.

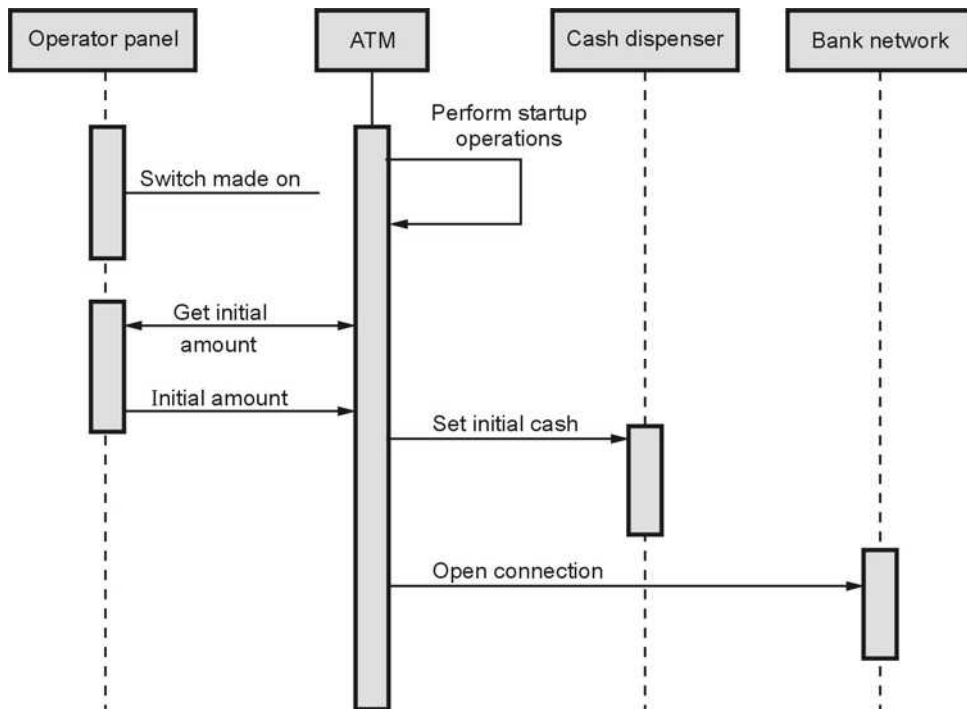


Fig. 2.8.6 Interaction diagram

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 find 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.

Ethnography can be combined with prototyping

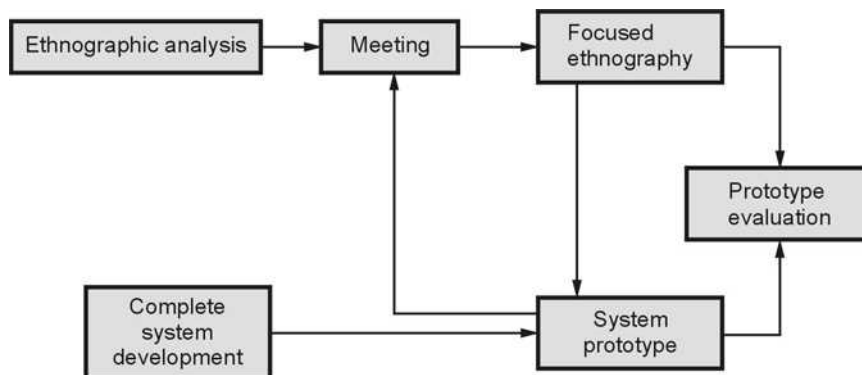


Fig. 2.8.7 Ethnographic process

The ethnography informs the development of prototype 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 2.8.1 *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 prioritised 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.

Use case diagram

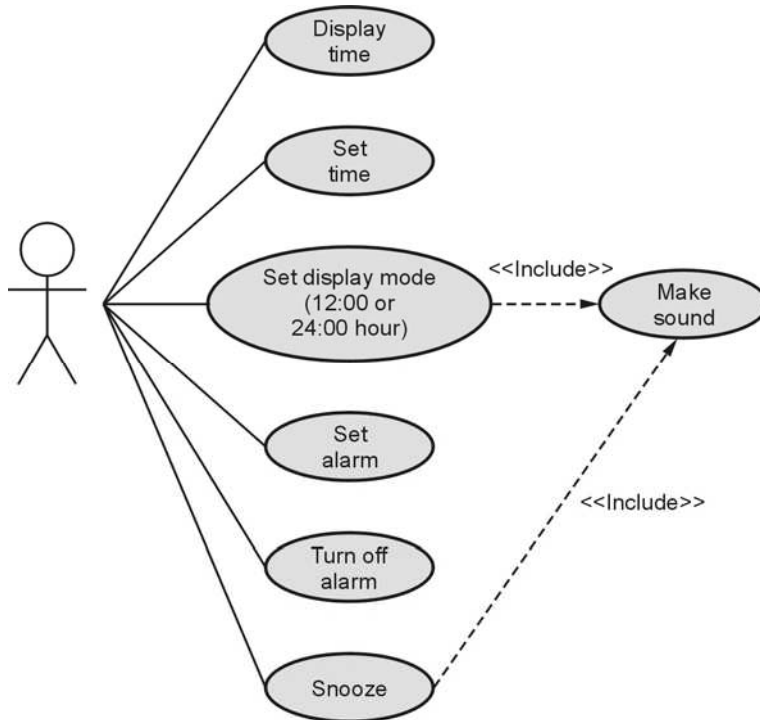


Fig. 2.8.8

Example 2.8.2 *Develop complete use case for the following :*

Making a withdrawal from ATM in a bank

Solution :

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	<ol style="list-style-type: none"> 1. Customer observes the control panel 2. Customer enters the ATM card 3. Customer enters the PIN 4. Customer selects the operation - withdrawal 5. Customer collects the cash, statement, card etc

Exceptions	<ol style="list-style-type: none"> 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	<ol style="list-style-type: none"> 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 ?

Example 2.8.3 *Develop complete use case for the following : Searching for books based on title in an on-line book store.*

Solution :

Use Case	On-line book store
Primary Actor	Customer
Goal in Context	To monitor all the functionalities required to establish the session
Scenario	<ol style="list-style-type: none"> 1. Customer logs in to purchase the book online 2. Customer browses through the catalog based on title of the book 3. Customer selects the book and adds it to the shopping cart. 4. The contents of the shopping cart are displayed. 5. The book details and shipping information is confirmed by the customer. 6. The customer pays for the book
Extensions	<ol style="list-style-type: none"> 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)

Example 2.8.4 *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 -

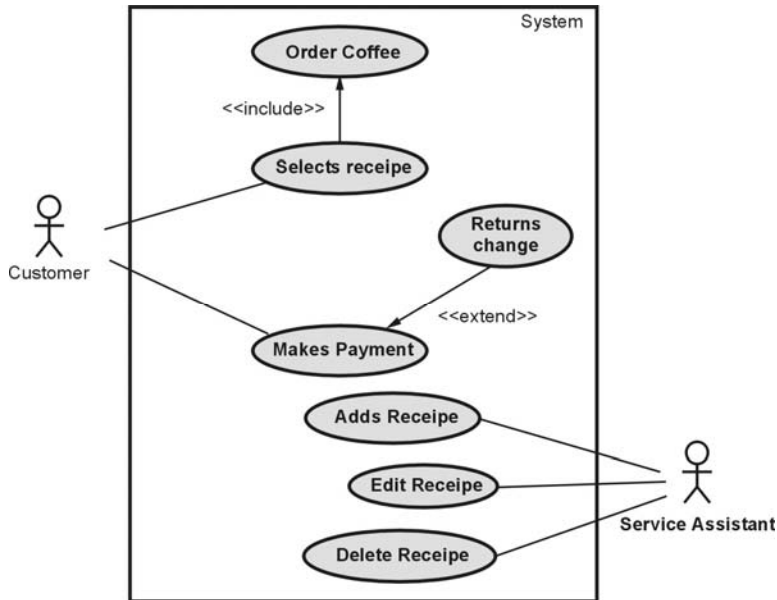


Fig. 2.8.9

ii) Activity Diagram

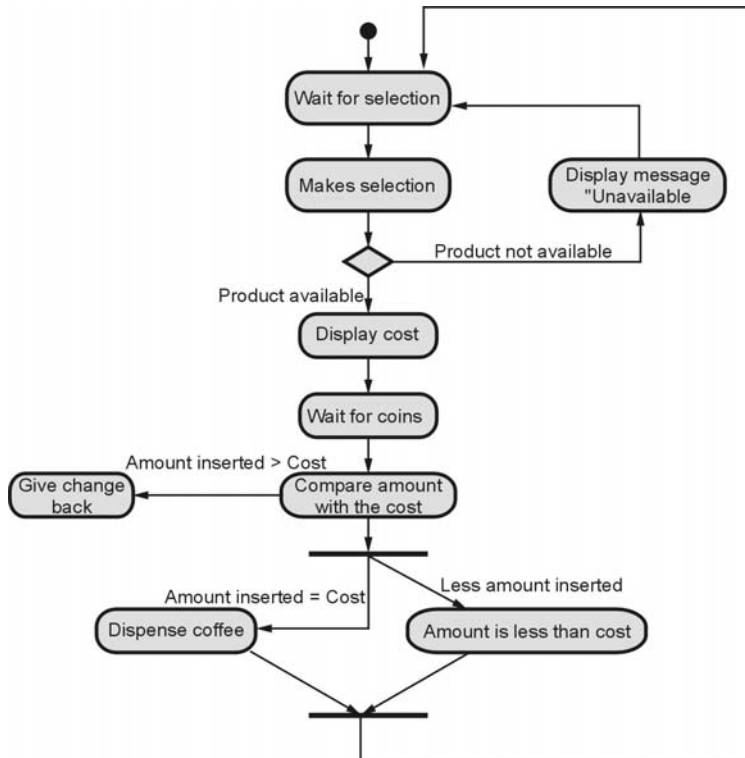


Fig. 2.8.10 Activity diagram for customer orders for the coffee

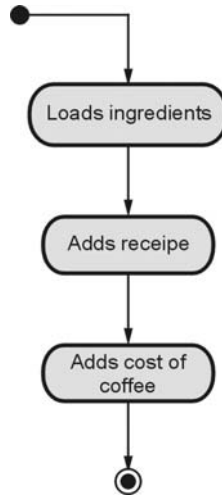


Fig. 2.8.11 Activity diagram for service assistant for adding the receipt

Sequence Diagram :

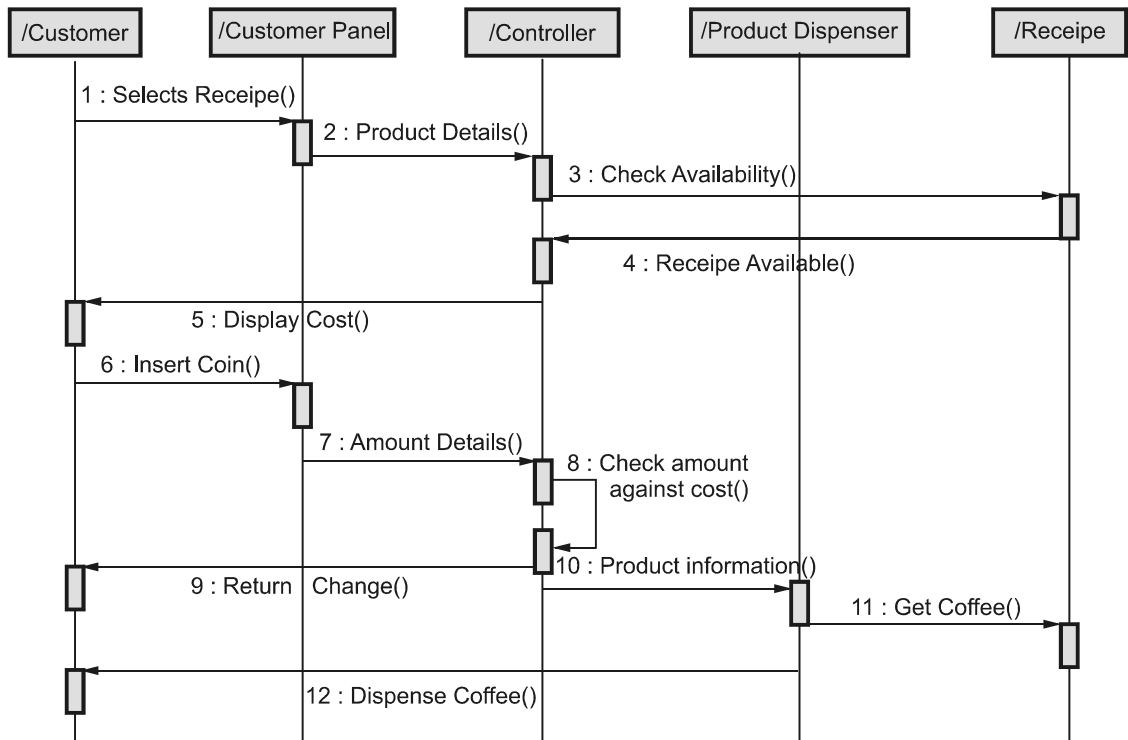


Fig. 2.8.12 Customer getting coffee from vending machine

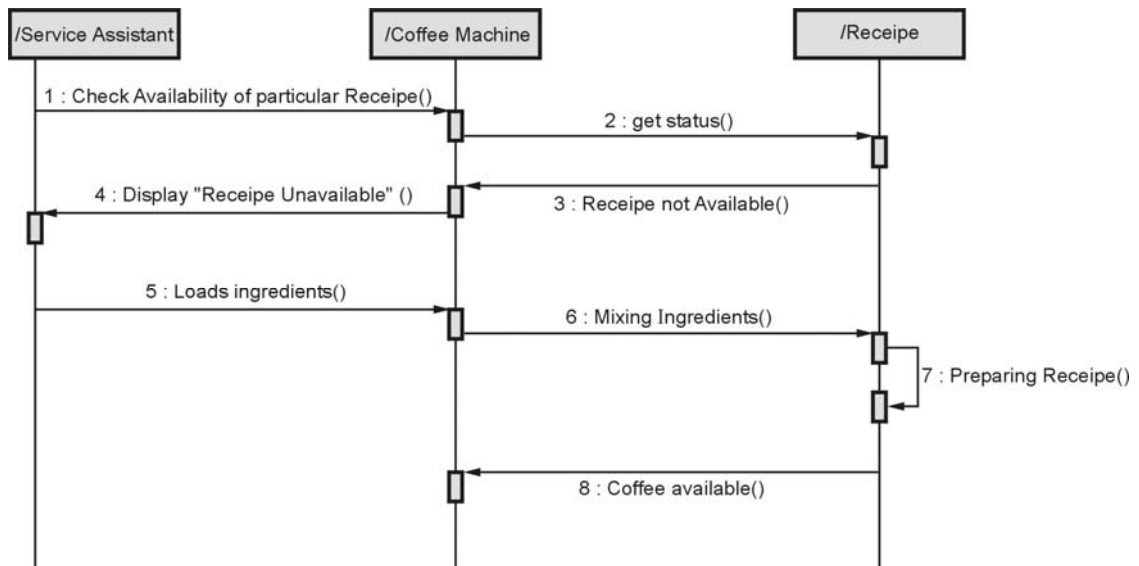


Fig. 2.8.13 Service assistant loads the ingredient and prepare coffee

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**

2.9 Requirement Validation

Requirement validation is a process in which it is checked that whether the gathered requirements represent the same system that customer really wants.

- In requirement validation the requirement errors are fixed. Requirements error costs are high so validation is very important. Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

- Requirement checking can be done in following manner
 1. **Validity** : Does the system provide the functions which best support the customer's needs?
 2. **Consistency** : Are there any requirements conflicts ?
 3. **Completeness** : Are all functions required by the customer included ?
 4. **Realism** : Can the requirements be implemented according to budget and technology ?
 5. **Verifiability** : Can the requirements be checked ?

Requirements validation techniques

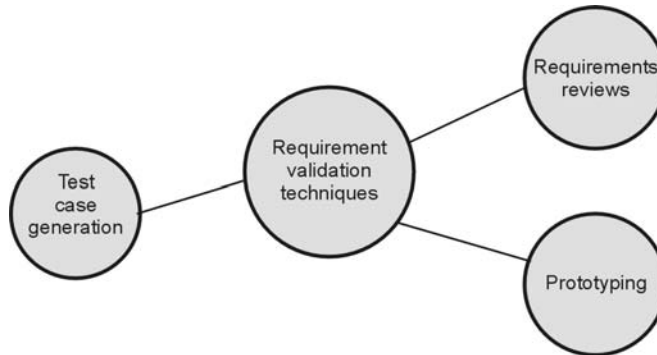


Fig. 2.9.1 Requirement validation technique

1. Requirements reviews : Requirement review is a systematic manual analysis of the requirements.

- The requirement review should be taken only after formulation of requirement definition. And both the customer and contractor staff should be involved in reviews.
- Reviews may be formal (with completed documents) or informal.
- Good communications should take place between developers, customers and users. Such a healthy communication helps to resolve problems at an early stage.

2. Prototyping : The requirements can be checked using executable model of system.

3. Test-case generation : In this technique, the various tests are developed for requirements. The requirement check can be carried out with -

- **Verifiability** : Is the requirement realistically testable ?
- **Comprehensibility** : Is the requirement properly understood ?
- **Traceability** : Is the origin of the requirement clearly stated ?
- **Adaptability** : Can the requirement be changed without a large impact on other requirements ?

2.10 Requirement Management**AU : May-16, Marks 12**

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

Why requirements get change ?

- Requirements are always incomplete and inconsistent. New requirements occur during the process as business needs change and a better understanding of the system is developed.
- System customers may specify the requirements from business perspective that can conflict with end user requirements.
- During the development of the system, its business and the technical environment may get changed.

2.10.1 Enduring and Volatile Requirements

Eduring requirements : These are the stable requirements that are derived from the core activity of the organization. These requirements are dependent upon the application domain of the software. For example: For banking system, transfer of money from one account to another is the enduring requirement.

Volatile requirements : For certain requirements if there is a possibility that those requirements may get changed during the development stage or after the system becomes operational, then such requirements are called volatile requirements.

Types of volatile requirements

1. Mutable requirements : If due to change in the environment, if the requirements get changed then such requirements are called as mutable requirements. For example: In the library management system, if the traditional library is turned into a digital library(containing e-books,on-line articles,e-learning or video conferencing facilities) then requirements for the library automation software will be changed.

2. Consequential requirements : Requirements that gets changed due to introduction of computer based systems, such requirements are called consequential requirements.

For example : In a tours and travel agencies, due to on-line ticket booking facilities requirements get changed. Such changed requirements are consequential requirements.

3. Emergent requirements : Due to customer's understanding of the system during the development stage certain requirements may get changed. Such types of requirements are called emergent requirements.

4. Compatibility requirements : Requirements which depend upon the specific system or business process in an organization are called compatible requirements

2.10.2 Requirements Management Planning

Following things should be planned during requirement process.

- Traceability is concerned with relationship between requirements their sources and the system design. Using traceability the requirement finding becomes easy.

Various types of traceability are

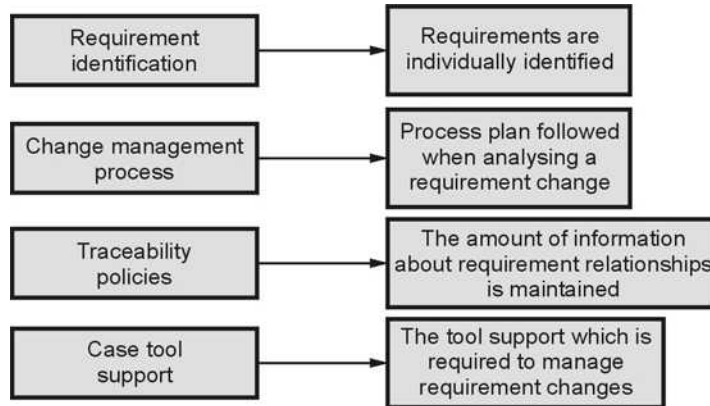


Fig. 2.10.1 Planning in requirement management process

- Source traceability** : These are basically the links from requirement to stakeholders who propose these requirements.
 - Requirements traceability** : These are the links between dependent requirements.
 - Design traceability** : These are the links from requirements to design.
- Traceability information is typically represented by a data structure **Traceability matrix**. If one requirement is dependant upon the other requirement then in that row-column cell 'D' is mentioned and if there is a weak relationship between the requirements then corresponding entry can be denoted by 'R'.

For example

Requirement ID	A	B	C	D	E	F
A		D			R	
B			D			
C				R		
D			D			R
E						
F	R			D		

For mentioning the traceability of small systems usually the traceability matrix is maintained.

- Case tool support is required for
 1. Requirement storage
 2. Change management
 3. Traceability management

2.10.3 Requirement Change Management

The requirement change management is a technique that can be applied to the processes in which requirements may get changed. The need for requirement change management is that even though the changes are made consistently in the requirements it is possible to incorporate those changes in a controlled manner. The requirement change management process can be applied in three stages.

1. Problem analysis and change specification
2. Change analysis and costing
3. Change implementation

Fig. 2.10.2 shows the stages of requirement change management process.

1. Problem analysis and change specification :

When requirement change request is made for some particular problem then the problem with older requirement is mentioned or sometimes simply change specification is given. Then first of all, problem analysis or change specification is analysed in order to **validate** the required change. If necessary the feedback of this analysis is given to the person who is demanding such change.

2. Change analysis and costing : Following actions are carried out in this stage :

- i) The effect of change is assessed using traceability information.
- ii) The cost of such change is estimated.
- iii) After getting the cost of changes the decision is made on whether to go for implementation of these changes or not.

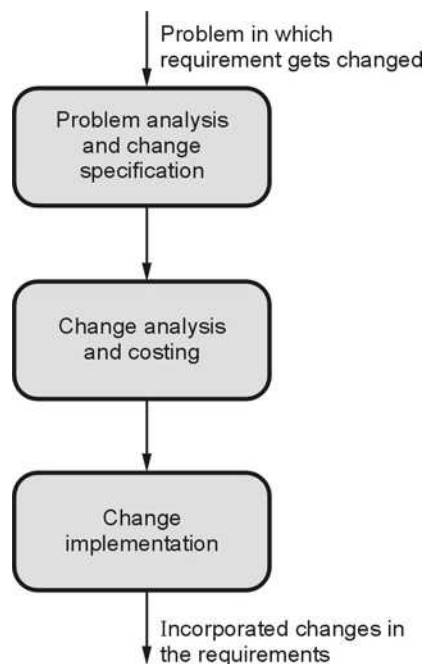


Fig. 2.10.2 Requirement change management process

3. Change implementation : Once it is decided to implement the proposed changes in the requirement, the requirement document has to be modified. The requirement document has to either re-written or re-organised. This can be achieved by making the modularity in the requirement specifications, so that it becomes easy to change individual section without affecting other part of requirement document.

Review Question

1. Describe the requirements change management process in detail.

AU : May-16, Marks 12

2.11 Structured System Analysis

AU : Dec.-13,15,16,17, 19, May-08,10,15,17,18,19, 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 and control flow diagrams are used to represent the functional model.
- Along with system modeling the specification can be written for the system using
 1. Process Specification
 2. Control Specification

2.11.1 Designing Data Flow Diagrams

- The data flow diagrams are used to model the information and function domain. Refinement of DFD into greater levels helps the analyst to perform functional decomposition.
- The guideline for creating a DFD is as given below –
 1. Level 0 DFD i.e. Context level DFD should depict the system as a single bubble.
 2. Primary input and primary output should be carefully identified.
 3. While doing the refinement isolate processes, data objects and data stores to represent the next level.
 4. All the bubbles (processes) and arrows should be appropriately named.
 5. One bubble at a time should be refined.
 6. Information flow continuity must be maintained from level to level.

Symbols used in DFD

Symbol	Notation
External Entity : External entities are objects outside the system, with which the system communicates. External entities are sources and destinations of the system's inputs and outputs.	
Process : A process transforms incoming data flow into outgoing data flow.	
Transition : It represents the flow of information from one entity to another.	
Data Store : Datastores are repositories of data in the system. They are sometimes also referred to as files or databases.	

- A simple and effective approach to expand the level 0 DFD to level 1 is to perform “grammatical parse” on the problem description. Identify nouns and verbs from it. Typically nouns represent the external entities, data objects or data stores and verbs represent the processes. Although grammatical parsing is not a foolproof but we can gather much useful information to create the data flow diagrams.

2.11.2 Rules for Designing DFD

1. No process can have only outputs or only inputs. The process must have both outputs and inputs.

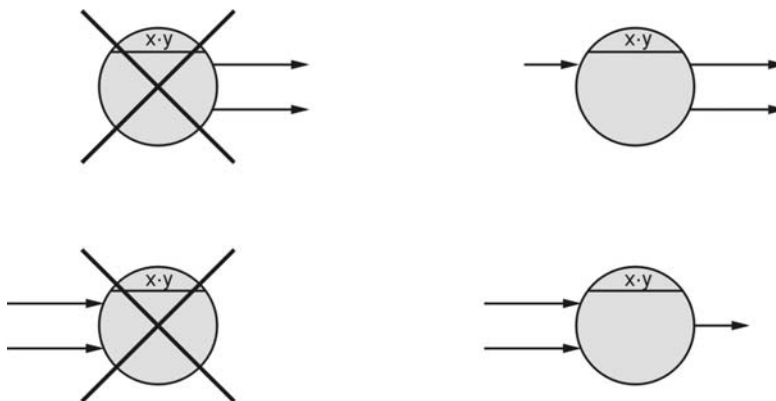


Fig. 2.11.1

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.

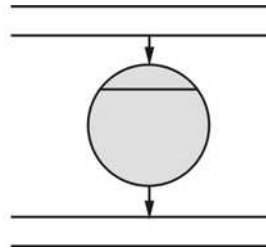
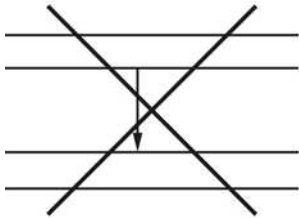


Fig. 2.11.2

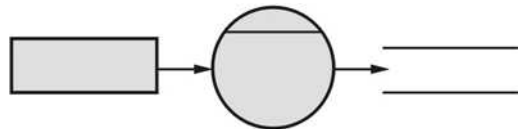
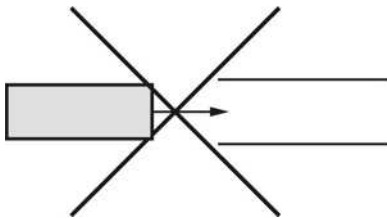


Fig. 2.11.3

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.

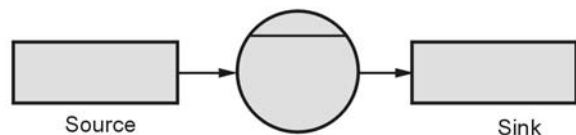
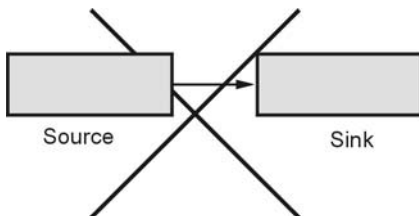


Fig. 2.11.4

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 2.11.1 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.

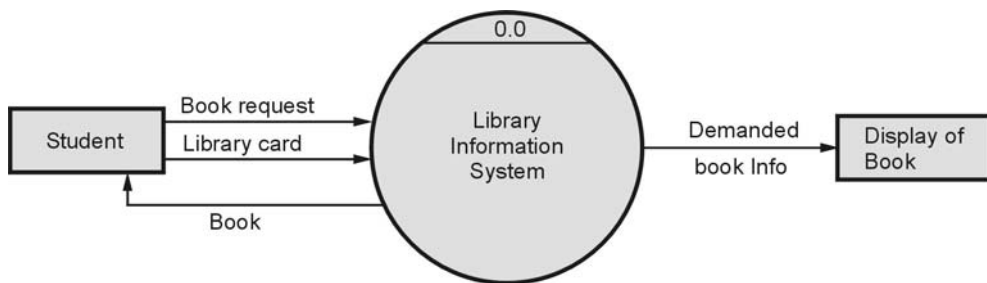


Fig. 2.11.5 Level 0 DFD (Context level DFD)

Level 1 DFD

In this level, the system is exposed with more processing details. The processes that need to be 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 shelves from which books can be located. This kind of information is represented by **data store**.

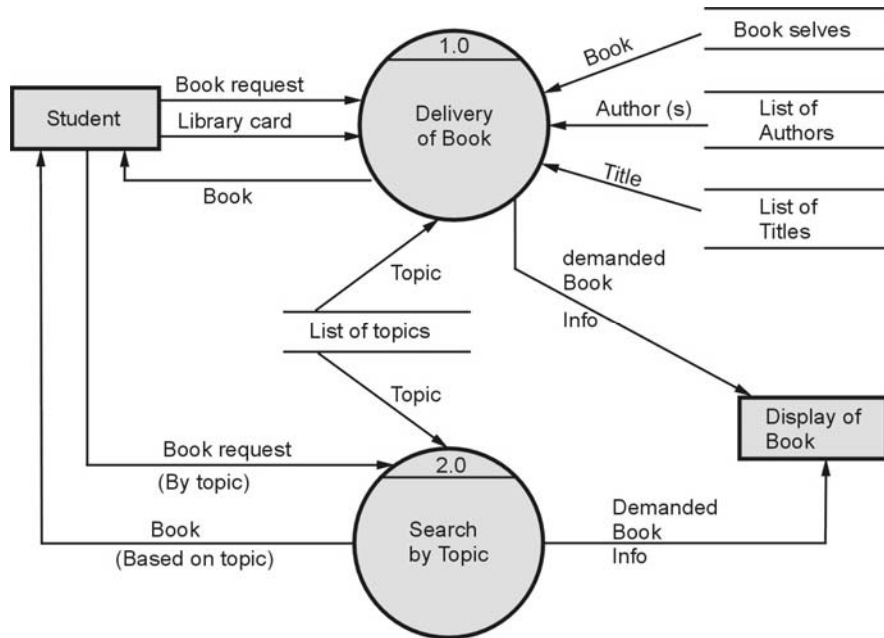


Fig. 2.11.6 Level 1 DFD

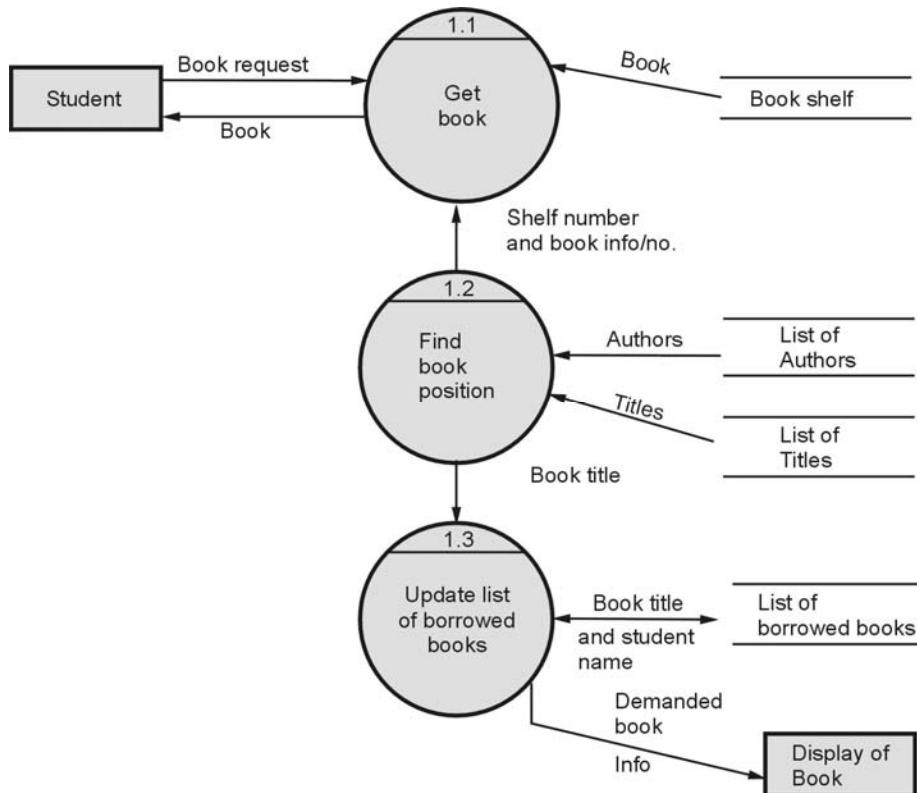


Fig. 2.11.7 Level 2 DFD

Example 2.11.2 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 depletion 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

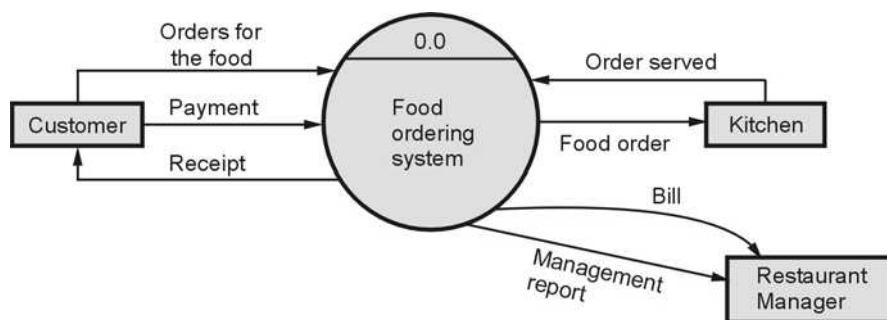
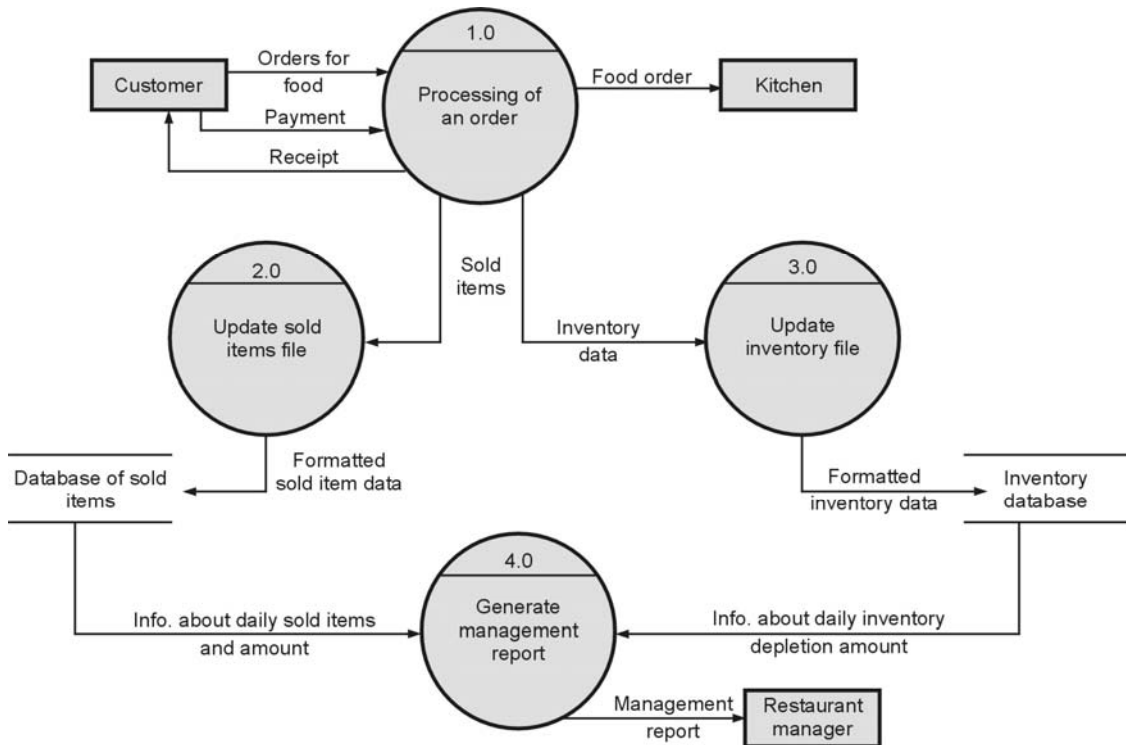


Fig. 2.11.8 Level 0 DFD (Context level DFD)

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. 2.11.9 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 depletion amount. This management report is given to restaurant manager.

In this DFD we will elaborate "processing of order".

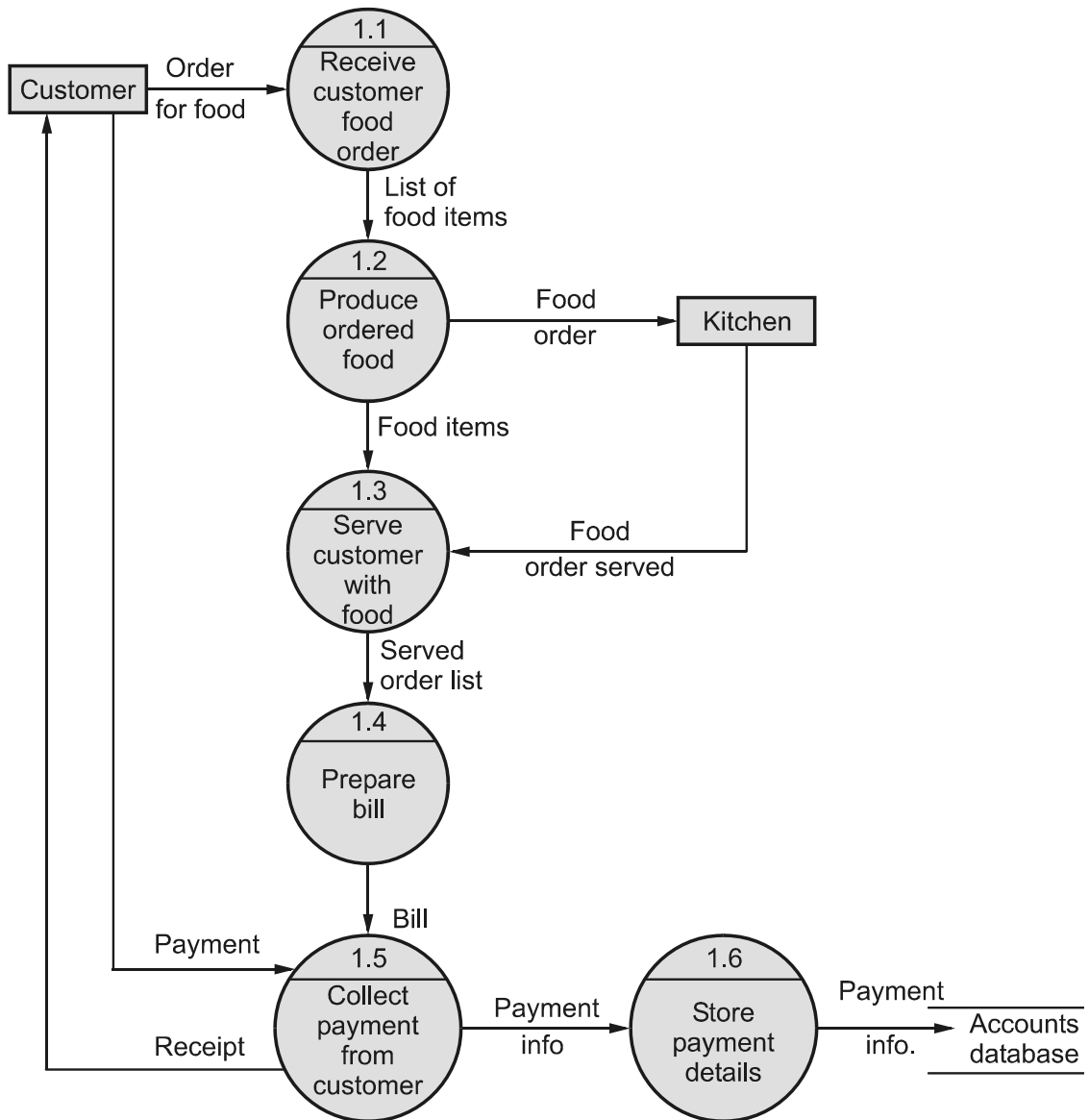


Fig. 2.11.10

The DFA for "update for inventory file " is shown in detail as follows –

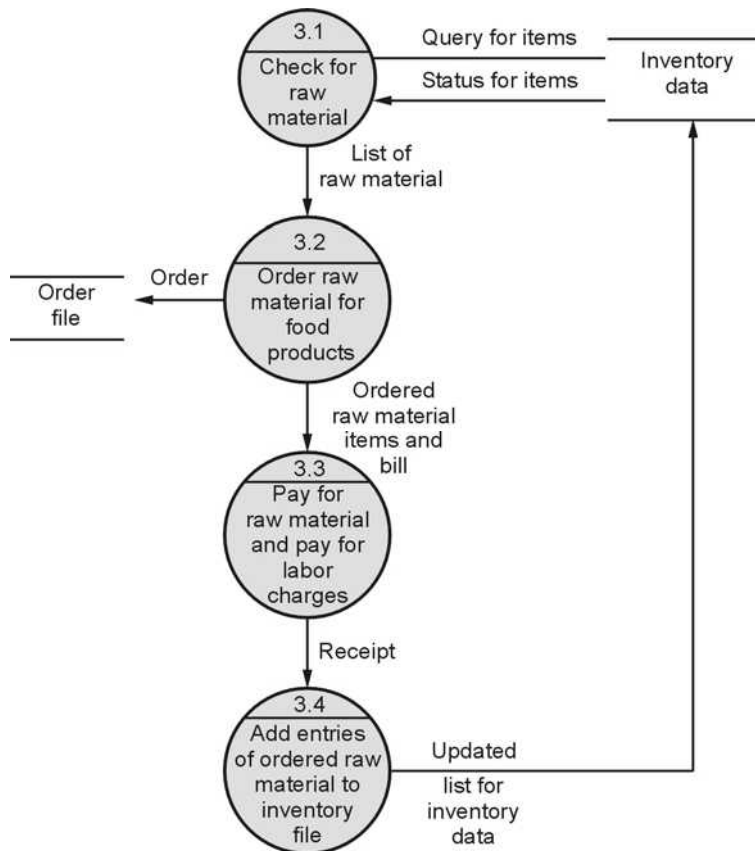


Fig. 2.11.11

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 sold 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. 2.11.12 on next page.)

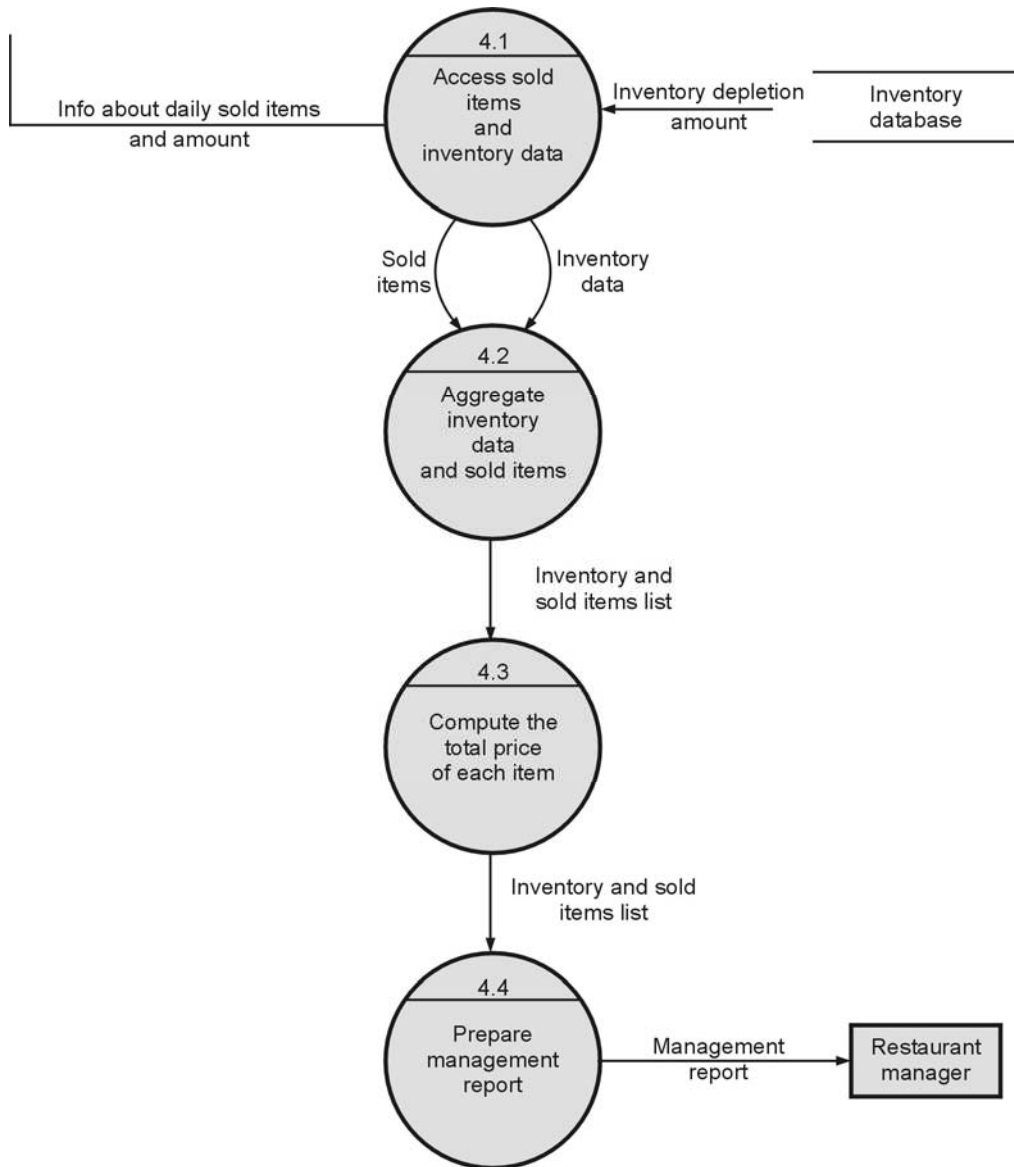


Fig. 2.11.12 Level 3 DFD

Example 2.11.3 Prepare a CFD 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

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. 2.11.14.

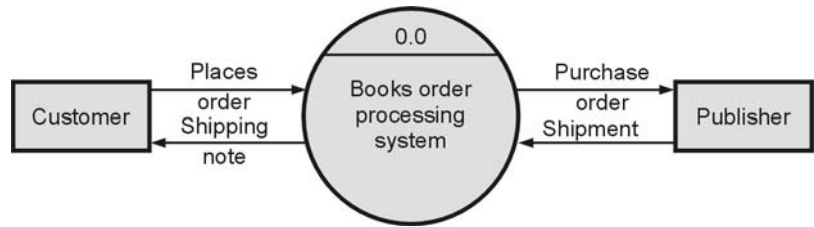


Fig. 2.11.13 Level 0 CFD

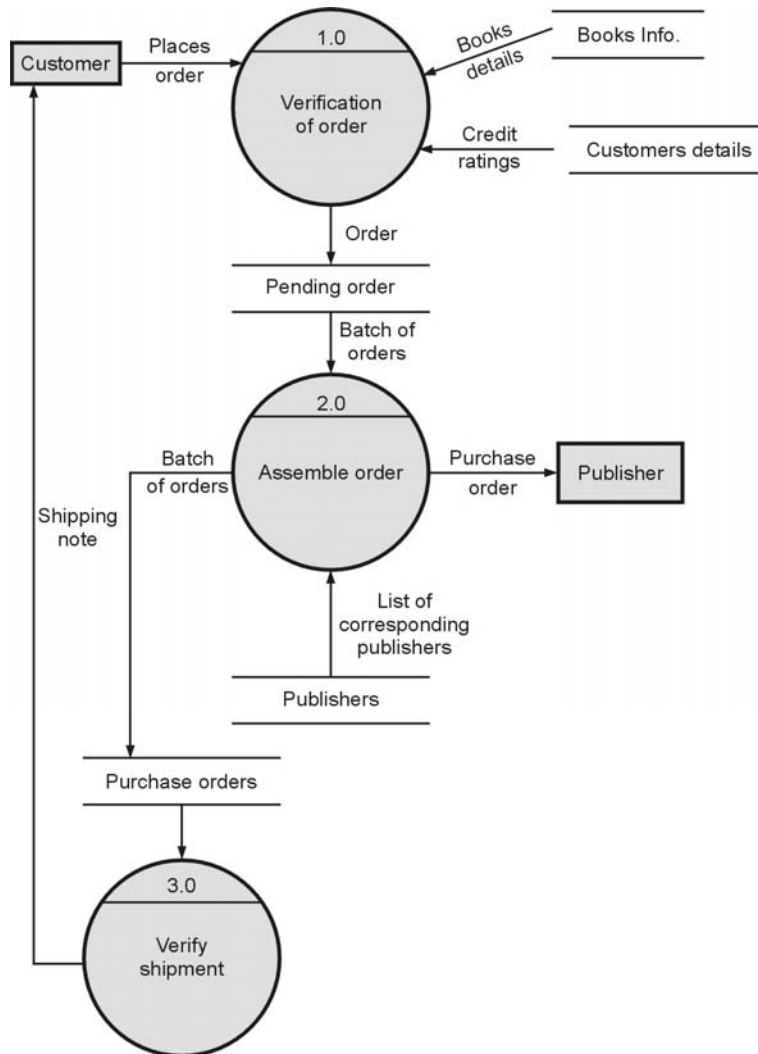


Fig. 2.11.14 Level 1 CFD

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 CFD for these details.

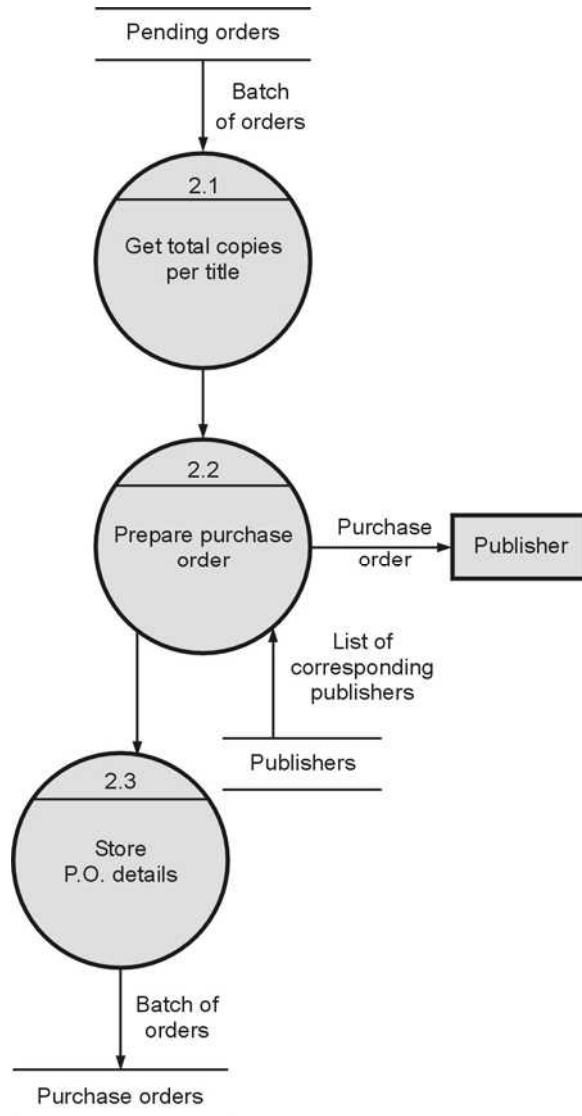


Fig. 2.11.15

Example 2.11.4 Design DFD for Hotel reservation system.

Solution : In a hotel reservation system, a **customer** can make online 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.

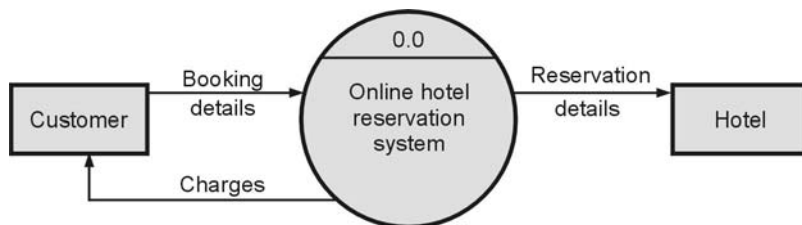


Fig. 2.11.16 Level 0 DFD

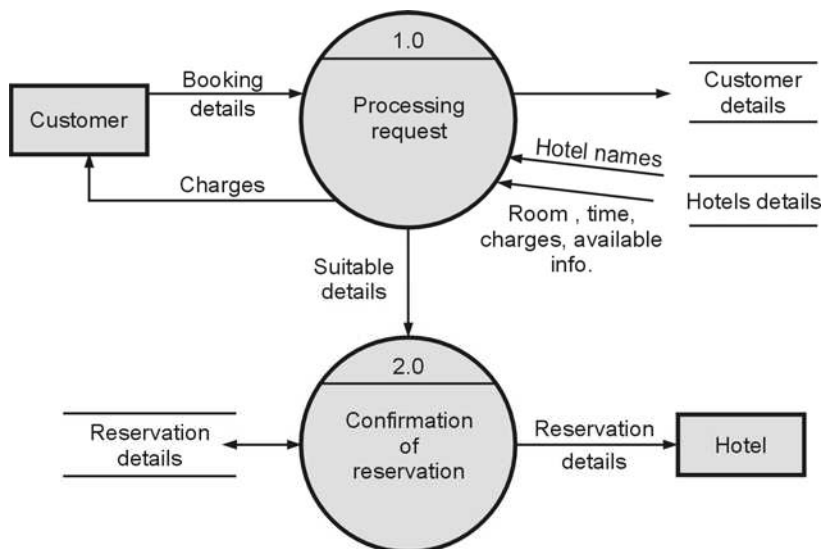


Fig. 2.11.17 Level 1 DFD

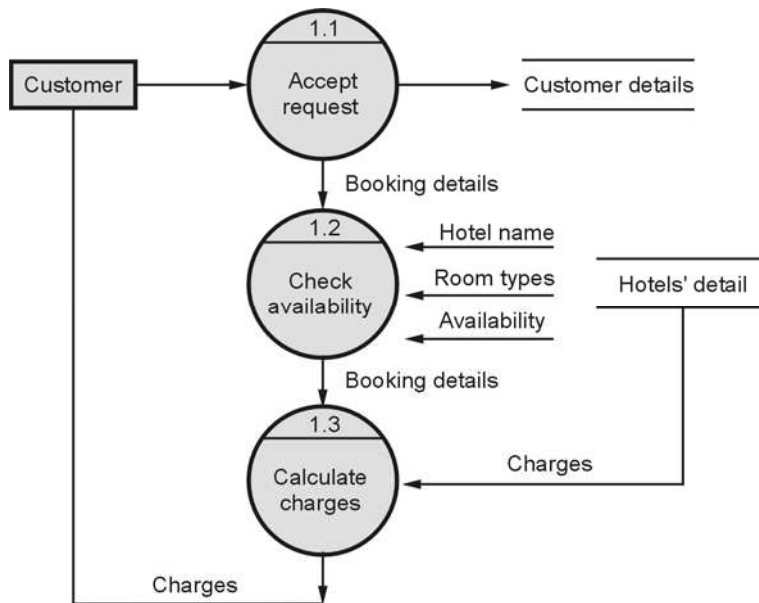


Fig. 2.11.18 Level 2 DFD

Example 2.11.5 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)

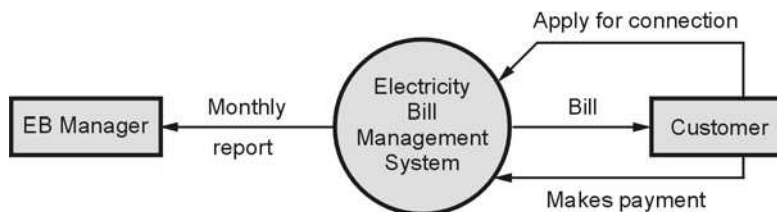


Fig. 2.11.19 Level 0 DFD

iii)

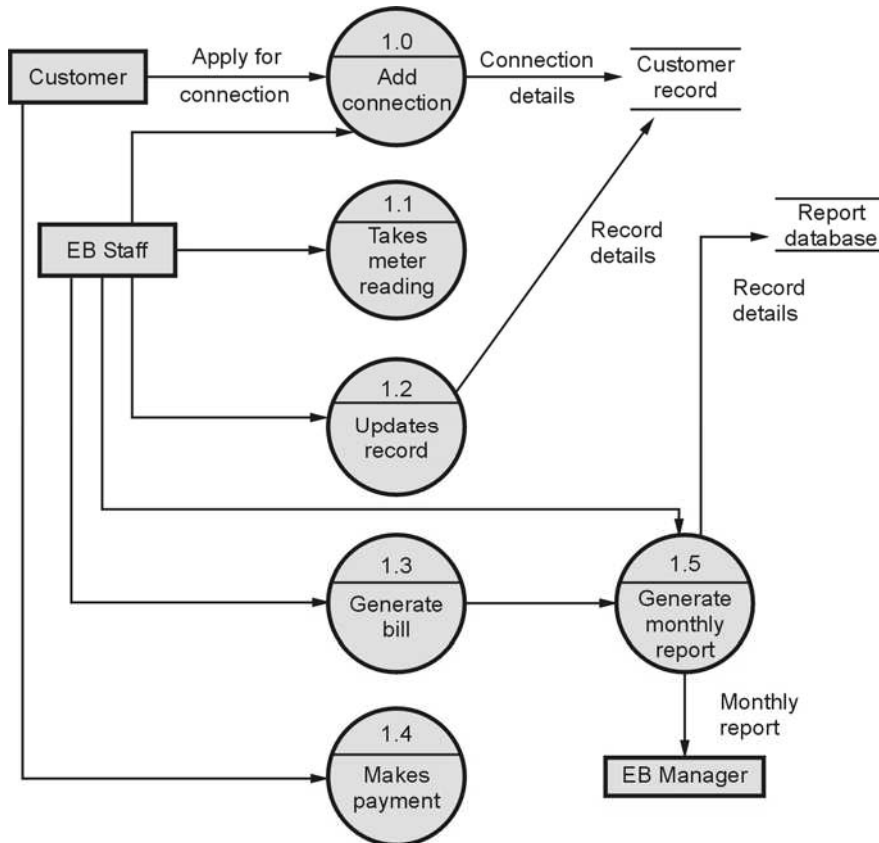


Fig. 2.11.20 Level 1 DFD

Example 2.11.6 Develop a dataflow diagram for burglar alarm system along with entity relationship diagram.

AU : May-10, Marks 8

Solution : 1) DFD :

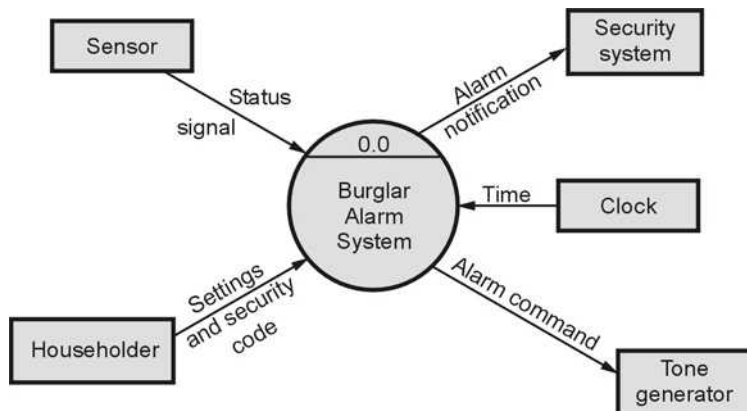
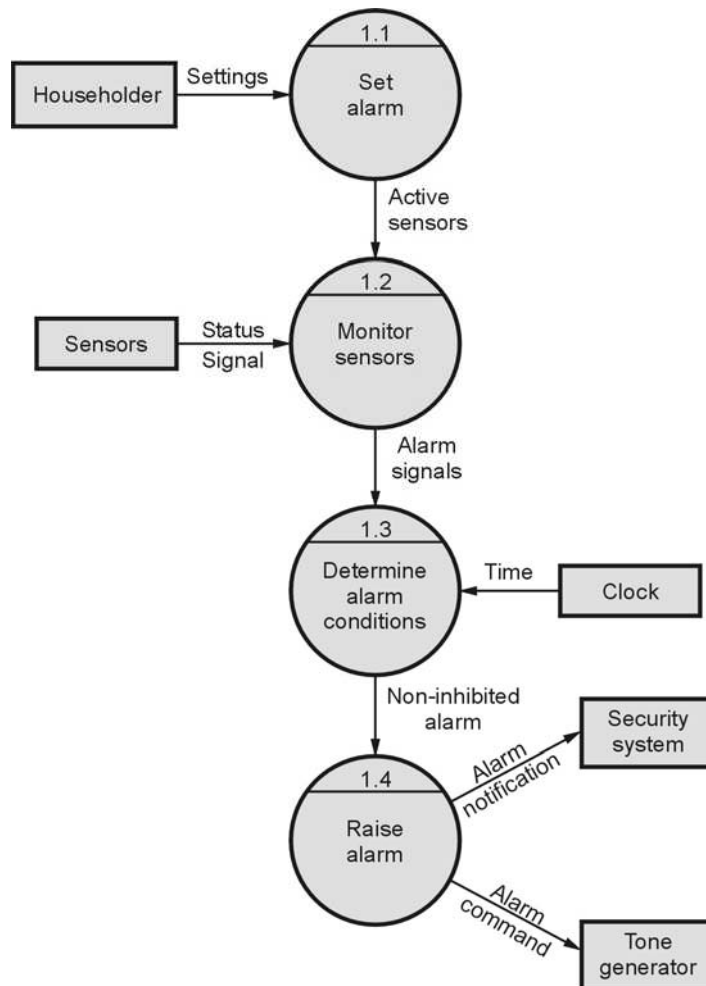


Fig. 2.11.21 Context diagram

**Fig. 2.11.22 Level 1 DFD**

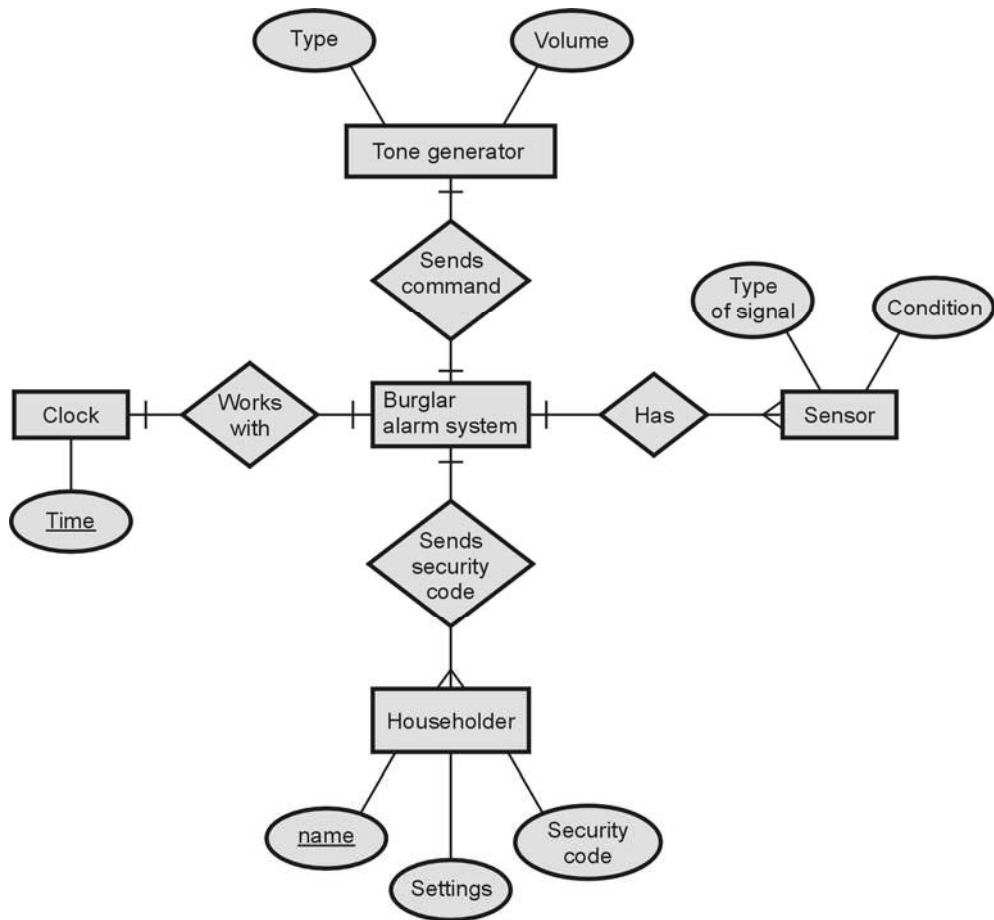
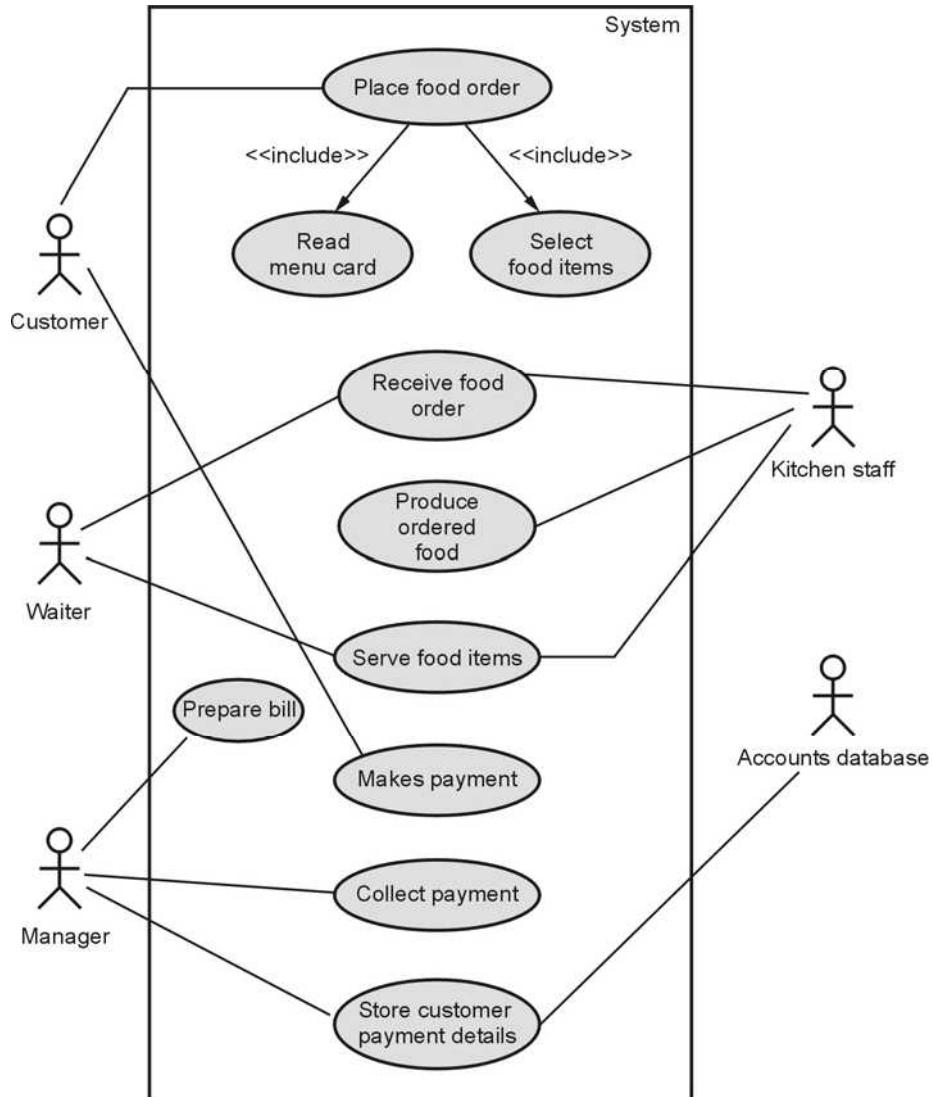


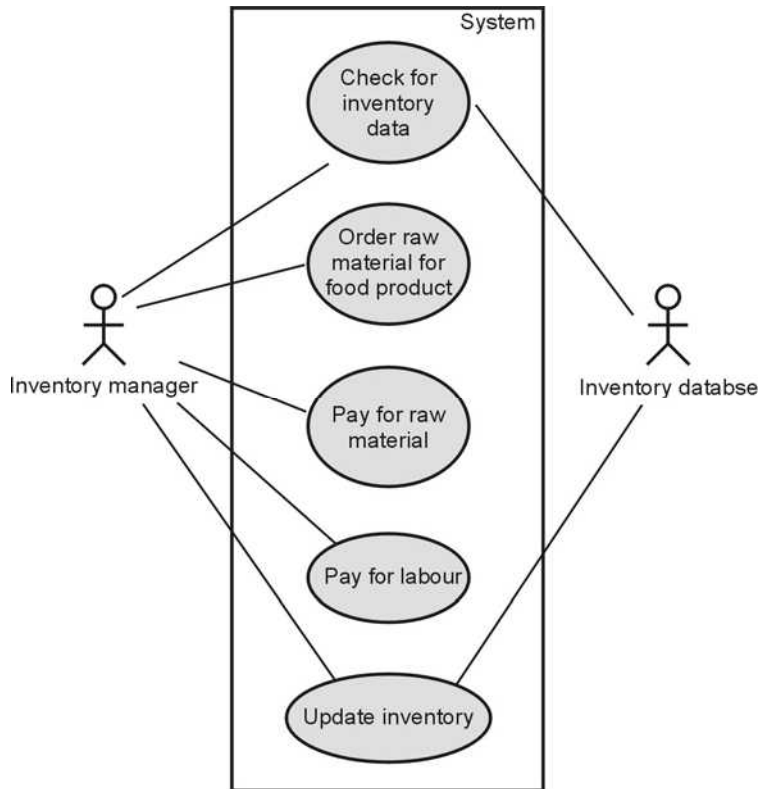
Fig. 2.11.23 ER diagram

Example 2.11.7 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 2.11.2.

Example 2.11.8 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 -

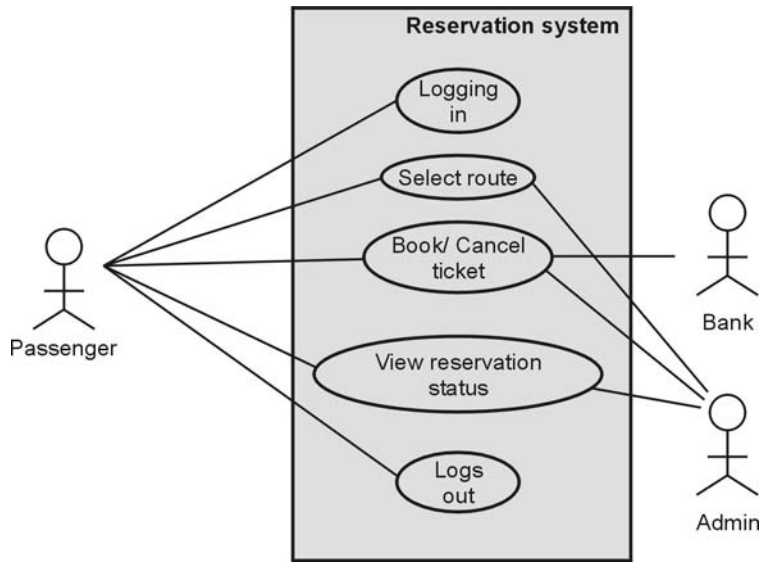


Fig. 2.11.24 Use case diagram

The level 0 and level 1 DFD can be as shown below –

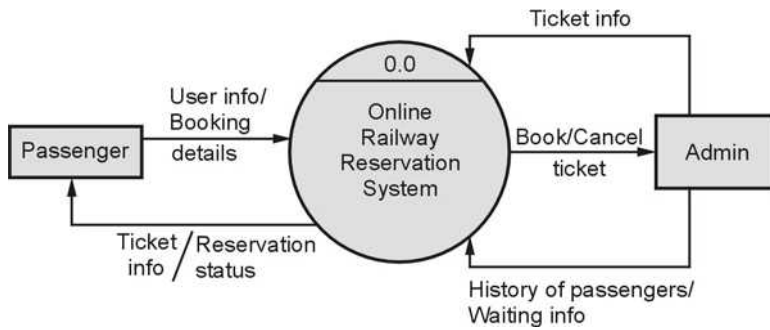
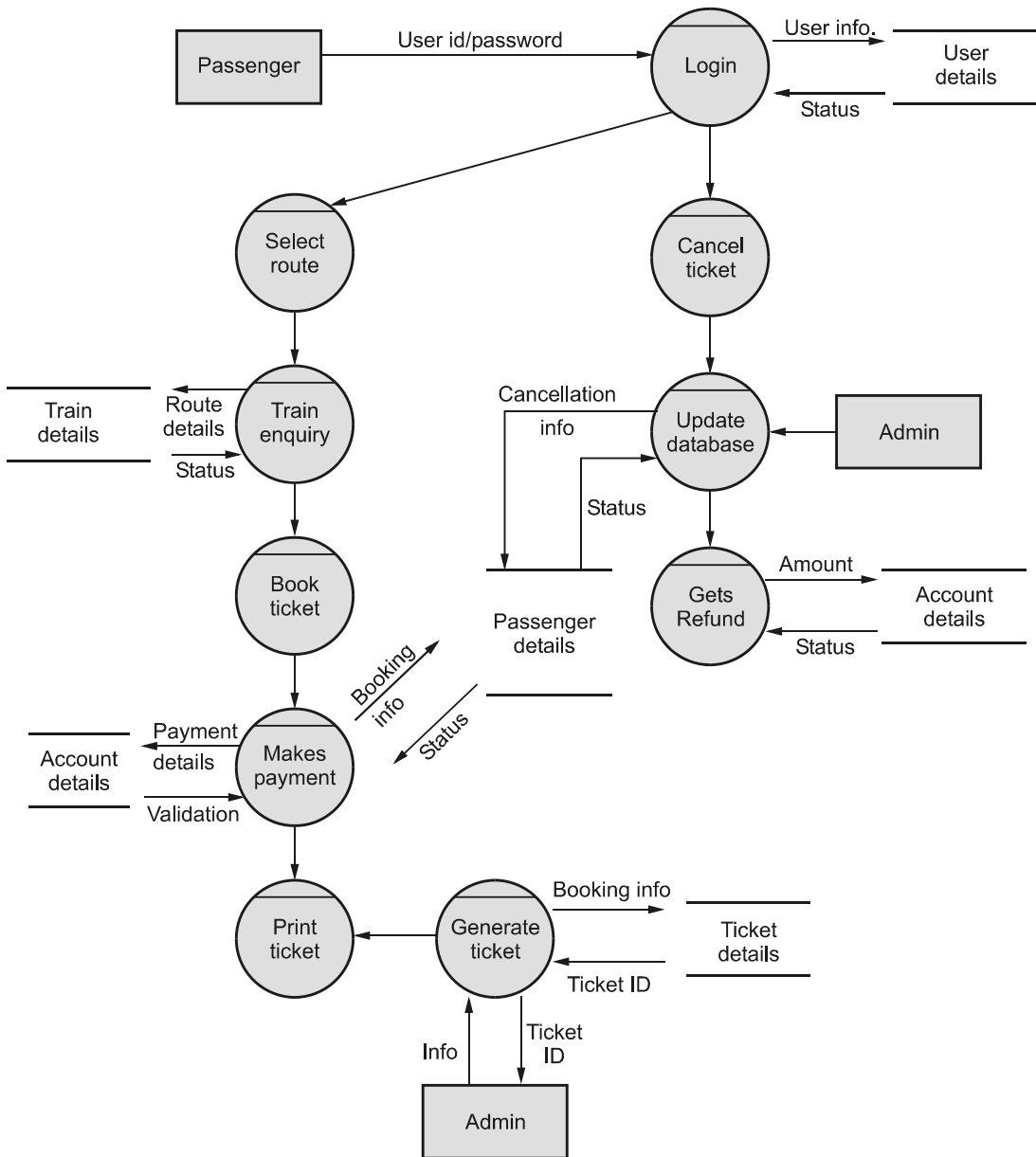


Fig. 2.11.25 Level 0 DFD

**Fig. 2.11.26 Level 1 DFD**

Example 2.11.9 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 :

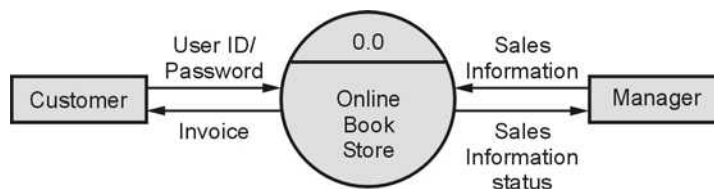


Fig. 2.11.27 Context level DFD

Refer Fig. 2.11.28 on next page.

Example 2.11.10 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. 2.11.29 on page 2 - 69.

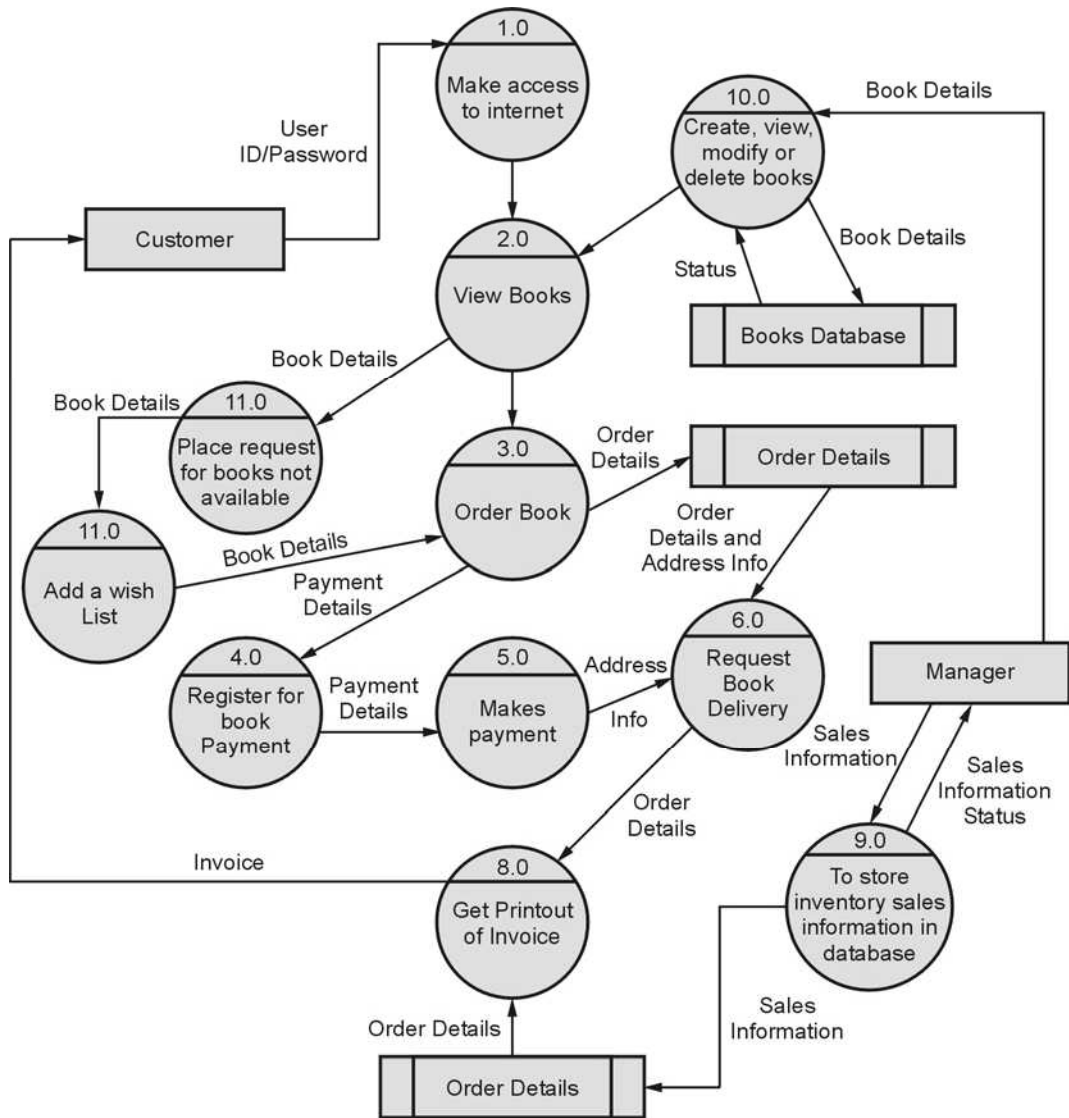


Fig. 2.11.28 Level 1 DFD

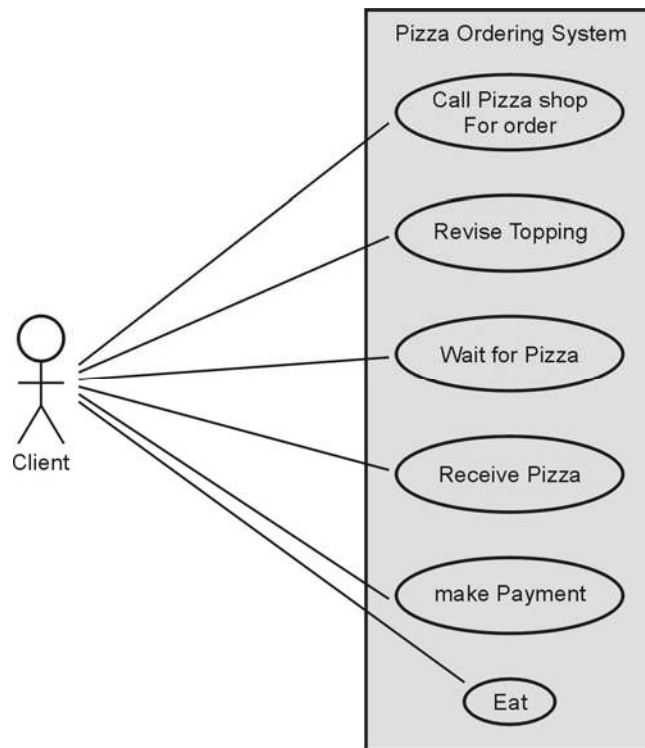


Fig. 2.11.29 Use case diagram for pizza ordering system

Example 2.11.11 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

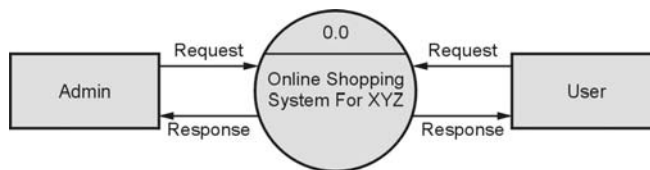


Fig. 2.11.31

Level 1 DFD

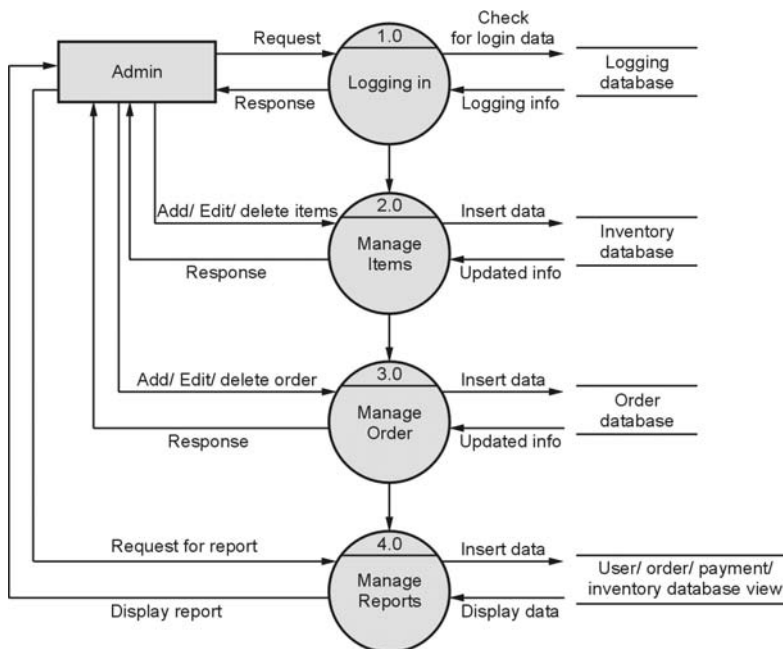
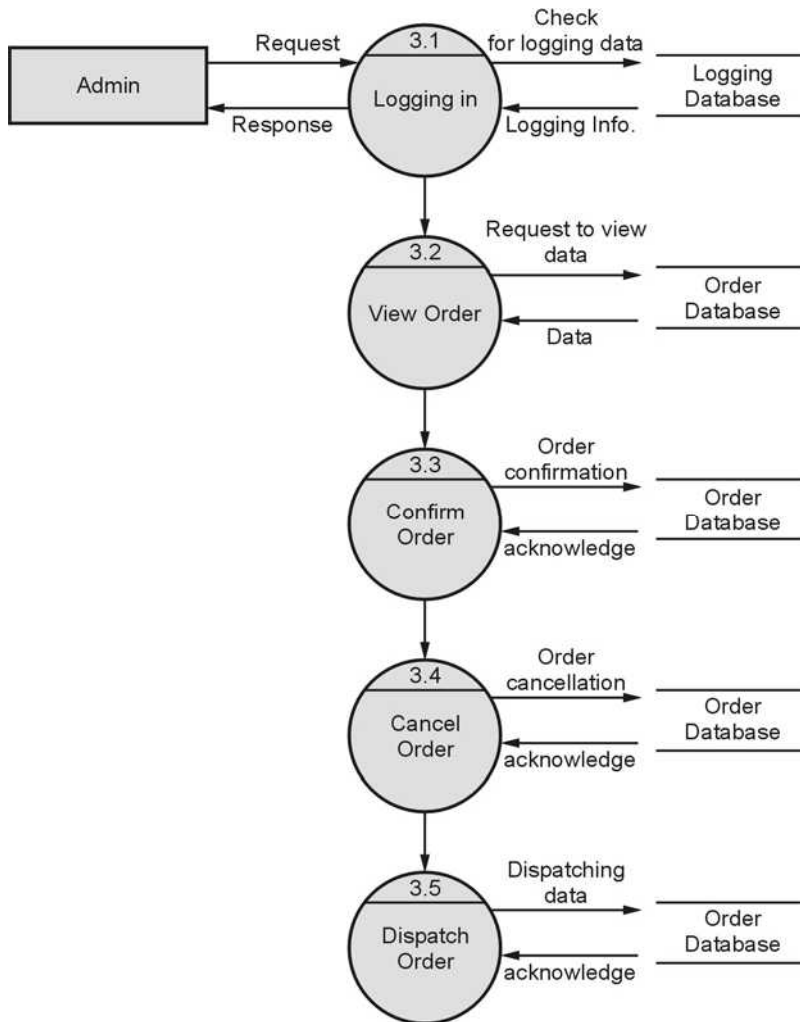
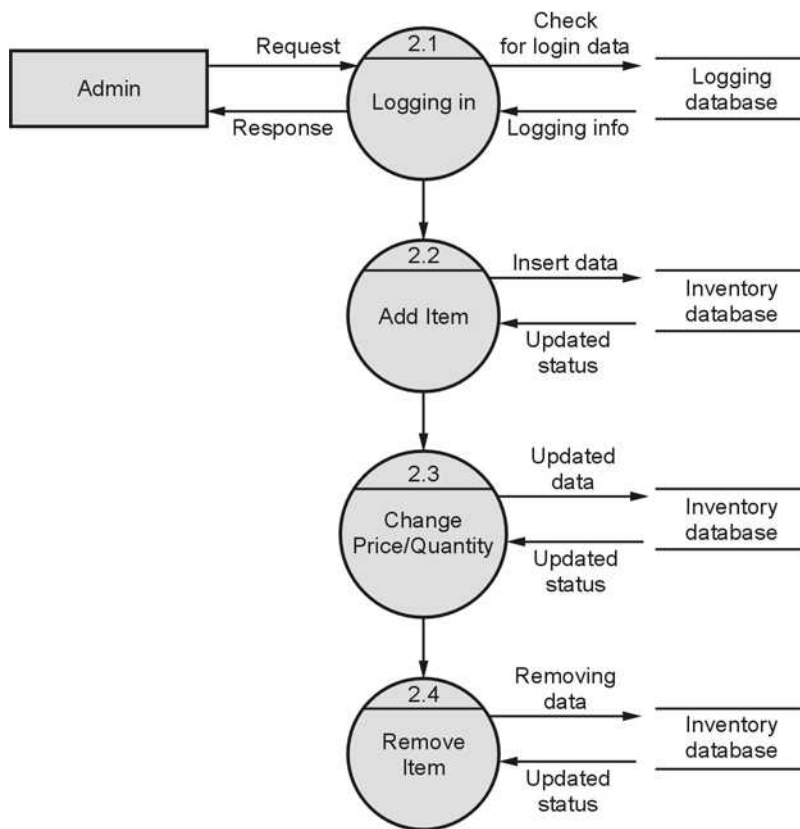
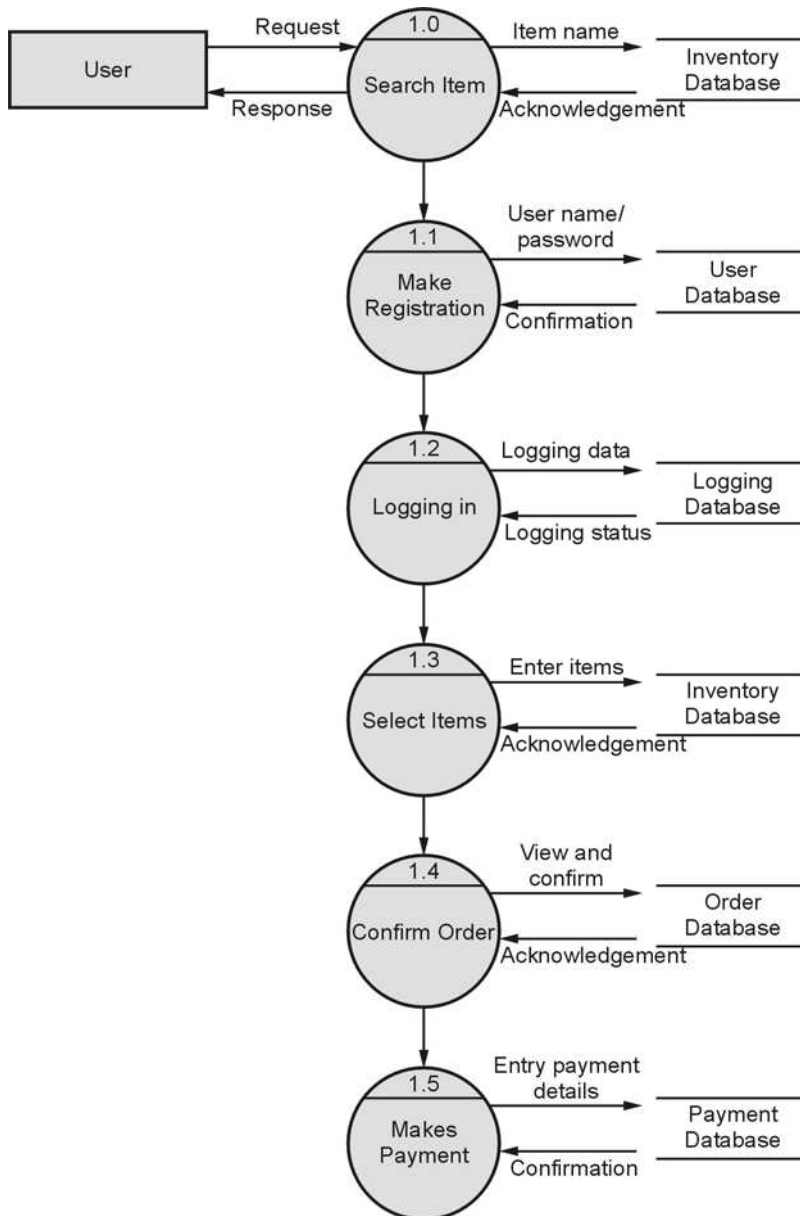


Fig. 2.11.32

Level 2 DFD : For manage order**Fig. 2.11.33**

Level 2 DFD : For manage items**Fig. 2.11.34**

Level 1 DFD : From user's perspective**Fig. 2.11.35**

Example 2.11.12 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.

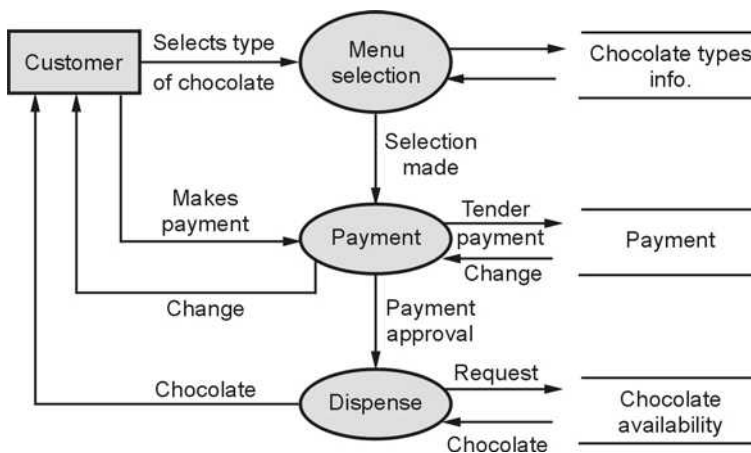
The Chocolate Vending Machine (CVM) system requirements are as follows : The CVM dispenses chocolates : (1) Very large Chocolates (VC) at ₹ 15, (2) Large Chocolates (LC) at ₹ 10 and (3) a Small Chocolates (SC) at ₹ 5. The vending machine only deals in coins. The CVM gives the proper change after the 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 then dispense the proper candy and change and display : Thank You!

AU : May-19, Marks 8 + 7

Solution :



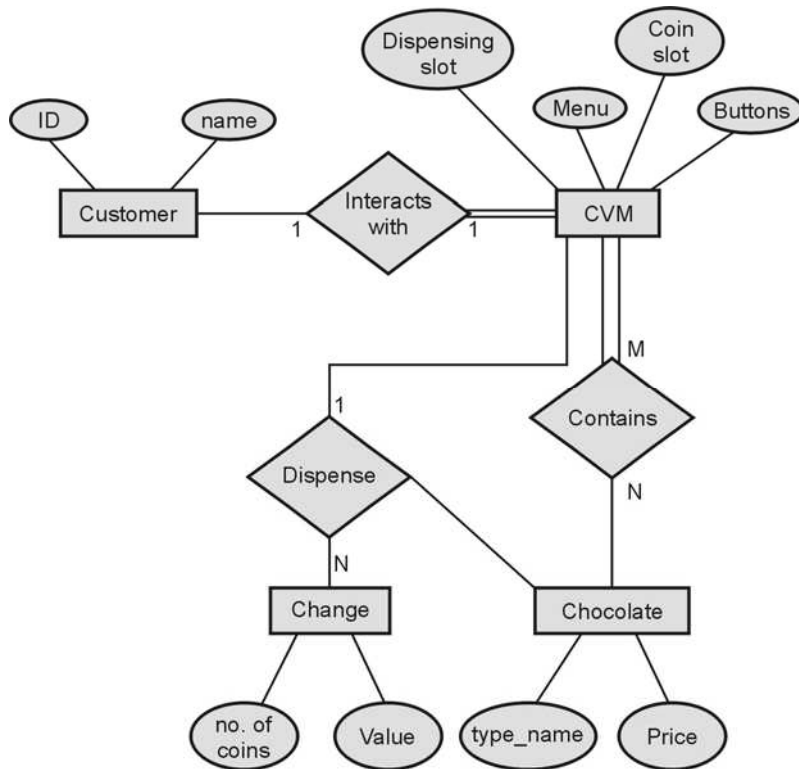
Level 0 DFD



Level 1 DFD

Entities in the system are :

- 1) Customer
- 2) CVM
- 3) Chocolate
- 4) Menu



2.11.3 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, CFD, 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

2.12 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)

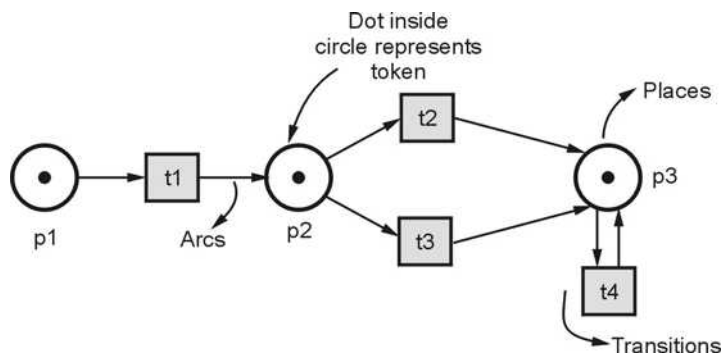


Fig. 2.12.1 Petri nets

- **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

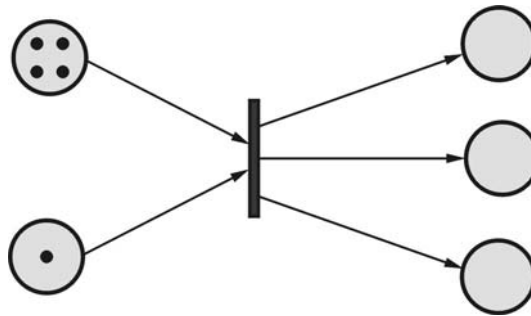


Fig. 2.12.2 Firing transitions

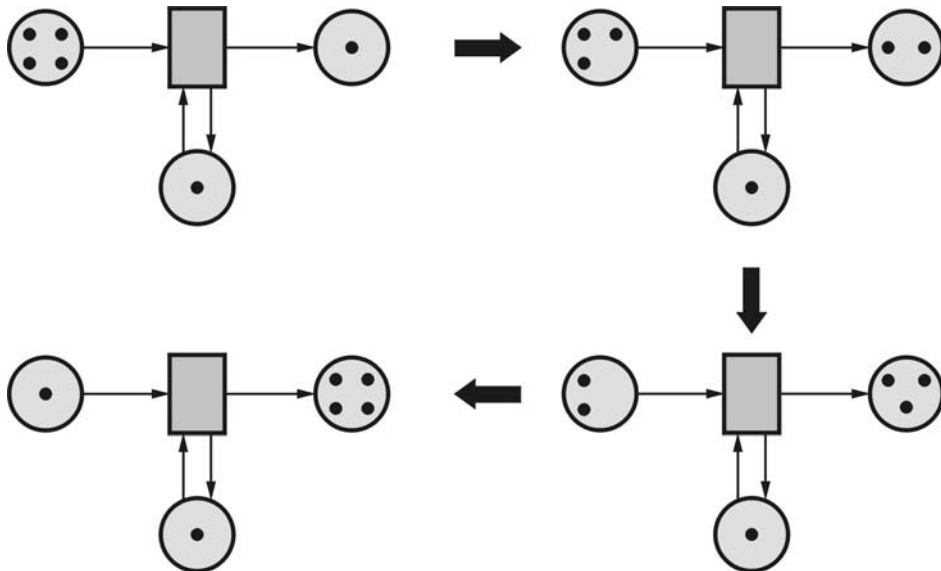


Fig. 2.12.3 Firing transitions in succession

Example 2.12.1 Draw a petrinet for a traffic light switching system.

Solution :

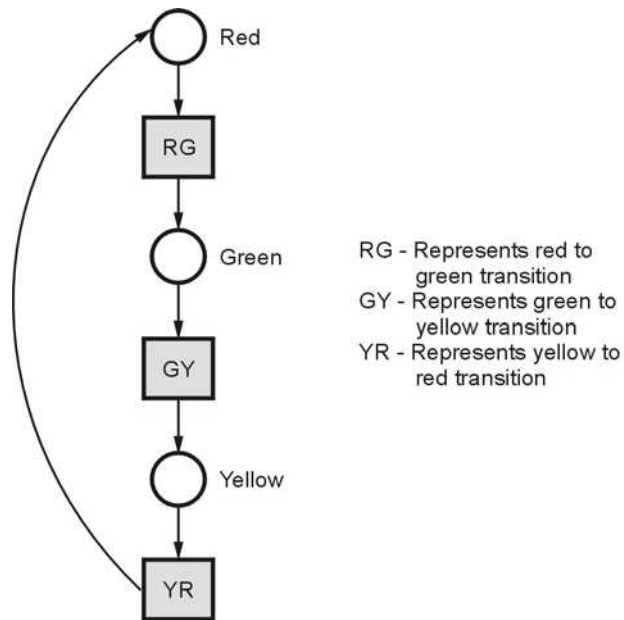
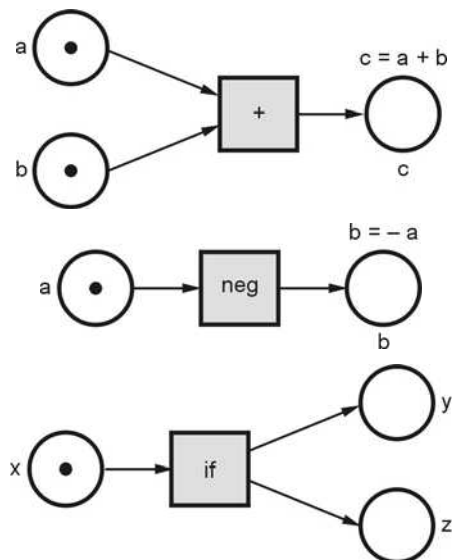


Fig. 2.12.4 Petrinet for traffic light signaling

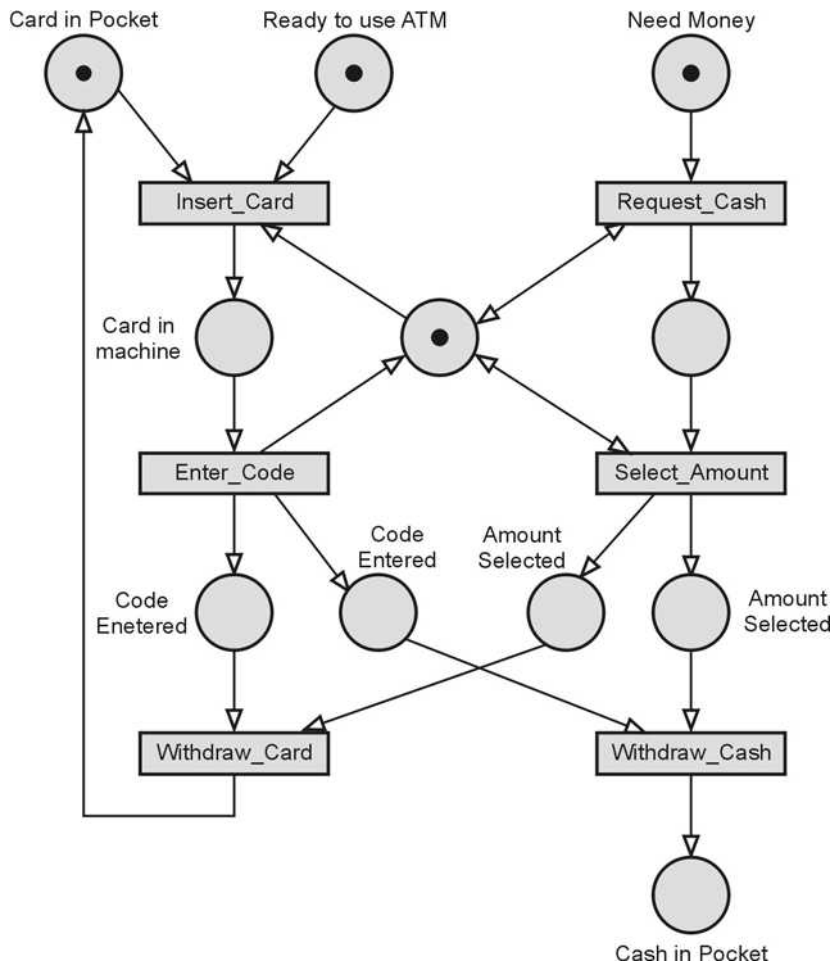
Example 2.12.2 Represent the operations i) $c = a + b$ ii) $b = -a$ iii) $x > 0$ then $y = x$ else $z = x$

Solution :



Example 2.12.3 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



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.

2.13 Data Dictionary

AU : Dec.-06, 08, 09, Marks 16

The data dictionary can be defined as an organized collection of all the data elements of the system with precise and rigorous definitions so that user and system analyst will have a common understanding of inputs, outputs, components of stores and intermediate calculations.

- The data models are less detail hence there is a need for data dictionary.
- Data dictionaries are lists of all of the names used in the system models.
- Descriptions of the entities, relationships and attributes are also included in data dictionary.
- Typically, the data dictionaries are implemented as a part of structured analysis and design tool
- The data dictionary stores following type of information

Name	Description
Name	The primary name of data or control item, the data store or external entity.
Alias	Other name used for the Name
Where-used or how is used	It describes where the data or control item is used. It also describes how that item is used(that means input to the process, output to the process)

The notations used in data dictionary are

Data construct	Notation	Meaning
Composition	=	Is composed of
Sequence	+	And
Selection	[]	Or
Repetition	{ }n	Repetition for n times
	()	Optional data
	...	Commented information

For example :

Consider the some reservation system. The data item “passenger” can be entered in the data dictionary as

name :	passenger
alias :	none
where used/	
how used :	passenger's query (input) ticket (output)
description :	
passenger =	passenger_name + passenger_address
passenger_name =	passenger_last_name + passenger_first name + passenger_middle_initial
passenger_address =	local_address + community_address + zip_code
local_address =	house_number + street_name + (apartment_number)
community_address =	city_name + [state_name province_name]

- The data dictionary defines the data items unambiguously
- One can give the detailed description of data items using data dictionary
- For large computer based system the size of data dictionary is very huge. It is also complex to maintain such a data dictionary manually. Hence automated (CASE) tools can be used to maintain the data dictionary.

Advantages :

1. Data dictionary support name management and avoid duplication
2. It is a store of organisational knowledge linking analysis, design and implementation.

Review Question

1. What is data dictionary ? Give an example.

AU : Dec.-06, Marks 8 , IT, Dec.-09, Marks 16; IT, Dec.-08, Marks 2 + 8

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 Distinguish between expected requirements and exciting requirements.**AU : IT, May-04**

Ans. : Expected requirements are essentially basic functions or features that customers normally expect from the product. Expected requirements are usually invisible unless they become visible when they are unfulfilled. Their absence will cause customer's dissatisfaction. For example : ease of software installation.

Exciting requirements are sort of "out of ordinary" functions or features of a product. Exciting requirements are also usually invisible unless they become visible when they are fulfilled and result in customer satisfaction; they do not leave customers dissatisfied when left unfulfilled. For example : In word processing software if the product contains the number of page layout capability then that becomes an exciting requirement.

Q.6 What is data dictionary ? Where it is used in software engineering ?**AU : IT, Dec.-04 , May-09, 13**

Ans. : The data dictionary represents all of the names used in the system models. The descriptions of the entities, relationships and attributes are also included in data dictionary. The data dictionary is used during structured analysis for providing the detail information of data objects being used in DFD. Nowadays, the data dictionary is implemented as a part of CASE.

Q.7 How requirements are collected for “user-interface” of software ?**AU : IT, Dec.-04**

Ans. : While designing the user-interface of software the requirement collection can be done by focusing on the profile of user who will interact with the system. Skill level, business understanding and general grasping to the new system are recorded. Users can be categorized into different categories and for each category of user requirements are elicited. The most commonly used elicitation technique is to conduct meeting software developer and the end user.

Q.8 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.9 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 break 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.10 Create a data dictionary that provides with a precise definition of telephone number, it should indicate, where and how this data item is used and why supplementary information that is relevant to it ?

AU : CSE, Dec.-07

Ans. : The data dictionary for telephone number is as given below.

name :	telephone number
aliases :	none
where used/how used :	assess against the set-up(o/p) dialled number (i/p)
description :	
	telephone number=[local number+STD]
	local number=prefix+access number
	STD=+91+area code+local number
	area code=[020 022]
	prefix=* a four digit number that starts with 2*
	access number=*any four digit number*

For example : Consider a telephone number +91 (020)24495496 of **Technical Publications** in Pune.

Here **+91** then followed by the **area code** 020 then comes **prefix=2449** which starts with 2 and finally the access **number** is 5496. This is the number in city Pune (the area code for Pune is 20). If we want to dial this number in city Pune itself then the local number is 24495496.

Q.11 What is the major distinction between user requirements and system requirements ?

AU : IT, May-08, 16, Marks 4

Ans. : The user requirements describe both the functional and non-functional requirements in such a way that they are understandable by the users who do not have detailed technical

knowledge. On the other hand the system requirements are more detailed specifications of system functions, services and constraints than user requirements.

The user requirements are specified using natural languages, tables and diagrams because these representations can be understood by all users.

The system requirements can be expressed using system models.

Q.12 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.13 How do we use the models that we create during requirement analysis ?

AU : CSE, Dec.-08

Ans. : Various models and their use during requirement analysis phase are given as below -

1. **Data model** - The entity relationship model is created during the data modelling. The ERD defines the relationship between data objects.
2. **Functional model** - The data flow diagrams are created in this modelling. The DFD depicts the information flow in three phases input, output and process.
3. **Behavioural model** - The behavioural models are used to describe the overall behaviour of a system. The state transition diagrams are used to represent the behavioural model.

Q.14 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.15 Define functional and non functional requirements. (Refer sections 2.2.1 and 2.2.2)

AU : Dec.-10, May-14

Q.16 What do requirement process involve ?

AU : May.-12

Ans. : Requirements engineering process involves elicitation, analysis and validation.

Q.17 List two advantages of traceability tables in the requirements management phase

AU : Dec.-13

Ans. : The two advantages of traceability tables are as given below -

1. The traceability matrix ensures the coverage between different types of requirements.
2. The quick change impact analysis can be made using traceability matrix.

Q.18 List the notations used in data flow models. (Refer section 2.11.1)

AU : May-14

Q.19 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.20 What is data dictionary ?

AU : Dec.-14, 15, 16, May-17

Ans. : The data dictionary represents all of the names used in the system models. The descriptions of entities, relationships and attributes are also included in the data dictionary.

Q.21 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.

Q.22 How are the requirements validated ?

AU : May-15

Ans. : Requirements are validated with the help of following validation techniques -

1. **Requirements reviews :** It is a manual analysis of requirements with the participation of customer and contractor staff.
2. **Prototyping :** The requirements can be checked using executable model of system.
3. **Test case generation :** In this case, various tests are developed for requirements.

Q.23 Define feasibility study and list the types.

AU : Dec.-15

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.24 List the characteristics of good SRS.

AU : May-16

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.25 What are the linkages between data flow and E-R diagram ?

AU : May-16

Ans. : 1) Both the Data Flow Diagrams (DFD) and ER diagram are system modeling techniques used during structured system analysis.

2) The ER diagram is used to represent data model while data flow diagrams are used to represent the functional model.

Q.26 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.27 What is the purpose of petrinet ?

AU : May-17

OR Define a petrinet.

AU : Dec.-19

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.28 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.29 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. :

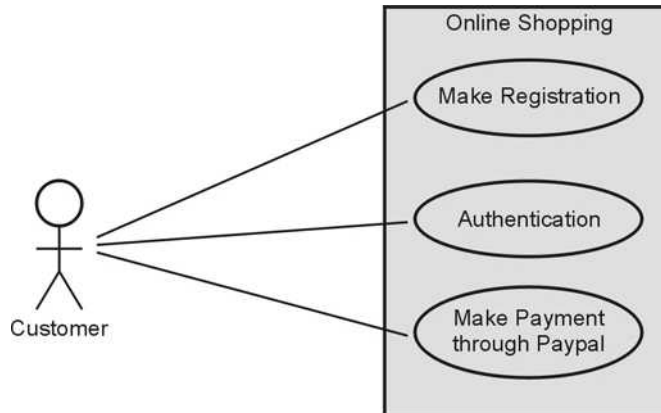


Fig. 2.1

Q.30 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.31 Draw the context flow graph of a ATM automation system.

AU : Dec.-17

Ans. :

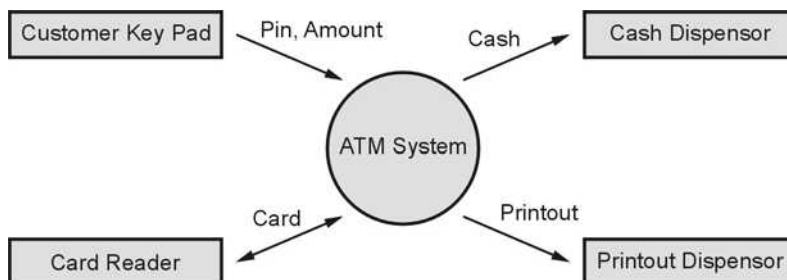


Fig. 2.2

Q.32 What are the various types of traceability in software engineering ?

AU : May-17

Ans. : Various types of traceability in software engineering are -

(1) Source traceability (2) Requirements traceability (3) Design traceability

Q.33 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.34 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.35 Differentiate : Functional and Non-functional requirements. (Refer section 2.23)

AU : May-19



3

Software Design

Syllabus

Design process - Design concepts - Design model - Design heuristic - Architectural design - Architectural styles, Architectural design, Architectural mapping using data flow - User interface design : Interface analysis, Interface design - Component level design : Designing class based components, Traditional components.

Contents

3.1 Introduction		
3.2 Design ProcessMay-05.....	Marks 8
3.3 Design ConceptsMay-06,08,16,19	
Dec.-05,17	Marks 16
3.4 Design Model		
3.5 Design HeuristicMay-13,16	Marks 8
3.6 Introduction to Architectural Design.....	May-16	Marks 4
3.7 Software ArchitectureMay-17,19	Marks 13
3.8 Data DesignMay-16	Marks 8
3.9 Architectural StyleMay-07,18,Dec.-19	Marks 16
3.10 Architectural DesignDec.-03,19	Marks 16
3.11 Architectural Mapping using Data Flow	Dec.-16, May-18	Marks 16
3.12 Concept of User Interface Design.....	May-07,13,16,	
Dec.-08,09,15,16,17.....	Marks 16
3.13 Component Level Design.....	May-15,16,17,18,	
Dec.-15,16,19.....	Marks 16

Two Marks Questions with Answers

3.1 Introduction

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**.

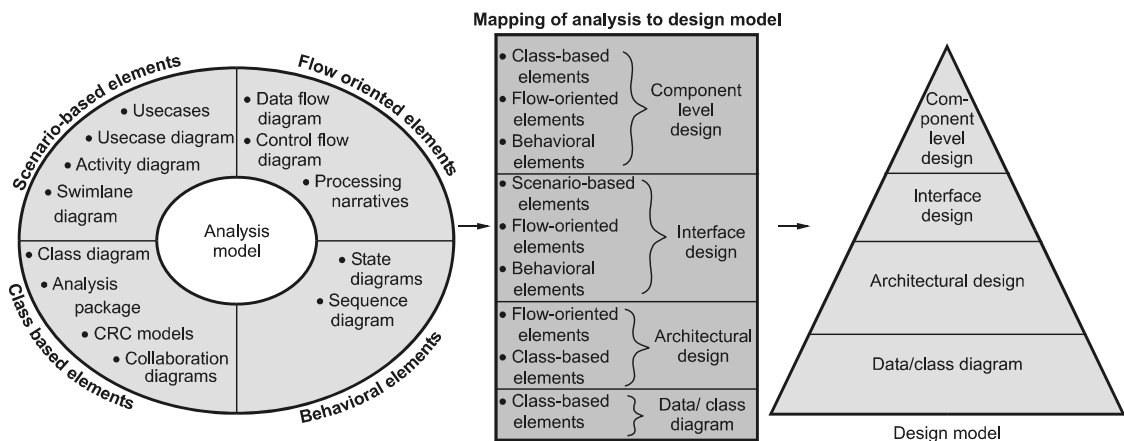


Fig. 3.1.1 Translating analysis model into the design model

- 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 possess **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, Reliability, Performance and Supportability) 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, 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 -

- | | |
|--|--|
| <ul style="list-style-type: none"> • Abstraction • Modularity • Architecture • Refinement • Pattern | <ul style="list-style-type: none"> • 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 correlation 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

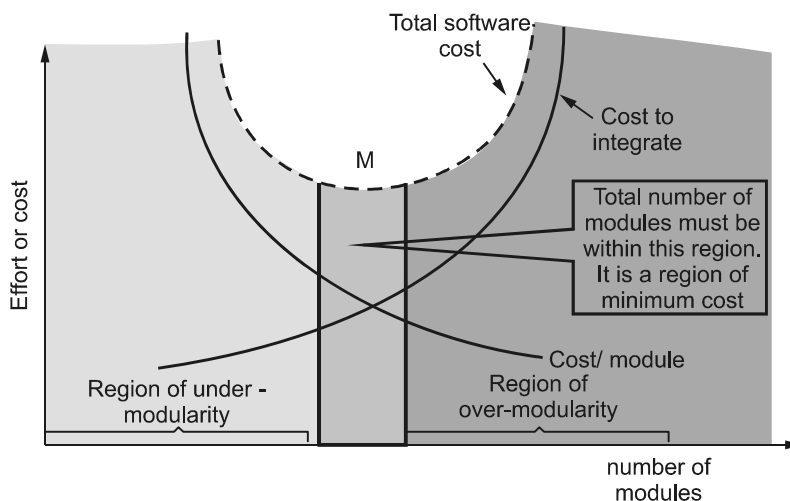


Fig. 3.3.1 Modularity and software cost

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

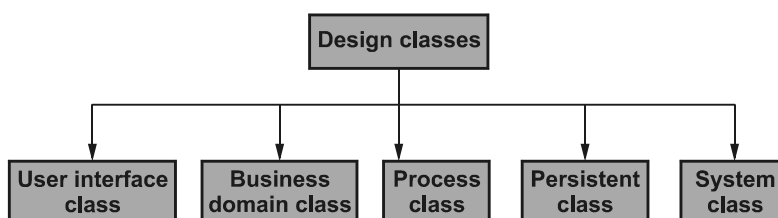


Fig. 3.3.2 Types of Design classes

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 possess 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, Marks 8
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 Model

- The **process dimension** denotes that the design model evolves due to various software tasks that get executed as the part of software process.

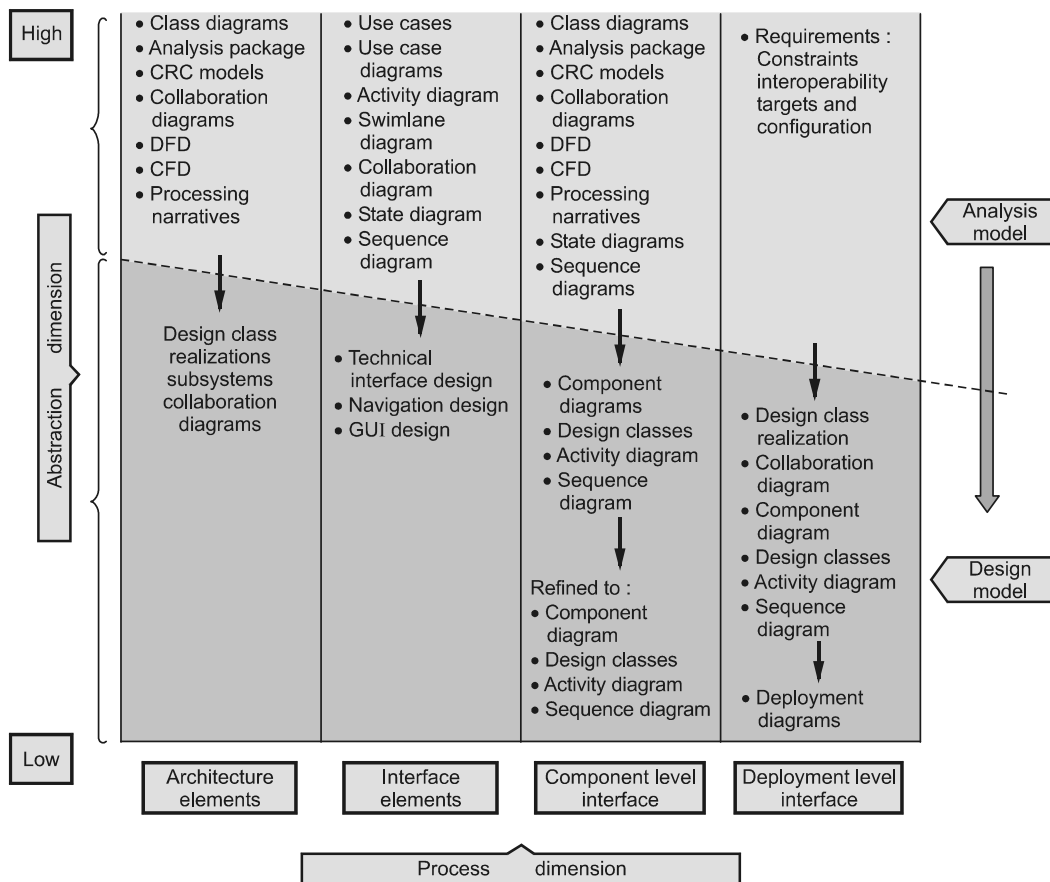


Fig. 3.4.1 Dimension of Design Model

- The **abstract dimension** represents level of details as each element of analysis model is transformed into design equivalent.
- In Fig. 3.4.1 the **dashed line** shows the boundary between analysis and design model.
- In both the analysis and design models the same UML diagrams are used but in analysis model the UML diagrams are abstract and in design model these diagrams are refined and elaborated. Moreover in design model the implementation specific details are provided.
- Along the horizontal axis various elements such as architecture element, interface element, component level elements and deployment level elements are given. It is not necessary that these elements have to be developed in sequential manner. First of all the preliminary architecture design occurs then interface design and component level design occur in parallel. The deployment level design ends up after the completions of complete design model.

3.4.1 Data Design Element

The data design represents the **high level of abstraction**. This data represented at data design level is **refined gradually** for implementing the computer based system. The data has great impact on the architecture of software systems. Hence structure of data is very important factor in software design. Data appears in the form of **data structures and algorithms** at the program **component level**. At the **application level** it appears as the **database** and at the **business level** it appears as **data warehouse and data mining**. Thus data plays an important role in software design.

3.4.2 Architectural Design Element

The architectural design gives the layout for overall view of the software. Architectural model can be built using following sources -

- Data flow models or class diagrams
- Information obtained from application domain
- Architectural patterns and styles.

3.4.3 Interface Design Elements

Interface Design represents the **detailed design** of the software system. In interface design how **information flows** from one component to other component of the system is depicted. Typically there are three types of interfaces –

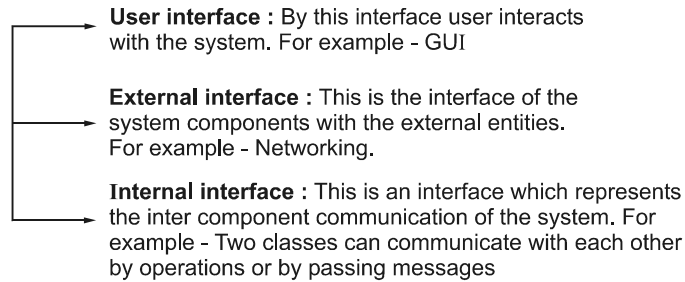


Fig. 3.4.2

3.4.4 Component Level Design Elements

The component level design is more detailed design of the software system along with the specifications. The component level design elements describe the internal details of the component. In component level design all the local data objects, required data structures and algorithmic details and procedural details are exposed.

Fig. 3.4.3 represents that component **order** makes use of another component **form**.

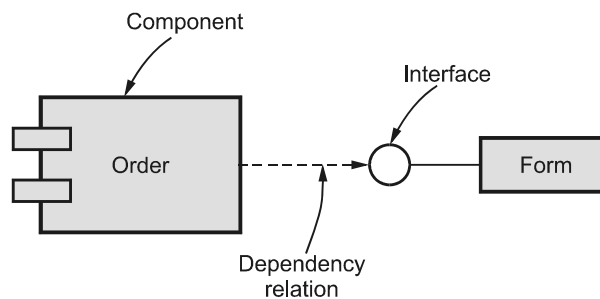


Fig. 3.4.3 Components

The **order** is dependent upon the component form. These two objects can be interfaced with each other.

3.4.5 Deployment Level Design Elements

The deployment level design elements indicate how software functions and software subsystems are assigned to the physical computing environment of the software product.

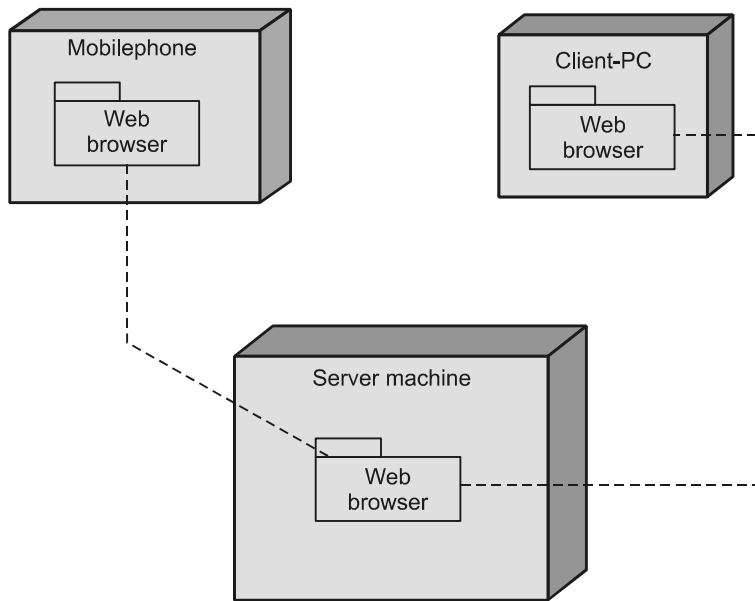


Fig. 3.4.4 Deployment diagram

For example web browsers may work in mobile phoned or they may run on client PC or can execute on server machines.

3.5 Design Heuristic

AU : May-13, 16 Marks 8

The program structure can be manipulated according the design heuristics as shown below.

1. **Evaluate the first iteration of the program structure to reduce the coupling and improve cohesion** - The module independency can be achieved in the program structure by exploding or imploding the modules. Exploding the modules means obtaining two or more modules in the final stage and imploding the modules means combining the result of different modules.
2. **Attempt to minimize the structures with high fan-out and strive for fan-in as depth increases** - At the higher level of program structure the distribution of control should be made. Fan-out means number of immediate subordinates to the module. Fan-in means number of immediate ancestors the module have.
3. **Keep scope of the effect of a module within the scope of control of that module** - The decisions made in particular module 'a' should not affect the module 'b' which lies outside the scope of module 'a'.
4. **Evaluate the module interfaces to reduce complexity and redundancy and improve consistency** - Mostly the cause of software errors is module interfaces.

The module interfaces should simply pass the information and should be consistent with the module.

5. **Define module whose function is predictable but avoid modules that are too restrictive** - The module should be designed with simplified internal processing so that expected data can be produced as a result. The modules should not restrict the size of local data structure, options with control flow or modes of external interfaces. Being module too much restrictive causes large maintenance.
6. **Strive for controlled entry modules by avoiding pathological connections** - Software interfaces should be constrained and controlled so that it will become manageable. Pathological connection means many references or branches into the middle of a module.

Review Questions

1. *Discuss the design heuristics for effective modularity design.*

AU : May-13, Marks 8, May-16, Marks 4

2. *Write short note -Design heuristics*

AU : May-16, Marks 4

3.6 Introduction to Architectural Design

AU : May-16, Marks 4

Definition : Architectural design is a design created to represent the data and program components that are required to build the computer based systems.

- Architectural design is a specialized activity in which using specific architectural style and by considering the system's structure and properties of system components - a large and complex system is built.
- The person who is responsible to design such system is called **software architect**.
- The architectural design gives a **layout** of the system that is to be built. In short, the **program structure** is created during architectural design along with the description of component properties and their inter-relationship.

3.7 Software Architecture

AU : May-17,19, Marks 13

The architectural design is the design process for identifying the subsystems making up the system and framework for subsystem control and communication.

The **goal** of architectural design is to establish the overall structure of software system. Architectural design represents the link between design specification and actual design process.

*Software Architecture is a structure of systems which consists of various **components**, externally visible **properties** of these components and the inter-**relationship** among these components.*

Importance of software architecture

There are three reasons why the software architecture is so important?

1. Software architecture gives the representation of the computer based system that is to be built. Using this system model even the **stakeholders** can take **active part** in the software development process. This helps in clear specification/understanding of requirements.
2. Some **early design decisions** can be taken using software architecture and hence system performance and operations remain under control.
3. The software architecture gives a clear cut **idea** about the computer based system which is to be built.

3.7.1 Structural Partitioning

The program structure can be partitioned horizontally or vertically.

Horizontal partitioning

Horizontal partitioning defines separate branches of the modular hierarchy for each major program function.

Horizontal partitioning can be done by partitioning system into : input, data transformation (processing) and output.

In horizontal partitioning the design making modules are at the top of the architecture.

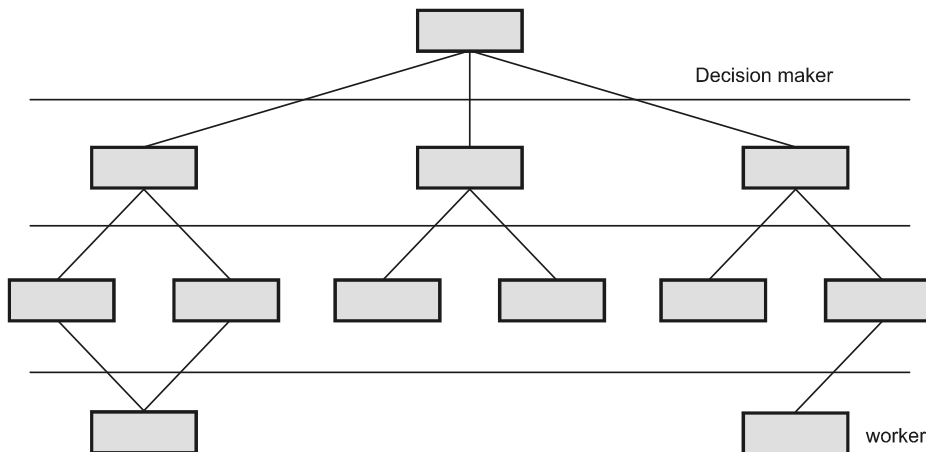


Fig. 3.7.1 Horizontal partitioning

Advantages of horizontal partition

1. These are easy to test, maintain and extend.
2. They have fewer side effects in change propagation or error propagation.

Disadvantage of horizontal partition

More data has to be passed across module interfaces which complicate the overall control of program flow.

Vertical partitioning

Vertical partitioning suggests the control and work should be distributed top-down in program structure.

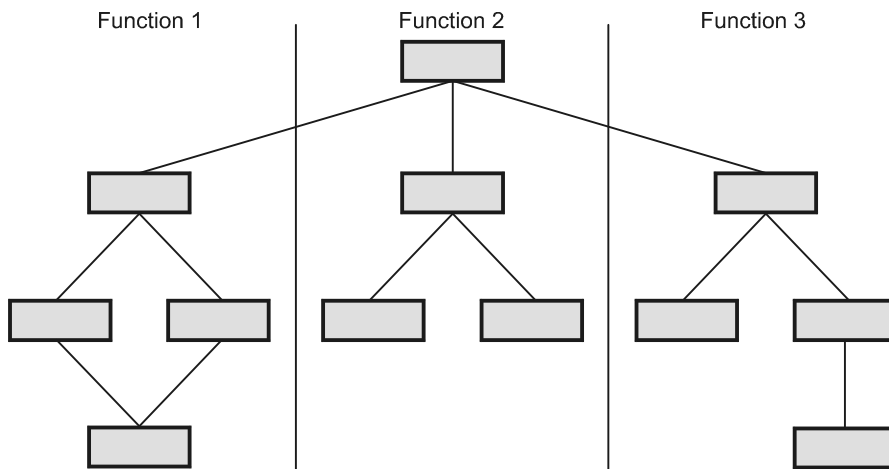


Fig. 3.7.2 Vertical partitioning

In vertical partitioning

- Define separate branches of the module hierarchy for each major function.
- Use control modules to co-ordinate communication between functions.

Advantages of vertical partition

1. These are easy to maintain the changes.
2. They reduce the change impact and error propagation.

Review Questions

1. What is software architecture ? Describe in detail different types of software architectures with illustrations.
AU : May-17, Marks 13
2. Define software architecture.
AU : May-19, Marks 2

3.8 Data Design**AU : May-16, Marks 8**

- Data design is basically the model of data that is represented at the high level of abstraction.
- The data design is then progressively refined to create implementation specific representations.
- Various elements of data design are
 - Data object – The data objects are identified and relationship among various data objects can be represented using entity relationship diagrams or data dictionaries.
 - Databases – Using software design model, the data models are translated into data structures and databases at the application level.
 - Data warehouses – At the business level useful information is identified from various databases and the data warehouses are created. For extracting or navigating the useful business information stored in the huge data warehouse then data mining techniques are applied.

Guideline for data design*1. Apply systematic analysis on data*

Represent data objects, relationships among them and data flow along with the contents.

2. Identify data structures and related operations

For the design of efficient data structures all the operations that will be performed on it should be considered.

3. Establish data dictionary

The data dictionary explicitly represents various data objects, relationships among them and the constraints on the elements of data structures.

4. Defer the low-level design decisions until late in the design process

Major structural attributes are designed first to establish an architecture of data. And then low-level design attributes are established.

5. Use information hiding in the design of data structures

The use of information hiding helps in improving quality of software design. It also helps in separating the logical and physical views.

6. Apply a library of useful data structures and operations

The data structures can be designed for reusability. A use of library of data structure templates (called as abstract data types) reduces the specification and design efforts for data.

7. Use a software design and programming language to support data specification and abstraction

The implementation of data structures can be done by effective software design and by choosing suitable programming language.

Difference between Function Oriented Design and Object Oriented Design

Sr. No.	Function oriented design	Object oriented design
1.	In function oriented design method, the software design is by designing functions.	In object oriented design method the software design is by designing objects.
2.	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.
3.	The function oriented approach is simple to design.	Using this design approach bugs can be easily located and corrected without affecting the other module.
4.	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 Passengerattributes : name, age, sex, source, destinationOperations : get_personal_data(), resever(), cancel(),Class Ticketattributes : Ticket_ID, Source, destination, date, fair_amountoperation : compute_fair()

Review Questions

1. Write short note -User-Data Design

AU : May-16, Marks 4

2. Discuss the differences between object oriented and function oriented design.

AU : May-16, Marks 8

3.9 Architectural Style

AU : May-07,18,19, Dec.-19, Marks 16

3.9.1 Architectural Styles

- The **architectural model or style** is a pattern for creating the system architecture for given problem. However, most of the large systems are heterogeneous and do not follow single architectural style.
- System categories define the architectural style

1. Components : They perform a function.

For example : Database, simple computational modules, clients, servers and filters.

2. Connectors : Enable communications. They define how the components communicate, co-ordinate and co-operate.

For example : Call, event broadcasting, pipes

3. Constraints : Define how the system can be integrated.

4. Semantic models : Specify how to determine a system's overall properties from the properties of its parts.

- The commonly used architectural styles are
 1. Data centered architectures. 2. Data flow architectures.
 3. Call and return architectures. 4. Object oriented architectures.
 5. Layered architectures.

3.9.1.1 Data Centered Architectures

In this architecture the data store lies at the centre of the architecture and other components frequently access it by performing add, delete and modify operations. The client software requests for the data to central repository. Sometime the client software accesses the data from the central repository without any change in data or without any change in actions of software actions.

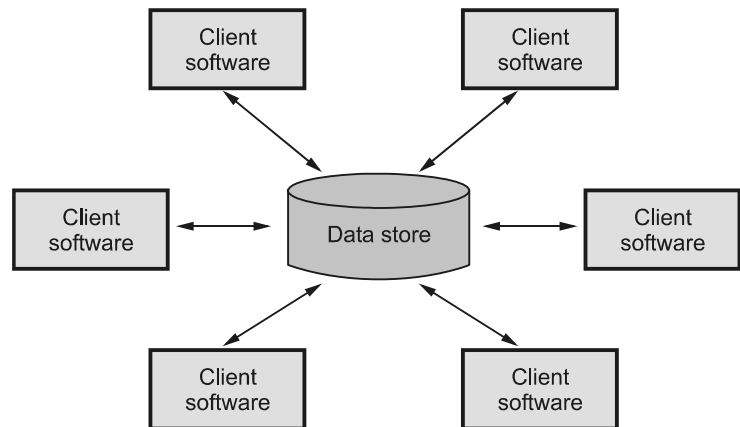


Fig. 3.9.1 Data centered architecture

Data centered architecture possesses the property of interchangeability. Interchangeability means any component from the architecture can be replaced by a new component without affecting the working of other components.

In data centered architecture the data can be passed among the components.

In data centered architecture,

Components are : Database elements such as tables, queries.

Communication are : By relationships

Constraints are : Client software has to request central data store for information.

3.9.1.2 Data Flow Architectures

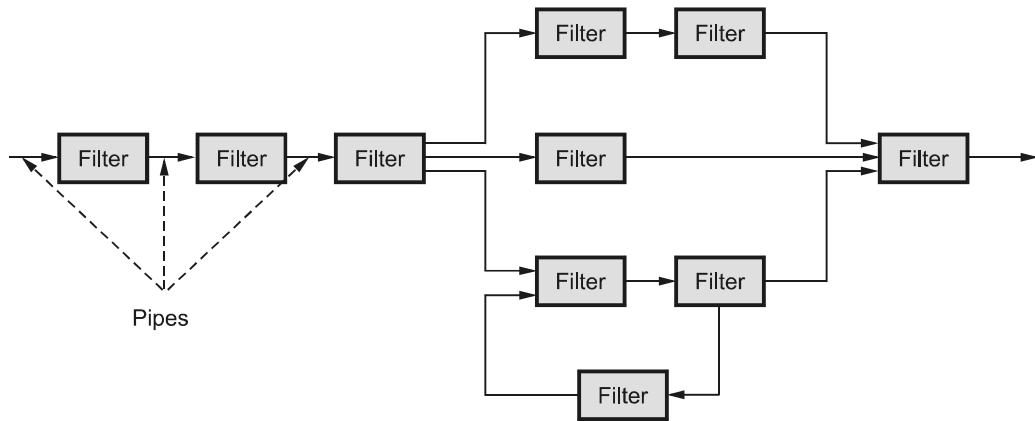


Fig. 3.9.2 Pipes and filters

In this architecture series of transformations are applied to produce the output data. The set of components called filters are connected by pipes to transform the data from one component to another. These filters work independently without a bothering about the working of neighbouring filter.

If the data flow degenerates into a single line of transforms, it is termed as batch sequential.

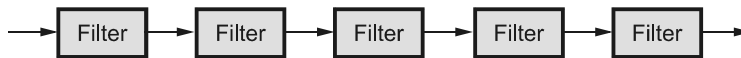


Fig. 3.9.3 Batch sequential

In this pattern the transformation is applied on the batch of data.

3.9.1.3 Call and Return Architecture

The program structure can be easily modified or scaled. The program structure is organized into modules within the program. In this architecture how modules call each other. The program structure decomposes the function into control hierarchy where a main program invokes number of program components.

In this architecture the hierarchical control for call and return is represented.

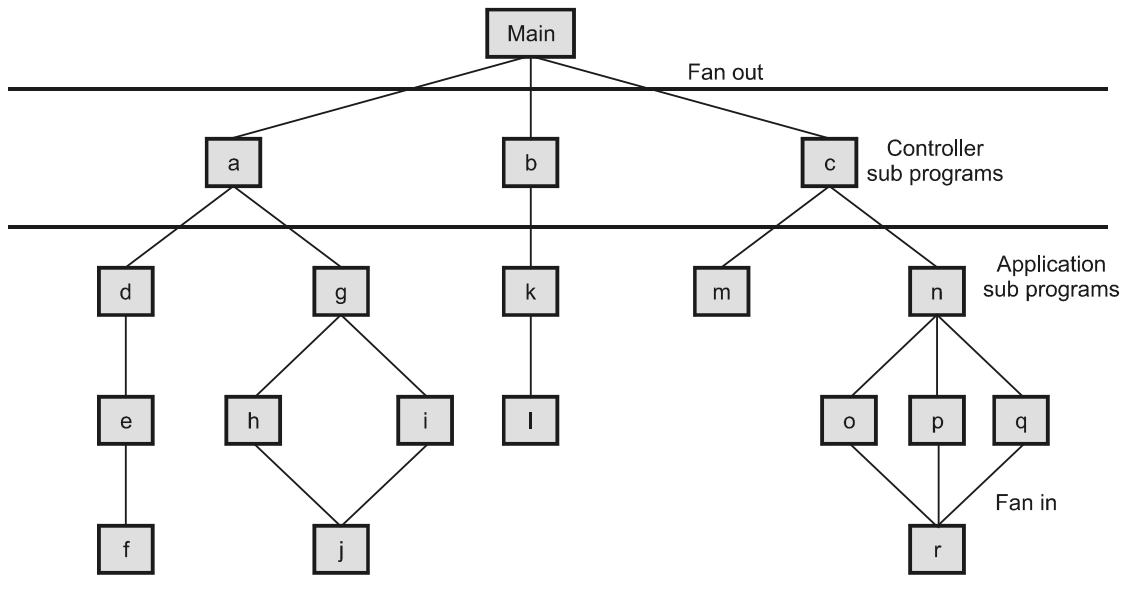


Fig. 3.9.4 Call and return Architecture

3.9.1.4 Object Oriented Architecture

In this architecture the system is decomposed into number of interacting objects.

These objects encapsulate data and the corresponding operations that must be applied to manipulate the data.

The object oriented decomposition is concerned with identifying objects classes, their attributes and the corresponding operations. There is some control models used to co-ordinate the object operations.

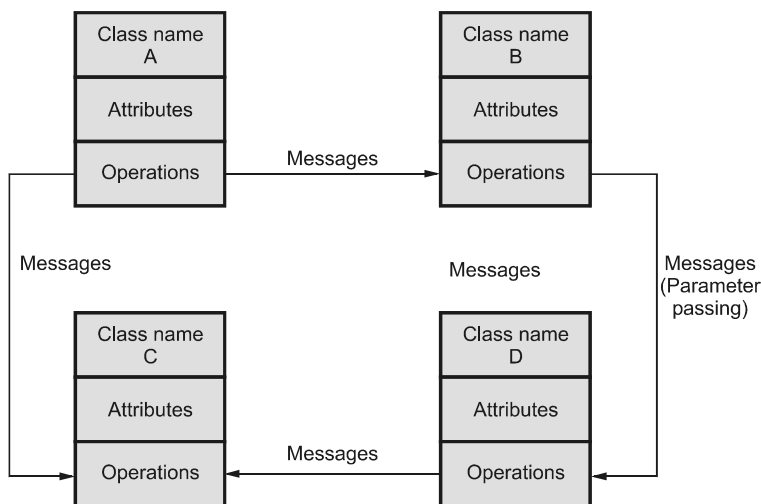


Fig. 3.9.5 Object oriented architecture

3.9.1.5 Layered Architecture

- 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.

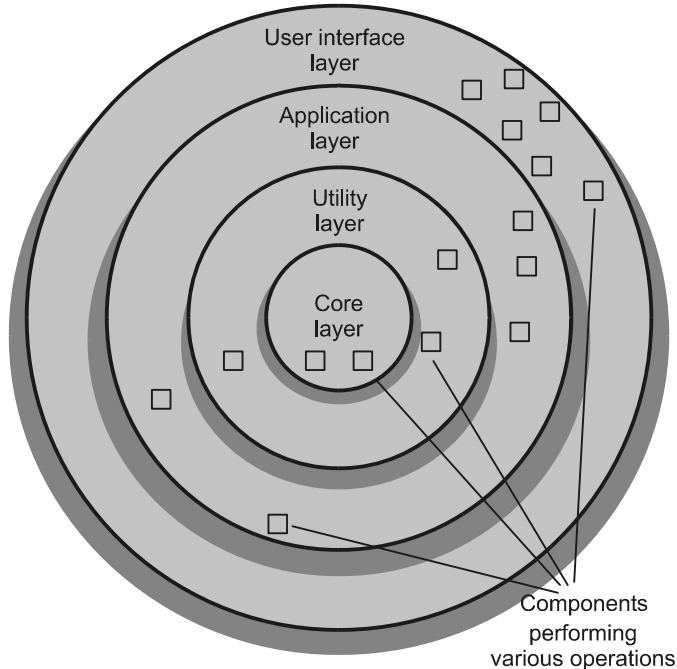


Fig. 3.9.6 Layer architecture of components

- 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.

3.9.2 Architectural Patterns

In this section we will understand *what are architectural patterns?* The architectural pattern is basically an approach for handling behavioural characteristics of software systems. Following are the architectural pattern domains

1. Concurrency

Concurrency means handling multiple tasks in parallel. For example in operating system, **multiple tasks** are executed in parallel. Hence concurrency is a pattern which represents that the system components can interact with each other in parallel. The benefit of this pattern is that system **efficiency** can be achieved.

2. Persistence

Continuity in the data can be maintained by the persistence pattern. In other words the data used in earlier execution can be made available further by storing it in files or in **databases**. These files/databases can be modified in the software system as per the need. In **object oriented** system the values of all attributes various operations that are to be executed are persistent for further use. Thus broadly there are two patterns. i) Database management pattern ii) Application level pattern.

3. Distribution

Distribution pattern refers to the way in which the system components communicate with each other in distributed systems. There are two major problems that occur in distribution pattern

- The nature of interconnection of the components
- The nature of communication

These problems can be solved by other pattern called **broker pattern**. The broker pattern lies between server and client components so that the client server communication can be established properly. When client want some service from server, it first sends message to broker. The broker then conveys this message to server and completes the connection. Typical example is CORBA. The CORBA is a distributed architecture in which broker pattern is used.

Review Questions

1. What are different types of architectural styles exist for software and explain any one software architecture in detail. **AU : May-07, Marks 16**
2. What is software architecture ? Describe the different software architectural styles with examples. **AU : May-18, Marks 16**
3. Explain and compare the following architectural styles : 1) Call and return architecture 2) Object-oriented architecture 3) Layered architecture **AU : May-19, Marks 11**
4. What is software architecture ? Outline the architectural styles with an example. **AU : Dec.-19, Marks 13**

3.10 Architectural Design

AU : Dec.-03, Marks 16

In architectural design at the initial stage a **context model** is prepared. This model defines the external entities that interact with the software. Along with this model the nature of software interaction with external entities is also described. The context model is prepared by using information obtained from analysis model and requirement specification. After that the designer creates **structure of the system** by defining and refining software components. Thus process of creations of context model and structural model of the system is iteratively carried out until and unless complete architectural

model of the system gets created. Let us discuss how an architectural design gets generated using some simple representations.

3.10.1 Representing System in Context

The context model is a graphical model in which the environment of the system is defined by showing the external entities that interact with the software system.

In architectural design the Architectural Context Diagram(ACD) is created. The difference between context model and architectural context diagram is that in ACD the nature of interaction is clearly described. Following are the basic terminologies associated with architectural context diagram.

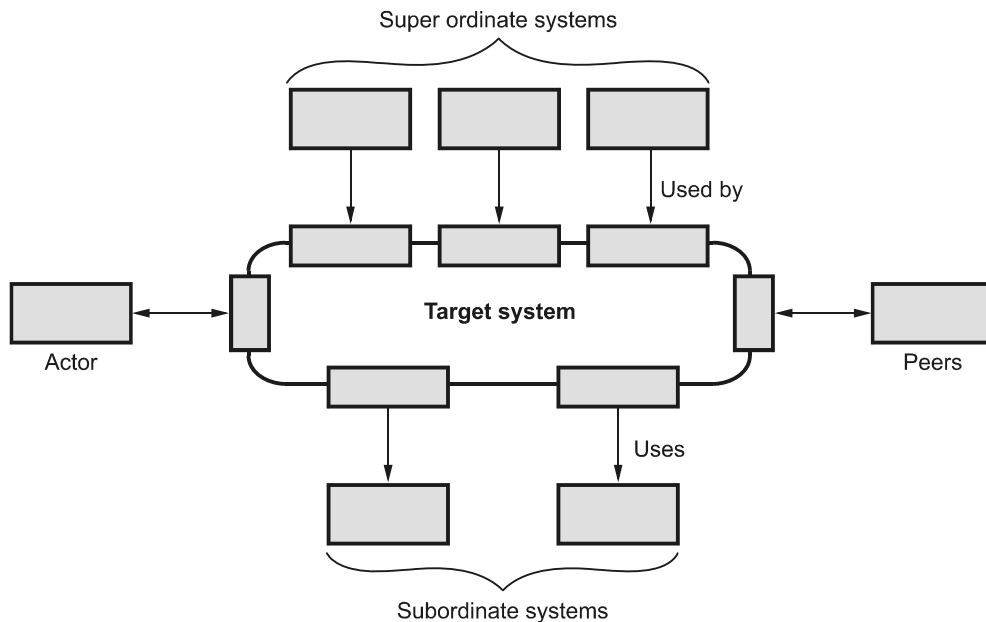


Fig. 3.10.1 Architectural context diagram

Target system : The target system is a computer based system for which the architectural context diagram has to be prepared.

Super ordinate systems : These systems are created at higher level of processing when the target system is being developed.

Sub ordinate systems : These systems are used by the target system for processing of the data that are necessary to complete target system functionality.

Actors : These are the systems or entities that interact with the target system for producing or consuming information of the target system.

Peer-level systems : These are the systems that interact on peer to peer basis.

For example -

Consider the *Inventory Control System* for which the Architectural Context Diagram can be prepared as below

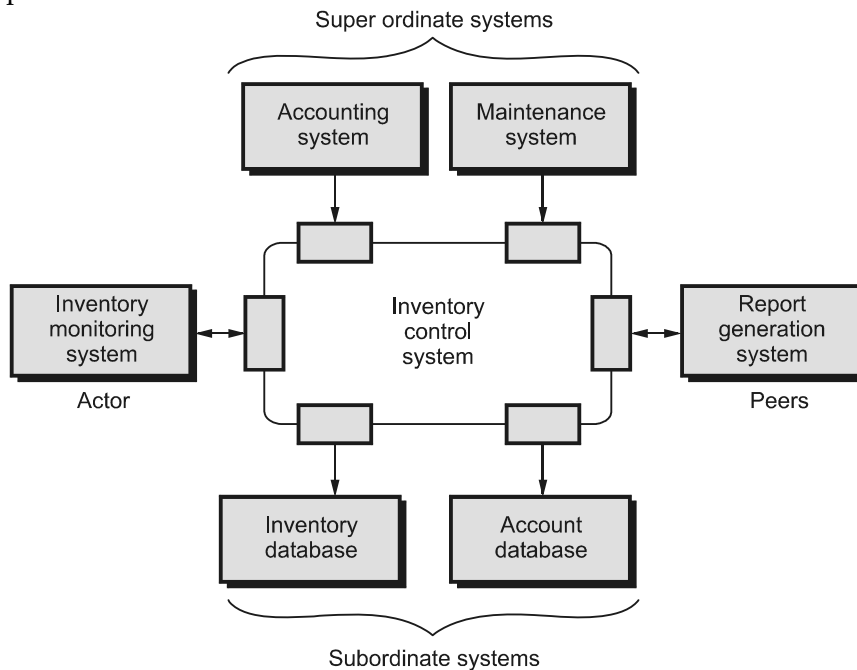


Fig. 3.10.2 ACD for inventory control system

3.10.2 Defining Archetypes

Defining archetype is a **basic step** in architectural design, more precisely in functionality based design. Archetype is a **core abstraction** using which the system can be structured. Using archetypes, a **small set** of entities that describe the major part of system behaviour can be described. Typically archetypes are the stable elements and they do not change even though system undergo through various changes. Identifying archetypes is a critical task and it requires well experienced architect. Following figure represents various archetypes.

Various types of archetypes are

Point or node : It refers to the highest level of abstraction in which there is a cohesive collection of input and output functionalities.

Detector : Detectors capture the core functionalities of the system. For example, in temperature control system sensors for temperature is a sensor.

Control unit or controller : Controllers are the entities that are useful for controlling behaviour of the system. For example, in temperature control system. When the

temperature exceeds beyond some threshold value alarming and dis-alarming system is required. Such a system acts as a controller or control unit.

Indicator or output : It represents the generic output functionalities. For example, monitoring system of any computer based system acts as an indicator.

In software engineering archetype is a number of **major components** that are used to describe the system which we want to build.

3.10.3 Refining Architecture into Components

To create full structure of the system it is required to refine the software architecture into components. Hence it is necessary to identify the components of the system. The components can be identified from **application domain** or from **infrastructure domain**. The architectural designer has to identify these components from these domains. There are two methods by which the components can be identified.

1. The **data flow diagram** is drawn from which the specialized components can be identified. Such components are the components that process the data flow across the interfaces.
2. The components can be the entities that follow following **functionalities** -

External communication : The components that take part in the communication with external entities are the communication components.

Control panel processing : These components perform all control panel management activities.

Detection : These are the components that perform detection activities.

Indicator management : These are the components that perform the output controlling activities.

3.10.4 Defining Instantiations of the System

In order to model a structure of the system simply refining the software into components is not sufficient. Further refinement is necessary by instantiation of the system. Instantiation of the system begins with identification of major components and then identification of its functionalities, characteristics and constraints is carried out in order to refine the system to greater extent.

Finally with sufficient detailing, the architectural model of the system gets ready.

3.11 Architectural Mapping using Data Flow

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

Transform flow

A transform flow is a sequence of paths which forms transition in which input data are transformed into output data.

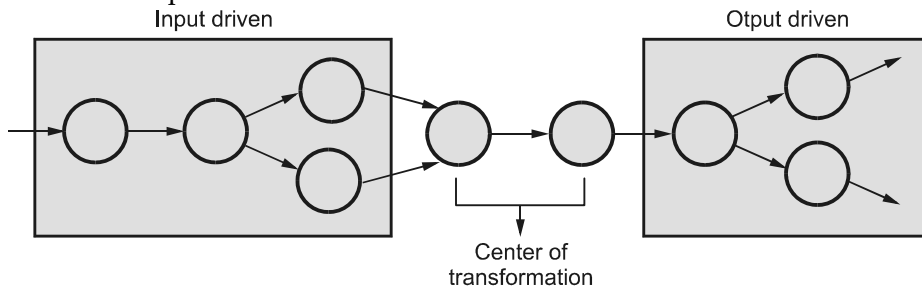


Fig. 3.11.1 Transform flow

Transaction flow

A transaction flow represents the information flow in which single data item triggers the overall information flow along the multiple paths. This triggering data item is called transaction.

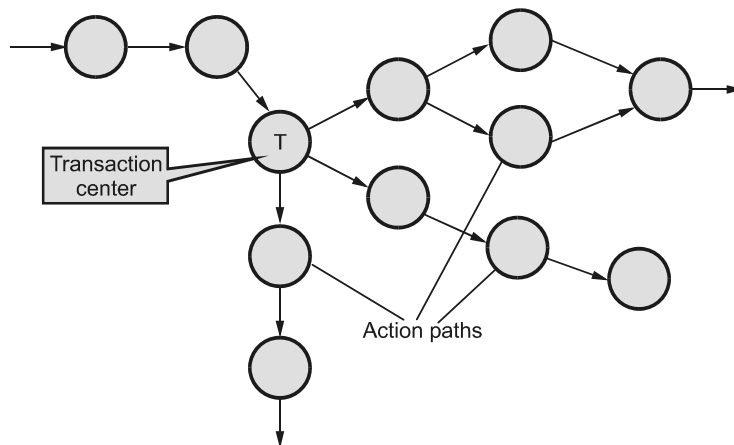


Fig. 3.11.2 Transaction flow

Example : Automated Railway Reservation System

Problem description : The automated railway reservation system is prepared for reserving the railway ticket online. Various activities that the user can perform are -

1. Registers for using the system
2. Make the train enquiry
3. Can view the status of reservation by entering PNR number

4. Can make the reservations
5. Can cancel the reservations.

After registering himself/herself to the system, user must have to log in. He/she can make the enquiry about the trains either by entering the train number or by entering the source and destinations.

User can see the availability of seats by entering the passenger details and journey details. If the seats are available then he can make the reservations by making the payment. The user then prints the E-Ticket.

User can view the latest status of his reservations by simply entering the PNR number.

If the user wants to cancel some reservations then he/she has to enter the PNR number. When user selects for the cancel reservations option then refund amount is paid to the customer and appropriate records are updated.

The Level 0, Level 1 and Level 2 DFD, Level 3 DFD are as shown below –

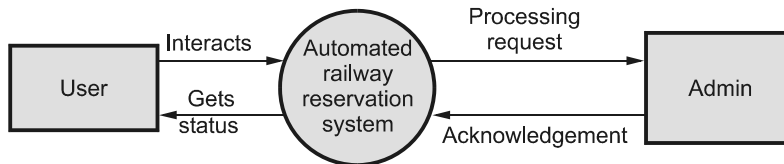


Fig. 3.11.3 Context Level DFD (Level 0)

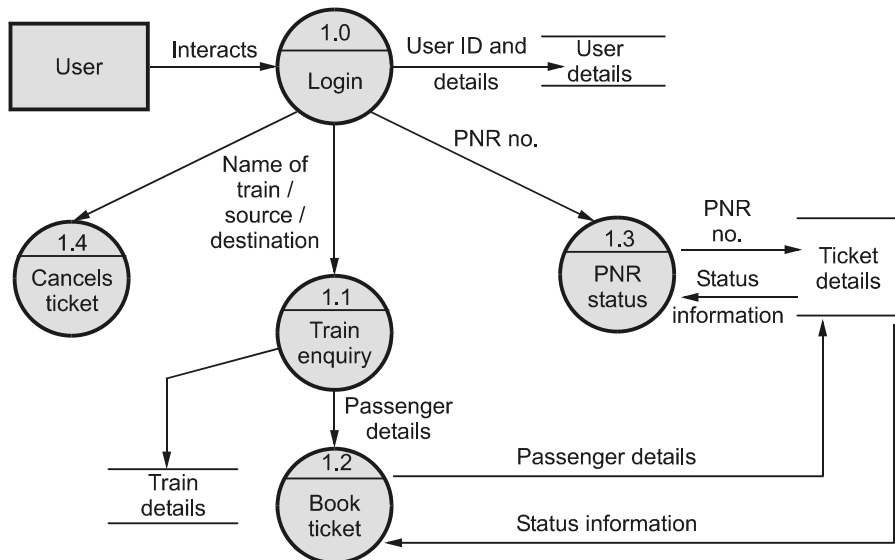


Fig. 3.11.4 Level 1 DFD

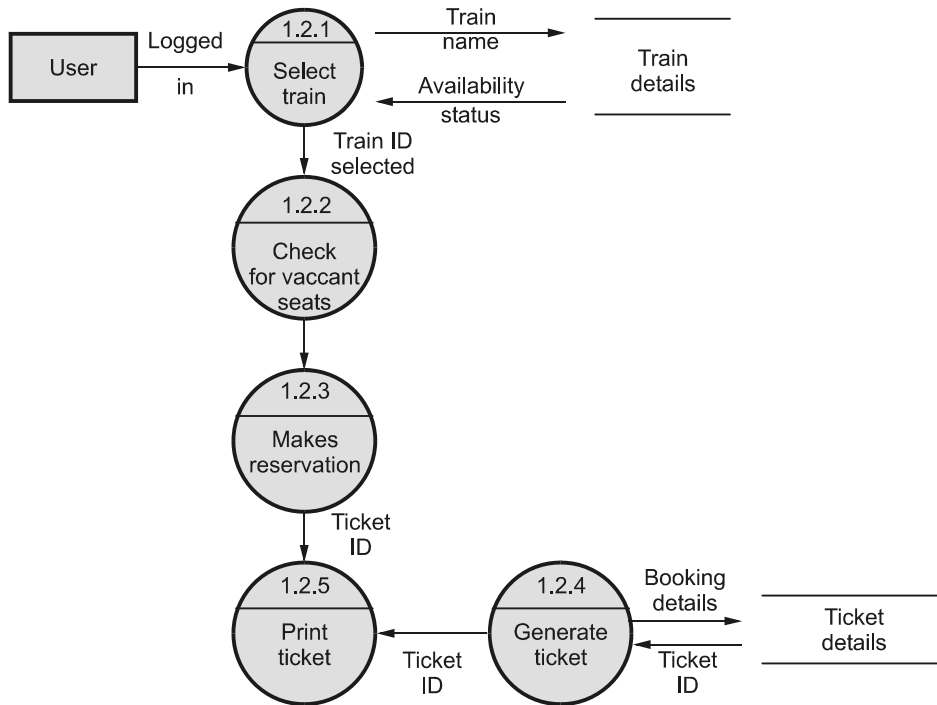


Fig. 3.11.5 Level 2 DFD for book ticket functionality

Refer Fig. 3.11.6 on next page.

3.11.1 Transform Mapping

Transform mapping is a collection of design steps using which a **DFD containing transform flow characteristics** is mapped into specific architectural style.

Design steps for transform mapping

Step 1 : Review fundamental system model to identify information flow

The fundamental system model can be represented by level 0 DFD and supporting information. This supporting information can be obtained from the two important documents called 'system specification' and 'software requirement specifications'. Both of them describe the information flow and structure at software interface.

Refer Fig. 3.11.3 for Level 0 DFD

Step 2 : Review and refine the data flow diagram for software

The level 0 DFD is refined and the higher level DFD is drawn. Refer Level 1 DFD and Level 2 in which the **Book Ticket** functionality is reviewed and refined. Refer Fig. 3.11.4 and 3.11.5.

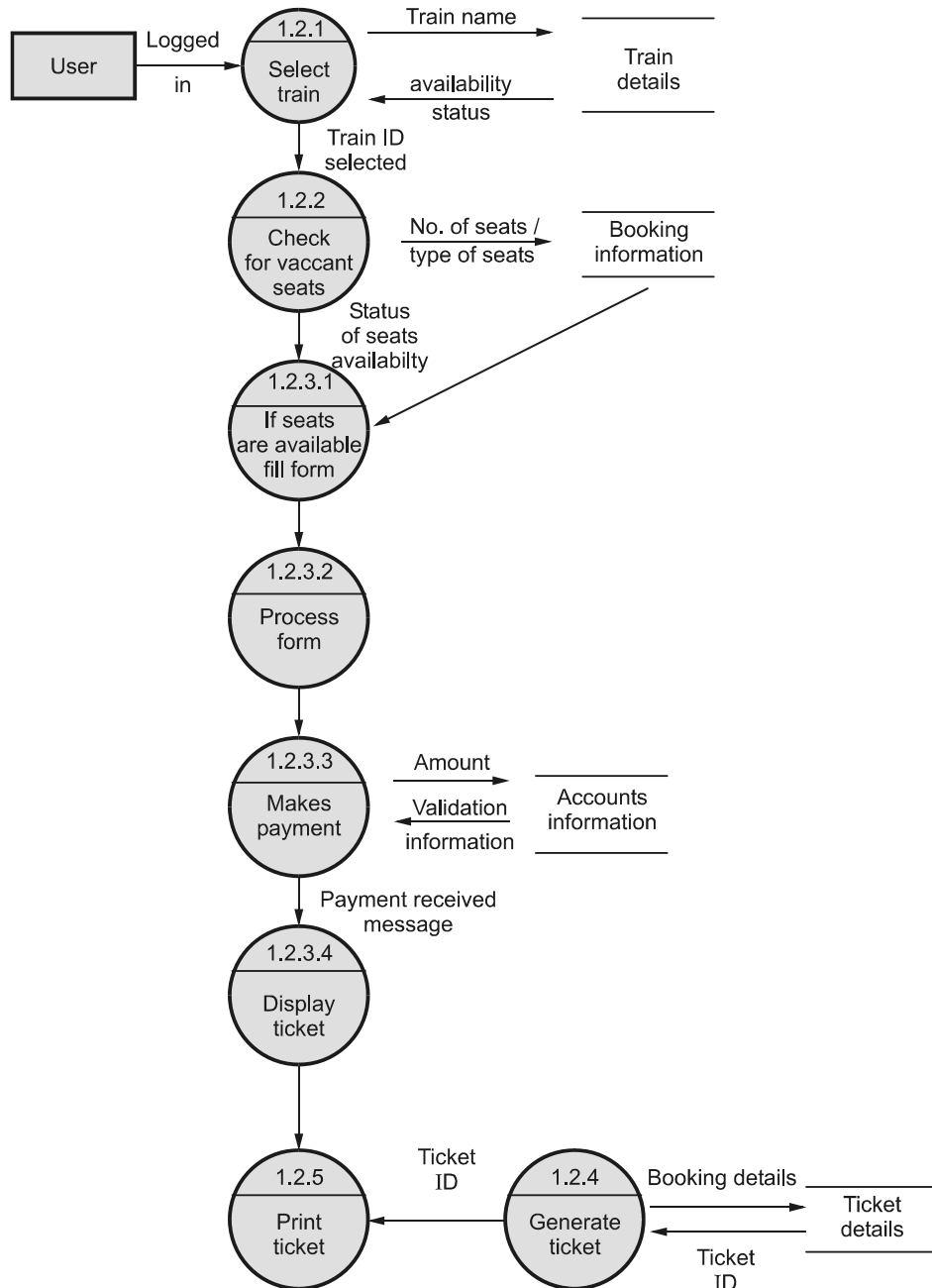


Fig. 3.11.6 Level 3 DFD for "Make Reservation"

Step 3 : Determine if the DFD has the transform or transaction flow characteristics

The information flow within the system is usually represented as transform flow. However, there can be dominance of transaction characteristics in the DFD. Based on characteristics of the DFD the transformation flow or transaction flow is decided.

For example - If we draw Level 3 DFD for the process **Make Reservation** then the transformation flow can be identified. Refer Fig. 3.11.6 for level 3 DFD.

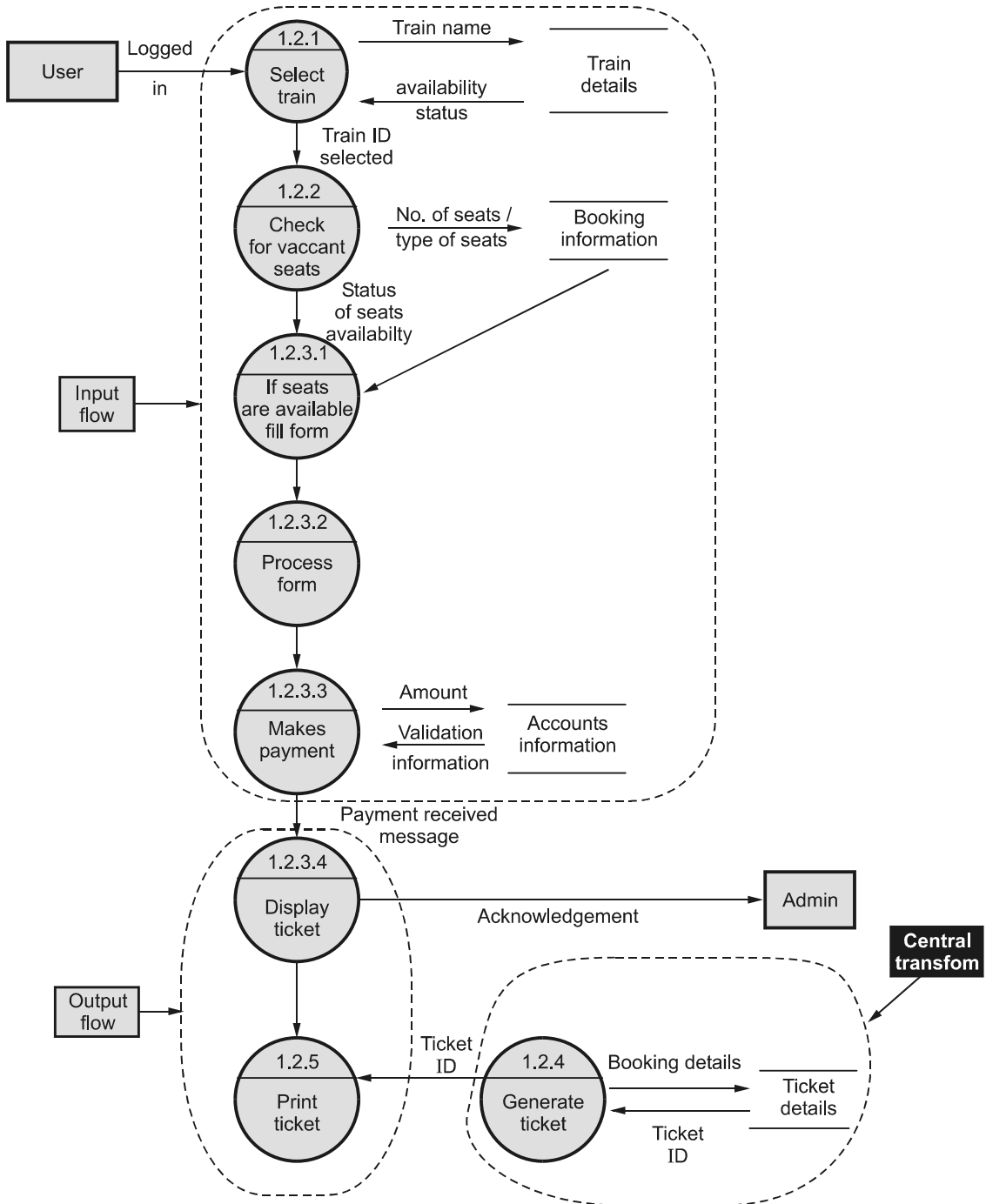


Fig. 3.11.7 Level 3 DFD for "Make Reservation"

Step 4 : Isolate the transform center by specifying incoming and outgoing flow boundaries

After identifying that the DFD shows the transformation flow, we can easily separate out the incoming and outgoing flow boundaries and the **transform center** can be identified.

Refer Fig. 3.13.7.

Step 5 : Perform first level factoring

After performing the first level factoring the program structure results in top down architecture where

- Top level components perform decision making

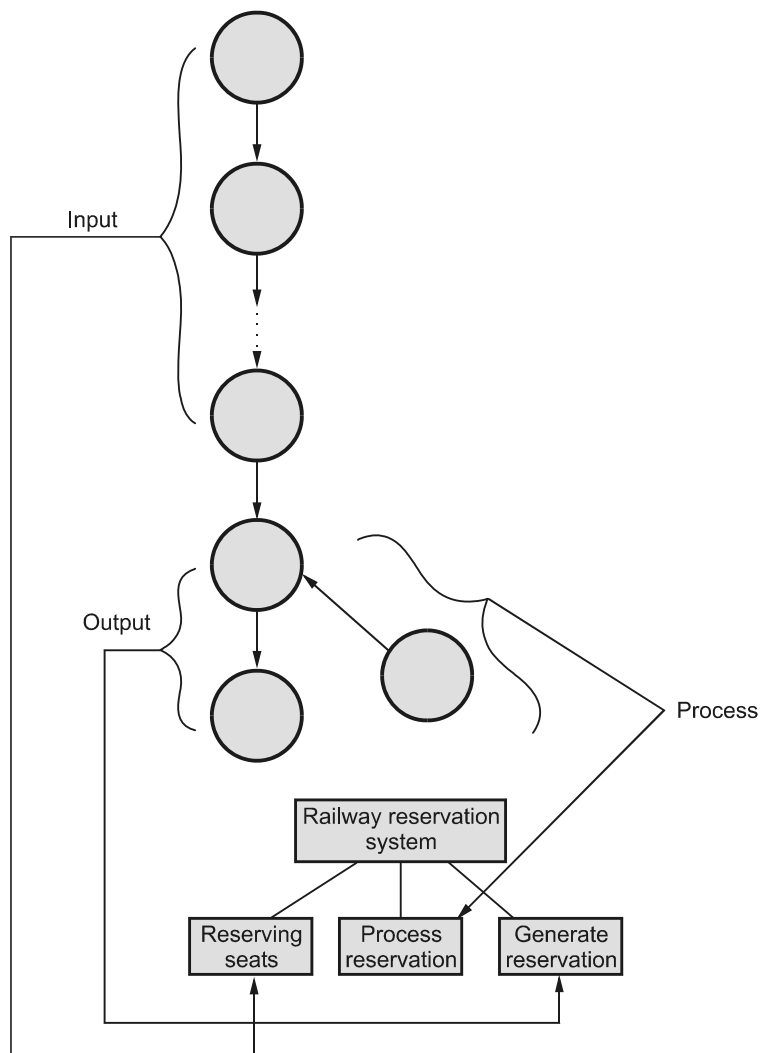


Fig. 3.11.8 First level factoring

- Low-level components perform most input, computation and output
- Middle-level perform some control and some amount of work.

When transform flow is identified the DFD is mapped into call and return architecture.

Refer Fig. 3.11.8

Step 6 : Perform second level factoring

In the second level factoring individual bubble of DFD is mapped into appropriate module within architecture.

There could be one-to-one mapping of bubble of DFD into the software module or two or three bubbles can be combined together to form a single software module.

After performing the second level factoring the architecture serves as the first-iteration design. Refer Fig. 3.11.9.

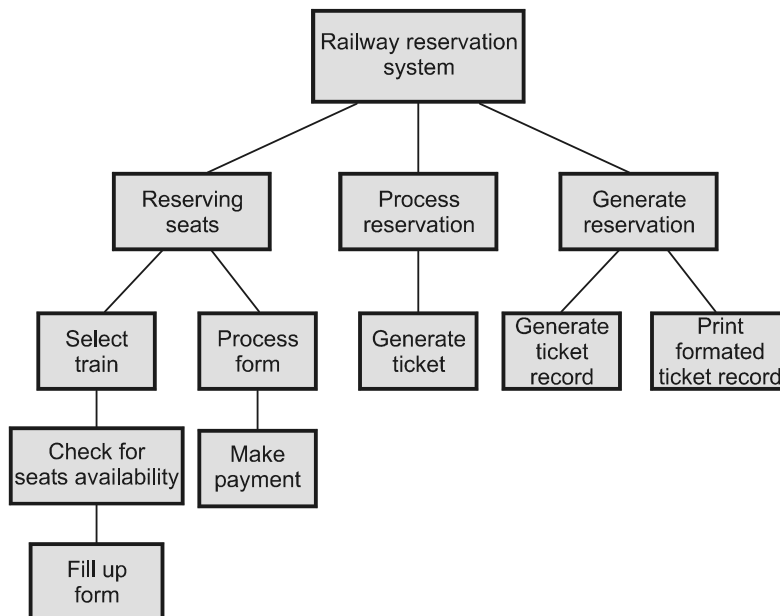


Fig. 3.11.9 Second level factoring

Step 7 : Refine the first iteration architecture using design heuristics for improved software quality

The first-iteration architecture can be refined by applying the module independency.

The modules can be exploded or imploded with high cohesion and minimum coupling. The refinement should be such that the structure can be implemented without difficulty, tested without confusion and can be easily maintained.

Refer refined program structure as shown in Fig. 3.11.10.

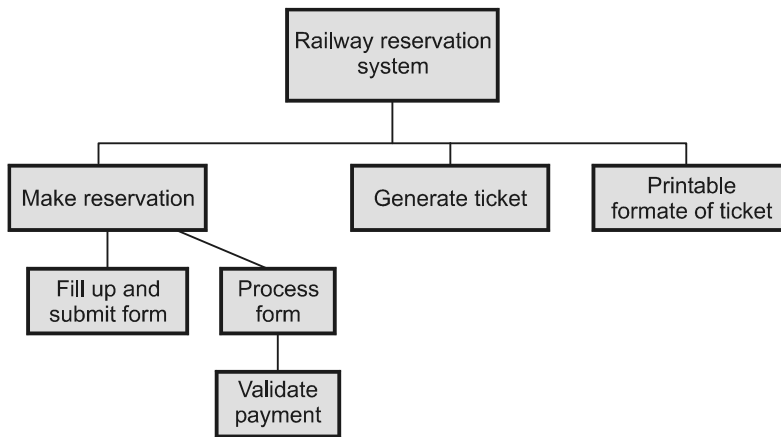


Fig. 3.11.10 Refined program structure

3.11.2 Transaction Mapping

In many software applications a single **data item** leads to one or more information flows. Each information flow specifies some distinct functionality. This data item is called **transaction center**.

Design Steps for Transaction Mapping

- Step 1 :** Review fundamental system model to identify information flow.
- Step 2 :** Review and refine the data flow diagram for software
- Step 3 :** Determine if the DFD has the transform or transaction flow characteristics
- Step 4 :** Identify the transaction center and flow characteristics along each of the action paths.

In transaction mapping we have to identify the location of transaction centre. From the transaction centre many action paths flow radially from it.

For example - Consider the Railway Reservation system which we have discussed in section 3.11.1, Refer the Level 1 DFD, in which after logging in to the system, **User** either selects for **Book Ticket**, **Ticket Enquiry**, **Cancel Ticket** or for **PNR status** option. Depending upon his selection the information flow will vary. Thus overall data flow is characterized by the transaction which is selected by the user. Hence it is called transaction mapping technique. There there can be multiple **action paths** fanning out from the transaction center. We can represent this concept by following Fig. 3.11.11.

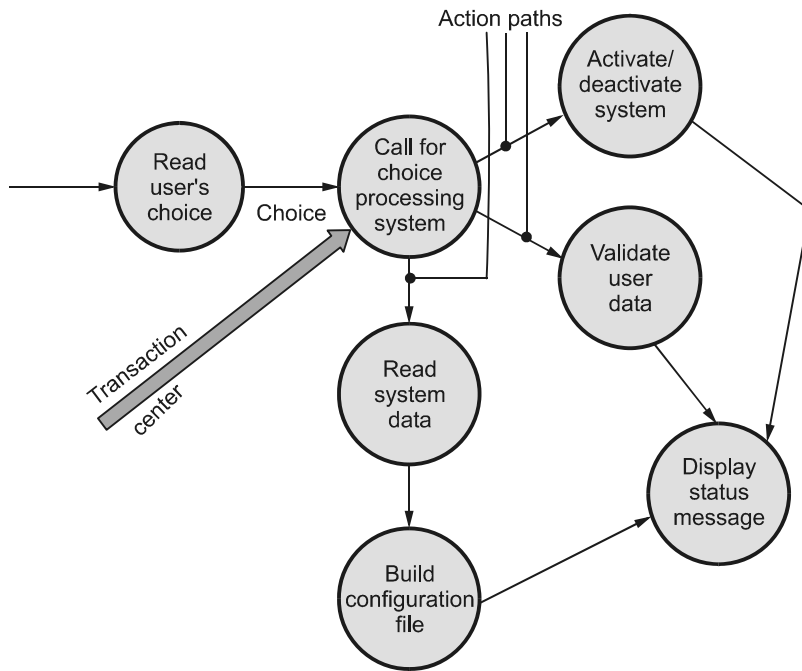


Fig. 3.11.11 Transaction flow for railway reservation system

Step 5 : Map DFD into transaction processing structure

The identified transaction flow is mapped into an architecture that contains an incoming branch and a dispatcher branch. Starting from the transaction centre the corresponding bubbles on incoming path (path coming to transaction centre) are mapped into the appropriate modules. The bubbles along the action path are mapped into the action modules.

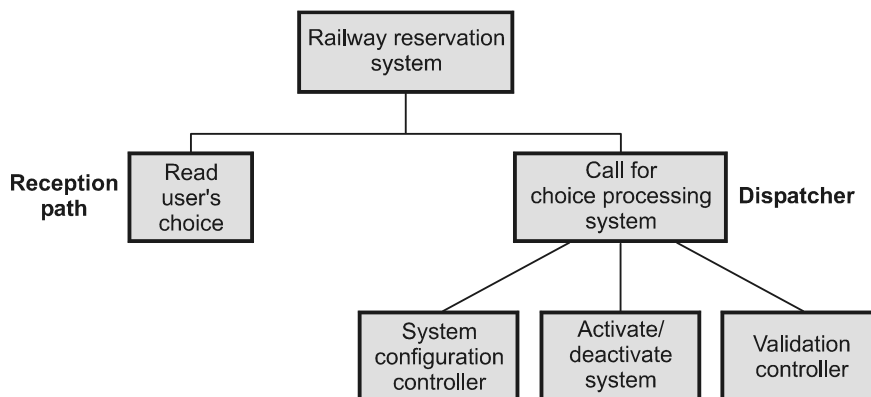


Fig. 3.11.12 First level factoring in transaction flow

Step 6 : Factor and refine the transaction structure and structure of each action path

Each action path of the data flow diagram has its own information flow characteristics. The action path related substructure is developed. This substructure serves as first iteration architecture.

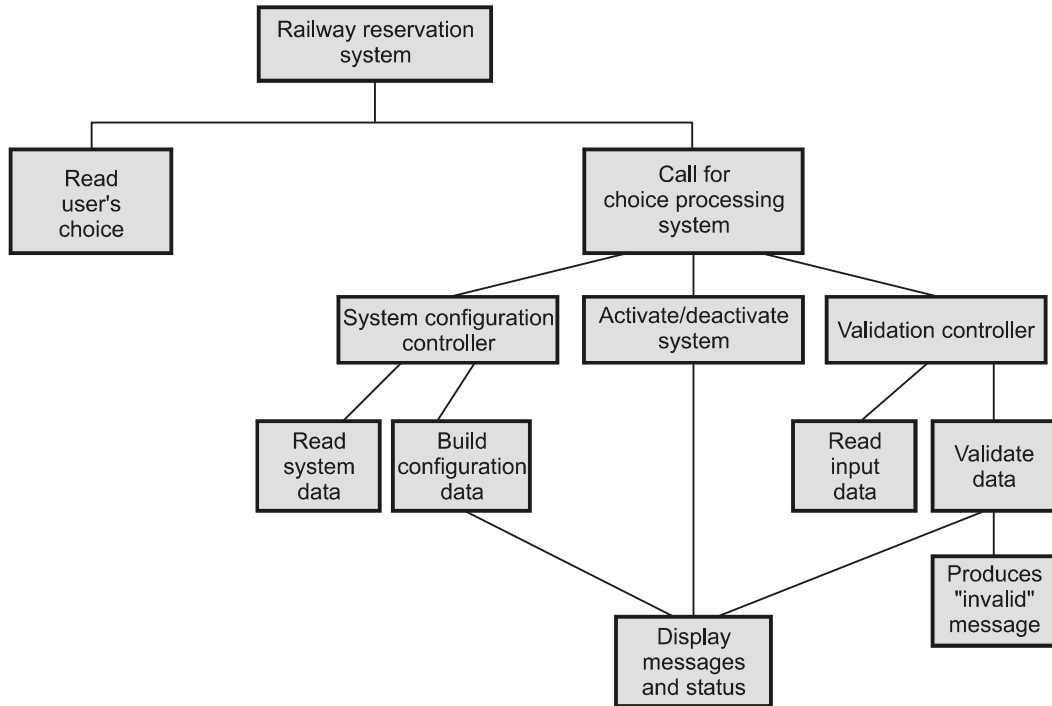


Fig. 3.11.13 First iteration architecture

Step 7 : Refine the first iteration architecture using design heuristic for improved software quality

The first-iteration architecture can be refined by applying the module independency.

The modules can be exploded or imploded with high cohesion and minimum coupling. The refinement should be such that the structure can be implemented without difficulty, tested without confusion and can be easily maintained.

Example 3.11.1 Consider the problem of determining the number of different words in an input file. Carry out structured design by performing transform and transaction analysis, construct structured chart.

AU : May-18, Marks 15

Solution : Step 1 : We will design the DFD for problem of determining the number of different words in an input file -

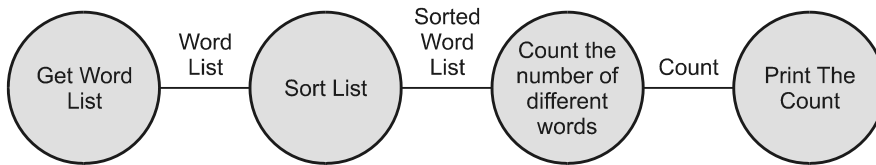


Fig.3.11.14 DFD For word counting problem

Step 2 : Now we will identify in input flow, output flow and central transform from the DFD. It is as shown below

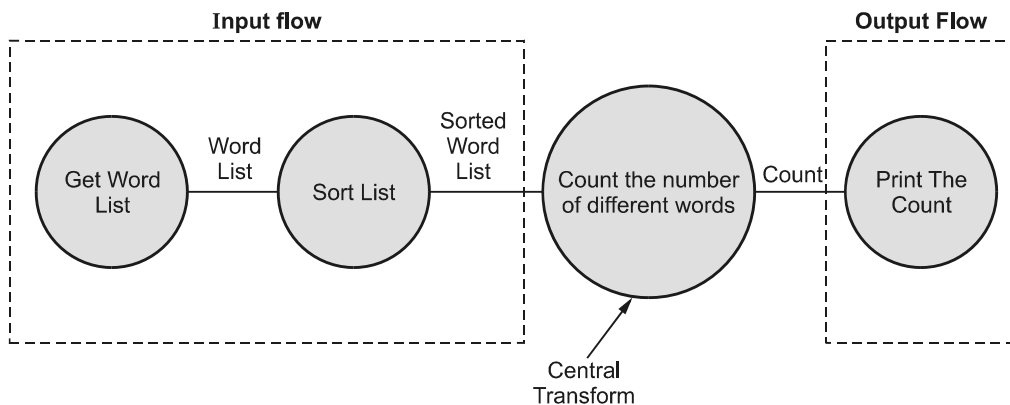


Fig.3.11.15

Step 3 : After identifying input flow, central transform and output flow - we can now perform first level factoring. This first level factoring can be represented using structure chart as below.

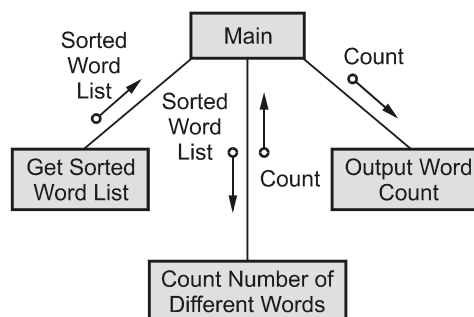


Fig. 3.11.16

Step 4 : Now factor the input module, and central transform. It is as shown below –

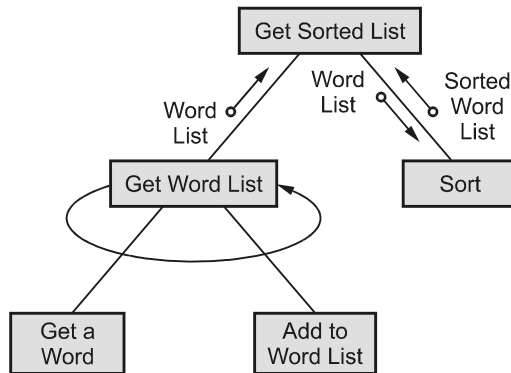


Fig.3.11.17 Factoring input flow

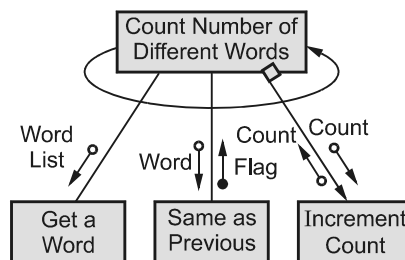


Fig. 3.11.18 Factoring central transform

Review Question

1. What is structured design ? Illustrate the structured design process from DFD to structured chart with a case study.

AU : Dec.-16, Marks 16

3.12 Concept of User Interface Design

AU : May-07, 16, 13, Dec.-08, 09, 17, 16, 15, 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.

3.12.1 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 -

3.12.1.1 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 losing 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.

3.12.1.2 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.

3.12.1.3 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.

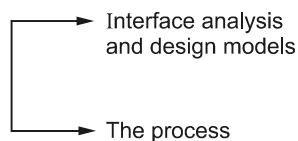
3.12.2 User Interface Analysis and Design

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 –



3.12.2.1 Interface Analysis and Design Model

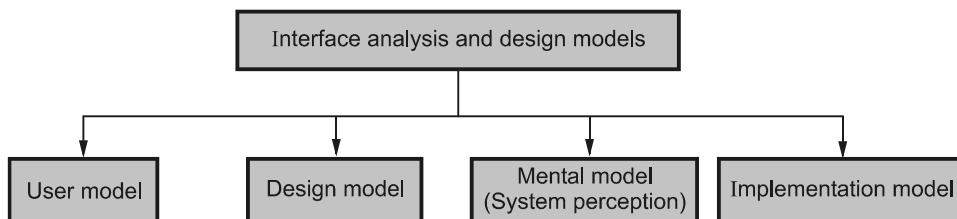


Fig. 3.12.1 Analysis and design model

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!

3.12.2.2 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.

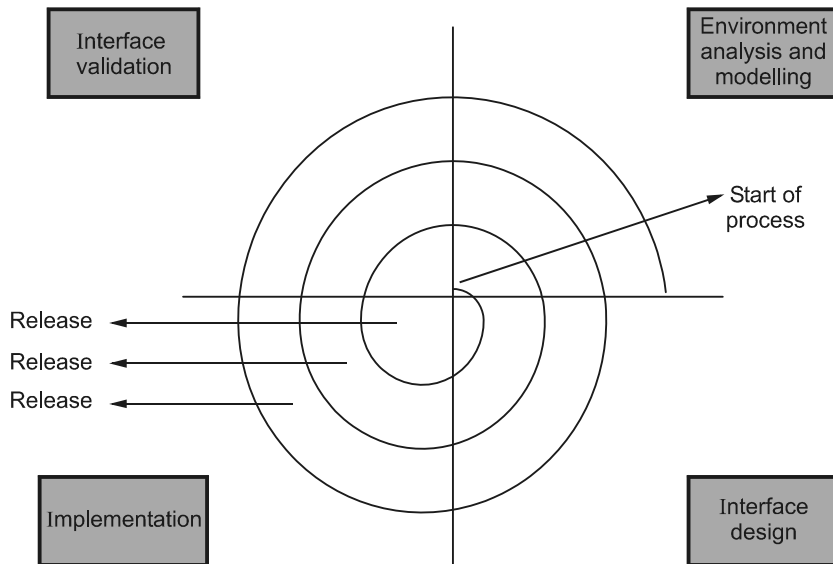


Fig. 3.12.2 Interface analysis and design process

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.

3.12.3 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.

3.12.3.1 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.

3.12.3.2 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. 3.12.3 on next page.

As shown in Fig. 3.12.3 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 up the information such as source and destination city, date and time of travel, number of passengers and type of reservation.

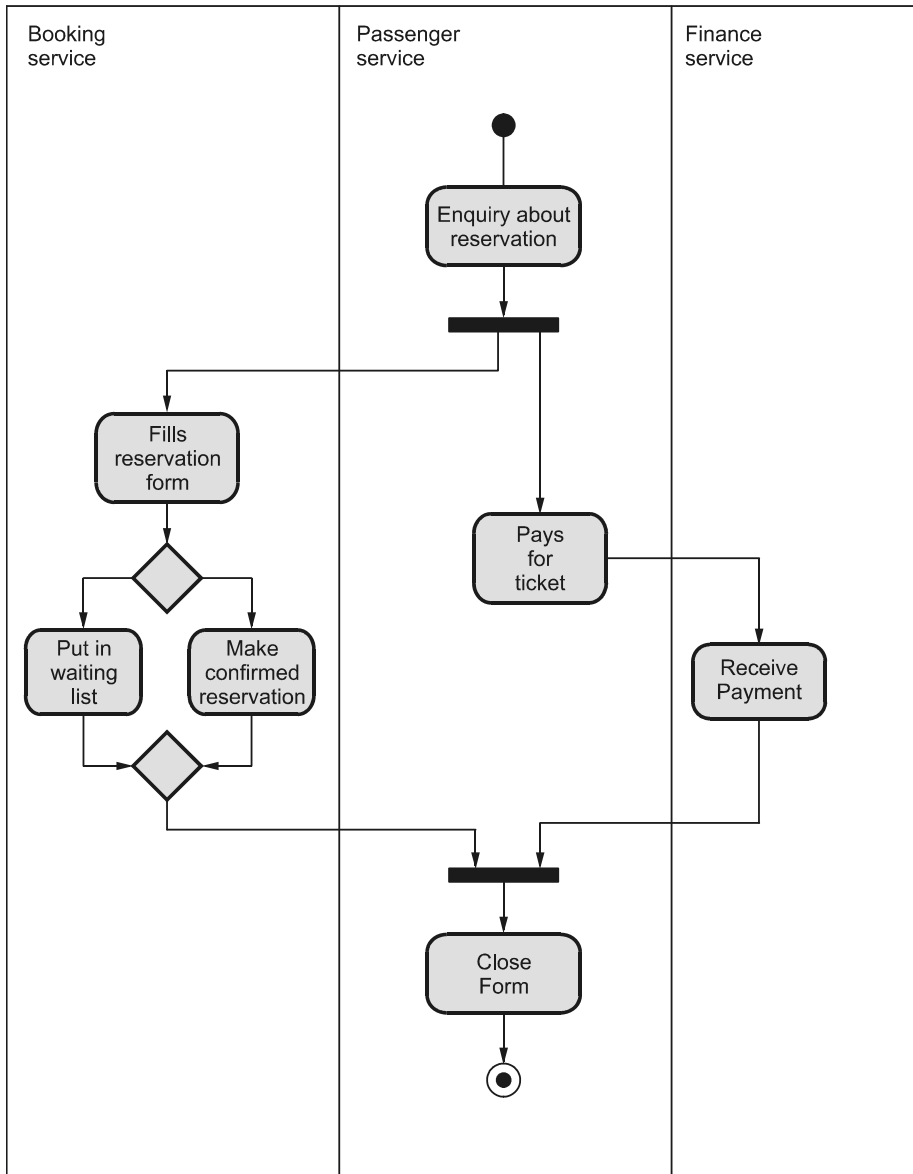


Fig. 3.12.3

- 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.

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**

3.12.3.3 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.

3.12.3.4 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.

3.12.4 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 –

3.12.4.1 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

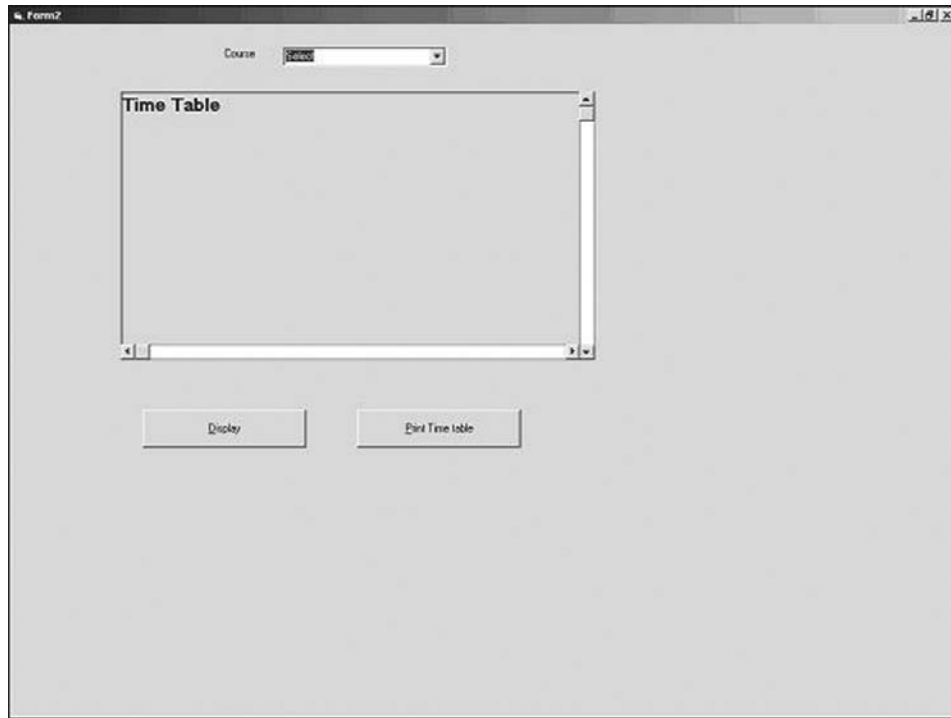
"A **student** will be typical user of this system. This user has to fill up an online **registration form**. In this form he has to submit the **student information** and then he has to select the **courses** which he/she wants to adopt."

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.

The screenshot shows a web-based registration form. The title bar reads 'On line Student Registration form'. The form has a light gray background. The 'Student Information' section is enclosed in a box and contains several input fields: 'Name' (text), 'Academic Year' (dropdown menu), 'Student ID' (text), 'Status' (text), 'Address' (text), 'Telephone number' (text), 'Subject' (text), and 'Marks obtained in %' (dropdown menu). Below this, the 'Course Selection' section contains 'Department' and 'Course' dropdown menus. At the bottom of the form are two buttons labeled 'Submit' and 'Exit'.

Fig. 3.12.4

If a **teacher** wants to find the particular student teacher will put the student ID and can get the complete information of him.

**Fig. 3.12.5****Fig. 3.12.6**

3.12.4.2 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.

3.12.4.3 Design Issues

There are four important interface design issues that are depicted by following Fig. 3.12.7.

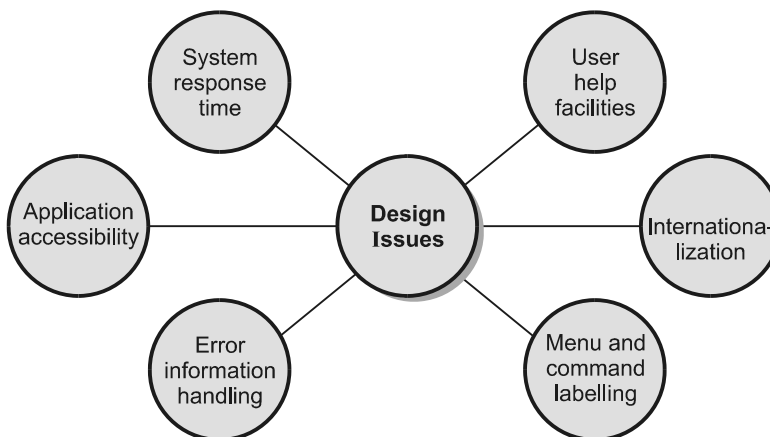


Fig. 3.12.7 Design issues

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).

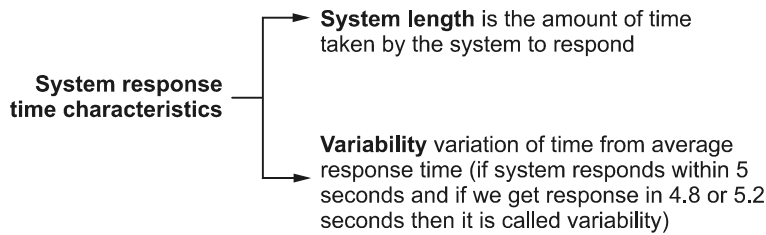


Fig. 3.12.8

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**

3.13 Component Level Design**AU : May-17, 18, 16, 15, Dec.-16, 15,19, May-16**

- Component is nothing but a model for computer software.
- The OMG Unified Modeling Language Specification defines the concept of component more formally and it is -

" Component is a modular, deployable and replaceable part of system that encapsulates the implementation and exposes the set of interfaces".

- Components are the part of software architecture and hence they are important factors in achieving the objectives and requirements of system to be built.
- Components can communicate and collaborate with other components.
- Design models can be prepared using object oriented views and conventional views.

3.13.1 Designing Class based Components

- Component is represented as a part of architectural model. It collects the information about the system as a part of analysis model.
- If **object oriented software engineering approach** is adopted then the component level design will emphasize on elaboration of analysis classes and refinement of **infrastructure classes**.
- The detailed description of attributes, operations and interfaces of these infrastructure classes is required during the system building.

3.13.1.1 Basic Design Principle

There are four design principles that are used during the component level design. These principles are -

1. The Open-closed Principle

This principle states that - A module or a component should be open for extension and should be closed for modification. A designer should design a component in such a manner that some functionalities can added to it if required but in doing so there should not be any change in the internal design of the component itself.

2. The Liskov Substitution Principle

This principle states that - subclasses should be substitutable for their base classes. This principle is proposed by **Barbara Liskov**. A component that contains the base class and if that component works properly then the same component should work properly even if the derived class of that base class is used by the component as a substitute.

3. Dependency Inversion Principle

The components should be dependant on the other abstract components and not on the concrete component. This is because if some component has to be extended then it becomes easy to enhance it using other abstract component instead of concrete component.

4. The interface Segregation Principle

Many client specific interfaces are better than one general purpose interface. According to this principle, designer should create specialized interfaces for each major category. So that the clients can access the interfaces based on the category. If a general purpose interface is created then multiple clients may try to access it for different operations at the same time.

3.13.1.2 Component Level Design Guideline

Ambler suggested following guideline for conducting component level design -

Components : Components are the part of architectural model. The naming conventions should be established for components. The component names should be specified from the problem domain. These names should be meaningful.

Interfaces : Interfaces serve important role in communication and collaboration. The interface should be drawn as a circle with a solid line connecting it to another element. This indicates that the element to which it is connected offers the interface. The interface should flow from left side of component. Only important interfaces must be drawn.

Dependencies and interfaces : The dependencies must be shown from left to right. The inheritance should be shown from bottom to top i.e. from derived to base class. The component interdependencies should be represented via interfaces.

3.13.1.3 Cohesion

In component level design for object oriented systems, the cohesion means a class or a component that consists of data and operations that are closely related to each other and related to the class itself. Various types of cohesions are -

Functional : In this type of cohesion the each module performs only one component.

Layer : This type of cohesion is used in packages, components and classes. In this type of cohesion the higher level layer access the functionalities of the lower levels but the lower level layers do not access the functionalities of the higher level layers.

Communicational : When all the operations of the class access the same set of data then this type of cohesion occurs.

Sequential : In this type of cohesion, the components are grouped together in such a manner that the output of one operation is provided as input to another operation. So that all the operations execute in some specific sequence.

Procedural : In this cohesion the procedures are invoked immediately one after the another.

Temporal : The cohesion in which the operations specify specific behaviour is called temporal cohesion.

Utility : The components, classes or operations are grouped together if they perform some specific operation otherwise they are not related with each other.

3.13.1.4 Coupling

Coupling is a mechanism of degree to which the classes are connected to one another.

Various types of coupling are

- Content coupling
- Common coupling
- Control coupling
- Stamp coupling
- Data coupling
- External coupling

3.13.2 Traditional Components



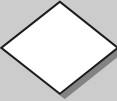
- **Component level design** is also called as **procedural design**. After data, architectural and interface design the component level design occurs.
- The **goal** of component level design is to translate design model into operational software.
- Graphical, tabular or text based notations are used to create a set of **structured programming constructs**. These structured programming constructs translate each component into **procedural design model**. Hence the work product of component level design is procedural design model.

3.13.2.1 Structured Programming

- There are **three constructs** of structured programming
 1. Sequence
 2. Condition
 3. Repetition
- Sequence** denotes a linear processing of the statements. **Condition** provides the facility to test the logical conditions (For example if-then-else conditions). **Repetition** is used to denote the looping.
- Following are the advantages of using structured programming constructs -
 1. The structured programming constructs **reduces** program **complexity**.
 2. It enhances **readability**, **testability** and **maintainability** of the procedure.
 3. The structured programming constructs are the **logical chunks** that allow a reader to recognize procedural element from each programming module. This ultimately enhances the readability of the program.

Graphical Design Notations

The structured programming constructs can be represented by graphical notations. These graphical notations are called **flow chart**. These notations are as follows -

Graphical Notation	Name	Meaning
	Box	It is used for processing step
	Flow of control	It is used to represent the flow of control from one construct to another.
	Diamond	It is used for representing the conditions such as if-then-else or repeat until.

With help of above graphical notations the three programming constructs can be represented as follows –

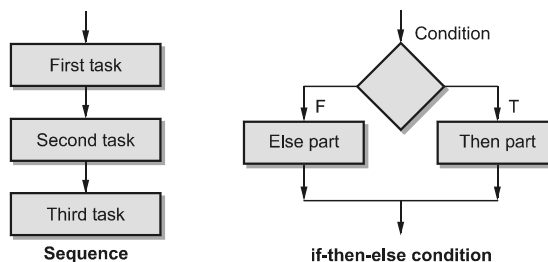


Fig. 3.13.1 (a)

These programming constructs can be nested one. That means one programming construct can be within another construct.

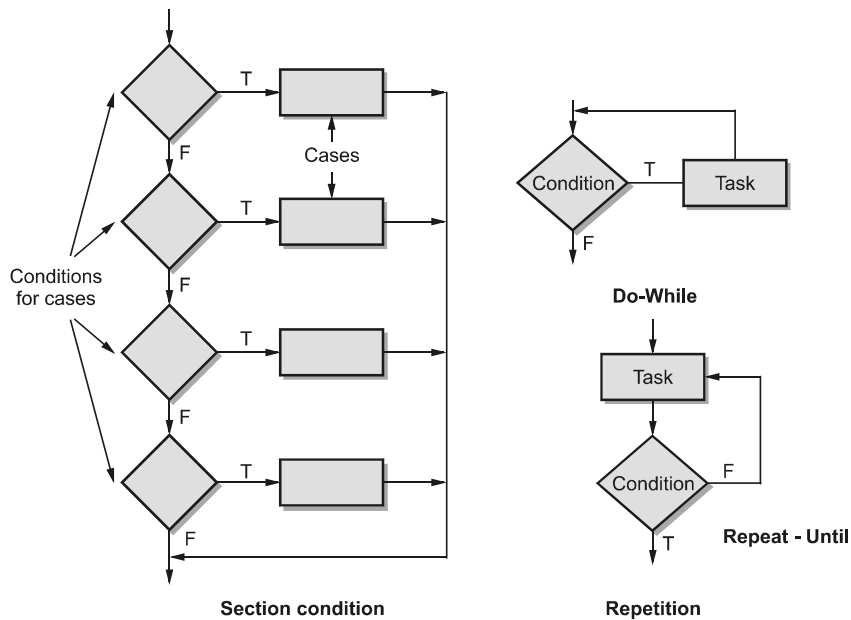


Fig. 3.13.1 (b) Flowchart

Another graphical notation is **box diagram**. This notation is also called as **Nassi and Shneiderman chart** (Nassi and Shneiderman are the names of the developer of this notations) or **NS chart**. These notations are shown in Fig. 3.17.2.

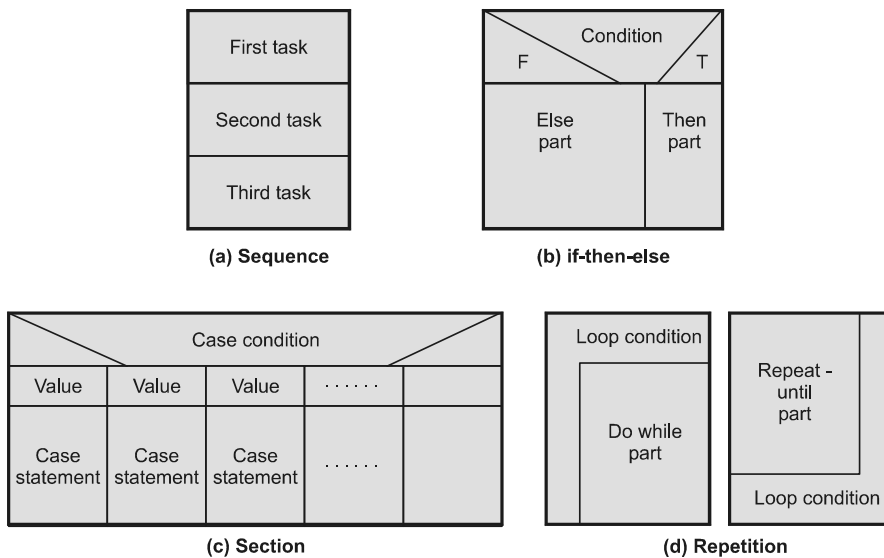


Fig. 3.13.2

Following are the **characteristics of the NS chart** -

1. The scope of the programming constructs such as repetition, if-then-else is well defined.
2. Arbitrary transfer of the control is not possible using this notation.
3. Recursion can be represented conveniently
4. The scope of local and global data can be defined systematically.

Tabular Design Notations

Programming constructs can be represented by tabular design notation. This an alternative representation to the graphical notations.

There are four sections in the tabular representation method -

1. The first section is at the upper left corner. It consists of list of conditions.
2. The second section is the lower left hand corner. It consists of list of actions.
3. The right hand portion is a matrix which represents the combination of conditions and actions. The combination of condition and actions together form **processing rules** for that particular procedure.

Following steps are applied to develop a decision table -

- List all the actions associated with particular procedure.
- List all the conditions associated with particular procedure.
- Associate each condition with each corresponding action. Any impossible permutation must be eliminated. All possible permutations of conditions and corresponding actions must be represented in the table.
- Create the processing rule indicating that particular action exists on particular condition.

For example -

Conditions	1	2	3	4	5			n
Condition 1	✓	✓			✓			
Condition 2		✓	✓					
Condition 3	✓			✓	✓			

Actions								
Action 1	✓		✓	✓				
Action 2		✓	✓	✓				
Action 3	✓				✓			

Program Design Language (PDL)

A program design language is also called pseudo code or structured English. This is similar to English but is used as a generic reference for design language. The PDL is not compiled. It is used to translate the design into the programming language.

A basic PDL syntax should possess the programming constructs for interface description, procedure definition, data declaration, condition constructs, repetition constructs and I/O constructs.

For example : Following is a PDL which demonstrates the procedure for searching the name John from the table. IF the name is found then index of the table is returned

```

Search(table,number of items)
  Set count to Zero
  Read first item from table
  DO FOR count is <=number of items
    IF table.name="John"
      RETURN table.index
    ELSE
      count=count+1;
      Read Next item from table
    ENDIF
  ENDFOR
END

```

Example 3.13.1 For a case study of your choice show the architectural and component design.

AU : May-15, Marks 16

Solution : Inventory control system

Problem statement : The inventory control system is handled by the inventory manager. He first logs in to the system. For checking the availability of the item he checks

the inventory. He can purchase the items for the stock and then updates the database accordingly. If any expired items are present in the stock then he removes those items. An order processing clerk can demand for the required items from the stock and then inventory manager processes this order. The billing of the items is prepared and is sent to the accounts department. The accountant pays the bills for the purchased stock. Similarly he deposits the money received when the order gets fulfilled for the customer. The stock summary report is generated periodically by the Inventory manager.

Architecture design : The architecture design represent the structure of the system. It can be represented using architectural context diagram. Refer Fig. 3.13.3.

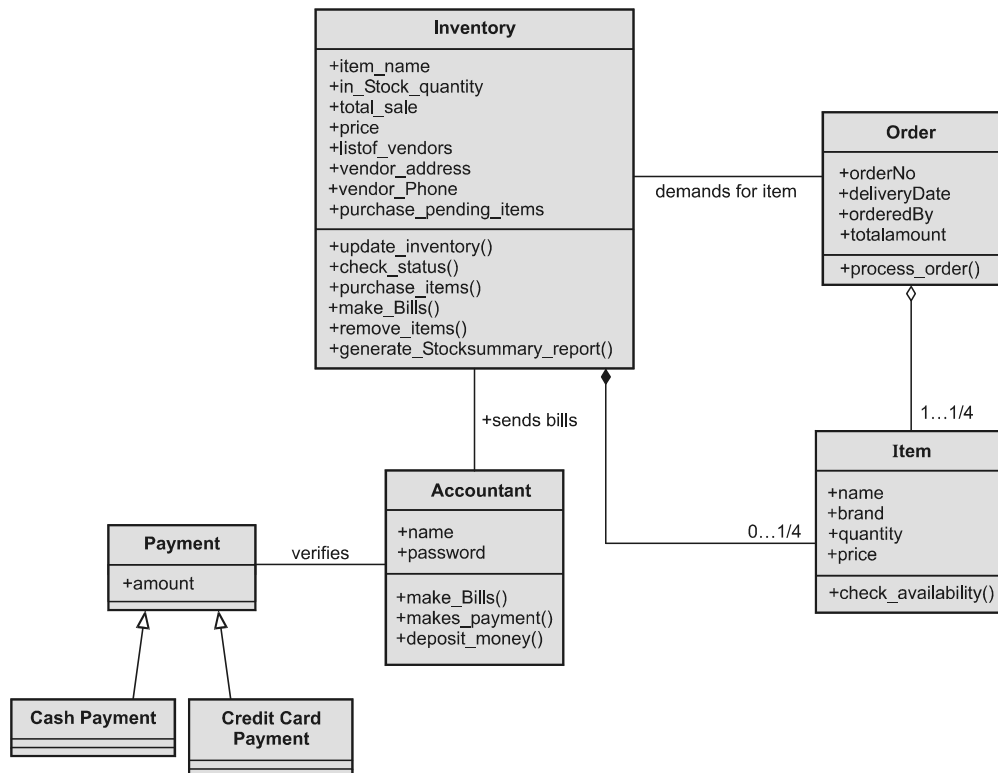


Fig.3.13.3

Component design : Component is represented as a part of architectural model. It collects the information about the system as a part of analysis model. The component level design can be done using the object oriented approach. If this approach is adopted then component level design will emphasize on elaboration of analysis classes and refinement of infrastructure classes. The class diagram is a common method of representing the component design.

Review Questions

1. Write short note -user-component level design. **AU : May-16, Marks 4**
2. Explain component level design with suitable examples. **AU : Dec.-16, Marks 8**
3. Explain the various coupling and cohesion methods used in software design. **AU : May-15, Marks 16**
4. Explain the cohesion and coupling types with examples. **AU : Dec.-15, May-18, Marks 16**
5. What is cohesion? How it is related to coupling? Discuss in detail different types of cohesion and coupling with suitable example. **AU : May-17, Marks 13**
6. Outline the steps in designing class based components with an example. **AU : Dec.-19, Marks 13**

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 What do you mean by horizontal and vertical partitioning ? (Refer section 3.7.1)

AU : IT, May-03

Q.4 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.5 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.6 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.7 What is meant by transaction mapping ? How it is used in software design ?**AU : IT, Dec.-04**

Ans. : In transaction mapping the user interaction subsystem is considered and DFD is mapped into transaction processing structure.

In transaction mapping technique, the user command which is given as input, flows into the system and produces more information flows, ultimately causes the output flow from the DFD.

Q.8 Distinguish between horizontal partitioning and vertical partitioning.**AU : IT, May-05**

Ans. :

Horizontal partitioning	Vertical partitioning
Horizontal partitioning can be done by partitioning system into: input, data transformation (processing) and output.	Vertical partitioning suggests the control and work should be distributed top-down in program structure.
This kind of partitioning have fewer side effects in change propagation or error propagation.	Vertical partitioning define separate branches of the module hierarchy for each major function. Hence these are easy to maintain the changes.

Q.9 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.10 What are the quality parameters considered for effective modular design ?

OR

State different criteria's applied to evaluate 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.11 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.12 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.13 "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 Meyers. 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.14 List out the elements of analysis model.

AU : IT, May-07

Ans. : The elements of analysis model are :

1. Data dictionary
2. Entity relationship diagram
3. Data flow diagram
4. State transition diagram
5. Control specification
6. Process specification

Q.15 What are the types of coupling ? (Refer section 3.3.7.2)

AU : IT, May-07

Q.16 Name the three levels of abstraction, which are in practice for the design.

(Refer section 3.3.1)

AU : IT, May-07

Q.17 What are the steps involved in design stage of a software ? (Refer section 3.2)

AU : CSE, May-07

Q.18 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	<ul style="list-style-type: none"> • Software requirement specification(SRS) • Project plan (it includes RMMM plan also) 	<ul style="list-style-type: none"> • The requirement gathering and analysis is made in this document. <p>The purpose of this document is to identify scope of the project, effort estimation and determine the project schedule.</p>
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 contains 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.19 What is an architectural style ? (Refer section 3.9)

AU : CSE, Dec.-07

Q.20 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.21 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.22 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.23 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.24 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.25 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.26 Define interface design. (Refer section 3.12.4)**AU : IT, Dec.-09****Q.27 List the architectural models that can be developed.****AU : Dec.-10**

Ans. : Following are the architectural models that can be developed -

1. Data centered architectural
2. Data flow architectures
3. Call and return architecture
4. Object oriented architecture
5. Layered architectures

Q.28 List out design methods.**AU : May-12**

Ans : The two software design methods are -

1. Object oriented design
2. Function oriented design

Q.29 What are the design quality attributes 'FURPS' meant ?**AU : Dec.-12**

Ans. : The design attributes FURPS stands for

- **Functionality :** Functionality can be checked by assessing the set of features and capabilities of the functions.
- **Usability :** The usability can be assessed by knowing the usefulness of the system
- **Reliability :** Reliability is a measure of frequency and severity of failure.
- **Performance :** It is a measure that represents the response of the system.
- **Supportability :** It is also called maintainability. It is the ability to adopt the enhancement or changes made in the software.

Q.30 Define data abstraction.**AU : May-13**

Ans. : Data abstraction is a kind of representation of data in which the implementation details are hidden.

Q.31 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.32 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.33 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.34 Draw diagrams to demonstrate the architectural styles.

AU : May-15

Ans. : The commonly used architectural styles are

1. Data centered architecture - Refer Fig. 3.9.1.
2. Data flow architecture - Refer Fig. 3.9.2.
3. Object oriented architecture - Refer Fig. 3.9.5.
4. Layered architecture - Refer Fig. 3.9.6.

Q.35 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.36 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.37 Write a note on FURPS model of design quality.

AU : Dec.-15,17

Ans. : FURPS stands for Functionality, Usability, Reliability, Performance and Supportability. These are the quality attributes used to measure design quality.

Q.38 What is the need for architectural mapping using data flow ?

AU : May-16

Ans. : Transform mapping and transaction mapping is used for architectural mapping using data flow diagrams.

Q.39 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.40 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.41 What UI design patterns are used for the following ?

a) Page layout b) Tables

c) Navigation through menus and web pages

d) Shopping cart

AU : Dec.-16, May-17,18

Ans. :

(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.42 What methods are used for breaking very long expression and statements ?

AU : Dec.-16

Ans. : 1) More control structures such as switch-case, if-else statements can be used
2) Make use of user-defined function for sub-expressions and call them.

Q.43 What is inheritance ?

AU : Dec.-19

Ans. : Inheritance is a feature of object oriented design in which the child module makes use of some of the characteristics of its parent module.

Q.44 Define a component. Give example.

AU : Dec.-19

Ans. : Component is a modular, deployable and replaceable part of system that encapsulates the implementation and exposes the set of interfaces.

For example - PrintJob, CalculateTax can be the components.



Notes

4

Testing and Maintenance

Syllabus

Software testing fundamentals - Internal and external views of Testing - White box testing - Basis path testing - control structure testing-black box testing- Regression Testing -Unit Testing - Integration Testing - Validation Testing - System Testing And Debugging - Software Implementation Techniques: Coding practices-Refactoring-Maintenance and Reengineering-BPR model-Reengineering process model-Reverse and Forward Engineering.

Contents

4.1 Definition of Testing	May-07,16,Dec.-04,05,08,10 Marks 8
4.2 Internal and External Views of Testing	
4.3 White Box Testing	May-04,07,08,15,17,18,19,
.....	Dec.-07, 11, 13, 14, 16, 17 . Marks 10
4.4 Black Box Testing	May-07, 12, 15, 16, 17,19
.....	Dec.-03, 04, 11, 13, 15, 16 . Marks 16
4.5 Comparison between Black Box Testing and White Box Testing	
.....	May-04,07,19, Dec.-05,17, .. Marks 6
4.6 Testing Strategy	May-05, 06, Marks 16
4.7 Unit Testing	Dec.-06,08,15,
.....	May-03, 09, Marks 16
4.8 Integration Testing	
.....	Dec.-08,15,19
.....	May-03,04,05,09,13,14,15,17,18
.....	Marks 16
4.9 Validation Testing	May-05,16, 19 Marks 16
4.10 System Testing	May-03,04,05, Dec.-14, Marks 16
4.11 Debugging	Dec.-10,14, May-15, 18, Marks 6
4.12 Software Implementation Techniques.....	May-16, 18, 19, Dec.-16, Marks 8
4.13 Maintenance and Reengineering	
4.14 Business Process Reengineering	Dec.-19..... Marks 8
4.15 Reengineering Process Model.....	May-19 Marks 11
4.16 Reverse and Forward Engineering	Dec.-19..... Marks 5
Two Marks Questions with Answers	

4.1 Definition of 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 decide 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 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.

4.3.1 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.

4.3.2 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 $n - 1, n, n + 1$.

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.

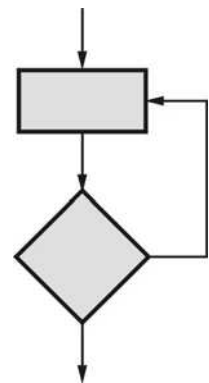


Fig. 4.3.1 Simple loop

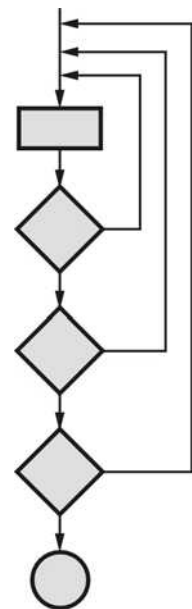


Fig. 4.3.2 Nested loops

3. Concatenated loops :

The concatenated loops can be tested in the same manner as simple loop tests.

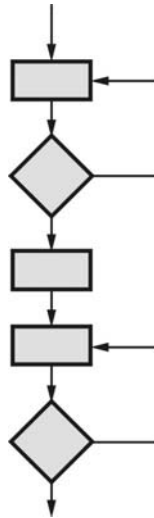


Fig. 4.3.3 Concatenated loops

4. Unstructured loops :

The testing cannot be effectively conducted for unstructured loops. Hence these types of loops needs to be redesigned.

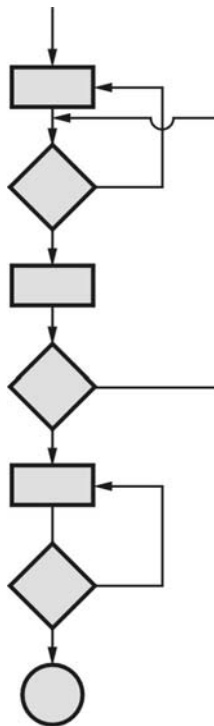


Fig. 4.3.4 Unstructured loops

4.3.3 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

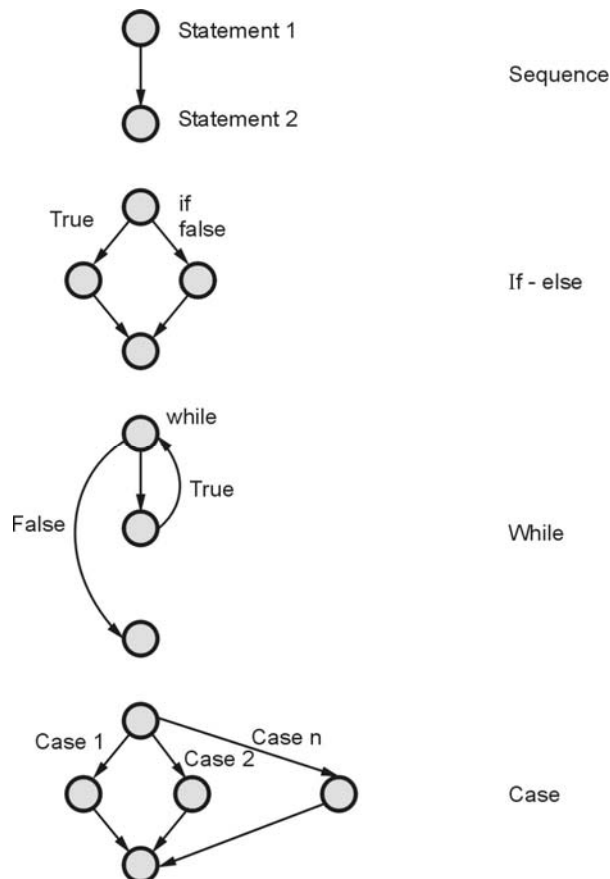


Fig. 4.3.5

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

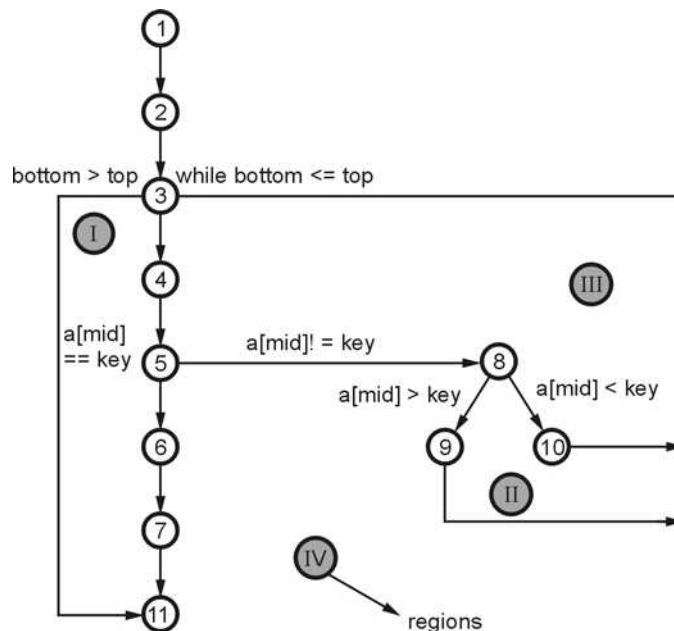


Fig. 4.3.6

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 = $E - N + 2 = 13 \text{ edges} - 11 \text{ nodes} + 2$
 $= 2 + 2 = 4$
- 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 4.3.1 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. 4.3.7. (See Fig. 4.3.7 on next page)

Example 4.3.2 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

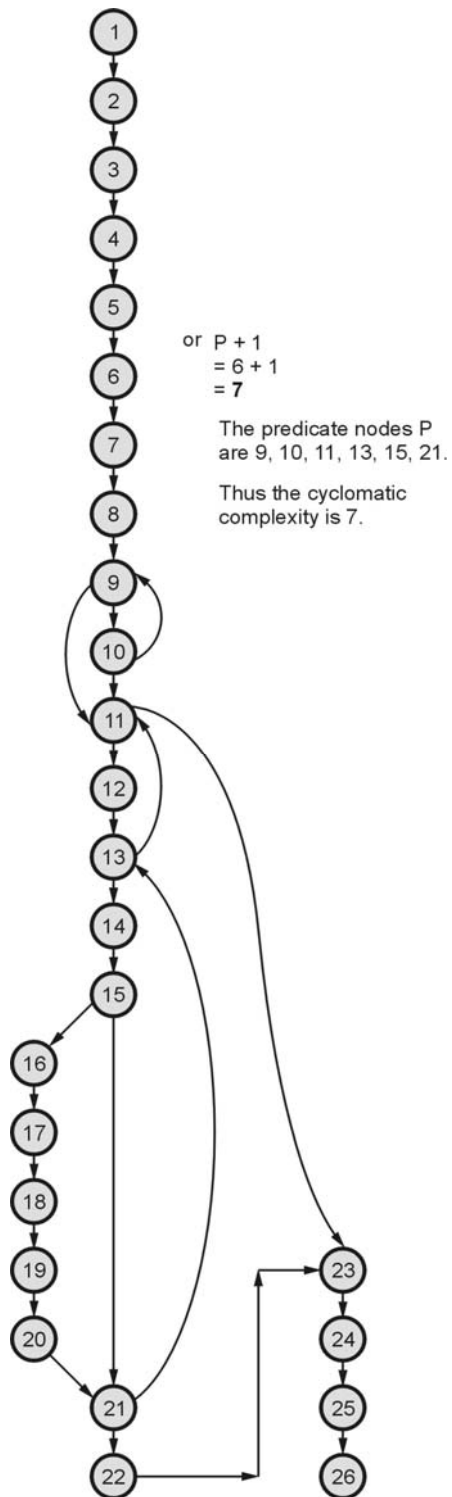


Fig. 4.3.7

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. 4.3.8 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.

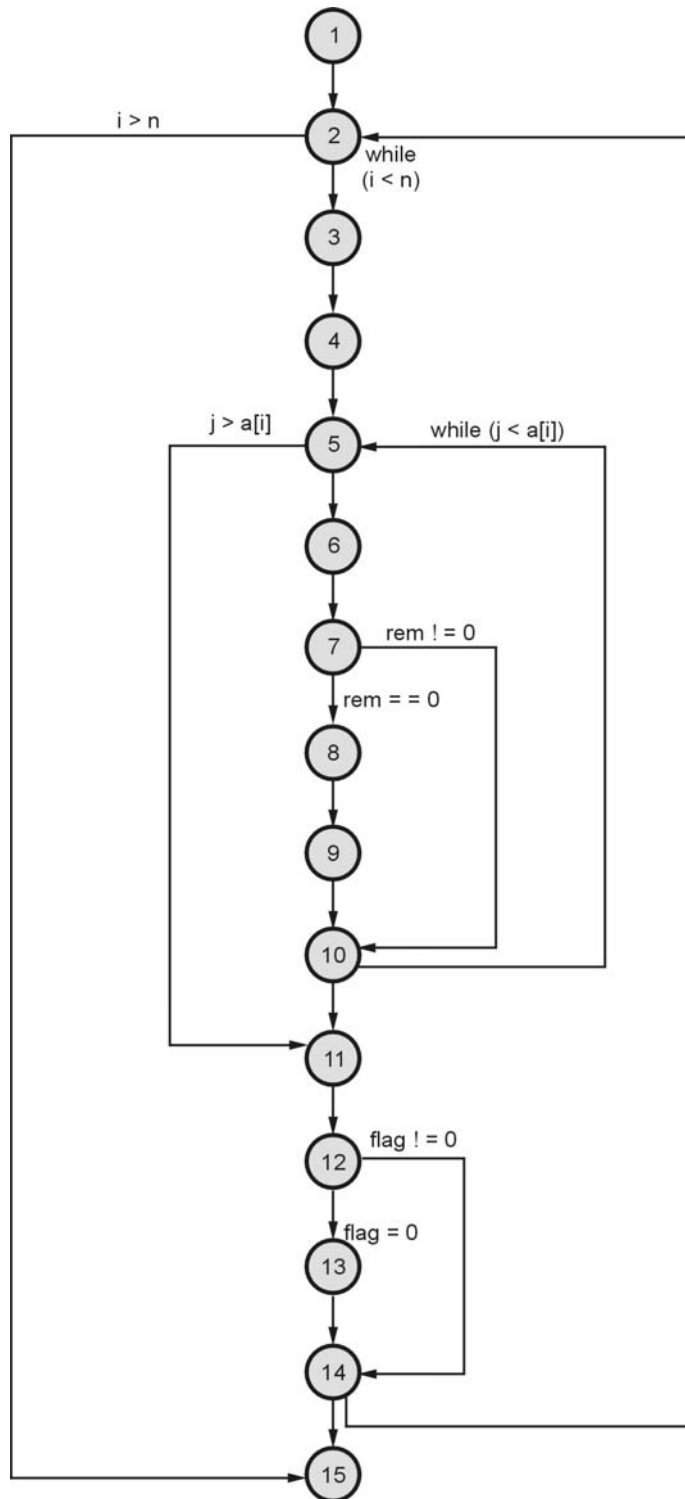


Fig. 4.3.8

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 4.3.3 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. 4.3.9 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

$$\begin{aligned}
 \text{Cyclomatic complexity} &= E - N + 2 \\
 &= 25 - 21 + 2 \\
 &= 6
 \end{aligned}$$

Method 2 : The total number of regions in Fig. 4.3.9 = 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

$$\begin{aligned}
 \therefore \text{Cyclomatic complexity} &= P + 1 \\
 &= 5 + 1 \\
 &= 6
 \end{aligned}$$

Thus the cyclomatic complexity is 6.

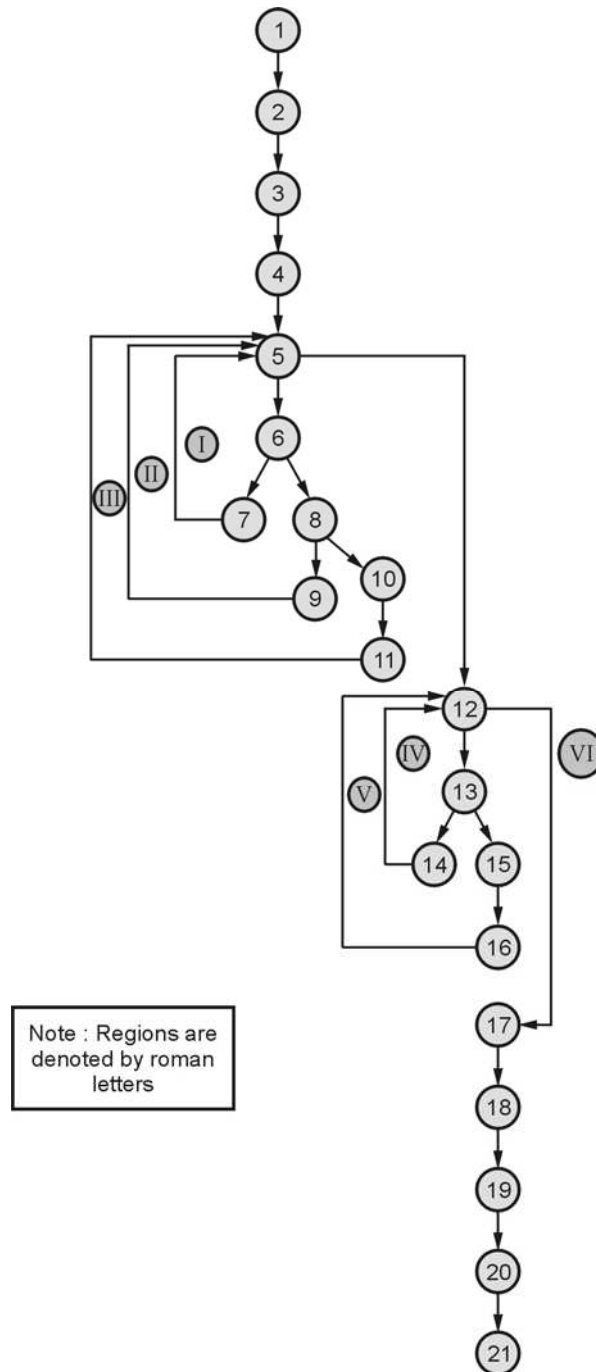


Fig. 4.3.9

Example 4.3.4 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. 4.3.10)

- 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

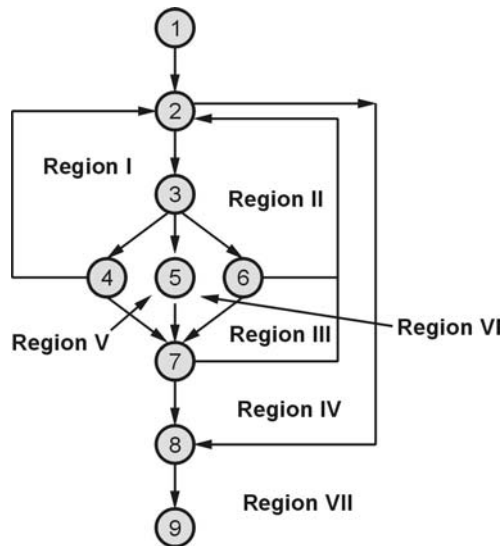


Fig. 4.3.10

Example 4.3.5 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 4.3.6 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 4.3.3.

Example 4.3.7 Consider the pseudo code for simple subtraction given below

1) Program 'Simple subtraction'

2) Input (x, y)

3) Output (x)

4) Output (y)

5) If $x > y$ then DO

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. 4.3.11

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

$$\therefore 9 - 9 + 2 = 2$$

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

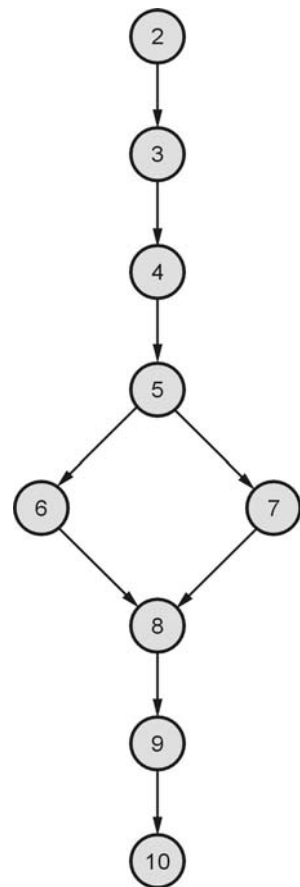


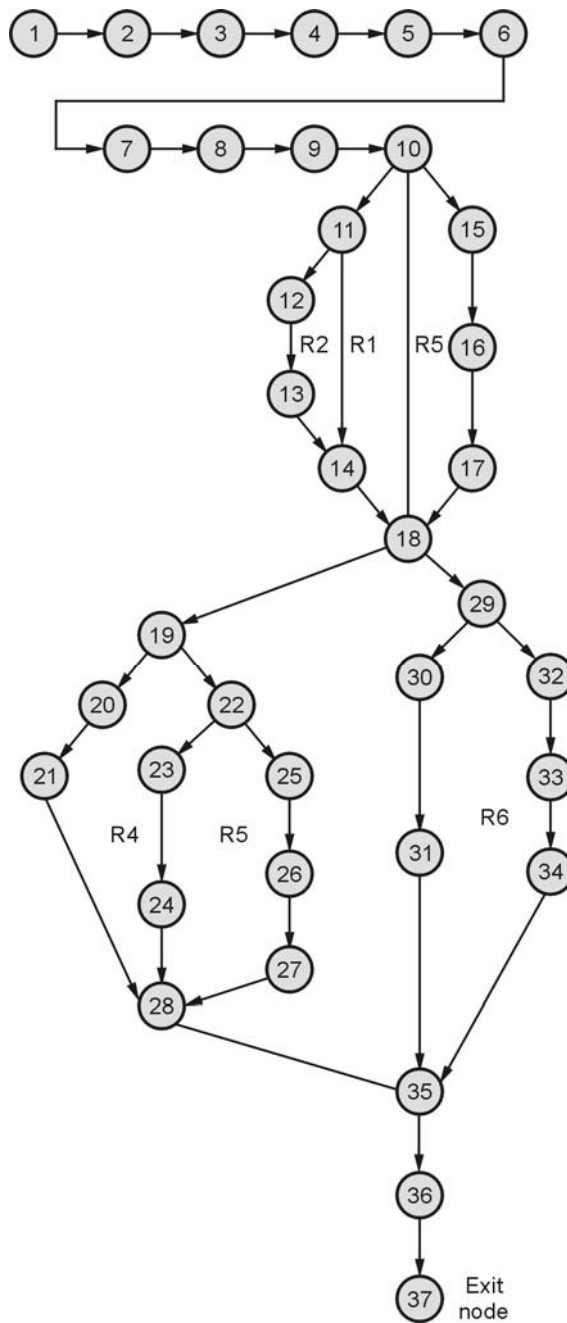
Fig. 4.3.11

Example 4.3.8 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**

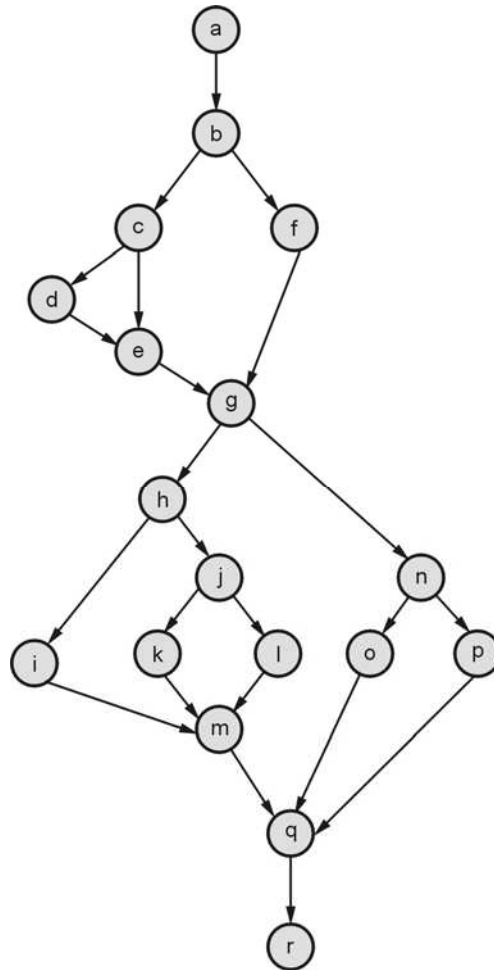
Solution : Step 1 : We will write the procedure for finding the type of triangle. Each important step is numbered so that we can draw the flow graph in the next step

```
(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) || (b == c) || (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

$$= 6 + 1 \text{ where } b, c, g, h, j \text{ and } n \text{ are predicate nodes}$$

$$= 7$$

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.
2. Explain how the various types of loops are tested.

AU : May-17, Marks 7

AU : Dec.-17, Marks 9

4.4 Black Box Testing

AU : May-07,12,15,16,17,19, Dec.-03,04,11,13,15,16, 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

4.4.1 Equivalence Partitioning

- 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.

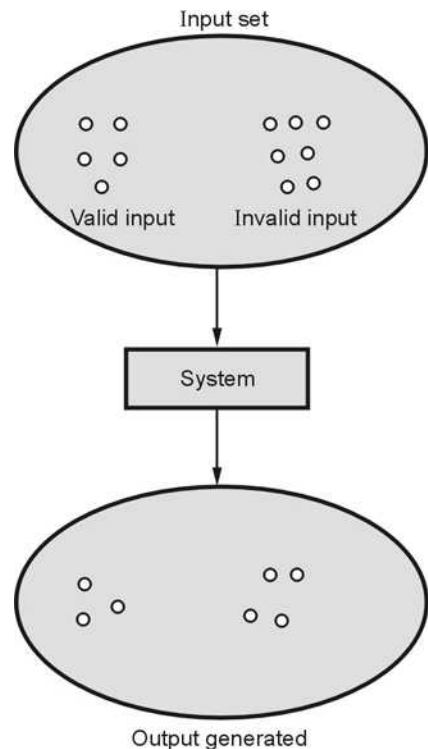


Fig. 4.4.1

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 4.4.1 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 4.4

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 $m_1 = m_2, c_1 = c_2$

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)

4.4.2 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 4.4.2 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 4.4.3 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 H_2 and O_2 by releasing oxygen.

Example 4.4.4 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

$$I_1 = \{ \text{Integer} < 2000 \}$$

$$I_2 = \{ \text{Integer} = 2000 \}$$

$$I_3 = \{ \text{Integer} > 2000 \text{ and } < 8000 \}$$

$$I_4 = \{ \text{Integer} = 8000 \}$$

$$I_5 = \{ \text{Integer} > 8000 \}$$

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 4.4.5 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 4.4.6 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

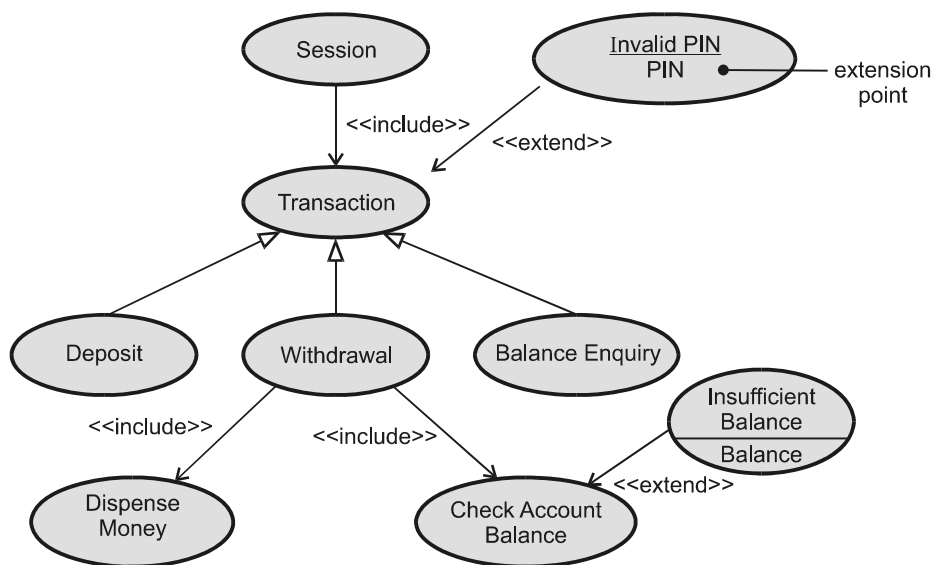


Fig. 4.4.2 Withdrawal of money from ATM

(ii) Activity diagram

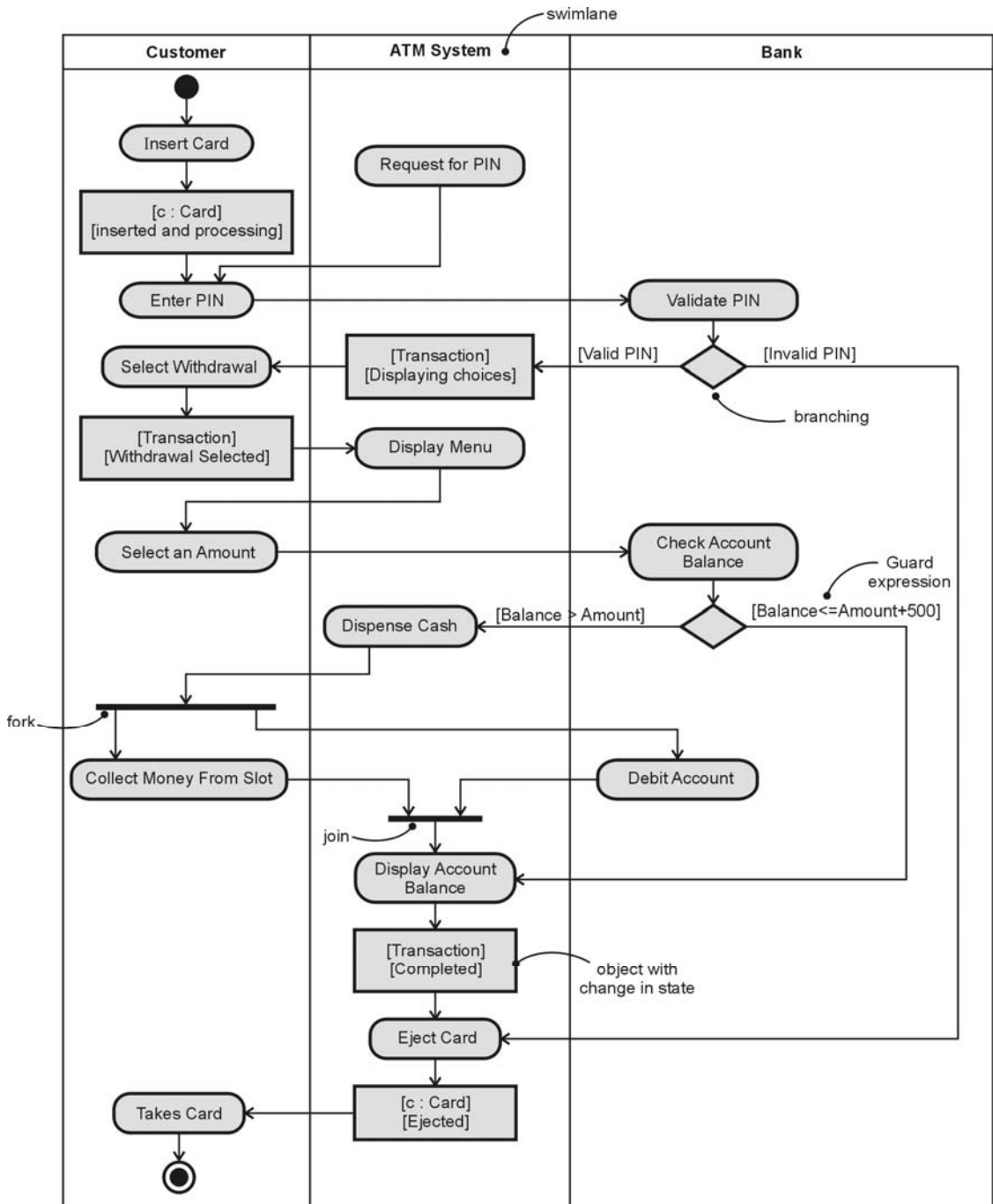


Fig. 4.4.3 Activity model

(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**

4.5 Comparison between Black Box Testing and White Box Testing**AU : May-04,07,19, Dec.-05,17, Marks 6**

Sr. No.	Black box testing	White box testing
1.	Black box testing is called behavioural testing.	White box testing is called glass box testing.
2.	Black box testing examines some fundamental aspect of the system with little regard for internal logical structure of the software.	In white box testing the procedural details, all the logical paths, all the internal data structures are closely examined.
3.	During black box testing the program cannot be tested 100 percent.	White box testing lead to test the program thoroughly.
4.	This type of testing is suitable for large projects.	This type of testing is suitable for small projects.

Advantages and Disadvantages of Black Box Testing**Advantages :**

1. The black box testing focuses on fundamental aspect of system without being concerned for internal logical structure of the software.
2. The advantage of conducting black box testing is to uncover following types of errors.
 - i. Incorrect or missing functions
 - ii. Interface errors
 - iii. Errors in external data structures
 - iv. Performance errors
 - v. Initialization or termination errors

Disadvantages :

1. All the independent paths within a module cannot be tested.
2. Logical decisions along with their true and false sides cannot be tested.
3. All the loops and the boundaries of these loops cannot be exercised with black box testing.
4. Internal data structure cannot be validated.

Advantages and Disadvantages of White Box Testing**Advantages :**

1. Each procedure can be tested thoroughly. The internal structures, data flows, logical paths, conditions and loops can be tested in detail.
2. It helps in optimizing the code.

3. White box testing can be easily automated.
4. Due to knowledge of internal coding structure it is easy to find out which type of input data can help in testing the application efficiently.

Disadvantages :

1. The knowledge of internal structure and coding is desired for the tested. Thus the skilled tester is required for whitebox testing. Due to this the testing cost is increased.
2. Sometimes it is difficult to test each and every path of the software and hence many paths may go untested.
3. Maintaining the white box testing is very difficult because it may use specialized tools like code analyzer, and debugging tools are required.
4. The missing functionality can not be identified.

Review Questions

1. Differentiate black box and white box testing.

AU : Dec.-17, Marks 4

2. Compare white box and black box testing.

AU : May-19, Marks 4

4.6 Testing Strategy

AU : May-05, 06, 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.

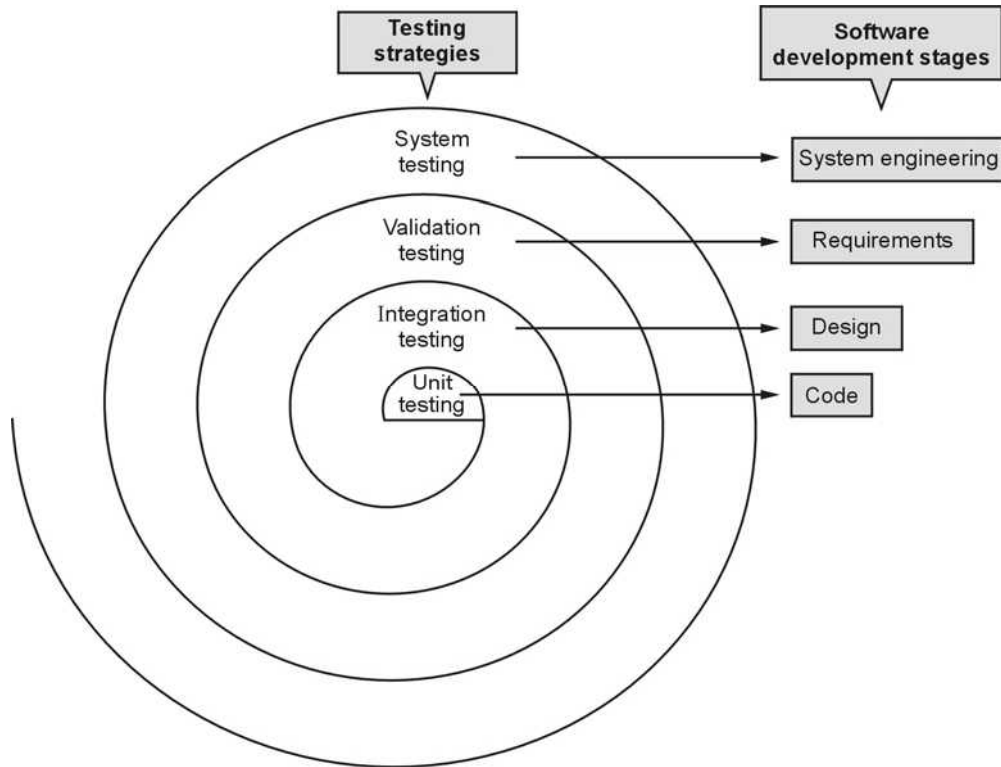


Fig. 4.6.1 Testing strategy

4.7 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.

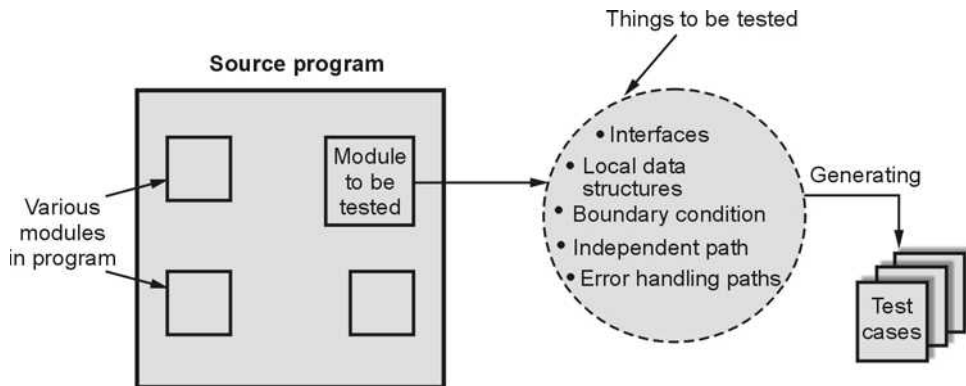


Fig. 4.7.1 Unit testing

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. 4.7.2.

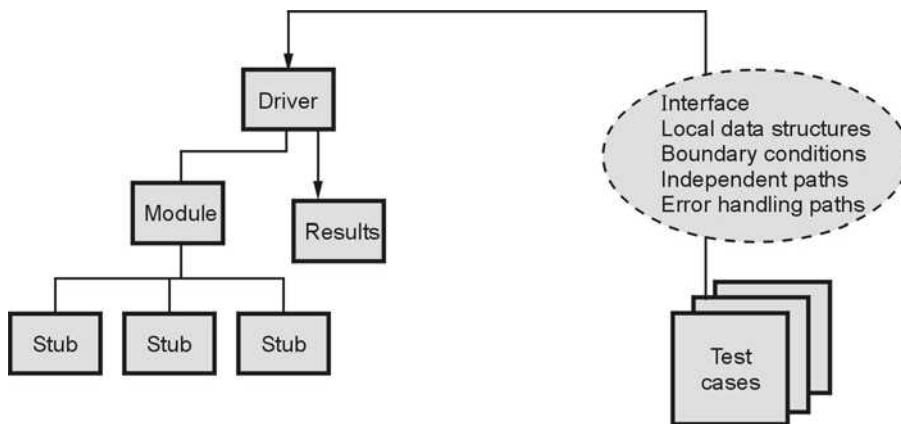


Fig. 4.7.2 Unit testing environment

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.

4.8 Integration Testing

AU : Dec.-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

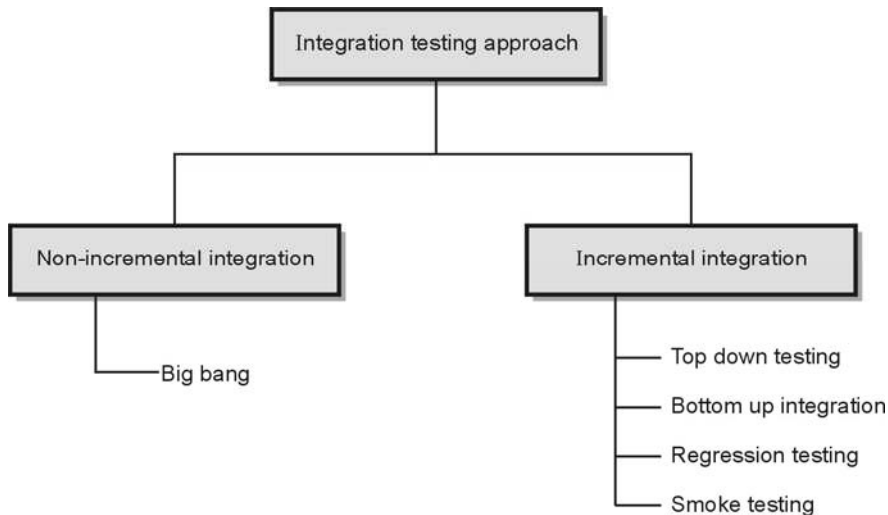


Fig. 4.8.1 Integration testing approach

- 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.

4.8.1 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 :

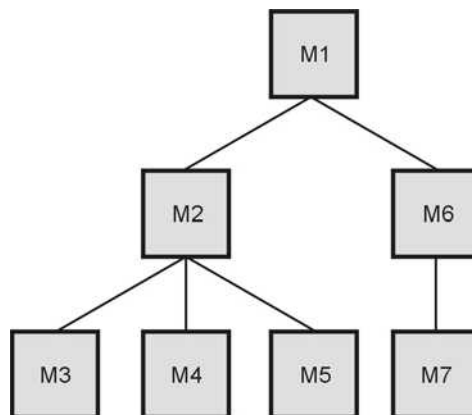


Fig. 4.8.2 Program structure

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.

4.8.2 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 :

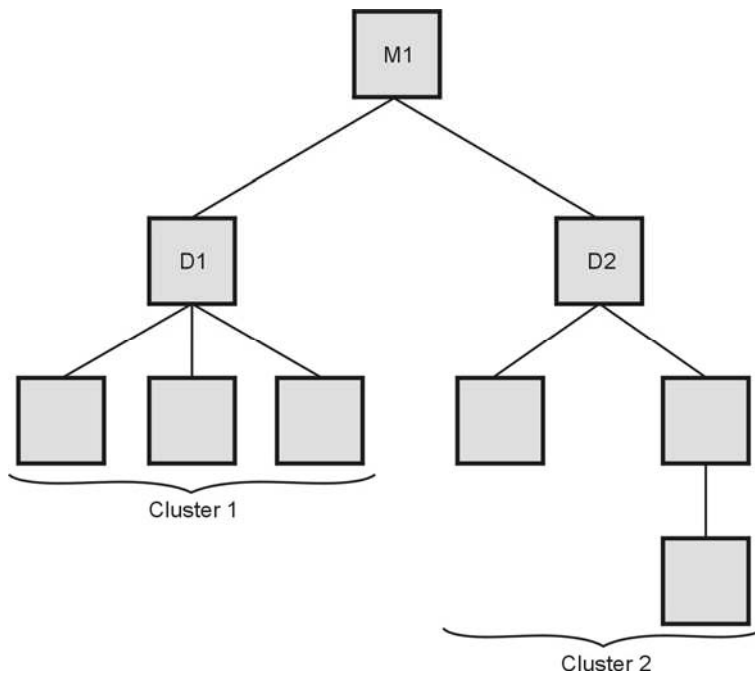


Fig. 4.8.3 Bottom up integration testing

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 :

Top-down Integration	Bottom-up Integration
The major controls or decisions are verified at early stage itself.	After integrating all the components at the bottom level, major controls or decisions can be verified.
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.
In top-down testing, testing stubs are created.	In bottom-up testing, test drivers are required.
Working system can be available at early stage.	Working system is available only after integrating all the components.

4.8.3 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.

4.8.4 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**

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 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.

4.10.1 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.

4.10.2 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.

4.10.3 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.

4.10.4 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

4.11 Debugging**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.

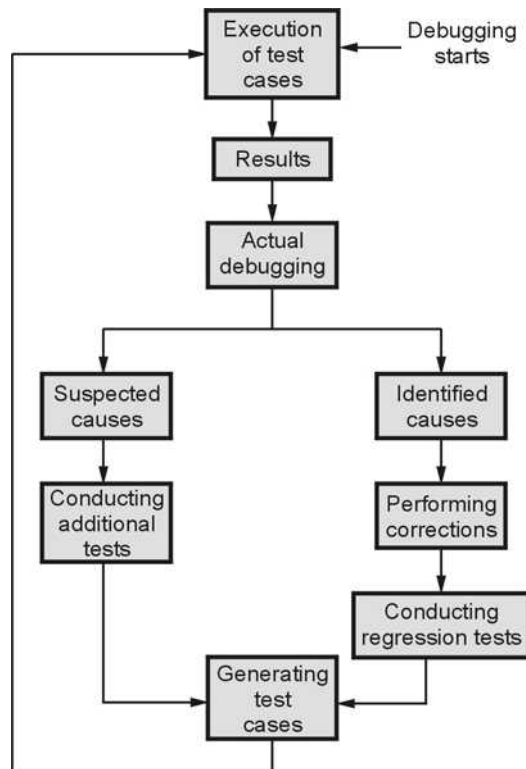


Fig. 4.11.1 Debugging process

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.

4.11.1 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

4.12 Software Implementation Techniques

AU : May-16,18,19, Dec.-16, Marks 8

After detailed system design we get a system design which can be transformed into implementation model. The goal coding is to implement the design in the best possible manner. Coding affects both testing and maintenance very deeply. The coding should be done in such a manner that the task of testing and maintenance phase should get simplified.

Various **objectives of coding** are -

1. Programs developed in coding should be readable.
2. They should execute efficiently.
3. The program should utilize less amount of memory.
4. The programs should not be lengthy.

If the objectives are clearly specified before the programmers then while coding they try to achieve the specified objectives. To achieve these objectives some programming principles must be followed.

4.12.1 Coding Practices

There are some commonly used programming practices that help in avoiding the common errors. These are enlisted below -

1. Control construct

The **single entry** and **single exit** constructs need to be used. The **standard control** constructs must be used instead of using wide variety of controls.

2. Use of gotos

The goto statements make the program unstructured and it also imposes overhead on compilation process. Hence avoid use of goto statements as far as possible and another alternative must be thought of.

3. Information hiding

Information hiding should be supported as far as possible. In that case only access functions to the data structures must be made visible and the information present in it must be hidden.

4. Nesting

Nesting means defining one structure inside another. If the nesting is too deep then it becomes hard to understand the code. Hence as far as possible - avoid deep nesting of the code.

For example

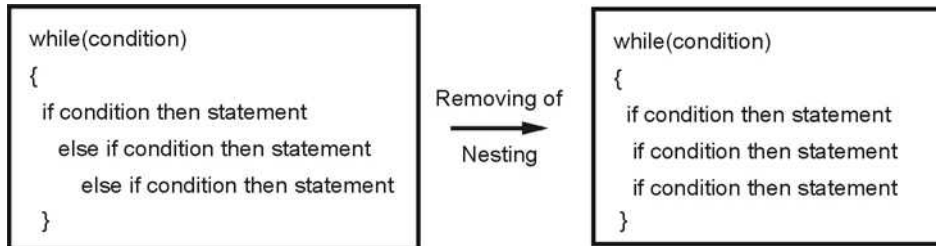


Fig. 4.12.1

Note that after removing of nesting the code becomes more readable but still the same output can be achieved.

5. User defined data types

Modern programming languages allow the user to use defined data types as the enumerated types. Use of user defined data types enhances the readability of the code.

For example : In C we can define the name of the days using enumerated datatype -

```
enum Day
{
    Sunday;
    Monday;
    Tuesday;
    Wednesday;
    Thursday;
    Friday;
    Saturday;
}workday;

enum Day Today=Friday;
```

6. Module size

There is no standard rule about the size of the module but the large size of the module will not be functionally cohesive.

7. Module interface

Complex module interface must be carefully examined. A simple rule of thumb is that the module interface with more than five parameters must be broken into multiple modules with simple interface.

8. Side effects

Avoid obscure side effects. If some part of the code is changed randomly then it will cause some side effect. For example if number of parameters passed to the function is changed then it will be difficult to understand the purpose of that function.

9. Robustness

The program is said to robust if it does something even though some unexceptional condition occurs. In such situations the programs do not crash but it exits gracefully.

10. Switch case with defaults

The choice being passed to the switch case statement may have some unpredictable value, and then the default case will help to execute the switch case statement without any problem. Hence it is a good practice to always have default case in the switch statement.

11. Empty catch block

If the exception is caught but if there is no action then it is not a good practice. Therefore take some default action even if it is just writing some print statement , whenever the exception is caught.

12. Empty if and while statements

In the if and while statements some conditions are checked. Hence if we write some empty block on checking these conditions then those checks are proved to be useless checks. Such useless checks should be avoided.

13. Check for read return

Many times the return values that we obtain from the read functions are not checked because we blindly believe that the desired result is present in the corresponding variable when the read function is performed. But this may cause some serious errors. Hence the return value must be checked for the read operation.

14. Return from Finally Block

The return value must come from finally block whenever it is possible. This helps in distinguishing the data values that are returning from the try-catch statements.

15. Trusted Data sources

Counter check should be made before accessing the input data. For example while reading the data from the file one must check whether the data accessed is NULL or not.

16. Correlated Parameters

Many often there occurs co-relation between the data items. It is a good practice to check these co-relations before performing any operation on those data items.

17. Exceptions Handling

If due to some input condition if the program does not follow the main path and follows an exceptional path. In such a situation, an exceptional path may get followed. In order to make the software more reliable, it necessary to write the code for execution.

4.12.2 Coding Standards

Any good software development approach suggests to adhere to some well-defined standards or rules for coding. These rules are called **coding standards**.

1. Naming Conventions

Following are some commonly used naming conventions in the coding

- Package name and variable names should be in lower case.
- Variable names must not begin with numbers.
- The type name should be noun and it should start with capital letter.
- Constants must be in upper case (For example PI, SIZE)
- Method name must be given in lower case.
- The variables with large scope must have long name. For example count_total, sum, Variables with short scope must have short name. For example i,j.
- The prefix is must be used for Boolean type of variables. For example isEmpty or isFull

2. Files

Reader must get an idea about the purpose of the file by its name. In some programming language like Java -

- The file extension must be java.
- The name of the file and the class defined in the file must have the same name.
- Line length in the file must be limited to 80 characters.

3. Commenting/Layout

Comments are non executable part of the code. But it is very important because it enhances the readability of the code. The purpose of the code is to explain the logic of the program.

- Single line comments must be given by //
- For the names of the variables comments must be given.

- A block of comment must be enclosed within /* and */.

4. Statements

There are some guidelines about the declaration and executable statements.

- Declare some related variables on same line and unrelated variables on another line.
- Class variable should never be declared public.
- Make use of only loop control within the for loop.
- Avoid make use of break and continue statements in the loop.
- Avoid complex conditional expressions. Make use of temporary variables instead.
- Avoid the use of do...while statement.

Advantages of Coding Standards

1. Coding standard brings uniform appearance in system implementation.
2. The code becomes readable and hence can be understood easily.
3. The coding standard helps in adopting good programming practices.

4.12.3 Refactoring

- A change in the code is very common in the software development process. Due to change in the requirements or due to addition of new functionality the changes occur in the coding.
- For accommodating those changes in the code we need to change the design. If we plan to code as per the changed design, then code becomes very complex. Productivity and quality deteriorates.
- Refactoring is the technique used to improve the code and avoid the design decay with time.
- Refactoring is done during the coding. It also plays an important role in test driven development in code improvement step.

Refactoring is defined as a **change made** to the internal structure of software for better understanding and performing cheaper to modifications without changing system's behaviour.

- The basic **objective** of refactoring is to improve the design of the system. Refactoring is basically done in order to **improve the design of code** that already exists.
- As result of refactoring one of the following things may occur -
 - The coupling may get reduced.

- Cohesion may get increased
- The open-closed principle may get followed strictly.
- Refactoring involves the changes in the code. If there is a requirement for code change then the refactoring occurs.
- The main risk of refactoring is that existing working code may **break** due to the changes being made (may lose the objective of the code). This is the main reason why most often refactoring is not done.
- In order to mitigate or avoid this risk following **two rules** must be followed.

Rule 1 : Re-factor in small steps

Rule 2 : For testing the existing functionalities make use of test scripts

- By following these two rules the bugs in the refactoring can be easily identified and corrected.
- In each refactoring only small change is made but the series of refactoring makes a significant transformation in the program structure.
- With refactoring design as well as coding gets improved. Because with refactoring the quality of design improves which helps in making the useful changes in the coding.
- Due to refactoring, the cost of **testing** and **debugging** gets reduced. The efforts required in making changes in the code get reduced and we get an **improved quality code**.
- Thus the **purpose** of refactoring is to create a **long lasting and healthy code**.

Review Questions

1. State the need for refactoring. How can a development model benefit by the use of refactoring ?

AU : May-16, Marks 8

2. What is refactoring ? When is it needed? Explain with an example.

AU : Dec.-16, Marks 6

3. Write a short note on – Refactoring.

AU : May-18, Marks 6

4. Define : Refactoring.

AU : May-19, Marks 2

4.13 Maintenance and Reengineering

- Software maintenance is an activity in which program is modified after it has been put into use.
- In software maintenance usually it is not preferred to apply major software changes to system's architecture.
- Maintenance is a process in which changes are implemented by either modifying the existing system's architecture or by adding new components to the system.

4.13.1 Need for Maintenance

The software maintenance is essential because of following reasons :

1. Usually the system **requirements are changing** and to meet these requirements some changes are incorporated in the system.
2. There is a strong relationship between system and its environment. When a system is installed in an environment, it changes that environment. This ultimately changes the system requirements.
3. The maintained system remains useful in their working environment.
 - Maintenance is applicable to software developed using any software life cycle model. The system changes and hence maintenance must be performed in order to :
 - a) Correct faults.
 - b) Improve the design.
 - c) Implement enhancement.
 - d) Interface with other systems.
 - e) Adoption of environment (different hardware, software, system features etc.).
 - f) Migrate legacy software.
 - g) Replacement of old software by new software.
 - In software maintenance report four key characteristics should be mentioned.
 - i) Maintaining control over the software's day to day functions.
 - ii) Maintaining control over software modification.
 - iii) Repairing of functions.
 - iv) Performance degradation should be avoided.

4.13.2 Types of Software Maintenance

Various types of software maintenance are

1. **Corrective maintenance** - Means the maintenance for correcting the software faults.
 2. **Adaptive maintenance** - Means maintenance for adapting the change in environment (different computers or different operating systems).
 3. **Perfective maintenance** - Means modifying or enhancing the system to meet the new requirements.
 4. **Preventive maintenance** - Means changes made to improve future maintainability.
- According to Lientz and Swanson (in 1980) and Nosek and Palvia (in 1990), if new requirements are added then it needs lot of efforts to maintain such systems.

- Above Fig. 4.13.1 shows that making some modifications in the existing system or adding some new functionalities is very costly from maintenance point of view.
- Nearly 65 % of efforts are required for such maintenance.
- If the operating environment gets changed then 18 % of maintenance cost will be required. But repairing or correcting faults is less costly which may cost around 17 % of total effort cost.

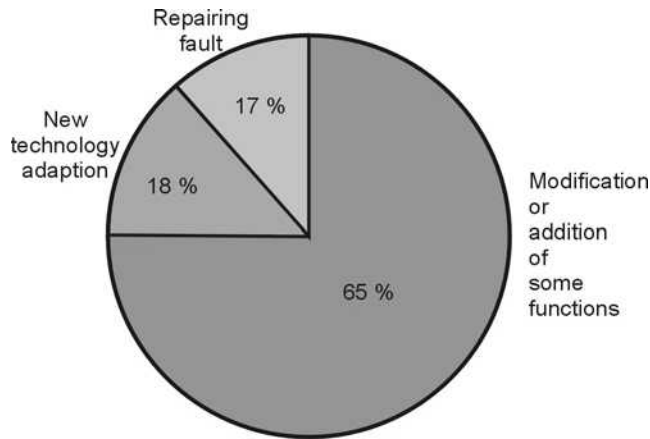


Fig. 4.13.1 Maintenance cost

4.13.3 Software Maintenance Process

The software evolution process is dependant upon the type of software being maintained.

The software maintenance process can be as shown below

1. In the maintenance process initially the **request** for change is made.
2. **Change management** - In this phase the status of all the change requests is identified, described.
3. **Impact analysis** - Following activities are performed in this phase.
 - i) Identify all systems and system products affected by a change request.
 - ii) Make an estimate of the resources needed to effect the change.
 - iii) Analyze the benefits of the change.
4. **System release planning** - In this phase the schedule and contents of software release is planned. The changes can be made to all types of software maintenance.

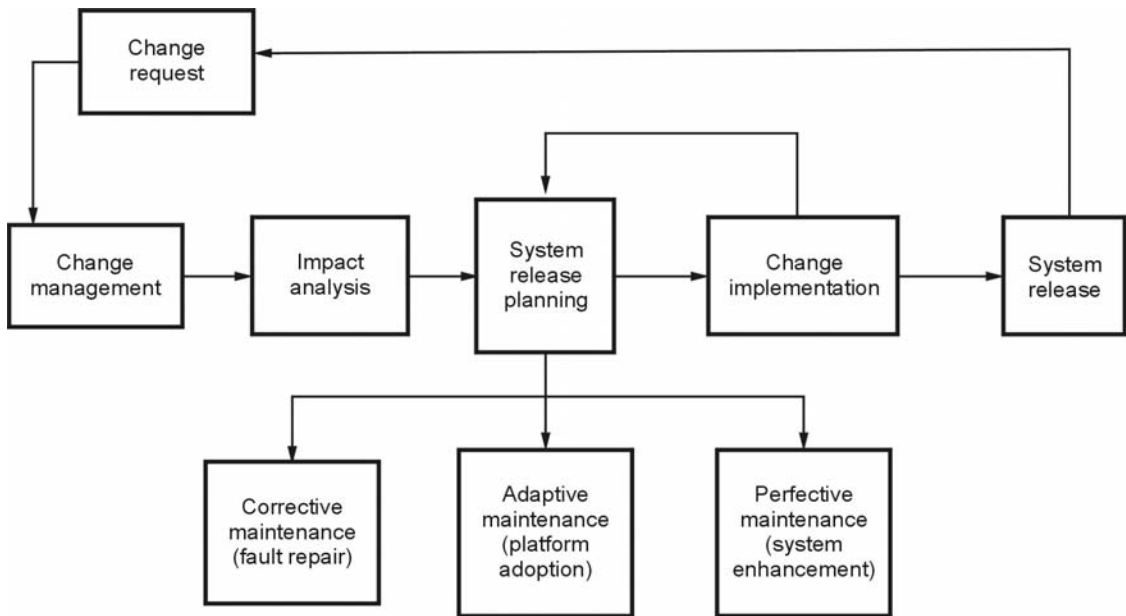


Fig. 4.13.2 Maintenance process

5. **Change implementation** - The implementation of changes can be done by first designing the changes, then coding for these changes and finally testing the changes. Preferably the regression testing must be performed while testing the changes.
6. **System release** - During the software release i) Documentation ii) Software iii) Training iv) Hardware changes v) Data conversion should be described.

Factors affecting maintenance costs

1. **Module Independence** - This is the ability to modify one part of the system.
2. **Programming Language** - For the higher level of the language, the maintenance is cheaper.
3. **Programming Style** - The way in which a program is written makes difference in the cost.
4. **Program Validation and Testing** - The more time and effort spent on design validation and program testing, the fewer errors and the less the need for corrective maintenance.
5. **Quality of Program Documentation** - The better the documentation, the easier it is to maintain.
6. **Configuration Management Techniques** - Keeping track of all the system documents and ensuring they are consistent is a major cost of maintenance.

7. Application Domain - If the application domain is not well understood then the chances of errors are more. This causes high cost on maintenance.
8. Staff Stability - Maintenance costs are reduced if developers have to maintain their own systems.
9. Age of the System - Older systems are difficult to maintain.
10. Dependence of the System on the External Environment - If the system is highly dependant upon external environment then lot much of maintenance is needed.
11. Hardware Stability - If the hardware platform will not change over the life of the system then the maintenance will not be needed.

4.13.4 Issues in Software Maintenance

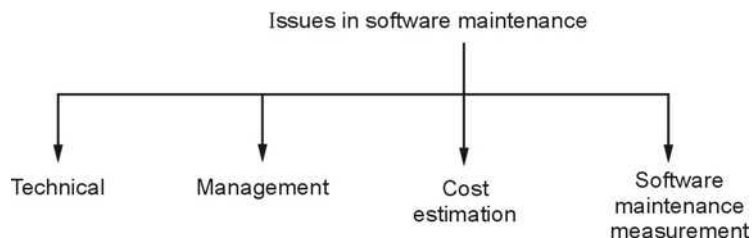


Fig. 4.13.3

1. **Technical** - This is a key issue in software maintenance. Technical maintenance is based on following factors such as limited understanding of system, testing, impact analysis, maintainability.
2. **Management** - Management issue includes organizational issues, staffing problem, process issue, organizational structure, outsourcing.
3. **Cost estimation** - This is one of the major issues in software maintenance. It is based on cost, experience of projects.
4. **Software maintenance measurement** - The software measurement factors such as size effort, schedule, quality, understandability, resource utilization, design complexity, reliability and fault type distribution.

4.14 Business Process Reengineering

AU : Dec.-19

Definition : The Business Process Re-engineering (BPR) is the implementation of radical change in the business process to achieve breakthrough results.

- The business process is a set of tasks that are performed in order to achieve the desired business outcome.
- For example - Purchasing services, designing a new product are the business processes that expect some business outcome.

- Every business process has definite customer who expects some output.
- Many times business process communicate with the participants from different organizations.
- The business system is made up of one or more business processes. The business process reengineering efforts are focused on one or more business processes.

4.14.1 BPR Model

- The BPR model is evolutionary process model. It is iterative in nature.
- There are six activities carried out in this model. Refer Fig. 4.14.1.

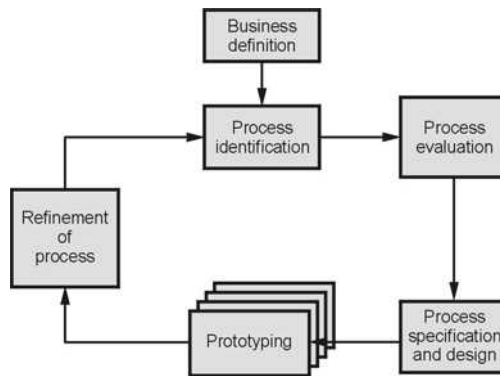


Fig. 4.14.1 BPR Model

1. Business Process Definition :

- The processes are defined based on business goals.
- The business goals are defined using the key factors -
 - i) Cost reduction
 - ii) Quality improvement
 - iii) Time reduction and
 - iv) Personnel development.

2. Process Identification :

- The critical processes are identified.
- They are prioritized according to their need for change.

3. Process Evaluation :

- The existing processes are analyzed.
- Various tasks that constitute a process are identified.
- The time and cost required by these tasks are measured.
- The quality performance issues are identified and isolated.

4. Process Specification and Design :

- The use cases are prepared for each process that need to be redesigned in BPR.
- Each use case captures the scenario that give out some outcome to the customer.
- On analyzing the use cases, new tasks are redesigned if required.

5. Prototyping :

- Before integrating it to the system, the redesigned process is prototyped for **testing** purpose.

6. Refinement and Instantiation :

- The feedback for each prototype in BPR model is collected.
- The processes are refined based on the feedback.

Review Question

1. Explain how Business Process Reengineering (BPE) helps to achieve a defined business outcome

AU : Dec,-19, Marks 8**4.15 Reengineering Process Model****AU : May-19, Marks 11**

Software reengineering is a process of modifying the system for maintenance purpose.

The software reengineering process model depicts six activities as shown in Fig. 4.15.1.

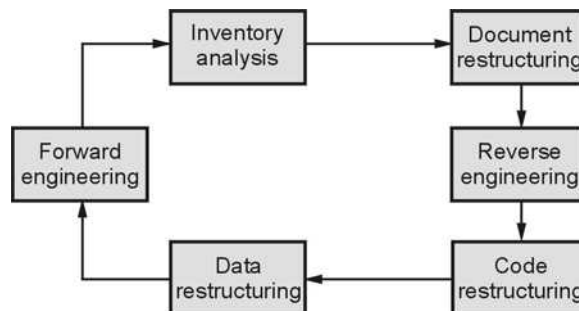


Fig. 4.15.1 Software reengineering process model

1. Inventory Analysis :

- The software organization possess the inventory of all the required applications.
- This inventory should be revisited periodically.
- The resources can be allotted to the candidate applications for reengineering work.
- According to the changes in the candidate applications their status need to be updated.

2. Document Restructuring :

- Documents are essential part of software project lifecycle. But inadequate or inappropriate approach of documentation leads to document restructuring activity.

- There are three alternatives that can be chosen for document restructuring -
 - a. Instead of having time consuming documentation, remain with weak documentation.
 - b. Update poor documentation if needed.
 - c. For the critical systems, rewrite the documents if needed.

3. Reverse Engineering :

- Reverse engineering is the process of **design recovery**.
- In reverse engineering the data, architectural and procedural information is extracted from a source code.
- The reverse engineering creates a representation of the program at a higher level of abstraction than the source code.

4. Code Restructuring :

- This is an activity in which the code is analyzed.
- If the programming constructs are violated then that code is restructured.
- Sometimes the code is rewritten in modern programming language.
- The resultant restructured code is tested and reviewed.

5. Data Restructuring :

- Data restructuring is large scale reengineering activity.
- In this activity, data architecture is dissected and necessary data models are refined.
- If data structures are weak then data are reengineered.
- The changes in data demand for changes in architecture or code.

6. Forward Engineering

- Forward engineering is a process in which the design information is recovered from the existing software and the overall quality of the code is improved.
- Many times new functionalities are added in the existing system to improve overall performance.

Review Question

1. List the phases in software re-engineering process model and explain each phase.

AU : May-19, Marks 11

4.16 Reverse and Forward Engineering

AU : Dec.-19, Marks 5

Reverse engineering is the process of design recovery. In reverse engineering the **data, architectural and procedural information** is extracted **from a source code**.

There are three **important issues** in reverse engineering.

1. **Abstraction level** : This level helps in obtaining the design information from the source code. It is expected that abstraction level should be high in reverse engineering. High abstraction level helps the software engineer to understand the program.

2. **Completeness level** : The completeness means detailing of abstract level. The completeness decreases as abstraction level increases.

For example - From a given source code listing one can easily develop a complete procedural design representation. But it is very difficult to develop complete set of data flow diagrams or entity relationship diagram. The completeness in reverse engineering develops the interactivity. The term interactivity means the degree to which the human is integrated with automated tools to create effective reverse engineering process. As the abstraction level increases the interactivity must increase to bring the completeness.

3. **Directionality level** : Directionality means extracting the information from source code and give it to software engineer. The directionality can be one way or two way. The one way directionality means extracting all the information from source code and give it to software engineer. The two way directionality means the information taken from source code is fed to a re-engineering tool that attempts to restructure or regenerate old programs.

Reverse Engineering Process

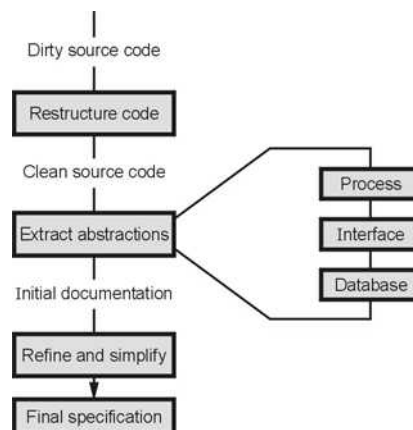


Fig. 4.16.1 Reverse engineering

- Initially the dirty source code or unstructured source code is taken and processed and the code is restructured. After restructuring process the source code becomes clean. Refer Fig. 4.16.1.
- The core to reverse engineering is an activity called extract abstractions.
- In extract abstraction activity, the engineer must evaluate older programs and extract information about procedures, interfaces, data structures or databases used.
- The output of reverse engineering process is a clear, unambiguous final specification obtained from unstructured source code. This final specification helps in easy understanding of source code.

Difference between Software Engineering and Reverse Engineering

Sr. No.	Software engineering	Reverse engineering
1.	Software engineering is a discipline in which theories, methods and tools are applied to develop a professional software product.	Reverse engineering is a process in which the dirty or unstructured code is taken, processed and it is restructured.
2.	Initially only user requirements are available for software engineering process.	A dirty or unstructured code is available initially.
3.	This process starts by understanding user requirements.	This process starts by understanding the existing unstructured code.
4.	The software engineering is conducted using, requirement gathering, analysis, design, implementation and testing.	The reverse engineering is conducted using restructuring the code, cleaning it, by extracting the abstractions. After refinement and simplification of the code final code gets ready.
5.	It is simple and straightforward approach.	It is complex because cleaning the dirty or unstructured code requires more efforts.
6.	Documentation or specification of the product is useful to the end-user.	Documentation or specification of the product is useful to the developer.

Difference between Reverse Engineering and Re-engineering

Reverse Engineering	Re-engineering
Reverse engineering is a process of finding out how a product works from already created software system.	Re-engineering is to observe the software system and build it again for better use.
In reverse engineering, the source code is re-created from the compiled code.	In re-engineering, new piece of code with similar or better functionality than the existing one - is created.

Reverse engineering is carried out for trying to understand inner working of the artifact with availability of any documents.

Re-engineering is carried out for designing something again. Many times from scratch.

Forward Engineering

If the poorly designed and implemented code is to be modified then following alternatives can be adopted -

1. Make lot of modifications to implement the necessary changes.
2. Understand inner workings of the program in order to make the necessary modifications.
3. Redesign, recode and test small modules of software that require modifications.
4. Completely redesign, recode and test the entire program using re-engineering tool.

Definition : Forward engineering is a process that makes use of software engineering principles, concepts and methods to re-create an existing application. This re-developed program extends the capabilities of old programs.

Forward Engineering for Object Oriented Architectures

- Forward engineering is a process of re-engineering conventional software into the object oriented implementation.
- Following are the steps that can be applied for forward engineering the conventional software -
 1. Existing software is reverse engineered in order to create data, functional, and behavioral models.
 2. If existing system extends the functionality of original application then use cases can be created.
 3. The data models created in this process are used to create the base for **classes**.
 4. Class Hierarchies, object-relationship models, object behavioral models, and subsystems are defined.
- During this forward engineering process, algorithms and data structures are reused from existing conventional application.

Difference between Forward and Reverse Engineering

- Forward engineering is a process of constructing a system for specific purpose.
- Reverse engineering is a process of de-constructing a system in order to extend the functionalities or in order to understand the working of the system.

Review Question

1. Outline how the reverse engineering process helps to improve the legacy software.

AU : Dec.-19, Marks 5**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.

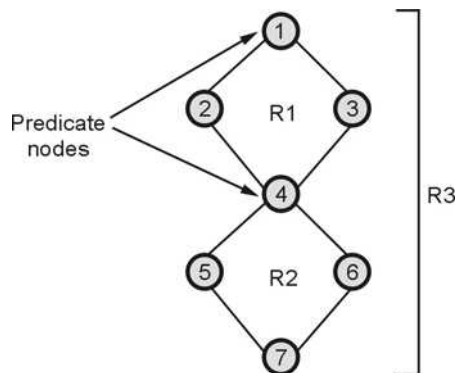
1. The number of regions of the flow graph correspond to the cyclomatic complexity.
2. Cyclomatic complexity, $V(G)$, for a flow graph, G , is defined as :

$$V(G) = E - N + 2,$$

E = Number of flow graph edges

N = Number of flow graph nodes

3. $V(G) = P + 1$

**Fig. 4.1**

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. $V(G) = 8 \text{ edges} - 7 \text{ nodes} + 2 = 3$
3. $V(G) = 2 \text{ predicate nodes} + 1 = 3$

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 roots	For a quadratic equation, two real roots exist if $d = b^2 - 4ac$ is greater than 0.	Compute $d = (b * b) - (4 * a * c)$ check if $(d > 0)$	If $(d > 0)$ then the message "Two real roots" will be printed.	For equation : $x^2 + 2x - 2 = 0$ $r1 = -1 + \sqrt{3}$ $r2 = -1 - \sqrt{3}$ Then solve $(x - r1) * (x - r2)$ and we will get $x^2 + 2x - 2$	Pass

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 Distinguish between verification and validation.**AU : May-08; IT, Dec.-07, 14, 17, May-05, 08, 09, 13, 14, 188**

Ans. :

Verification	Validation
Verification refers to the set of activities that ensure software correctly implements the specific function.	Validation refers to the set of activities that ensure that the software that has been built is traceable to customer requirements .
Are we building the product right ?	Are we building the right product ?
After a valid and complete specification the verification starts.	Validation begins as soon as project starts.
The verification is conducted to ensure whether software meets specification or not.	Validation is conducted to show that the user requirements are getting satisfied.

Q.18 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.19 Which is called as a glass box testing ? What is the objective of this ?
(Refer section 4.3)**

AU : Dec.-07

Q.20 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.21 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 -

Cyclomatic complexity

$$= E - N + 2$$

$$= 7 - 6 + 2 = 3$$

Cyclomatic complexity

$$= P + 1$$

$$= 2 + 1 = 3$$

Cyclomatic complexity

$$= \text{Regions encountered}$$

$$= 3$$

Hence cyclomatic complexity of given program is 3.

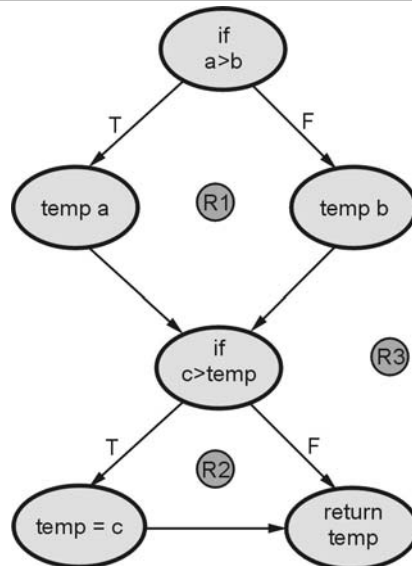


Fig. 4.2

Q.22 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.23 What is meant by smoke testing ? (Refer section 4.8.4)

AU : Dec.-08

Q.24 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.25 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.26 What is meant by integration testing ? (Refer section 4.8)

AU : May-09

Q.27 State any four software testing principles. (Refer section 4.1.2)

AU : Dec.-09

Q.28 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.29 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 = $(90/100) \times 0.9 = 0.81$

Q.30 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.31 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.32 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.33 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.34 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.35 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.36 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.37 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.38 Write the best practices for "CODING?"**AU : May-15, Dec.-15**

Ans. : Some of the commonly used best practices for coding are -

1. The standard control construct must be used instead of using wide variety of controls.
2. Avoid use of goto statements.
3. Information hiding must be supported as far as possible.
4. Avoid deep nesting of code. Nesting means defining one structure inside another.
5. Make use of switch case statements with default case.
6. Make use of exception handling technique to make program execution smooth.

Q.39 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.40 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.41 Differentiate between Verification and Validation

AU : Dec.-14

Ans. : Refer Q.45.

Q.42 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.43 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.44 Write the best practices for "CODING" ?

AU : May-15

Ans. : Some of the commonly used best practices for coding are -

1. The standard control construct must be used instead of using wide variety of controls
2. Avoid use of goto statements
3. Information hiding must be supported as far as possible.
4. Avoid deep nesting of code. Nesting means defining one structure inside another.
5. Make use of switch case statements with default case.
6. Make use of exception handling technique to make program execution smooth.

Q.45 What is the difference between verification and validation ? Which type of testing address verification ? Which type of testing address validation ?

AU : Dec.-16

Ans. :

Sr. No.	Verification	Validation
1	Verification testing comprises of various activities that ensure software correctly implements the specific function	Validation refers to set of activities that ensure that the software that has been built is traceable to customer requirements.
2	The performance testing is testing address verification.	The acceptance testing is testing address validation.

Q.46 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.47 List two testing strategies that address verification. Which types of testing address validation ?**AU : May-17**

Ans. : Refer Q.45.

Q.48 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.49 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.50 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.51 List the levels of testing.**AU : May-19**

Ans. : (1) Unit testing (2) Integration testing (4) Validation testing (5) System testing.

Q.51 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.52 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.



5

Project Management

Syllabus

Software Project Management : Estimation - LOC, FP Based Estimation, Make/Buy Decision COCOMO I & II Model - Project Scheduling - Scheduling, Earned Value Analysis Planning - Project Plan, Planning Process, RFP Risk Management - Identification, Projection - Risk Management - Risk Identification - RMMM Plan - CASE TOOLS

Contents

5.1 Software Project Management

5.2 Estimation	May - 05,06,07,09,16,17,18,19,	
	Dec.-07,10,14,15,16,19	Marks 16
5.3 Make Buy Decision	Dec.-06,May-16	Marks 8
5.4 COCOMO I Model	May-07,08,14,15,17,	
	Dec.-05,13,	Marks 16
5.5 COCOMO II Model	Dec.- 07,10, 14,15,16,19	
	May-07,16,17,18	Marks 16
5.6 Project Scheduling	May-05, 15, 16, 17,	
	Dec.-06,07,15, 16,	Marks 16
5.7 Project Plan		
5.8 Planning Process	May-05,	Marks 16
5.9 Risk Management	May-06,14,15,19,	
	Dec.-10, 15, 19	Marks 8
5.10 CASE Tools		

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 Estimation

AU : May-05,06,07,09,16,17,18,19, 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).

5.2.1 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**.

5.2.2 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

$$S = [S_{\text{opt}} + 4 * S_{\text{m}} + S_{\text{pess}}] / 6$$

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.

5.2.3 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 5.2.1 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

- $\text{Quality} = \text{Number of faults/KLOC}$
- $\text{Cost} = \$/\text{KLOC}$
- $\text{Documentation} = \text{Pages of documentation/KLOC}$
- 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.

5.2.4 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

$$S = [S_{\text{opt}} + (4 * S_m) + S_{\text{pess}}] / 6$$

$$\text{Expected value} = [4700 + (4 * 6000) + 10000] / 6 \rightarrow 6450$$

- A review of historical data indicates -
 - Average productivity is 500 LOC per month
 - Average labor cost is \$6000 per month

Then cost for lines of code can be estimated as

$$\text{cost/LOC} = (6000/500) = \$12$$

By considering total estimated LOC as 32620

- Total estimated project cost = $(32620 * 12) = \$391440$
- Total estimated project effort = $(32620/500) = 65 \text{ Person-months}$

Example 5.2.1 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

$$\text{Cost/LOC} = 8000/620 = ₹ 13 \text{ approx}$$

By considering total estimated LOC as 33360

- Total estimated project cost = $(33360 * 13)$
= ₹ 433680

- Total estimated project effort = $(33360/620)$
= 54 person-months

Example 5.2.2 *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.

5.2.5 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**.
 - Does the system need reliable backup and recovery ?
 - Are data communications required ?
 - Are there distributed processing functions ?
 - Is performance of the system critical ?
 - Will the system run in an existing, heavily utilized operational environment?
 - Does the system require on-line data entry ?
 - Does the on-line data entry require the input transaction to be built over multiple screens or operations ?
 - Are the master files updated on-line ?
 - Are the inputs, outputs, files or inquiries complex ?
 - Is the internal processing complex ?
 - Is the code which is designed being reusable ?
 - Are conversion and installation included in the design ?
 - Is the system designed for multiple installations in different organizations ?
 - 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 x (0.65 + (0.01 x Sum(F_i))



- Once the functional point is calculated then we can compute various measures as follows
 - $\text{Productivity} = \text{FP}/\text{person-month}$
 - $\text{Quality} = \text{Number of faults}/\text{FP}$
 - $\text{Cost} = \$/\text{FP}$
 - $\text{Documentation} = \text{Pages of documentation}/\text{FP}$.

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.

5.2.6 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 INQUIRIES	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

$$\sum (Fi) \rightarrow 52$$

The estimated number of adjusted FP is derived using the following formula :-

$$\text{FP ESTIMATED} = (\text{FP COUNT TOTAL} * [\text{COMPLEXITY ADJUSTMENT FACTOR}]$$

$$\text{FP ESTIMATED} = \text{COUNT TOTAL} * [0.65 + (0.01 * \sum (Fi))]$$

$$\text{Complexity adjustment factor} = [0.65 + (0.01 * 52)] = 1.17$$

- FP ESTIMATED = (381 * 1.17) = 446** (Function point count adjusted with complexity adjustment factor)
- A review of historical data indicates -
 - Average productivity is 6.5 FP/Person month
 - Average labor cost is \$6000 per month
- Calculations for cost per function point, total estimated project cost and total effort
 - The cost per function point = $(6000 / 6.5) = \$923$
 - Total estimated project cost = $(446 * 923) = \$411658$
 - Total estimated effort = $(446 / 6.5) = 69 \text{ Person-month.}$

Example 5.2.3 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				
		X	Simple	Average	Complex	
Number of user inputs	7	X		4		28
Number of user outputs	10	X	4			40
Number of user inquiries	6	X	3			18
Number of files	17	X	7			119
Number of external interfaces	4	X		7		28
Count total						233

$$\begin{aligned}
 \text{Function point} &= \text{Count total} \times [0.65 + 0.01 \times \sum(F_i)] \\
 &= 233 \times [0.65 + 0.01 \times 32] \\
 &= 233 \times [0.65 + 0.32] \\
 &= 233 \times 0.97
 \end{aligned}$$

$$\text{FP} = 226.01$$

Hence adjusted function point is **226.01**.

Example 5.2.4 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)

$$= \Sigma \text{ Functional units} \times \text{Weighting factor.}$$

$$= ((10 \times 3) + (8 \times 7) + (13 \times 7) + (17 \times 10) + (11 \times 4))$$

$$= (30 + 56 + 91 + 170 + 44)$$

$$\text{UFP} = 391$$

Adjusted function point (FP)

$$= \text{UFP} \times \text{Complexity adjustment factor}$$

$$= 391 \times 1.10$$

$$\text{FP} = 430.10$$

Example 5.2.5 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	$2 \times 7 = 14$
ILF	4	10	$4 \times 10 = 40$
EQ	22	4	$22 \times 4 = 88$
EO	16	5	$16 \times 5 = 80$
EI	24	4	$24 \times 4 = 96$
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 5.2.6 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)

$$\begin{aligned}
 &= \sum \text{Functional units} \times \text{Weighting Factor} \\
 &= (10 \times 3) + (12 \times 7) + (20 \times 7) + (15 \times 10) + (12 \times 4) \\
 &= (30 + 84 + 140 + 150 + 48)
 \end{aligned}$$

$$\text{UFP} = 452$$

Adjusted Function Point (FP)

$$\begin{aligned}
 &= \text{UFP} \times \text{Complexity adjustment factor} \\
 &= 452 \times 1.10
 \end{aligned}$$

$$\text{FP} = 497.2$$

Example 5.2.7 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	X	4	8
Number of user output	1	X	5	5
Number of user inquiries	1	X	4	4
Number of files	2	X	10	20
Count total			= 37	

$$\begin{aligned}
 \text{Function point} &= \text{Count total} \times \left[0.65 + \left[0.01 \times \sum_{i=1}^{14} F_i \right] \right] \\
 &= 37 \times (0.65 + (0.01 \times (14 \times 3))) = 37 \times (0.65 + 0.42) \\
 &= 37 \times (1.07)
 \end{aligned}$$

Function point = 39.59

Example 5.2.8 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 : Function 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

$$\begin{aligned}
 \text{Count total} &= \sum (\text{Information domain characteristic} \times \text{Complexity adjustment values})
 \end{aligned}$$

$$= [(30 \times 4) + (52 \times 5) + (22 \times 4) + (12 \times 10) + (2 \times 7)]$$

$$= 120 + 260 + 88 + 120 + 14$$

$$\text{Count total} = 602$$

$$\begin{aligned}\text{FP} &= 602 \times \left[0.65 + \left[0.01 \times \sum_{i=1}^{14} F_i \right] \right] \\ &= 602 \times [0.65 + (0.01 \times (14 \times 3))] \\ &= 602 \times (0.65 + 0.42) \\ &= 602 \times 1.07 \\ &= 644.14\end{aligned}$$

∴ 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**

5.3 Make Buy Decision

AU : Dec.-06, 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.

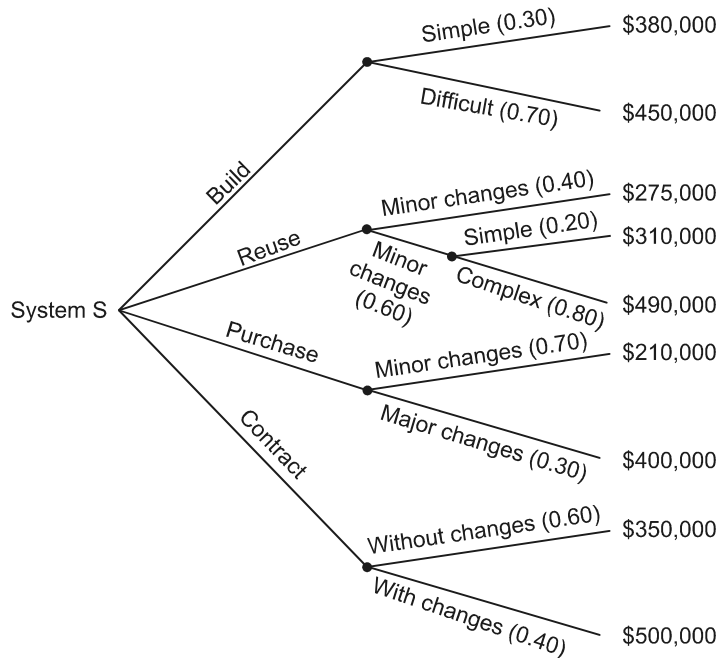


Fig. 5.3.1

Expected cost can be computed for each branch using following formula.

$$\text{Expected cost} = \frac{\sum \text{Path Probability of decision tree path}}{\text{Estimated path cost}} \times \text{Estimated path cost}$$

For example for the branch **system S → Reuse** can be computed as :

$$\begin{aligned}
 \text{Expected cost}_{\text{Reuse}} &= 0.40(\$275 \text{ K}) + [0.60 (0.20 (\$310 \text{ K}) + 0.80 (\$490 \text{ K}))] \\
 &= \$110 \text{ K} + [0.60 (\$ 62 \text{ K} + \$ 392 \text{ K})] \\
 &= \$110 \text{ K} + [0.60 (\$ 454 \text{ K})] \\
 &= \$110 \text{ K} + \$272.4 \text{ K} \\
 &= \$382 \text{ K}
 \end{aligned}$$

Thus the expected cost at each node can be computed. It is summarised 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.

5.3.1 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.

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

A software company loses some control over the software as it is developed by third person.

The trend of outsourcing will be continued in software industry in order to survive in competitive world.

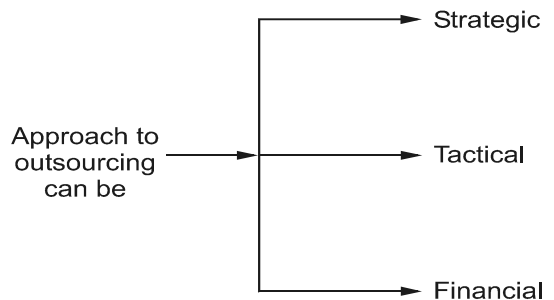


Fig. 5.3.2 Approaches for outsourcing

Review Question

1. Write short note - Make/Buy decision

AU : May-16, Marks 8

5.4 COCOMO I Model

AU : May-07,08,14,15,17, Dec.-05,13, 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 "**C**onstructive **C**ost **M**odel".

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 5.4.1

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 techniques 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

$$E = a_b(\text{KLOC})^{b_b}$$

i.e. $E = 3.0 (30)^{1.12}$ where lines of code = 30000 = 30 KLOC

$$E = 135 \text{ person-month}$$

ii) Duration estimation

$$\begin{aligned} D &= C_b(E)^{d_b} \\ &= 2.5(135)^{0.35} \end{aligned}$$

$$D = 14 \text{ months}$$

iii) Persons estimation

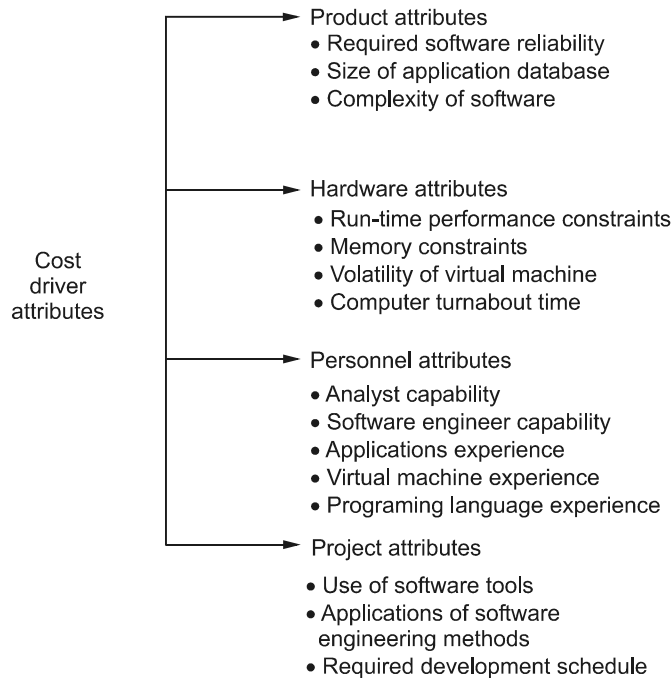
$$P = E/D$$

$$= 135/14$$

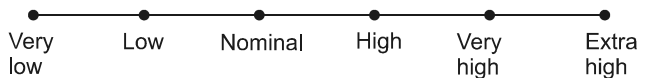
$$P = 10 \text{ persons approximately}$$

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 5.4.2

The formula for effort calculation can be -

$$E = a_i (\text{KLOC})^{b_i} \cdot \text{EAF person-months}$$

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 5.4.3

The duration and person estimate is same as in basic COCOMO model. i.e.

$$D = c_b (E)^{d_b} \text{ months} \quad \text{i.e. use values of } c_b \text{ and } d_b \text{ coefficients that are in Table 5.4.1}$$

$$P = E/D \text{ persons}$$

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)

$$\left. \begin{array}{l} a_i = 2.8 \\ b_i = 1.20 \end{array} \right\} \text{ for embedded software}$$

$$\begin{aligned} \therefore E &= 2.8(30)^{1.20} * 1.15 \\ &= 191 \text{ person-month} \\ D &= c_b(E)^{d_b} = 2.5(191)^{0.32} \\ &= 13 \text{ months approximately} \\ P &= E/D \\ &= 191/13 \\ P &= 15 \text{ persons approximately} \end{aligned}$$

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**

5.5 COCOMO II Model

AU : Dec.-07,10,14,15,16,19, 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 -

$$\text{PM} = (\text{NAP}^{(1-\% \text{reuse}/100)}) / \text{PROD}$$

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

$$PM = 2.94 \times \text{size}^B \times M$$

$$M = RUSE \times PDIF \times PERS \times PREX \times SCED \times FCIL$$

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

$$\text{Effort} = A \times \text{Size}^B \times M$$

- 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 -
 - The estimate about **new** lines of code that is added in the program.
 - Equivalent number of source lines of code (ESLOC) used in **reuse model**.
 - 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 5.5.1 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

$$\text{Object point} = 12 * 2 + 10 * 5 + 80 * 10$$

$$\text{Object point} = 874$$

Example 5.5.2 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.

$$P = E/D$$

where P is total number of persons involved in the project. The constants are –

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

1) Given that, System = Semidetached

Lines of code = 2000 lines = 2 KLOC

$$\therefore E = a_b(KLOC)^{b_b}$$

$$E = 3.0(2)^{1.15}$$

$$E = 6.65 \text{ person-month}$$

$$\therefore D = c_b(E)^{d_b}$$

$$D = 4.8 \text{ months}$$

$$\therefore P = E/D$$

$$P = 1.3 \approx 1 \text{ person}$$

Thus 1 person can handle this project within approximately **5 months**.

2) Given that, System = Embedded

$$\text{Lines of code} = 30,000 \text{ lines} = 30 \text{ KLOC}$$

$$\therefore E = a_b(\text{KLOC})^{b_b}$$

$$= 3.6(30)^{1.20}$$

$$E \approx 213 \text{ person - month}$$

$$D = c_b(E)^{d_b}$$

$$= 2.5(213)^{0.32}$$

$$\therefore D \approx 14 \text{ months}$$

$$P = E/D$$

$$\approx 213/14$$

$$\approx 15 \text{ persons.}$$

That means 15 persons can complete this project within approximately **14 months**.

3) Given that, system = Organic

$$\text{Lines of code} = 1 \text{ lakh} = 100 \text{ KLOC}$$

$$\therefore E = a_b(\text{KLOC})^{b_b}$$

$$= 2.4(100)^{1.05}$$

$$E \approx 302 \text{ person-month}$$

$$D = c_b(E)^{d_b} = 2.5(302)^{0.38}$$

$$\approx 21 \text{ months}$$

$$\therefore P = E/D$$

$$\approx 302/21$$

$$\approx 14 \text{ persons.}$$

That means this project can be completed within 21 months by 14 persons, approximately,

4) Given that, System = Organic

$$\text{Lines of code} = 10 \text{ lakh} = 1000 \text{ KLOC}$$

$$\begin{aligned}
 \therefore E &= a_b (\text{KLOC})^{b_b} = 2.4(1000)^{1.05} \\
 &\approx 3390 \text{ person - month} \\
 D &= c_b (E)^{d_b} \\
 &\approx 2.5 (3390)^{0.38} \\
 &\approx 55 \text{ months} \\
 \therefore P &= E/D \\
 &\approx 3390/55 \\
 &\approx 61 \text{ persons}
 \end{aligned}$$

This project can be completed within 55 months by 61 people approximately.

Example 5.5.3 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.

$$\begin{aligned}
 \therefore \text{UFP} &= 15 \times 4 + 3 \times 5 + 11 \times 7 = 152 \\
 \text{CAF} &= 0.65 + 0.01 \times \left(\sum F_i \right) = 0.65 + 0.01 \times (14 \times 3) = 1.07 \\
 \text{FP} &= \text{UFP} \times \text{CAF} = 152 \times 1.07 = 163 \\
 1\text{FP} &= 20 \text{ LOC} \\
 \therefore 163 &= 3260 \text{ LOC} = 3.26 \text{ KLOC.}
 \end{aligned}$$

The basic COCOMO equation take the form :

$$E = a_b (\text{KLOC})^{b_b}$$

$$D = C_b (\text{KLOC})^{d_b}$$

For organic mode

$$E = 2.4 (3.26)^{1.05}$$

$$E = 8.28 \text{ PM}$$

$$D = 2.5 (3.26)^{0.38}$$

$$D = 3.9 \text{ M}$$

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**

5.6 Project Scheduling

AU : May-05, 15, 16, 17, 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**.

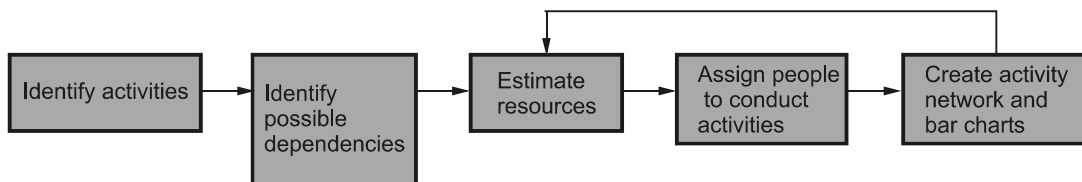


Fig. 5.6.1 Project scheduling process

- 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.

5.6.1 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.

$$E_a = m \left(t_d^4 / t_a^4 \right)$$

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_0 will be the delivery time at which least effort is expended.
- As we move to left of t_0 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

$$E = L^3 / P^3 t^4$$

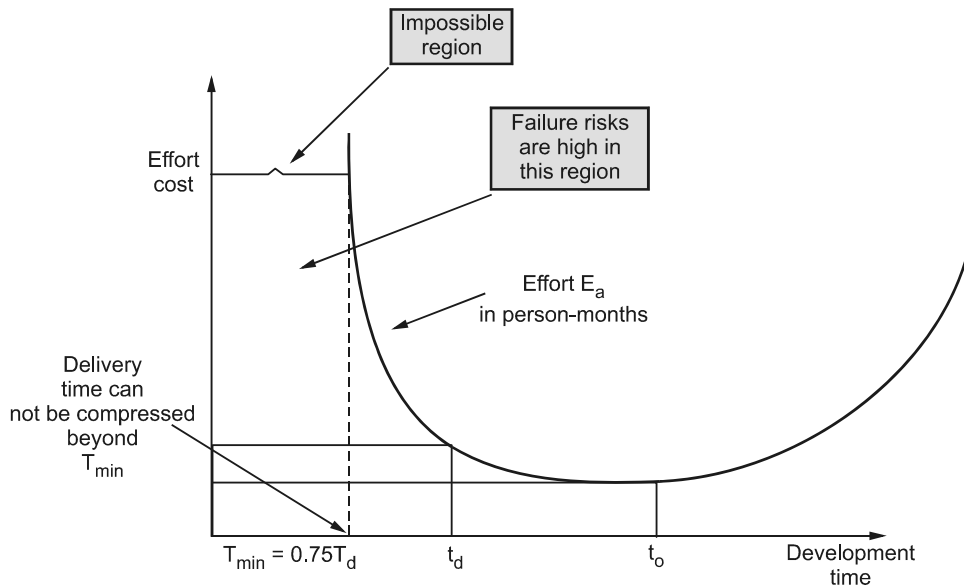


Fig. 5.6.2 Effort and delivery time

That also implies that by extending the end date six months, we can reduce the number of people.

5.6.2 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 longevity
 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 5.6.1 *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

5.6.3 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. 5.6.3.

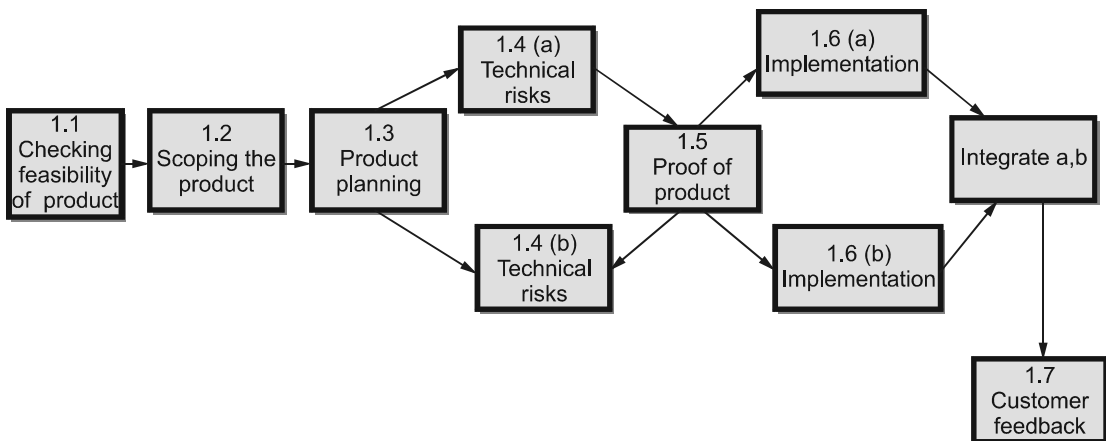


Fig. 5.6.3 Task network

- 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.

5.6.4 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

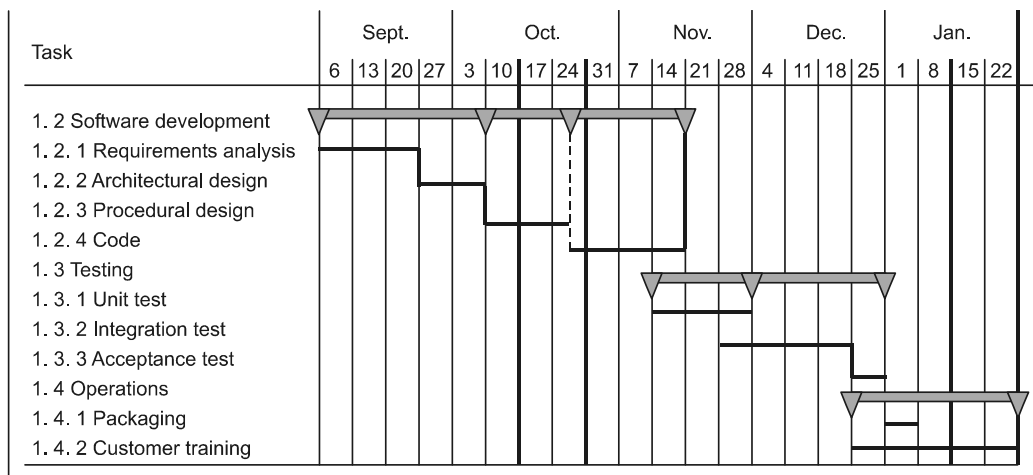


Fig. 5.6.4 Time line chart

Project table

Tasks	Planned start	Actual start	Planned end	Actual end	Effort assignment
Requirement analysis	6 th Sept'05	6 th Sept'05	20 th Sept'05	22 nd Sept'05	Jayashree, Padma, Lucky
Architectural design	27 th Sept'05	27 th Sept'05	3 rd Oct'05	7 th Oct'05	Trupti, Varsha
Procedural design	10 th Oct'05	12 th Oct'05	24 th Oct'05	25 th Oct'05	Varsha, Sachin, Devendra
:	:	:	:	:	:
Customer training	1 st Jan'06	4 th Jan'06	22 nd Jan'06	25 th Jan'06	Smita, Yogita

Table 5.6.1 Project table

5.6.5 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 practitioners 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 additional 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.

5.6.6 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) :

$PV = \text{Planned Completion (\%)} * \text{Budget At Completion (BAC)}$

$EV = \text{Actual Completion (\%)} * BAC$

$SV = EV - PV$

$CV = EV - AC$

$SPI = EV/PV$

$CPI = EV/AC$

Example 5.6.2 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 -

$$\begin{aligned}\text{Earned value} &= \% \text{ of work} \times \text{Budget} \\ &= 27 \% \times 50000 \\ &= \frac{27}{100} \times 50000 = 13,500\end{aligned}$$

∴ The earned value is \$ 13,500.

Example 5.6.3 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

$$\begin{aligned} 1) \quad \text{Planned Value (PV)} &= \text{Planned Completion (\%)} * \text{BAC} \\ &= 15 \% * 900000 \\ &= \frac{15}{100} * 900000 \end{aligned}$$

$$\therefore \quad \text{PV} = \text{₹ 135000}$$

$$\begin{aligned} 2) \quad \text{Earned Value (EV)} &= \text{Actual Completion (\%)} * \text{BAC} \\ &= 10 \% * 900000 \\ &= \frac{10}{100} * 900000 \end{aligned}$$

$$\therefore \quad \text{EV} = \text{₹ 90000}$$

$$\begin{aligned} 3) \quad \text{Cost Performance Index (CPI)} &= \frac{\text{EV}}{\text{AC}} \\ &= \frac{900000}{100000} \end{aligned}$$

$$\text{CPI} = 0.90$$

As $\text{CPI} < 1$ means project is **over budget**.

$$\begin{aligned} 4) \quad \text{Schedule Performance Index (SPI)} &= \frac{EV}{PV} \\ &= \frac{900000}{135000} \\ \text{SPI} &= 0.67 \end{aligned}$$

As $SPI < 1$ means project is **behind schedule**.

This indicates project is in real trouble.

Example 5.6.4 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)

$$\begin{aligned} \text{i) Planned Value (PV)} &= \text{Planned Completion (\%)} * \text{BAC} \\ &= 25\% * 80,000 \\ &= \frac{25}{100} * 80,000 \end{aligned}$$

$$\text{PV} = ₹ 20,000$$

$$\begin{aligned} \text{ii) Earned Value (EV)} &= \text{Actual Completion (\%)} * \text{BAC} \\ &= 30\% * 80,000 \\ &= \frac{30}{100} * 80,000 \end{aligned}$$

$$\text{EV} = ₹ 24,000$$

Now we will compute the Cost Performance Index (CPI) and Schedule Performance Index (SPI).

$$\begin{aligned}
 \text{iii) Cost Performance Index (CPI)} &= \frac{EV}{AC} \\
 &= \frac{24,000}{40,000}
 \end{aligned}$$

$$\text{CPI} = 0.6$$

$$\begin{aligned}
 \text{iv) Schedule Performance Index (SPI)} &= \frac{EV}{PV} \\
 &= \frac{24,000}{20,000}
 \end{aligned}$$

$$\text{SPI} = 1.2$$

The $\text{CPI} < 1$, then that means the project is over budget. For every ₹ 1 spent we are getting 60 paisa worth performance.

The $\text{SPI} > 1$, then that means the project is ahead of the schedule. That means project can be delivered before the given schedule.

Example 5.6.5 Given the following project plan of tables Table 5.6.2 and Table 5.6.3 :

Table 5.6.2

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 5.6.3

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.

$$\therefore PV = 32000$$

- 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	Total
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.

- $$SPI = \frac{EV}{PV}$$

$$= \frac{19000}{32000}$$

$$\approx 0.6$$

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.

AU : May-15, Marks (8+8)

3. Discuss Putnam resources allocation model. Derive the time and effort equations.

AU : May-16, Marks 12

5.7 Project Plan

A project plan must be prepared in advance from the available information. The project planning is an **iterative process** and it gets completed only on completion of the project. This process is iterative because new information gets available at each phase of project development. Hence the plan needs to be modified on regular basis for accommodating new requirements of the project.

Following is a structure of project plan -

Project plan	
1.1 Introduction	The goals, objectives and constraints of the project must be described in this section.
1.2 Project organization	The organization of development team should be described. The number of people involved along with their roles must be described in detail.
1.3 Risk analysis	All possible risks should be identified. Not only this but, the possible risk reduction strategies must be decided.
1.4 Hardware and software requirements	This section specifies the required hardware and software.
1.5 Work breakdown	Various project activities are grouped together to define the project work breakdown. Deciding the project milestone and deliverables is an important task in defining the work breakdown.
1.6 Project schedule	The tentative schedule of project activities must be determined.
1.7 Report generation	The structure of the project report, when it should be generated must be decided.

5.8 Planning Process

AU : May-05, Marks 16

The first step to be taken in project management is **Project planning**. There are five major activities that are performed in project planning -

1. Project estimation
2. Project scheduling
3. Risk analysis
4. Quality management planning
5. Change Management planning

Software estimation begins with a description of the **scope of software product**.

For the meaningful project development the scope must be **bounded**. The problem for which the product is to be built is then decomposed into a set of smaller problems. Each of these is estimated using historical data (metrics) and / or previous experience as a guide. The two important issues-**problem complexity** and **risk** are considered before final estimate is made.

There are many useful techniques for time and effort estimation. **Process and project** metrics can provide historical perspective and powerful input for generation of quantitative estimates.

Estimation of recourses, cost and schedule for a software engineering effort requires -

- Experience,
- Access to good historical information (metrics) and
- The courage to commit to quantitative predictions when quantitative information is available.

While estimating the project, both the project planner and the customer should recognize that **variability** in software requirement means **instability** in cost and schedule. When customer changes the requirements, then estimation needs to be revisited.

5.9 Risk Management

AU : May-06,14,15,19, Dec.-10, 15,19, Marks 8

Definition of risk : The risk denotes the uncertainty that may occur in the choices due to past actions and risk is something which causes heavy losses.

Definition of risk management : Risk management refers to the process of making decisions based on an evaluation of the factors that threats to the business.

Various activities that are carried out for risk management are -

1. Risk identification
2. Risk projection
3. Risk refinement
4. Risk mitigation, monitoring and management.

Process of Risk Management

- The risk management process continues until the project gets completed successfully.

- The risk planning phase is required in order to minimize or avoid the risks.
- These risks get monitored and mitigated in the risk monitoring phase.
- After risk mitigation certain amount of risks may remain in the project, the risk planning is required again, in order to minimize or avoid those risks.
- Thus frequent reviews and corrective actions must be taken in order to make the project with minimized risk. Hence the **risk management is an iterative process**.

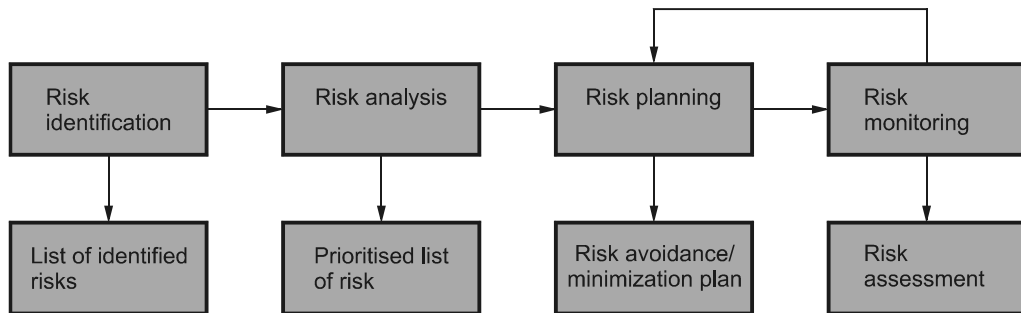


Fig. 5.9.1 Risk management process

From Fig. 5.9.1 risk management is performed in following stages.

1. **Risk identification** : In this phase all possible risks are anticipated and a list of potential risks is prepared.
2. **Risk analysis** : The after-effects of the risks are obtained and a list is prepared in which, the risks that need to be handled are prioritised.
3. **Risk planning** : The risk avoidance or risk minimization plan is prepared in this phase.
4. **Risk monitoring** : Identified risks must be mitigated. Hence risk mitigation plan must be prepared once the risks are discovered.

5.9.1 Software Risks

There are two characteristics of the risks

1. The risk may or may not happen. It shows the **uncertainty** of the risks.
2. When risks occur, unwanted consequences or **losses** will occur.

Different types of risk

1. Project risk

Project risks arise in the software development process then they basically affect budget, schedule, staffing, resources, and requirements. When project risks become severe then the total cost of project gets increased.

2. Technical risk

These risks affect quality and timeliness of the project. If technical risks become reality then potential design implementation, interface, verification and maintenance problems gets created. Technical risks occur when problem becomes harder to solve.

3. Business risk

When feasibility of software product is in suspect then business risks occur. Business risks can be further categorized as

- i) Market risk - When a quality software product is built but if there is no customer for this product then it is called market risk (i.e. no market for the *product*).
- ii) Strategic risk - When a product is built and if it is not following the company's business policies then such a product brings strategic risks.
- iii) Sales risk - When a product is built but how to sell is not clear then such a situation brings sales risk.
- iv) Management risk - When senior management or the responsible staff leaves the organization then management risk occurs.
- v) Budget risk - Losing the overall budget of the project is called budget risk.

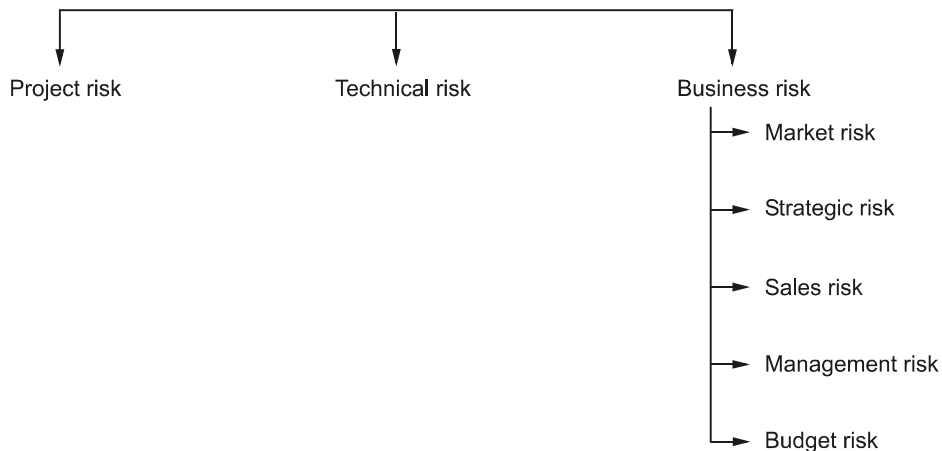


Fig. 5.9.2 Categorization of risk

Another categorization of risk proposed by **Charette** is -

Known risks are those risk that are identified after evaluating the project plan. These risks can also be identified from other sources such as environment in which the product gets developed, unrealistic dead lines, poor requirement specification and software scope. There are two types of known risks - *predictable* and *unpredictable* risks.

Predictable risks are those risks that can be identified in advance based on past project experience. For example : Experienced and skilled staff leaving in between or improper communication with customer resulting in poor requirement specification.

Unpredictable risks are those risks that can not be guessed earlier.

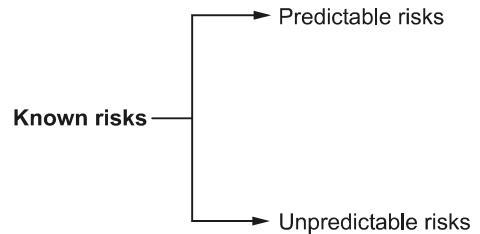


Fig. 5.9.2(a)

For example certain changes in Government policies may affect the business project.

5.9.2 Risk Identification

Risk identification can be defined as the efforts taken to specify threats to the project plan. Risks identification can be done by identifying the known and predictable risks.

The risk identification is based on two approaches

1. Generic risk identification - It includes potential threat identification to software project.
2. Product-specific risk identification - It includes product specific threat identification by understanding people, technology and working environment in which the product gets built.

Normally the risk identification is done by the project manager who follows following steps -

Step 1 : Preparation of risk item check list

The risk items can be identified using following known and predictable components

- i) Product size - The risk items based on overall size of the software product is identified.
- ii) Business impact - Risk items related to the marketplace or management can be predicted.
- iii) Customer characteristics - Risks associated with customer-developer communication can be identified.
- iv) Process definition - Risks that get raised with the definition of software process. This category exposes important risks items because whichever is the process definition made, is then followed by the whole team.
- v) Development environment - The risks associated with the technology and tool being used for developing the product.

- vi) Staff size and experience - Once the technology and tool related risks items are identified it is essential to identify the risk associated with sufficient highly experienced and skilled staff who will do the development.
- vii) Technology to be built - complexity of the system should be understood and related risk items needs to be identified.

After preparing a risk item checklist a questionnaire is prepared. These set of questions should be answered and based on these answers the impact or seriousness of particular risk item can be judged.

Step 2 : Creating risk components and drivers list.

The set of risk components and drivers list is prepared along with their probability of occurrence. Then their impact on the project can be analysed.

Let us understand which are the risk components and drivers.

5.9.2.1 Risk Components and Drivers

U.S. Air force has written a guideline for risk identification which is based on identification of risk component and risks drivers. It has suggested following types of risk components -

1. *Performance risk* - It is the degree of uncertainty that the product will satisfy the requirements
2. *Cost risk* - It is the degree of uncertainty that the project will maintain the budget.
3. *Support risk* - It is the degree of uncertainty that the software project being developed will be easy to correct, modify or adapt.
4. *Schedule risk* - It is the degree of uncertainty that the software project will maintain the schedule and the project will be delivered in time.

Associated with these components are the risk drivers that are used to analyse the impact of risk. These four risk drivers are listed below

For the risk impact assessment a table is built in which impact of each risk driver on each software component can be specified.



Fig. 5.9.4

5.9.2.2 How to Assess Overall Project Risk ?

The best approach is to prepare a set of questions that can be answered by project managers in order to assess the overall project risks. These questions can be

1. Will the project get proper support by the customer manager ?
2. Are the end-users committed to the software that has been produced ?
3. Is there a clear understanding of requirements ?
4. Is there an active involvement of the customer in requirement definition ?
5. Is that the expectations set for the product are realistic ?
6. Is project scope stable ?
7. Are there team members with required skills ?
8. Are project requirements stable ?
9. Does the technology used for the software is known to the developers ?
10. Is the size of team sufficient to develop the required product ?
11. Is that all the customers know the importance of the product/ requirements of the system to be built ?

Thus the number of negative answers to these questions represents the severity of the impact of the risk on overall project.

5.9.3 Risk Projection

The risk projection is also called risk estimation.

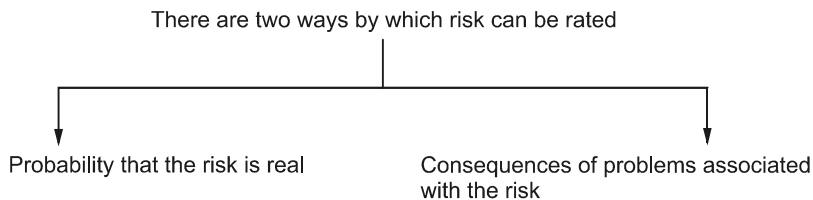


Fig. 5.9.5

The project planner, technical staff, project manager performs following steps to perform following steps for risk projection -

- Establish a scale that indicates the probability of risk being real.
- Enlist the consequences of the risk.
- Estimate the impact of the risk on the project and product.
- Maintain the overall accuracy of the risk projection in order to have clear understanding of the software that is to be built.

These steps help to prioritize the risks. Once the risks are prioritized then it becomes easy to allocate the resources for handling them.

5.9.3.1 Building Risk Table

1. Building the risk table is the simplest and most commonly used technique adopted by project managers in order to project the risks. The sample risk table is as given below -

Risk table				
Risk	Category	Probability	Impact	RMMM
Is the skilled staff available	Staff	50 %	Catastrophic	
Is that the team size sufficient	Staff	62 %	Critical	
Have the staff received sufficient training	Staff	25 %	Marginal	
Will technology meet the expectations	Technology	30 %	Critical	
Is the software management tool available	Environment	40 %	Negligible	
How much amount of reused software is required?	Project size	60 %	Marginal	
Will customer change the requirement?	Customer	20 %	Critical	

While building the risk table

- The project team first of all enlists all probable risks with the help of risk item checklist.
- Each risk is then categorized. As we know various categories of risk can be a) Project size b) Technology c) Customer d) Staff e) Business f) Developing environment.
- Probability of occurrence of each risk is then estimated by each team member individually.
- Then impact of each risk is assessed. While calculating the impact of each risk, each using the cost drivers each component of risk (*performance, cost, support, and schedule*) is assessed and it then averaged to quote the overall impact of particular risk.

2. After building this table it is then sorted by probability and impact. The high probability and high impact risks will be at the top of the table. And low probability and low impact risk will be at the bottom of the table. This arrangement of the table is called **first-order prioritization**.
3. Then the project manager goes through this first-order prioritized risk table and draws a horizontal line at some point in the table. This line is called **cut off line**. The risks table above the cut off line is now considered for further risk analysis.
4. The risk table below the cut off line is again sorted and a **second-order prioritization** is applied on this table.
5. The risk table above the cut-off line is having the risks with high probability and high impact and such risks should occupy the significant amount of management time.
6. All the risks that lie above the cut off line should be managed. Using Risk mitigation, monitoring and management plan the last column of the risk table is filled up.

5.9.3.2 Assessing Risk Impact

While assessing the risks impact three factors are considered

- Nature of risk
- Scope of the risk
- Timing at which risk occurs.

Nature of risk denotes the type or kind of risk. For example if software requirement is poorly understood, the software processes gets poorly designed and ultimately it will create a problem in unit testing. **Scope** of the risk means severity of the risk. And **timing** of risk means determining at which phase of software development life cycle the risk will occur and how long it will persist.

U.S. Air Force has suggested following steps in order to determine the impact of risk -

1. The probability of all the components of risk (*performance, cost, support and schedule*) is calculated and averaged.
2. Using risk drivers (*catastrophic, critical, marginal, negligible*) the impact of risk on each components is determined.
3. Build the risk table and analyse the high impact, high probability risks.

Risk exposure

The risk exposure can be calculated by following formula

$$\text{Risk Exposure} = \text{Probability of occurrence of risk} \times \text{Cost}$$

For example : Consider a software project with 77 percent of risk probability in which 15 components were developed from the scratch. Each component have on an average 500 LOC and each LOC have an average cost of \$10. Then the risk exposure can be calculated as ,

$$\begin{aligned}\text{First of all we will compute cost} &= \text{Number of components} \times \text{LOC} \times \text{cost of each LOC} \\ &= 15 \times 500 \times 10 = \$75000\end{aligned}$$

$$\begin{aligned}\text{Then Risk Exposure} &= \text{Probability of occurrence of risk} \times \text{Cost} \\ &= 77 / 100 \times 75000 \\ &= \$57750\end{aligned}$$

Thus risk exposure for each risk from risk table is calculated. The total risk exposure of all risks helps in determining the final cost of the project.

5.9.4 RMMM

RMMM stands for risk mitigation, monitoring and management. There are three issues in strategy for handling the risk is

1. Risk avoidance
2. Risk monitoring
3. Risk management.

Risk mitigation

Risk mitigation means preventing the risks to occur(risk avoidance). Following are the steps to be taken for mitigating the risks.

1. Communicate with the concerned staff to find of probable risk.
2. Find out and eliminate all those causes that can create risk before the project starts.
3. Develop a policy in an organization which will help to continue the project even though some staff leaves the organization.
4. Everybody in the project team should be acquainted with the current development activity.
5. Maintain the corresponding documents in timely manner. This documentation should be strictly as per the standards set by the organization.
6. Conduct timely reviews in order to speed up the work.
7. For conducting every critical activity during software development, provide the additional staff if required.

Risk monitoring

In risk monitoring process following things must be monitored by the project manager,

1. The approach or the behaviour of the team members as pressure of project varies.
2. The degree in which the team performs with the spirit of “team-work”.
3. The type of co-operation among the team members.
4. The types of problems that are occurring.
5. Availability of jobs within and outside the organization.

The project manager should monitor certain mitigation steps. For example.

If the current development activity is monitored continuously then everybody in the team will get acquainted with current development activity.

The **objective** of risk monitoring is

1. To check whether the predicted risks really occur or not.
2. To ensure the steps defined to avoid the risk are applied properly or not.
3. To gather the information which can be useful for analyzing the risk.

Risk management

Project manager performs this task when risk becomes a reality. If project manager is successful in applying the project mitigation effectively then it becomes very much easy to manage the risks.

For example, consider a scenario that many people are leaving the organization then if sufficient additional staff is available, if current development activity is known to everybody in the team, if latest and systematic documentation is available then any ‘new comer’ can easily understand current development activity. This will ultimately help in continuing the work without any interval.

5.9.5 RMMM Plan

The RMMM plan is a document in which all the risk analysis activities are described. Sometimes project manager includes this document as a part of overall project plan. Sometimes specific RMMM plan is not created, however each risk can be described individually using risk information sheet. Typical template for RMMM plan or Risk information sheet can be,

Risk information sheet			
Project name <enter name of the project for which risks can be identified>			
Risk id <#>	Date <date at which risk is identified >	Probability <risk probability>	Impact <low/medium/high>
Origin <the person who has identified the risk>		Assigned to <who is responsible for mitigating the risk>	
Description <Description of risk identified>			
Refinement/Context <associated information for risk refinement>			
Mitigation/Monitoring <enter the mitigation/monitoring steps taken>			
Trigger/Contingency plan <if risk mitigation fails then the plan for handling the risk>			
Status <Running status that provides a history of what is being done for the risk and changes in the risk. Include the date the status entry was made>			
Approval <name and signature of person approving closure>.		Closing date <date>	

The risk information sheet can be maintained by database systems. After documenting the risks using either RMMM plan or Risk information sheet the risk mitigation, monitoring and analysis activities are stopped.

Review Questions

1. State the need for risk management and explain the activities under risk management.

AU : May-15, Marks 16

2. Explain in detail about the risk management in a software development life cycle.

AU : Dec.-15, Marks 16

3. What are the categories of software risks ? Give an overview about risk management.

AU : May-14, Marks 6

4. Define : Risk.

AU : May-19, Marks 2

5. List the types of risk and give examples for each.

AU : May-19, Marks 5

6. List and explain the phases in risk management.

AU : May-19, Marks 6

7. Present a detailed note on risk management.

AU : Dec.-19, Marks 13

5.10 CASE Tools

What is CASE ?

The Computer Aided Software Engineering (CASE) tools automate the project management activities, manage all the work products.

Importance of CASE Tools

- The CASE tools assist to **perform various software life cycle activities** such as analysis, design, coding and testing.
- Software engineers and **project managers** make use of CASE tools.
- The use of CASE tools in the software development process **reduces** the significant amount of **efforts**.
- CASE is used in conjunction with the process model that is chosen.
- CASE tools help software engineer to produce the **high quality software efficiently**.
- Use of CASE tool **automates** particular **task of software development activity**. But it is always useful to use a set of CASE tools instead of using a single CASE tool. If different CASE tools are not integrated then the data generated by one tool will be an input to other tools. This may also require a format conversions, as tools developed by different vendors may use different formatting. There are chances, that many tools do not allow exporting data and maintain data in proprietary formats.

5.10.1 Building Blocks of CASE

- The CASE tools may exist as a single tools or a collection of multiple tools. These tools must communicate with various elements such as hardware, database, people, network, operating system and so on. This communication creates an **integrated environment**.

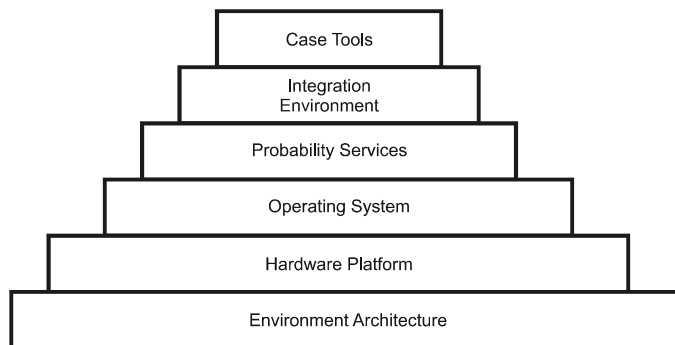


Fig. 5.10.1 Building block for CASE Tools

Fig. 5.10.1 represents the building block for CASE. The bottom most layer consists of **environment architecture** and **hardware platform**. The environment architecture consists of collection of system software and human work pattern that is applied during the software engineering process.

- A set of **probability services** connects the CASE tools with the integration framework.
- The **integration framework** is a collection of specialized programs which allows the CASE tools to communicate with the database and to create same look and feel for the end-user. Using probability services CASE tools can communicate with the cross platform elements.
- At the top of this building block a collection of CASE tools exist. CASE tools basically assist the software engineer in developing a complex component.
- CASE tools can exist in variety of manner. A single CASE tool can be used, or a collection of CASE tools may exists which acts as some package. The CASE tools may serve as a bridge between other tools.

5.10.1.1 Taxonomy

- To create an effective CASE environment, various categories of tools can be developed.
- CASE tools can be classified by
 1. by function or use
 2. by user type (e.g. manager, tester), or
 3. by stage in software engineering process (e.g. requirements, test)

The taxonomy of CASE tools is as given below.

1) Business process engineering tools

This tool is used to model the business information flow. It represents business data objects, their relationships and how data objects flow between different business areas within a company.

2) Process modeling and management tools

It models software processes. First the processes need to be understood then only it could be modelled. This tool represents the key elements of the processes. Hence it is possible to carry out work tasks efficiently.

3) Project planning tools

These tools help to plan and schedule projects. Examples are PERT and CPM. The objective of this tool is finding parallelism and eliminating bottlenecks in the projects.

4) Risk analysis tools

It helps in identifying potential risks. These tools are useful for building the risk table and thereby providing detailed guidance in identification and analysis of risks. Using this tool one can categorize the risks as catastrophic, critical, marginal, or negligible. A cost is associated with each risk which can be calculated at each stage of development.

5) Project management tools

These track the progress of the project. These tools are extension to the project planning tools and the use of these tools is to update plans and schedules.

6) Requirements tracing tools

The objective of these tools is to provide a systematic approach to isolate customer requirements and then to trace these requirements in each stage of development.

7) Metrics and management tools

These tools assist to capture specific metrics that provide an overall measure of quality. These tools are intended to focus on process and product characteristics. For example "defects per function point", "LOC/person-month".

8) Documentation tools

Most of the software development organizations spend lot of time in developing the documents. For this reason the documentation tools provide a way to develop documents efficiently. For example - word processors that give templates for the organization process documents.

9) System software tools

These tools provide services to network system software, object management and distributed component support. For example - e-mail, bulletin boards, and www access.

10) Quality assurance tools

These are actually metrics tools that audit source code to insure compliance with language standards.

11) Database management tool

It provides consistent interfaces for the project for all data, in particular the configuration objects are primary repository elements.

12) Software configuration management tools

It assists with identification, version control, change control, auditing, and status accounting.

13) Analysis and design tools

It creates models of the system. Some create formal models. Others construct data flow models. These models contain representation of data, function and behavior. Such tools helps in architectural, component level and interface design.

14) PRO/SIM tools

These are prototyping and simulation tools. They can help predict real time system response and allow mockups of such systems to be fashioned.

15) Interface design and development tools

These tools are used in developing user interface. It includes various components such as menu, icons, buttons, scrolling mechanisms etc. For example - JAVA, Visual Studio.

16) Prototyping tools

These tools support to define screen layout rapidly for interactive applications.

17) Programming tools

The programming tool category include the programs that support most of the conventional programming languages. For example - compilers, debuggers, editors, database query languages.

18) Web development tools

These tools help in developing the web based applications. The various components of these tools are text, graphics, form, scripts, and applets.

19) Integration and testing tools

These tools include various category of tools such as data acquisition tools, static measurement, dynamic measurement, simulation, cross functional tools.

20) Static analysis tools

The static testing tools are used for deriving the test cases. There are three types of static testing tools.

- i. Code based testing tools – These tools take source code as input and generate test cases.
- ii. Specialized testing language – Using this language the detailed test specification can be written for each test case.
- iii. Requirement-based testing tools – These tools help in designing the test cases as per user requirements.

21) Dynamic analysis tools

These interact with an executing program, checking path coverage, and testing assertions. Intrusive tools insert code in the tested program. Non intrusive tools use a separate hardware processor.

22) Test management tools

These tools manage and co-ordinate regression testing, perform comparisons of output, and act as test drivers.

23) Client/server testing tools

The client server tools are used in client server environment to exercise the GUI and network communication requirements for client and server.

24) Reengineering tools

These tools performs a set of maintenance activities. These tools perform various functions such as

- Reverse engineering to specification tools.
- Code restructuring.
- On-line system re-engineering.

5.10.2 Workbenches

- CASE workbench is a set of tools which supports a particular phase in the software process.
- These tools work together to provide the complete support to the software development activity.
- In the workbench common services are provided which are used by all the tools. Some kind of data integration is also supported.
- For example -**Analysis and Design Workbench** (Refer Fig. 5.10.2)
 - It support the generation of system models during design and analysis activities.
 - It provides graphical editors plus shared repository.
 - It may include code generators to create source code from design information.

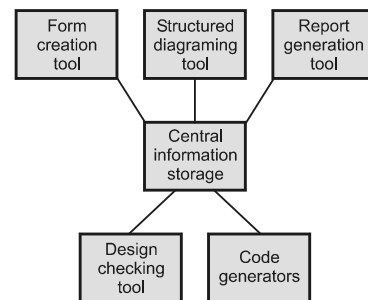


Fig. 5.10.2 Analysis and design workbench

Advantages

1. It is available at low cost.
2. There is productivity improvement due to support of workbenches.
3. It results in standardized documentation for software system.

Drawbacks

1. These systems are closed environments with tight integration with tools.
2. The import and export facilities for various types of data is limited.
3. It is difficult to adapt method for specific organizational need.

5.10.3 Environments

- CASE tools create a pool of software engineering information. The integrated CASE environment allows a transfer of information into and out of this pool. For such transfer there is a need for some architectural components. These components are –
 - Database for storing the information
 - Object management system. Because using the objects information can be transferred in the information pool.
 - Control mechanism.
 - User interface
- The Fig. 5.10.3 shows the simple model for integrated CASE environment. This environment consists of various levels.

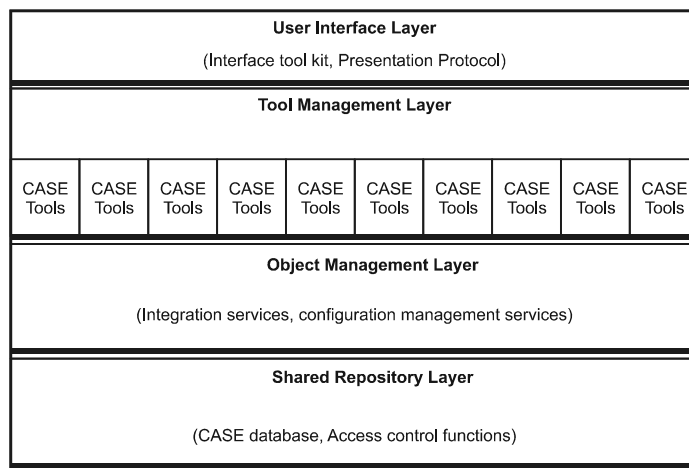


Fig. 5.10.3 Model for integrated CASE environment

- The **user interface layer** consists of **interface tool kit** and **presentation protocols**. The **interface tool kit** consists of collection of software required for interface management and display objects. The **presentation protocol** decides a common look and feel of the presentation interface.

Then comes a **tools layer**. It consists of set of Tools Management Services (TMS).

- The next layer is **Object Management Layer (OML)**. It performs the configuration management. The services of this layer allows the identification of all the configuration objects. So that the case tool can be plugged into the integrated CASE environment.
- The bottom most layers is shared **repository layer**. It consists of **CASE database** and **access control functions**. These access control functions help the object management layer to access the CASE Database.

5.10.4 Components of CASE

Various components of CASE tools are –

1. Central Repository :

- The central repository contains the common, integrated and consistent information.
- The central repository acts like a data dictionary.
- It contains product specification, requirement documents, project management information and reports.

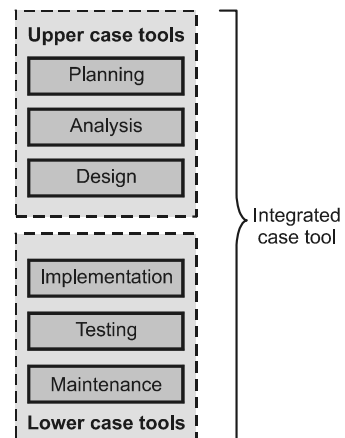


Fig. 5.10.4 Components of CASE tools

2. Upper Case Tools :

- Uppercase tool focus on the planning, analysis phase and sometimes the design phase of the software development lifecycle.

3. Lower Case Tools :

- Lower CASE Tool Computer-Aided Software Engineering (CASE) software tool that directly supports the implementation (programming) and integration tasks.
- Lower CASE tools support database schema generation, program generation, implementation, testing, and configuration management.

4. Integrated CASE tools :

- These are type of tools that integrate both upper and lower CASE, for example making it possible to design a form and build the database to support it at the same time.

- An automated system development environment that provides numerous tools to create diagrams, forms and reports.
- It also offers analysis, reporting and code generation facilities and seamlessly shares and integrates data across and between tools.

Two Marks Questions with Answers

Q.1 What is risk assessment ? Give the algorithm.

AU : IT, Dec.-05

Ans. : Risk assessment is an activity of assessing likelihood of the risk and impact of that risk. It is given to be a triplet as $[R_r, I_r, X_i]$ where R_i is risk, I_i is likelihood of risk and X_i is the impact of the risk.

Algorithm :

1. For the given project define the risk reference level.
2. Establish the relationship between the triplet $[R_r, I_r, X_i]$ and risk reference level.
3. Predict the set of reference points that define a region of termination bounded by area of certainty.
4. Predict how compound combinations of risks will affect a reference level.

Q.2 What are software planning activities ?

AU : IT, Dec.-03, CSE, May-09

OR

Highlight the activities in project planning.

AU : May-15

Ans. : The project planning activities can be carried out in following steps.

1. Identify the constraints in the project.
2. Make initial assessment of the project.
3. Define project milestones and deadlines.
4. While developing the project.
 - i) Prepare project schedule.
 - ii) Initiate project activities as per the project schedule.
 - iii) Review progress of the project.
 - iv) Update project schedule if required.
 - v) Revise project constraints and deliverables.
 - vi) If any problem arises then perform technical review.
5. Repeat the step 4 until the project is ready.

Q.3 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 or 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

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

2. Bohem simple model

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

Q.4 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.5 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.6 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.7 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.8 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.9 List down few process and product metrics.**AU : CSE, Dec.-06****Ans. :** Product metrics are

- Size metric - This metric is used for measuring the size of the software.
 - LOC based metric - This metric makes use of lines of code to measure the software.
 - FP based metric - This metric makes use of functional points to measure the software.
- Complexity metric - A software module can be described by a control flow graph.
 - Cyclomatic complexity
 - 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.10 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.11 How to measure the function point (FP) ? (Refer section 5.2.5)

AU : IT, Dec.-07

Q.12 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.13 What are risk management activities ?

AU : CSE, May-09

Ans. : Following are risk management activities -

1. Risk identification - All possible risks are anticipated and a list of potential risks is prepared.
2. Risk analysis - Under this activity the after effects of risks are obtained and risks are prioritised accordingly.
3. Risk planning - Risk avoidance or risk minimization plan is prepared.
4. Risk monitoring - Identified risks are mitigated.

Q.14 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.15 What information does a software project plan provide ?

AU : May-10

Ans. : Software project plan provides following information -

1. The quality plan describes the quality procedures and standards that will be used in a project.
2. The validation plan describes the approach, resources and schedule required for system validation.
3. The configuration management procedures and structures used are also described by the project plan.
4. The maintenance requirements of the system, maintenance cost and effort requirement information is described by the software project plan.

Q.16 What are the processes of risk management ? (Refer section 5.9)

AU : Dec.-10

Q.17 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.18 What are predictable risks ? Give two examples for such risks.

Ans. : Predictable risks are those risks that can be identified in advance based on past project experience. For example : Experienced and skilled staff leaving the organization in between.

Q.19 What is project planning?

AU : May.-12

Ans. : Project planning is an activity in which the project plan is prepared. This plan is mainly concerned with schedule and budget.

Q.20 An organic software occupies 15,000 LOC. How many programmers are needed to complete ?

AU : Dec.-12

Ans. : Given that :

$$\begin{aligned}
 \text{System} &= \text{Organic} \\
 \text{Lines of Code} &= 15\text{KLOC} \\
 E &= ab(\text{KLOC})^{b_b} \\
 &= 2.4(15)^{1.05} \\
 &= 41 \text{ persons-month}
 \end{aligned}$$

$$\begin{aligned}
 D &= c_b(E)d_b \\
 &= 2.5(41)^{0.38} \\
 &= 10 \text{ months} \\
 P &= 41/10 \\
 P &= 4 \text{ persons.}
 \end{aligned}$$

Approximately 4 programmers are required to complete the work.

Q.21 What do you mean by estimation risk ?

AU : May-13

Ans. : Estimation risk is measured by the degree of uncertainty in the quantitative estimates for cost, schedule and resources.

Q.22 Define software quality.

AU : Dec.-13

Ans. : The software quality can be defined as the degree to which a system component or process meets specific requirements.

Q.23 What is risk ? Give an example of risk.

AU : Dec.-13, 14

Ans. : Risk : The software risk can be defined as a problem that can cause some loss or can threaten the success of the project.

Example : Experienced and skilled staff leaving the organization in between.

Q.24 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.

Q.25 What is error tracking ?

AU : May-14

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.26 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.27 Define: Risk. (Refer Q.23)

AU : Dec.-14

Q.28 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.29 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.30 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.31 Define risk and its types.

AU : Dec.-15

Ans. : Risk is basically the uncertainty that may occur in the choices due to past actions and causes heavy losses.

Types of risks are - 1) Project Risk 2) Technical Risk 3) Business Risk

Q.32 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.33 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.34 What is Risk Management**AU : Dec.-16**

Ans. : Risk management refers to the process of making decision based on an evaluation of factors that treats to the business.

- Various activities that are carried out for risk management are
 - 1) Risk identification 2) Risk projection
 - 3) Risk refinement 4) Risk mitigation, monitoring and management.

Q.35 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.36 What are the different types of productivity estimation measures ?**AU : May-17**

Ans. : i) LOC based Estimation is used for productivity estimation as Productivity = K LOC/person-month

ii) Function Point based estimation can be used for productivity estimation as
Productivity = FP/person-month

Q.37 List two customer related and technology related risks.**AU : May-17****Ans. : Customer Related Risks**

- i) Is customer technically sophisticated in the product area
- ii) Does customer change the requirements quite often ?

- Technology Related Risks

i) Is the technology to be built is new to your company ?

ii) Is a specialized user interface demanded by product requirements ?

Q.38 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.

Q.39 Write a note on Risk Information Sheet (RIS).

AU : Dec.-17

Ans. : Risk Information sheet is used to store information about a risk.

- When new risks are identified then entry of such risks is made into the risk information sheet.
 - It is a form that can be submitted to the appropriate person or included in a database with other project risks.
 - **Use :** 1) This form is used for documentation and communication of risks.
- 2) It provides standardized format so risk information is readily accessible and understandable.

Q.40 What is EVA ?

AU : May-18

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.41 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.42 Compare : Project risk vs. Business risk.**AU : May-19**

Ans. : Project risks are related to the project objectives. The business risk is related to uncertainty in profits and danger of loss and the events that could pose a risk.

Q.43 List CASE tools for the following phases of SDLC : Design, testing.**AU : May-19**

Ans. : Design tools

- (1) Rational rose software for UML design.
- (2) Flow chart maker.
- (3) Star UML for designing the class diagram, use case diagram.

Testing tool

- (1) Selenium
- (2) TestComplete.

Q.44 Define reverse engineering.**AU : May-19**

Ans. : Reverse engineering is a process in which the dirty or unstructured code is taken processed and it is restructured.

Q.45 What is budgeted cost of work scheduled ?**AU : Dec.-19**

Ans. : The budgeted cost of work scheduled(BCWS) is the sum of the budgets for all work scheduled to be accomplished with a given time period. It is also called as Planned Value(PV). The BCWS also includes the cost of previous work completed and can address a specific period of performance or date in time.

Q.46 Write any two differences between known risk and predictable risks**AU : Dec.-19**

Ans. :

Known Risk	Predictable Risk
Known risks are those risks that can be identified in advance after evaluating the project plan, business and technical environment.	Predictable risks are those risks that can be identified in advance based on past project experience.
For example – unrealistic delivery date, poor requirement specification.	For example –Experienced and skilled staff leaving in between the project.



May - 2019
Software Engineering

Semester – IV (CSE)
Regulation 2017

AU
Solved Paper
(80101)

Time : Three Hours]

[Maximum Marks : 100

PART A - (10 × 2 = 20 Marks)

Note : Answer ALL questions.

- Q.1** Define : Software engineering. (Refer Two Marks Q.1 of Chapter - 1)
- Q.2** List any two agile process models. (Refer Two Marks Q.33 of Chapter - 1)
- Q.3** Differentiate : Functional and non-functional requirements.
(Refer Two Marks Q.35 of Chapter - 2)
- Q.4** State two advantages of using petri nets. (Refer Two Marks Q.33 of Chapter - 2)
- Q.5** How does the data flow diagram, help in design of software system ?
(Refer Two Marks Q.34 of Chapter - 2)
- Q.6** List the levels of testing. (Refer Two Marks Q.51 of Chapter - 4)
- Q.7** Define : Reverse engineering. (Refer Two Marks Q.44 of Chapter - 5)
- Q.8** List two advantages of using COCOMO model.
(Refer Two Marks Q.41 of Chapter - 5)
- Q.9** Compare : Project risk vs Business risk. (Refer Two Marks Q.42 of Chapter - 5)
- Q.10** List CASE tools for the following phases of SDLC : Design, testing.
(Refer Two Marks Q.43 of Chapter - 5)

PART B - (5 ×13 = 65 Marks)

- Q.11 a)** Compare the waterfall, prototyping and spiral model. List the features of each model, advantages and disadvantages and a type of application where the model will be acceptable. (Refer example 1.7.2) [13]

OR

- b)** i) Define agility. List any five principles of agility. (Refer section 1.9) [5]
ii) Explain the phases in extreme programming process. (Refer section 1.10) [8]
- Q.12 a)** 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. (Refer example 2.5.2) [13]

OR

- b) List any two techniques used for eliciting requirements. Compare the two techniques and list where each is applicable. **(Refer section 2.8)** [13]

- Q.13 a)** List and explain any five fundamental software design concepts. **(Refer section 3.3)** [13]

OR

- b) i) Define software architecture. **(Refer section 3.7)** [2]
ii) Explain and compare the following architectural styles : **(Refer section 3.9)**
1) Call and return architecture [4]
2) Object-oriented architecture [4]
3) Layered architecture [3]

- Q.14 a)** i) Compare white box and black box testing. **(Refer section 4.5)** [4]
ii) 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. **(Refer example 4.3.8)** [9]

OR

- b) i) Define : Refactoring. **(Refer section 4.12)** [2]
ii) List the phases in software re-engineering process model and explain each phase. **(Refer section 4.15)** [11]

- Q.15 a)** List the features of LOC and FP based estimation models. Compare the two models and list the advantages of one over other. **(Refer section 5.2)** [7 + 6]

OR

- b) i) Define : Risk. **(Refer section 5.9)** [2]
ii) List the types of risk and give examples for each. **(Refer section 5.9)** [5]
ii) List and explain the phases in risk management. **(Refer section 5.9)** [6]

PART C - (1 × 15 = 15 Marks)

- Q.16 a)** Given the requirements for an Automated Teller Machine (ATM) system (see below), design the following : **(Refer example 4.4.6)**
i) Use case diagram. [4]
ii) Activity diagram detailing each use case. [6]
iii) List test cases for any one functionality from your use case diagram. [5]
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.

OR

- b) i) Draw the level 0 and level 1 data flow diagram for the following system. [8]
ii) Identify entities in the system and draw a diagram showing the relationship between entities. [7]

The Chocolate Vending Machine (CVM) system requirements are as follows : The CVM dispenses 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 then dispense the proper candy and change and display : Thank You! (Refer example 2.11.12)



Notes

December - 2019
Software Engineering

Semester – IV (CSE), V(IT)
Regulation 2017

AU
Solved Paper
(90158)

Time : Three Hours]

[Maximum Marks : 100

PART A - (10 × 2 = 20 Marks)

Note : Answer ALL questions.

- Q.1** What is a software process ? (Refer Two Marks Q.23 of Chapter - 1)
- Q.2** Define an evolutionary prototype. (Refer Two Marks Q.34 of Chapter - 1)
- Q.3** What are non-functional requirements. (Refer Two Marks Q.3 of Chapter - 2)
- Q.4** Define a Petri net. (Refer Two Marks Q.27 of Chapter - 2)
- Q.5** What is inheritance ? (Refer Two Marks Q.43 of Chapter - 3)
- Q.6** Define a component. Give example. (Refer Two Marks Q.44 of Chapter - 3)
- Q.7** What is a test case ? (Refer Two Marks Q.51 of Chapter - 4)
- Q.8** Outline the need for system testing. (Refer Two Marks Q.52 of Chapter - 4)
- Q.9** What is budgeted cost of work scheduled ? (Refer Two Marks Q.45 of Chapter - 5)
- Q.10** Write any two differences between "known risks" and "Predictable risks".
(Refer Two Marks Q.46 of Chapter - 5)

PART B - (5 × 13 = 65 Marks)

- Q.11 a)** Outline the spiral life cycle model with a diagram. (Refer section 1.6)

OR

- b)** What is agility ? Elaborate the agile principles. (Refer section 1.9)

- Q.12 a) i)** Discuss the distinct tasks involved in requirement engineering process.

(Refer section 2.6)

[9]

- ii)** What does win-win mean in the context of negotiation during the requirements engineering activity ? (Refer example 2.6.1)

[4]

OR

- b)** Draw a Petri Net that depicts the operation of an "Automated Teller Machine". State the functional requirements you are considering. (Refer example 2.12.3)

[13]

- Q.13 a)** What is software architecture ? Outline the architectural styles with an example. (Refer section 3.9)

[13]

OR

- b) *Outline the steps in designing class based components with an example.*
(Refer section 3.13)

[13]

- Q.14 a) *Elaborate path testing and regression testing with an example.*
(Refer section 4.8)

[13]

OR

- b) i) *Explain how Business Process Reengineering (BPE) helps to achieve a defined business outcome.* (Refer section 4.14) [8]
ii) *Outline how the reverse engineering process helps to improve the legacy software.*
(Refer section 4.16) [5]

- Q.15 a) *Elaborate the cost estimation COCOMO II cost estimation model.*
(Refer section 5.5)

[13]

OR

- b) *Present a detailed note on risk management.* (Refer section 5.9)

[13]

PART C - (1 × 15 = 15 Marks)

- Q.16 a) *Prepare a software requirement specification document for a "Library Management System".* (Refer example 2.5.3) [15]

OR

- b) *Outline the steps in function point analysis with an example.* (Refer section 5.2)

[15]

□□□