

SUBJECT CODE : 3150711

As per New Syllabus of
GUJARAT TECHNOLOGICAL UNIVERSITY

Semester - V (CE / CSE) (Professional Elective - I)

SOFTWARE ENGINEERING

Anuradha A. Puntambekar

M.E. (Computer)

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



SOFTWARE ENGINEERING

Subject Code : 3150711

Semester - V (Computer Engineering / Computer Science & Engineering) (Professional Elective - I)

First Edition : August 2020

This edition is for sale in India only. Sale and Purchase of this book outside of India is unauthorized by the publisher.

© 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, Telefax : +91-020-24495497
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 978-81-946628-2-2



9788194662822

Course 18

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 - 3150711

Credits	Examination Marks				Total Marks	
	Theory Marks		Practical Marks			
	ESE (E)	PA (M)	ESE (V)	PA (I)		
04	70	30	30	20	150	

1. Introduction to Software and Software Engineering

The Evolving Role of Software, Software: A Crisis on the Horizon and Software Myths, Software Engineering: A Layered Technology, Software Process Models, The Linear Sequential Model, The Prototyping Model, The RAD Model, Evolutionary Process Models, Agile Process Model, Component-Based Development, Process, Product and Process. (**Chapter - 1**)

2. Agile Development

Agility and Agile Process model, Extreme Programming, Other process models of Agile Development and Tools. (**Chapter - 2**)

3. Managing Software Project

Software Metrics (Process, Product and Project Metrics), Software Project Estimations, Software Project Planning (MS Project Tool), Project Scheduling & Tracking, Risk Analysis &Management (Risk Identification, Risk Projection, Risk Refinement , Risk Mitigation). (**Chapter 3**)

4. Requirement Analysis and Specification

Understanding the Requirement, Requirement Modeling, Requirement Specification (SRS), Requirement Analysis and Requirement Elicitation, Requirement Engineering. (**Chapter - 4**)

5. Software Design

Design Concepts and Design Principal, Architectural Design, Component Level Design (Function Oriented Design, Object Oriented Design), User Interface Design, Web Application Design. (**Chapter - 5**)

6. Software Coding and Testing

Coding Standard and coding Guidelines, Code Review, Software Documentation, Testing Strategies, Testing Techniques and Test Case, Test Suites Design, Testing Conventional Applications, Testing Object Oriented Applications, Testing Web and Mobile Applications, Testing Tools (Win runner, Load runner). (**Chapter - 6**)

7. Quality Assurance and Management

Quality Concepts and Software Quality Assurance, Software Reviews (Formal Technical Reviews), Software Reliability, The Quality Standards: ISO 9000, CMM, Six Sigma for SE, SQA Plan. (**Chapter - 7**)

8. Software Maintenance and Configuration Management

Types of Software Maintenance, Re-Engineering, Reverse Engineering, Forward Engineering, The SCM Process, Identification of Objects in the Software Configuration, Version Control and Change Control (**Chapter - 8**)

9. DevOps

Overview, Problem Case Definition, Benefits of Fixing Application Development Challenges, DevOps Adoption Approach through Assessment, Solution Dimensions,

What is DevOps?, DevOps Importance and Benefits, DevOps Principles and Practices, 7 C's of DevOps Lifecycle for Business Agility, DevOps and Continuous Testing, How to Choose Right DevOps Tools, Challenges with DevOps Implementation, Must Do Things for DevOps, Mapping My App to DevOps - Assessment, Definition, Implementation, Measure and Feedback (**Chapter - 9**)

10. Advanced Topics in Software Engineering

Component-Based Software Engineering, Client/Server Software Engineering, Web Engineering, Reengineering, Computer-Aided Software Engineering, Software Process Improvement, Emerging Trends in software Engineering. (**Chapter - 10**)

TABLE OF CONTENTS

Chapter - 1 Introduction to Software and Software Engineering	
	(1 - 1) to (1 - 28)
1.1 Introduction	1 - 2
1.2 Evolving Role of Software	1 - 2
1.3 What is Software Engineering ?.....	1 - 3
1.4 Software Characteristics.....	1 - 4
1.5 Software : Crisis on the Horizon	1 - 6
1.6 Software Myths	1 - 7
1.7 Software Engineering : A Layered Technology	1 - 8
1.8 Process Framework	1 - 9
1.8.1 Common Process Framework.....	1 - 9
1.9 Need for Software Process Model.....	1 - 11
1.10 Software Process Model.....	1 - 12
1.10.1 Linear Sequential Model	1 - 12
1.10.2 Evolutionary Process Model	1 - 15
1.10.2.1 Prototype Model.	1 - 16
1.10.2.2 Spiral Model.	1 - 17
1.10.3 Incremental Process Model	1 - 20
1.10.3.1 RAD Model	1 - 21
1.10.4 Comparison between Various Process Models.....	1 - 23
1.11 Agile Process Models.....	1 - 26
1.12 Component Based Development.....	1 - 26
1.13 Product and Process	1 - 27
<hr/>	
Chapter - 2 Agile Development	
	(2 - 1) to (2 - 16)
2.1 Agility and Agile Process Model.....	2 - 2

2.1.1 Agility Principles.....	2 - 3
2.1.2 Concept of Agile Process.....	2 - 4
2.2 Extreme Programming.....	2 - 4
2.2.1 XP Values	2 - 4
2.2.2 Process	2 - 5
2.2.3 Industrial XP.....	2 - 7
2.3 Other Process Models of Agile Development.....	2 - 8
2.3.1 Adaptive Software Development (ASD).....	2 - 9
2.3.2 Dynamic System Development Method (DSDM)	2 - 10
2.3.3 Scrum	2 - 11
2.3.4 Feature Driven Development (FDD).....	2 - 13
2.3.5 Crystal	2 - 14
2.3.6 Agile Modeling (AM)	2 - 15
2.4 Tools for Agile Process.....	2 - 16

Chapter - 3 Managing Software Project	(3 - 1) to (3 - 42)
3.1 Software Process and Project Metrics.....	3 - 2
3.1.1 Metrics in Process and Project Improvement.....	3 - 2
3.1.2 W5HH Principle	3 - 3
3.2 Software Measurement	3 - 4
3.2.1 Size Oriented Metrics	3 - 5
3.2.2 Function Oriented Metrics	3 - 6
3.2.3 Object-Oriented Metrics.....	3 - 10
3.2.4 Attributes of Effective Software Metrics	3 - 10
3.3 Product Metrics	3 - 11
3.3.1 Metrics For Analysis Model.....	3 - 12
3.3.2 Metrics For Design Model.....	3 - 12
3.3.2.1 Architectural Design Complexity.	3 - 12
3.3.2.2 Metrics for Object Oriented Design	3 - 12
3.3.2.3 User Interface Design	3 - 13
3.3.3 Metrics for Source Code	3 - 14

3.3.4 Metrics for Testing	3 - 17
3.3.5 Metrics for Maintenance	3 - 17
3.4 Software Project Estimations	3 - 17
3.4.1 Software Sizing	3 - 17
3.4.2 Problem based Estimation	3 - 19
3.4.3 Example of LOC based Estimation	3 - 19
3.4.4 Example of FP based Estimation	3 - 20
3.4.5 Process based Estimation	3 - 22
3.4.6 Example of Process based Estimation	3 - 22
3.4.7 Estimation with Use-Cases	3 - 23
3.4.8 Reconciling Estimates	3 - 25
3.5 Software Project Planning	3 - 25
3.5.1 MS Project Planning Tool	3 - 26
3.6 Project Scheduling and Tracking.....	3 - 27
3.6.1 Project Scheduling Process	3 - 27
3.6.2 Defining a Task Set for the Software Project.....	3 - 27
3.6.3 Task Network	3 - 29
3.6.4 Time Line Chart (Gantt Chart).	3 - 29
3.6.5 Tracking the Schedule	3 - 31
3.7 Risk Analysis and Management	3 - 32
3.7.1 Software Risks	3 - 32
3.7.2 Reactive Vs. Proactive Risk Strategies	3 - 33
3.8 Risk Identification	3 - 34
3.8.1 Risk Components and Drivers.....	3 - 35
3.9 Risk Projection	3 - 36
3.9.1 Building Risk Table.....	3 - 37
3.9.2 Assessing Risk Impact	3 - 38
3.10 Risk Refinement.....	3 - 39
3.11 Risk Mitigation	3 - 39
3.12 Risk Plan.....	3 - 41

Chapter - 4 Requirements Analysis and Specification(4 - 1) to (4 - 104)

4.1 Introduction	4 - 2
4.1.1 Need for Requirements to be Stable and Correct.....	4 - 3
4.2 Requirement Engineering Task.....	4 - 3
4.2.1 Inception.....	4 - 4
4.2.2 Elicitation	4 - 5
4.2.3 Elaboration	4 - 5
4.2.4 Negotiation.....	4 - 5
4.2.5 Specification.....	4 - 6
4.2.6 Validation	4 - 6
4.2.7 Requirement Management.....	4 - 6
4.3 Initiating Requirements Engineering Process.....	4 - 8
4.3.1 Identification of Stakeholders.....	4 - 8
4.3.2 Recognizing Multiple Viewpoints.....	4 - 9
4.3.3 Working towards the Collaboration.....	4 - 10
4.3.4 Questioning	4 - 10
4.4 Eliciting Requirements.....	4 - 11
4.4.1 Collaborative Requirements Gathering	4 - 11
4.4.2 Quality Function Deployment.....	4 - 13
4.4.3 Usage Scenarios	4 - 14
4.4.4 Elicitation Work Product.....	4 - 14
4.5 Developing Use Cases	4 - 15
4.6 Negotiating Requirements.....	4 - 19
4.7 Validating Requirements	4 - 20
4.8 Prioritizing Requirements	4 - 20
4.9 Building Requirements Analysis Model	4 - 22
4.9.1 Overall Objectives	4 - 23
4.10 Elements of Requirements Analysis	4 - 23
4.11 Scenario Based Modeling	4 - 24

4.11.1 Writing Use Cases	4 - 24
4.11.2 Activity Diagram.....	4 - 29
4.11.3 Swimlane Diagram.....	4 - 30
4.12 Class Based Modeling	4 - 34
4.12.1 Objects and Object Classes	4 - 34
4.12.2 Generalization and Inheritance Relationship	4 - 35
4.12.2.1 Association Relationship	4 - 38
4.12.3 Object Identification	4 - 40
4.12.4 Class-Responsibility-Collaborator(CRC) Modelling	4 - 42
4.12.5 Object Relationship Model	4 - 45
4.13 Data Modeling	4 - 47
4.13.1 Data Objects, Attributes and Relationships.....	4 - 47
4.13.2 Cardinality Modality	4 - 49
4.13.3 Entity Relationship Diagram	4 - 50
4.14 Flow Oriented Modeling.....	4 - 56
4.14.1 Data Flow Diagram	4 - 57
4.14.1.1 Designing Data Flow Diagrams	4 - 57
4.14.2 Examples	4 - 60
4.14.3 Difference between DFD and ER	4 - 75
4.15 Behavioral Modeling.....	4 - 77
4.15.1 State Representation.....	4 - 77
4.15.2 Sequence Diagram.....	4 - 78
4.16 Software Requirements Specification(SRS)	4 - 79
4.16.1 Characteristics of Good SRS	4 - 84
4.16.2 Example of SRS	4 - 86

Chapter - 5 Software Design

(5 - 1) to (5 - 46)

5.1 Definition of Software Design.....	5 - 2
5.1.1 Designing within the Context of Software Engineering	5 - 2
5.2 Design Concepts	5 - 3
5.2.1 Abstraction	5 - 3
5.2.2 Modularity	5 - 4

5.2.3 Architecture	5 - 5
5.2.4 Refinement	5 - 6
5.2.5 Pattern	5 - 6
5.2.6 Information Hiding.....	5 - 6
5.2.7 Functional Independence	5 - 7
5.2.7.1 Cohesion	5 - 7
5.2.7.2 Coupling	5 - 8
5.2.8 Refactoring	5 - 8
5.3 Design Principles.....	5 - 9
5.4 Design Model.....	5 - 10
5.4.1 Data Design Element	5 - 11
5.4.2 Architectural Design Element.....	5 - 11
5.4.3 Interface Design Elements	5 - 11
5.4.4 Component Level Design Elements	5 - 11
5.4.5 Deployment Level Design Elements.....	5 - 12
5.5 Architectural Design	5 - 12
5.5.1 Software Architecture	5 - 12
5.5.1.1 Structural Partitioning	5 - 13
5.5.1.2 Comparison between Horizontal and Vertical Partition	5 - 15
5.5.2 Architectural Style	5 - 15
5.5.2.1 Data Centered Architectures	5 - 15
5.5.2.2 Data Flow Architectures	5 - 16
5.5.2.3 Call and Return Architecture	5 - 17
5.5.2.4 Object Oriented Architecture	5 - 18
5.5.2.5 Layered Architecture	5 - 19
5.6 Component Level Design	5 - 19
5.6.1 Function Oriented Design.....	5 - 20
5.6.2 Object Oriented Design.....	5 - 25
5.7 User Interface Design	5 - 29
5.7.1 User Interface Design Principles.....	5 - 30
5.7.2 Golden Rules	5 - 30

5.7.2.1 Place the User in Control	5 - 31
5.7.2.2 Reduce the User's Memory Load	5 - 31
5.7.2.3 Make the Interface Consistent	5 - 32
5.7.3 Interface Design Steps.....	5 - 33
5.7.4 User Interface Design Process	5 - 33
5.8 Web Application Design	5 - 35
5.8.1 Design Pyramid	5 - 35
5.8.1.1 Interface Design	5 - 36
5.8.1.2 Aesthetic Design	5 - 39
5.8.1.3 Content Design	5 - 40
5.8.1.4 Architecture Design	5 - 40
5.8.1.5 Navigation Design	5 - 43
5.8.1.6 Component Level Design	5 - 45

Chapter - 6 Software Coding and Testing	(6 - 1) to (6 - 46)
--	----------------------------

6.1 Coding Standard and Coding Guidelines	6 - 2
6.1.1 Coding Guideline	6 - 2
6.1.2 Coding Standards.....	6 - 4
6.2 Code Review	6 - 5
6.2.1 Planning for Review	6 - 6
6.2.2 Self Review	6 - 6
6.2.3 Group Review Meeting	6 - 7
6.3 Software Documentation	6 - 8
6.4 Introduction to Software Testing	6 - 9
6.4.1 Testing Objectives	6 - 9
6.4.2 Testing Principles	6 - 9
6.4.3 Why Testing is Important ?	6 - 10
6.5 Testing Strategies	6 - 10
6.5.1 Unit Testing	6 - 11
6.5.2 Integration Testing	6 - 12
6.5.2.1 Top Down Integration Testing.	6 - 14

6.5.2.2 Bottom Up Integration Testing	6 - 14
6.5.2.3 Regression Testing	6 - 15
6.5.2.4 Smoke Testing	6 - 16
6.5.3 Validation Testing	6 - 16
6.5.3.1 Acceptance Testing	6 - 17
6.5.4 System Testing	6 - 17
6.5.4.1 Recovery Testing	6 - 18
6.5.4.2 Security Testing	6 - 18
6.5.4.3 Stress Testing	6 - 18
6.5.4.4 Performance Testing	6 - 18
6.6 Testing Conventional Applications	6 - 19
6.7 White Box Testing	6 - 19
6.7.1 Basis Path Testing	6 - 19
6.7.1.1 Flow Graph Notation	6 - 20
6.7.1.2 Graph Matrices	6 - 24
6.7.2 Control Structure Testing	6 - 25
6.7.2.1 Condition Testing	6 - 26
6.7.2.2 Loop Testing	6 - 26
6.7.2.3 Data Flow Testing	6 - 28
6.8 Black-Box Testing	6 - 29
6.8.1 Equivalence Partitioning	6 - 30
6.8.2 Boundary Value Analysis (BVA)	6 - 31
6.8.3 Graph based Testing	6 - 32
6.8.4 Orthogonal Array Testing	6 - 32
6.9 Comparison between Black Box and White Box Testing	6 - 34
6.10 Testing Technique and Test Case	6 - 36
6.10.1 Test Case Execution	6 - 37
6.11 Test Suites Design	6 - 38
6.12 Testing Object Oriented Applications	6 - 39
6.12.1 Conventional Test Case Design Methods	6 - 39

6.12.2 Fault based Testing	6 - 39
6.12.3 Scenario based Testing	6 - 39
6.13 Object-Oriented Testing Strategies	6 - 40
6.13.1 Unit Testing in OO Context	6 - 40
6.13.2 Integration Testing in OO Context	6 - 40
6.13.3 Difference between OO Testing Strategy and Conventional Testing Strategy	6 - 41
6.14 Testing Web Applications	6 - 42
6.15 Testing Mobile Applications	6 - 43
6.16 Testing Tools	6 - 45
6.16.1 Win Runner	6 - 45
6.16.2 Load Runner	6 - 46

Chapter - 7 Quality Assurance and Management (7 - 1) to (7 - 16)

7.1 Quality Concepts.....	7 - 2
7.1.1 Quality.	7 - 2
7.1.2 Quality Control.	7 - 2
7.1.3 Quality Assurance	7 - 3
7.1.4 Cost of Quality	7 - 4
7.2 Software Quality Assurance (SQA)	7 - 5
7.2.1 Software Quality Assurance (SQA) Activities.	7 - 5
7.3 Software Reviews	7 - 7
7.3.1 Formal Technical Reviews (FTR)	7 - 7
7.3.1.1 The Review Meeting	7 - 8
7.3.1.2 Review Reporting and Record Keeping	7 - 8
7.3.1.3 Review Guidelines	7 - 9
7.4 Software Reliability.....	7 - 9
7.4.1 Measure of Reliability and Availability.	7 - 10
7.4.2 Software Safety	7 - 10
7.5 Quality Standards	7 - 12
7.5.1 ISO 9000	7 - 12
7.5.2 CMM	7 - 14

7.5.3 Six Sigma.....	7 - 14
7.6 SQA Plan	7 - 15

Chapter - 8 Software Maintenance and Configuration Management (8 - 1) to (8 - 14)

8.1 Software Maintenance	8 - 2
8.1.1 Need for Maintenance	8 - 2
8.1.2 Types of Software Maintenance	8 - 2
8.2 Re-Engineering.....	8 - 3
8.3 Reverse Engineering	8 - 4
8.3.1 Reverse Engineering Process	8 - 5
8.4 Forward Engineering.....	8 - 6
8.4.1 Forward Engineering for Client Server Architectures	8 - 6
8.4.2 Forward Engineering for Object Oriented Architectures	8 - 7
8.5 Introduction to Software Configuration Management (SCM).....	8 - 8
8.5.1 Need for SCM	8 - 8
8.6 Software Configuration Items.....	8 - 8
8.7 SCM Process	8 - 9
8.7.1 Identification of Objects in Software Configuration	8 - 10
8.7.2 Change Control	8 - 11
8.7.3 Version Control	8 - 13
8.7.4 Configuration Audit	8 - 13
8.7.5 Status Reporting	8 - 14

Chapter - 9 DevOps (9 - 1) to (9 - 18)

9.1 Overview.....	9 - 2
9.1.1 Difference between DevOps and Agile	9 - 2
9.2 Problem Case Definition	9 - 3
9.2.1 Challenges in Application Development	9 - 4
9.3 Benefits of Fixing Application Development Challenges	9 - 4
9.4 DevOps Adoption Approach through Assessment	9 - 4

9.5 Solution Dimensions	9 - 5
9.6 What is DevOps?.....	9 - 6
9.7 DevOps Importance and Benefits	9 - 6
9.8 DevOps Principles and Practices.....	9 - 8
9.9 The 7 C's of DevOps Lifecycle for Business Agility	9 - 9
9.10 DevOps and Continuous Testing.....	9 - 10
9.10.1 Continuous Testing Process in DevOps	9 - 10
9.10.2 Advantages of Continuous Testing.....	9 - 11
9.10.3 Challenges in Continuous Testing	9 - 11
9.11 How to Choose Right DevOps Tools	9 - 12
9.12 Challenges with DevOps Implementation	9 - 13
9.13 Must Do Things for DevOps	9 - 14
9.14 Mapping My App to DevOps.....	9 - 14
9.15 Assessment, Definition, Implementation, Measure and Feedback.....	9 - 15

Chapter - 10 Advanced Topics in Software Engineering

(10 - 1) to (10 - 18)

10.1 Component Based Software Engineering (CBSE).....	10 - 2
10.1.1 Component and Component Models.	10 - 2
10.1.2 Component Models.	10 - 3
10.1.3 CBSE Process	10 - 3
10.2 Client Server Software Engineering	10 - 4
10.2.1 Two Tier Architecture	10 - 4
10.2.2 Three Tier Architecture	10 - 5
10.3 Web Engineering	10 - 6
10.3.1 Attributes of Web Based Applications.....	10 - 6
10.3.2 Design Model for Web Based Applications	10 - 7
10.4 Computer Aided Software Engineering (CASE).....	10 - 8
10.4.1 Building Blocks of CASE	10 - 8
10.4.1.1 Taxonomy of CASE Tools	10 - 9

10.4.2 Integrated CASE Environment	10 - 12
10.5 Software Process Improvement	10 - 14
10.5.1 SPI Model	10 - 14
10.6 Emerging Trends in Software Engineering.....	10 - 15
10.6.1 Process Trends.....	10 - 15
10.6.2 Collaborative Development	10 - 15
10.6.3 Model Driven Development	10 - 16
10.6.4 Test Driven Development.....	10 - 16

1

Introduction to Software and Software Engineering

Syllabus

The evolving role of software, Software : A crisis on the horizon and software myths, Software engineering : A layered technology, Software process models, The linear sequential model, The prototyping model, The RAD model, Evolutionary process models, Agile process model, Component-based Development, Process, Product and process.

Contents

1.1	<i>Introduction</i>	
1.2	<i>Evolving Role of Software</i>	
1.3	<i>What is Software Engineering ?</i>	
1.4	<i>Software Characteristics</i>	Winter-2012, 2013, Marks 7
1.5	<i>Software : Crisis on the Horizon</i>	
1.6	<i>Software Myths</i>	
1.7	<i>Software Engineering : A Layered Technology</i>	Winter-2011, 2014, 2017, 2018, Summer-2012, 2019, Marks 7
1.8	<i>Process Framework</i>	Winter-2013, 2014, Summer-2018, Marks 7
1.9	<i>Need for Software Process Model</i>	Summer-2016, Marks 3
1.10	<i>Software Process Model</i>	Winter-2011, 2012, 2014, 2017, 2018, 2019, Summer-2011, 2012, 2013, 2014, 2015, 2016, 2019, Marks 7
1.11	<i>Agile Process Models</i>	Winter-17, Marks 4
1.12	<i>Component Based Development</i>	
1.13	<i>Product and Process</i>	Winter-2011, 2019, Summer-2014, 2019, Marks 7

1.1 Introduction

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.

1.2 Evolving Role of Software

The evolving role of software means changing role of **software**. Basically any software appears in two roles :

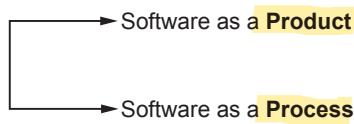


Fig. 1.2.1

Being a **product** the role of **software** can be recognised by its **computing potentials**, **hardware capabilities** and **accessibility** of **network computers** by the **local hardware**. Being a **product** it also acts as an **information transformer** i.e. **producing**, **managing**, **modifying** and **conveying** the source of information.

Being a **process** the software acts as a **vehicle** for **driving the product**. In this role the duty of software is to **control the computer** or to establish **communications** between the **computers**. Thus software plays **dual role**. It is both a product and a vehicle for delivering a product.

The role of software is significantly changing over past decade. There are many **factors affecting the role of software** and those are -

- **Changes in computer architectures** (Right from Pentium I to supercomputers)
- Improvement in **hardware performance**.
- Vast **increase in amount of memory**
- Wide variety of **input and outputs**(Ranging from simple text to multimedia videos)

Following table presents the different roles of software in different era of computing.

Era of computing	Software
Early Years	Batch processing Custom software
Second Era	Multi user Real time systems Database systems Product software

Third Era	Distributed systems
Forth Era	Object oriented systems Expert systems Parallel computing Network computers
Fifth Era	Web technologies mobile computing

- As 1990 began, **Toffler** described power shift. That is-old systems such as Government, educational, economic, military make use of computers performing the assigned tasks. And software lead to **democratization of knowledge**.

Democratization of knowledge means use of computers at all the public places and development of useful software applications allowed more and more people to access it. New technologies and improved user experience increasing number of users of the computers. Due to increasing scale, consumers have greater access to use and purchase technologically sophisticated products and services.

But this evolutionary role of software brings some crucial problems. Here are some sample **problems** encountered due to evolution on software

1. Advances in hardware demand for more capable software.
2. Ability to build new programs can not meet the demand for new programs and such programs are not sufficient for business and market needs.
3. Vast use of computer based systems brings less use of manpower and ultimately society becomes more dependant on machine and not on man.
4. Constant struggle for high reliability and quality software.

Review Questions

1. Explain the role of software in democratization of knowledge.
2. Explain the evolving role of software.

1.3 What is Software Engineering ?

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

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

Role of Software Engineer

The software engineer has to adopt systematic and organized approach in order to produce high quality software. He is also responsible for selecting the most appropriate method for software development.

1.4 Software Characteristics

GTU : Winter-2012, 2013, Marks 7

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 in Fig. 1.4.1.

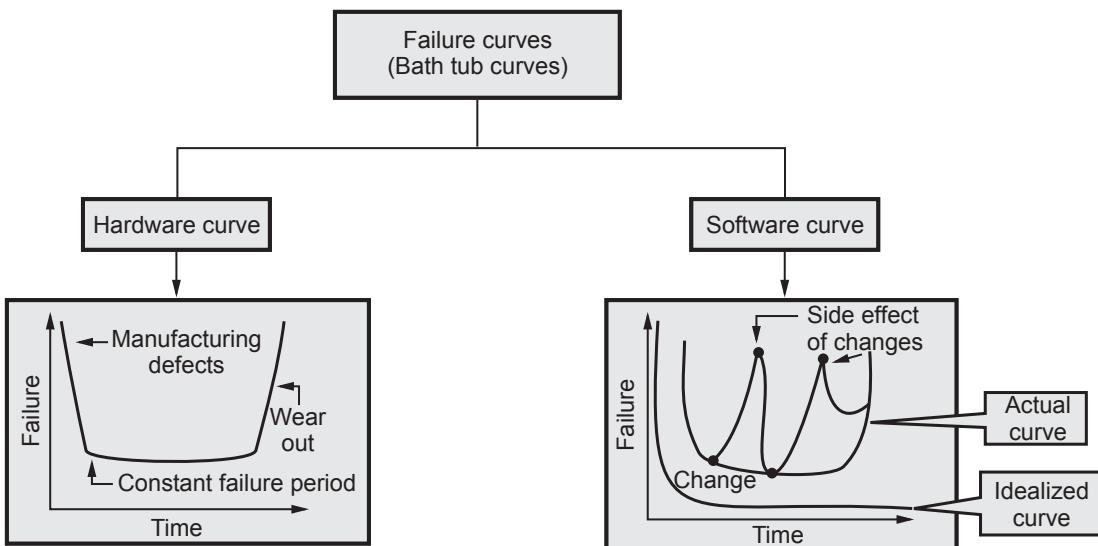


Fig. 1.4.1 Failure curves for hardware and software

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*.

Comparison between Hardware and Software Product characteristics

Sr. No.	Hardware product characteristics	Software product characteristics
1.	Hardware products are manufactured . The quality problems that occur in hardware manufacturing phase can not be removed easily .	Software is engineered and not manufactured . During software development process, various problems can be identified and rectified .
2.	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.	On the other hand, software does not get affected from the environmental maladies . But due to some undiscovered errors the failure rate is high and drops down as soon as the errors get corrected.
3.	There are spare parts available for the hardware components.	There are no spare parts for software components to replace.
4.	The hardware unit is assembled from components	Most software is custom built rather than being assembled from components .

Review Questions

1. What is software engineering ? What is the role of software engineer? Compare hardware and software characteristics.

GTU : Winter-2012, Marks 7

2. Explain the difference between software and hardware characteristics.

GTU : Winter-2013, Marks 4

1.5 Software : Crisis on the Horizon

The software crisis means the decisive time or turning point that software developers encounter during software development. Hence **software crisis** represent various problems that are faced by the software developers during software development process.

To understand software crisis consider following problem that is often faced by software industry.

Software cost is getting increased tremendously day-by-day. The software purchase expenses are higher than the hardware purchase. This is becoming worrying trend over the years. Following graph shows the ratio of h/w and s/w cost vs years -

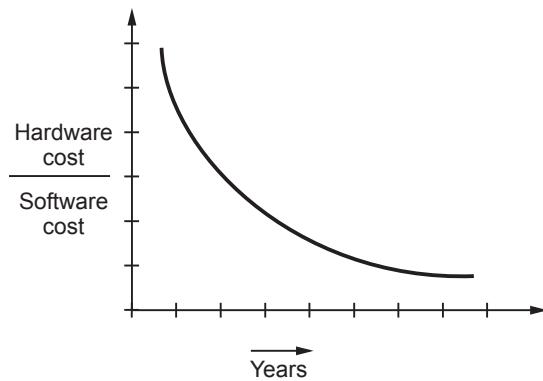


Fig. 1.5.1 Software cost : A software crisis

This graph shows that cost of software is increasing rapidly than the hardware cost. Following are the symptoms of present software crisis -

1. Day-by-day, software purchase cost is getting more than the hardware purchase cost. Hence major part of budget of any software industry is on software purchase.
2. Software products are difficult to alter, maintain, enhance, debug or modify.
3. Software resources are not being used optimally.
4. User requirements are often evolving and cannot be satisfied fully.
5. Software product not being reliable.
6. Many time software products get crashed on occurrence of specific condition.
7. Delivery of software product within specified budget and on scheduled time.

Various factors that have contributed to make software crisis

Following are some factors that bring the software crisis -

- i) Increasing size or volume of software.
- ii) Lowered productivity or quality improvement.

- iii) Lack of skilled staff.
- iv) Inadequate software training.
- v) Growing demand for more software.

Solutions to present software crisis -

The most effective solutions to present software crisis can be i) use and spread of software engineering practices among software engineers and ii) further enhancement in software engineering disciplines.

1.6 Software Myths

There are some misbeliefs in the software industry about the software and process of building software. For any software developer it is a must to know such beliefs and reality about them. Here are some typical myths -

- **Myth :** Using a collection of standards and procedures one can build software. (Management Myth)
Reality : Eventhough we have all standards and procedures with us for helping the developer to build software, it is not possible for software professionals to build desired product. This is because - the collection which we have should be complete, it should reflect modern techniques and more importantly it should be adaptable. It should also help the software professional to bring quality in the product.
- **Myth :** Add more people to meet deadline of the project. (Management Myth)
Reality : Adding more people in order to catch the schedule will cause the reverse effect on the software project i.e. software project will get delayed. Because, we have to spend more time on educating people or informing them about the project.
- **Myth :** If a project is outsourced to a third party then all the worries of software building are over. (Management Myth)
Reality : When a company needs to outsource the project then it simply indicates that the company does not know how to manage the projects. Sometimes, the outsourced projects require proper support for development.
- **Myth :** Even if the software requirements are changing continuously it is possible to accommodate these changes in the software. (Customer Myth)
Reality : It is true that software is a flexible entity but if continuous changes in the requirements have to be incorporated then there are chances of introducing more and more errors in the software. Similarly, the additional resources and more design modifications may be demanded by the software.
- **Myth :** We can start writing the program by using general problem statements only. Later on using problem description we can add up the required functionalities in the program. (Customer Myth)
Reality : It is not possible each time to have comprehensive problem statement.

We have to start with general problem statements; however by proper communication with customer the software professionals can gather useful information. The most important thing is that the problem statement should be unambiguous to begin with.

- **Myth :** Once the program is running then its over! (Practitioner's Myth)
Reality : Even though we obtain that the program is running major part of work is after delivering it to customer.
- **Myth :** Working program is the only work product for the software project. (Practitioner's Myth)
Reality : The working program/software is the major component of any software project but along with it there are many other elements that should be present in the software project such as documentation of software, guideline for software support.
- **Myth :** There is no need of documenting the software project; it unnecessarily slows down the development process. (Practitioner's Myth)
Reality : Documenting the software project helps in establishing ease in use of software. It helps in creating better quality. Hence documentation is not wastage of time but it is a must for any software project.

1.7 Software Engineering : A Layered Technology

**GTU : Winter-2011, 2014, Marks 7, Winter-2017, 2018, Marks 3,
Summer-2012, 2019, Marks 4**

- 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**.

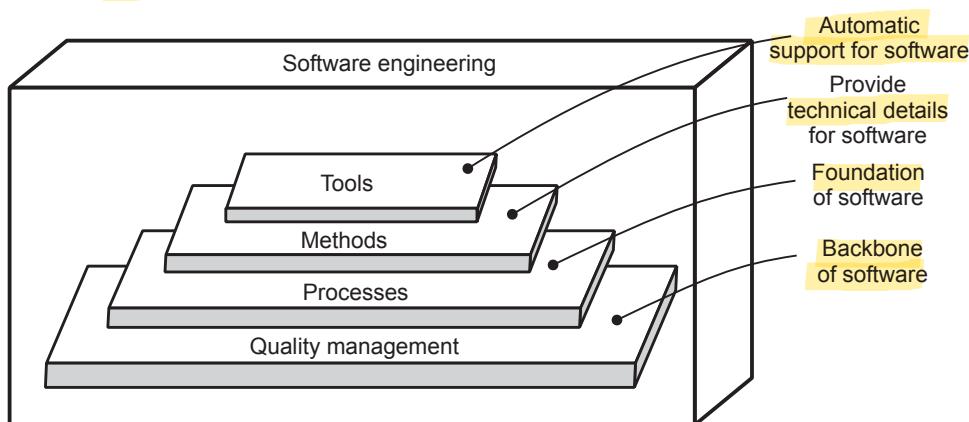


Fig. 1.7.1 Software engineering : A layered approach

- 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.

Review Questions

1. Define Software Engineering. Draw and explain software Engineering layers.

GTU : Summer-2012, Winter-2014, Marks 7

2. Explain software engineering as a layered technology.

GTU : Winter-2011, Marks 3,

Winter-2017,2018, Marks 3, Summer-2019, Marks 4

1.8 Process Framework

GTU : Winter-2013, 2014, Marks 7, Summer-2018, Marks 3

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.

1.8.1 Common Process Framework

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

As shown in Fig. 1.8.1, the software process is characterized by process framework activities, task sets and umbrella activities. (See Fig. 1.8.1 on next page.)

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.

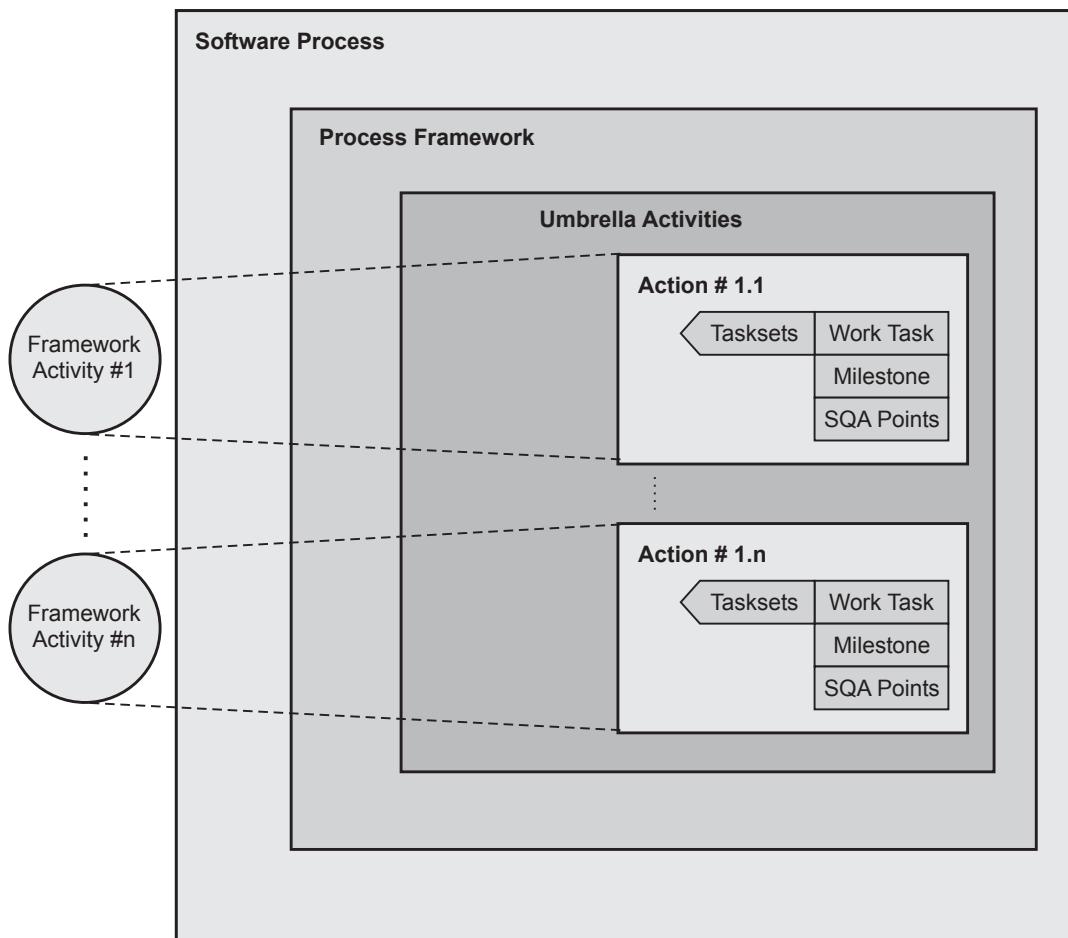


Fig. 1.8.1 Software process framework

- **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

imp.

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.

Review Questions

1. Discuss all generic framework activities with respect to any one process model.

GTU : Winter-2013, Winter-2014, Marks 7

2. Discuss **umbrella activities** and its role in **software development life cycle (SDLC)**.

GTU : Winter-2014, Marks 7

3. What is process ? Discuss the **process framework activities**.

GTU : Summer 18, Marks 3

1.9 Need for Software Process Model

GTU : Summer-2016, Marks 3

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.

Review Question

- What is the importance of process model in development of software system.

GTU : Summer-2016, Marks 3

1.10 Software Process Model

**GTU : Winter-2011, 2012, 2014, 2017, 2018, 2019,
Summer-2011, 2012, 2013, 2014, 2015, 2016, 2019, Marks 7**

- 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.

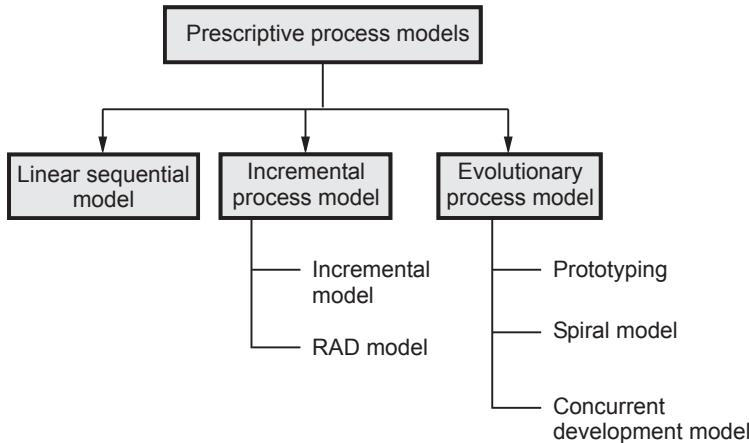


Fig. 1.10.1 Prescriptive process model

- Various prescriptive process models are as shown in Fig. 1.10.1.

1.10.1 Linear Sequential Model

- The linear sequential model is also called as '**waterfall 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.

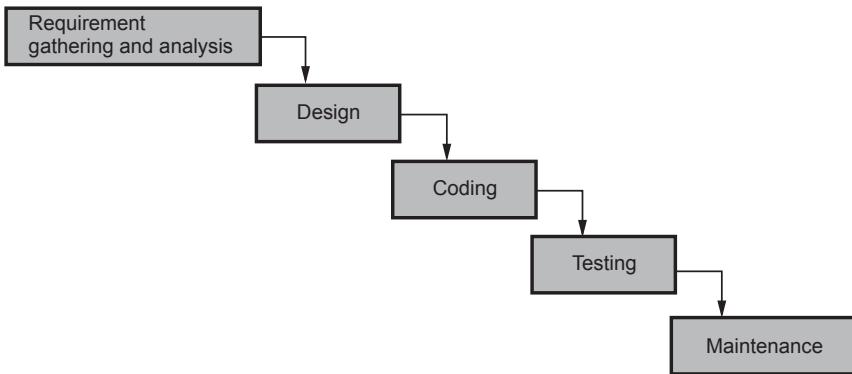


Fig. 1.10.2 Waterfall model

- 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 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 1.10.1 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.

Example 1.10.2 What is meant by 'blocking states' in linear sequential model ?

Solution : 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.

Example 1.10.3 Provide three examples of software projects that would be amenable to the waterfall model, Be specific.**GTU : Winter-2019, Marks 3**

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.

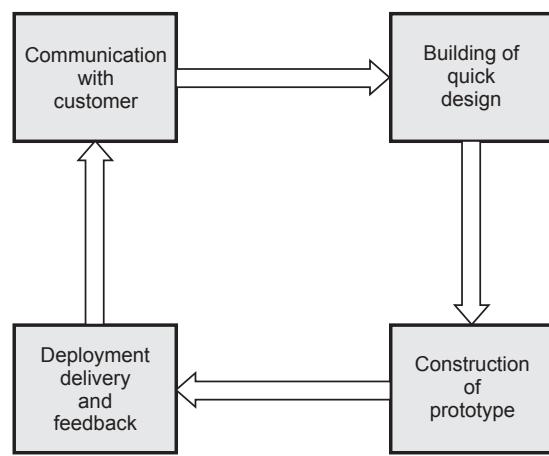
b) Interactive system that allows railway passengers to find time and other information from the terminals installed in the station.

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.

c) Inventory Management System : The waterfall model is a linear sequential model in which progress can be seen as flowing steadily downwards through various phases. Once the customer is satisfied with all the requirement analysis, the further development of the project becomes simplified. This model is more appropriate for handling such projects because the requirements are very limited and specific for such type of projects.

1.10.2 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**.

**Fig. 1.10.3 Prototype model**

1.10.2.1 Prototype Model

- 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.
- 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 prototype model

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 prototype model and incremental process model

Sr. No.	Prototype model	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.

1.10.2.2 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. 1.10.4.
- 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.

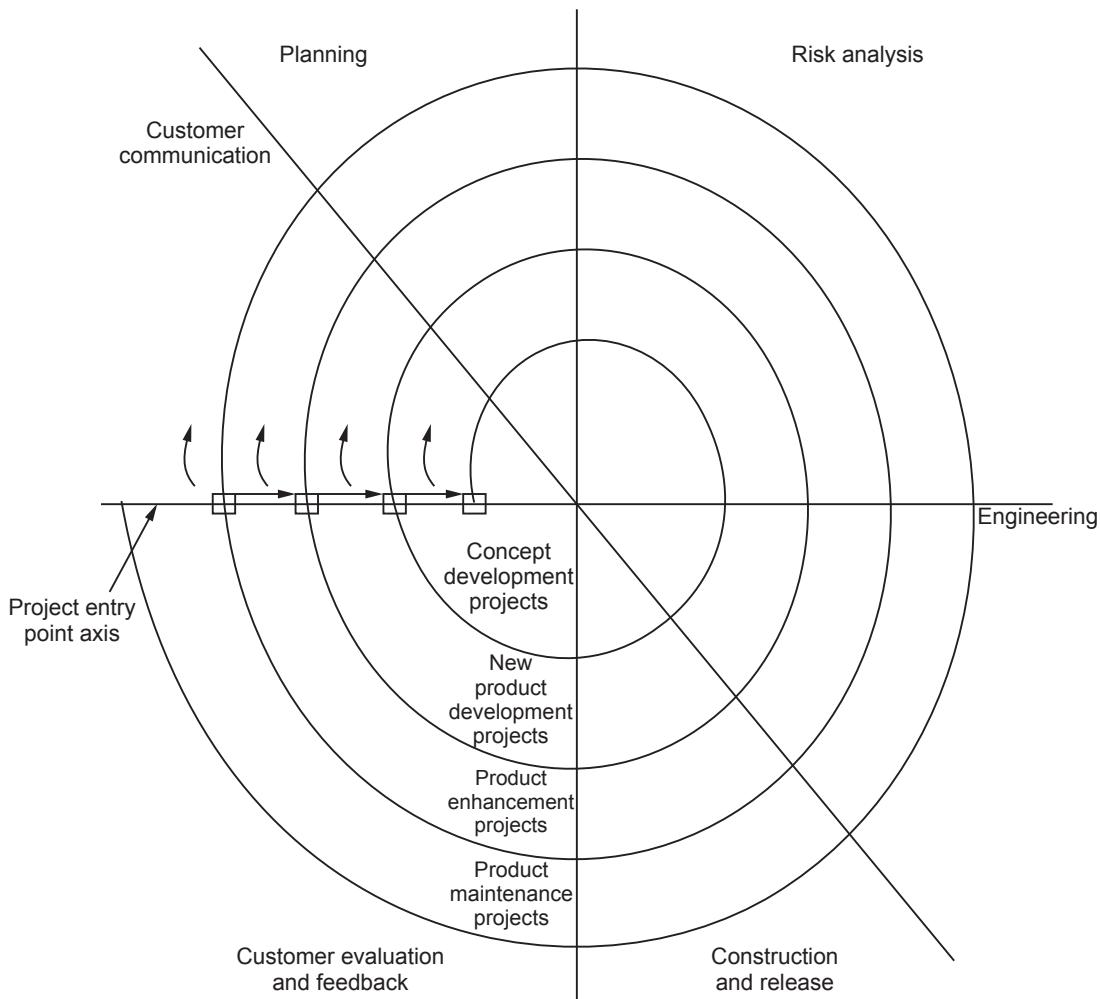


Fig. 1.10.4 Spiral model

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

- The task regions can be described as :
- i) **Customer communication** - In this region, it is suggested to establish customer communication.
 - ii) **Planning** - All planning activities are carried out in order to define resources time line and other project related activities.
 - iii) **Risk analysis** - The tasks required to calculate technical and management risks are carried out.

- iv) **Engineering** - In this task region, tasks required to build one or more representations of applications are carried out.
- v) **Construct and release** - All the necessary tasks required to construct, test, install the application are conducted. Some tasks that are required to provide user support are also carried out in this task region.
- vi) **Customer evaluation** - Customer's feedback is obtained and based on customer evaluation required tasks are performed and implemented at installation stage.
- In each region, number of **work tasks** are carried out depending upon the characteristics of project. For a small project relatively small number of work tasks are adopted but for a complex project large number of work tasks can be carried out.
- In spiral model, the software engineering team **moves around the spiral** in a **clockwise direction** beginning at the core.

Advantages of spiral model

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

Drawbacks of spiral model

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

When to choose it ?

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

Comparison between Spiral model and Prototyping model

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

2.	All the end-users need not be involved in all the phases of development .	All the end-users are involved in all phases of development .
3.	Funding are not stable for the projects that can be developed using spiral model.	Funding are stable for these type of 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.

Example 1.10.4 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 ?

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 1.10.5 How does "Project Risk" factor affect the spiral model of software development ?

Solution : 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.

Example 1.10.6 How does a spiral model represent a process suitable to represent a real time problem ?

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.

1.10.3 Incremental Process Model

In this model with limited functionality the basic software product is created for user's understanding. This model is refined and expanded in later phases.

1.10.3.1 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.

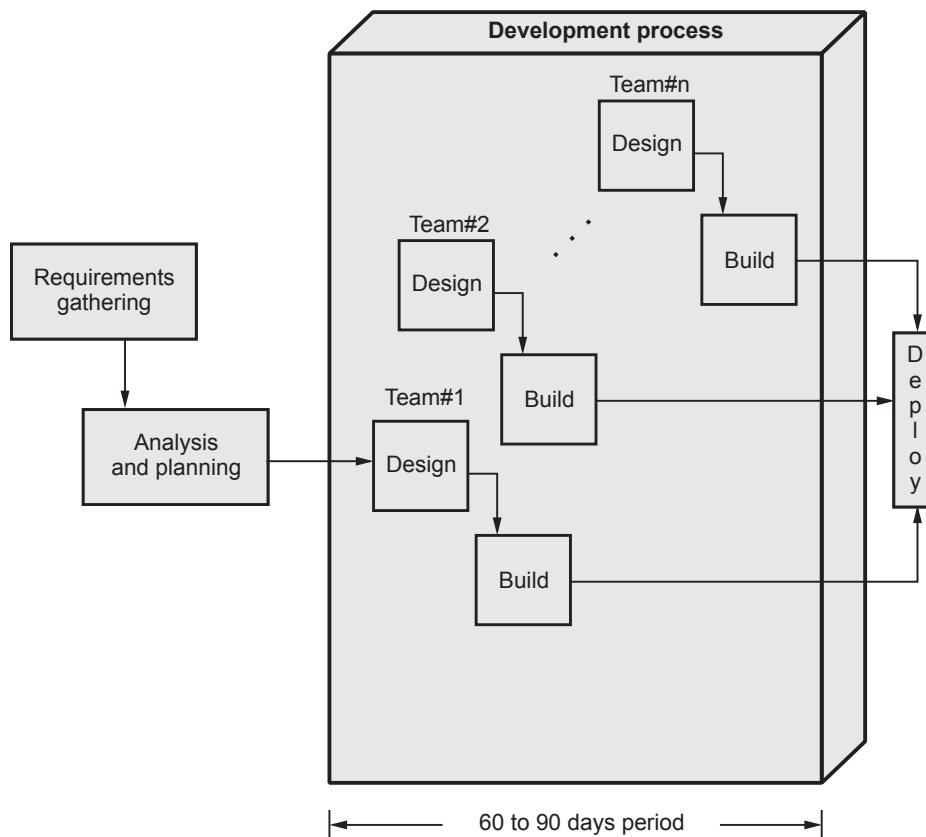


Fig. 1.10.5 Rapid application development

- 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.

Drawbacks of rapid application development

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

Example 1.10.7 Which type of applications suit RAD model ? Justify your answer.

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

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

Example 1.10.8 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.

1.10.4 Comparison between Various Process Models

1. Evolutionary and Incremental Process Model

Sr. No.	Evolutionary process model	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 cannot 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.

2. 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.

3. Waterfall, Spiral, Prototype and Incremental Model

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 .

4. Prototype Model and RAD Model

Sr. No.	Prototype Model	RAD Model
1.	Requirements are gathered initially but there can be change in the requirements in the later stage.	The RAD model is basically an incremental model and requirements are precisely defined and there is no change in the requirements.
2.	All the requirements can't be specified in the early stage of the model.	All the requirements are defined in the early stage of the model.
3.	This model allows the development team that have less experience on similar projects.	This model require that the development team must be well experienced.
4.	The project can be developed with limited number of resources using prototype model.	The RAD model requires heavy number of resources.
5.	The training for the development team is not available for such type of projects.	The people can be trained to work on projects using RAD model.
6.	Requirements can indicate the complexity of the projects.	The requirements can not indicate the complexity of the projects.
7.	Users of the projects that are developed using Prototype model can be novice or less experiences persons.	Users of the projects that are developed using RAD model are experienced persons. They are the persons who have used previously used such type of projects.

Review Questions

1. Explain in brief the spiral model. **GTU : Summer-2011, Winter-2011, Marks 7**
2. Explain in brief the process model which is used in situations where requirements are well defined and stable. (*Hint* : Waterfall model - Refer section 1.10.1). **GTU : Summer-2011, Marks 7**
3. Explain incremental model for system development. Differentiate it with spiral model **GTU : Summer-2012, Marks 7**
4. Explain spiral model and its advantages. Compare prototype model and spiral model. **GTU : Winter-2012, Summer-2014, Marks 7**
5. Explain in detail the process model which is normally suited for the development of large scale software system. (*Hint* : spiral model - Refer section 1.10.2.2) **GTU : Summer-2013, Marks 7**
6. Describe two main features of spiral model and discuss working of prototyping model with its diagram. **GTU : Winter-2014, Marks 7**
7. Discuss incremental process model with its diagram and compare with waterfall model. **GTU : Winter-2014, Summer-2015, Marks 7**

8. Explain prototype model and compare it with waterfall process model.

GTU : Summer-2014, Marks 7

9. Compare prototype and RAD model.

GTU : Summer-2015, 2019, Winter-2017, 2018, Marks 3

10. Explain prototype process model.

GTU : Summer-2016, Marks 4

11. Explain the process model which is used for development of large-scale system.

GTU : Summer-2016, Marks 7

12. Explain the process model which is normally suits for development of large - scale software system.

GTU : Winter-2017, Marks 4

13. Explain spiral model in brief with suitable diagram.

GTU : Winter-2018, Marks 4

14. What is the importance of process model in development of software system ? Explain prototype process model.

GTU : Winter-2018, Marks 7

15. Explain Waterfall process model.

GTU : Summer-19, Winter-2019, Marks 7

1.11 Agile Process Models

GTU : Winter-2017, Marks 4

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.

There are various process models used as agile process model. But the most famous agile process model is extreme Programming.

Review Question

1. Explain agile development in detail.

GTU : Winter 17, Marks 4

1.12 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.

1.13 Product and Process

GTU : Winter-2011, Summer-2014, 2019, Winter-2019, Marks 7

Development of product depends upon the process which is followed during the development.

The comparison between the two is as follows -

Sr. No.	Software process	Software product
1.	Processes 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.	Process may be small in size and possessing limited functionality. It defines framework for effective product generation.	Software product consists of multiple program codes , related documents such as SRS , designing documents, user manuals, test cases and so on.
3.	Generally only one person uses the process, hence there is a lack of user interface .	Good graphical user interface is most required by any software product.
4.	Process is generally developed by process engineers .	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.	The output of process is product.	Product development occurs by following process.
6.	For example : Program developed for parsing the input.	For example : A word processing software.

Review Questions

1. Distinguish between a program and software product.

GTU : Winter-2011, Marks 4

2. What is software engineering ? What is process ? What is product ?

GTU : Summer-2014, Marks 7

3. What is software engineering ? What is process ? What is product ?

GTU : Summer-2019, Marks 3

4. Compare product and process.

GTU : Winter-2019, Marks 4



2

Agile Development

Syllabus

Agility and agile process model, Extreme programming, Other process models of agile development and tools.

Contents

2.1	<i>Agility and Agile Process Model</i>	Summer-2018, 2019,	
	Winter-2018,	Marks 4
2.2	<i>Extreme Programming</i>	Winter-2019,	Marks 7
2.3	<i>Other Process Models of Agile Development</i>	Summer-2016, 2018,	
	Winter-2017, 2018,	Marks 7
2.4	<i>Tools for Agile Process</i>		

2.1 Agility and Agile Process Model

GTU : Summer-2018, 2019, Winter-2018, 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**.

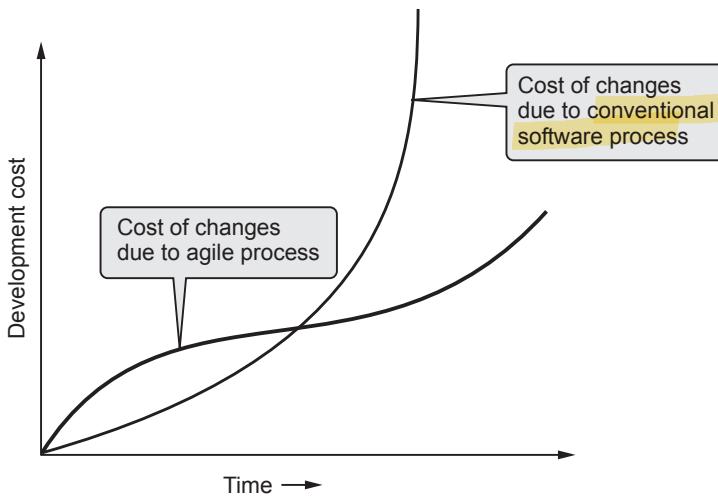


Fig. 2.1.1 Influence of software development approach on agile process

2.1.1 **Agility Principles**

There are famous 12 principles used as **agility principles** -

1. Satisfy the customer by **early** and **continuous delivery** of **valuable software**.
2. The **changes in the requirements** must be **accommodated**. Even though the changes occur **late in** the software development process, the agile process should help to accommodate them.
3. Deliver working **software quite often**. Within the **shorter time span** deliver the working unit.
4. **Business people** and **developers** must work together **throughput** the project.
5. Motivate the people who are building the projects. Provide the **environment** and support to the **development team** and trust them for the job to be done.
6. The working software is the **primary measure** of the **progress** of the software development.
7. The agile software development approach **promote** the **constant** project development. The **constant speed** for the development of the **product** must be maintained.
8. To enhance the **agility** there should be **continuous technical excellence**.
9. The proper attention to be given to technical excellence and good design.
10. The **simplicity** must be maintained **while developing** the project using this approach.
11. The teams must be the **self-organizing team** for getting **best architecture**, requirements and design.

12. At regular intervals the team thinks over the issue of becoming effective. After the careful review the team members adjusts their behavior accordingly.

2.1.2 Concept of Agile Process

Agile process is based on following assumptions about software projects

1. It is difficult to predict the software requirements in advance. Similarly the customer priority often get changed.
2. It is difficult to predict how much design is necessary before the implementation.
3. All the software development activities such as analysis, design, construction and testing are just difficult to predict.

Characteristics of Agile process are -

1. Agile processes must be adaptable to technical and environmental changes. That means if any technological changes occur, then the agile process must accommodate it.
2. The development of agile processes must be incremental. That means, in each development the increment should contain some functionality that can be tested and verified by customer.
3. The customer feedback must be used to create the next increment of the process.
4. The software increment must be delivered in short span of time.
5. It must be iterative, so that each increment can be evaluated regularly.

Review Questions

1. Discuss agility principles used in agility software development.

2. Discuss the concept of agility.

GTU : Summer-2018, 2019, Marks 4

3. List the different agile process model and explain any one with suitable example.

GTU : Winter-2018, Marks 4

2.2 Extreme Programming

GTU : Winter-2019, Marks 7

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

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

2.2.2 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. 2.2.1.

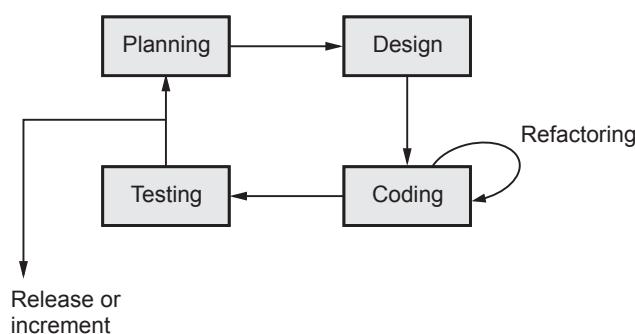


Fig. 2.2.1 XP process

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

	XP Principle	Description
Planning	User story-cards	Instead of creating a large requirement document user stories are written by the customer in which what they need is mentioned.
	Release planning	A release plan for overall project is prepared from which the iteration plan can be prepared for individual iteration.
	Small releases	The developer breaks down the user stories into small releases and a plan for releasing the small functionalities is prepared.
	Iterative process	Divide the development work into small iterations. Keep the iteration of nearly constant length. Iterative development helps in quick or agile development.
	Stand up meetings	The stand up meetings must be conducted for the current outcomes of the project.
Designing	Simple design	Simple design always takes less time than the complex design. It is always good to keep the things simple to meet the current requirements.
	Spike solution	For answering the tough technical problems create the spike solutions. The goal of these solutions should be to reduce the technical risks.
	Refactoring	Refactoring means reductions in the redundancy, elimination of unused functionalities, redesign the obsolete designs. This will improve the quality of the project.
Coding	Customer availability	The most essential requirement of the XP is availability of the customer. In Extreme programming the customer not only helps the developer team but it should be the part of the project.
	Paired programming	All the code to be included in the project must be coded by groups of two people working at the same computer. This will increase the quality of coding.
	Collective code ownership	By having collective code ownership approach the everyone contributes new ideas and not any single person becomes the bottleneck of the project. Anyone can change any line of code to fix a bug or to refactor.

Testing	Unit testing	The test-first development is done in XP. The test framework that contains the automated test case suite is used to test the code. All the code must be tested using unit testing before its release.
	Continuous integration	As soon as one task is finished integrate it into the whole system. Again after such integration unit testing must be conducted.
	No overtime	Working overtime lose the spirit and motivation of the team. Conduct the release planning meeting to change the project scope or to reschedule the project.

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

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

Banking software checking balance, depositing and withdrawing

When the customer wants to **check the balance** then he just enters his account number and the total amount present in his account is displayed to him.

Then customer can **withdraw** some amount from his account. The limit of amount to be withdrawn must be specified. If the customer crosses that limit then he must be warned. The amount withdrawn must be deducted from the available balance.

The customer can **deposit** the desired amount in his account. The deposited amount must be added to the available balance.

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

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

2.2.3 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 -

- Proper environment must be available to support XP.
- Team should contain appropriate and skilled stakeholder
- The organization should support quality programs and continuous improvement.
- The organizational culture should support new values of agile team.

2. Project Community : Skilled and efficient people must be chosen as the agile team members for the success of the project. The team is referred as the **community** when extreme programming approach is considered. The project **community** consists of **technologies, customers, and other stakeholders** who play the vital role for the success of the project. The **role** of the community members must be **explicitly defined**.

3. Project Chartering : Project chartering means **assessing** the justification for the project as a business application. That means, the IXP team assess whether the project satisfies the goals and objectives of the organization.

4. Test Driven management : For assessing the state of the project and its progress the industrial XP needs some **measurable criteria**. In test driven management the project is tested with the help of these measurable criteria.

5. Retrospectives : After delivering the software increment, the **specialized review** is conducted which is called as **retrospective**. The intention of retrospectives is to improve the industrial XP process.

6. Continuous learning : The team members are inspired and encouraged to learn new methods and techniques that can **improve the quality** of the product.

Review Questions

1. Discuss XP values and XP process.
2. Explain Extreme Programming (XP) in detail.

GTU : Winter-2019, Marks 7

2.3 Other Process Models of Agile Development

GTU : Summer-2016, 2018, Winter-2017, 2018, Marks 7

Various other agile process models apart from extreme programming are

1. **Adaptive Software Development**
2. Dynamic System Development Method
3. **Scrum**
4. Feature Driven Development
5. Agile Modeling

Let us discuss them in detail.

2.3.1 Adaptive Software Development (ASD)

- The adaptive software development approach was proposed by Jim Highsmith. This approach is useful in building the complex software systems using iterative approach.
- The focus of this method is on working in collaboration and team self organization.
- The life cycle of ASD consists of three phases of software development and those are -
 - Speculation
 - Collaboration
 - Learning.
 (Refer Fig. 2.3.1)

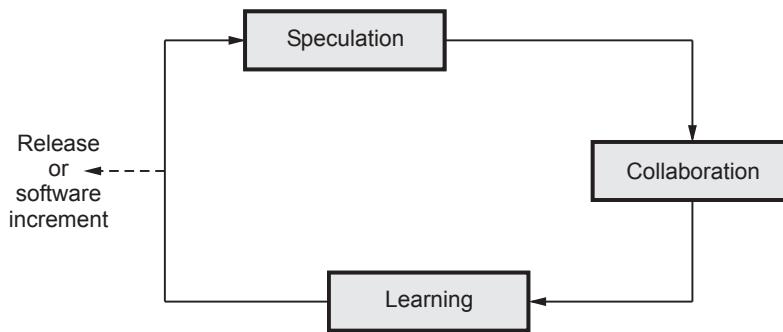


Fig. 2.3.1 Adaptive software development life cycle

- Speculation :** This is an initial phase of the adaptive software development process. In this phase the adaptive cycle planning is conducted. In this cycle planning mainly three types of information is used such as - Customer's mission statement, project constraints (delivery date, user description, budgets and so on) and basic requirements of the project.
- Collaboration :** The motivated people work in collaboration to develop the desired software product. In this phase collaboration among the members of development team is a key factor. For successful collaboration and coordination it is necessary to have following qualities in every individual -
 - Assist each other without resentment
 - Work hard.
 - Posses the required skill set.
 - Communicate problems and help each other to accomplish the given task.
 - Criticize without any hate.

3. Learning : As the team members go on developing the components, the emphasize is on learning new skills and techniques. There are three ways by which the team members learn -

- **Focus groups :** The feedback from the end-users is obtained about the software component being developed. Thus direct feedback about the developed component can be obtained.
- **Formal technical review :** This review for software components is conducted for better quality.
- **Postmortems :** The team analyses its own performance and makes appropriate improvements.

2.3.2 Dynamic System Development Method (DSDM)

In this agile method, the project deadline is met using the the incremental prototyping approach. This is an iterative development process.

The Dynamic System Development Method (DSDM) consortium has defined an agile process model called **DSDM life cycle**.

Various phases in this life cycle model are as follows -

1. **Feasibility study :** By analyzing the business requirements and constraints the viability of thee application is determined in this phase.
2. **Business study :** The functional and informational requirements are identified and then the business value of the application is determined. The basic application architecture is decided in this phase.
3. **Functional model iteration :** The incremental approach is adopted for development. The basic functionalities are demonstrated to the customer by building the suitable increments. The intention of iterative cycle is to gather additional requirements by eliciting the requirements from the customer as each prototype is being developed.
4. **Design and build iteration :** Each prototype is revisited during the functional model iteration to ensure that the business requirements are satisfied by each software component. Sometimes if possible, the design and build activities can be carried out in parallel.
5. **Implementation :** In this phase, the software increment is placed in the working environment. If changes are suggested or if the end-user feels it incomplete then the increment is placed in iteration for further improvement.

The DSDM can be combined with XP method or ASD concepts to create a combination model.

2.3.3 Scrum

- SCRUM is an **agile process model** which is used for developing the complex software systems.
- It is a **lightweight process framework** that can be used to manage and control the software development using **iterative and incremental** approach. Here the term **lightweight** means the **overhead** of the process is kept as **small as possible** in order to **maximize** productive time.
- This model is developed by Jeff Sutherland and Ken Schwaber in 1995.

Principles

- Various **principles** using which the SCRUM works are as given below -
 1. There are **small working teams** on the software development projects. Due to this there is maximum communication and **minimum overhead**.
 2. The tasks of people must be partitioned into small and clean **packets or partitions**.
 3. The process must **accommodate** the technical or business **changes** if they occur.
 4. The process should produce **software increments**. These increments must be inspected, tested, documented and built on.
 5. During the product building the constant **testing and documentation** must be conducted.
 6. The SCRUM process must **produce the working model** of the product whenever demanded or required.
- Various development activities (requirements analysis, design, evolution and delivery) are guided by SCRUM principles.

Development Activities

In SCRUM emphasize is on **software process pattern**. The software process pattern defines a set of development activities. Refer Fig. 2.3.2.

Various development activities in SCRUM are -

1. **Backlog** : It is basically **a list** of **project requirements** or **features** that must be **provided** to the **customer**. The items can be **included** in the **backlog list** at any time. The product manager analyses this list and updates the priorities as per the requirements.
2. **Sprint** : These are the **work units** that are needed to **achieve** the **requirements** mentioned in the backlogs. Typically the sprints have **fixed duration** or **time-box** (typically of **2 to 4 weeks**). Thus sprints allow the team members to work in stable and short-term environment.

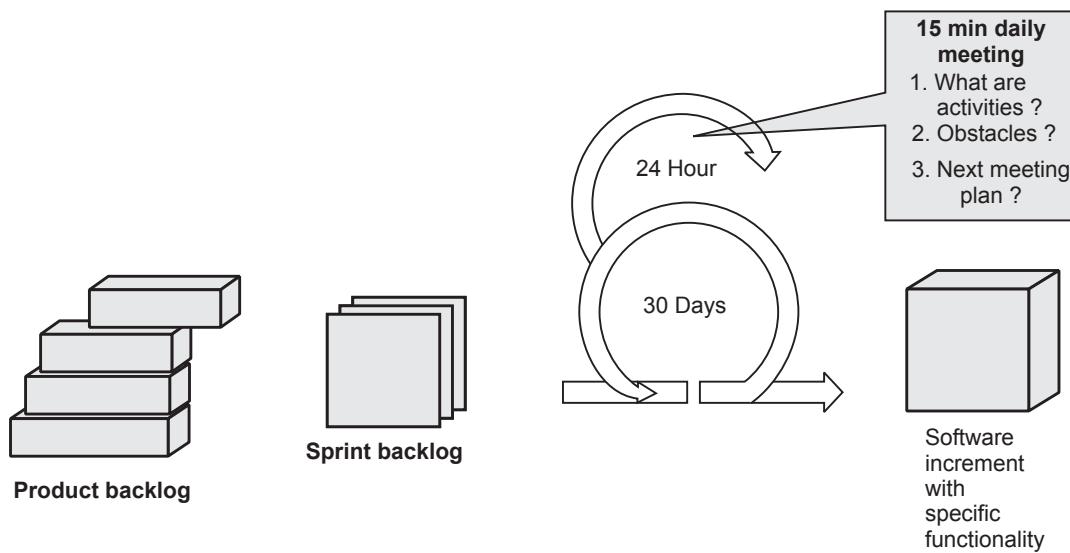


Fig. 2.3.2 SCRUM workflow activities

3. Meetings : These are 15 minutes daily meetings to report the completed activities, obstacles and plan for next activities. Following are three questions that are mainly discussed during the meetings

- i) What are the tasks done since last meeting ?
- ii) What are the issues (obstacles) that team is facing ?
- iii) What are the next activities that are planned ?

4. Demo : During this phase, the software increment is delivered to the customer. The implemented functionality which is demonstrated to the customer. Note that demo focuses on only implemented functionalities and not all the planned functionalities (and yet to get implemented) of the software product.

Roles :

1. **Scrum Master** - The Scrum master leads the meeting and analyses the response of each team member. The potential problems are discussed and solved in the meeting with the help of master.
2. **Team Members** - These are the persons working in a team to develop the software solutions.

Advantages and Disadvantages : important

Advantages :

1. SCRUM model brings transparency in project development status.
2. It provides flexibility towards the changes.
3. There is improved communication, minimum overhead in development process.
4. The productivity can be improved.

Disadvantages :

1. Some decisions are hard to track in fixed time span.
2. There are problems to deal with non-functional requirements of the system.

2.3.4 Feature Driven Development (FDD)

- Originally Peter Coad suggested this approach for object oriented software engineering.
- Stephen Palmer and John Flesing has extended and enhanced Coad's work.
- In FDD, the feature means **client valued function**. It is an iterative and incremental software development process

In FDD, the collaborative activities are carried out. These activities are called as process as shown in Fig. 2.3.3.

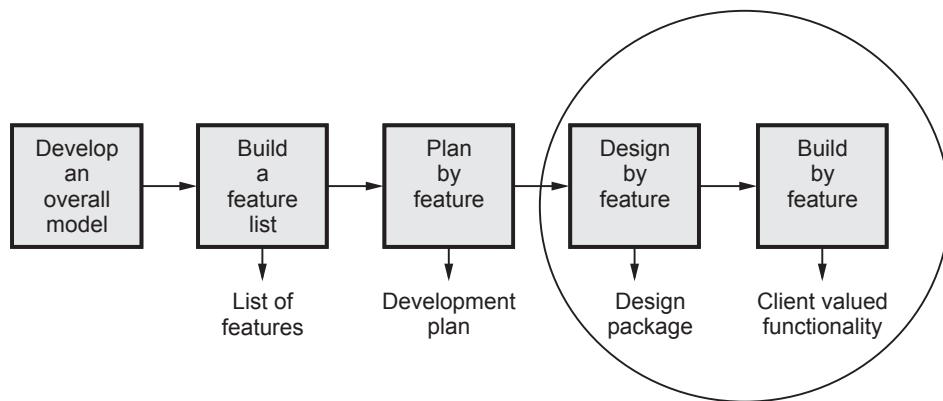


Fig. 2.3.3 Feature driven development life cycle

- Various phases in the FDD life cycle are
 1. **Develop overall model**: In this phase the high-level walkthrough of scope and detailed domain walkthroughs are conducted. Later on peer reviews and discussions are carried out on these walkthroughs and domain area models are created. These domain area models are then merged into the overall models.
 2. **Build features list**: Initially the list of **features** is created. The domain is functionally decomposed into various **subject areas**. These subject areas contain the **business activities**. The steps within business activity forms the categorized feature list. Features are basically the **client valued functions** and can be expressed in the form.

<action> <result> <by|for|of|to> <object>

For example

"Display product-specifications of the product"
↓ ↓ ↓ ↓
<Action> <result> <of> <object>

3. **Plan by feature :** After completing the building of feature list the development plan is created. The features are assigned as **classes** and are chief programmer or the class owner is assigned with appropriate classes.
 4. **Design by feature :** A **design package** was produced for each feature. A chief programmer selects a small group of features and these features are to be developed within two weeks. For each feature the **sequence diagram** is created.
 5. **Build by feature :** Finally a complete client valued function is developed for each feature. The class owners develop the actual code for their classes and this code is promoted to the main build.
- Following are some **benefits of the features** -
 1. Features represent small block of deliverable functionalities hence user can better describe, understand and review them.
 2. The features can be arranged into hierarchical business related grouping.
 3. The team can develop every feature within the two weeks.
 4. The features are typically smaller in size and therefore can be analyzed effectively.
 5. Project planning, scheduling and tracking can be driven by features.

The FDD can be used to develop complex projects or bigger projects. It can also be used for the developing the projects having more novice developers.

2.3.5 Crystal

- Cockburn and Highsmith suggested the **crystal family** of agile methods.
- The primary goal of this method is to deliver useful and working software.
- In this model, a set of methodologies are defined which contains the core elements that are common to all. These methodologies also contain roles, process patterns, work products and practice that are unique to each.
- Thus the crystal family is actually a set of agile processes that are useful for different types of projects. The agile team has to select the members of the crystal family that is most appropriate for their ongoing project and environment.

2.3.6 Agile Modeling (AM)

- The agile modeling can be used for large and complex projects. By adopting agile modeling techniques the scope and complexity of the project can be understood. The problems can be partitioned effectively among the group of people. And the quality of the working model can be assessed at every step of development.
- Scott Ambler described the Agile modeling as - "Agile modeling is a collection of values, principles and practices for modeling the software. Agile Modeling (AM) is a practice-based methodology for effective modeling and documentation of software-based systems. This modeling technique is more effective and light weight than the traditional modeling technique."
- Various **features** suggested by Scott Ambler for agile modeling are as follows -
 1. **Specify the purpose for the model**: Prior to development of the software model the goals and objective of the model must be known to the developer.
 2. **Make use of multiple models**: Various models can be used to gain the insight in the software product. Each model can have different perspective. A small amount of set of models can be useful for building the desired software product.
 3. **Follow a definite path**: Use only those models that give long term value so that the work can proceed in definite direction. Every work product must be maintained as changes occur.
 4. **Give importance to contents and not the presentation**: The correct information in the model is more important than the flawed representation.
 5. **Understand the models and supporting tools**: It is necessary to understand the strengths and weaknesses of the model that are used and the tools that are used in development process.
 6. **Adapt locally**: The needs of modeling team must be satisfied by adopting appropriate modeling approach.

Review Questions

1. Explain the Scrum agile development process.
2. Write a note on - Feature driven development.
3. List the different agile process model and explain any one with suitable example.

GTU : Summer-2016, Marks 7

4. Explain the merits and demerits of SCRUM.

GTU : Winter-2017, Marks 4; Winter-2018, Marks 7

5. Explain adaptive software development process model.

GTU : Summer-2018, Marks 7

2.4 Tools for Agile Process

- The objective of agile development tools is to help in one or more activities in agile development. Due to use of such tools the operational development software can be created rapidly.
 - The agile tool set consists of automated support for project planning, use case development, and requirements gathering, code generation and testing.
 - The project management tools focus on preparing 'Earned value charts' instead of preparing the Gnatt charts.
 - The purpose of agile tools is to enhance the environment.
 - Examples of various agile tools
1. **Actif Extreme** : This tool is developed by Microtool. It provides the support for various technical activities.
 2. **Ideogramic UML** : It is developed by Ideogramic. This tool is used within the agile process.
 3. **Together tool set** : It is distributed by Borland. It provides the support for the technical activities within the XP (Extreme Programming) and other agile processes.



3

Managing Software Project

Syllabus

Software metrics (Process, Product and project metrics), Software project estimations, Software project planning (MS project tool), Project scheduling and tracking, Risk analysis and management (Risk identification, Risk projection, Risk refinement, Risk mitigation).

Contents

3.1	Software Process and Project Metrics	Winter-2017, Summer-2019 Marks 4	
3.2	Software Measurement	Summer-2013, 2014, Winter-2013, 2017,	Marks 7
3.3	Product Metrics		
3.4	Software Project Estimations	Winter-2019,	Marks 7
3.5	Software Project Planning		
3.6	Project Scheduling and Tracking	Summer-2016, Winter-2018, 2019,	Marks 7
3.7	Risk Analysis and Management	Winter-2011, Summer-2013, 2016, 2018, 2019,	Marks 7
3.8	Risk Identification	Summer-2011,	Marks 7
3.9	Risk Projection		
3.10	Risk Refinement		
3.11	Risk Mitigation	Summer-2015, Winter-2018,	Marks 7
3.12	Risk Plan	Winter-2017, 2019,	Marks 7

3.1 Software Process and Project Metrics

GTU : Winter-2017, Summer-2019, Marks 4

Software Process and Project Metrics are Quantitative measures that enable Software engineers to gain insight into the efficacy of the Software Process and the Project.

The software measures are **collected** by **software engineers** and software metrics are **analyzed** by **software managers**.

The work product of software metrics is a set of productivity metrics and quality metrics.

3.1.1 Metrics in Process and Project Improvement process indicators = metrics

Process metrics are the set of process indicators that are used to improve the software processes. Process metrics is collected over the complete software life cycle. The software process can be improved with the help of process metrics. It can be illustrated as below :

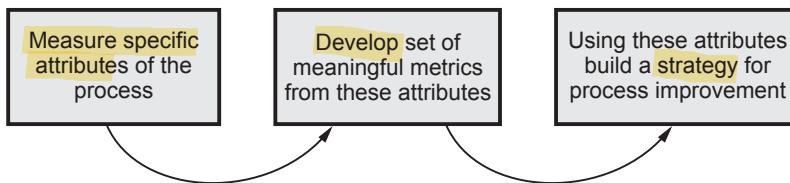


Fig. 3.1.1 Process improvement

In making improvements to any software system, there are three basic quality factors to consider : **product, people, and technology**. These three are the major determinants of software cost, schedule, productivity, and quality.

- The factor **people** includes hiring the best people you can find, motivating them to do the best job, and training them on the skills needed to perform their jobs effectively.
- The **technology** factor includes acquiring and installing tools that help automate. The technology also includes use of new software languages to develop the desired quality product (e.g., Java, Visual C++, Oracle).
- The complexity of the factor **product** has great impact on quality and team performance.

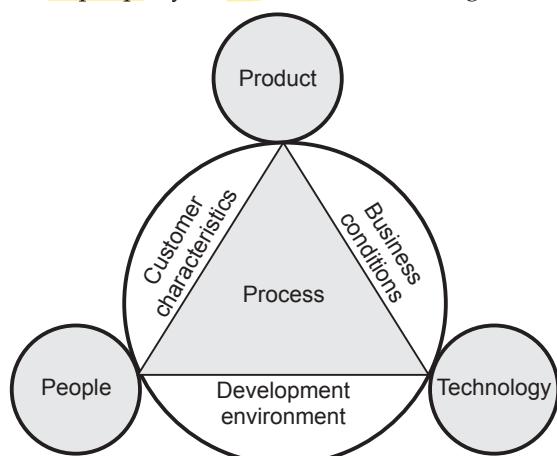


Fig. 3.1.2 Quality factors for improvement

- Out of these three components if one component is altered then it will impact other two factors. For example : If an organization makes use of new testing tool (*technology*) then the staff (*people*) must be trained to work with this tool. The organization must consider if this new tool helps in generating the desired software product.
- This process triangle resides within the circle. This circle specifies the environmental conditions such as :
 - Customer characteristics(communication and collaboration between user and developer).
 - Business conditions(Organizational policies, Business rules)
 - Development environment (use of new technologies, use of automated tools).

Grady suggested some guideline for collecting software metrics. These are known as **Software Metrics Etiquette**. These are as given below -

1. Never use metrics to please the individuals or to threaten the individuals or team.
2. Use common sense and organizational sensitivity while understanding the metrics data.
3. Collectively together with project manager and software team set the goals and use of metrics for the software.
4. The metrics data directs the area of process improvement.
5. Do not focus on single metrics so that other important metrics may get omitted.

3.1.2 W5HH Principle

- Bary Bohem suggested an approach for addressing - project objectives, deciding milestones and schedules, roles and responsibilities, project management, technical approaches and required resources by WWWWWHH principle.
- The W5HH principle is nothing but the series of questions in the form of why, what, when, who, how and so on.
- Answers to these questions lead to definition of key project characteristics. The answers of these questions are also useful in building the resultant project plan.
- Following are those W5HH questions -

- 1) Why is the system being developed ?

Answer to this question assesses the validity of business reasons for the software work. It also justifies the expenditure of people, time and money.

- 2) What will be done ?

Answer to this question will help the software team member to identify the project tasks and milestones.

3) When will be done ?

The answer to this question will help to prepare the project schedule with identified project tasks and milestones.

4) Who is responsible for a function ?

By answering this question, the roles and responsibilities required to develop the system can be defined.

5) Where are they organisationally located ?

All the roles can not be defined within the software team itself. There are customers, users and other stakeholders holding some responsibilities.

6) How to do the job technically with proper management?

Answer to this question will help to establish the technical strategy about the project.

7) How much of each resource required ?

This answer will be useful for deriving the project estimate or cost of the project.

Review Questions

1. Explain 4 P's of effective project management in detail.

GTU : Summer-2019, Marks 4

2. Explain software project management and W5HH principle.

GTU : Winter-2017, Summer-2019, Marks 4

3.2 Software Measurement

GTU : Summer-2013, 2014, Winter-2013, 2017, Marks 7

Software measurement is an ability to measure attributes of software and software development process so that the software engineering activities can be improved.

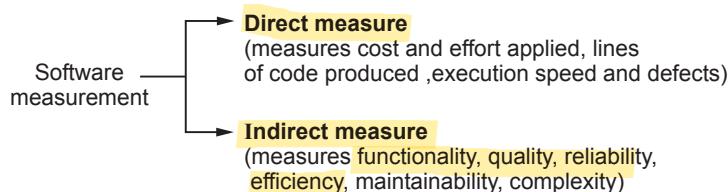


Fig. 3.2.1 Software measurement

The measurement of software can be classified into two categories -

Let us discuss the metrics based on these approaches -

1. Direct Metrics : It refers to immediately measurable attributes. For example - line of code, execution speed.

2. Indirect Metrics : It refers to the aspects that are not immediately quantifiable or measurable. For example-functionality of the program.

3.2.1 Size Oriented Metrics

- 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 3.2.1 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
 - Quality = Number of faults/KLOC
 - Cost = \$/KLOC
 - Documentation = Pages of documentation/KLOC
- The size measure is based on the lines of code computation. The lines of code is defined as one line of text in a source file.
- While counting the lines of code the Simplest Standard is :
 - Don't count blank lines.
 - Don't count comments.
 - Count everything else.
- The size oriented measure is not universally accepted method.

Advantages

1. Artifact of software development which is easily counted.
2. Many existing methods use LOC as a key input.
3. A large body of literature and data based on LOC already exists.

Disadvantages

1. This measure is dependent upon the programming language.
2. This method is well designed but shorter program may get suffered.
3. It does not accommodate non procedural languages.
4. In early stage of development it is difficult to estimate LOC.

3.2.2 Function Oriented Metrics

- The oriented 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. It uses function points.
 - Function points are derived using :
1. Countable measures of the software requirements domain
 2. Assessments of the software complexity.

How to calculate function point ?

- The data for following information domain characteristics are collected :
1. Number of user inputs - Each user input which provides distinct application data to the software is counted.
 2. Number of user outputs - Each user output that provides application data to the user is counted, e.g. screens, reports, error messages.
 3. Number of user inquiries - An on-line input that results in the generation of some immediate software response in the form of an output.
 4. Number of files - Each logical master file, i.e. a logical grouping of data that may be part of a database or a separate file.
 5. Number of external interfaces - All machine-readable interfaces that are used to transmit information to another system are counted.
- The organization needs to develop criteria which determine whether a particular entry is simple, average or complex.
 - The weighting factors should be determined by observations or by experiments.

Domain Characteristics	Count	Weighting factor			Count
		Simple	Average	Complex	
Number of user input	X	3	4	6	
Number of user output	X	4	5	7	
Number of user inquiries	X	3	4	6	

Number of files	X	7	10	15	
Number of external interfaces	X	5	7	10	
Count Total					

- The count table can be computed with the help of above given table.
 - Now the software complexity can be computed by answering following questions. These are **complexity adjustment values**.
1. Does the system need reliable backup and recovery ?
 2. Are data communications required ?
 3. Are there distributed processing functions ?
 4. Is performance of the system critical ?
 5. Will the system run in an existing, heavily utilized operational environment ?
 6. Does the system require on-line data entry ?
 7. Does the on-line data entry require the input transaction to be built over multiple screens or operations ?
 8. Are the master files updated on-line ?
 9. Are the inputs, outputs, files or inquiries complex ?
 10. Is the internal processing complex ?
 11. Is the code which is designed being reusable ?
 12. Are conversion and installation included in the design ?
 13. Is the system designed for multiple installations in different organizations ?
 14. Is the application designed to facilitate change and ease of use by the user ?
- Rate each of the above factors according to the following scale :
 - Function Points (FP) = Count total x $(0.65 + (0.01 \times \text{Sum}(F_i))$



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

Advantages

1. This method is independent of programming languages.
2. It is based on the data which can be obtained in early stage of project .

Disadvantages

- 1) This method is more suitable for business systems and can be developed for that domain.
- 2) Many aspects of this method are not validated.
- 3) The functional point has no significant meaning. It is just a numerical value.

Comparison between size oriented and function oriented metrics

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.

Example 3.2.1 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 (UFP)						233

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

$$\text{FP} = 226.01$$

Hence adjusted function point is **226.01**.

Example 3.2.2 Explain function point analysis method. Compute the function points for the following data set : Inputs = 8, Outputs = 12, Inquiries = 4, Logical files = 41, Interfaces = 1 and $\sum F_i = 41$. **GTU : Summer-2013, Winter-2013, Marks 7**

Solution : The given functional units are

1. Inputs = 8
2. Outputs = 12
3. Inquiries = 4
4. Logical files = 41
5. Interfaces = 1

Assume complexity adjustment factors and weighting factors as average.

$$\begin{aligned}
 \text{UFP} &= 8 \times 4 + 12 \times 5 + 4 \times 4 + 41 \times 10 + 1 \times 7 \\
 &= 32 + 60 + 16 + 410 + 7
 \end{aligned}$$

$$\text{UFP} = 525$$

$$\begin{aligned}
 \text{CAF} &= (0.65 + 0.01 \sum F_i) \\
 &= 0.65 + 0.01 (41) = 1.06
 \end{aligned}$$

$$\text{FP} = \text{UFP} \times \text{CAF} = 525 \times 1.06$$

$$\text{FP} = 556.5$$

3.2.3 Object-Oriented Metrics

Lorenz and Kidd have proposed set of metrics for object oriented projects :

Metrics	Description
Number of scenario scripts	The number scenarios can be typically obtained by designing the use cases for the system. It basically represents the user interaction with the applications. The total number of scenario is dependant upon size of application and number of test cases.
Number of key classes	In any object oriented design classes are the logical entities of the system. Hence total number of key classes ultimately represent the amount of effort required to develop software.
Number of support classes	The support classes are logical entities required to support the key classes. Thus total number of support classes represent amount of software reusability and total amount of reusability.
Average number of support class per key class	There are more number of support class per key class for the applications with GUI than applications without GUI. Knowing this average number makes the design more simplified.
Number of subsystems	Subsystem means aggregation of classes. To schedule the project systematically it is necessary to identify all the subsystem of the system being developed.

3.2.4 Attributes of Effective Software Metrics

The effective software metrics should have following attributes.

- Simple and computable** - The derivation of metric should be easy to compute and should not be a time consuming activity.
- Empirically and intuitively persuasive** - It should be immediate and can be derived based on observations.
- Consistent and objective** - The metric should produce unambiguous results. Anybody should get the same result by using these metrics when same set of information is used.
- Consistent in its use of units and dimensions** - The mathematical units and dimensions used for the metric should be consistent. And there should not be intermixing of units.
- Programming language independent** - The metric should be based on analysis model, design model and program structure. It should be independent of programming languages, syntax or semantic of any programming language.
- Metric should be effective mechanism for high quality feedback** - The metric should provide a way to produce high quality software product.

Review Questions

1. Explain different metrics - Size , functional and complexity
2. Explain software metrics used for software cost estimation.

GTU : Summer-2014, Marks 7

GTU : Winter-2017, Marks 7

3.3 Product Metrics

Product metrics are computed from data collected from analysis and design models, source code and test cases.

Measures, Metrics and Indicators

1. **Measure** : It is a quantitative indication of the extent, amount, dimension, or size of some attribute of a product or process.
2. **Metrics** : It is the degree to which a system, component, or process possesses a given attribute. The software metrics relate several measures. For example - average number of errors found per review.
3. **Indicators** : Indicators mean combination of metrics that provides insight into the software process, project or product.

Attributes of Effective Software Metrics

The effective software metrics should have following attributes.

1. **Simple and computable** - The derivation of metric should be easy to compute and should not be a time consuming activity.
2. **Empirically and intuitively persuasive** - It should be immediate and can be derived based on observations.
3. **Consistent and objective** - The metric should produce unambiguous results. Anybody should get the same result by using these metrics when same set of information is used.
4. **Consistent in its use of units and dimensions** - The mathematical units and dimensions used for the metric should be consistent. And there should not be intermixing of units.
5. **Programming language independent** - The metric should be based on analysis model, design model and program structure. It should be independent of programming languages, syntax or semantic of any programming language.
6. **Metric should be effective mechanism for high quality feedback** - The metric should provide a way to produce high quality software product.

3.3.1 Metrics For Analysis Model

Metrics for the analysis model are useful in estimating the project. In order to determine the metrics in analysis model “size” of the software is used as a measure. The size of software is measured in LOC or FP.

3.3.2 Metrics For Design Model

We need metrics for design model for determining the measurement of design quality. This metric guides the software design activity as design evolves.

3.3.2.1 Architectural Design Complexity

Two scientists Card and Glass has suggested three design complexity measures as

- *Structural complexity*

Structural complexity depends upon the **fan-out** for modules. It can be defined as :

$$S(k) = f_{\text{out}}^2(k)$$

Where f_{out} represents fan-out for module k [fan out means number of modules that are subordinating module k]

- *Data complexity*

Data complexity is the complexity within the interface of internal module. For some module k it can be defined as

$$D(k) = \frac{\text{tot_var}(k)}{[f_{\text{out}}(k)+1]}$$

Where tot_var is total number of input and output variables going to and coming out of the module.

- *System complexity*

System complexity is the combination of structural and data complexity. It can be denoted as

$$Sy(k) = S(k)+D(k)$$

When structural, data and system complexity get increased the overall architectural complexity also gets increased.

3.3.2.2 Metrics for Object Oriented Design

Whitmire has suggested nine measurable characteristics of object oriented design and those are -

- 1) **Size** - It can be measured using following factors. (See Fig. 3.3.1 on next page.)
- 2) **Complexity** - It is a measure representing the characteristics that how the classes are interrelated with each other.

- 3) **Coupling** - It is a measure stating the collaborations between classes or number of messages that can be passed between the objects.
- 4) **Completeness** - It is the measure representing all the requirements of the design component.
- 5) **Cohesion** - It is the degree by which the set of properties that are working together to solve particular property.
- 6) **Sufficiency** - It is the measure representing the necessary requirements of the design component.
- 7) **Primitiveness** - The degree by which the operations are simple. In other words, the measure by which number of operations are independent upon other.
- 8) **Similarity** - The degree to which two or more classes are similar with respect to their functionalities and behaviour.
- 9) **Volatility** - Due to changes in requirements or some other reasons modifications in the design of application may occur. Volatility is a measure that represents the probability of changes that will occur.

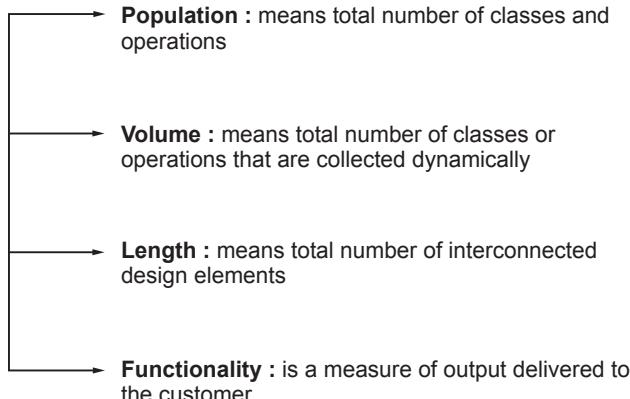


Fig. 3.3.1 Factors that determine the size Metrics For Object Oriented Design

These product metrics for object oriented design is applicable to design as well as analysis model.

3.3.2.3 User Interface Design

There are numerous methods for evaluating user interface. In this section we will discuss two such metrics proposed by different software practitioners.

1. Layout appropriateness

Andrew Sears has suggested *Layout appropriateness* metric for the UI design. The *layout appropriateness* requires description of sequence of the actions performed on layout entities such as icons, menus, windows and so on. In order to accomplish certain task using GUI the user makes transition from one layout entity to another. The *Layout Appropriateness* is a metric used for computing the “cost” of all the transitions made by user.

$$\text{Cost} = \sum_{\text{All transitions in a layout}} [\text{frequency of the transition} * \text{cost of the transition}]$$

2. Cohesion metrics

The cohesion metrics can be defined as the relative connection of on-screen content to other on-screen contents. UI cohesion for screen is high. Kokol has proposed the empirical model for cohesion metrics for UI design.

It is suggested that the effective design of GUI can be done if it is based on user demands and needs. That means designed interface should **satisfy** the user.

3.3.3 Metrics for Source Code

Halstead's complexity measurement was developed to measure a program module's complexity directly from source code, with emphasis on computational complexity.

The Halstead's measures are based on four scalar numbers derived directly from a program's source code :

n_1 is number of distinct operators

n_2 is number of distinct operands

N_1 is total number of operators

N_2 is total number of operands

From these numbers, five measures are derived :

Measure	Symbol	Formula
Program length	N	$N = N_1 + N_2$
Program vocabulary	n	$n = n_1 + n_2$
Volume	V	$V = N * (\log_2 n)$
Difficulty	D	$D = (n_1/2) * (N_2/2)$
Effort	E	$E = D * V$

Halstead's uses certain measures such as program length, program vocabulary, volume, difficulty and effort for the given algorithm. By this Halstead's is trying to show that the program length can be calculated, volume of the algorithm can be estimated. The above given table shows how actually these measures can be obtained.

The Halstead's measures are applicable to operational systems and to development efforts once the code has been written. Thus using Halstead's measurement experimental verification can be performed in software science.

Program length

The length of a program is total usage of operators and operands in the program.

$$\text{Length} = N_1 + N_2$$

Program vocabulary

The program vocabulary is the number of unique operators and operands used in the program.

$$\text{Vocabulary} \cdot n = n_1 + n_2$$

Program volume

The program volume can be defined as minimum number of bits needed to encode the program.

$$V = N \log_2 n$$

Length estimation

$$N = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

Guideline for calculating operands and operators

1. All the variables and constants are considered as operands.
2. Local variables with same name, if occurring in different functions are counted as unique operand.
3. Function calls are considered as operators.
4. The looping statements, do ... while, while, for are operators. The statements if, if ... else are operators. The switch ... case statements are considered as operators.
5. The reserve words, return, default, continue, break, sizeof are all operators.
6. The brackets, commas, semicolons are operators.
7. The unary and binary operators are considered as operators. The & is considered as operator.
8. In arrays, array name and index are considered as operands and [] is considered as operator.
9. All hash directives can be ignored.
10. Comments are not considered.
11. In Goto statement, goto is considered as operator and label as operand.

Example 3.3.1 Obtain Halstead's length and volume measure for following C function.

```
void swap (int a [ ], int i)
{
    int temp ;
    temp = a[i] ;
    a[i] = a[i+1] ;
    a[i+1] = temp ;
}
```

Solution :

We will first find out the operands and operators from above function along with their occurrences.

Operands	Occurrences	Operators	Occurrences
swap	1	()	1
a	5	{ }	1
i	5	void	1
temp	3	int	3
1	2	[]	5
		,	1
		;	4
		=	3
		+	2
n₁ = 5	N₁ = 16	n₂ = 9	N₂ = 21

$$N = N_1 + N_2 = 16 + 21 = 37 \quad n = n_1 + n_2 = 14$$

$$\begin{aligned} \text{Estimated length} &= n_1 \log n_1 + n_2 \log n_2 \\ &= 5 \log 5 + 9 \log 9 \\ &= 5 * 2.32 + 9 * 2.19 \\ &= 11.60 + 19.77 = 31.37 \end{aligned}$$

$$\begin{aligned} \text{Volume} &= N \times \log n \\ &= 37 * \log (14) \\ \text{Volume} &= 37 * 2.63 = 97.64 \end{aligned}$$

3.3.4 Metrics for Testing

Halstead's metrics for estimating the testing efforts are as given below.

The Halstead effort can be defined as

$$e = V/PL$$

Where V is the program volume and PL is the program level. The program level can be computed as

$$PL = 1/[(n_1/2) \times (N_2/n_2)]$$

The percentage of overall testing effort = testing effort of specific module / testing efforts of all the modules

3.3.5 Metrics for Maintenance

The stability of software product is given by an IEEE standard which suggests a metrics software maturity index(SMI) for that matter. It is given as follows -

$$SMI = (M - (A + C + D))/M$$

Where,

M = Number of modules in current version

A = Number of added modules in current version

C = Number of changed modules in current version

D = Number of deleted modules in current version compared to the previous version

When SMI reaches to the value 1.0 the product becomes more and more stabilized. This SMI metrics is used for planning the software maintenance activities.

3.4 Software Project Estimations

GTU : Winter-2019, Marks 7

Software project estimation is just similar to 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).

3.4.1 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**.

3.4.2 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 -
 - These are useful to estimate the **size** each element of software.
 - 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 -
 - Optimistic
 - Most likely
 - Pessimistic

For example, following formula

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

considers for "most likely" estimate where S is the estimation size variable, S_{opt} represents the optimistic estimate, S_m represents the most likely estimate and S_{pess} represents the pessimistic estimate values.

3.4.3 Example of LOC based Estimation

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

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

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

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

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

Then cost for lines of code can be estimated as

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

By considering total estimated LOC as 32620

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

3.4.4 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, discussed in section 3.4.3.

INFO DOMAIN VALUE	Opt.	most likely	pessimistic	est. 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

$$\sum (F_i) \rightarrow 52$$

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

$$FP\ ESTIMATED = (FP\ COUNT\ TOTAL * [COMPLEXITY\ ADJUSTMENT\ FACTOR])$$

$$FP\ ESTIMATED = COUNT\ TOTAL * [0.65 + (0.01 * \sum (F_i))]$$

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

- **FP ESTIMATED = (381 * 1.17) = 446** (Function point count adjusted with complexity adjustment factor)

- A review of historical data indicates -
 1. Average productivity is 6.5 FP/Person month
 2. Average labor cost is \$6000 per month

Calculations for cost per function point, total estimated project cost and total effort

1. The cost per function point = $(6000 / 6.5) = \$923$
2. Total estimated project cost = $(446 * 923) = \$411658$
3. Total estimated effort = $(446 / 6.5) = 69$ Person-month.

3.4.5 Process based Estimation

- This is the most commonly used estimation technique for estimating the project based on the processes used.
- In this technique, the process is decomposed into relatively size set of tasks and the effort required to accomplish each task is estimated.
- The estimation begins with identification of software functions obtained from the project scope. Then a series of software process activities must be performed for each function. The function and software process activities are presented in a tabular form. Then planner estimates the efforts for each software function.

3.4.6 Example of Process based Estimation

Consider the same project described in section 3.4.3. We can estimate it using process based estimation technique by creating a table as shown below. In this technique for each of the business function software engineering tasks such as analysis, design, coding and testing is carried out. The estimated efforts for each of these functions are identified and their sum is obtained. In the following table, the sum of all the rows and columns is taken and the effort for each business function is estimated in percentage.

ACTIVITY	CC	Planning	Risk analysis	Engineering	Construction release	CE	TOTAL
TASK →				Analysis	Code	Test	
				Design			
BUSINESS FUNCTION ↓							
User interface and control facilities(UICF)				0.50	2.75	0.50	4.50
						N A	8.25

2D graphics analysis (2DGA)				0.75	4.50	0.75	2.00	N A	8.00
3D Geometric analysis function (3DGA)				0.50	4.50	1.00	3.50	N A	9.5
Database Management (DBM)				0.50	3.25	1.00	1.00	N A	5.75
Computer graphics display				0.50	3.25	0.70	1.50	N A	5.95
Facility(CGDF)									
Peripheral control function(PCF)				0.25	2.00	0.50	1.50	N A	4.25
Design Analysis Modules (DAM)				0.50	2.00	0.50	2.00	N A	5.00
Total	0.25	0.25	0.25	3.50	22.25	4.95	16.00		48
Percentage effort	1%	1 %	1 %	7 %	46 %	11%	33 %		

CC means customer communication CE means customer evaluation

The percentage effort is calculated based on the total 48.

For instance we get the total for the task **Design** as **22.25**

$22.25 \times 100 / 48 = 46\%$ approximately

Thus the percentage efforts for all the software engineering tasks are computed.

Based on average labor cost of \$6000 per month

1. Total estimated project Effort = **48 persons-month**

2. Total estimated project cost=(6000*48)=\$288000

3.4.7 Estimation with Use-Cases

- Use-cases also provide a software team with insight into software scope and requirements. But there are some difficulties in developing estimations using use cases due to the following reasons
 - There are varieties of ways by which use cases can be described. There is no unique format for them.

- Use-cases represent an **external view** (User's view) of the software. These are also written at different levels of **abstraction**. That means some use cases may be written to expose only primary functions whereas some other use cases may describe each function in more detail.
- The **complexity** of the functions and features cannot be described by use cases.
- The **complex behaviour** involved in many functions and features is also not described in use cases.
- In spite of these difficulties the use cases can be used for estimation, but only if they are considered within the context of **structured hierarchy** that use cases describe.

Smith argues that any level of structured hierarchy -

- 1 Can be described by at the most 10 Use-cases.
- 2 Each use-case would not contain more than 30 distinct scenarios.
- Therefore before using the use cases for the estimation -
 - The **level** within the structured hierarchy must be established.
 - The **average length** of each use case must be determined.
 - The **type of software** application for which estimation is calculated must be defined.
 - The **rough system architecture** is defined.
 - After establishing these characteristics, **empirical data** can be used to estimate LOC or FP for each use case. Then the **structured hierarchical data** can be used to compute the efforts required to develop the system.
 - Following formula is useful to compute this estimation -

$$\text{LOC estimates} = N * \text{LOC}_{\text{avg}} + [(S_a / S_h - 1) + (P_a / P_h - 1)] * \text{LOC}_{\text{adjust}}$$

where

N = Actual number of Use-cases

LOC_{avg} = Historical average LOC / Use-case for this type subsystem

$\text{LOC}_{\text{adjust}}$ = Adjustment based on 'n %' of LOC_{avg}

n = Defined locally and represents the difference between this project and 'Average' project

Sa = Actual scenarios per use-case

Sh = Average scenarios per use-case for this type subsystem

Pa = Actual pages per use-case

Ph = Average pages per use-case for this type of subsystem

The above expression is used to develop a rough estimation of the number of LOC, based on the actual number of use-cases adjusted, by the number of scenarios and page length of the use-cases.

3.4.8 Reconciling Estimates

As we have discussed that estimation can be obtained using LOC, FP, process based and use-case based techniques. We obtain the estimated efforts for

LOC = 68 person-month

FP = 69 person-month

Process based = 48 person-month

That means the ABC project can be completed with range from a low of 48 Person-months to high of 69 Person-months. The Average Estimates is 61 Person-months.

Review Questions

1. What is software project estimation ? How use cases are used in estimation ?
2. What is LOC-based estimation ? Explain with a suitable example.
3. What do you mean by software project estimation ? Explain the process based estimation with an example.
4. Explain software metrics used for software cost estimation.

GTU : Winter-2019, Marks 7

3.5 Software Project Planning

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 resources, 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.

3.5.1 MS Project Planning Tool

- Microsoft has offered a project management tool named Microsoft Project for project planning activities.
- It helps the project manager in
 - i) Developing the project plan
 - ii) Assigning the resources to the tasks
 - iii) Tracking the progress
 - iv) Managing budgets
 - v) Analysis of workload

Features

Various features of MS planning tool are

1. It generates variety of reports and templates with industry standard.
2. Themes can be customized and some elements can be removed.
3. The date can be set for the tasks to accomplish.
4. It has got the compatibility with other programs such as Microsoft Word, Excel, Visio and so on.
5. It can support for the cloud services to share the work among the geographically distant developers.

3.6 Project Scheduling and Tracking

GTU : Summer-2016, Winter-2018, 2019, Marks 7

3.6.1 Project Scheduling Process

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.

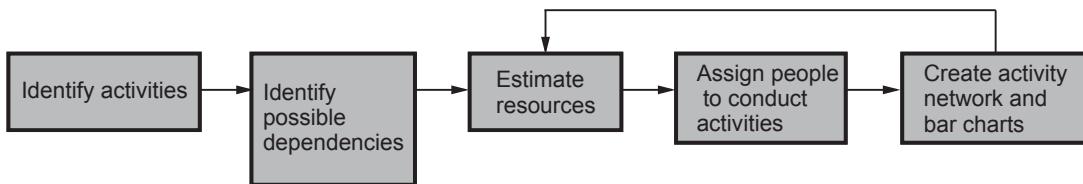


Fig. 3.6.1 Project scheduling process

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 :

- People may leave or remain absent.
- Hardware may get failed.
- 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 -

- Human effort
- Sufficient disk space on server
- Specialized hardware
- Software technology
- 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.

3.6.2 Defining a Task Set for the Software Project

- **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.

3.6.3 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. 3.6.2.

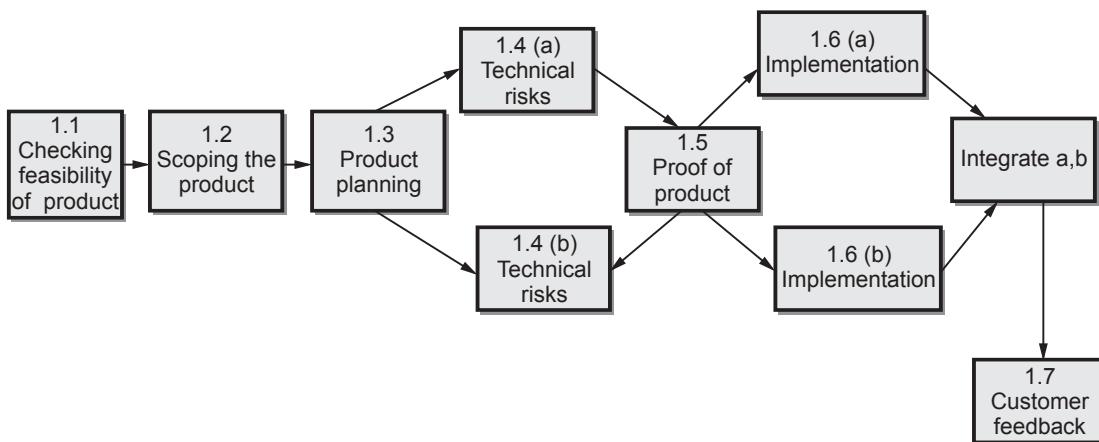


Fig. 3.6.2 Task network

- 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.

3.6.4 Time Line Chart (Gantt 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 Gantt 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

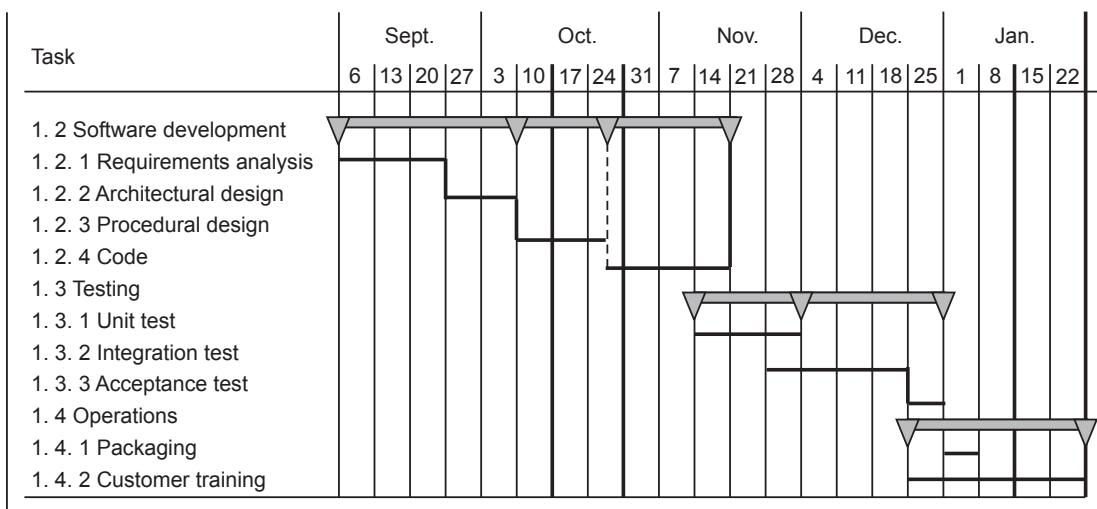


Fig. 3.6.3 Time line chart

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 3.6.1 Project table

3.6.5 Tracking the 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 **various ways** by which tracking of the project schedule can be done -

1. Conduct periodic meetings. In this meeting various problems related to the project get discussed. The progress of the project is reported to the project manager.
2. Evaluate results of all the project reviews.
3. Compare 'actual start date' and 'scheduled start date' of each of the project task.
4. Determine if the milestones of the project is achieved on scheduled date.
5. Meet informally the software practitioners. This will help the project manager to solve many problems. This meeting will also be helpful for assessing the project progress.
6. Assess the progress of the project quantitatively.

Thus for tracking the schedule of the project the project manager should be an experienced person. In fact project manager is the only responsible person who is controlling the software project. When some problems occur in the project then addition resources may be demanded, skilled and experienced staff may be employed or project schedule can be redefined. For handling the severe deadlines, project manager uses a technique of **time boxing**. In this technique each it is understood that the complete product cannot be delivered on given time. Part by part i.e. in the series of increments the product can be delivered to the customer. The project manager uses time box technique means he is associating each task with a box. That means each task is put in a "time box" and within that time frame each task must be completed. When the current task reaches to boundary of its time box, then the next task must be started (even if current task is remaining incomplete). Some researchers had argued upon - leaving the task incomplete when current task reaches to the boundary 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.

Review Questions

1. What is project scheduling ? What are the basic principles of project scheduling ?
2. What is a task network in project scheduling ? Explain with an example ?
3. What is a task network ? How it is used in scheduling of software project ?

4. What is time line chart ? How it is used in scheduling of software project ?
5. Discuss the techniques used in tracking the project schedule.
6. Explain project scheduling and tracking with suitable example. **GTU : Summer-2016, Marks 7**
7. Explain project scheduling process. Also explain Gantt chart in detail.

GTU : Winter-2018, 2019, Marks 7

3.7 Risk Analysis and Management

GTU : Winter-2011, Summer-2013, 2016, 2018, 2019, Marks 7

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 threaten 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. |

3.7.1 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 get 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.

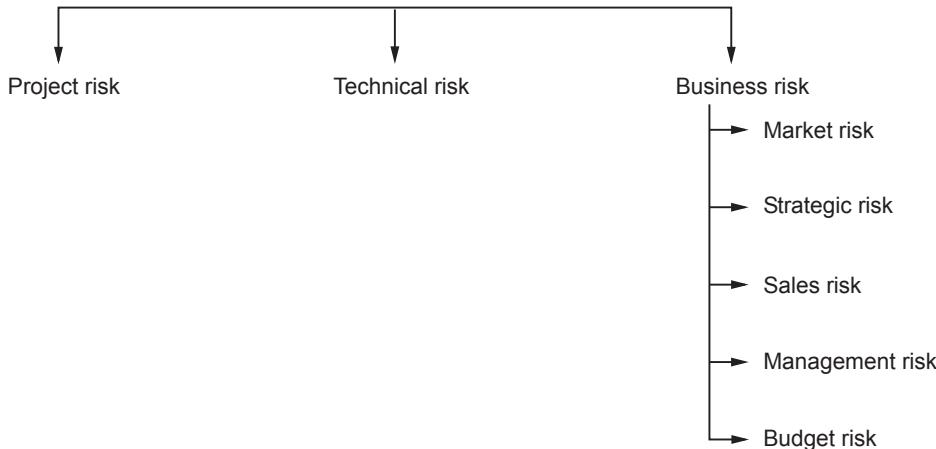


Fig. 3.7.1 Categorization of risk

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.

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

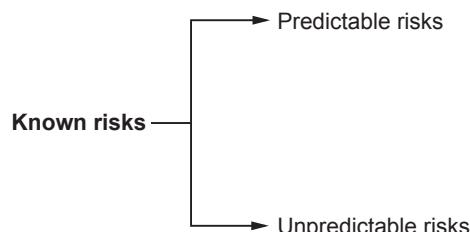


Fig. 3.7.2

3.7.2 Reactive Vs. Proactive Risk Strategies

Reactive and proactive risk strategies are the approaches used for managing the risks.

Reactive risk strategy

- Reactive risk management is a risk management strategy in which when project gets into trouble then only corrective action is taken. But when such risks can not

be managed and new risks come up one after the other, the software team flies into action in an attempt to correct problems rapidly. These activities are called “firefighting” activities.

- Resources are utilized to manage such risks. And if still the risks do not get managed then project is in danger.
- In this strategy no preventive care is taken about the risks. They are handled only on their occurrences.
- This is an older approach of risk management.

Proactive risk strategy

- Proactive risk management strategy begins before the technical activity by considering the probable risk.
- In this strategy potential risks are identified first then their probability and impact is analyzed. Such risks are then specified according to their priorities (i.e. high priority risks should be managed first!). Finally the software team prepares a plan for managing these risks.
- The objective of this strategy is to avoid the risks(*prevention is better than cure!!!*). But it is not possible to avoid all the risks, hence team prepares the risk management plan in such a manner that risk controlling can done efficiently.
- This is an intelligent strategy for risk management and now a day it is used by most of the IT industries.

Review Questions

1. What are the types of risks ? Explain in brief.
2. What do you mean by risk ? What is software risk? Explain all type of software risk ?

GTU : Winter-2011, Marks 7

3. Write short note on risk management.
4. Explain risk management.
5. Enlist and discuss the types of risks.
6. Explain risk management.

GTU : Summer-2013, Marks 7

GTU : Summer-2016, Marks 7

GTU : Summer-2018, Marks 3

GTU : Summer-2019, Marks 7

3.8 Risk Identification

GTU : Summer-2011, Marks 7

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.

3.8.1 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. 3.8.2

Review Question

1. *Describe difference between risk components and risk drivers.*

GTU : Summer-2011, Marks 7

3.9 Risk Projection

The risk projection is also called **risk estimation**.

There are two ways by which risk can be rated

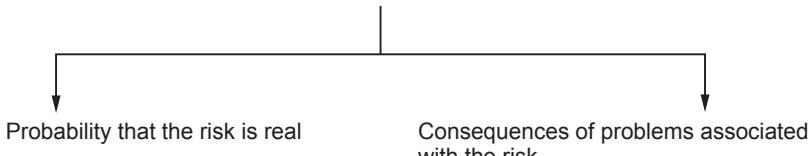


Fig. 3.9.1

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.

3.9.1 Building Risk Table

- 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.

3.9.2 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

Risk Exposure = Probability of occurrence of risk × 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 ,

First of all we will compute

$$\begin{aligned} \text{cost} &= \text{Number of components} * \text{LOC} * \text{cost of each LOC} \\ &= 15 * 500 * 10 = \$75000 \end{aligned}$$

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

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.

Review Question

- How risk projection is carried out risk table ?

3.10 Risk Refinement

Risk refinement is a process of specifying the risk in more detail. The risk refinement can be represented using **CTC format** suggested by **D.P.Gluch**.

The CTC stands for *condition-transition-consequence*. The condition is first stated and then based on this condition sub conditions can be derived. Then determine the effects of these sub conditions in order to refine the risk. This refinement helps in exposing the underlying risks. This approach makes it easier for the project manager to analyze the risk in greater detail.

3.11 Risk Mitigation

GTU : Summer-2015, Winter-2018, Marks 7

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

- Risk avoidance**
- Risk monitoring**
- 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**.

- Communicate with the **concerned staff** to find of probable risk.
- 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.

Review Questions

1. Explain Risk Management, Monitoring and Mitigation.
2. Discuss RMMM.

GTU : Summer-2015, Marks 7

GTU : Winter-2018, Marks 4

3.12 Risk Plan

GTU : Winter-2017, 2019, Marks 7

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. What is risk management ? Explain RMMM plan.
2. Explain RMMM.

GTU : Winter-2017, Marks 7

GTU : Winter-2019, Marks 4



Notes

4

Requirements Analysis and Specification

Syllabus

Understanding the requirement, Requirement modeling, Requirement specification (SRS), Requirement analysis and requirement elicitation, Requirement engineering.

Contents

4.1	<i>Introduction</i>	Summer-2016, · · · · · Marks 7
4.2	<i>Requirement Engineering Task</i>	Winter-2017, Summer-2012, 2018, 2019, · Marks 7
4.3	<i>Initiating Requirements Engineering Process</i>		
4.4	<i>Eliciting Requirements</i>	Winter-2013, Summer-2018, Marks 3
4.5	<i>Developing Use Cases</i>		
4.6	<i>Negotiating Requirements</i>		
4.7	<i>Validating Requirements</i>	Summer-2011, · · · · · Marks 7
4.8	<i>Prioritizing Requirements</i>		
4.9	<i>Building Requirements Analysis Model</i>		
4.10	<i>Elements of Requirements Analysis.</i>	Summer-2015, · · · · · Marks 7
4.11	<i>Scenario Based Modeling.</i>	Summer-2016, Winter-2017, Marks 7
4.12	<i>Class Based Modeling</i>	Summer-2016, · · · · · Marks 7
4.13	<i>Data Modeling</i>	Summer-2012, 2014, 2015, Winter-2019, · · · · · Marks 7
4.14	<i>Flow Oriented Modeling</i>	Summer-2012, Winter-2011, Marks 7
4.15	<i>Behavioral Modeling</i>	Summer-2012, · · · · · Marks 7
4.16	<i>Software Requirements Specification(SRS)</i>	Winter-2011, Winter-2017, · Marks 7

4.1 Introduction

GTU : Summer-2016, Marks 7

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

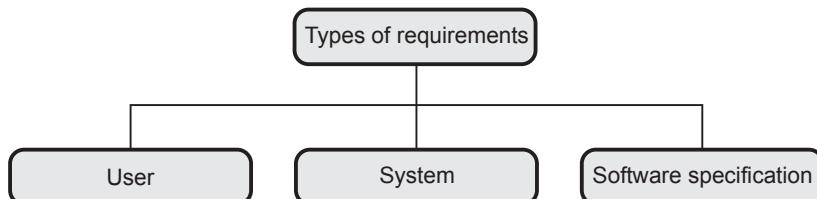


Fig. 4.1.1 Types of requirements

- **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.

Functional and Non functional Requirements

Functional requirements are the type of requirements that describe all the required functionality or system services. Functional requirements for library management system are.
ex.

- i) The user is able to search for the desired book or magazine.
- ii) The unique-id should be provided for every order.
- iii) The user can download or print the required article for personal study.

Now functional requirements are the type of requirements that describe system properties and constraints.

In short non functional requirements arise through.

- i) User needs ii) Budget constraints iii) Organizational policies iv) Safety regulations

Non functional requirements for library management system are

- i) The user interface should respond within 20 seconds.
- ii) The user interface should be according to some international standards.
- iii) The user who wishes to read the article online must be authenticated first.

4.1.1 Need for Requirements to be Stable and Correct

Following are those reasons which specify that there is a need for the requirements to be stable and correct -

As requirements are identified and analysis model is created the software team and other stakeholders negotiate the priority, availability and relative cost of the requirement. During this negotiation there are chances that the requirements may get changed. During requirement analysis, each requirement is validated against the needs of the customer. If the requirements are stable, correct and unambiguous then they remain unchanged throughout the requirement analysis process. And finally the working model that satisfied customers need can be created effectively.

Review Questions

1. What is requirement engineering ? Why requirements must be stable and correct ?
2. What is requirement engineering ? List the functional and non functional requirement for library management system.

GTU : Summer-2016, Marks 7

4.2 Requirement Engineering Task

GTU : Winter-2017, Summer-2012, 2018, 2019, Marks 7

- Requirement engineering is the process characterized for achieving following goals -
 - Understanding customer requirements and their needs
 - Analyzing the feasibility of the requirement
 - Negotiating the reasonable solutions
 - Specification of an unambiguous solution.

- Managing all the requirements of the project
- Finally transforming the requirements into the operational systems
- Requirement engineering process performs following **seven distinct functions** -

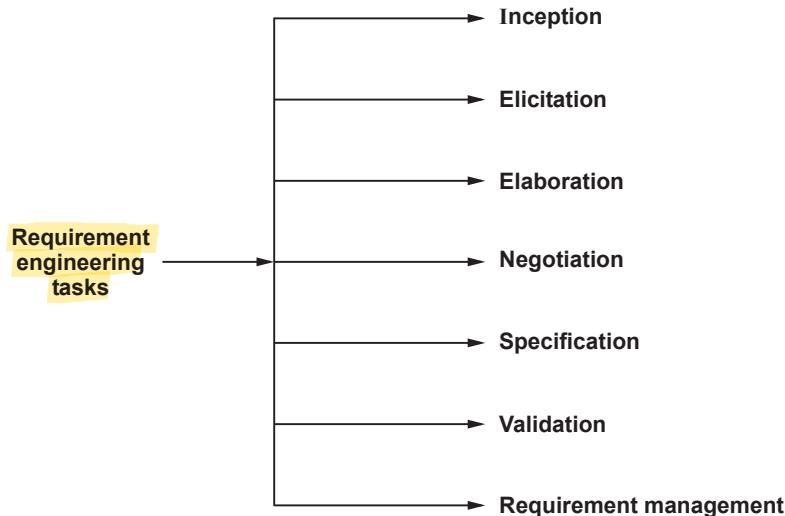


Fig. 4.2.1 Requirement engineering tasks

Let us now discuss these tasks in detail -

4.2.1 Inception specify beginning of the s/w project

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** and then the specific software project needs to be developed.

There exist several **stakeholders** who define the **business ideas**. Stakeholders mean an **entity** that takes active participation in **project development**. In software project development, the stakeholders that are responsible for defining the **ideas** are **business managers, marketing people, product managers** and so on. Their role is to do rough feasibility study and to **identify the scope of the project**.

During the **inception** a set of **context free questions** is discussed. 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**.

4.2.2 Elicitation requirement discovery

Before the requirements can be analyzed and modelled they must undergo through the process of elicitation process. 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 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**.

4.2.3 Elaboration

- Elaboration is an activity in which the information about the **requirements** is **expanded** and **refined**. This information is **gained** during **inception** and **elicitation**.
- 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**. Basically these scenarios describe how **end-user** will **interact** with the system.
- 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. Thus various UML (Unified Modelling Diagrams) are developed during this task.
- Finally the **analysis model** gets developed during the **elaboration phase**.

4.2.4 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**. For that purpose, requirement engineers must ask the customers and stakeholders to **rank** their **requirements** and then priority of **these requirements** is decided. Using **iterative approach** some requirements are **eliminated**, **combined** or **modified**. This process continues until the users' **satisfaction** is achieved.

4.2.5 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.
- For a large system it is always better to develop the specification using **natural language** and in a written **document form**. The use of **graphical models** is more useful for specifying the requirements.
- Specification is the final work product of requirement engineering process. It describes the functions, constraints and performance of computer based systems.

4.2.6 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.
- The most commonly **used** requirement validation mechanism is **formal technical review (FTR)**. In FTR, the review team **validates** the software **requirements**. The review team consists of **requirement engineers**, **customers**, **end users**, **marketing person** and so on. This review team basically identifies **conflicting requirements**, **inconsistencies** or **unrealistic requirements**.

4.2.7 Requirement Management

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.

Requirement management process

Following things should be planned during requirement process.

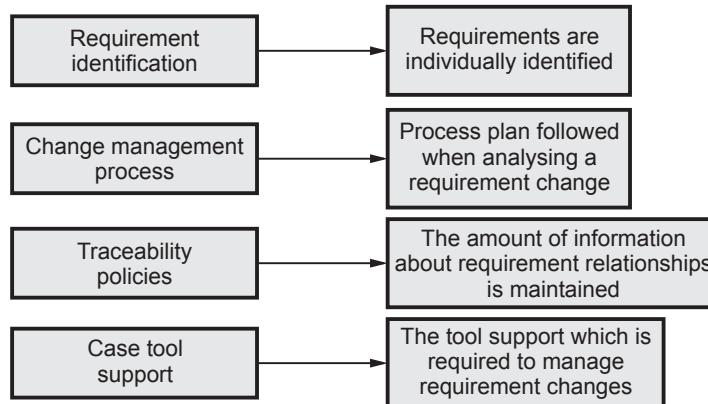


Fig. 4.2.2 Planning in requirement management process

- Traceability is concerned with relationship between requirements, their sources and the system design.

Various types of traceability are

1. **Source traceability** - These are basically the links from requirement to stakeholders who propose these requirements.
 2. **Requirements traceability** - These are the links between dependant requirements.
 3. **Design traceability** - These are the links from requirements to design.
- Case tool support is required for
 1. Requirement storage
 2. Change management
 3. Traceability management
 - Traceability information is typically represented by a data structure **Traceability matrix**. If one requirement is dependent 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.

Review Questions

1. Why requirements elicitation is difficult ? What is meant by requirement negotiation ?
2. List and explain requirements engineering tasks. **GTU : Summer-2012, 2018, Marks 7**
3. Write a short note on requirement engineering. **GTU : Winter-2017, Summer-2019, Marks 7**

4.3 Initiating Requirements Engineering Process

To initiate requirement engineering process meaningful conversation between customer and software engineers must be held. There are various steps that are required to initiate the requirements engineering. These steps are -

- Identification of stakeholders
- Recognizing multiple viewpoints
- Working towards the collaboration
- Questioning

Let us discuss these steps in details -

4.3.1 Identification of Stakeholders

Definition of stakeholder : Stakeholder means anyone who gets benefited from the system in a direct or indirect way when a system is getting developed.

For example : Consultant, product engineers, software developer, customer, marketing people, end user, support and maintenance engineer and others.

Every stakeholder has different view for the system and everyone gets benefited differently when the system gets developed successfully.

During the **requirement inception** phase, the requirement engineers must create a list of stakeholders who will participate in project development process and their contribution towards requirement elicitation. This list may get increased as the project progresses.

4.3.2 Recognizing Multiple Viewpoints

In the project development process there are variety of stakeholders that get involved in. Hence a project has multiple viewpoint. Normally the viewpoint is influenced by the kind of stakeholder who is participating in the project.

Stakeholder	Viewpoint
End user	The end users expect that the developed system should be easy to learn and the features included in the project should be familiar to them.
Marketing people	These stakeholders expect that the project should possess some exciting feature so that market can be attracted towards the product and then selling of the product should become easy.
Business managers	The system getting developed should be within the specified budget and it should satisfy the demands of the market.
Software developers	These stakeholders expect that the requirements which they get should be realistic and unambiguous so that systematic infrastructure of the project can be built.
Support engineers	The support engineers expect that the project should get developed in such a manner that it can be easily maintained.

All these stakeholders contribute a lot in requirement engineering process. The information gathered using different view points can be conflicting or may be inconsistent. Hence the requirement engineers **categorize** all stakeholders' information in a systematic manner so that **decision makers** can choose proper set of requirements.

Example 4.3.1 *Describe two real life situations in which the customer and the end-user is the same. Describe two situations in which they are different.*

Solution : Two situations in which the customer and the end user is the same person

1. For an anti-virus product the customer and the end-user might be the same person. According to his needs (such as single user, with internet security, anti-spam and so on) the customer develops the antivirus software product and the same person makes use of it.
2. The Management Information System(MIS) product can be developed by the customer according to his needs and requirements and can be used by the same person.

Two situations in which the customer and the end user is different

1. An on-line shopping application can be purchased by some customer and it can be used by different group of people.
2. An on-line Banking system is purchased by the bank and is used by its account holders. In this case the customer and the end-user is not the same person.

4.3.3 Working towards the Collaboration

For building a successful system, the **collaboration** among the **stakeholders** and **software engineers** is must.

The job of requirement engineers is to identify the areas of commonality, inconsistencies and conflicts. The commonality denotes the requirements on which all the stakeholders agree. These identified requirements are then categorized. During collaboration process different stakeholders can present their own viewpoint, but **business manager** or **senior technical person** makes the **final decision** about the requirements which should be eliminated.

4.3.4 Questioning

During the project inception phase variety of **context free questions** are asked. For example

Set of questions for identifying the stakeholders

1. Who will use the system getting developed?
2. Who is requesting for this system?
3. Is there another way/source of getting the solution?
4. What will be the economical benefit from the system after its successful completion?

These questions will help to identify the stakeholders that are interested in getting the system developed.

Set of questions used to gain preliminary understanding of the system

1. What are the characteristics of **good** output that would be generated by successful solution ?
2. What kind of problems will be projected by the solution ?
3. Can anybody describe the business domain in which the solution will be used ?
4. Is there any special performance issue or constraint that affects the success of the system getting developed ?

These questions will help to gain clear understanding of the system and associated problems.

Set of questions for performing effective communication

1. Are you an authentic person to answer the questions ?
2. Are your answers official ?
3. Am I asking too many questions ?
4. Is there any alternative source of getting answers of my questions ?
5. Are my questions relevant to the problems ?
6. Am I asking anything else ?

This questions are useful to initiate the communication and are essential for successful elicitation.

4.4 Eliciting Requirements

GTU : Winter-2013, Summer-2018, Marks 3

Questioning is useful only at **inception** of the project but for **detailed requirement elicitation** it is **not sufficient**. During requirement elicitation certain activities such as problem solving, **elaboration**, **negotiation** and **specification** must be carried out. Various ways by which the **requirement elicitation** can be done are -

1. **Collaborative requirement gathering**
2. **Quality function deployment**
3. **Use scenarios**
4. **Elicitation work product**

Let us discuss these aspects of requirement elicitation in detail -

4.4.1 Collaborative Requirements Gathering

Collaborative requirement gathering is done using collaborative, team-oriented approach.

Facility Application Specification Technique (FAST) is an approach in which **joint team of customers and developers** work together to identify the problem, propose elements of solution, negotiate different approaches and prepare a specification for preliminary set of solution requirements.

Guideline for FAST approach -

1. A meeting should be conducted and attended by both software engineers and customers. The place of meeting should be a neutral site.
2. Rules for preparation and participation must be prepared.

3. An agenda should be prepared in such a way that it covers all the important point as well as it allows all the new innovative ideas.
 4. A facilitator controls the meeting. He could be customer, developer or outsider.
 5. A definition mechanism is used. The mechanism can be work sheets, flip charts, wall stickers, electronic bulletin board, chart room, virtual forum.
 6. The goal is to identify the problem, decide the elements of solution, negotiate different approaches and specify the preliminary set of solution requirements.
- In FAST meeting each FAST attendee is asked to prepare - a list of objects, list of services and a list of constraints.
 - The list of objects consists of all the objects used in the system, the objects that are produced by the system and the objects that surround the system.
 - The list of services contain all the required functionalities that manipulate or interact with the objects.
 - The list of constraints consists of all the constraints of the system such as cost, rules, memory requirement, speed accuracy etc.
 - As the FAST meeting begins, the very first issue of discussion is the need and justification for the new product. Once everyone agrees upon the fact that the product is justified, each participant has to present his lists.
 - These lists are then discussed, manipulated and these modified or refined lists are combined by a group.
 - The combined list eliminates redundant entries adds new ideas that come up during the discussion. The combined list is refined in such a way that it helps in building the system.
 - The combined list should be prepared in such a way that a “consensus lists” can be prepared, for object, services and constraints.
 - A team is divided into subteams. Each subteam develops a minispecification from each consensus list.
 - Finally a complete draft specification is developed.

For example -

A FAST team is working on a commercial product. A following product description is given as below -

“Nowadays the market for video game is growing rapidly. We would like to enter this market with more features, like attractive GUI, multiple sound setting, realistic (3D) animations. This product is tentatively called ‘Gamefun’. At the end of game, scores of each player should be displayed”.

The FAST attendee prepare following lists -

- 1) List of objects - Display, menu, a sound, an event (moving from one level to another) and so on.
- 2) List of services - Setting sounds, setting colors in GUI, HELP, instructions for players, score card etc.
- 3) List of constraints - Must be user friendly, must have high speed, must accommodate less size, should have less cost.

The for Menu (object) can be as given below -

- Contains 'Start game' and 'exit' options.
- List of all functional keys with corresponding functionality.
- Software provides interaction guidance, quick tour, sound controls.
- All players will play or interact through keys.
- Software provides facility for change in the look of GUI.
- Software displays scores of each player.

4.4.2 Quality Function Deployment

Quality function deployment is a quality management technique which translates the customer needs and wants into technical requirements. This technique was introduced in Japan.

Under quality function deployment three types of requirements can be defined -

Normal requirements -

The requirements as per goals and objectives of the system are called normal requirements. These requirements can be easily identified during the meeting with the customer.

Types of requirements under QFD



For example : Handling mouse and keyboard events for any GUI based system.

Expected requirements -

These types of requirements are such requirements which system must be having even if customer did not mention about them. These are such requirements that if they are not present then the system will be meaningless.

For example : A software package for presentation (like Microsoft Power Point) must have option of 'new slide insert', so that user will be able to insert a new slide at any position during his presentation.

Exciting requirements -

When certain requirements are satisfied by the software beyond customer's expectations then such requirements are called exciting requirements.

For example : Spell check facility in Microsoft Word is an exciting requirement. Various types of deployments that can be conducted during software development process are -

Function deployment : For determining the value of each function this deployment can be done.

Information deployment : After identifying various functionalities events and data objects must be identified.

Task deployment : The task associated with each function must be identified.

Value analysis : Identify the priorities of requirements. The technique of QFD requires proper interaction with customer.

4.4.3 Usage Scenarios

- During requirement gathering overall vision for systems **functions and features** get developed.
- In order to understand how these functions and features are used by different classes of end users, developers and users create a set of scenarios. This set identifies the usefulness of the system to be constructed. This set is normally called as **use-cases**.
- The use-cases provide a description of how the system will be used.

4.4.4 Elicitation Work Product

Following are some work products that get produced during requirement elicitation -

- A statement of **feasibility study** performed in order to find the need of the project.
- Statement for the scope of the system.
- A list of various stakeholders such as customer, end-users, technical persons, and many others who participate during requirement elicitation.
- A technical description of system environment
- A list of requirements and constraints.
- A set of usage scenarios along with operating conditions.
- The prototype that may get developed for defining the requirements in better manner.

These work products are then reviewed by all the people who participate in requirement elicitation.

Example 4.4.1 List five requirement of Library management system.

GTU : Winter-2013, Marks 3

Solution : Requirements of library management system :

1. The system should record all the details of students, staff and books.
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 administrator to make the modification in the student/ staff or book information.
6. The system should allow the administrator to create an account for the new student.
7. The system should display the availability of book or magazine.

Example 4.4.2 Write functional and non-functional requirements of hotel management system.

GTU : Summer-2018, Marks 4

Solution : Functional requirements :

- 1) The system will record customer's personal information such as (first name, last name, occupants, room number, rate, no. of days of stay and so on).
- 2) The system will record expected and actual check - in and check - out data and time.
- 3) The system will keep track all meals purchased in the hotel.
- 4) The system should record payment and payment type.

Non - functional requirements :

- 1) The user interface screen should not take more than 2 seconds to load.
- 2) The login information must be verified within 5 seconds.
- 3) The system should run as a stand alone system on windows operating system.
- 4) Customer Service representative and Managers should be able to login the Hotel management system.

Review Question

1. Explain the requirements elicitation work products ?

4.5 Developing Use Cases

- Use cases are fundamental units of modelling language in which **functionalities** are distinctly represented.

- Use-case depicts the software or system from **end-user's point of view**.
- The first step in writing use cases is to identify **actors**. Actors are entities that use the system or product within the context of function and behaviour of the system. Actors **represent the role** of the people as the system gets operated. Actor is anything that **communicates** with the system or product. It is **external** to the system. Every actor has one or more goals when using the system.
- Requirement elicitation is an evolutionary activity. It is not possible to identify all the actors in the first iteration itself. Hence **primary actors** are identified in the first iteration. These actors interact with the system to achieve required function. In next subsequent iterations, the **secondary actors** can be identified. The secondary actors support the system in such a way that the primary actors can perform their task.
- After identifying actors next step is to develop the use cases. **Jacobson** has suggested following set of questions that should be answered by the use cases -
 - What are the goals of the system ?
 - Who are the primary and secondary actors ?
 - What are the preconditions that should exist before the development of the system ?
 - What are the mains functions that must be performed by the actor ?
 - What are those exceptions that must be considered by the system ?
 - What are the changes in the behavior of the actor ?
 - What kind of system information to be produced by the actor ?
 - What kind of information is gained by the actor from the system ?
 - Will actor inform any change in the system to the external environment ?

The basic use case represents the high level scenario that describes the interaction between the actor and the system.

Example 4.5.1 Develop use case for Bank system.

Solution :

Following is the template suggested by **Cockburn**

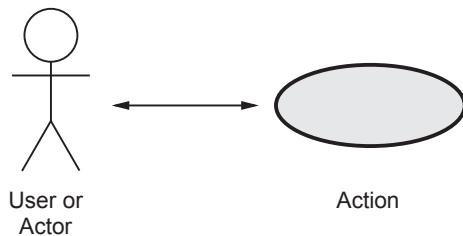


Fig. 4.5.1 Use case notation

Use Case template	
Use Case	ATM system
Primary Actor	Customer
Goal in Context	To monitor all the functionalities required to establish the session
Preconditions	ATM system has to be programmed and as to recognise and validate the PIN.
Trigger	On completion of every activity a beep has to be generated by the system.
Scenario	<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, deposit, inquiry) 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?

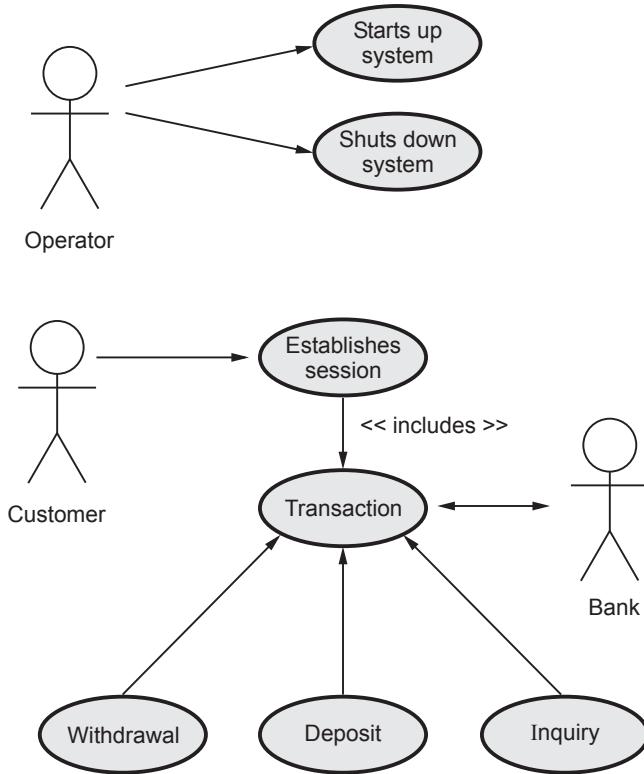


Fig. 4.5.2 Use cases for ATM system

The high-level use case for the ATM system can be as shown below -

Example 4.5.2 Write an use case for 'login' with a template and diagram.

Solution :

Use case template for login	
Use case	Course registration system.
Primary Actor	Student.
Goal in Context	To monitor all the functionalities required to register for the course.
Preconditions	None
Scenario / Basic Flow	<ol style="list-style-type: none"> 1. The system requests student to enter his name and password. 2. The student enters his/her name and password. 3. The system validates the entered name and password and logs the student into the system.
Exceptions	<ol style="list-style-type: none"> 1. The control panel is not ready. 2. User name is invalid. 3. Password is incorrect.

Priority	Essential and must be implemented in course registration system.
Postcondition	If user logs in successfully then course information must be displayed to him / her.
Secondary Actor	Administrator.

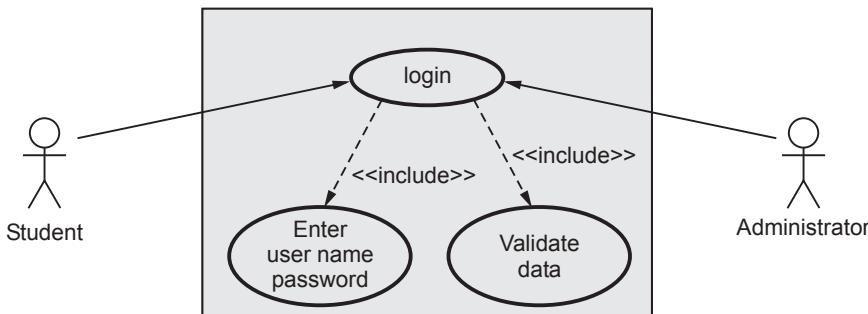


Fig. 4.5.3 Use case diagram

Review Question

1. What do you need to know in order to develop an effective use case ? Describe a standard use case documentation template.

4.6 Negotiating Requirements

During requirements engineering, the inception, elicitation and elaboration determine the customer requirements. But the requirements obtained by these tasks are not in sufficient details. There is an important requirement engineering task named **negotiation** in which the developer communicates with the customer so that some requirements specified by the customer can be negotiated or modified in order to meet the budget and deadline of the project.

Negotiations are carried out for following **two reasons** -

1. In order to develop the **project plan**.
2. By negotiations the real-world constraints are understood by both the developer and the customer.

If WIN_WIN situation is achieved after the negotiation then it is called as best negotiation. The WIN-WIN situation means customer gets satisfied because his major needs are going to get fulfilled and developer gets satisfied because the project development will be based on realistic goals and achievable budgets and deadlines.

The communication with the customer is the most important activity for the negotiation. Apart from this following are some **important activities suggested by Bohm for negotiations** -

1. Identify the important stakeholders of the system
2. Determine the requirements that are most important for the customer.
3. Negotiate with the customer in order to achieve the WIN-WIN condition.

4.7 Validating Requirements

GTU : Summer-2011, Marks 7

The analysis model is reviewed for consistency and unambiguity. Various requirements are prioritised and grouped in the software packages.

Following are some important questions that are considered while validating the requirements. This is also known as **requirements validation checklist** -

1. Does each requirement consistent ?
2. Are the requirements abstract ? That means - does the technical details of requirements are appropriate ?
3. Does particular requirement really necessary ?
4. Does each requirement bounded and unambiguous ?
5. Does the requirement have its reflection on information, function and behavior of the system ?
6. Does each requirement achievable by the software system ?
7. Does the requirement have some source ?
8. Does particular requirement conflict with some other requirement ?
9. Does each requirement tested after its implementation ?
10. Are the requirements used to simplify the requirements model ?

The requirements of the system have accurate reflection of the customer needs and they serve as the foundation for the software design.

Review Question

1. *Describe requirements validation.*

GTU : Summer-2011, Marks 7

4.8 Prioritizing Requirements

Requirements prioritization means determining which requirements of software product should be included in which release. It helps in balancing the business benefit of each requirement against the cost.

How prioritization helps ?

The prioritization of requirements help in following ways -

1. Prioritization helps to concentrate on the most important user and customer requirements.

2. It focuses on development effort
3. It manages the projects effectively.
4. Using such prioritization, it assists in planning for staged deliveries.
5. It helps in allocating the resources.

Why Prioritization of requirements ?

If the customer do **not differentiate their requirements** by importance and urgency, project managers must make these decisions on their own by prioritizing the requirements.

Method for prioritizing the Requirements:Cost Value Approach

- This is the **easiest method** for prioritizing the requirements.
- This approach was created by Joachim Karlsson and Kevin Ryan.
- In this method -
 1. **Requirements Engineer** review candidate requirements for their completeness and unambiguity.
 2. **Customers and users** apply **pair wise comparison**(value and cost of the requirements) to estimate the relative **value** of candidate requirements.
 3. **Software engineers** apply **pair wise comparison** to estimate relative cost of implementing each requirement.

The cost value analysis of the requirements is represented by following graph. The graph indicates the high priority, medium priority and low priority zone of requirements.

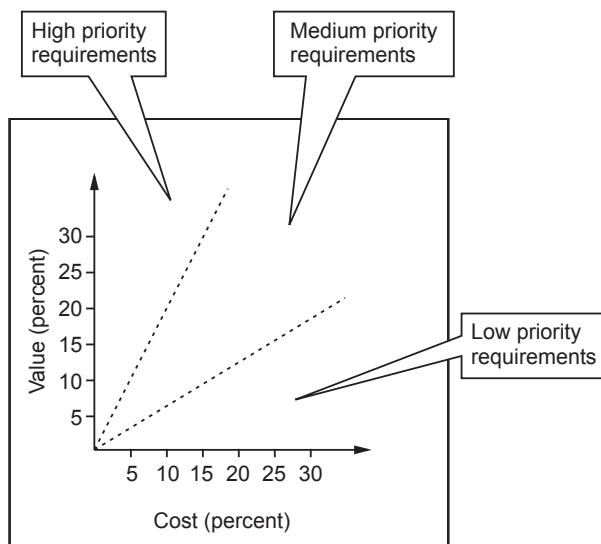


Fig. 4.8.1 Cost value Analysis

Advantages of Cost value Method

1. It is a reliable method for prioritizing the requirements.
2. It produces informative or clear results about prioritization.
3. Estimating the relative values is much more easier than estimating the absolute values.

Disadvantages of Cost value Method

1. This method is suitable for small number of requirements(less than 20 requirements)
2. This method does not consider the dependencies among the requirements.
3. One need to understand this method completely before applying it.

4.9 Building Requirements Analysis Model

Requirement analysis is an intermediate phase between **system engineering** and **software design**.

Requirement analysis produces a software specification.

How is requirement analysis helpful ?

Analyst - The requirement analysis helps the 'analyst' to refine software allocation. Using requirement analysis various models such as **data model**, **functional model** and **behavioral model** can be defined.

Designer - After requirement analysis, the designer can design for data, architectural interface and component level designs.

Developer - Using requirements specification and design the software can be developed.

What are requirement analysis efforts ?

1. Problem recognition

The requirement analysis is done for understanding the need of the system. The scope of the software in context of a system must be understood.

2. Evaluation and synthesis

Following are the tasks that must be done in evaluation and synthesis phase.

- i) Define all externally observable data objects evaluate data flow.

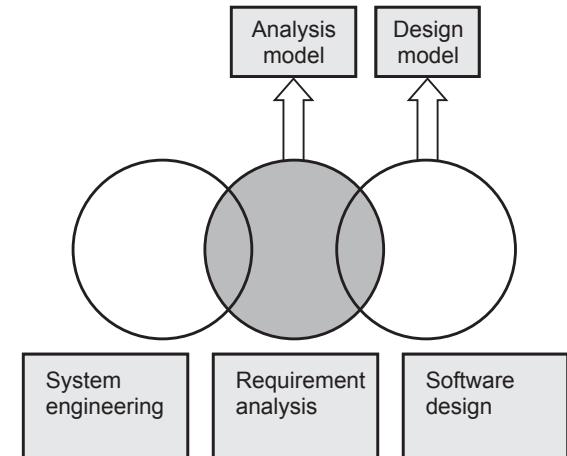


Fig. 4.9.1 Requirement analysis : An intermediate step

- ii) Define software functions.
- iii) Understand the behaviour of the system.
- iv) Establish system interface characteristics.
- v) Uncover the design constraints.

3. Modelling

After evaluation and synthesis, using data, functional and behavioral domains the data model, functional model and behavioral model can be built.

4. Specification

The requirement specification (SRS) must be built.

5. Review

The must be reviewed by project manager and must be refined.

4.9.1 Overall Objectives

- Following are **three objectives** of analysis model -
 1. Describe customer requirement.
 2. Create a basis for software design.
 3. Prepare valid requirements list.
- Analysis model bridges the gap between **system description and design model**. The system description describes overall **system functionality** and design model describes the **software architecture**, user interface and component level structure.
- There is no clear division of analysis and design tasks. Some design can be carried out during analysis and some analysis might be conducted during the design of the software.

4.10 Elements of Requirements Analysis

GTU : Summer-2015, Marks 7

Analysis modelling approach	
Structured approach	Object oriented approach
The analysis is made on data and processes in which data is transformed as separate entities .	The analysis is made on the classes and interaction among them in order to meet the customer requirements.
Data objects are modelled in such a way that data attributes and their relationship is defined in structured approach .	Unified Modelling Language(UML) and unified processes are used in object oriented modelling approach.

But the commonly used analysis model combines features of both these approaches because the best suitable analysis model bridges the software requirements and software design.

Following are the elements of analysis model -

- Scenario based elements
- Flow-oriented elements
- Behavioural elements
- Class based elements

Following Fig. 4.10.1 illustrates the elements of analysis model.

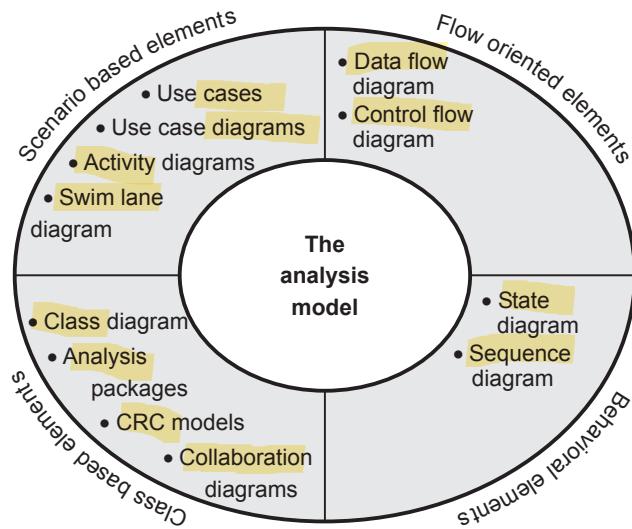


Fig. 4.10.1 Analysis model

Review Question

1. Explain requirement analysis with example.

GTU : Summer-2015, Marks 7

4.11 Scenario Based Modeling

GTU : Summer-2016, Winter-2017, Marks 7

When user of the computer-based system gets satisfied completely then that system or a product is said to be successful product. The software engineers try to understand how an user will interact with the system by properly characterising the requirements and by building the analysis models. In analysis modelling, at the beginning, creation of scenarios in the form of use cases, activity diagrams and swimlane diagram occurs.

4.11.1 Writing Use Cases

Use case diagrams represent the overall scenario of the system. A scenario is nothing but a sequence of steps describing an interaction between a user and a system.

For example : In library, student searches the book in catalog, finds book, reserves the book. He returns a book or pays fine for a book—all these activities constitute a scenario.

Thus use case is a set of scenarios tied together by some goal. The use case diagrams are drawn for exposing the functionalities of the system.

Actor

An actor is an entity which interacts with the system. Actors carry out use cases.

For example : In above use case diagram student is an actor.

A single actor may perform many use cases; conversely, a use case may have several actors. It is not necessary that the user should be an actor. The external system that gets some value or produces some value can be an actor. For example, Account system can be actor.

Actor is nothing but a role played by a person, system, device or even an enterprise, that has a stake in the successful operation of the system.

Use cases

The use cases represent the behavior of the system. Typically various functions are represented as use cases. For example, in the above use case diagram Reserve a book, Borrow book, Return book, Payment of fine are the use cases. Use cases can be represented as follows.



Relationships

Association : It identifies an interaction between actors and use cases. Each association represents a dialog.

Include relationship : Identifies a reusable use case that is *unconditionally* required for the execution of another use case. The decision about when and why to use the included use case is taken by the calling use case.

Extend relationship : Identifies a reusable use case that *conditionally* interrupts the execution of another use case by using its functionality. The decision of when the extending use case should be used is taken by the extending use case itself.

Generalization : Identifies an inheritance relationship between actors or between use cases.

Fig. 4.11.1 shows the association, include relationship, generalization and Extended use case for Clinical system. (See Fig. 4.11.1 on next page.)

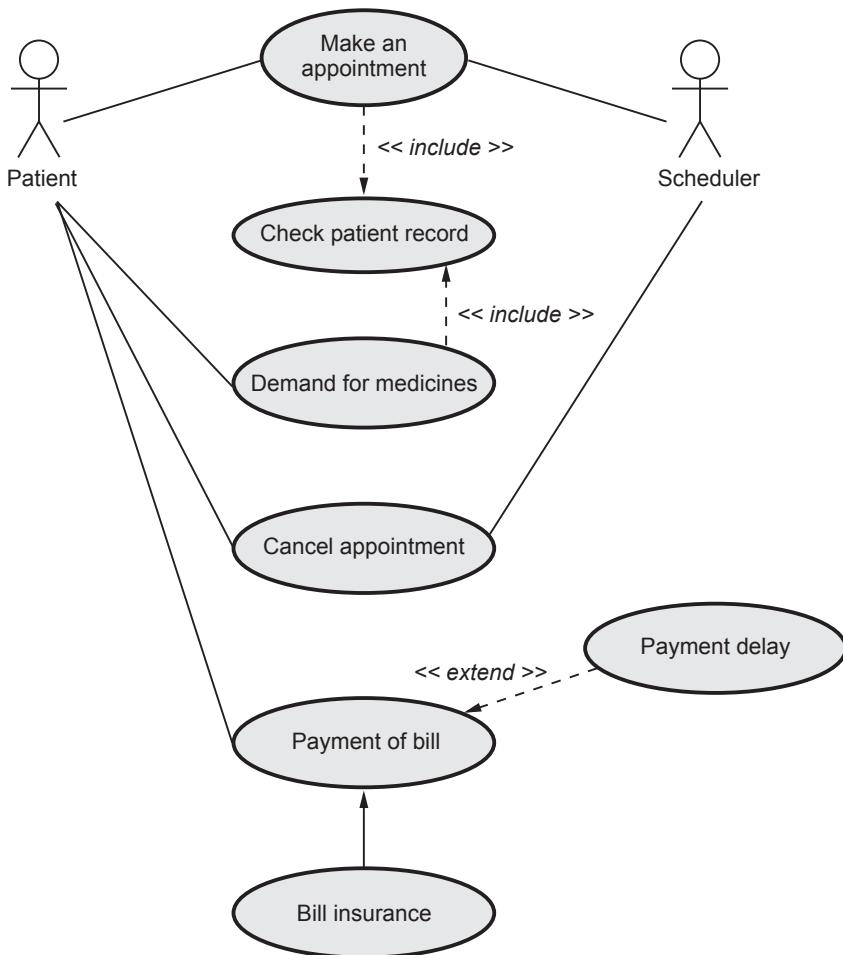


Fig. 4.11.1 Use case diagram with various relationship

Patient cancels appointment is shown by simple association relationship. To make an appointment check patient record is a include relationship. Payment of Bill use case is supported by an extended use case "payment delay".

For a generalization of "Payment of Bill" the child use case is "Bill insurance."

Example 4.11.1 Design use case diagram for user interaction with ATM system.

Solution : For an ATM system following are the scenarios with the user.

1. User enters the card, types user name and PIN.
2. User makes enquiry for balance. He may demand for getting the mini statement.
3. User withdraws the desired amount of money.
4. User may deposit some money.
5. Finally user closes the session.

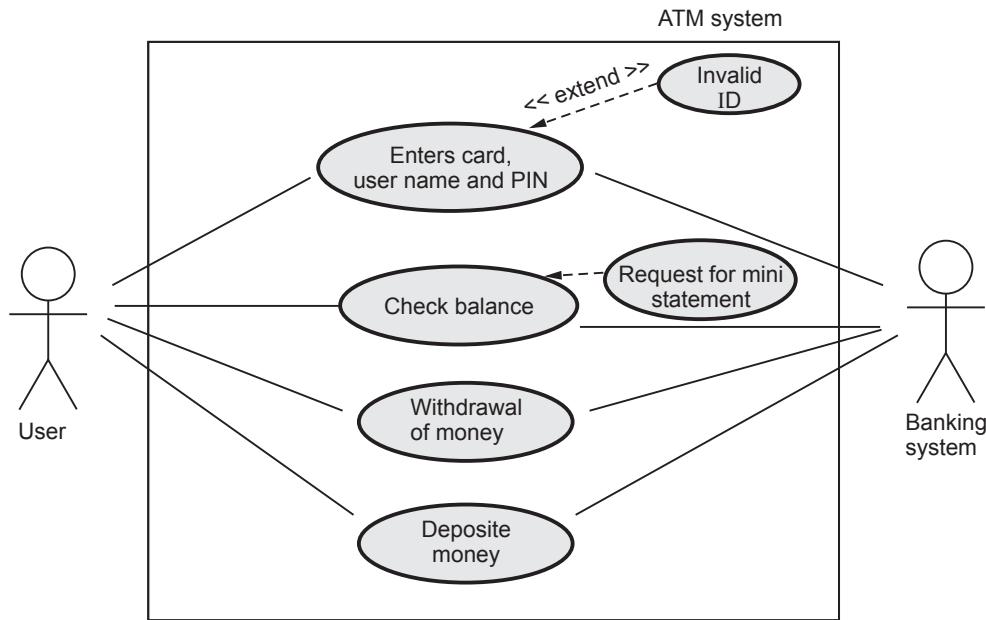


Fig. 4.11.2

Example 4.11.2 Draw a use case diagram for library management system.

GTU : Summer-2016, Marks 7

Solution :

Use case : For library management system

1. For library system user can be a staff or a student who initially registers himself as a new user.
 - a) For registering as a new user registration form is filled up.
 - b) Then librarian issues a library card on which some ID is assigned to the card holder.
2. The user requests for a new book.
3. The user then reserves a desired book.
4. The user may renew a book.
5. User pays fine if the book is returned late.
6. User can also submit his feedback by returning the feedback form.
7. There is another important actor in this system and that is "Librarian". The librarian adds the record in the library database.
8. Librarian deletes a record, if the book is not existing in the library or if student leaves the college.
9. Librarian updates the database.

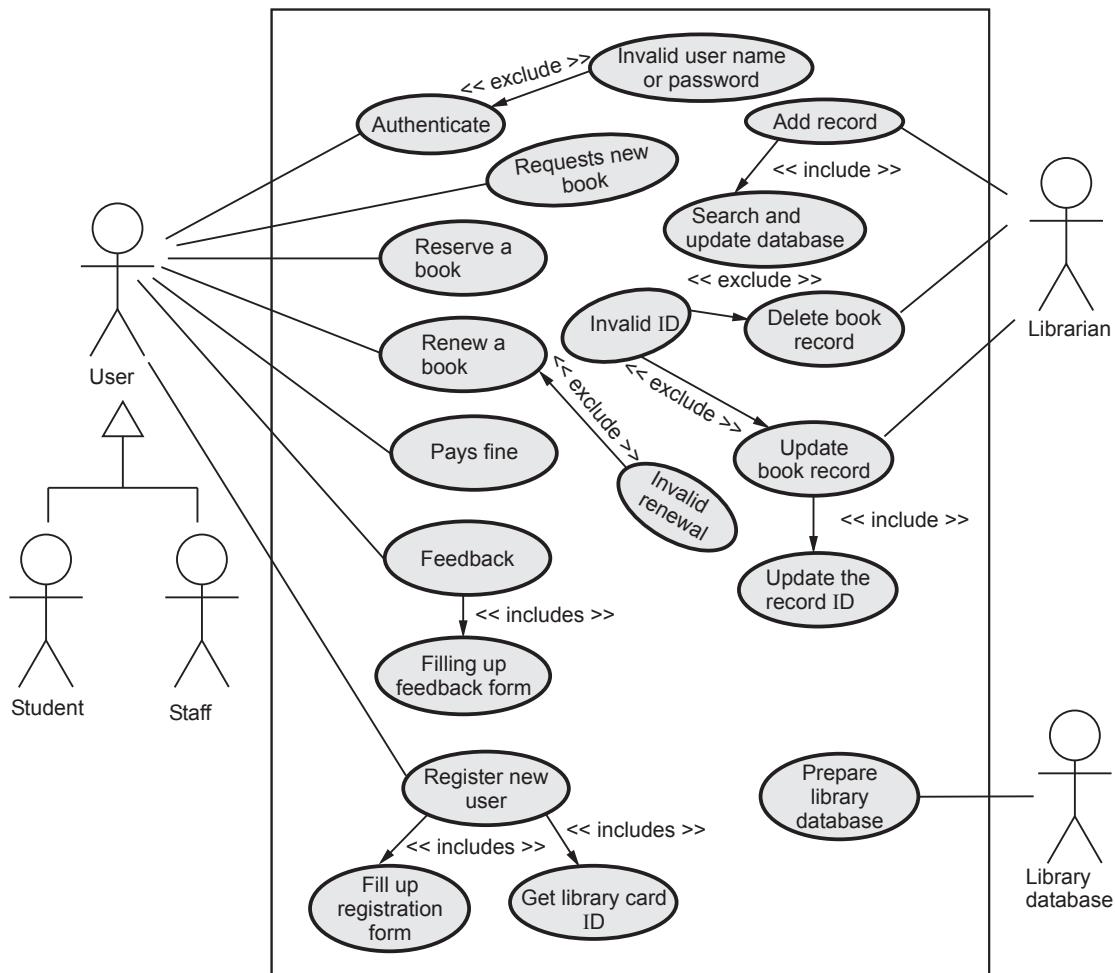


Fig. 4.11.3 Use case diagram for library management system

Example 4.11.3 Give the actors, use cases and use case diagram for "online mobile order and purchase system". Write the scope of the system.

Solution : The basic idea is that customers can buy products using online. It consists of product details, security system, status and exits. The administrator can enter the name and password and generate the report and can perform operations like add, search, delete the products in the database. The online mobile shopping system enables vendors to set up online shops, customers to browse through the shops and a system administrator to approve and reject requests for new shops and maintain lists of shop categories. Also on the agenda is designing an online shopping site to manage the items in the shop and also help customers purchase them online without having to visit the shop physically. Web customer actor uses some web site to make purchases online. Top level use cases are View Items, Make Purchase and Client Register.

Administrator : Person responsible for the system. The administrator is the person in charge of managing and administering the system. He also assumes the role of supervisor in the sense that he enforces the "Terms of use" of the site, and has the right to revoke client's privileges by deleting their account.

Bank : The supplier's bank. The bank gets involved to debit the client's account and credit the supplier's account during a purchase.

System : Dummy actor representing the system. This actor is a dummy actor used to represent the system in interactions diagrams for the use cases.

Client : Person purchasing products online. The client, also known as customer, is the person that logs onto the shopping cart online system to purchase products of his/her choice.

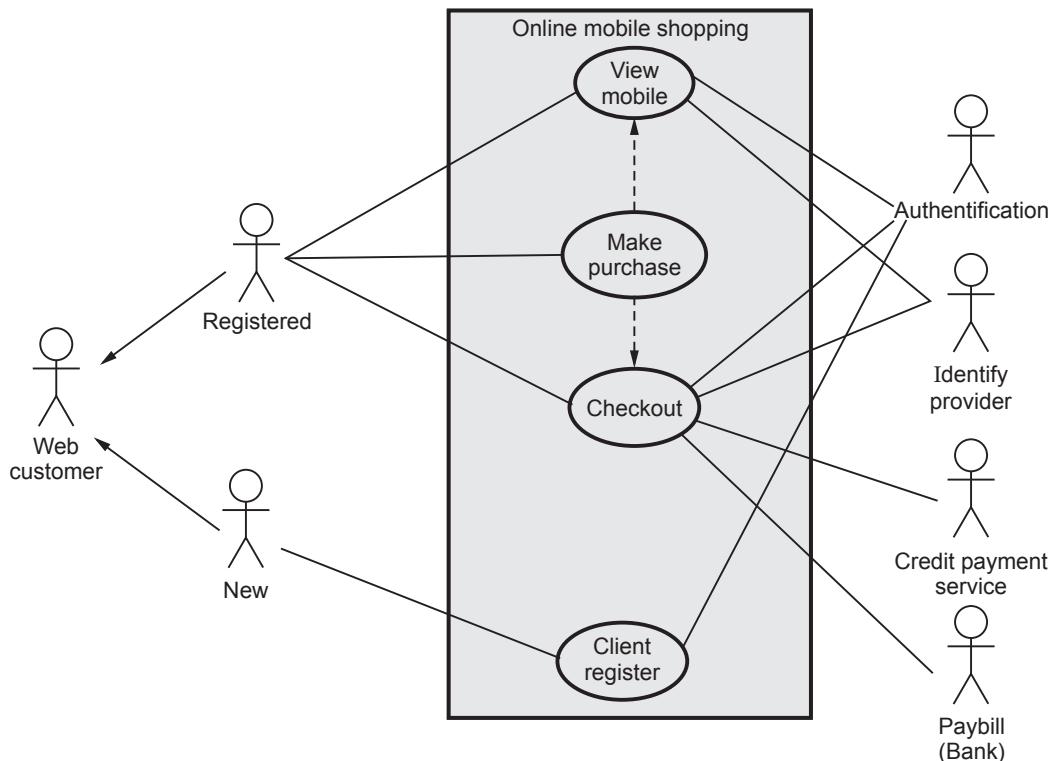


Fig. 4.11.4 Use case diagram for online mobile purchase

4.11.2 Activity Diagram

The activity diagram is a graphical representation for representing the flow of interaction within specific scenarios. It is similar to a flowchart in which various activities that can be performed in the system are represented. This diagram must be read from top to bottom. It consists of **forks** and **branches**. The fork is used to represent

that many activities can be parallelly carried out. This diagram also consists of **merge**, where multiple branches get combined. Before transitioning into final activity state there comes a **join**. A typical structure of activity diagram is -

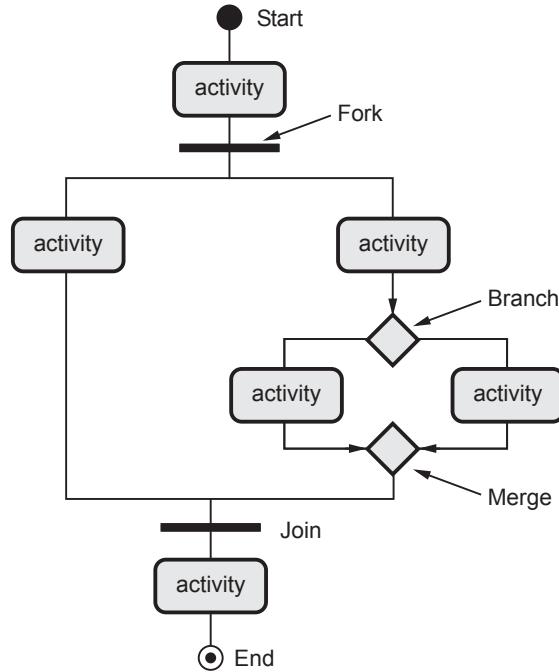


Fig. 4.11.5 Components of activity diagram

Below is the railway reservation system. The activity diagram for reserving a ticket is shown below. (Also refer Fig. 4.11.6 on next page)

4.11.3 Swimlane Diagram

The activity diagram shows various activities performed, but it does not tell you who is responsible for these activities. In swimlane diagram the activity diagram is partitioned according to the class who is responsible for carrying out these activities.

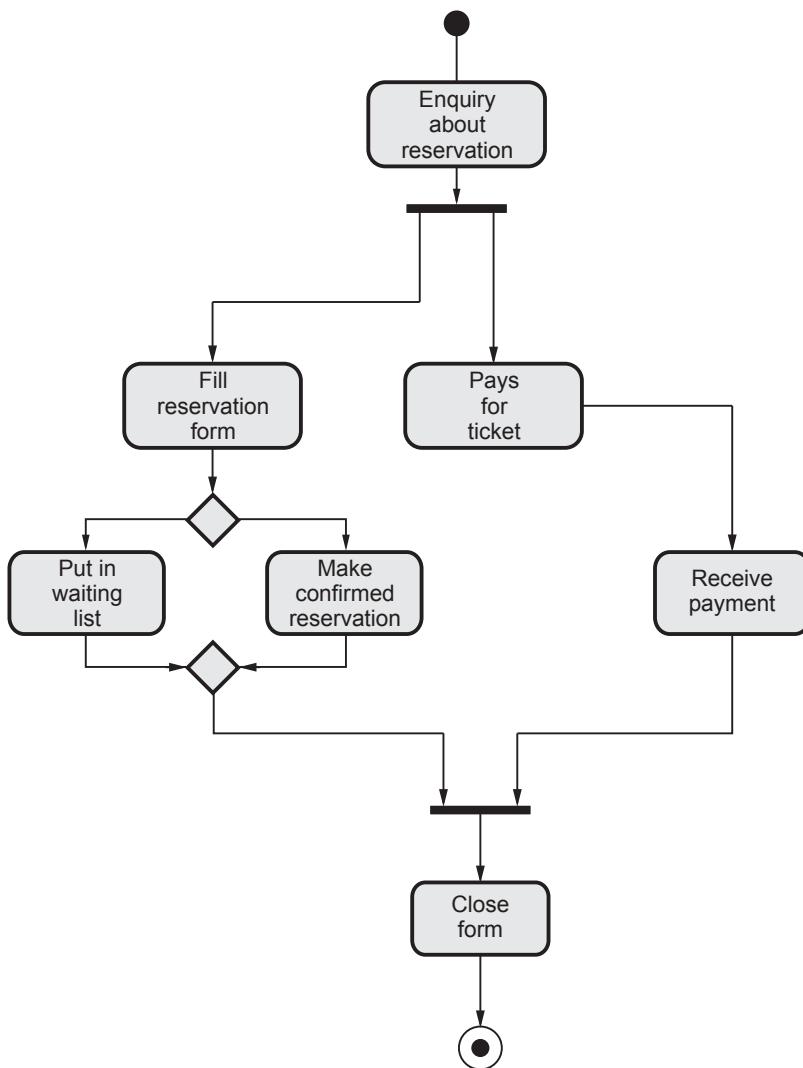


Fig. 4.11.6 Activity diagram for reservation

Example 4.11.4 Draw swimlane diagram for railway reservation activity.

Solution : Refer Fig. 4.11.7 on next page.

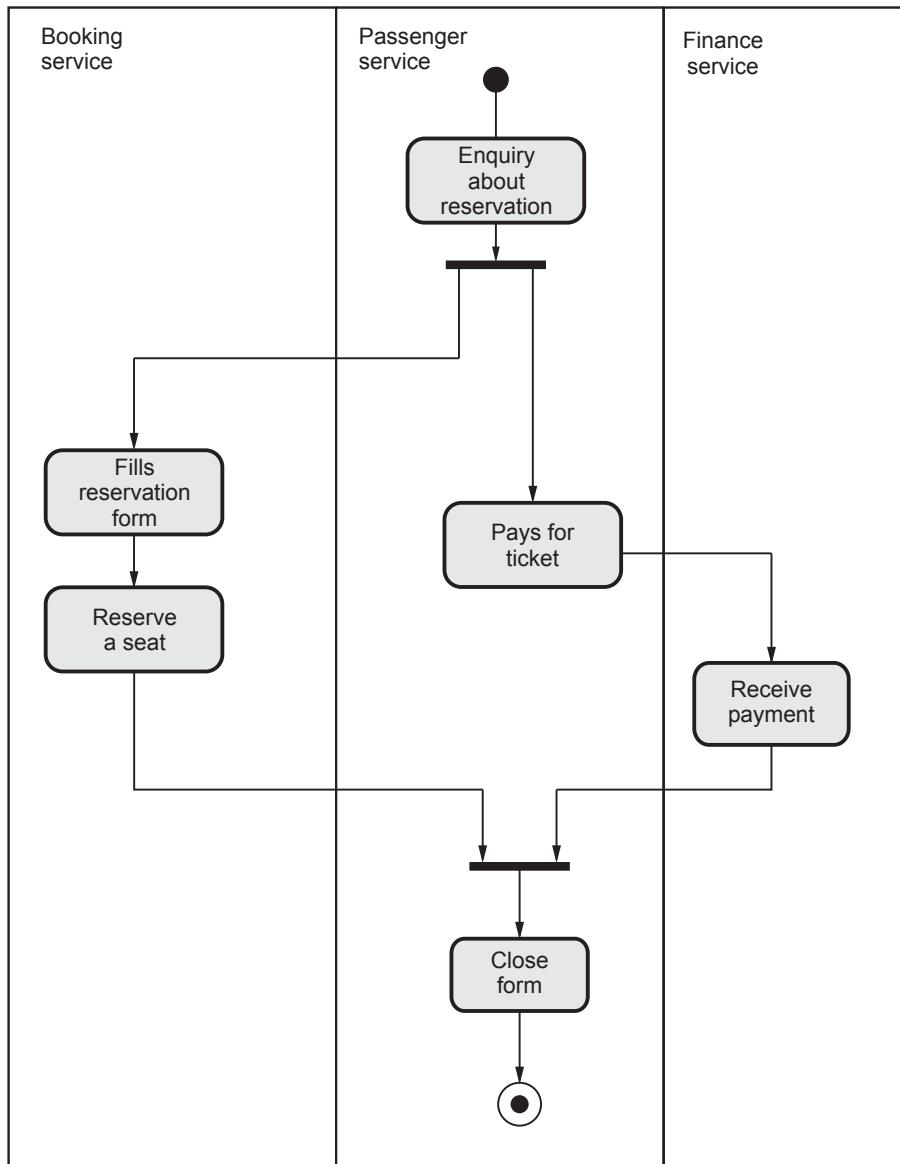


Fig. 4.11.7 Swimlane diagram for reservation

Example 4.11.5 Create the swimlane diagram for, monitoring of sensor in a 'Safehome security system' from control panel.

Solution : Refer Fig. 4.11.8 on next page.

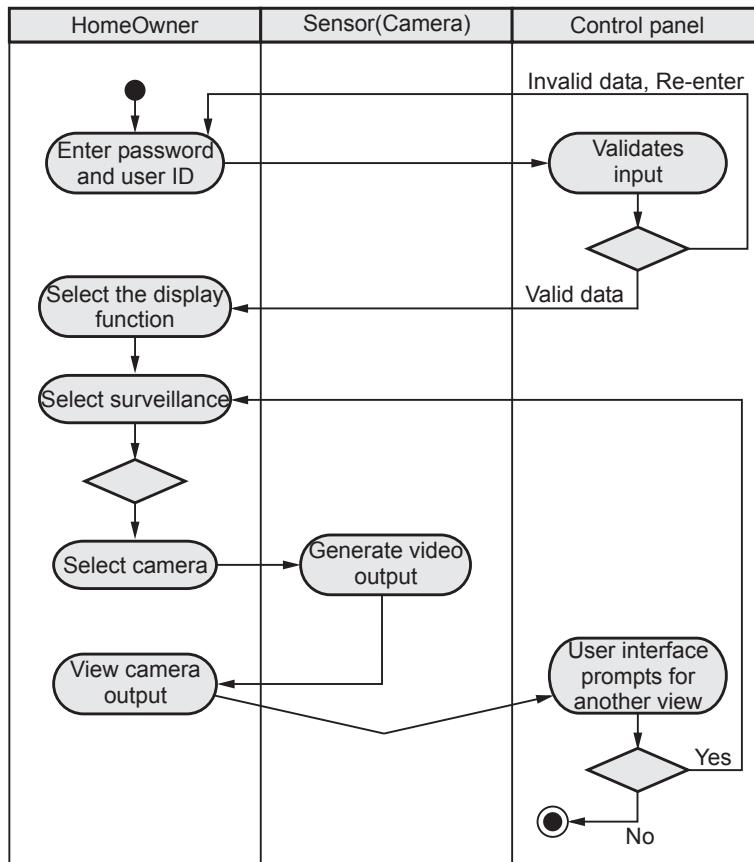


Fig. 4.11.8

Example 4.11.6 What is **activity diagram** and **Swim-lane**? Draw activity diagram for billing counter of a shopping mall.

GTU : Winter-2017, Marks 4

Solution : Activity diagram and Swim-lane : Refer sections 4.11.2 and 4.11.3.

Refer Fig. 4.11.9 on next page.

Activity diagram for customer at billing counter

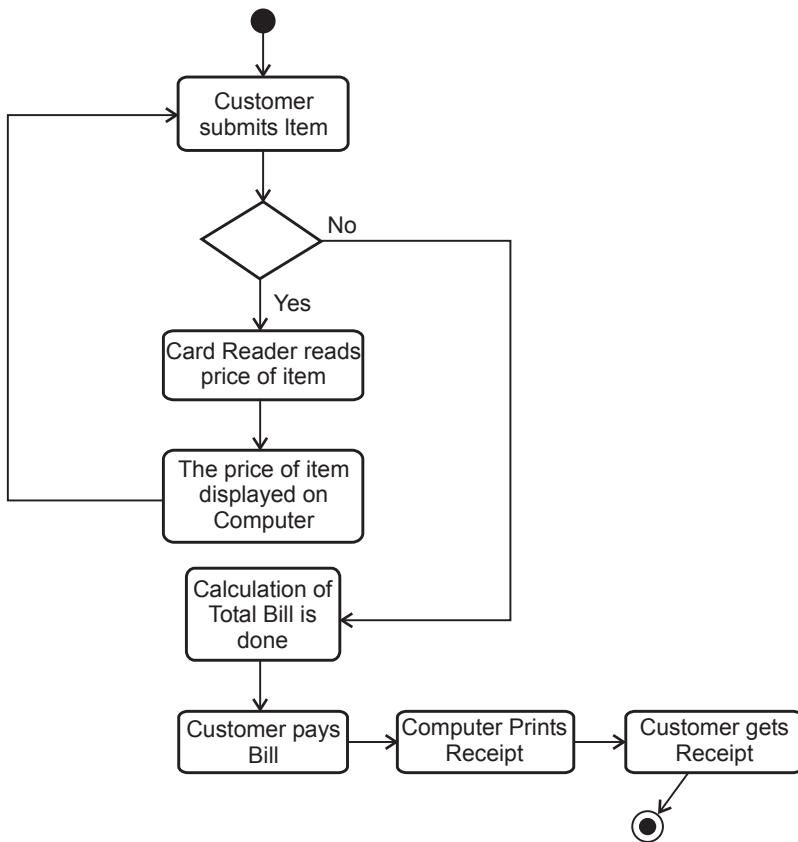


Fig. 4.11.9

4.12 Class Based Modeling

GTU : Summer-2016, Marks 3

Class diagrams in UML are used to capture the static view of the system. Basically class diagram represents how to put various objects together. The class diagram gives an overview of a system, in which various classes and relationship among these classes is represented. In UML class is a kind of classifier. For example, in class diagram manager, executive, salesperson can be represented by a class Employee. Thus each specific type of class is an instance of a class.

4.12.1 Objects and Object Classes

There are many definitions of an object, According to Booch -

"An object has state, behavior and identity; the structure and behavior of similar objects are defined in their common class; the terms instance and object are interchangeable".

Thus an object represents an individual, identifiable item, unit or entity, either real or abstract, with a well-defined role in the problem domain. Object are the entities which represent the instances of real world. And object class are templates for objects. Using object classes objects can be created.

In UML an object class is represented by rectangle which consists of three sections. It is as shown below.

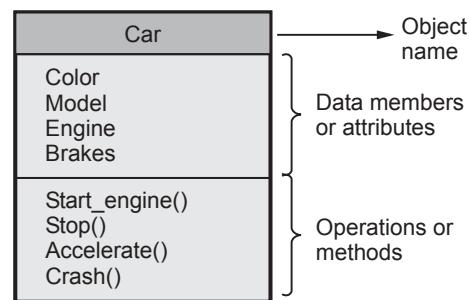


Fig. 4.12.1 Car object

The above object Car consists of various attributes or data members such as Color, Model, Engine and Brakes. The object can **communicate** with other object by **message passing**. The message passing can be done by using various operations or methods.

Hence any operation of the object consists of -

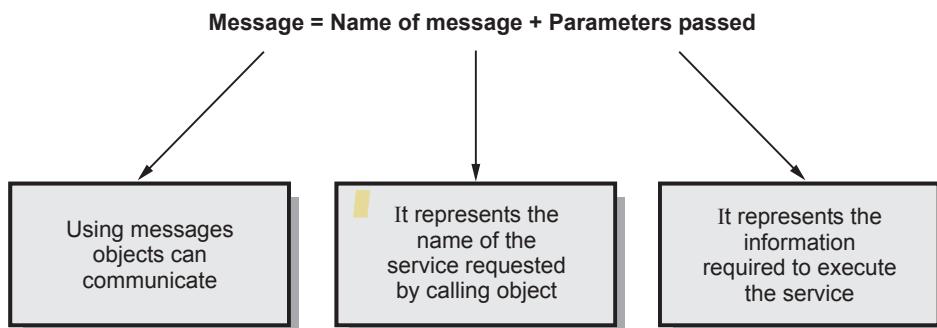
Name = Procedure name

Information = Parameter list.

The example of message is :

Circle.set_radius(10);

The object *Circle* has an operation *set_radius()* in which the radius is set by passing the value of radius.



4.12.2 Generalization and Inheritance Relationship

Generalization is one of the important features of object oriented systems. A class hierarchy can be formed where one class is generalisation of one or more other classes. That means there can be one **super class** whose attributes and operations can be inherited by the one or more **sub classes**. The sub classes may add its own attributes and methods. The example of generalization hierarchy is as shown below.

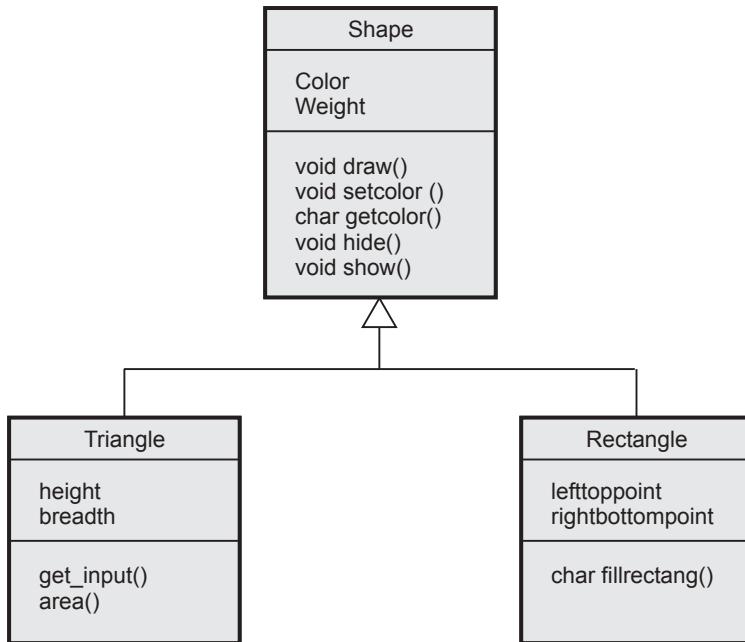


Fig. 4.12.2 Inheritance

In UML generalization is implemented as **inheritance**.

In this example the super class is a shape class from which two classes can be derived namely *triangle* and *rectangle*. The *triangle* and *rectangle* classes make use of some properties of base class shape and at the same time these classes have their own attributes and operations. Note that the inheritance relationship is shown by,



The arrowhead points to **base class** or **super class** and the other end of arrow is connected to **derived classes**. These classes inherit the properties of super class. Such classes are sometimes called as **child classes**.

The generalization indicates **type of relationship**. If the child classes are derived from more than one parent then it shows **multiple inheritance**. Fig. 4.12.3 shows the multiple inheritance.

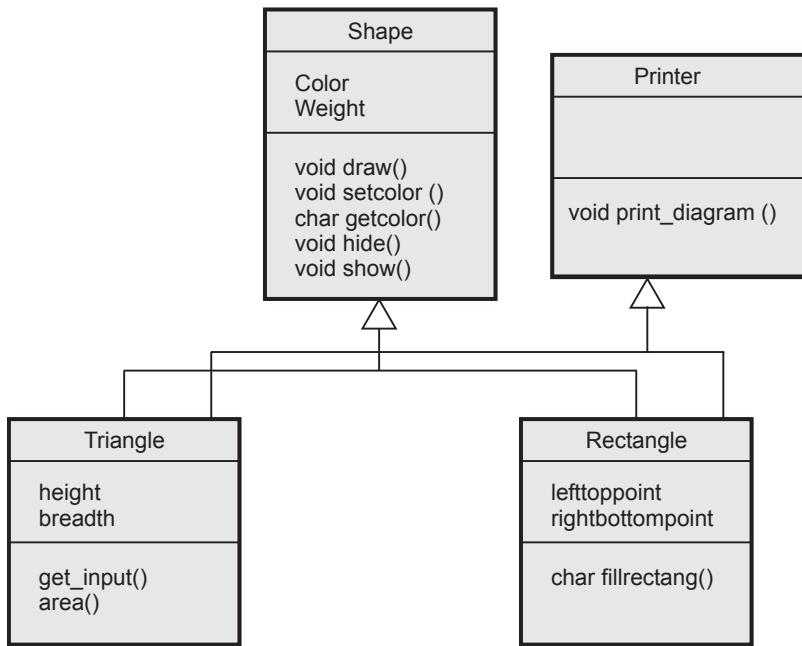


Fig. 4.12.3 Multiple inheritance

Advantages of inheritance

1. Inheritance brings **reusability** at design as well as at programming level.
2. The inheritance gives the complete knowledge about the domains and the systems used.
3. Inheritance specifies different types of classes that can be derived from one or more classes. Thus it brings **abstraction** mechanism in the design of the system.

Drawbacks of inheritance

1. Derived classes are not self contained. In order to understand the derived classes the super class must be known.
2. The inheritance graph created during analysis phase is not sufficient and cannot be directly used for implementation purpose. If it is directly used as it is then it leads to inefficient code generation.
3. The design of inheritance graph may vary during analysis, design and at implementation levels. Hence at each of these phases these graphs need to be maintained. And appropriate design must be selected for final implementation.

4.12.2.1 Association Relationship

Associations represent conceptual relationship between two classes. Associations **carry information** about the relationship among the objects of the system. Sometimes it is simply a link between two classes. The association may be **unidirectional** as well as **bidirectional**. Associations are **general** but may indicate that an attribute of an object is an associated object or that a method relies on an associated object. For example.

Associations have explicit notation to express **navigability**. If you can navigate from one class to another, you show an arrow in the direction of the class you can navigate to. If you can navigate in both directions, it is common practice to not show any arrows at all. Associations typically represent "**has a ...**" relationship.

Example 4.12.1 Draw a class diagram for library management system.

Solution : Aggregation : In the class diagram we can has 'part-of' relationship which represents the aggregation. The aggregation can be represented by an edge with diamond end pointing to the super class. In above class diagram, 'Library management system' is a super class which consists of various classes such as User, Book and Librarian. Similarly, 'User' is a super class which has 'Account' class. They share aggregate relationship.

Multiplicity : One or more instances of particular class can be associated with one or more instances of another class. For denoting one instance we write 1 and to denote many instances * is used. For example, in above given class diagram, many users can be associated with many books. On the other hand, every single user will have only one account. (See Fig. 4.12.5 on next page).

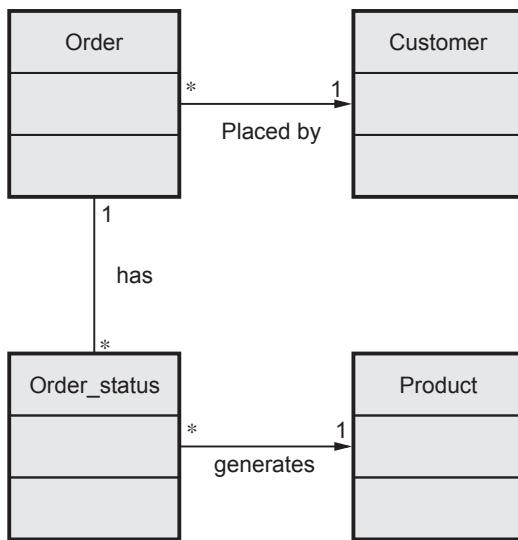


Fig. 4.12.4 Associations

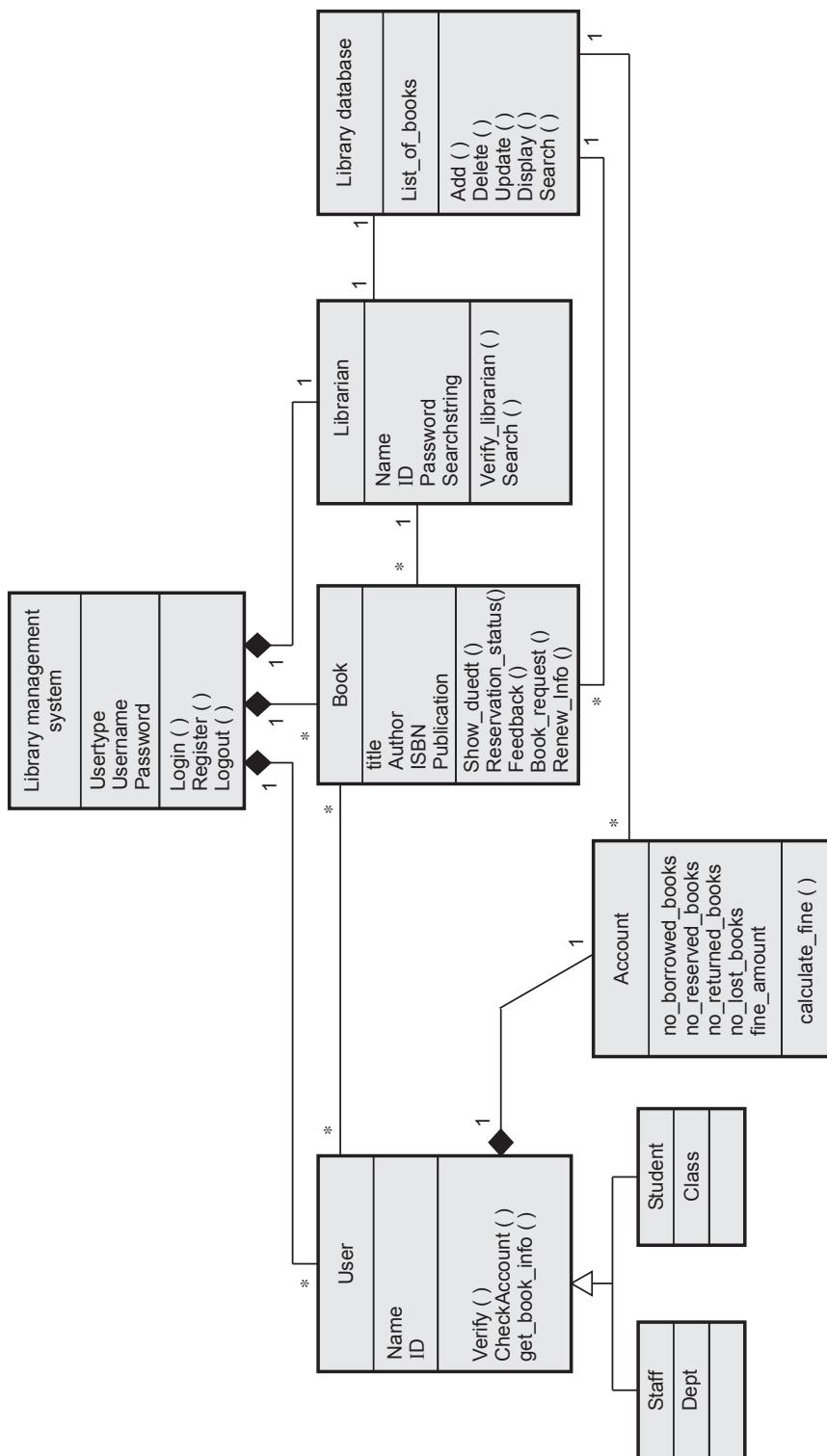


Fig. 4.12.5 Class diagram for library management system

4.12.3 Object Identification

An object identification is an important activity. It is the most crucial activity in object oriented designing. There are various **approaches** used for **object identification**.

1. Using natural language description the objects can be identified. The simple rule is that the nouns in the problem description correspond to objects and the verbs correspond to operations or behaviour. This is known as HOOD method.
2. The roles such as student, customer; the tangible entities such as course, book; the events such as schedule; the locations such as library, office support the objects. This is Coad and Yourdon approach.
3. By understanding the behaviour of the system objects can be obtained. The entities which initiate and participates the behaviour of the system are the objects.
4. Using system scenario the objects can be identified.

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

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

In above problem statement underlined words denote objects of the system. Hence -

1) ATM system is controller object.

2) Individual components can be

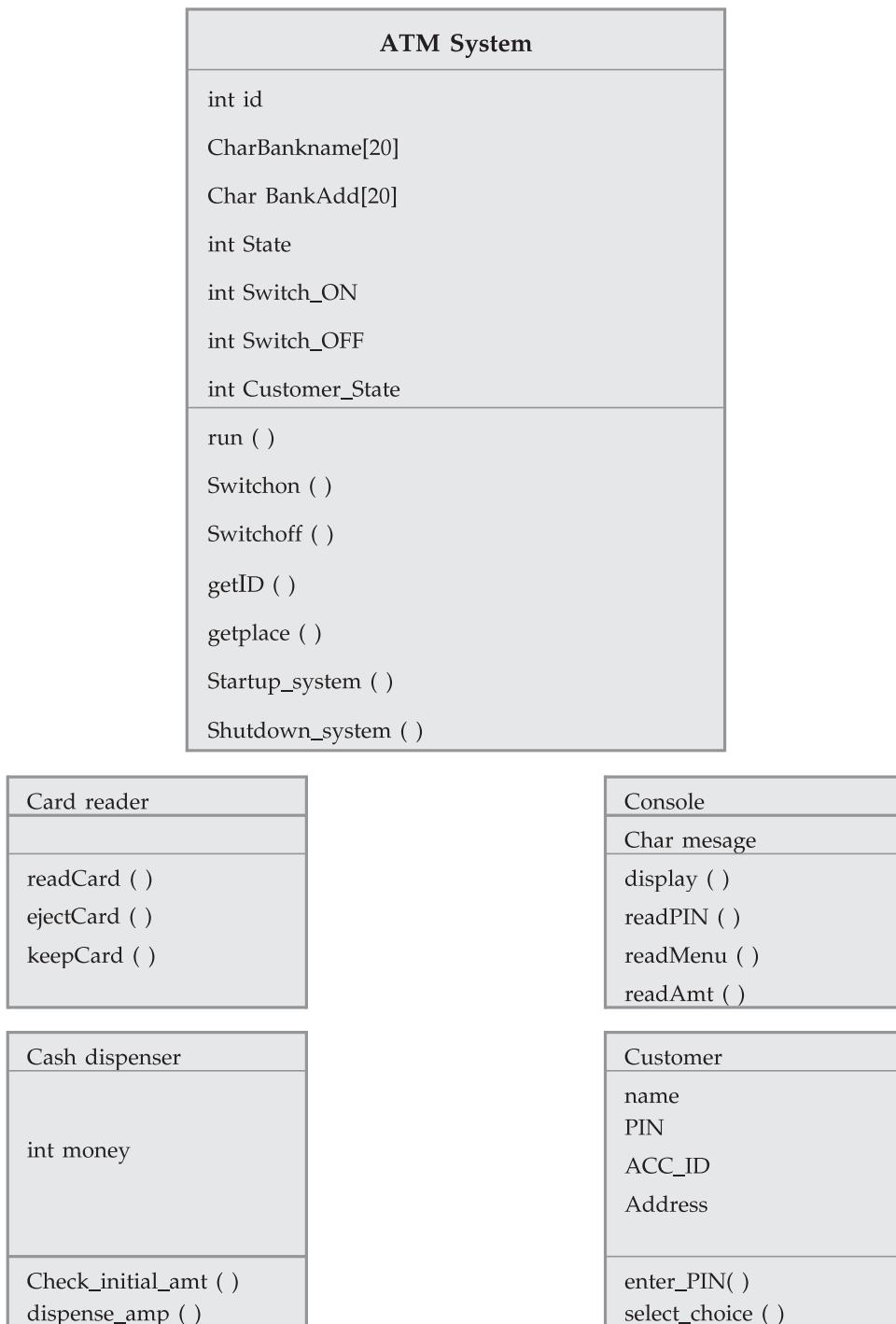
- Card reader
- Console
- Cash dispenser

3) The initiating object is customer.

4) The controlling objects are

- Session
- Transaction

Various objects can then be as shown below.



4.12.4 Class-Responsibility-Collaborator(CRC) Modelling

- The Class-Responsibility-Collaborator modelling is a technique which helps in identifying and organizing the classes that are relevant to the system requirements. Using this technique the classes required by the system are identified their responsibilities are defined and their collaborations are identified.
- Ambler** has described the CRC modelling as : "The CRC model is a collection of standard **index cards**. This card is divided into three sections. The first section contains name of the class, type of the class and characteristics of the class. Then the second section is described at the left side and third section is described at the right side of CRC card. The second section describes the responsibilities and the third section describes the Collaborations.

Class Name:	
Class type:	
Characteristics:	
Responsibilities:	Collaborations:

Fig. 4.12.6 Structure of CRC index card

- The typical structure of CRC card is as shown by following Fig. 4.12.6.
- Class** - For selecting particular entity as a class following guideline can be used -
 - Information retaining** - There are some objects that have potential of retaining the information. This information can be used by the system for defining the functionalities. Hence chosen object must contain some useful information.
 - Services** - The set of identified objects specify the required services. These services are provided by the various functionalities of the system. Hence the objects must be chosen in such a way that they provide some services to the system.
 - Common attributes** - Every object defines some set of attributes. These attributes can be member variables or member functions. Some member variables are common to all the objects. The selection of object can be done based on such common characteristics of the system.
 - Common operations** - Every object defines some set of attributes. These attributes can be member variables or member functions. Some member functions are common to all the objects. The selection of object can be done based on such common characteristics of the system.
 - Multiple attributes** - The object that contains multiple attributes can retain more information about the system. Such object can provide some useful service to the system. While selecting particular entity as an object it is necessary to ensure that the chosen object has multiple attributes. This will lead to focus on major information.

6. Essential requirements - The external entities are responsible for producing or consuming some kind of information. These entities can act as an object. Selection of such entities as an object helps to perform various operations that are required for getting the solution of the problem defined by the system.

Fishersmith has suggested following types of classes.

- 1. Device classes** - The external devices can be declared as the classes such as keyboard, mouse, display, control panel
- 2. Interaction classes** - The interaction among various objects can be represented by the classes. For example : Account or a purchase
- 3. Property classes** - When particular property of the problem environment has to be represented then it can be represented by a class.

Various characteristics of objects are :

- 1. Tangibility** - Does the class represent the tangible thing or represent an abstract information?
 - 2. Sequentiality** - Whether the class is concurrent or sequentially defined?
 - 3. Inclusiveness** - Does the class include other class or it is independent of other class? This represents **atomic** or **aggregate** property.
 - 4. Persistence** - Is the class permanent or temporary ? Sometimes the class created at the design time are removed during the implementation. Such classes are temporary classes and there are some classes that remain in the database. Such classes are called permanent classes.
 - 5. Integrity** do the class allow other resources to access it or does it protect itself from other resources ? If the class does not allow other resources to access itself then such classes are called **guarded** and if the classes allow other resources to access itself then such classes are called **corruptible**.
- **Responsibilities** : The responsibilities of a class can be determined by its attributes and responsibilities. **Wirfs-Brock** has suggested following guidelines for allocating the responsibilities to the classes -
 - 1. System intelligence should be evenly distributed**

System intelligence means the ability of the system. In simple words, "what system can do" and "what system knows" means system intelligence. Regarding distribution of responsibility, one approach was adopted earlier, that is : there will be two types of classes **smart class** and **dumb class**. The class who carries out more responsibilities is supposed to be the smart class and the class who carries out few responsibilities or who acts as the 'servant' of smart class is called the dumb class. But this approach has some drawbacks. Firstly, the major responsibilities get concentrated among few classes,

secondly making changes in such system becomes more complicated and lastly by adopting such approach system may require more number of classes.

Hence system intelligence must be evenly distributed among several classes. That means **cohesiveness** of the system must be improved.

We can check whether the system intelligence is evenly distributed or not by using CRC model index card. In this case, a list of responsibilities is made. If a particular class has more number of responsibilities than expected then that means intelligence is centred on that corresponding class.

2. Each responsibility should be mentioned in generalised way.

The responsibilities mean attributes and operations should be stated in most **general** way. Similarly **polymorphism** should be adopted wherever possible.

3. Information and the behaviour related to particular class should remain in that class only.

The data and related operations should be packaged in a single entity called class. This indicates the **encapsulation** property.

4. Information about particular class should be for that class only and it should not be distributed among several classes.

If the related information is not localized and if it is distributed among several classes then such systems become more difficult to manage.

5. Appropriate responsibilities can be shared among multiple classes.

All the related objects must exhibit the same behaviour at the same time in order to maintain the consistency in the system operation.

- **Collaborations**

The class can fulfill its responsibilities by

- Using its own set of operations and attributes.
- By **collaborating** with other classes.

Collaborations can be defined as is nothing but the identification of relationships between classes. When a set of classes collaborate with some other classes for fulfilling some requirements then they actually form a **subsystem**.

When a complete CRC model has been developed the customer and a software engineer must review this model. While reviewing such model following approach must be adopted -

1. All members of review must be given CRC index cards. Cards that indicate the collaborations must be separated out.
2. Appropriate classification of all the scenarios of the use cases must be done.

3. A token is maintained. There is review leader who reads out the use-cases. When a particular phrase occurs then the review leader passes the token to the person who is holding the CRC index card of that particular phrase (in fact it is a class name). It is then examined whether all possible responsibilities of that class are mentioned in the CRC card or not.
4. The review then determines whether or not the responsibilities of corresponding class are properly noted on the CRC card.
5. If the responsibilities and collaborations noted on the index card can not fulfil the scenarios of use cases then some modifications can be made in the CRC card.

Example 4.12.2 Write a CRC for the class 'Email system user'.

Solution :

Class: Email System User	
Attributes: user_id, password	
Responsibilities	Collaborator
Enters the user id and password for logging in.	Authentication sub-system
Reads the mail Sends the mail	Mailing sub-system
View/delete address-book entries.	address book sub-system
chatting	messenger subsystem

4.12.5 Object Relationship Model

- Using CRC modelling we can identify the classes and objects. Then following steps are adopted

Step 1 : The relationship among the several classes must be established.

Step 2 : Identify the responsibilities of each class.

Step 3 : Define the collaborator classes so that the responsibilities can be achieved.

When two classes are connected then the **relationship** exists between the two. The most common type of relationship is a binary relationship. For example a client server relationship is a kind of binary relationship.

- According to **Rumbaugh** relationship can be derived by examining the problem statement. According to him the **verb** or **verb phrases** represent the **relationships**. Hence it is expected to do the grammatical parsing and isolate the verbs from the problem statement. Typically communication, locations, ownerships and satisfaction conditions indicate the relationships.

- Following are the 3 steps used to derive the object relationship model -

- Using CRC index cards the Collaborator objects can be identified.*

Following Fig. 4.12.7 represents the various CRC index cards that can be connected together. Note that these objects are simply connected together no named relationship exists at this stage.

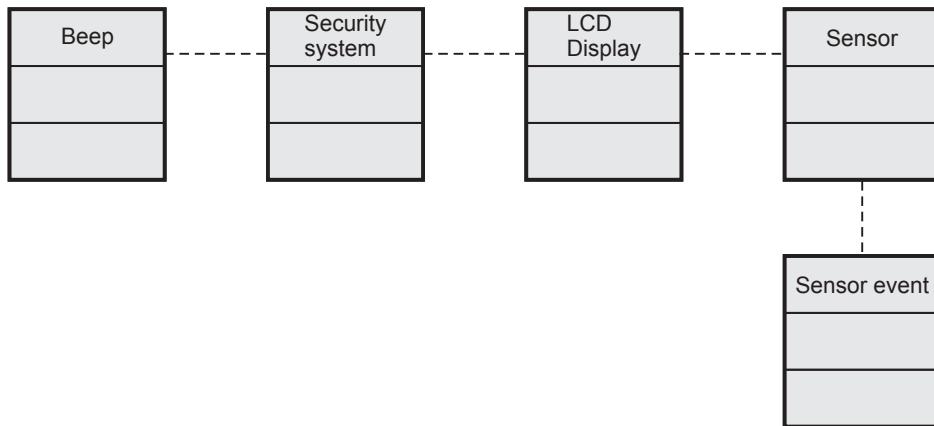


Fig. 4.12.7 CRC index cards

- Evaluate the responsibilities and collaborators of CRC index cards and label each connected lines.*

Now each relationship of the object must be named appropriately. Thus relationship gets established among the multiple objects.

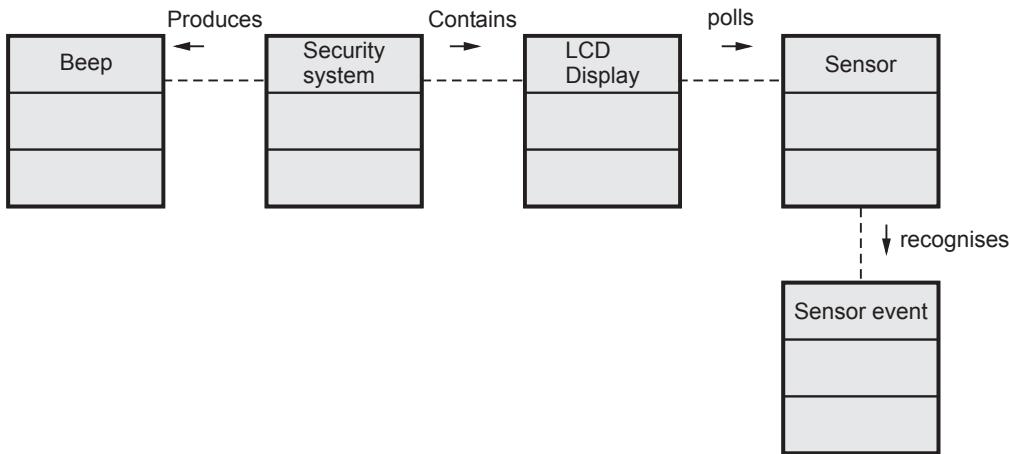


Fig. 4.12.8 Relationship among CRC cards

- Determine the cardinality.*

Once the named relationship gets established then cardinality can be determined. There are four types of cardinalities exists : 0 to 1, 1 to 1, 0 to many and 1 to many.

For example :

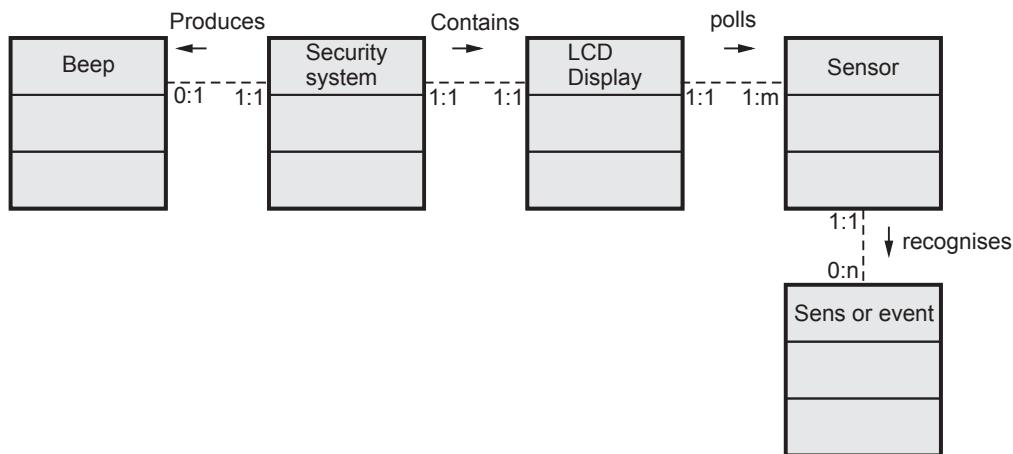


Fig. 4.12.9 Cardinality of relationship

Review Question

- What is object oriented design of a system ? Draw the use case diagram and class diagram for library management system.

GTU : Summer-2016, Marks 7

4.13 Data Modeling

GTU : Summer-2012, 2014, 2015, Winter-2019, Marks 7

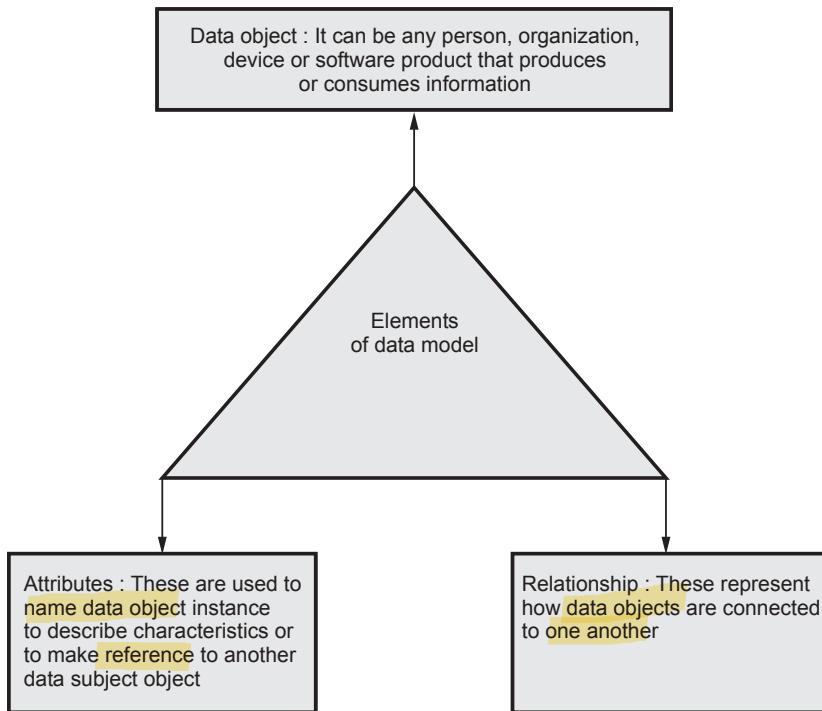
- Data modelling is the basic step in the analysis modelling. In data modelling the data objects are examined independently of processing.
- The data domain is focused. And a model is created at the customer's level of abstraction.
- The data model represents how data objects are related with one another. (See Fig. 4.13.1 on next page)

4.13.1 Data Objects, Attributes and Relationships

What is data object ?

- Data object is a set of attributes (data items) that will be manipulated within the software(system).
- Each instance of data object can be identified with the help of unique identifier. **For example :** A student can be identified by using his roll number.
- The system cannot perform without accessing to the instances of object.
- Each data object is described by the attributes which themselves are data items.

Data object is a collection of attributes that act as an aspect, characteristic, quality or descriptor of the object.

**Fig. 4.13.1 Elements of data model**

Object : <i>Vehicle</i>
<i>Attributes:</i>
<i>Make</i>
<i>Model</i>
<i>Color</i>
<i>Owner</i>
<i>Price</i>

Fig. 4.13.2 Object

The vehicle is a data object which can be defined or viewed with the help of set of attributes.

Typical data objects are

- External entities such as printer, user, speakers.
- Things such as reports, displays, signals.
- Occurrences or events such as interrupts, alarm, telephone call.
- Roles such as manager, engineer, customer.
- Organizational units such as division, departments.
- Places such as manufacturing floor, workshops.
- Structures such as student records, accounts, file.

What are attributes ?

Attributes define properties of data object

Typically there are three types of attributes -

1. **Naming attributes** - These attributes are used to name an instance of data object.
For example : In a *vehicle* data object *make* and *model* are naming attributes.
2. **Descriptive attributes** - These attributes are used to describe the characteristics or features of the data object. For example : In a *vehicle* data object *color* is a descriptive attribute.
3. **Referential attribute** - These are the attributes that are used in making the reference to another instance in another table. For example : In a *vehicle* data object *owner* is a referential attribute.

What is relationship ?

Relationship represents the connection between the data objects. For example

The relationship between a shopkeeper and a toy is as shown below

Here the toy and shopkeeper are two objects that share following relationships -

- Shopkeeper orders toys.
- Shopkeeper sells toys.
- Shopkeeper shows toys.
- Shopkeeper stocks toys.

4.13.2 Cardinality Modality important

Cardinality in data modeling, cardinality specifies how the number of occurrences of one object is related to the number of occurrences of another object.

- One to one (1:1) - One object can relate to only one other object.
- One to many (1:N) - One object can relate to many objects.
- Many to many (M:N) - Some number of occurrences of an object can relate to some other number of occurrences of another object.

Modality indicates whether or not a particular data object must participate in the relationship.

Modality of a relationship is 0 (zero) if there is no explicit need for the relationship to occur or the relationship is optional. The modality is 1 (one) if an occurrence of the relationship is mandatory.

Example

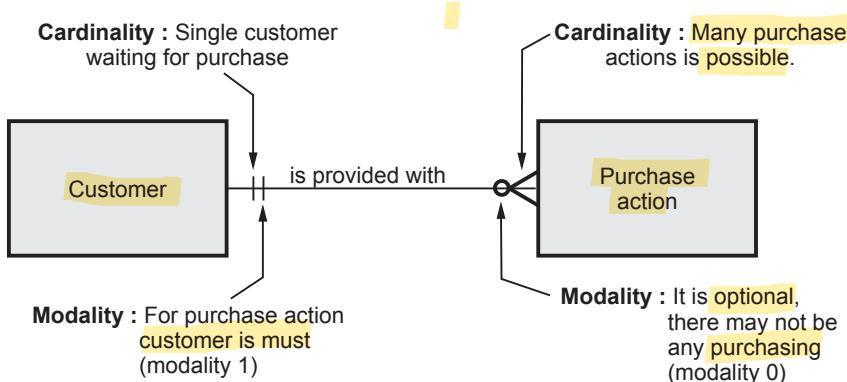


Fig. 4.13.3 Cardinality and modality

4.13.3 Entity Relationship Diagram

- The object relationship pair can be graphically represented by a diagram called **Entity Relationship Diagram(ERD)**.
- The ERD is mainly used in database applications but now it is more commonly used in data design.
- The ERD was originally proposed by Peter Chen for design of relational database systems.
- The primary purpose of ERD is to represent the relationship between data objects.
- Various components of ERD are -

Entity

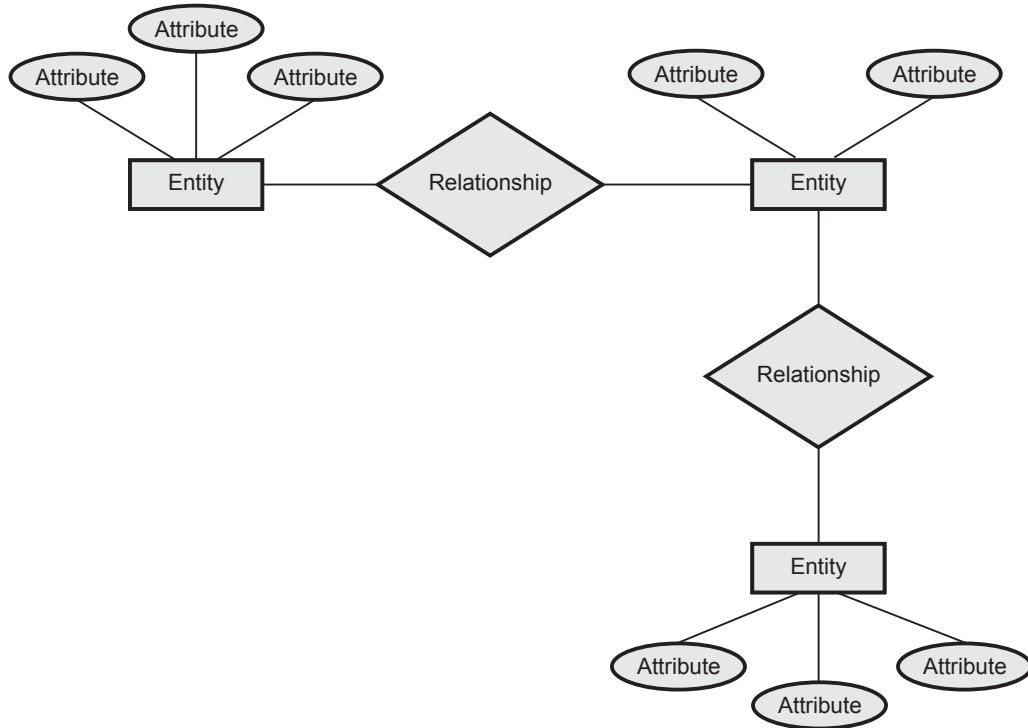
- Drawn as a rectangle.
- An entity is an object that exists and is distinguishable.
- Similar to a record in a programming language with attributes.

Relationship

- Drawn as a diamond.
- An association among several entities.
- Relationships may have attributes.
- Relationships have cardinality (e.g., one-to-many).

Attribute

- Drawn as ellipses.
- Similar to record fields in a programming language.

**Fig. 4.13.4 ER diagram**

- Each attribute has a set of permitted values, called the domain.
- Primary key attributes may be underlined.
- The typical structure of ER-diagram is illustrated as follows.

Notations used in ER diagram

Entity

It is an object and is distinguishable it is similar to record.



Weak entity

When this entity is dependant upon some another entity then it is called weak entity.



Attribute

The attributes are properties or characteristics of an entity.



Derived attribute

It is a kind of attribute which is based on another attribute.



Key attribute

A key attribute is an unique attribute representing distinguishing characteristic of entity. Typically primary key of record is a key attribute.

Multivalued attribute

A multivalued attribute have more than one value.

Relationship

When two entities share some information then it is denoted by relationship.

Notations to show cardinality

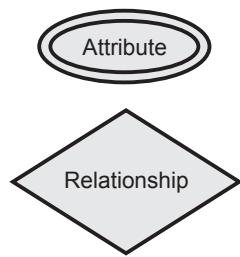
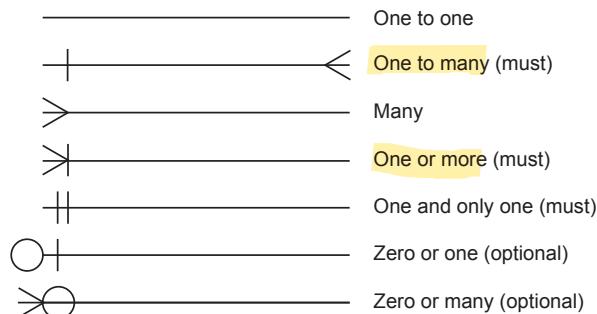


Fig. 4.13.5 Cardinality

Example 4.13.1 Draw an ER diagram for the relationship of teacher and courses. Also specify the association, cardinality and modality.

Solution :

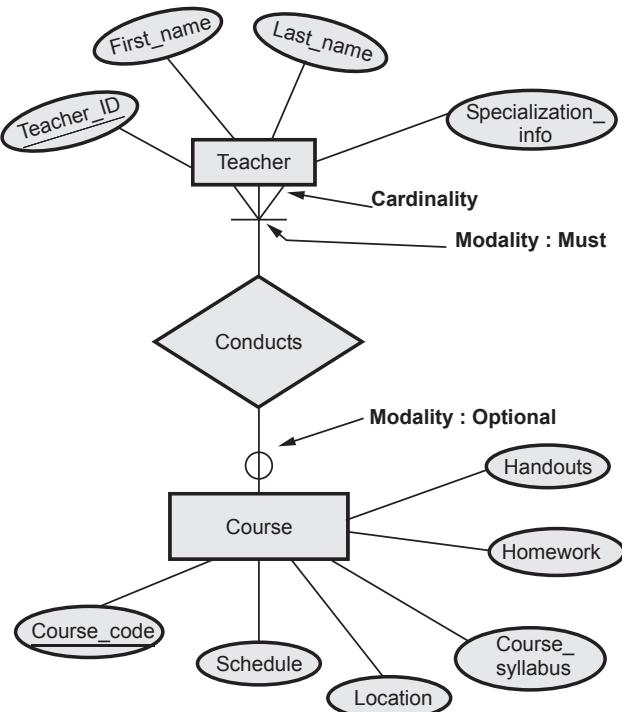


Fig. 4.13.6

Association

In the above ER diagram, a relationship **conducts** is introduced. "Teacher" is associated with "Course" by conducting it.

Cardinality

Many Teachers can conduct the single course.

Modality

For conduction of a course, there must be a Teacher.

There may a situation that a Teacher is not conducting any course.

Example 4.13.2 Draw an ER diagram for the relationship between customer and banking system.

Solution :

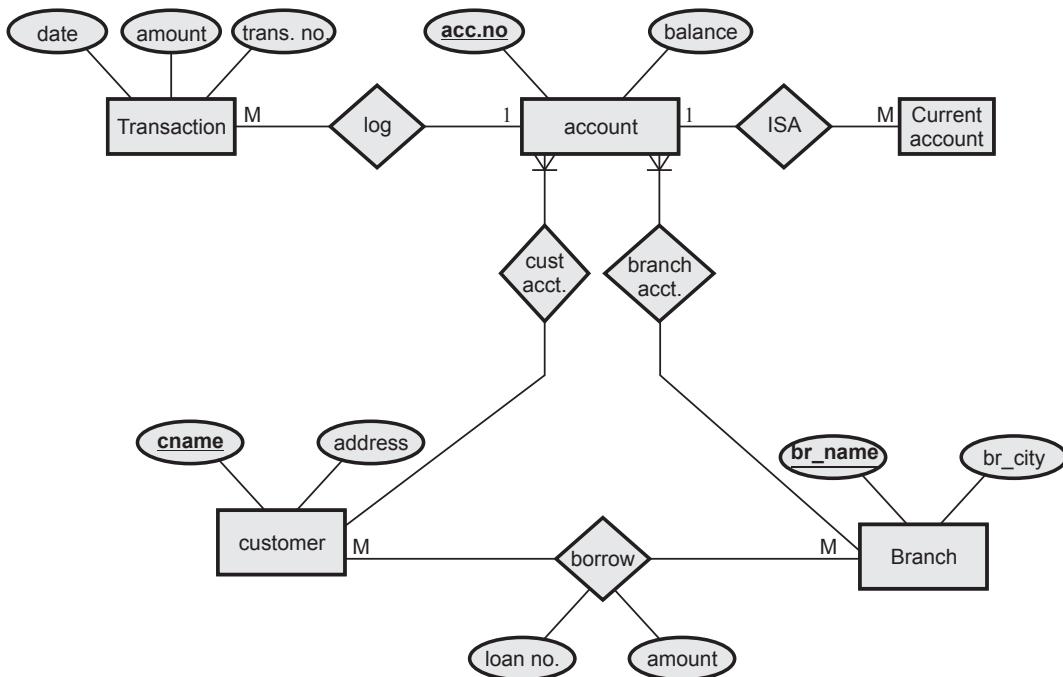


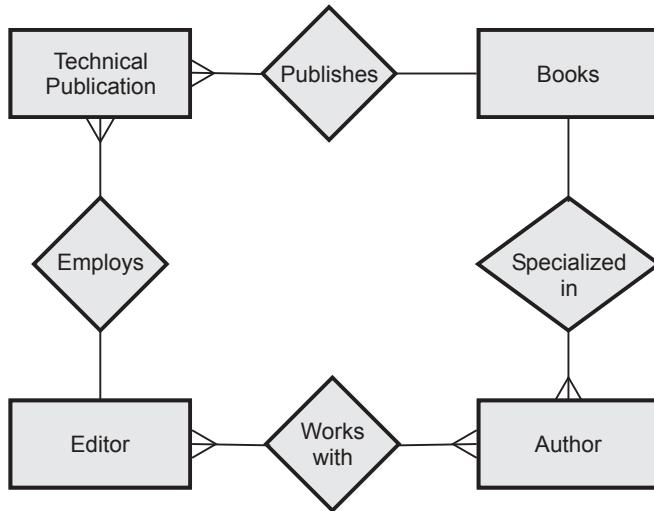
Fig. 4.13.7 ERD

Example 4.13.3 Technical Publication (A well known publishing company) publishes Engineering books on various subjects. The books are written by authors who specialize in his/her subject. The publication employs editors who take sole responsibility of editing one or more manuscripts. While writing a particular book, each author interacts with editor. But the author is supposed to submit his work to publisher always. In order improve competitiveness, the publication tries to employ a variety of authors who are specialist in more than one subject.

Draw the E-R diagram for above described scenario.

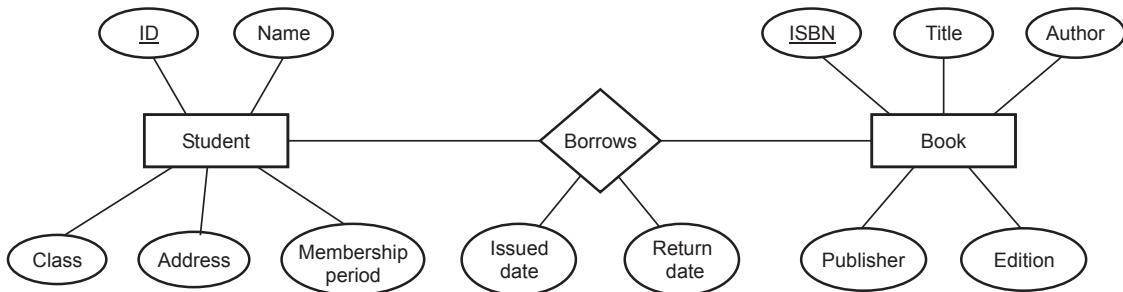
Solution :

The data modelling and entity relationship diagram helps the analyst to observe the data within the context of software application.

**Fig. 4.13.8**

Example 4.13.4 Prepare an E-R diagram for a simple library management system.

GTU : Summer-2012, Marks 7

Solution :**Fig. 4.13.9 ER diagram for library management system**

Example 4.13.5 Draw E-R diagram for university result system.

GTU : Summer-2014, Marks 7

Solution : Refer Fig. 4.13.10 on next page.

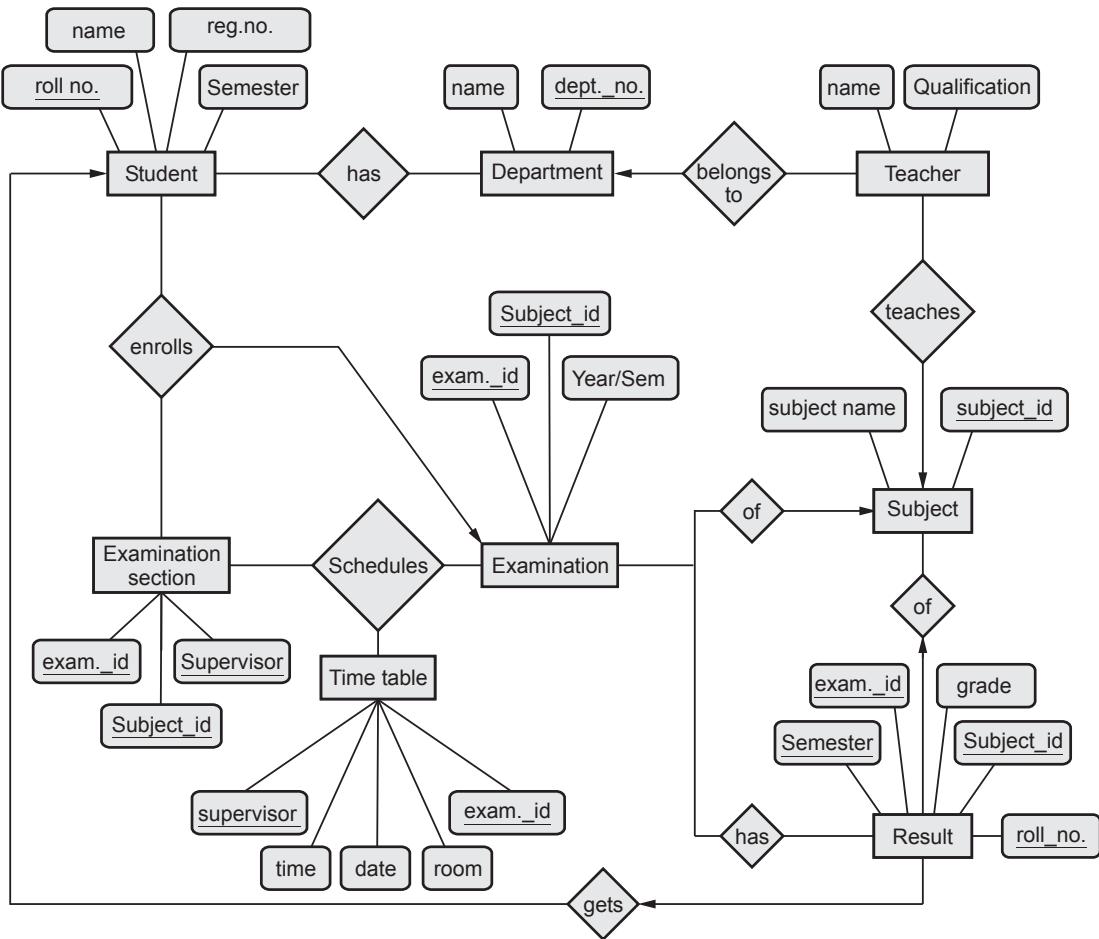


Fig. 4.13.10

Example 4.13.6 Draw E-R diagram for online shopping system.

GTU : Winter-2019, Marks 3

Solution :

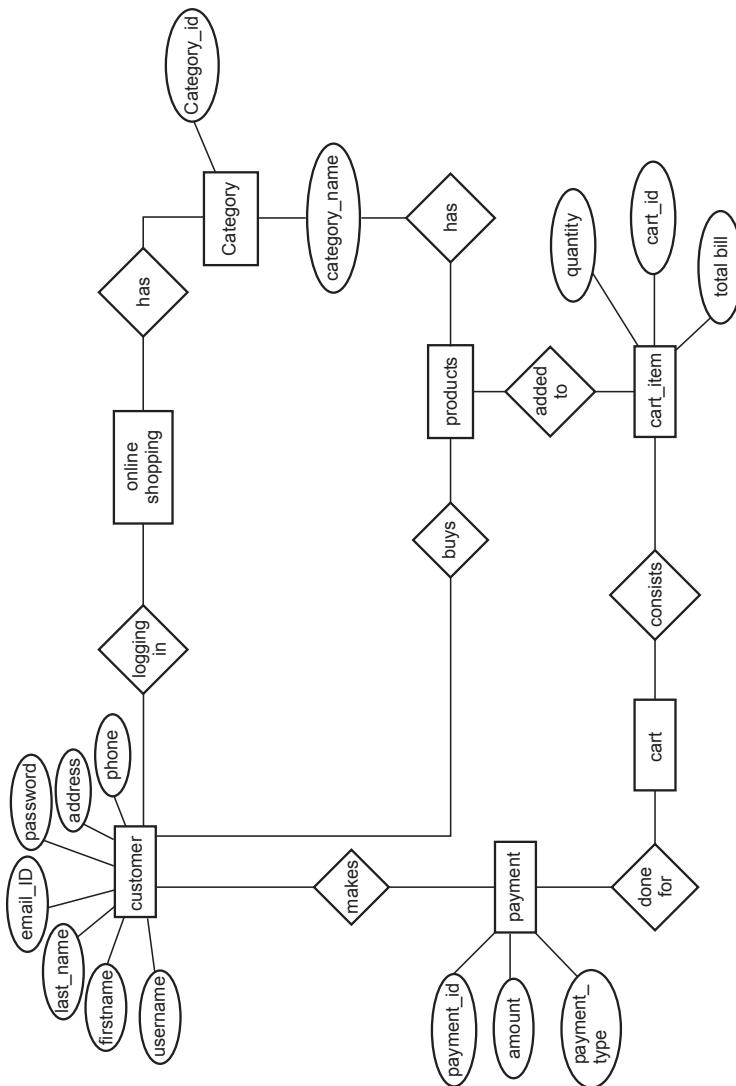


Fig. 4.3.11

Review Questions

- What is relationship ? Explain cardinality and modality with examples.

GTU : Summer-2015, Marks 7

- Explain different symbols of E-R diagrams. Draw E-R diagram for Library management system.

GTU : Summer-2015, Marks 7

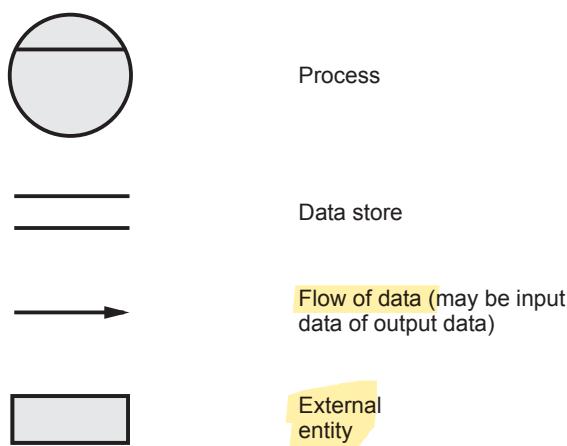
4.14 | Flow Oriented Modeling

GTU : May-2012, Dec.-2011, Marks 7

The flow oriented modelling is **down** by designing special kind of diagrams called data flow diagrams and control flow diagrams.

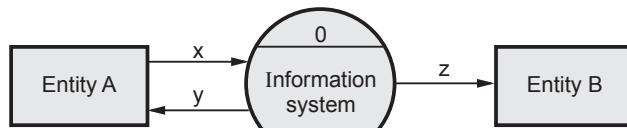
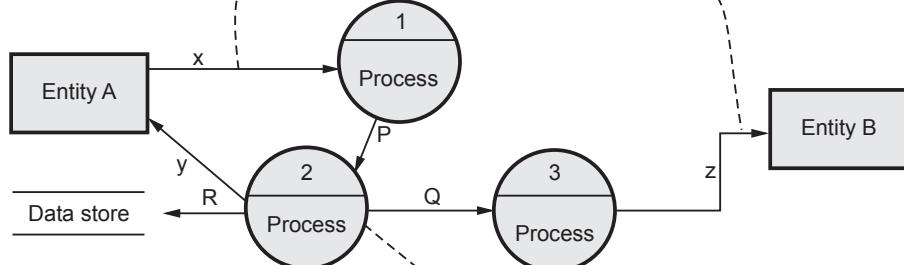
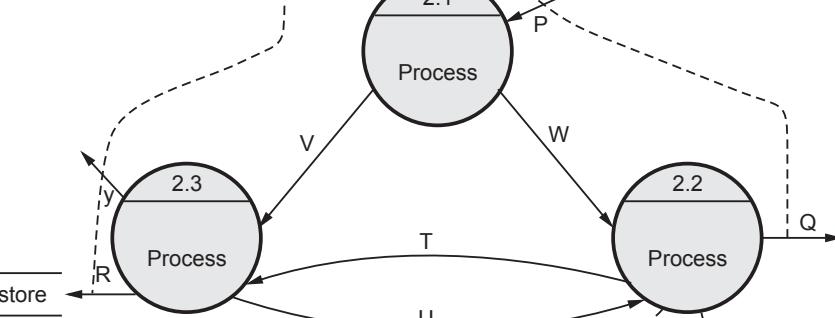
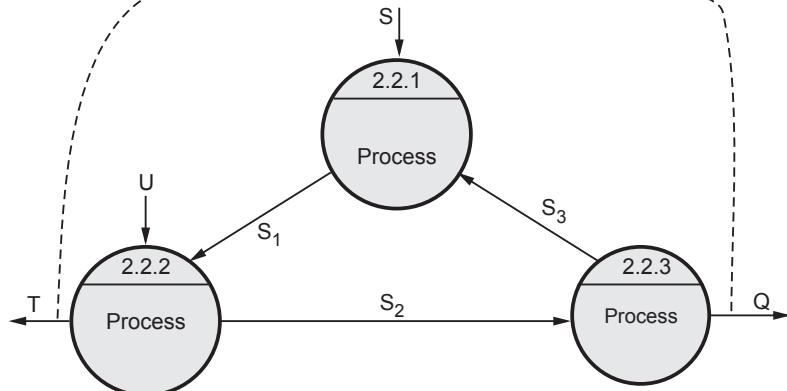
4.14.1 Data Flow Diagram

- The data flow diagrams depict the information flow and the transforms that are applied on the data as it moves from input to output.
- The symbols that are used in data flow diagrams are -
- The data flow diagrams are used to represent the system at any level of abstraction.
- The DFD can be partitioned into levels that represent increase in information flow and detailed functionality.
- A level 0 DFD is called as 'fundamental system model' or 'context model'. In the context model the entire software system is represented by a bubble with input and output indicated by incoming and outgoing arrows.
- Each process shown in level 1 represents the sub functions of overall system.
- The number of levels in DFD can be increased until every process represents the basic functionality.
- As the number of levels gets increased in the DFD, each bubble gets refined. The following figure shows the leveling in DFD. Note that the information flow continuity must be maintained.



4.14.1.1 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 -
 - Level 0 DFD i.e. Context level DFD should depict the system as a single bubble.
 - Primary input and primary output should be carefully identified.
 - While doing the refinement isolate processes, data objects and data stores to represent the next level.
 - All the bubbles (processes) and arrows should be appropriately named.
 - One bubble at a time should be refined.
 - Information flow continuity must be maintained from level to level.

Level 0 DFD**Level 1 DFD****Level 2 DFD****Level 3 DFD****Fig. 4.14.1 Levels of DFD**

- 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.

Rules for designing DFD

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

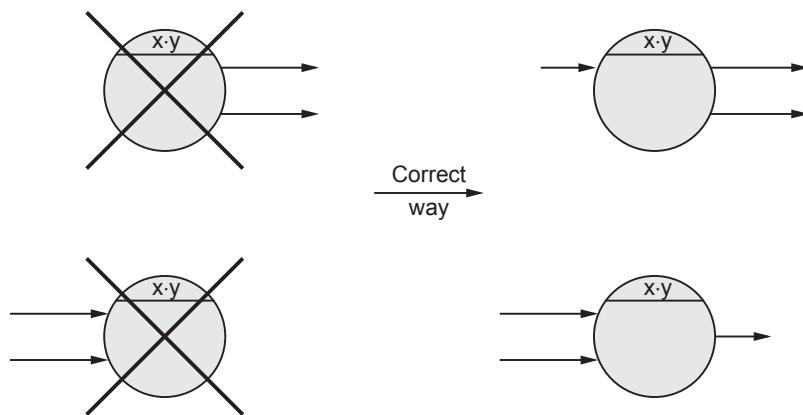


Fig. 4.14.2

- The verb phrases in the problem description can be identified as processes in the system.

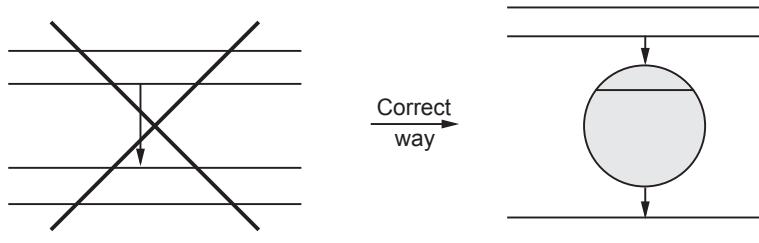


Fig. 4.14.3

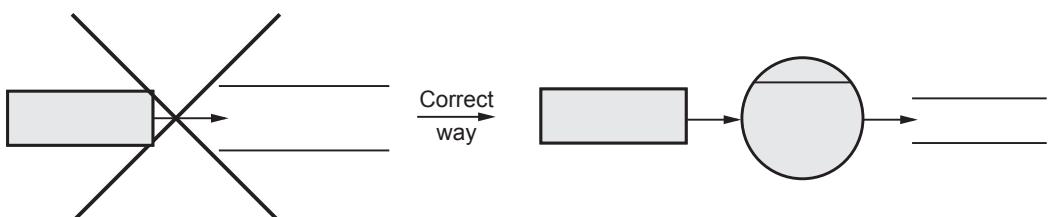


Fig. 4.14.4 Rules for DFD

3. There should not be a direct flow between data stores and external entity. This flow should go through a process.
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.

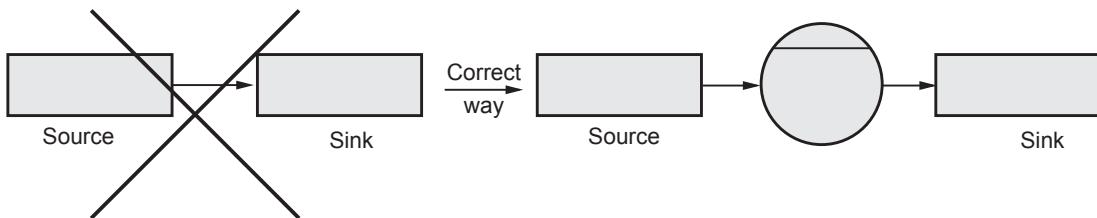


Fig. 4.14.5 Rule for DFD

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.

4.14.2 Examples

Example 4.14.1 DFD for library information system.

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.

Draw first level of DFD.

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 -

- The demanded book will be given to student. Hence **Book** will be the output.

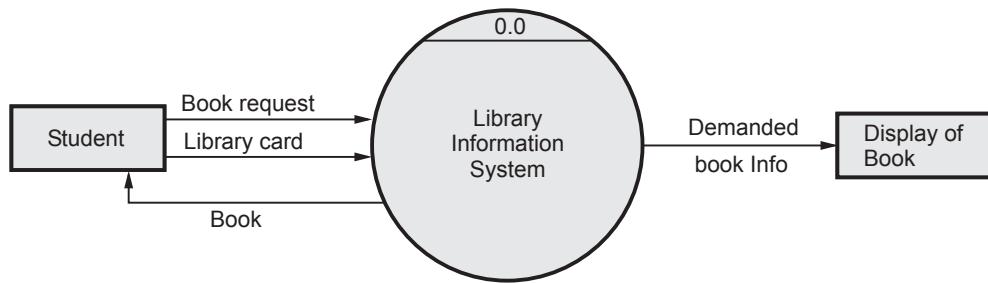


Fig. 4.14.6 Level 0 DFD (Context level DFD)

- The library information system should display **demanded book information** which can be used by customer while selecting the book.

Level 1 DFD

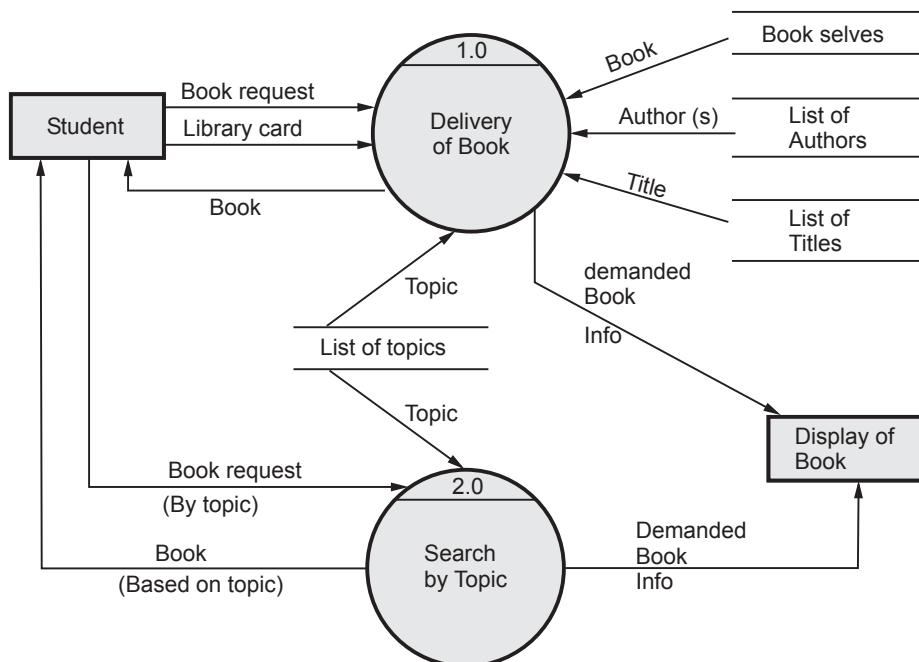


Fig. 4.14.7 Level 1 DFD

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**.

Level 2 DFD

Out of scope : The purchasing of new books/replacement of old books or charging a fine all these maintenance activities are not considered in this system.

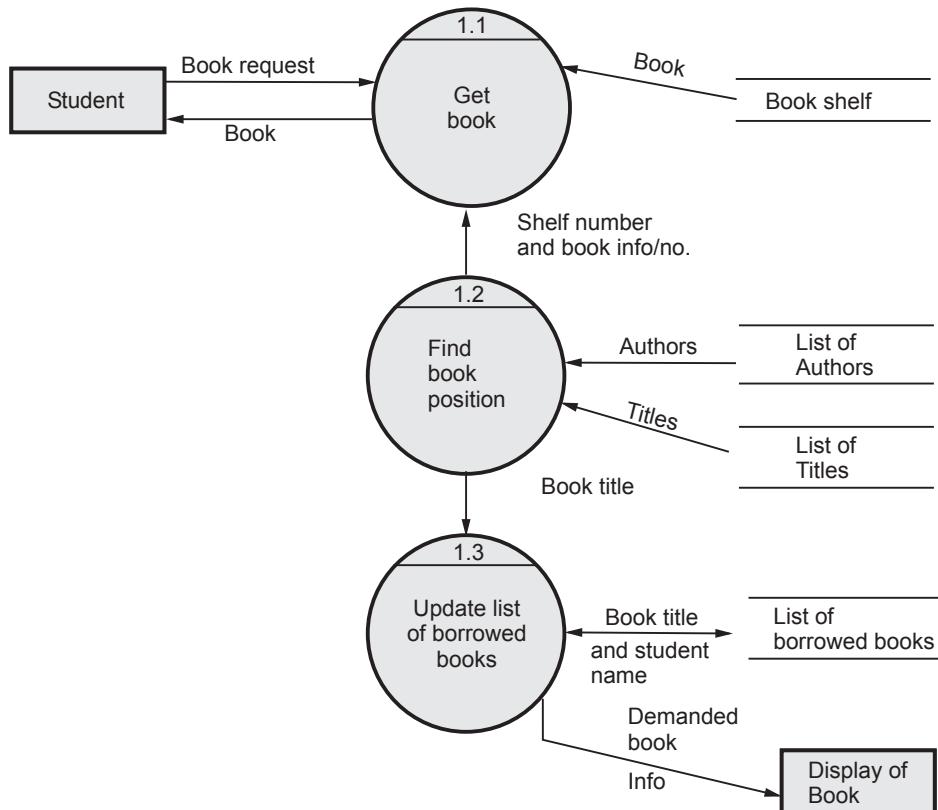


Fig. 4.14.8

Example 4.14.2 Draw DFD for food ordering system.

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**.

Solution :

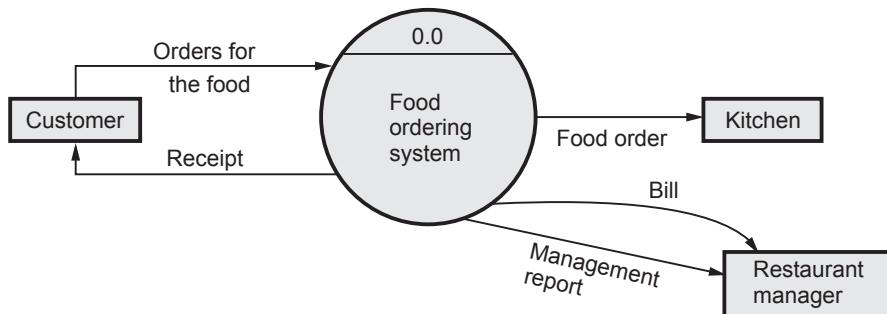


Fig. 4.14.9 Level 0 DFD (Context level DFD)

Level 0 DFD

In this level, the system is designed globally with input and output. The input to food ordering system are -

- As customer orders for the food. Hence food order is an input.

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

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.

Level 2 DFD : “Processing of an order” is shown in detail.

Level 3 DFD : In this DFD we will elaborate “Generate management report” activity in more detail. For generating management report we have to access sold items data and inventory data. Then aggregate both solid items data and inventory data. Total price of each item has to be computed. Then from these calculation a management report has to be prepared and given to the restaurant manager. These details can be shown in this DFD -

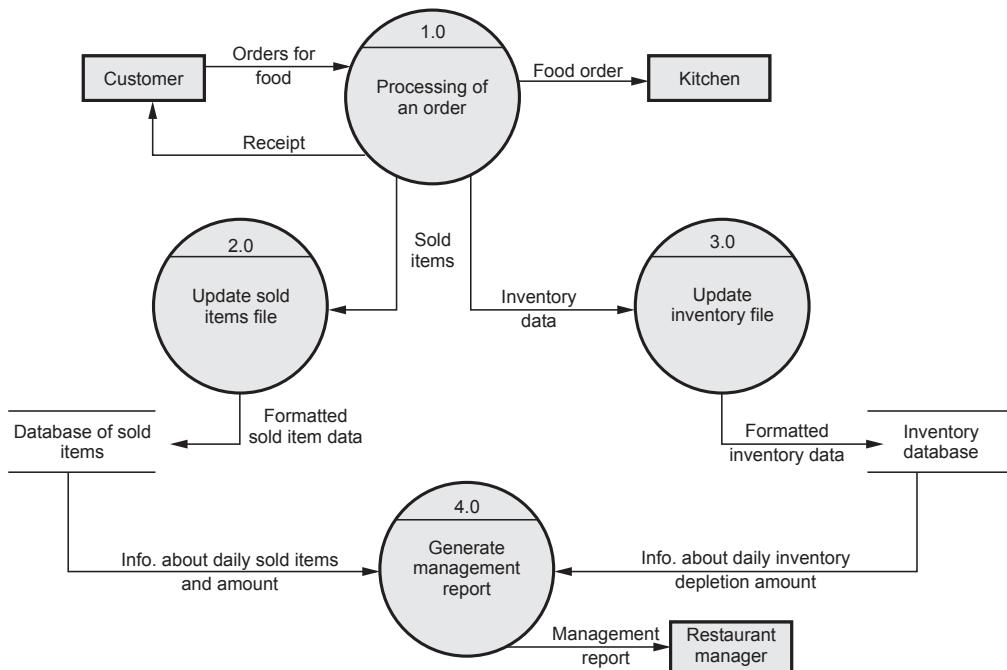


Fig. 4.14.10 Level 1 DFD

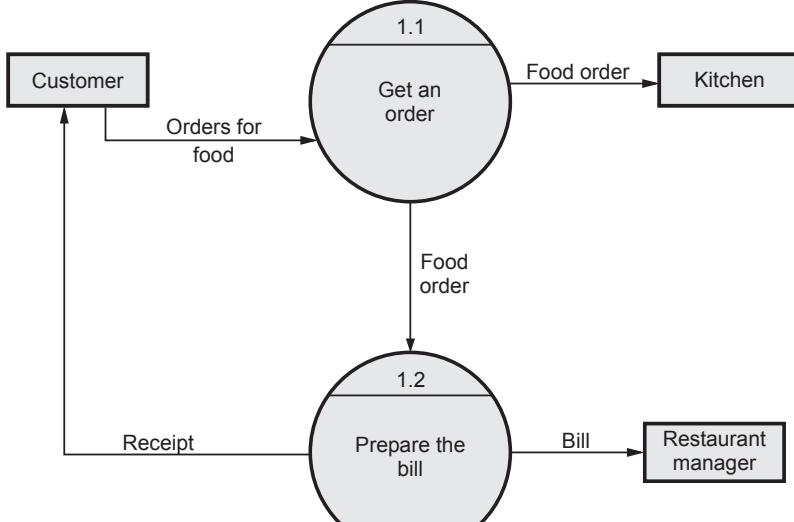


Fig. 4.14.11 Level 2 DFD

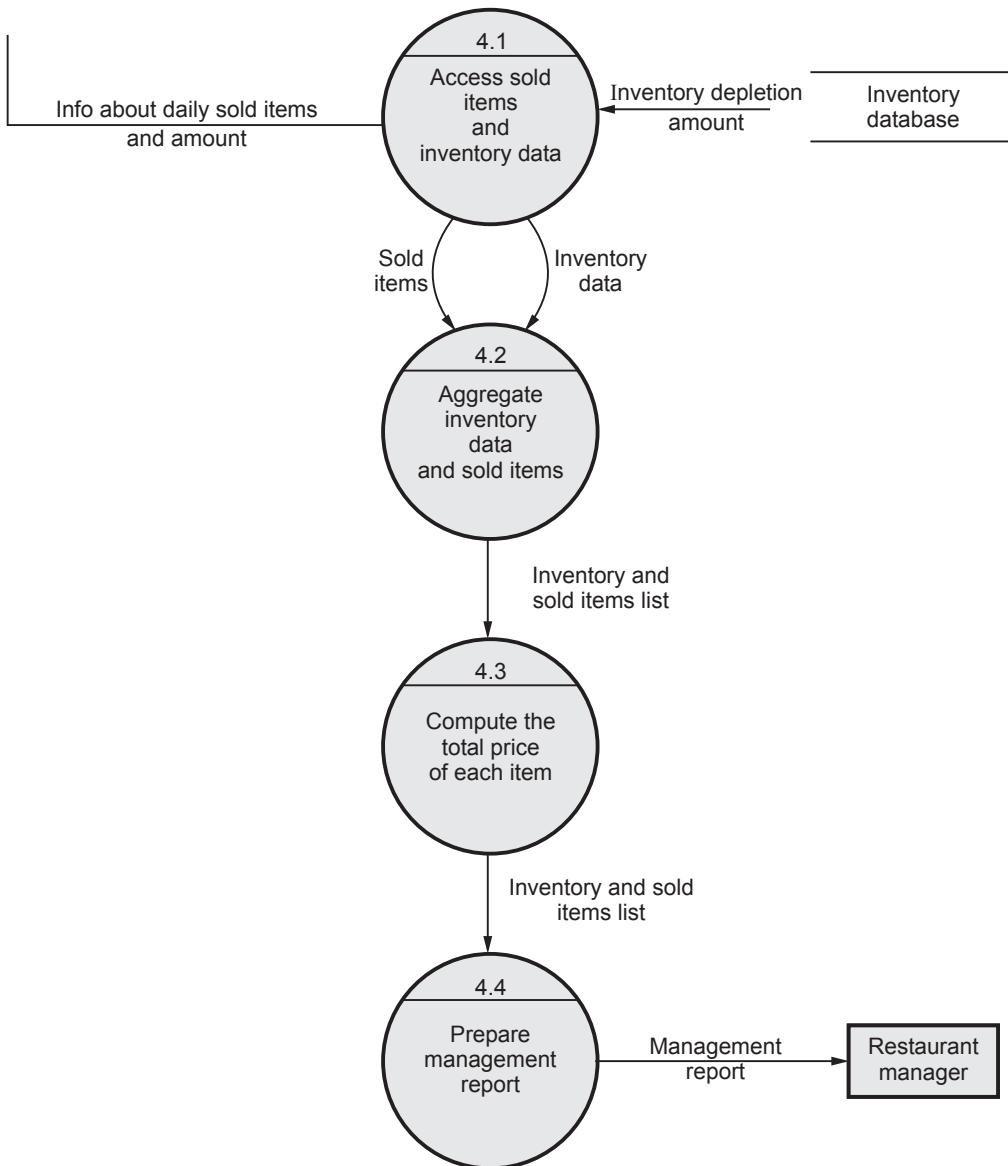


Fig. 4.14.12 Level 3 DFD

Example 4.14.3 Prepare a CFD for a Books order processing system. 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.

Solution : 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 below -

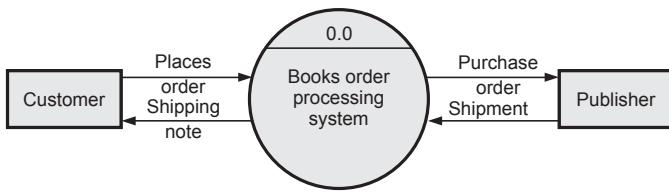


Fig. 4.14.13 Level 0 CFD

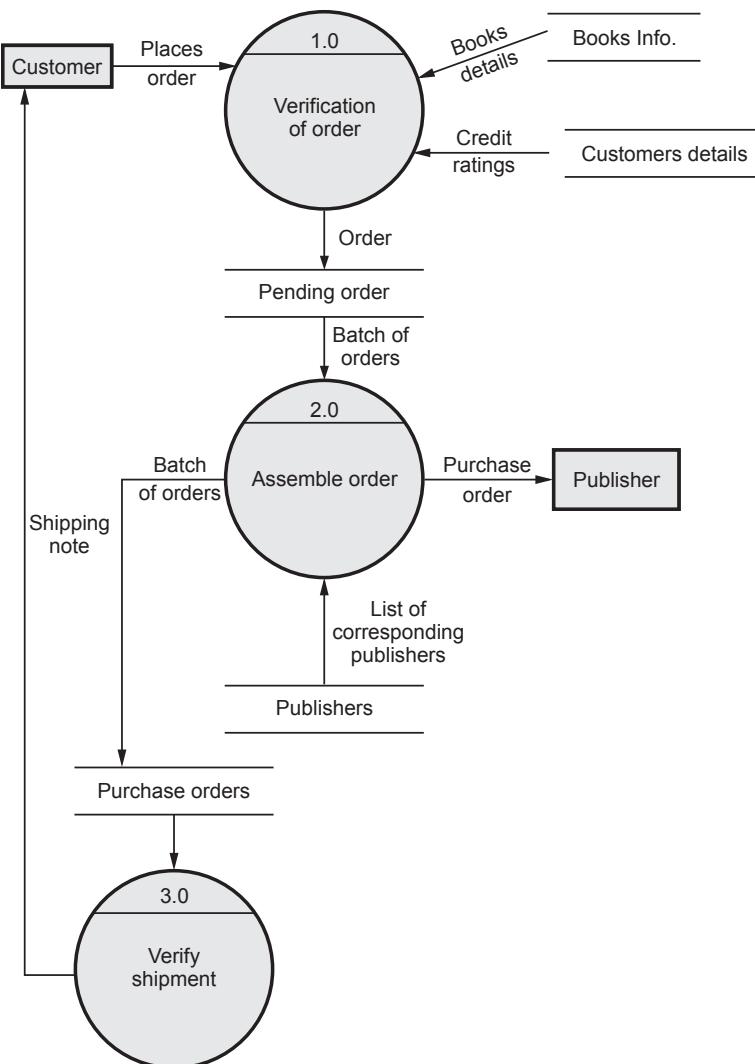


Fig. 4.14.14 Level 1 CFD

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.

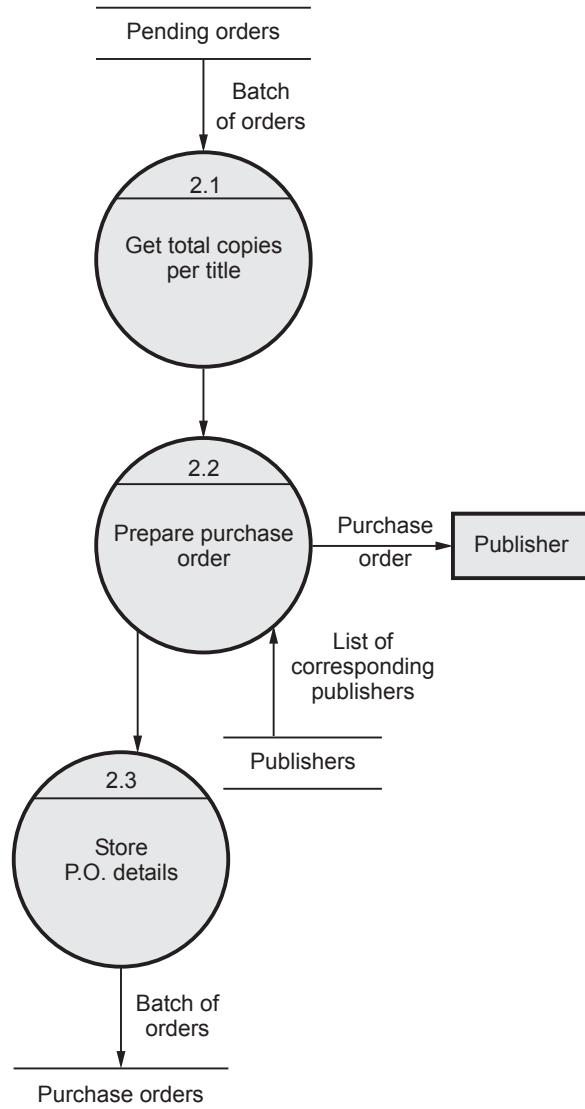


Fig. 4.14.15

Example 4.14.4 In a hotel reservation system, a **customer** can make online booking for a hotel, by specifying the accommodation 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.

Draw Level 0, Level 1, Level 2 DFDs.

Solution : The system then gives these booking details to the corresponding **hotel**. Design DFD for this system.

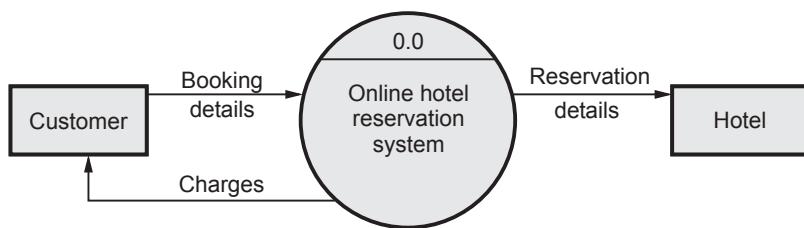


Fig. 4.14.16 Level 0 DFD

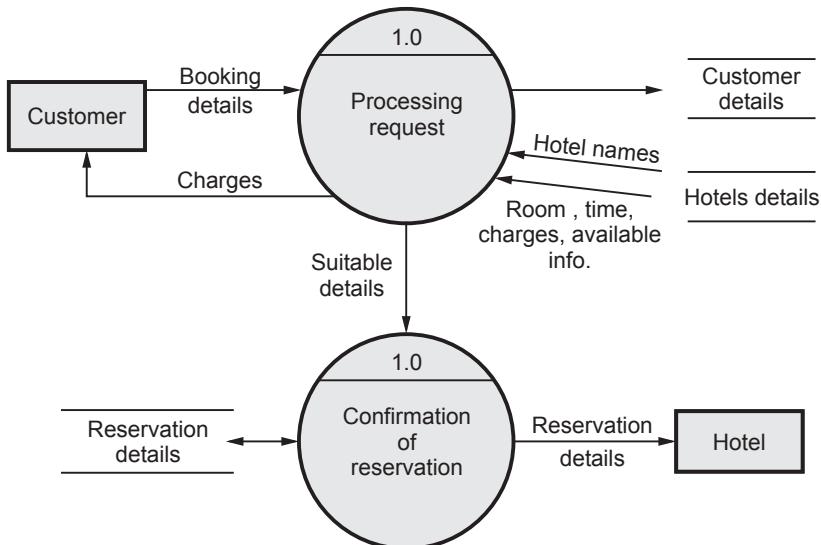


Fig. 4.14.17 Level 1 DFD

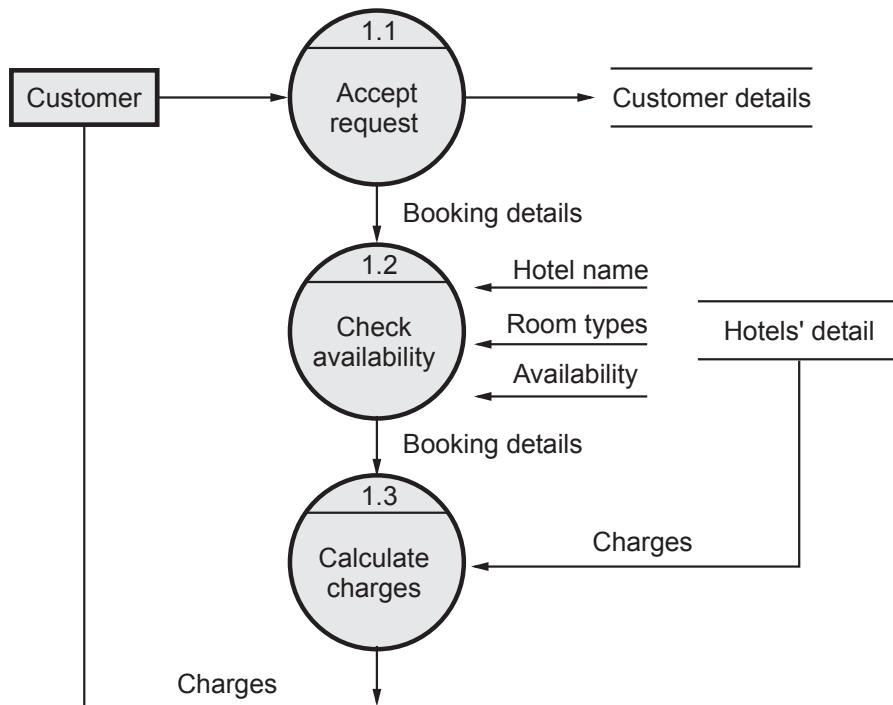


Fig. 4.14.18 Level 2 DFD

Example 4.14.5 Based on your experience with a bank ATM draw a DFD modelling the processing involved when customer withdraws cash from the machine.

GTU : Summer-2012, Marks 7

Solution : The DFD for ATM system can be drawn with level 0 DFD and level 1 DFD. The level 0 DFD is the context level DFD in which only inputs and outputs that are interacting with the system are given.

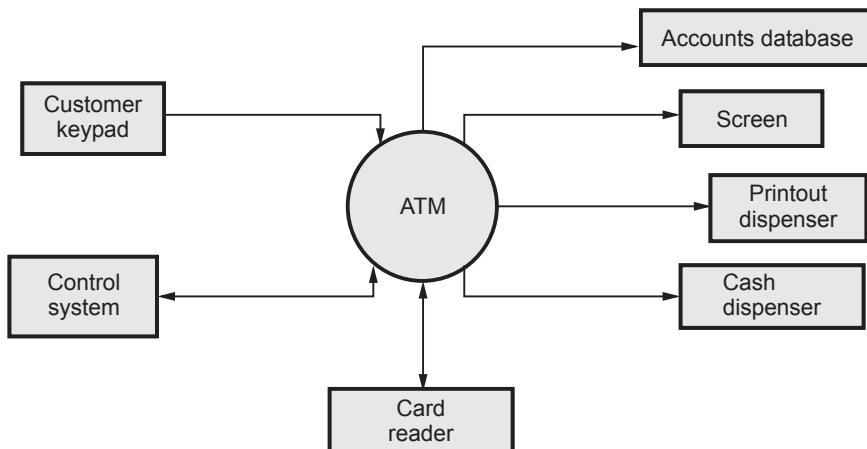


Fig. 4.14.19 Level 0 DFD

More detailing is done in level 1.

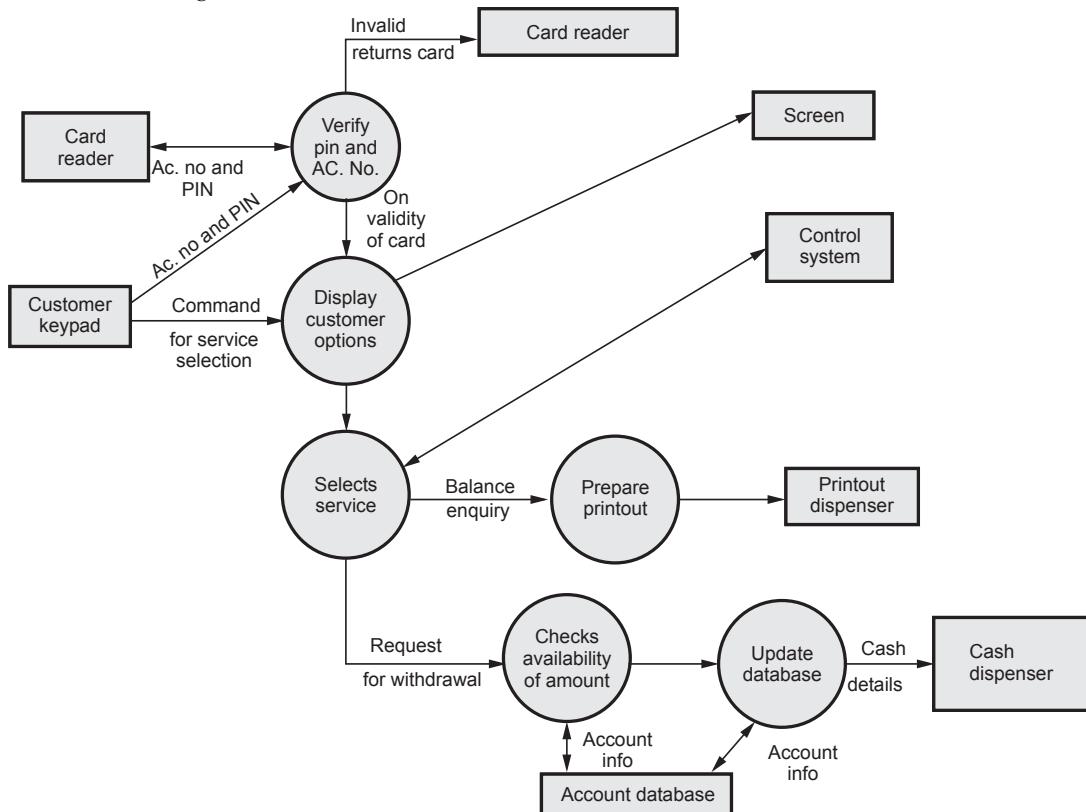


Fig. 4.14.20

Example 4.14.6 Draw level 0 DFD and level 1 DFD for railway reservation system. Also write the data dictionary for the same.

Solution : **Problem description :** For an online railway reservation system, passenger can make online booking for the seats, by specifying the requirements. These requirements are - type of reservation A/C coach, non A/C coach, two tier or three tier number of seats, name of the train, start and destination of journey. The system then checks for availability of such requirements. If such a reservation is possible then system generates **availability** of reservation. The passengers **checks** for availability and the **charges are then calculated** for the selected requirement, and these are acknowledged to the passenger. If the passenger is satisfied then he **confirms** the reservation.

The required DFD can be drawn in levels as follows -

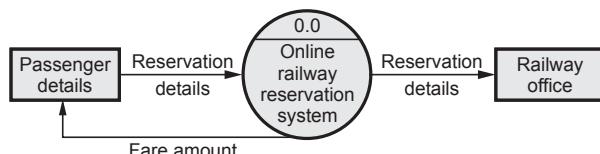


Fig. 4.14.21 Level 0 DFD

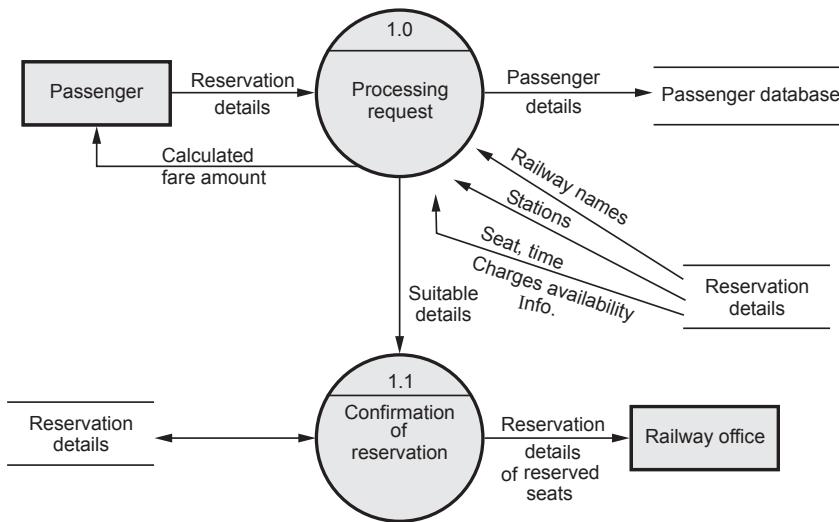


Fig. 4.14.22 Level 1 DFD

Data Dictionary

Name : Reservation Details or Ticket

Aliases : None

Where used/how used : For reserving seat details of passenger are used as input. Reservation details of reserved seat is output.

Description :

Passenger name = First name + Middle name + Surname.

Age = In number of years.

Sex = Male/Female.

Reservation number = Boggue number + Coach number

Starting location = Name of city

Destination location = Name of city

Train details = Train number + Name of the train + Up or down
+ Date and time of departure + Time of arrival

Fare = Total amount of fare for entire journey.

Example 4.14.7 Consider a photocopying machine operated by software. Draw level 0, 1 and 2 DFD for this software.

Solution : Assumptions made for software of photocopy machine : The photocopy machine will be controlled by some commands such as START, STOP or PAUSE. If user gives START command then required number of photocopies can be made by reloading

papers. If paper gets jammed then diagnostic activities must be carried out and then reload the paper and then make the copies. The input to this system will be user commands and output will be photocopies.

The DFD for such a software will be -

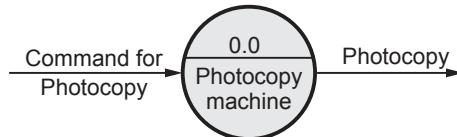


Fig. 4.14.23 Level 0 DFD

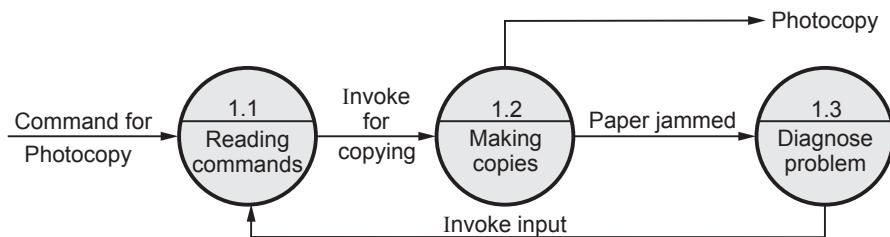


Fig. 4.14.24 Level 1 DFD

Now in level 1 DFD, we will elaborate the process 'making copies'.

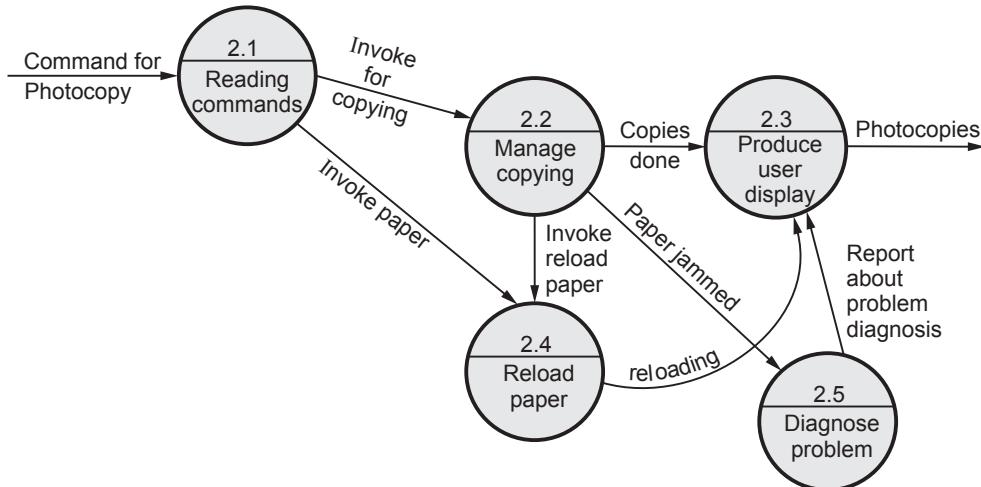


Fig. 4.14.25 Level 2 DFD

The State Transition Diagram (STD) can be drawn as follows -

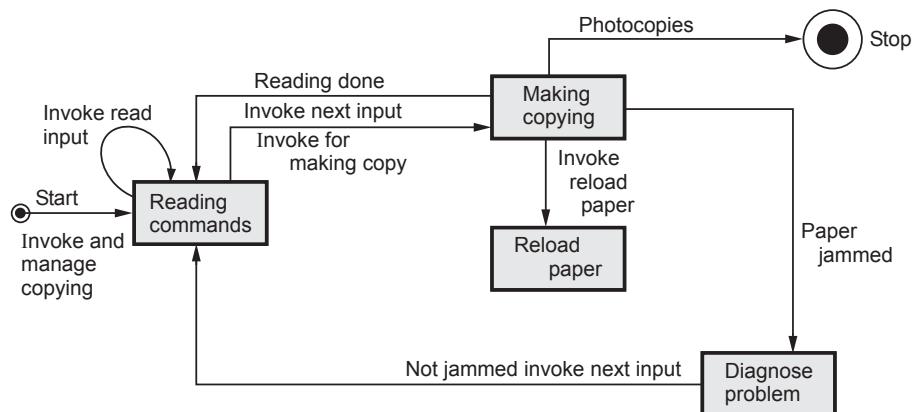


Fig. 4.14.26 STD

Example 4.14.8 Draw the data flow diagram upto level 1, for a 'temperature monitoring system' in an intensive care unit of a hospital.

Solution : DFD Level 0

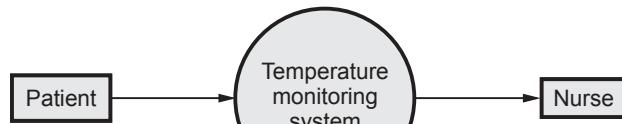


Fig. 4.14.27 DFD Level 0

DFD Level 1

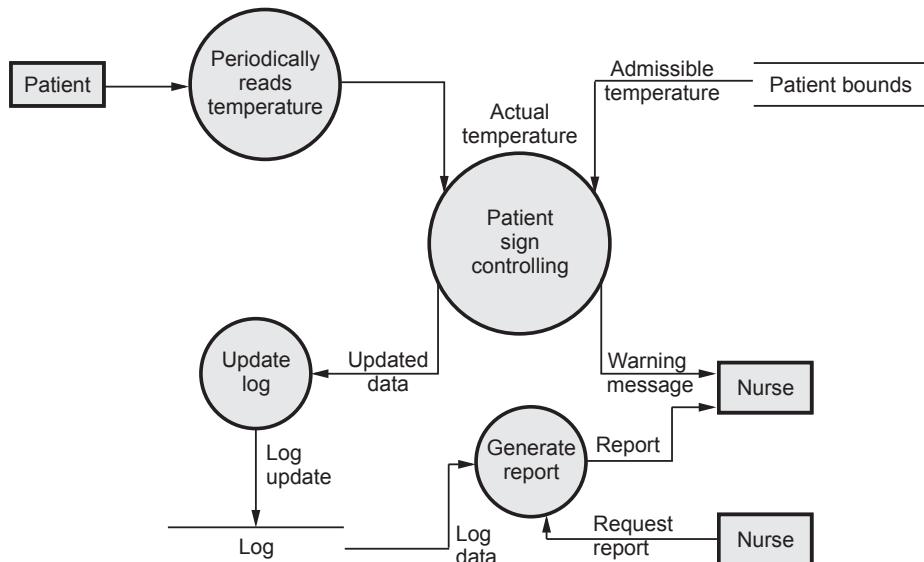


Fig. 4.14.28 DFD Level 1

Example 4.14.9 Draw and explain context level, level 1 and level 2 DFD for college gathering.

Solution In this system whole system is represented with the help of input processing and output. The input can be

1. Students
2. This system should show output as gathering.

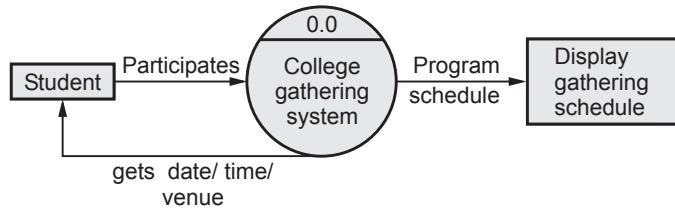


Fig. 4.14.29 Level 0 DFD

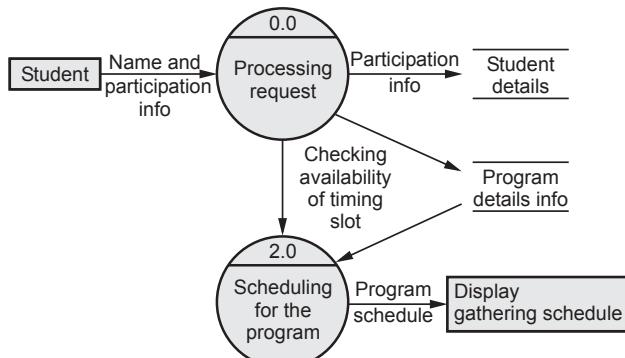


Fig. 4.14.30 Level 1 DFD

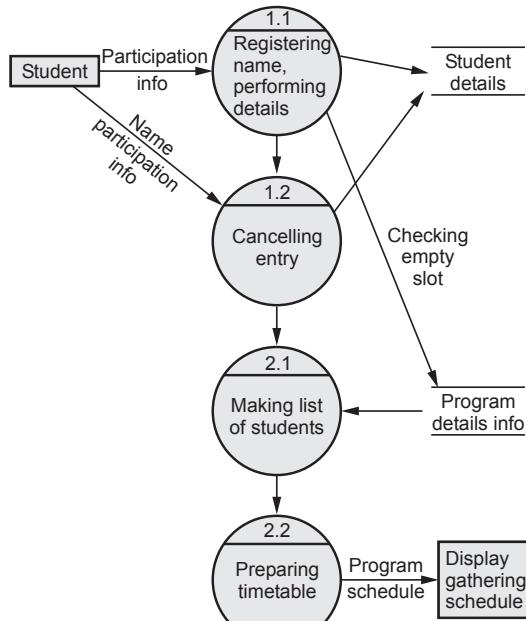


Fig. 4.14.31 Level 1 DFD

Example 4.14.10 Draw a DFD for a vehicle renting system.

Solutin : A context diagram and A Level 1 diagram for the Car Rental System, is as given below -

DFD Level 0 :

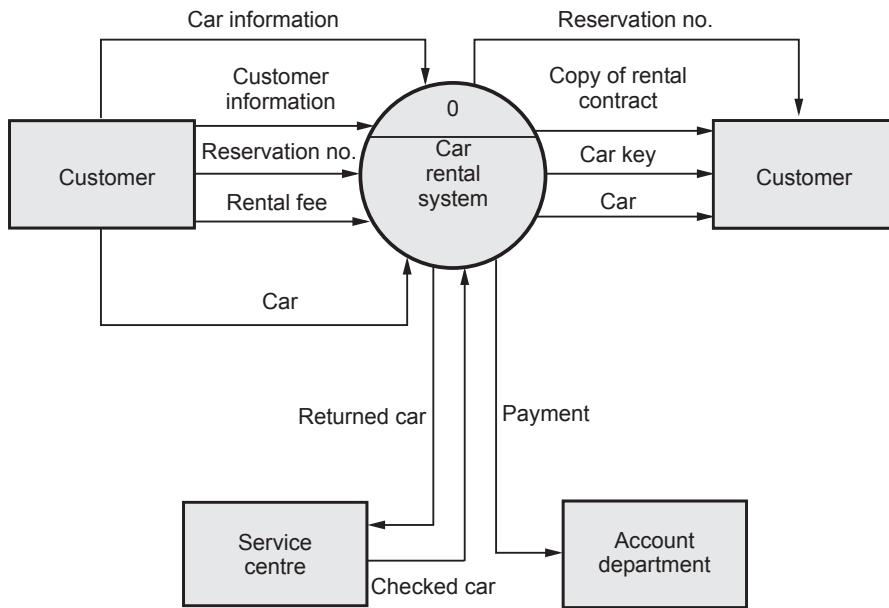


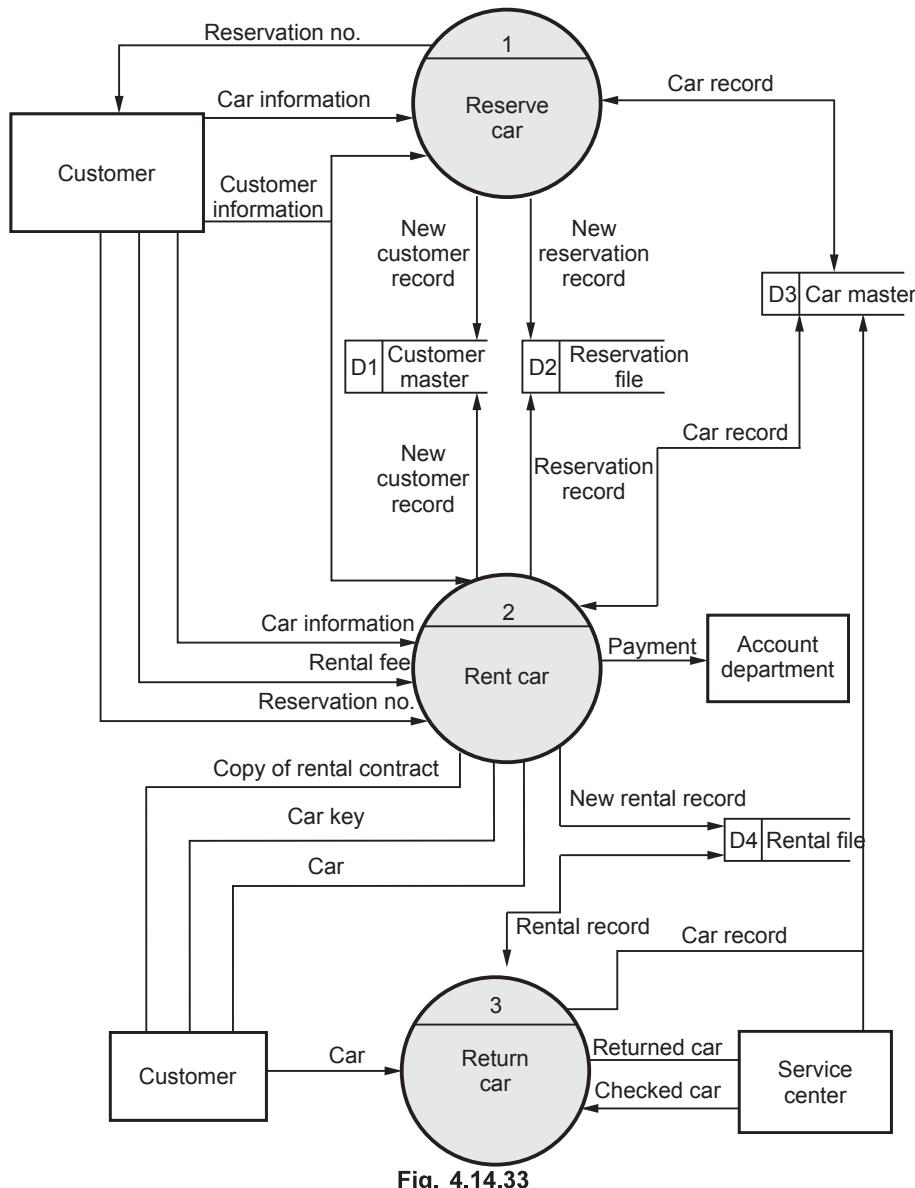
Fig. 4.14.32

DFD Level 1 : See Fig. 4.14.33 on next page.

4.14.3 Difference between DFD and ER

Sr. No.	DFD	ER
1.	The DFD stands for data flow diagram.	The ER stands for Entity Relationship diagram.
2.	The DFD describes the system's functions and processes.	The Entity relationship describes the data.
3.	A data flow diagram models the flow of information through a system	An entity relationship diagram models people, objects, places, and events which are called entities for which data is stored in a system.
4.	It basically describes how data enters a system, is transformed in a system, and is stored in a system.	The ERD also shows the relationships and cardinality between the system's entities.
5.	DFD model is a multi-leveled representation of the system. The first level is a kind of level in which the abstract information is represented. This level is called context diagram. Then system is represented using multiple levels.	The ER diagram is a detailed representation of the system data. It describes the relationship between the data.

6.	The DFD processes are represented using circles, ovals and rectangles. The arrows represent the flow of information. The parallel lines represent the storage for the data.	In ER diagram, entities are represented using rectangular boxes, the relationship between the entities are represented by diamonds. Lines and standard notations are used to represent the cardinality.
7.	Rule: There must be at least one data flow entering into and leaving the process or store. All data must go through a process. All processes within the system must be linked to another process or data store.	Rule: In ER diagram, all entities must represent the set of similar things. The data should be represented by the ER diagram.



Review Questions

1. Explain the data flow model with example and diagram.
2. Explain difference between DFD and diagram.

GTU : Winter-2011, Marks 7

4.15 Behavioral Modeling

GTU : Summer-2012, Marks 7

The dynamic behaviour of the system can be represented by creating the behavioural model of the system. Basically the behavioural model represents how software will respond to external events or stimuli. Following steps need to be followed while creating behavioural model -

1. Evaluate use-case diagrams for understanding the sequence of interactions between the system.
2. Identify the events for interactions and co-relate the classes with these events.
3. Create a sequence for each use-case.
4. Create a state diagram for the system.
5. Finally review behavioural model.

4.15.1 State Representation

The state transition diagram is basically a collection of states and events. The events cause the system to change its state. The state transition diagram also represents what actions are to be taken on occurrence of particular event.

	Initial state	It indicates the starting state of the system.
	Final state	This is a special state indicating end of the activity.
	Simple state	A state with no substructure.
	Composite state	A state which occurs with two or more concurrent states with one active state at a time is a composite state.

For example : Following state chart is for ATM system when the customer performs transaction using ATM card.

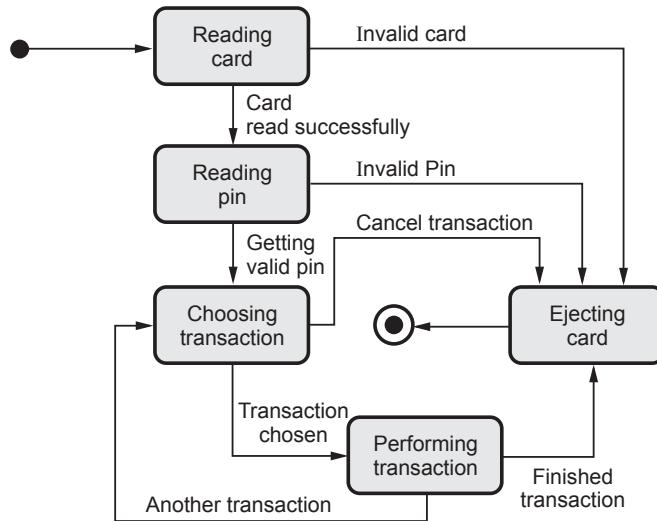


Fig. 4.15.1 Transaction

Example 4.15.1 Prepare a state diagram for microwave oven showing all states of it.

GTU : Summer-2012, Marks 7

Solution :

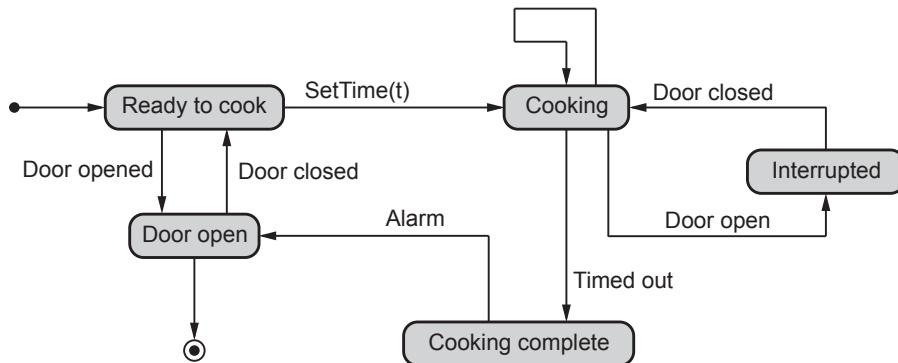


Fig. 4.15.2

4.15.2 Sequence Diagram

In the sequence diagram how the object interacts with the other object is shown. There are sequences of events that are represented by a sequence diagram.

It is a time-oriented view of the interaction between objects to accomplish a behavioural goal of the system.

Various notations used in sequence diagram are

	User - who is responsible for activating various events.
	An active object who takes part in various operations.
	It represents the life line i.e. a time span of the object.
	Active procedure on which object is active.
	To interact two objects this kind of association is shown. Along with this association the message is given
	Destruction of object.

Example 4.15.2 Draw a sequence diagram for ATM system.

Solution : (See Fig. 4.15.3 on next page.)

4.16 Software Requirements Specification(SRS)

GTU : Winter-2011, 2017, Marks 7

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.

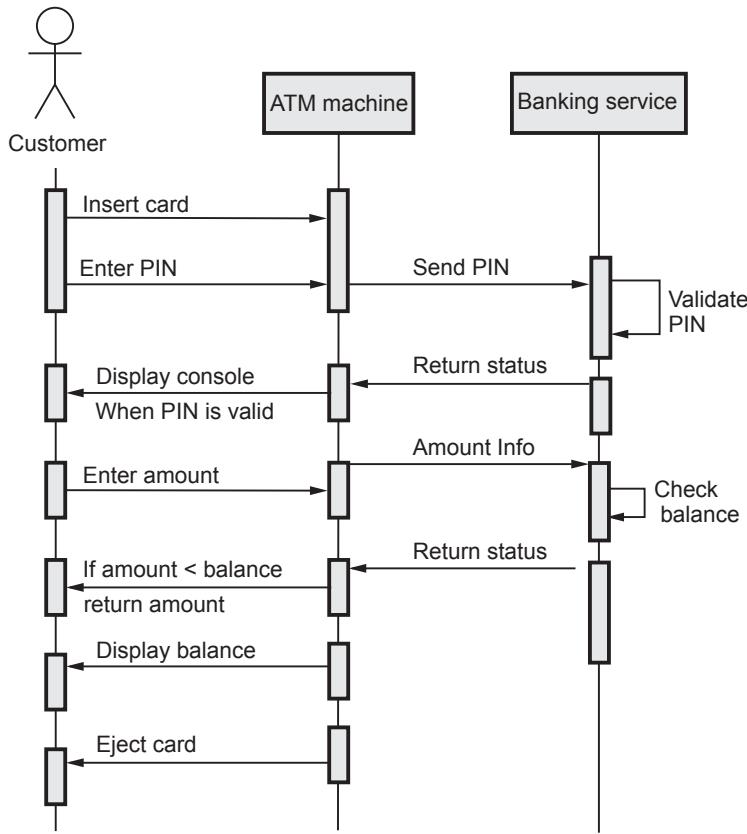
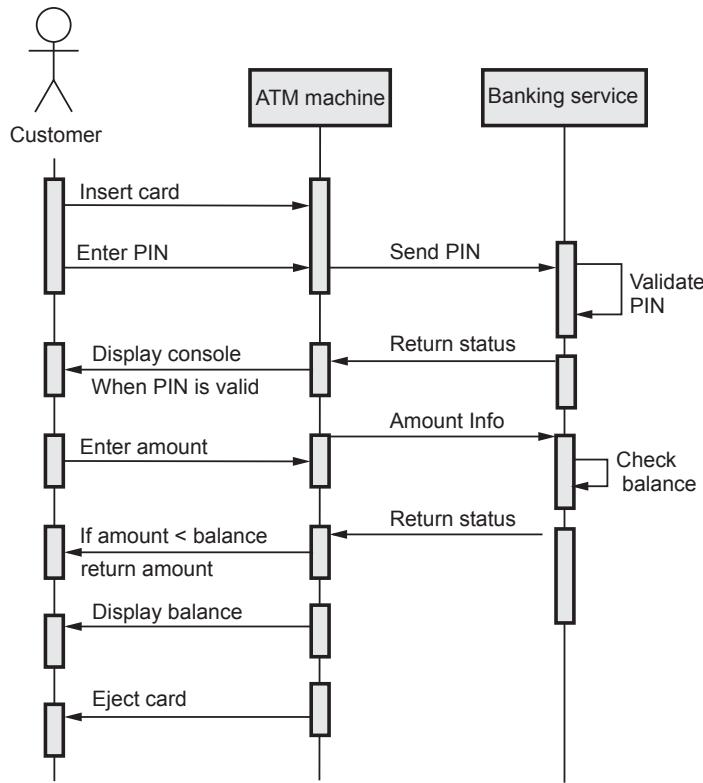


Fig. 4.15.3 Sequence diagram

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.

**Fig. 4.16.1 Sequence diagram**

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

Document Title

Author(s)
Affiliation
Address
Date
Document Version

1. Introduction

1.1 Purpose of this document

Describes the purpose of the document.

1.2 Scope of this document

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

1.3 Overview

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

2. General Description

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

3. Functional Requirements

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

- Short, imperative sentence stating highest ranked functional requirement.

1. Description

A full description of the requirement.

2. Criticality

Describes how essential this requirement is to the overall system.

3. Technical issues

Describes any design or implementation issues involved in satisfying this requirement.

4. Cost and schedule

Describes the relative or absolute costs of the system.

5. Risks

Describes the circumstances under which this requirement might not able to be satisfied.

6. Dependencies with other requirements

Describes interactions with other requirements.

7. Any other appropriate

4. Interface Requirements

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

4.1 User Interfaces

Describes how this product interfaces with the user.

4.1.1 GUI

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

4.1.2 CLI

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

4.1.3 API

Describes the application programming interface, if present.

4.2 Hardware Interfaces

Describes interfaces to hardware devices.

4.3 Communications Interfaces

Describes network interfaces.

4.4 Software Interfaces

Describes any remaining software interfaces not included above.

5. Performance Requirements

Specifies speed and memory requirements.

6. Design Constraints

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

7. Other Non Functional Attributes

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

7.1 Security

7.2 Binary Compatibility

7.3 Reliability

7.4 Maintainability

7.5 Portability

7.6 Extensibility

7.7 Reusability

7.8 Application Compatibility

7.9 Resource Utilization

7.10 Serviceability

... others as appropriate

8. Operational Scenarios

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

9. Preliminary Schedule

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

10. Preliminary Budget

This section provides an initial budget for the project.

11. Appendices

11.1 Definitions, Acronyms, Abbreviations :

Provides definitions terms, and acronyms, can be provided.

11.2 References

Provides complete citations to all documents and meetings referenced.

4.16.1 Characteristics of Good SRS

Following are some characteristics of a good SRS -

1. Correct

The SRS must be correct. That means all the requirements must be correctly mentioned, or the requirements must be realistic by nature. For instance : While developing a word processing software, if there is a requirement for spell check facility and if software cannot find the spelling errors from the document, then that means requirement is incorrect.

2. Complete

To make the SRS complete, it should be specified what a software designer wants to create a software. The SRS is said to be complete only in following situations -

- 1) When SRS consists of all the requirements related to functionality, performance, attributes, design constraints or external interfaces.
- 2) When labels and corresponding references are mentioned for all the figures, diagrams and tables in the SRS.
- 3) When expected responses to the input data is mentioned by considering validity and invalidity of an input.

3. Unambiguous

When requirements are understood correctly then only unambiguous SRS can be written. Unambiguous specification means only one interpretation can be made from the specified requirements. In other words, there should be an unique interpretation of each statement in SRS. If for particular term there are multiple meanings then, those terms should be mentioned in glossary with proper meaning. The requirements should not be mentioned in the same manner.

After preparing SRS, it should be reviewed by third party. Normally SRS is written in some natural language like SRS. The SRS can be examined with the help of language processors, so that lexical, syntactic and semantic errors can be exposed.

4. Consistent

If there are not conflicts in the specified requirements then SRS is said to be consistent. Three types of conflicts that may occur in a SRS -

- i) **Logical or temporal conflict** : When one requirement specifies that event X should occur before event Y and if another requirement specifies that event Y should occur before event X.
- ii) **Characteristics conflicts of real-world object** : If one requirement suggests to make use of "radio button" and other specifies the "Check box button", then it represents conflicting characteristics.
- iii) **Two different descriptions about the same real world object** : If one requirement is specifying "Enter" and other specifies "submit" then it describes one and the same action.

5. Stability

In SRS, it is not possible to specify all the requirements. The SRS must contain all the essential requirements. Each requirement must be clear and explicit.

6. Verifiable

The SRS should be written in such a manner that the requirements that are specified within it must be satisfied by the software.

For instance - "The GUI should look good". This requirement is not verifiable because one cannot specifically define "what is mean by good ?".

7. Traceable

If origin of requirement is properly given or references of the requirements are correctly mentioned then such a requirement is called as traceable requirement. Various types of traceability are -

- i) **Forward Traceability** : Each requirement is referred in the SRS document by its unique name or reference number.
- ii) **Backward Traceability** : If the reference to the requirement is mentioned in earlier document, then it is backward traceability.

4.16.2 Example of SRS

Software Requirements Specification For Attendance Maintenance System

Prepared by Anjali

December 1, 2007

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	Maintenability	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.

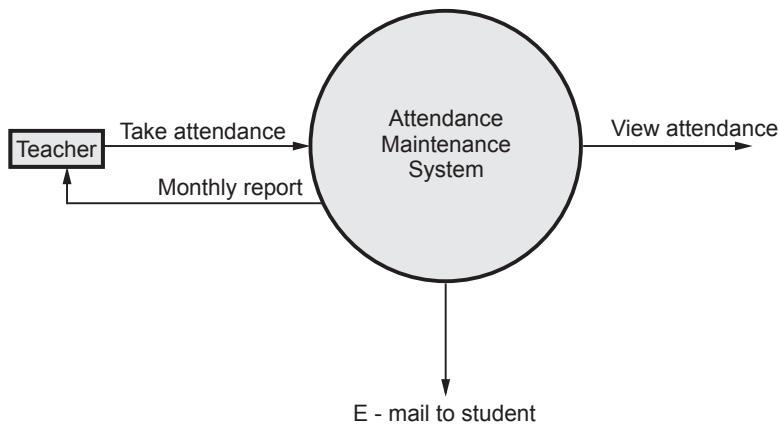
1.3 Overview

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 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.

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.

**Fig. 4.16.2**

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_15* is opened then we may view attendance record of SE COMPUTER class in year 2015 and a record for semester-I can be viewed.

GUI 3 : The display of record should be (See Fig. 4.16.3)

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.

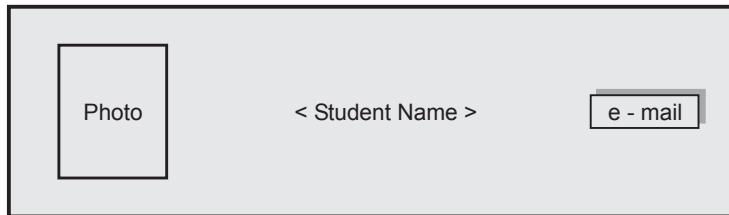


Fig. 4.16.3 Student record

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 cannot be guaranteed.

7.3 Availability

The system should be available during college hours.

7.4 Maintenability

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

7.5 Reusability

The same system will be used in each new semester.

8. Operational Scenarios

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

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

9. Preliminary Schedule

The system has to be implemented within 6 months.

Example 4.16.1 A library lends books and magazines to member, who is registered in the system. Also it handles the purchase of new titles for the library. Popular titles are bought into multiples copies. Old books and magazines are removed when they are out of date or in poor condition. A member can reserve a book or magazine that is not currently available in the library, so that when it is returned or purchased by the library, that person is notified. The library can easily create, replace and delete information about the titles, members, loans and reservation in the system.

Prepare software requirement specification and use case diagram.

GTU : Winter-2011, Marks 7

Solution :

Library Management System

Prepared By: Ms. ABC

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.
11. The system should generate daily or monthly report for the stock.
12. The system should allow addition of book/magazine records if new books are purchased.
13. The system should allow deletion of book/magazine records if the books are old and in poor condition.

4. Interface Requirements

4.1 GUI

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

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

GUI 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 student 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.

Use case diagram for library management system : Refer Fig. 4.11.3 from section 4.11.

Example 4.16.2 Software is to be developed for hotel management system in which information is provided for all type of activities conducted in hotel. The major users of the system are hotel staff, people who stay in the hotel and people who visit the restaurant. Information for the billing system, hotel account management, staff salary, hotel menu information, hotel room information is provided by software.

Prepare software requirement specification and use case diagram.

GTU : Winter-2011, Marks 7

Solution :

Hotel Management System

Prepared By: Mr.XYZ

1. Introduction

1.1 Purpose

This document gives the detail functional and non functional requirements for Hotel 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 hotel management system will automate the major hotel operations. Various subsystems will be Reservation and booking system, Billing and Accounts system, general management and report generation systems.

1.3 Definitions, Acronyms and Abbreviations

- HMS - Hotel management System
- SRS - Software Requirement Specification
- GUI - Graphical User Interface
- Stakeholder - The person who will participate in this system. For example Customer, accountant, Hotel manager, administrator.

1.3 Overview

This system provides an easy solution for the customers to reserve the accommodations in the hotel and for the administrator to automate all types of hotel related activities such as stock maintenance, billing and so on.

2. Overall Description

The Hotel management system is developed for handling the activities for various users such as Hotel manager, receptionist, customer, hotel staff and so on.

2.1 Product Perspective

This is a standalone system.

2.2 Hardware Interfaces

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

2.3 Software Interfaces

There are various databases in the system. These databases can be created in Oracle or MySQL. The hotel rooms and customer information is maintained by the HMS.

The room database include the room number, type, status(occupied or vacant). The customer database will include information such as first name, last name, address, phone number, number of occupants, assigned room, rate of the room, credit card number, expects date and time for check in and check out, advance payment made, Total charges and feedback.

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 customer.
- 2 The system should display the rooms that are available.
- 3 The system should allow for room selection from the available list.
- 4 The system should record the expected check in and check out date and time.
- 5 The system should allow the modification in the customer information without reentering the entire information.
- 6 The system should display the mode of payment and validity.
- 7 The system should allow to cancel the reservation.
- 8 The system should calculate and display the additional charges for the services used by the customer.
- 9 The system shall allow to assign password to the users.
10. The system should record the feedback of the customer.
11. The system should keep track of the purchased items stock.
12. The system should generate daily or monthly report for the stock.
13. The system should allow addition of information such as room numbers, status, type of the room, room rate, customer profile and so on.
14. The system should manage the payroll for the hotel staff.
15. The system should generate report for the payroll.

4. Interface Requirements

4.1 GUI

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

GUI 2 : The main menu will be displayed for Reservation, Cancellation, maintenance and Report generation.

Hotel management system

Please select the desired option

Room reservation

Cancellation

Maintenance

Report generation

GUI 2 : When the reservation button is clicked then the form will be displayed. This form will allow the user to enter the customer details such as last name, first name, phone number and so on.

The availability of desired room type should also be displayed by the system.

GUI 3 : If the user selects for the **cancellation** then the user has to fill up another form which for cancellation.

4.2 Hardware Interface

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

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 room availability 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 must provide password to log on the system. The password can be changed by the system administrator. The customer should not be allowed to the access to any of the databases.

7.2 Reliability

The system must be reliable to prevent any unauthorized access.

7.3 Availability

The system should be available during the hotel operating hours.

7.4 Maintainability

There should be the facility to add/modify information about the rooms numbers, types of rooms, charges.

8. Operational Scenarios

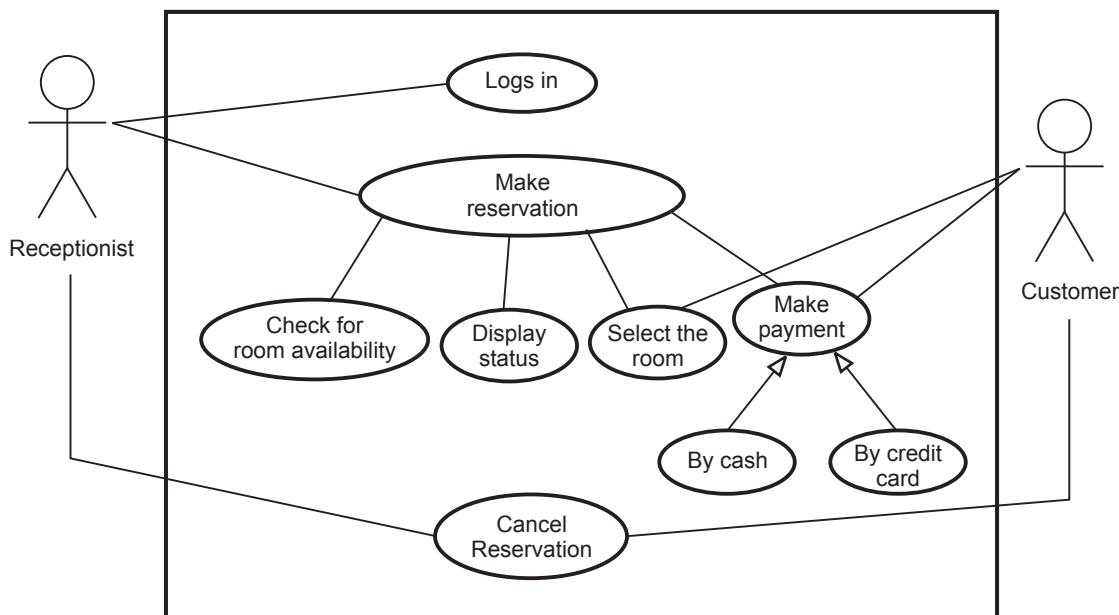
The customer makes the request for the type of room he/she wants to the receptionist. The system shows the availability of the room. If the desired type of room is available then this status is displayed. The customer gives the personal details and the room is reserved by the customer.

If the customer wants to cancel the reservation then he submits the reservation number. The reservation details are displayed on the screen. The reservation is cancelled by the manager and the corresponding room is marked as available. If any charges for the cancellation needs to be paid then the customer pays the charges.

9. Preliminary Schedule

The system must be implemented within 6 months.

Use case diagram for hotel management system



Example 4.16.3 Enlist characteristic of SRS. Write a SRS for Hospital management system.

GTU : Winter-2017, Marks 7

Solution : Characteristics of SRS : Refer section 4.16.1.

SRS for hospital management system :

1. Introduction

1.1 Purpose

This document gives detailed functional and non functional requirements for the hospital management 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 product will maintain the records of indoor and outdoor patients and provide separate billing method.

1.3 Overview

This system provides an easy solution to hospitals to manage billing of indoor, outdoor patients.

2. General description

The user will enter patients' details, billing information. The information includes name of the patient, address, phone number, his disease, his doctor and so on. If the patient is IPD then his total bill calculation is done. For IPD patient the OPD charges are applied to him. The user will also have to select the laboratory tests done by the patient, the date of discharge of the patient and the amount of discount if there is any. The mode of payment must also be mentioned. If any medi-claim policy is there then such information must be accommodated.



Fig. 4.16.4

3. Functional requirements

This section provides the requirement overview of the product. The project will require the PHP as a front end and at the back end the database MYSQL will be running. Various functional modules that can be implemented by the product will be -

1. Patient's registration
2. Appointment scheduling
3. OPD billing
4. IPD billing
5. Pharmacy
6. Medical tests
7. Staff payment

3.1 Description

This is the first activity of the product. The patient's information must be registered. This will then add the patient's details in the database. This information can be made available later on if the same patient comes for follow-ups.

The appointment schedule can be done based on availability of doctors, availability of laboratories and availability of operation theatres(if required).

For the OPD patients the billing will be in prepaid mode. The pharmacy department will require patient's personal information as well as the name of the doctor associated with that patient. For IPD patients the pharmacy bill will be directly transferred to its final billing.

The patient has to undergo through various tests. The data about this test i.e. test report must be maintained. The total bill for these conducted tests must be calculated and for the IPD patients it can be transferred to final billing.

For handling staff payments the small payroll system must be implemented. This system will keep track of the information such as name of the staff, number of days he/she worked, his pay-scale, overtime work(if any).

3.2 Technical issues

This product will work on client-server architecture. It will require an internet server and which will be able to run PHP applications. The product should support some commonly used browsers such as Internet Explorer, Mozilla FireFox. External interfaces include keyboard and mouse, enabling navigations across the screens.

4. Interface requirements

Various interfaces for the product could be

1. Login page
2. Registration form
3. There will be screens displaying information about doctors availability for the IPD/OPD patients.
4. There will be screens that will capture the information about the available Schedule for the Laboratories if the patient has to undergo some medical tests.
5. There will be screens that will capture information regarding the billing details of the patients.
6. The medical tests reports of the patients can be generated by the system.
7. Pharmacy billing report can be generated separately.
8. Discharge-summary report can be generated.

4.1 GUI

The receptionist will be the first entity which will start handling the product. Hence user can be administrator, staff nurse, or a cashier. For security purpose the user has to first login. The sample login page will be -

The user will enter the patient's information. The sample GUI for patient's registration will be like this

Patient Appointment and Registration Form			
Appointment ID	<input type="text"/>	Date of Appointment	<input type="text"/> ▼
Registration ID	<input type="text"/>	Time of Appointment	<input type="text"/> ▼
Patient's Information :			
Name	<input type="text"/>	Phone	<input type="text"/>
Address	<input type="text"/>	Email	<input type="text"/>
Doctor's Name	<input type="text"/>	Medical Test	<input type="text"/>
Type of Patient (OPD/IPD)	<input type="text"/>		
Medical History (If any)	<input type="text"/>		
SUBMIT		CLEAR	

For outdoor patients, to get an appointment, doctors' availability must be known. The screens displaying the dates and timings on which particular doctor is available can be as follows -

Availability of doctors			
Name of doctor	Specialist in	Date	Timings

The pharmacy billing report can be generated by the system. The bill format can be as follows -

4.2 Hardware interface

1. The application demands that all the PCs must be present in the intranet. So that proper sharing of information from various departments will be possible.
2. PC should be sufficiently fast with adequate memory space. At least 64 MB RAM and 2 GB hard-disk space is required to run this application.
3. Screen resolution of atleast 800×600 is required to properly view the screens.
4. It should support printers such as injet/laser/dot-matrix or any. The appropriate printer drivers must be installed on these PCs. The printers are required to print the patient's reports/Bills.

4.3 Software interface

1. Any windows operating system.
2. The PHP must be installed. For the database handling MYSQL must be installed. These products are open source3 products.
The above mentioned products must be for development of the application.
3. The final application must be packaged in a set up program, so that the product can be easily installed on the client's machine.

5. Performance requirements

None.

6. Design constraints

None.

7. Other non functional attributes

7.1 Security

This application will be password protected. The user has to enter correct user name and password.

7.2 Reliability

The application should be highly reliable and it should generate all the updated information in correct order.

Name of Medical Store																							
Registration ID : _____	Date : _____																						
Name of Patient : _____	Doctor's Name : _____																						
Address : _____																							
Phone : _____																							
<table border="1"><thead><tr><th>Sr. No.</th><th>Name of medicine</th><th>Qty.</th><th>Price</th></tr></thead><tbody><tr><td>1.</td><td></td><td></td><td></td></tr><tr><td>2.</td><td></td><td></td><td></td></tr><tr><td>3.</td><td></td><td></td><td></td></tr><tr><td>4.</td><td></td><td></td><td></td></tr></tbody></table>				Sr. No.	Name of medicine	Qty.	Price	1.				2.				3.				4.			
Sr. No.	Name of medicine	Qty.	Price																				
1.																							
2.																							
3.																							
4.																							
Tax deducted : _____	Total amount : _____																						
Signature																							

7.3 Availability

Any information about the patient/doctor should be quickly available from any computer to the authorised user. The previously visited patient's data must also be maintained and should be made available to the administrator by simply entering his Registration ID.

7.4 Maintainability

The application should be maintainable in such a manner that if any new requirement occurs then it should be easily incorporated in an individual module.

7.5 Portability

The application should be portable on any windows based system.

8. Preliminary schedule

This product must be completed within 7 months.



Notes

5

Software Design

Syllabus

Design concepts and design principal, Architectural design, Component level design (Function oriented design, Object oriented design), User interface design, Web application design.

Contents

5.1	<i>Definition of Software Design</i>	
5.2	<i>Design Concepts</i>	Summer-2016, 2019, Winter-2017, 2018, 2019, · · · Marks 7
5.3	<i>Design Principles</i>	
5.4	<i>Design Model</i>	
5.5	<i>Architectural Design</i>	Summer-2018, 2019, Winter-2017, 2018, · · · Marks 7
5.6	<i>Component Level Design</i>	Summer-2016, 2019, · · · Marks 7
5.7	<i>User Interface Design</i>	Summer-2012, 2014, 2016, 2018, 2019, Winter-2017, 2018, 2019, · · · Marks 7
5.8	<i>Web Application Design</i>	

5.1 Definition of Software Design

Software design is model of software which translates the requirements into finished software product in which the details about software data structures, architecture, interfaces and components that are necessary to implement the system are given.

5.1.1 Designing within the Context of Software Engineering

- Software design is at the core of software engineering and it is applied irrespective of any process model.
- After analysing and modelling the requirements, software design is done which serves as the basis for code generation and testing.
- Software designing is a process of translating analysis model into the design model.
- The analysis model is manifested by scenario based, class based, flow oriented and behavioural elements and feed the design task.
- The classes and relationships defined by CRC index cards and other useful class based elements provide the basis for the **data or class design**.
- The **architectural design** defines the relationship between major structural elements of the software. The architectural styles and design patterns can be used to achieve the requirements defined for the system.

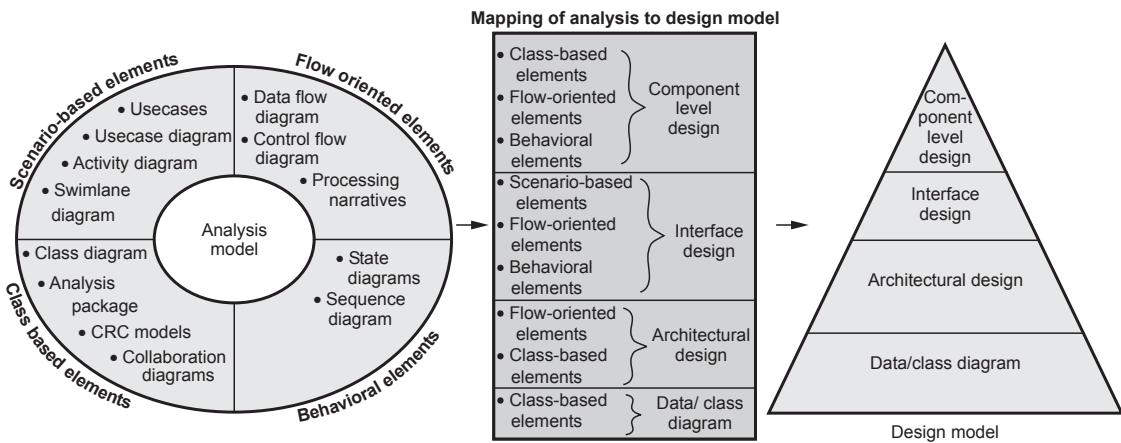


Fig. 5.1.1 Translating analysis model into the design model

- The **interface design** describes how software communicates with systems. These systems are interacting with each other as well as with the humans who operate them. Thus interface design represents the flow of information and specific type of behaviour. The usage scenarios and behavioural models of analysis modelling provide the information needed by the interface design.

- The component-level design transforms structural elements of software architecture into procedural description of software module. The information used by the component design is obtained from class based model, flow based model and behavioural model.
- Software design is **important** to assess the **quality** of software. Because design is the only way that we can accurately translate the user requirements into the finished software product.
- Without design unstable system may get developed. Even if some small changes are made then those changes will go fail .It will become difficult to test the product. The quality of the software product can not be assessed until late in the software process.

5.2 Design Concepts GTU : Summer-2016, 2019, Winter-2017, 2018, 2019, Marks 7

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 |
|--|---|

5.2.1 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.

5.2.2 Modularity

- The software is divided into **separately named** and **addressable components** that called as **modules**.
- Monolithic software is hard to grasp for the software engineer, hence it has now become a trend to divide the software into number of products. But there is a co-relation between the number of modules and overall cost of the software product. Following argument supports this idea -

"Suppose there are two problems A and B with varying complexity. If the complexity of problem A is greater than the complexity of the problem B then obviously the efforts required for solving the problem A is greater than that of problem B. That also means the time required by the problem A to get solved is more than that of problem B."

The overall complexity of two problems when they are combined is greater than the sum of complexity of the problems when considered individually. This leads to **divide and conquer strategy** (according to divide and conquer strategy the problem is divided into smaller subproblems and then the solution to these subproblems is obtained). Thus dividing the software problem into manageable number of pieces leads to the concept of modularity. It is possible to conclude that if we subdivide the software indefinitely then effort required to develop each software component will become very small. But this conclusion is invalid because the total number of modules get increased the efforts 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

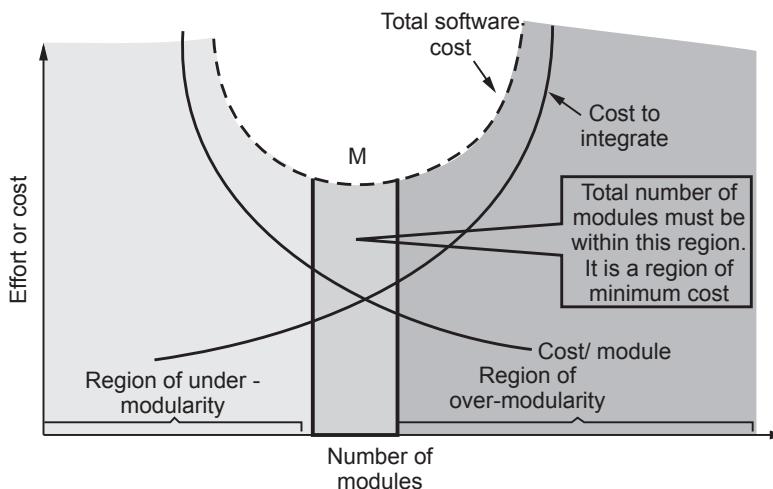


Fig. 5.2.1 Modularity and software cost

will also get increased. The total cost required to develop such software product is shown by following Fig. 5.2.1.

The above Fig. 5.2.1 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.

5.2.3 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.

5.2.4 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.

5.2.5 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.

5.2.6 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.

5.2.7 | 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.

5.2.7.1 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. **Communication 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.

5.2.7.2 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 :
 - Data coupling** - The data coupling is possible by parameter passing or data interaction.
 - Control coupling** - The modules share related control data in control coupling.
 - Common coupling** - In common coupling common data or a global data is shared among the modules.
 - 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.

5.2.8 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.

Review Questions

1. Explain the following design concepts.
i) Modularity ii) Architecture iii) Refinement
2. What do you understand by refactoring ? Give the importance of refactoring in improving the quality of software.
3. What are the characteristics of a well formed design class ?
4. Differentiate between abstraction and refinement.
5. Explain the different design concepts. GTU : Summer-2016, Marks 7
6. Compare coupling and cohesion. Explain different types of coupling and its effects on software modules. GTU : Winter-2017, Marks 4; Winter-2018, Marks 7
7. Define coupling and cohesion. What is the difference between cohesion and coupling ? GTU : Summer-2019, Marks 4
8. Compare coupling and cohesion. GTU : Winter-2019, Marks 3

5.3 Design Principles

Davis suggested a set of principles for software design as :

- The design process should not suffer from “tunnel vision”.
- The design should be traceable to the analysis model.
- The design should not reinvent the wheel.
- The design should “minimize the intellectual distance” between the software and the problem in the real world.
- The design should exhibit uniformity and integration.
- The design should be structured to accommodate change.
- The design should be structured to degrade gently.
- Design is not coding and coding is not design.
- The design should be assessed for quality.
- The design should be reviewed to minimize conceptual errors.

Review Question

1. State and explain any four design principles.

5.4 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.
- The **abstract dimension** represents level of details as each element of analysis model is transformed into design equivalent.
- In following Fig. 5.4.1 the **dashed line** shows the boundary between analysis and design model.

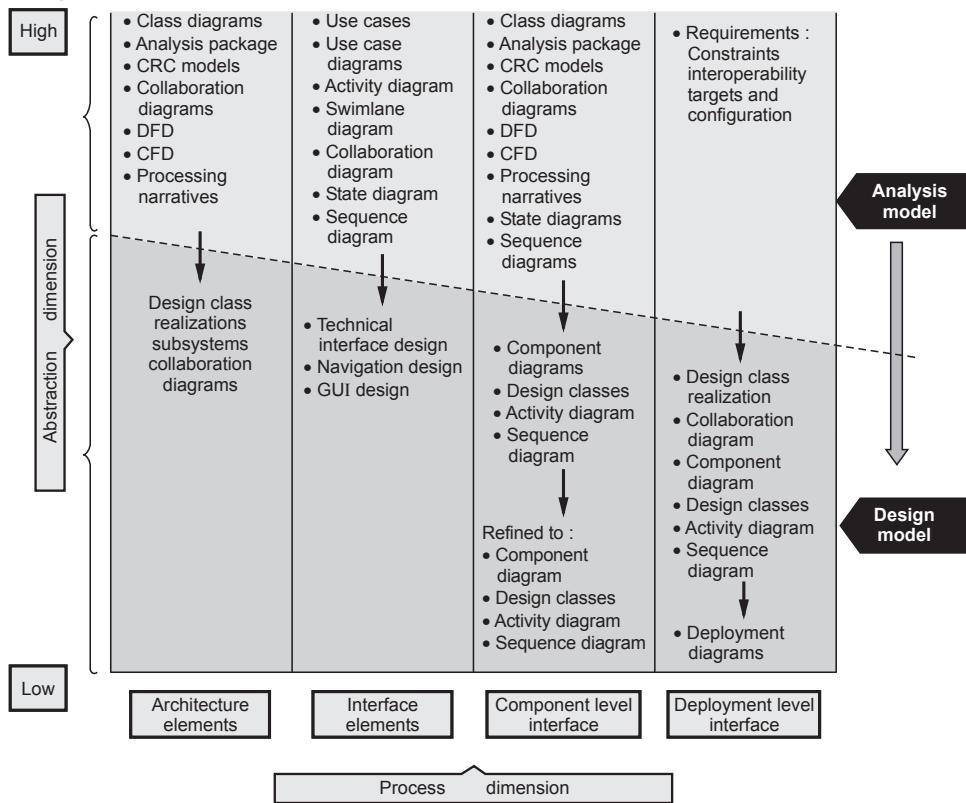


Fig. 5.4.1 Dimension of design model

- 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.

5.4.1 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.

5.4.2 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.

5.4.3 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 -

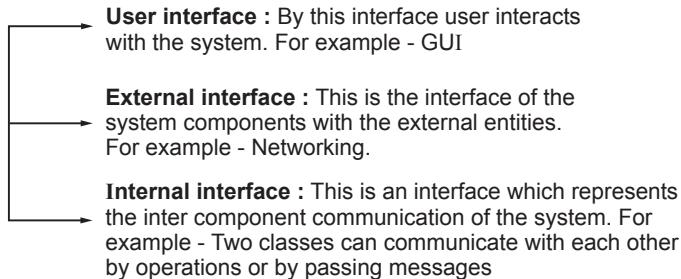


Fig. 5.4.2

5.4.4 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. 5.4.3 represents that component **order** makes use of another component **form**.

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

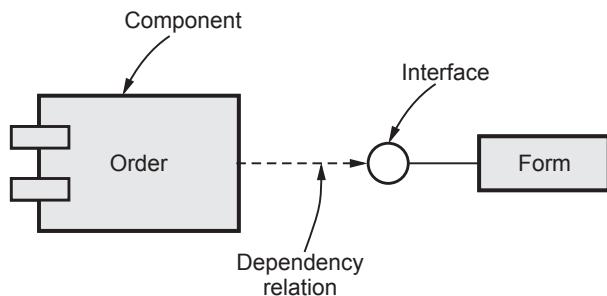


Fig. 5.4.3 Components

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

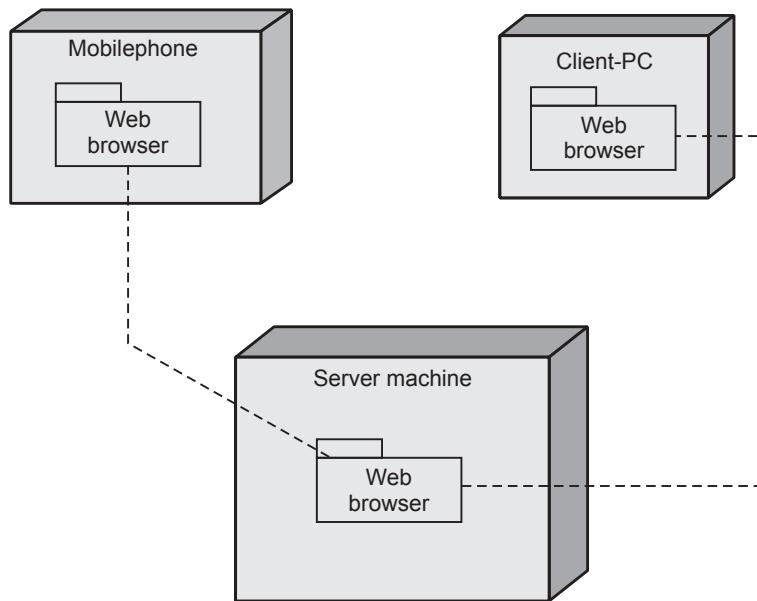


Fig. 5.4.4 Deployment diagram

5.5 Architectural Design GTU : Summer-2018, 2019, Winter-2017, 2018, Marks 7

5.5.1 Software Architecture

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.

5.5.1.1 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.

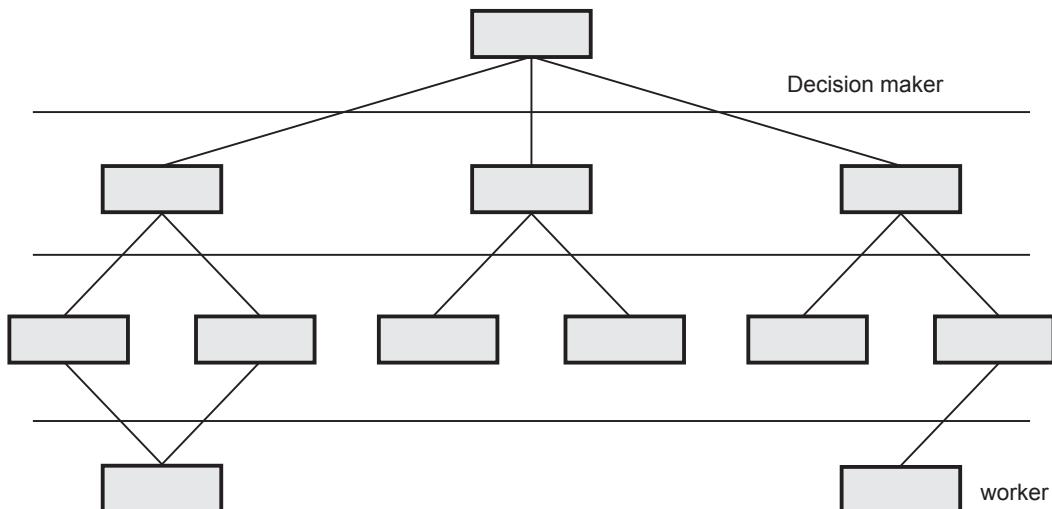


Fig. 5.5.1 Horizontal partitioning

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.

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.

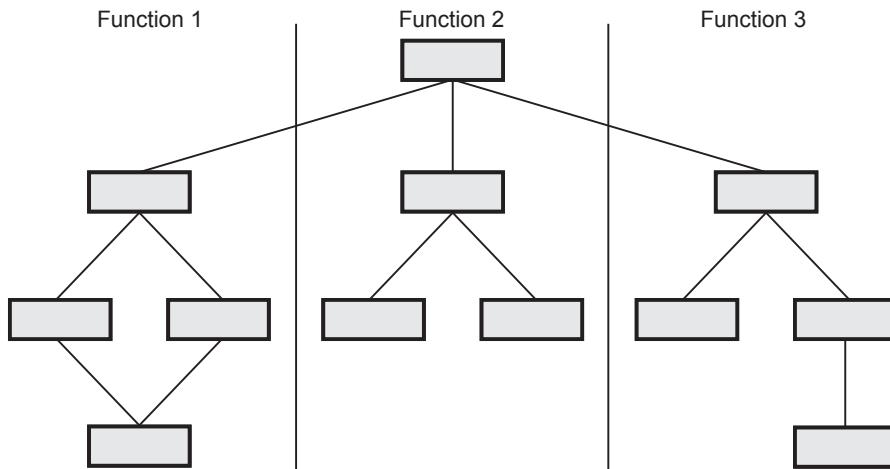


Fig. 5.5.2 Vertical partitioning

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.

5.5.1.2 Comparison between Horizontal and Vertical Partition

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.

5.5.2 Architectural Style

- 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
 - Components : They perform a function.
For example : Database, simple computational modules, clients, servers and filters.
 - Connectors : Enable communications. They define how the components communicate, co-ordinate and co-operate.
For example : Call, event broadcasting, pipes
 - Constraints : Define how the system can be integrated.
 - Semantic models : Specify how to determine a system's overall properties from the properties of its parts.
- The commonly used architectural styles are
 - Data centered architectures.
 - Data flow architectures.
 - Call and return architectures.
 - Object oriented architectures.
 - Layered architectures.

5.5.2.1 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.

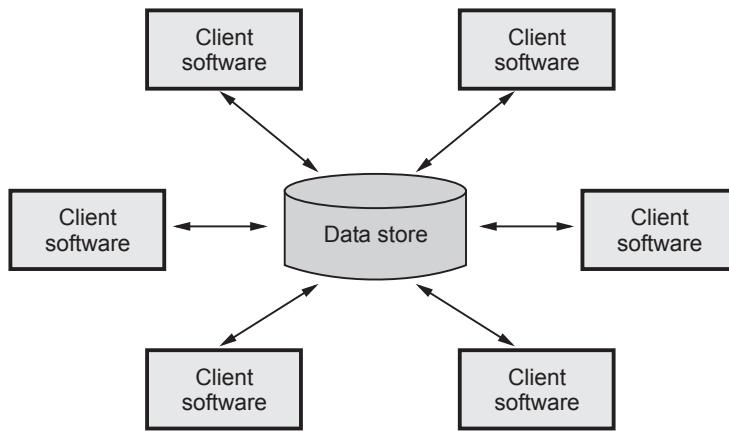


Fig. 5.5.3 Data centered architecture

Data centered architecture posses 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.

5.5.2.2 Data Flow Architectures

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

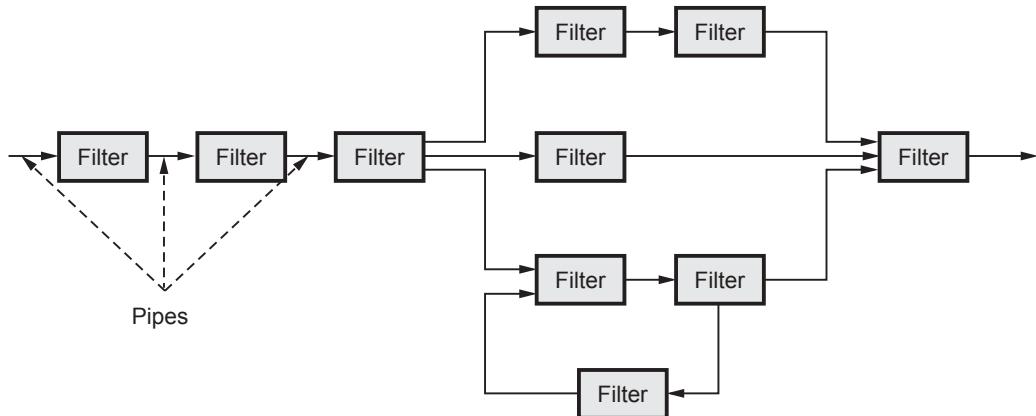


Fig. 5.5.4 Pipes and filters

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.

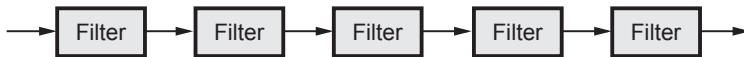


Fig. 5.5.5 Batch sequential

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

Comparison between Data Flow and Data Center Architecture

Data flow architecture	Data centered architecture
In this architecture series of transformations are applied to produce output data.	In this architecture the data store lies at the center of the architecture and other components frequently access it by performing add, delete, and modify operations.
In this architecture the set of components are called filters and are connected by pipes to transform data from one component to another.	In data centered architecture the components are the database elements such as tables and queries. And the communication among these components takes place with the help of relationship.
The filters normally work independently without bothering about working of neighboring filter.	The components of this architecture perform with the central repository without affecting the working of other components.
<pre> graph TD subgraph DF_Architecture [Data Flow Architecture] direction TB F1[Filter] --- P1[Pipe] F2[Filter] --- P1 F3[Filter] --- P1 F4[Filter] --- P2[Pipe] F5[Filter] --- P2 F6[Filter] --- P3[Pipe] F7[Filter] --- P3 F8[Filter] --- P4[Pipe] F9[Filter] --- P4 F10[Filter] --- P5[Pipe] F11[Filter] --- P5 F12[Filter] --- P6[Pipe] F13[Filter] --- P6 F14[Filter] --- P7[Pipe] F15[Filter] --- P7 F16[Filter] --- P8[Pipe] F17[Filter] --- P8 F18[Filter] --- P9[Pipe] F19[Filter] --- P9 F20[Filter] --- P10[Pipe] F21[Filter] --- P10 F22[Filter] --- P11[Pipe] F23[Filter] --- P11 F24[Filter] --- P12[Pipe] F25[Filter] --- P12 F26[Filter] --- P13[Pipe] F27[Filter] --- P13 F28[Filter] --- P14[Pipe] F29[Filter] --- P14 F30[Filter] --- P15[Pipe] F31[Filter] --- P15 F32[Filter] --- P16[Pipe] F33[Filter] --- P16 F34[Filter] --- P17[Pipe] F35[Filter] --- P17 F36[Filter] --- P18[Pipe] F37[Filter] --- P18 F38[Filter] --- P19[Pipe] F39[Filter] --- P19 F40[Filter] --- P20[Pipe] F41[Filter] --- P20 F42[Filter] --- P21[Pipe] F43[Filter] --- P21 F44[Filter] --- P22[Pipe] F45[Filter] --- P22 F46[Filter] --- P23[Pipe] F47[Filter] --- P23 F48[Filter] --- P24[Pipe] F49[Filter] --- P24 F50[Filter] --- P25[Pipe] F51[Filter] --- P25 F52[Filter] --- P26[Pipe] F53[Filter] --- P26 F54[Filter] --- P27[Pipe] F55[Filter] --- P27 F56[Filter] --- P28[Pipe] F57[Filter] --- P28 F58[Filter] --- P29[Pipe] F59[Filter] --- P29 F60[Filter] --- P30[Pipe] F61[Filter] --- P30 F62[Filter] --- P31[Pipe] F63[Filter] --- P31 F64[Filter] --- P32[Pipe] F65[Filter] --- P32 F66[Filter] --- P33[Pipe] F67[Filter] --- P33 F68[Filter] --- P34[Pipe] F69[Filter] --- P34 F70[Filter] --- P35[Pipe] F71[Filter] --- P35 F72[Filter] --- P36[Pipe] F73[Filter] --- P36 F74[Filter] --- P37[Pipe] F75[Filter] --- P37 F76[Filter] --- P38[Pipe] F77[Filter] --- P38 F78[Filter] --- P39[Pipe] F79[Filter] --- P39 F80[Filter] --- P40[Pipe] F81[Filter] --- P40 F82[Filter] --- P41[Pipe] F83[Filter] --- P41 F84[Filter] --- P42[Pipe] F85[Filter] --- P42 F86[Filter] --- P43[Pipe] F87[Filter] --- P43 F88[Filter] --- P44[Pipe] F89[Filter] --- P44 F90[Filter] --- P45[Pipe] F91[Filter] --- P45 F92[Filter] --- P46[Pipe] F93[Filter] --- P46 F94[Filter] --- P47[Pipe] F95[Filter] --- P47 F96[Filter] --- P48[Pipe] F97[Filter] --- P48 F98[Filter] --- P49[Pipe] F99[Filter] --- P49 F100[Filter] --- P50[Pipe] F101[Filter] --- P50 F102[Filter] --- P51[Pipe] F103[Filter] --- P51 F104[Filter] --- P52[Pipe] F105[Filter] --- P52 F106[Filter] --- P53[Pipe] F107[Filter] --- P53 F108[Filter] --- P54[Pipe] F109[Filter] --- P54 F110[Filter] --- P55[Pipe] F111[Filter] --- P55 F112[Filter] --- P56[Pipe] F113[Filter] --- P56 F114[Filter] --- P57[Pipe] F115[Filter] --- P57 F116[Filter] --- P58[Pipe] F117[Filter] --- P58 F118[Filter] --- P59[Pipe] F119[Filter] --- P59 F120[Filter] --- P60[Pipe] F121[Filter] --- P60 F122[Filter] --- P61[Pipe] F123[Filter] --- P61 F124[Filter] --- P62[Pipe] F125[Filter] --- P62 F126[Filter] --- P63[Pipe] F127[Filter] --- P63 F128[Filter] --- P64[Pipe] F129[Filter] --- P64 F130[Filter] --- P65[Pipe] F131[Filter] --- P65 F132[Filter] --- P66[Pipe] F133[Filter] --- P66 F134[Filter] --- P67[Pipe] F135[Filter] --- P67 F136[Filter] --- P68[Pipe] F137[Filter] --- P68 F138[Filter] --- P69[Pipe] F139[Filter] --- P69 F140[Filter] --- P70[Pipe] F141[Filter] --- P70 F142[Filter] --- P71[Pipe] F143[Filter] --- P71 F144[Filter] --- P72[Pipe] F145[Filter] --- P72 F146[Filter] --- P73[Pipe] F147[Filter] --- P73 F148[Filter] --- P74[Pipe] F149[Filter] --- P74 F150[Filter] --- P75[Pipe] F151[Filter] --- P75 F152[Filter] --- P76[Pipe] F153[Filter] --- P76 F154[Filter] --- P77[Pipe] F155[Filter] --- P77 F156[Filter] --- P78[Pipe] F157[Filter] --- P78 F158[Filter] --- P79[Pipe] F159[Filter] --- P79 F160[Filter] --- P80[Pipe] F161[Filter] --- P80 F162[Filter] --- P81[Pipe] F163[Filter] --- P81 F164[Filter] --- P82[Pipe] F165[Filter] --- P82 F166[Filter] --- P83[Pipe] F167[Filter] --- P83 F168[Filter] --- P84[Pipe] F169[Filter] --- P84 F170[Filter] --- P85[Pipe] F171[Filter] --- P85 F172[Filter] --- P86[Pipe] F173[Filter] --- P86 F174[Filter] --- P87[Pipe] F175[Filter] --- P87 F176[Filter] --- P88[Pipe] F177[Filter] --- P88 F178[Filter] --- P89[Pipe] F179[Filter] --- P89 F180[Filter] --- P90[Pipe] F181[Filter] --- P90 F182[Filter] --- P91[Pipe] F183[Filter] --- P91 F184[Filter] --- P92[Pipe] F185[Filter] --- P92 F186[Filter] --- P93[Pipe] F187[Filter] --- P93 F188[Filter] --- P94[Pipe] F189[Filter] --- P94 F190[Filter] --- P95[Pipe] F191[Filter] --- P95 F192[Filter] --- P96[Pipe] F193[Filter] --- P96 F194[Filter] --- P97[Pipe] F195[Filter] --- P97 F196[Filter] --- P98[Pipe] F197[Filter] --- P98 F198[Filter] --- P99[Pipe] F199[Filter] --- P99 F200[Filter] --- P100[Pipe] F201[Filter] --- P100 F202[Filter] --- P101[Pipe] F203[Filter] --- P101 F204[Filter] --- P102[Pipe] F205[Filter] --- P102 F206[Filter] --- P103[Pipe] F207[Filter] --- P103 F208[Filter] --- P104[Pipe] F209[Filter] --- P104 F210[Filter] --- P105[Pipe] F211[Filter] --- P105 F212[Filter] --- P106[Pipe] F213[Filter] --- P106 F214[Filter] --- P107[Pipe] F215[Filter] --- P107 F216[Filter] --- P108[Pipe] F217[Filter] --- P108 F218[Filter] --- P109[Pipe] F219[Filter] --- P109 F220[Filter] --- P110[Pipe] F221[Filter] --- P110 F222[Filter] --- P111[Pipe] F223[Filter] --- P111 F224[Filter] --- P112[Pipe] F225[Filter] --- P112 F226[Filter] --- P113[Pipe] F227[Filter] --- P113 F228[Filter] --- P114[Pipe] F229[Filter] --- P114 F230[Filter] --- P115[Pipe] F231[Filter] --- P115 F232[Filter] --- P116[Pipe] F233[Filter] --- P116 F234[Filter] --- P117[Pipe] F235[Filter] --- P117 F236[Filter] --- P118[Pipe] F237[Filter] --- P118 F238[Filter] --- P119[Pipe] F239[Filter] --- P119 F240[Filter] --- P120[Pipe] F241[Filter] --- P120 F242[Filter] --- P121[Pipe] F243[Filter] --- P121 F244[Filter] --- P122[Pipe] F245[Filter] --- P122 F246[Filter] --- P123[Pipe] F247[Filter] --- P123 F248[Filter] --- P124[Pipe] F249[Filter] --- P124 F250[Filter] --- P125[Pipe] F251[Filter] --- P125 F252[Filter] --- P126[Pipe] F253[Filter] --- P126 F254[Filter] --- P127[Pipe] F255[Filter] --- P127 F256[Filter] --- P128[Pipe] F257[Filter] --- P128 F258[Filter] --- P129[Pipe] F259[Filter] --- P129 F260[Filter] --- P130[Pipe] F261[Filter] --- P130 F262[Filter] --- P131[Pipe] F263[Filter] --- P131 F264[Filter] --- P132[Pipe] F265[Filter] --- P132 F266[Filter] --- P133[Pipe] F267[Filter] --- P133 F268[Filter] --- P134[Pipe] F269[Filter] --- P134 F270[Filter] --- P135[Pipe] F271[Filter] --- P135 F272[Filter] --- P136[Pipe] F273[Filter] --- P136 F274[Filter] --- P137[Pipe] F275[Filter] --- P137 F276[Filter] --- P138[Pipe] F277[Filter] --- P138 F278[Filter] --- P139[Pipe] F279[Filter] --- P139 F280[Filter] --- P140[Pipe] F281[Filter] --- P140 F282[Filter] --- P141[Pipe] F283[Filter] --- P141 F284[Filter] --- P142[Pipe] F285[Filter] --- P142 F286[Filter] --- P143[Pipe] F287[Filter] --- P143 F288[Filter] --- P144[Pipe] F289[Filter] --- P144 F290[Filter] --- P145[Pipe] F291[Filter] --- P145 F292[Filter] --- P146[Pipe] F293[Filter] --- P146 F294[Filter] --- P147[Pipe] F295[Filter] --- P147 F296[Filter] --- P148[Pipe] F297[Filter] --- P148 F298[Filter] --- P149[Pipe] F299[Filter] --- P149 F300[Filter] --- P150[Pipe] F301[Filter] --- P150 F302[Filter] --- P151[Pipe] F303[Filter] --- P151 F304[Filter] --- P152[Pipe] F305[Filter] --- P152 F306[Filter] --- P153[Pipe] F307[Filter] --- P153 F308[Filter] --- P154[Pipe] F309[Filter] --- P154 F310[Filter] --- P155[Pipe] F311[Filter] --- P155 F312[Filter] --- P156[Pipe] F313[Filter] --- P156 F314[Filter] --- P157[Pipe] F315[Filter] --- P157 F316[Filter] --- P158[Pipe] F317[Filter] --- P158 F318[Filter] --- P159[Pipe] F319[Filter] --- P159 F320[Filter] --- P160[Pipe] F321[Filter] --- P160 F322[Filter] --- P161[Pipe] F323[Filter] --- P161 F324[Filter] --- P162[Pipe] F325[Filter] --- P162 F326[Filter] --- P163[Pipe] F327[Filter] --- P163 F328[Filter] --- P164[Pipe] F329[Filter] --- P164 F330[Filter] --- P165[Pipe] F331[Filter] --- P165 F332[Filter] --- P166[Pipe] F333[Filter] --- P166 F334[Filter] --- P167[Pipe] F335[Filter] --- P167 F336[Filter] --- P168[Pipe] F337[Filter] --- P168 F338[Filter] --- P169[Pipe] F339[Filter] --- P169 F340[Filter] --- P170[Pipe] F341[Filter] --- P170 F342[Filter] --- P171[Pipe] F343[Filter] --- P171 F344[Filter] --- P172[Pipe] F345[Filter] --- P172 F346[Filter] --- P173[Pipe] F347[Filter] --- P173 F348[Filter] --- P174[Pipe] F349[Filter] --- P174 F350[Filter] --- P175[Pipe] F351[Filter] --- P175 F352[Filter] --- P176[Pipe] F353[Filter] --- P176 F354[Filter] --- P177[Pipe] F355[Filter] --- P177 F356[Filter] --- P178[Pipe] F357[Filter] --- P178 F358[Filter] --- P179[Pipe] F359[Filter] --- P179 F360[Filter] --- P180[Pipe] F361[Filter] --- P180 F362[Filter] --- P181[Pipe] F363[Filter] --- P181 F364[Filter] --- P182[Pipe] F365[Filter] --- P182 F366[Filter] --- P183[Pipe] F367[Filter] --- P183 F368[Filter] --- P184[Pipe] F369[Filter] --- P184 F370[Filter] --- P185[Pipe] F371[Filter] --- P185 F372[Filter] --- P186[Pipe] F373[Filter] --- P186 F374[Filter] --- P187[Pipe] F375[Filter] --- P187 F376[Filter] --- P188[Pipe] F377[Filter] --- P188 F378[Filter] --- P189[Pipe] F379[Filter] --- P189 F380[Filter] --- P190[Pipe] F381[Filter] --- P190 F382[Filter] --- P191[Pipe] F383[Filter] --- P191 F384[Filter] --- P192[Pipe] F385[Filter] --- P192 F386[Filter] --- P193[Pipe] F387[Filter] --- P193 F388[Filter] --- P194[Pipe] F389[Filter] --- P194 F390[Filter] --- P195[Pipe] F391[Filter] --- P195 F392[Filter] --- P196[Pipe] F393[Filter] --- P196 F394[Filter] --- P197[Pipe] F395[Filter] --- P197 F396[Filter] --- P198[Pipe] F397[Filter] --- P198 F398[Filter] --- P199[Pipe] F399[Filter] --- P199 F400[Filter] --- P200[Pipe] F401[Filter] --- P200 F402[Filter] --- P201[Pipe] F403[Filter] --- P201 F404[Filter] --- P202[Pipe] F405[Filter] --- P202 F406[Filter] --- P203[Pipe] F407[Filter] --- P203 F408[Filter] --- P204[Pipe] F409[Filter] --- P204 F410[Filter] --- P205[Pipe] F411[Filter] --- P205 F412[Filter] --- P206[Pipe] F413[Filter] --- P206 F414[Filter] --- P207[Pipe] F415[Filter] --- P207 F416[Filter] --- P208[Pipe] F417[Filter] --- P208 F418[Filter] --- P209[Pipe] F419[Filter] --- P209 F420[Filter] --- P210[Pipe] F421[Filter] --- P210 F422[Filter] --- P211[Pipe] F423[Filter] --- P211 F424[Filter] --- P212[Pipe] F425[Filter] --- P212 F426[Filter] --- P213[Pipe] F427[Filter] --- P213 F428[Filter] --- P214[Pipe] F429[Filter] --- P214 F430[Filter] --- P215[Pipe] F431[Filter] --- P215 F432[Filter] --- P216[Pipe] F433[Filter] --- P216 F434[Filter] --- P217[Pipe] F435[Filter] --- P217 F436[Filter] --- P218[Pipe] F437[Filter] --- P218 F438[Filter] --- P219[Pipe] F439[Filter] --- P219 F440[Filter] --- P220[Pipe] F441[Filter] --- P220 F442[Filter] --- P221[Pipe] F443[Filter] --- P221 F444[Filter] --- P222[Pipe] F445[Filter] --- P222 F446[Filter] --- P223[Pipe] F447[Filter] --- P223 F448[Filter] --- P224[Pipe] F449[Filter] --- P224 F450[Filter] --- P225[Pipe] F451[Filter] --- P225 F452[Filter] --- P226[Pipe] F453[Filter] --- P226 F454[Filter] --- P227[Pipe] F455[Filter] --- P227 F456[Filter] --- P228[Pipe] F457[Filter] --- P228 F458[Filter] --- P229[Pipe] F459[Filter] --- P229 F460[Filter] --- P230[Pipe] F461[Filter] --- P230 F462[Filter] --- P231[Pipe] F463[Filter] --- P231 F464[Filter] --- P232[Pipe] F465[Filter] --- P232 F466[Filter] --- P233[Pipe] F467[Filter] --- P233 F468[Filter] --- P234[Pipe] F469[Filter] --- P234 F470[Filter] --- P235[Pipe] F471[Filter] --- P235 F472[Filter] --- P236[Pipe] F473[Filter] --- P236 F474[Filter] --- P237[Pipe] F475[Filter] --- P237 F476[Filter] --- P238[Pipe] F477[Filter] --- P238 F478[Filter] --- P239[Pipe] F479[Filter] --- P239 F480[Filter] --- P240[Pipe] F481[Filter] --- P240 F482[Filter] --- P241[Pipe] F483[Filter] --- P241 F484[Filter] --- P242[Pipe] F485[Filter] --- P242 F486[Filter] --- P243[Pipe] F487[Filter] --- P243 F488[Filter] --- P244[Pipe] F489[Filter] --- P244 F490[Filter] --- P245[Pipe] F491[Filter] --- P245 F492[Filter] --- P246[Pipe] F493[Filter] --- P246 F494[Filter] --- P247[Pipe] F495[Filter] --- P247 F496[Filter] --- P248[Pipe] F497[Filter] --- P248 F498[Filter] --- P249[Pipe] F499[Filter] --- P249 F500[Filter] --- P250[Pipe] F501[Filter] --- P250 F502[Filter] --- P251[Pipe] F503[Filter] --- P251 F504[Filter] --- P252[Pipe] F505[Filter] --- P252 F506[Filter] --- P253[Pipe] F507[Filter] --- P253 F508[Filter] --- P254[Pipe] F509[Filter] --- P254 F510[Filter] --- P255[Pipe] F511[Filter] --- P255 F512[Filter] --- P256[Pipe] F513[Filter] --- P256 F514[Filter] --- P257[Pipe] F515[Filter] --- P257 F516[Filter] --- P258[Pipe] F517[Filter] --- P258 F518[Filter] --- P259[Pipe] F519[Filter] --- P259 F520[Filter] --- P260[Pipe] F521[Filter] --- P260 F522[Filter] --- P261[Pipe] F523[Filter] --- P261 F524[Filter] --- P262[Pipe] F525[Filter] --- P262 F526[Filter] --- P263[Pipe] F527[Filter] --- P263 F528[Filter] --- P264[Pipe] F529[Filter] --- P264 F530[Filter] --- P265[Pipe] F531[Filter] --- P265 F532[Filter] --- P266[Pipe] F533[Filter] --- P266 F534[Filter] --- P267[Pipe] F535[Filter] --- P267 F536[Filter] --- P268[Pipe] F537[Filter] --- P268 F538[Filter] --- P269[Pipe] F539[Filter] --- P269 F540[Filter] --- P270[Pipe] F541[Filter] --- P270 F542[Filter] --- P271[Pipe] F543[Filter] --- P271 F544[Filter] --- P272[Pipe] F545[Filter] --- P272 F546[Filter] --- P273[Pipe] F547[Filter] --- P273 F548[Filter] --- P274[Pipe] F549[Filter] --- P274 F550[Filter] --- P275[Pipe] F551[Filter] --- P275 F552[Filter] --- P276[Pipe] F553[Filter] --- P276 F554[Filter] --- P277[Pipe] F555[Filter] --- P277 F556[Filter] --- P278[Pipe] F557[Filter] --- P278 F558[Filter] --- P279[Pipe] F559[Filter] --- P279 F560[Filter] --- P280[Pipe] F561[Filter] --- P280 F562[Filter] --- P281[Pipe] F563[Filter] --- P281 F564[Filter] --- P282[Pipe] F565[Filter] --- P282 F566[Filter] --- P283[Pipe] F567[Filter] --- P283 F568[Filter] --- P284[Pipe] F569[Filter] --- P284 F570[Filter] --- P285[Pipe] F571[Filter] --- P285 F572[Filter] --- P286[Pipe] F573[Filter] --- P286 F574[Filter] --- P287[Pipe] F575[Filter] --- P287 F576[Filter] --- P288[Pipe] F577[Filter] --- P288 F578[Filter] --- P289[Pipe] F579[Filter] --- P289 F580[Filter] --- P290[Pipe] F581[Filter] --- P290 F582[Filter] --- P291[Pipe] F583[Filter] --- P291 F584[Filter] --- P292[Pipe] F585[Filter] --- P292 F586[Filter] --- P293[Pipe] F587[Filter] --- P293 F588[Filter] --- P294[Pipe] F589[Filter] --- P294 F590[Filter] --- P295[Pipe] F591[Filter] --- P295 F592[Filter] --- P296[Pipe] F593[Filter] --- P296 F594[Filter] --- P297[Pipe] F595[Filter] --- P297 F596[Filter] --- P298[Pipe] F597[Filter] --- P298 F598[Filter] --- P299[Pipe] F599[Filter] --- P299 F600[Filter] --- P300[Pipe] F601[Filter] --- P300 F602[Filter] --- P301[Pipe] F603[Filter] --- P301 F604[Filter] --- P302[Pipe] F605[Filter] --- P302 F606[Filter] --- P303[Pipe] F607[Filter] --- P303 F608[Filter] --- P304[Pipe] F609[Filter] --- P304 F610[Filter] --- P305[Pipe] F611[Filter] --- P305 F612[Filter] --- P306[Pipe] F613[Filter] --- P306 F614[Filter] --- P307[Pipe] F615[Filter] --- P307 F616[Filter] --- P308[Pipe] F617[Filter] --- P308 F618[Filter] --- P309[Pipe] F619[Filter] --- P309 F620[Filter] --- P310[Pipe] F621[Filter] --- P310 F622[Filter] --- P311[Pipe] F623[Filter] --- P311 F624[Filter] --- P312[Pipe] F625[Filter] --- P312 F626[Filter] --- P313[Pipe] F627[Filter] --- P313 F628[Filter] --- P314[Pipe] F629[Filter] --- P314 F630[Filter] --- P315[Pipe] F631[Filter] --- P315 F632[Filter] --- P316[Pipe] F633[Filter] --- P316 F634[Filter] --- P317[Pipe] F635[Filter] --- P317 F636[Filter] --- P318[Pipe] F637[Filter] --- P318 F638[Filter] --- P319[Pipe] F639[Filter] --- P319 F640[Filter] --- P320[Pipe] F641[Filter] --- P320 F642[Filter] --- P321[Pipe] F643[Filter] --- P321 F644[Filter] --- P322[Pipe] F645[Filter] --- P322 F646[Filter] --- P323[Pipe] F647[Filter] --- P323 F648[Filter] --- P324[Pipe] F649[Filter] --- P324 F650[Filter] --- P325[Pipe] F651[Filter] --- P325 F652[Filter] --- P326[Pipe] F653[Filter] --- P326 F654[Filter] --- P327[Pipe] F655[Filter] --- P327 F656[Filter] --- P328[Pipe] F657[Filter] --- P328 F658[Filter] --- P329[Pipe] F659[Filter] --- P329 F660[Filter] --- P330[Pipe] F661[Filter] --- P330 F662[Filter] --- P331[Pipe] F663[Filter] --- P331 F664[Filter] --- P332[Pipe] F665[Filter] --- P332 F666[Filter] --- P333[Pipe] F667[Filter] --- P333 F668[Filter] --- P334[Pipe] F669[Filter] --- P334 F670[Filter] --- P335[Pipe] F671[Filter] --- P335 F672[Filter] --- P336[Pipe] F673[Filter] --- P336 F674[Filter] --- P337[Pipe] F675[Filter] --- P337 F676[Filter] --- P338[Pipe] F677[Filter] --- P338 F678[Filter] --- P339[Pipe] F679[Filter] --- P339 F680[Filter] --- P340[Pipe] F681[Filter] --- P340 F682[Filter] --- P341[Pipe] F683[Filter] --- P341 F684[Filter] --- P342[Pipe] F685[Filter] --- P342 F686[Filter] --- P343[Pipe] F687[Filter] --- P343 F688[Filter] --- P344[Pipe] F689[Filter] --- P344 F690[Filter] --- P345[Pipe] F691[Filter] --- P345 F692[Filter] --- P346[Pipe] F693[Filter] --- P346 F694[Filter] --- P347[Pipe] F695[Filter] --- P347 F696[Filter] --- P348[Pipe] F697[Filter] --- P348 F698[Filter] --- P349[Pipe] F699[Filter] --- P349 F700[Filter] --- P350[Pipe] F701[Filter] --- P350 F702[Filter] --- P351[Pipe] F703[Filter] --- P351 F704[Filter] --- P352[Pipe] F705[Filter] --- P352 F706[Filter] --- P353[Pipe] F707[Filter] --- P353 F708[Filter] --- P354[Pipe] F709[Filter] --- P354 F710[Filter] --- P355[Pipe] F711[Filter] --- P355 F712[Filter] --- P356[Pipe] F713[Filter] --- P356 F714[Filter] --- P357[Pipe] F715[Filter] --- P357 F716[Filter] --- P358[Pipe] F717[Filter] --- P358 F718[Filter] --- P359[Pipe] F719[Filter] --- P359 F720[Filter] --- P360[Pipe] F721[Filter] --- P360 F722[Filter] --- P361[Pipe] F723[Filter] --- P361 F724[Filter] --- P362[Pipe] F725[Filter] --- P362 F726[Filter] --- P363[Pipe] F727[Filter] --- P363 F728[Filter] --- P364[Pipe] F729[Filter] --- P364 F730[Filter] --- P365[Pipe] F731[Filter] --- P365 F732[Filter] --- P366[Pipe] F733[Filter] --- P366 F734[Filter] --- P367[Pipe] F735[Filter] --- P367 F736[Filter] --- P368[Pipe] F737[Filter] --- P368 F738[Filter] --- P369[Pipe] F739[Filter] --- P369 F740[Filter] --- P370[Pipe] F741[Filter] --- P370 F742[Filter] --- P371[Pipe] F743[Filter] --- P371 F744[Filter] --- P372[Pipe] F745[Filter] --- P372 F746[Filter] --- P373[Pipe] F747[Filter] --- P373 F748[Filter] --- P374[Pipe] F749[Filter] --- P374 F750[Filter] --- P375[Pipe] F751[Filter] --- P375 F752[Filter] --- P376[Pipe] F753[Filter] --- P376 F754[Filter] --- P377[Pipe] F755[Filter] --- P377 F756[Filter] --- P378[Pipe] F757[Filter] --- P378 F758[Filter] --- P379[Pipe] F759[Filter] --- P379 F760[Filter] --- P380[Pipe] F761[Filter] --- P380 F762[Filter] --- P381[Pipe] F763[Filter] --- P381 F764[Filter] --- P382[Pipe] F765[Filter] --- P382 F766[Filter] --- P383[Pipe] F767[Filter] --- P383 F768[Filter] --- P384[Pipe] F769[Filter] --- P384 F770[Filter] --- P385[Pipe] F771[Filter] --- P385 F772[Filter] --- P386[Pipe] F773[Filter] --- P386 F774[Filter] --- P387[Pipe] F775[Filter] --- P387 F776[Filter] --- P388[Pipe] F777[Filter] --- P388 F778[Filter] --- P389[Pipe] F779[Filter] --- P389 F780[Filter] --- P390[Pipe] F781[Filter] --- P390 F782[Filter] --- P391[Pipe] F783[Filter] --- P391 F784[Filter] --- P392[Pipe] F785[Filter] --- P392 F786[Filter]</pre>	

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

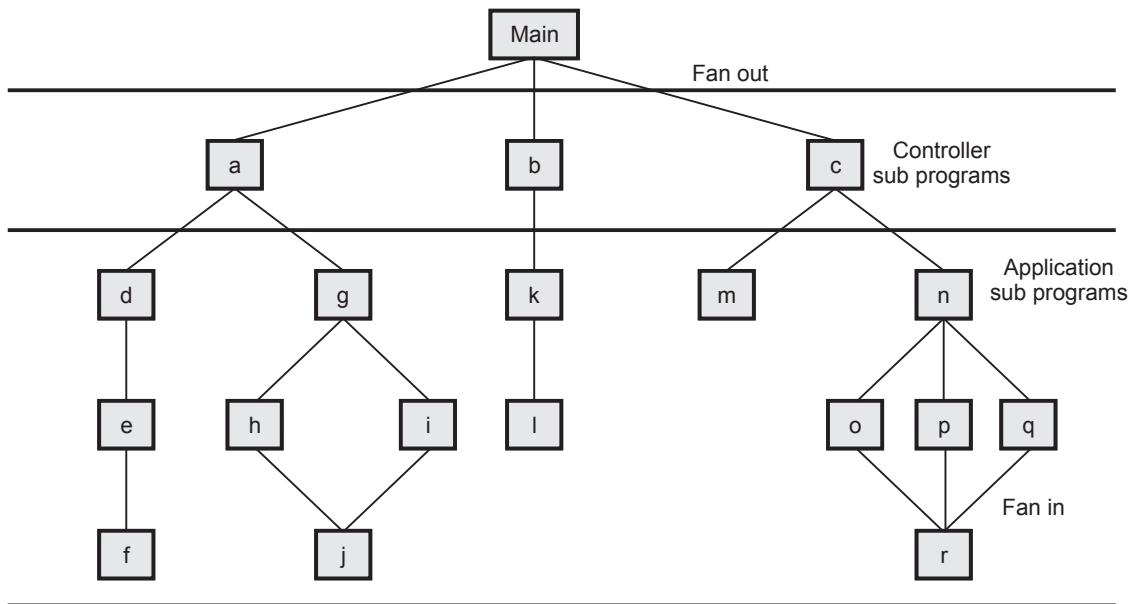


Fig. 5.5.6 Call and return Architecture

5.5.2.4 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.

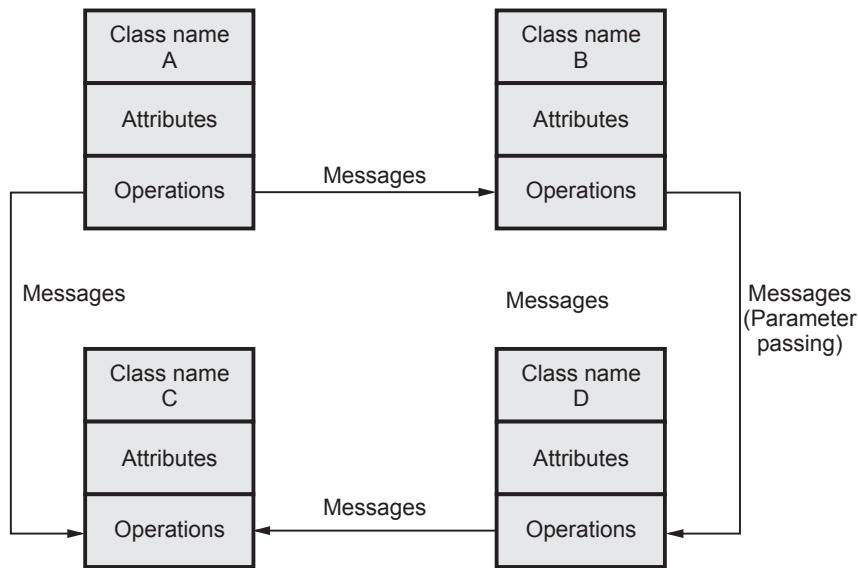


Fig. 5.5.7 Object oriented architecture

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.

5.5.2.5 | 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.

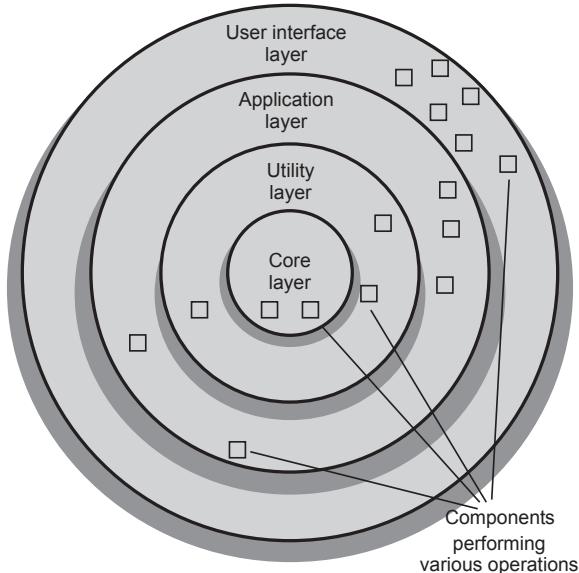


Fig. 5.5.8 Layer architecture of components

- 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.

Review Questions

- What is architectural design ? Enlist different style and patterns of architecture.

GTU : Winter-2017, 2018, Marks 7, Summer-2019, Marks 4

- What is software architecture ? Explain any two architectural styles of software.

GTU : Summer-2018, Marks 4

5.6 | Component Level Design

GTU : Summer-2016, 2019, Marks 7

- 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.
- The goal of software design strategy is to organize the program modules in such a way that they can be easily developed and can be changed easily if needed. The most commonly used strategies of software design are
 - 1. Function oriented design
 - 2. Object oriented design

5.6.1 Function Oriented Design

In function oriented design, the design of the software system is divided into various **interacting units**. Each unit then becomes a function. Thus system is designed from **functional point of view**. This approach is more used for developing smaller systems. But not preferred for larger systems because it is difficult to know what large systems do.

Various models that can be created in function oriented design are

- i) Data flow diagrams
- ii) Data dictionaries
- iii) Structure charts
- iv) Pseudo codes.

i) Data flow diagram

We have discussed Data flow diagrams in section 4.14.1.

ii) Data Dictionaries

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.

iii) Structure Charts

- The structure chart is a principle tool of structured design.
- The basic element in the structured chart is module. Module is defined as a collection of program statement with four attributes.

- **Input and output :** What the module gets from the invoker is called input and what the receiver gets from the module is called output.
- **Function :** The function processes the input and produces the output.
- **Mechanics :** The code or the logic by which the function is carried out.
- **Internal data :** It is the own workspace.

- The two modules can be connected to each other by a connector as shown below-
- The module uses data and flags. The data is processed by different modules. The flag is used as a control signal. It can be set or reset.

For example if we have two modules one for getting the employee detail(caller) and another module is for finding the employee name(called). Then the caller module will send the data as employee's ID and using that ID the called module will find the name of employee. If the employee ID is valid then that message will be given by the called module to a caller module. The use of data and flag can be as shown in Fig. 5.6.2.

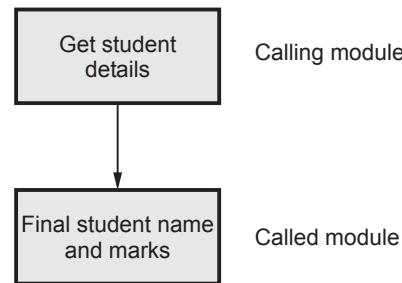
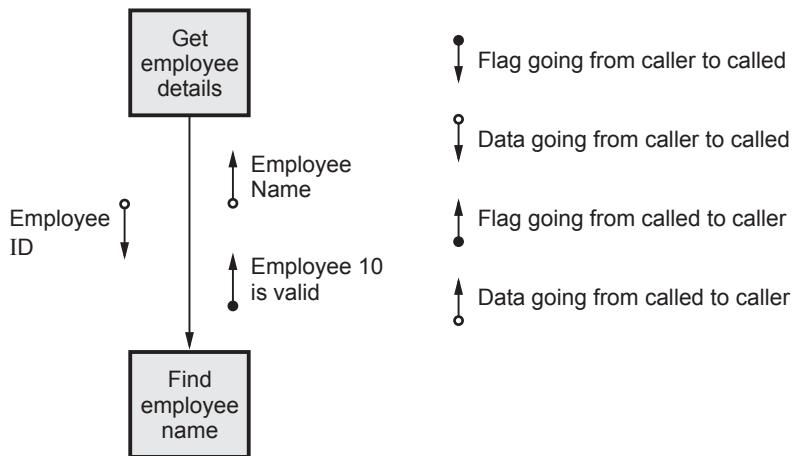
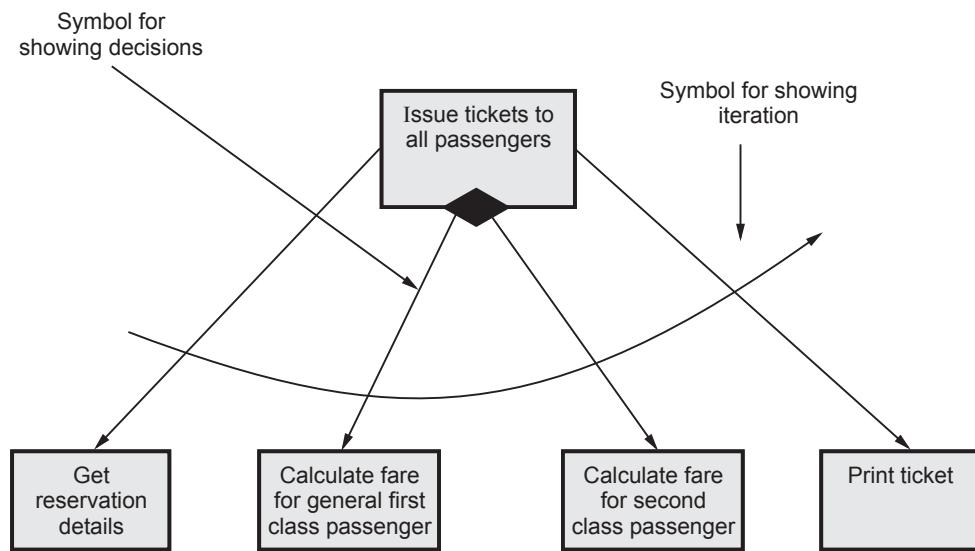


Fig. 5.6.1 Connector

**Fig. 5.6.2 Use of flag**

The iterations and decisions on a structured chart is as shown in Fig. 5.6.3.

**Fig. 5.6.3 Interaction**

The example of a structure chart is shown in Fig. 5.6.4.

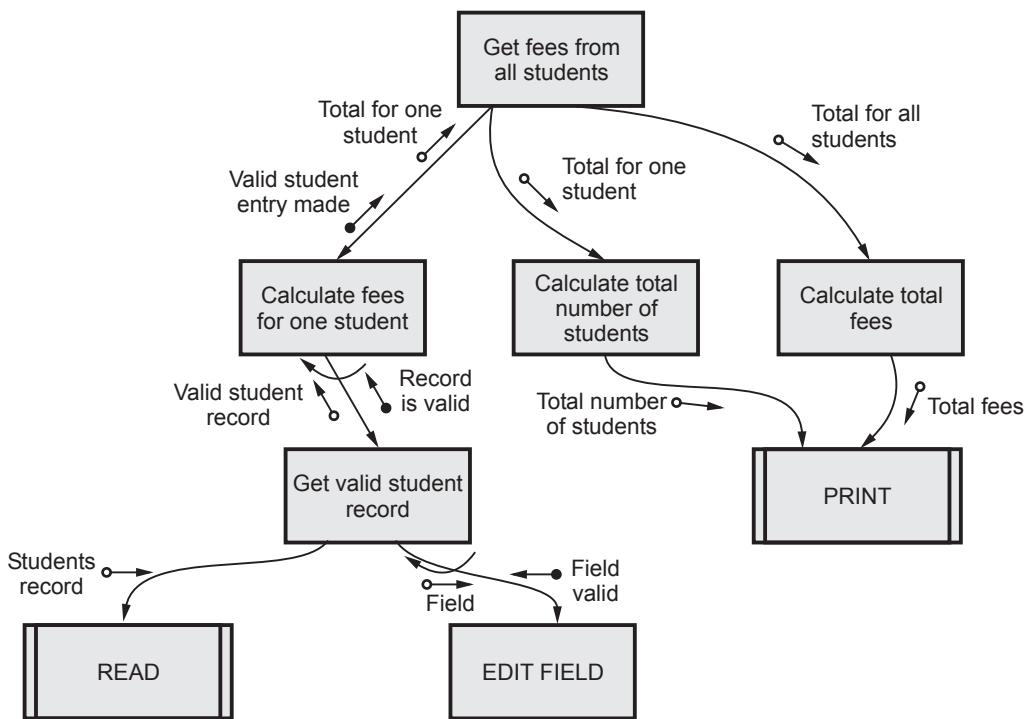


Fig. 5.6.4 Structure chart

iv) Pseudocode

Pseudo code is a combination of algorithm written in simple language and programming language statements. This code is effective building block for actual program.

For example - If we want to find out smallest number among three variables then the pseudo code can be written as follows.

1. Read values of a, b, and c variables.

2.

```

If (a < b)
{
    IF (a < c)
        print "a is small"
    Else
        print "c is small"
}
Else if (b < c)
    print "b is small"
Else
    print "c is small"
    
```

Advantages

1. This code helps in quick building of programming code.
2. It is simple to use.

5.6.2 Object Oriented Design

The software product can be designed using structured oriented method or using object oriented techniques. There are some merits and demerits associated with these systems which directly affect the software development process.

Structured analysis	Object oriented analysis
The structured analysis is a method in which focus is on functionality of the system.	The object oriented analysis is a technique in which main emphasis is on objects. Each object is a combination of data members and member functions.
The Data Flow Diagrams(DFD),Control Flow Diagrams(CFD) and Entity Relationship Diagrams(ERD) can be drawn in structured analysis.	The use cases, class diagram, sequence diagram, state chart diagram, activity diagram, component diagrams can be drawn in this method.
This is a simple technique of system analysis.	In this technique, the overhead of partitioning the structure into modules is involved.
Finding bugs is difficult because focus of this approach is merely on functionalities. That means the system structure is according to the functionalities.	Finding bugs is simple because system structure is modular.

Advantages of OOD

1. The object oriented techniques focus on real world concepts due to which **Communication** between system engineer and the customer becomes easy. This helps in **understanding** the system as a whole. And proper analysis of the system can be carried out.
2. The complex system can be partitioned into small **modules**. This helps in managing the large complex system. **Errors** can be identified efficiently. Even if one module is modified to some extent another module remains unaffected. This ultimately **reduces the maintenance cost** of the project.
3. One component of the system can be reused in the same system or in another system. This process is called **reusability**. Some times because of reusability the overall cost of system development gets reduced to a large extent.

Disadvantages of OOD

1. These systems are very **slow**.
2. These systems require **more memory**.

The object oriented design using UML

The object oriented design can be done using Unified Modeling Language (UML). Following are some diagrams that can be drawn to represent the object oriented design.

1. Class Diagram : Refer section 4.12
2. Use Case Diagram : Refer section 4.11
3. State Chart Diagram : Refer section 4.15.1
4. Activity Diagram : Refer section 4.11.2 and 4.11.3
5. Sequence Diagram : Refer section 4.15.2

Introduction to MS Visio Tool

The Microsoft Visio tool is commonly used for designing the UML diagrams. Using this tool you can draw all kinds of UML diagrams. You can drag and drop different components of object oriented design and can join them together using appropriate connector. Here are some steps used to create the class diagram in Visio

Step 1 : Start Visio 2003 or 2007. Then choose the drawing type. Select *Software* folder, choose **UML-model diagram** icon and **UML Static Structure Diagram**.

- Place the mouse over the shapes in UML Static Structure Diagram to view more details.

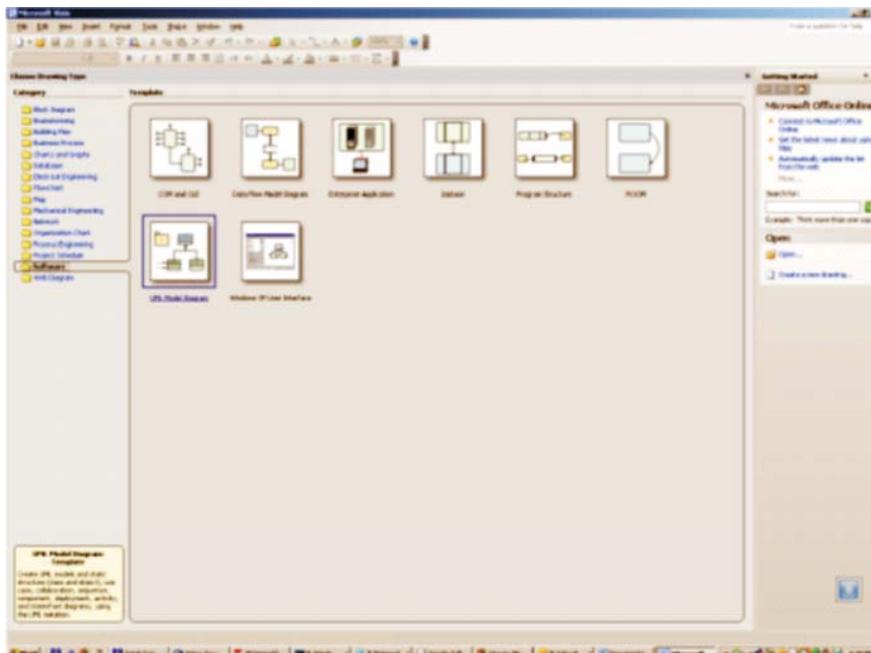


Fig. 5.6.5 Select UML Static Structure Diagram

Step 2 : Adding Class

- Click on the Class shape and drag it on the page.
- You can change the name of the class. Right click on the shape and a **Class Properties window** will appear. You can change its name from the name field.
- You can add attributes for the class by clicking **Attributes** in the right of Class Properties window. Refer following screen shots for creating class and setting its properties.

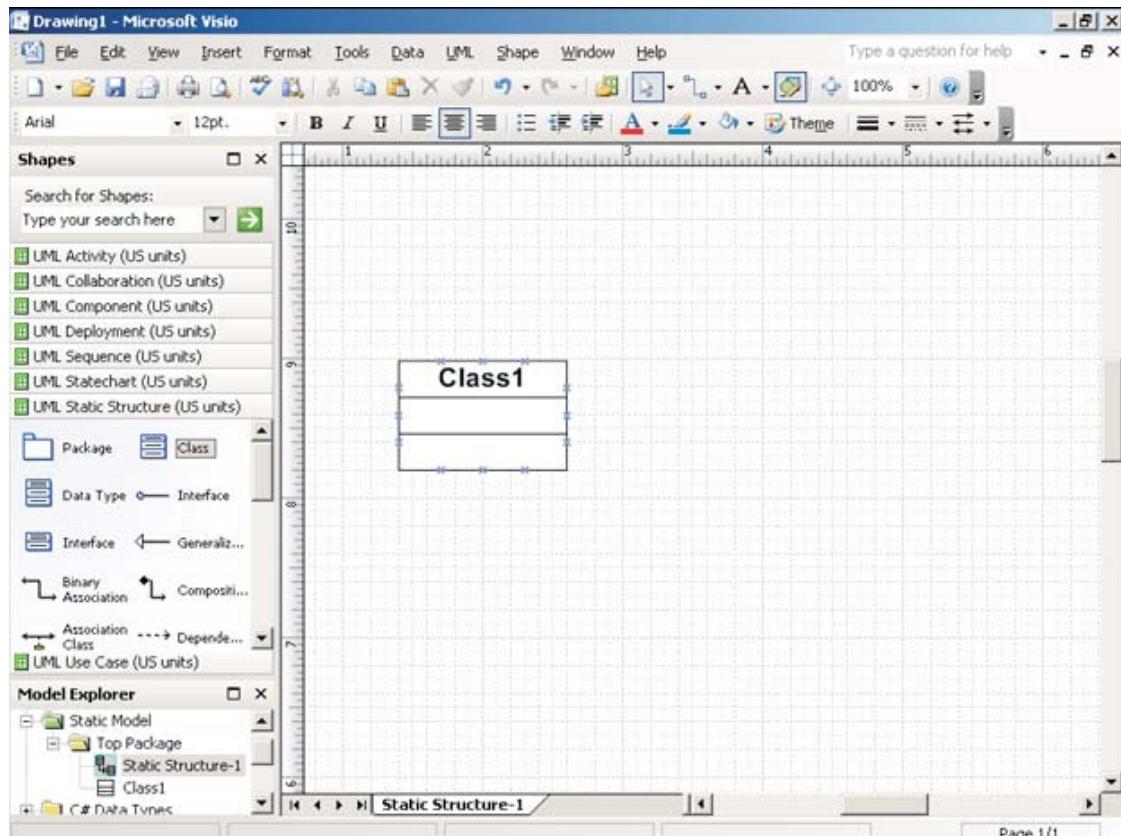


Fig. 5.6.6

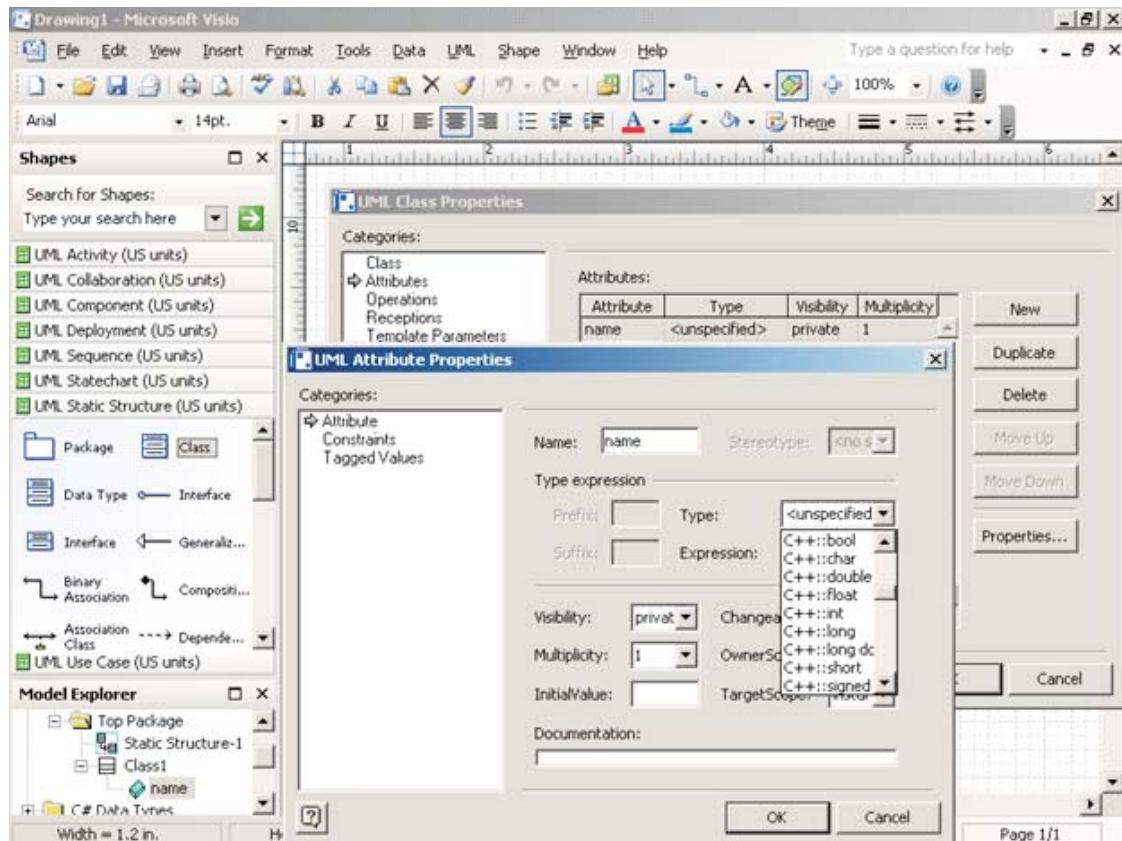


Fig. 5.6.7

Step 3 : Adding Relation

Just click and drag the type of the relation to connect two classes. You can add the relations such as association, aggregation, generalization relationship.

Double click on the association to specify it. You can give the association a name and/or enter role names in the column "End Name".

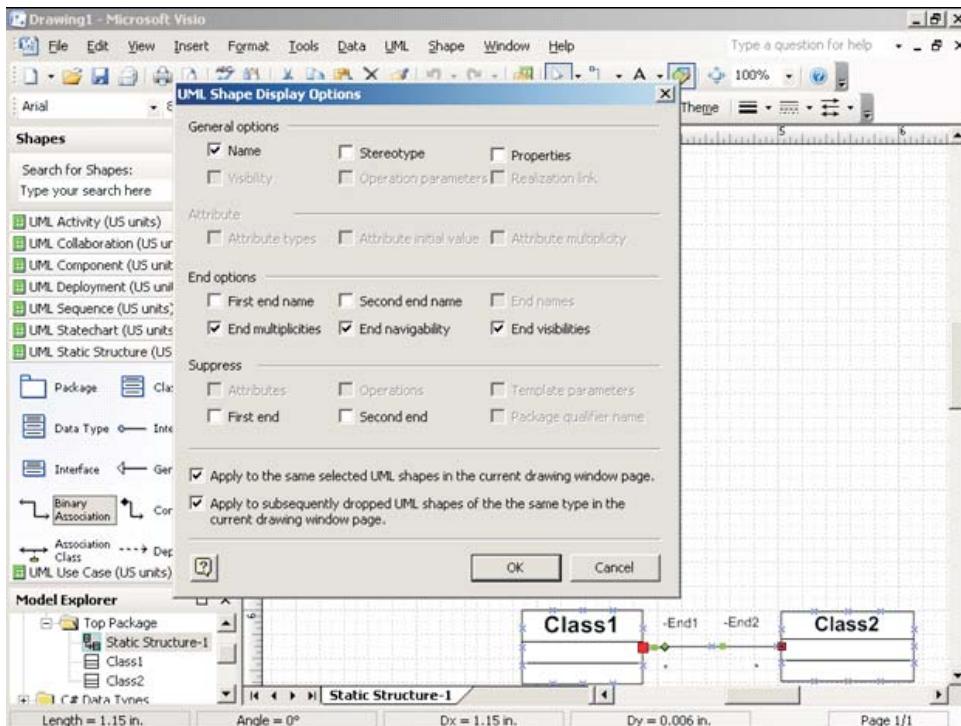


Fig. 5.6.8

Step 4 : Save the diagram in different formats. You can copy and paste the diagram in Word Document.

Review Questions

doubt

1. What is object design of a system ? Draw the use case diagram and class diagram for library management system. **GTU : Summer-2016, Marks 7**
2. State the differences between procedural design and object oriented design. **GTU : Summer-2019, Marks 3**

5.7 User Interface Design

GTU : May-2012, Summer-2012, 2014, 2016, 2018, Winter-2017, 2018, 2019, Marks 7

- In user interface design the effective interaction of system with user is provided.
- Typically system user judges a system by its interface rather than its functionality. A poorly designed interface leads improper use of the system. Many software systems can not be used because of poorly designed user interface.
- Most of the users in business systems interact with the systems using graphical user interfaces. However text based interfaces is also used.

Advantages of Graphical User Interfaces (GUI)

1. GUIs are easy to use. An inexperienced user can use the system easily with the graphical 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.

5.7.1 User Interface Design Principles

While designing the user interface the capability, need and experience of system user should be considered. The designer should take into account the people's physical and mental limitations. For instance a short memory or in game playing software a child can recognize pictures rather than text. He should also be aware of the fact that people make mistakes and the interface should tolerate these mistakes in a friendly manner. Following design principles are used.

1. **User familiarity** - Instead of using computer terminology make use of user oriented terminologies. For example in 'Microsoft Office' software, the terms such as document, spreadsheet, letter, folder are used and use of the terms directory, file and identities are avoided.
2. **Consistency** - The appropriate level of consistency should be maintained in the user interface. For example the commands or menus should be of same format.
3. **Minimal surprise** - The commands should operate in a known way. This makes user to easily predict the interface. For example the in word processing document under the tool menu there should be a facility of spelling and grammar checking.
4. **Recoverability** - The system should provide recovering facility to user from his errors so that user can correct those error. For example an undo command should be given so that user can correct his errors, or while deleting something the confirmation action must be provided, so that user can think again while deleting something.
5. **User guidance** - The user interface can be effectively used by a novice user if some user guidance such as help systems, online manuals etc. are supplied.

5.7.2 Golden Rules very very important

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 -

5.7.2.1 Place the User in Control

While analysing any requirement during requirement analysis the user often demands for the system which will satisfy user requirements and help him to get the things done. That means the user always wants to control the computerized system. Following are the design principles that allow user to control the system :

- Define the interaction modes in such a way that user will be restricted from doing the unnecessary actions

Interaction mode means the current state in which user is working and being in such a mode user is supposed to do the related tasks only if the user has to perform unnecessary actions at such time then the GUI becomes frustrating.

- The interaction should be flexible

The user interaction should be flexible. For example in the Microsoft power point slide show the slide transition is possible using mouse clicks as well as using keyboard. Such flexibility allows any user to operate the system as per his comfort.

- Provide the facility of 'undo' or 'interruption' in user interaction

This is the feature which allows the user to correct himself whenever necessary without loosing his previous work. For example : In the software like 'Paint' one can draw some objects and perform 'redo' or 'undo' actions for performing his desired drawing.

- Allow user to customize the interaction

It is observed that in while handling user interface certain actions need to be done repeatedly. It saves the time if these actions are collected in a Macro.

- Hide technical details from the user

This feature is essential for a casual user. User should not be aware of system commands, operating system functions or file management functions. For example while printing some document instead of giving the command for printing if user clicks on the icon indicating print then it becomes convenient for a casual user to take the print out.

- The objects appearing on the screen should be interactive with the user

This also means that user should be in position to adjust the object appearing on the screen. For example in 'Paint' one should be able to stretch the oval or edit the text written in the object.

5.7.2.2 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 sahasik-intuitive

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.

5.7.2.3 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.

5.7.3 Interface Design Steps

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.

5.7.4 User Interface Design 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 above Fig. 5.7.1 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.

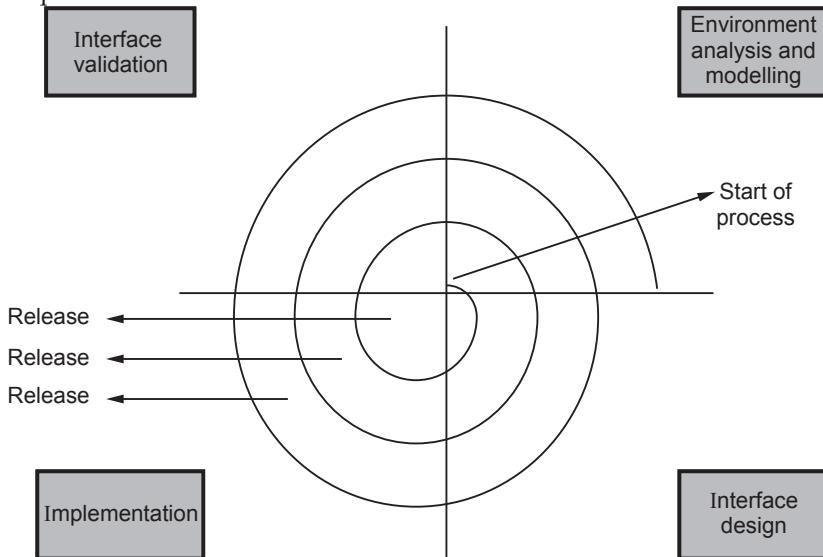


Fig. 5.7.1 Interface analysis and design process

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.

Review Questions

1. Explain the Golden rules of user interface design.
2. What are the interface design steps that can be applied while designing the interface ?
3. Explain the significance of User Interface (UI) in a system. Also explain the design Model for UI.
(Hint : For significance refer section 5.7 and for design model refer section 5.7.4)
4. Explain the user interface concept with suitable example.
5. What is the importance of user interface ? Explain user interface design rules.
6. Describe golden rules of User Interface Design.
7. Explain user interface design issues.
8. What is the importance of User Interface ? Explain User Interface design rules.

GTU : Summer-2012, Marks 7

GTU : Summer-2014, Marks 7

GTU : Summer-2016, Marks 7

GTU : Winter-2017, 2018, Marks 3

GTU : Summer-2018, Marks 3

GTU : Summer, Winter-2019, Marks 7

5.8 Web Application Design

In web engineering, the web design can be of two types - it can be **generic** and it can be **specific**.

From **generic viewpoint** various design models get developed as a outcome of design phase. Using these design models the user requirements can be translated into the executable code form. At the same time design must be **specific**. That means the web design must address key attributes of the web application in such a manner that the web engineer can build the complete web application and can test it.

5.8.1 Design Pyramid

The web design can be defined as the design that leads to a model that contains appropriate mix of aesthetic, content and technology. This mix may vary depending upon the nature and design activities of the web application. Following Fig. 5.8.1 (See Fig. 5.8.1 on next page) represents the design pyramid. Each level of this pyramid represents the design activities -

Interface design

This kind of defines the structure and organisation of user interface. It includes the representation of screen layout, various modes of interactions and navigation mechanism.

Aesthetic design

This kind of design defines the **look and feel** of the web application. It includes text style, graphic design, color scheme, text size and related aesthetic decisions.

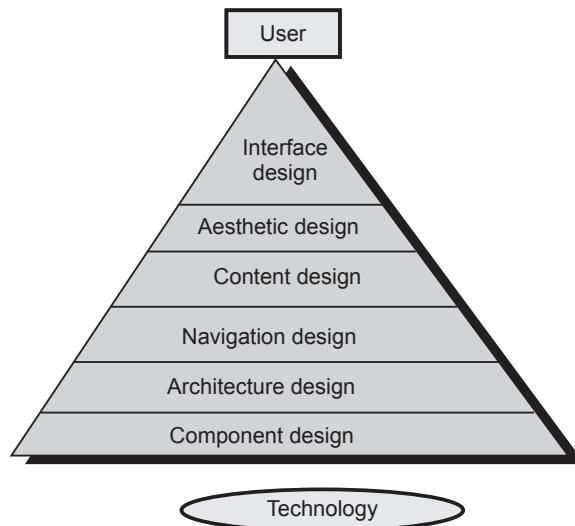


Fig. 5.8.1 The web design pyramid

Content design

This design defines the layout, structure and outline of the contents present in the web application.

Architecture design

This design defines the hypermedia structure of the web application.

Navigation design

This design defines the navigational flow between the content objects.

Component design

This design defines the processing logic required to implement functional components.

5.8.1.1 Interface Design

The interface design must be easy to use, easy to learn, easy to navigate, intuitive, consistent, efficient, error free and functional. It should provide a satisfying experience to the end user.

What are interface design principles and guidelines?

- Anticipation**

The interface design must anticipate user's next move.

- Consistency**

The GUI navigation controls must be consistent throughout the application.

- **Communication**

The status of the activity initiate by the user must be defined by the interface design. This will help in proper conversation between the end-user and web application.

- **Controlled autonomy**

The user movement must be controlled by the interface design. The conversation of user and web application must lie within the web application itself.

- **Efficiency**

The interface design should be such that the user's work efficiency must be optimized. The user need not have to wait for too long. Always look at the user's productivity and not at the computer.

- **Flexibility**

The interface should be flexible enough to enable some users to accomplish tasks directly or in somewhat random fashion.

- **Focus**

The web application interface should stay focused on the user task at hand.

- **Fitt's law**

The time to acquire target is a function of the distance to and the size of the target. According to Fitt's law for a online purchase system, if user clicks on **buy product** then immediately a **pull down menu** of product category (For example: Laptop, Pen Drive, IPod, Digital Camera) must appear. This product category should not be located somewhere in the corner of the screen. Because then the time for the user to acquire it would be too long.

- **Human interface objects**

Human interface objects can be seen, heard, touched or perceived. The human interface object must be seen on the GUI. These objects should have standard resulting behaviour. The human interface objects should be understandable, self consistent and stable.

- **Latency reduction**

Rather than making user to wait for some internal operation to complete, the web application should use multitasking. This will allow the user to proceed with the work as if the operation has been completed. The latency reduction also includes the some kind of acknowledgement to the user so that user may understand what is happening. For example : providing the clock or progress bar to indicate processing, some kind of audio feedback when immediate action does not occur.

- **Metaphor**

Metaphors assist the user to use the web application. For example user can fill up the form by selecting the items from the list. He need not have to write the complete name of item. Metaphors are simple to learn and use.

- **Protect user's work**

The work product must be automatically saved so that the contents will not get lost if some error occurs.

- **Readability**

All the information presented by the user interface must be readable by everybody.

- **Visible navigation**

Avoid invisible navigation. The navigation should be obvious even to the casual user. The user need not have to search the screen to determine how to link to other content or services.

What are interaction control mechanisms?

Various interaction control mechanisms are -

- **Graphic icons -**

Buttons, switches or graphical images assist user to select some item or to specify some decision.

- **Graphic images -**

Some graphical representation is selectable by the user and may exhibit some content object or web application functionality.

- **Navigation menus -**

Keyword menus can be presented vertically or horizontally to the user. Using this menu the user can choose from hierarchy of subtopics when primary menu option is selected.

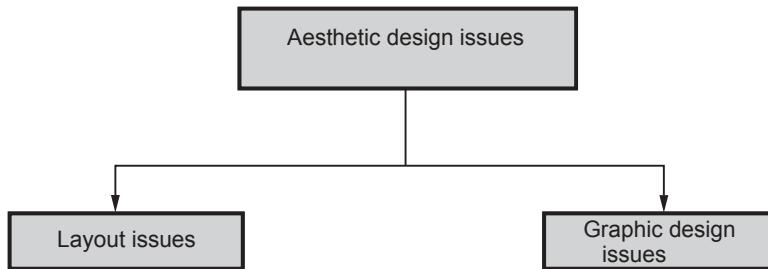
What are the tasks in the interface design?

Following are some tasks that can be carried out for web application interface design

- Review and refine the information present in the analysis model.
- Develop rough sketch of web application interface layout.
- Map user objectives into appropriate interface actions.
- Define the task set associated with each action.
- Storyboard screen images for each interface action.
- Refine interface layout using aesthetic design.
- Develop procedural representation for user's interaction.
- Develop behavioural representation for the interface.
- Interface layout of each state must be described in detail.
- Finally review and refine the interface design model.

5.8.1.2 Aesthetic Design

Aesthetic design means the graphical design. In aesthetic design there are two important issues that must be focussed.



- Layout issues
- Graphics design issues

Let us discuss these issues one by one -

Layout issues

There is no absolute rule for defining the screen layout. However following guideline is used while designing the layout.

1. Don't afraid of white space. In other words the web page should not be overloaded with lot of information.
2. The emphasis should be on content.
3. The layout of the screen must be arranged from top to bottom and from left to write. The high priority elements must be placed on the upper left corner of the web page.
4. As far as possible avoid usage of scroll bar. Although use of scroll bar is an important element of graphics, users never like to use scroll bars. Instead, one can divide the information over the multiple pages.
5. Group navigation, content and function geographically within the page.
6. The layout must be designed by considering resolution and browser window size.

Graphics design issues

The graphics design is concerned with the look and feel of the web application. In the aesthetic design first of all the **layout design** is done and then the global color schemes, typefaces, sizes, styles and other required aesthetic elements of an application are decided.

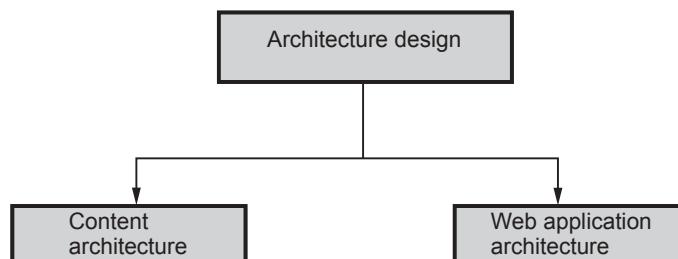
5.8.1.3 Content Design

- The content design develops the representation for the **content objects**. It also defines the mechanism required to instantiate the relationship of one content object with other.
- During analysis model, the content objects are identified and the relationships among these objects defined.
- During the content design the content object is more closely aligned with data object. The content object has attributes that specifies the **content specific information** during **analysis** and **implementation specific information** during **design**.
- Once all the content objects are modelled the information within each object must be authored and then formatted in such a manner that the customer's need get satisfied.

5.8.1.4 Architecture Design

The architecture design focuses on two types of design architectures -

- Content architecture
- Web application architecture



Content architecture

The content architecture focuses on the definition of overall hypermedia of structure of web application. Various content architectures are -

1. Linear structures

When there is a predictable sequence of interactions then linear structure is used. For example a Web Tutorial in which the web pages appear in a particular sequence. The structure of linear structure architecture is represented by following Fig. 5.8.2. (See Fig. 5.8.2 on next page.)

2. Grid structures

In this structure the web application contents can be organised categorically in two or more dimensions. It is as shown by Fig. 5.8.3. (Fig. 5.8.3 on next page.)

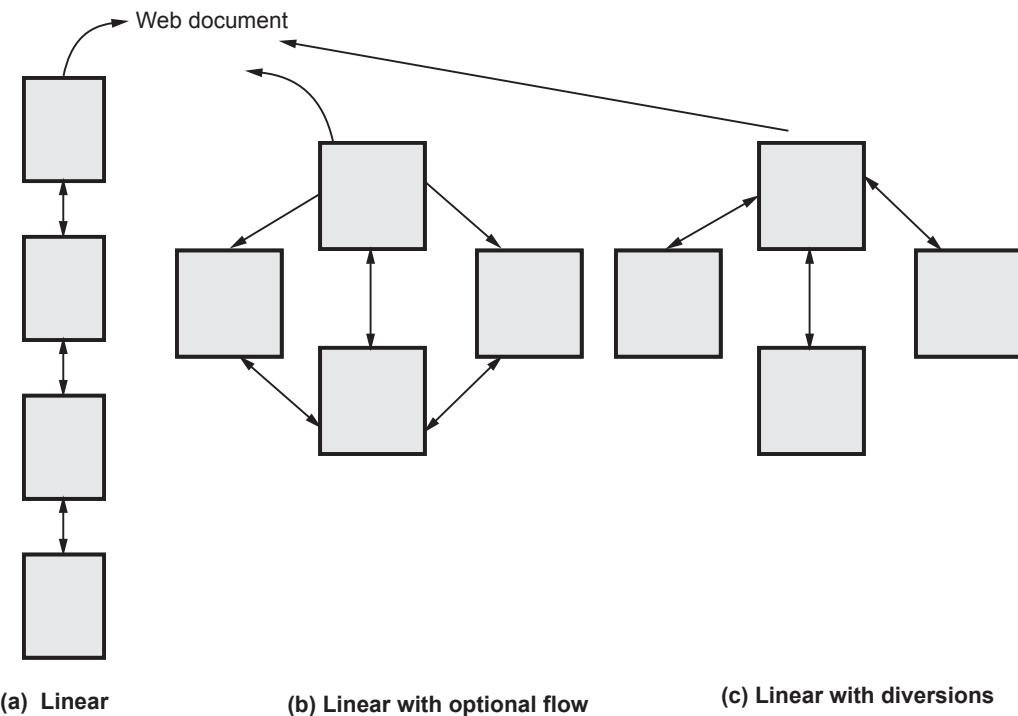


Fig. 5.8.2 Linear structure

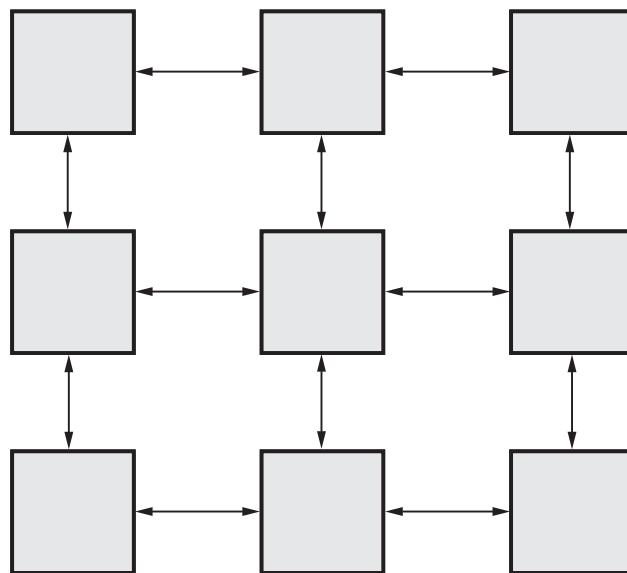


Fig. 5.8.3 Grid structure

3. Hierarchical structure

This is the most commonly used web architecture structure. The contents can be arranged in hierarchical manner and the flow of control is only along the vertical branches.

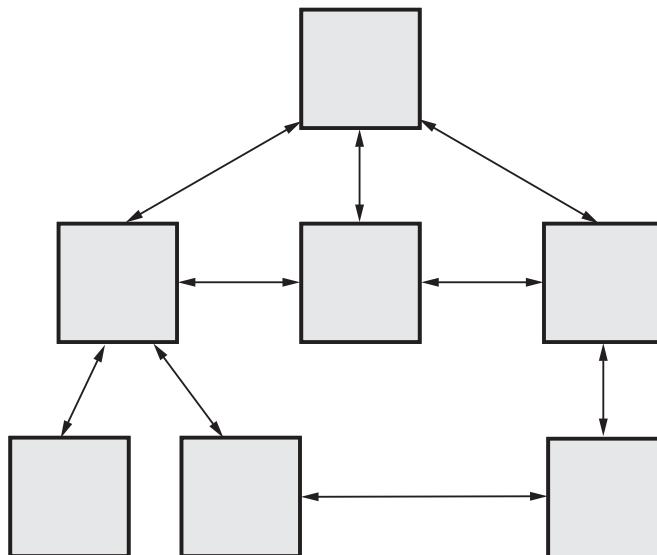


Fig. 5.8.4 Hierarchical structure

4. Networked or pure web structure

In this architecture, the architectural components should be designed in such a manner that control can be passed from one component to every other component.

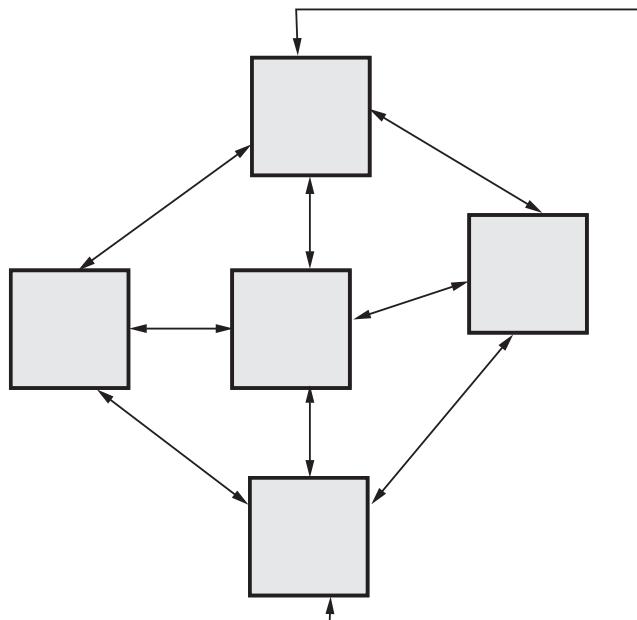


Fig. 5.8.5 Network structure

Web application architecture

The web application architecture is useful for achieving the business objectives. It is defined by **MVC architecture**.

The **Model View Controller (MVC)** architecture is a kind of infrastructure model that decouples user interface from web application functionality and informational content. The architecture is partitioned in three layers Model, View and Controller.

The **model** contains all application specific content and processing logic. It includes content object, access to external information source and all processing functionality.

The **view** defines all the interface specific functions.

The **controller** manages the **access to** model and view layers. It also controls the data flow among them.

The working of MVC architecture can be written something like this -

Step 1 : User requests for some data. This request is submitted to the controller. The controller selects the view object which is appropriate for the user request.

Step 2 : Once the type of request is determined the behavioural request is transmitted to the **model**. The model implements the functionality. Sometimes it can access the corporate database, if needed.

Step 3 : The data developed by the model must be formatted and organised in the appropriate view object.

Step 4 : The view object is then transmitted from application server to the client's machine in order to display the required information in desired form.

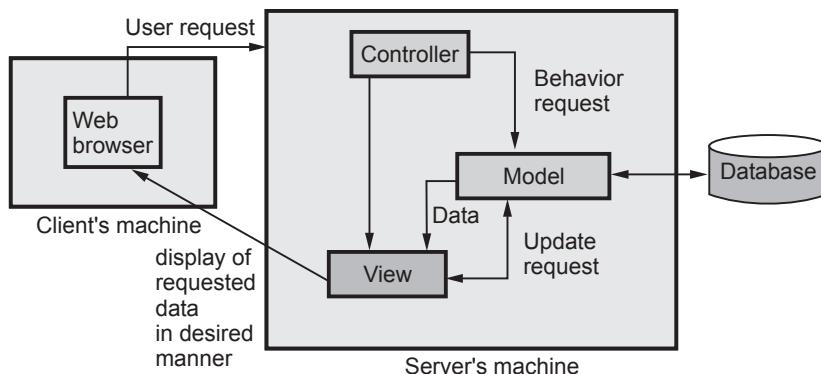


Fig. 5.8.6 Model view architecture

5.8.1.5 Navigation Design

After establishing the web application architecture, the navigation pathways are defined in navigation design. The navigation pathways are nothing but the links which

allow the user to access the content object and functionalities of the web application. To define the navigational pathways designer must do following things -

1. Identify the semantics of navigation for different users of the site.
2. Define the mechanism (syntax) or a method for achieving the navigation.

Navigation semantics

In navigation design the user hierarchy and related use cases are considered. Because each use case is associated with actor. And each actor may use web application differently and therefore have different navigational requirements. Secondly, each actor will define some classes or functionalities.

When user interacts with the web application, he may encounter a series of navigation semantic units (NSU).

Gnaho and Larcher describes the NSU as follows -

The NSU consists of a sub structure called Ways of Navigating (WoN). The WoN represents the navigations ways or paths using which the user can accomplish certain goal. The WoN consists of Navigation Node (NN) which are connected by navigation links. One high level NSU may contain several low level NSUs.

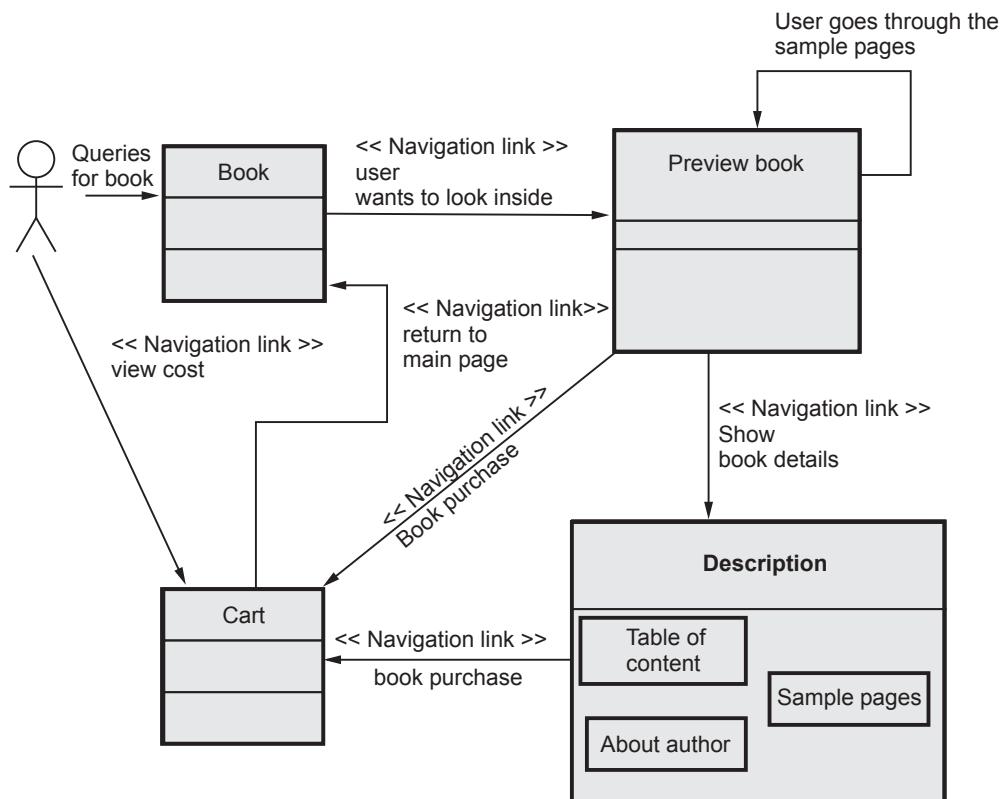


Fig. 5.8.7 Creating NSU

To understand the concept of NSU consider following scenario -

This web application is for online **book** purchasing. A user can select a desired book and can purchase it using on-line **shopping cart**. If he wishes then he can get a **preview** of that book. This preview may contain some **description** such as table of content, sample pages from the book, author's information, cost and so on.

Navigation syntax

In this section we will see various options of creating navigation -

- Navigational links

Using text based links, button, icons or some graphical images the navigational links can be defined.

- Horizontal navigational bars

It consists of major contents containing appropriate links.

- Vertical navigation column

It consists of major contents containing appropriate links. It lists virtually all major content objects within the web application.

- Tabs

It is nothing more than variation of navigation bar or column. By this navigation link all the functional categories on a tab sheet are represented.

- Site maps

It consists of all the content objects and functionalities including the table of content. The links on table of content can display the content object directly.

In addition to this, the web designer should establish some standard conventions.

For example - Icons, or graphical images representing navigational links must be clickable. For text based hyperlinks the appropriate color change mechanism must be applied.

5.8.1.6 Component Level Design

Following are some tasks of modern web application -

1. Perform localized processing for generating the web contents.
2. Enhance navigation capability in dynamic order.
3. Provide computation or data processing capability.
4. Provide the mechanism for database query processing and handling.
5. Establish data interfaces with external corporate system.

These tasks can be accomplished by designing and constructing the program components that are identical in the form to software component for conventional software.



Notes

6

Software Coding and Testing

Syllabus

Coding standard and coding guidelines, Code review, Software documentation, Testing strategies, Testing techniques and test case, Test suites design, Testing conventional applications, Testing object oriented applications, Testing web and mobile applications, Testing tools (Win runner, Load runner).

Contents

6.1 Coding Standard and Coding Guidelines	Summer-2016, 2018,	Marks 7
6.2 Code Review	Summer-2018, 2019,	Marks 4
6.3 Software Documentation		
6.4 Introduction to Software Testing		
6.5 Testing Strategies	Summer-2012, 2016, 2018, 2019, Winter-2019,	Marks 7
6.6 Testing Conventional Applications		
6.7 White Box Testing	Winter-2017, 2018, Summer-2012, 2019,	Marks 7
6.8 Black-Box Testing	Winter-2013,	Marks 7
6.9 Comparison between Black Box and White Box Testing	Winter-2012, Summer-2015, Marks 7	
6.10 Testing Technique and Test Case	Winter-2019,	Marks 3
6.11 Test Suites Design		
6.12 Testing Object Oriented Applications		
6.13 Object-Oriented Testing Strategies		
6.14 Testing Web Applications		
6.15 Testing Mobile Applications		
6.16 Testing Tools		

6.1 Coding Standard and Coding Guidelines

GTU : Summer-2016, 2018, Marks 7

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 instead of getting the job of programmer simplified 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.

6.1.1 Coding Guideline

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

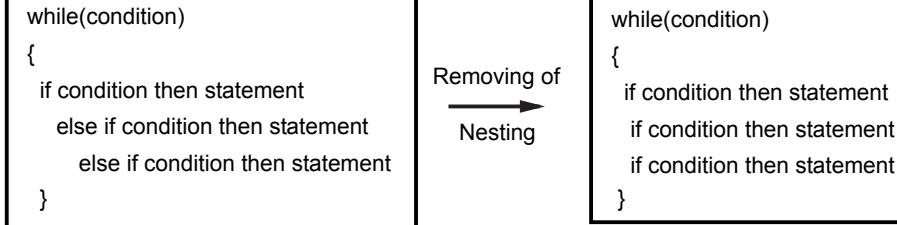


Fig. 6.1.1

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. 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.

8. Use of Meaningful Variable names for specific purpose

Never make use of same variable name in temporary loops for multiple purposes. Each variable must be given some descriptive name so that the program becomes easy to understand.

9. Well Documentation

The code must be well documented with the help of comment statements at appropriate places.

6.1.2 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 ij.

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.

Review Question

1. Explain the various **coding standard**.

GTU : Summer-2016, 2018, Marks 7

6.2 Code Review

GTU : Summer-2018, 2019, Marks 4

- The main goal of code inspection is to find out the detect defects in the coding.
- An inspection is a **review** of a code by a group of peers following a clearly defined process.
- Code inspection **technique** improve the **quality** of the code by finding the defects.
- Following are some **characteristics** of code inspection process -
 1. The **inspection** must be conducted by the **technical people** for technical people.
 2. It is a **structured process** in which every participant have definite role.
 3. The focus of code inspection is to **identify** the problems and **not to solve them**.
 4. The review data is **recorded** and monitored for further improvement.
- The code inspection is performed by a team of reviewer. The **author** of the team is **moderator**.
- The **moderator** has the **overall responsibility** to ensure that the review is done in a **proper manner** and all steps in the review process are followed.
- The code inspection is carried out in **different phases** such as **planning, preparation and overview, group review meeting, rework and followup**.

6.2.1 Planning for Review

- The **objective** of planning is to prepare for code review.
- The **moderator** ensures that the code is ready for inspection.
- The moderator checks the **entry criteria** for the coding. The entry criteria for coding is that the code is **compiled** properly and the **static analysis** tools are applied.
- The **review team** also **gets formed** in this phase.
- Then the code is **distributed** to the review team for **reviewing**. The design documents, relevant checklists and standards are also provided to the team members. With the help of these documents the correctness of code can be ensured.

6.2.2 Self Review

- In **overview and preparation phase**, the code is reviewed by the reviewers.
- The moderator may arrange an **opening meeting**. In this meeting the author may provide a **brief overview** of the product and any **special areas** that need to be looked **at carefully**. But the meeting is optional and can be omitted.
- The main task of this phase is that **each reviewer has to do self review of the code**.
- A reviewer goes through the **entire code** and logs all the **potential defects** he finds in the **self-preparation log**.
- The reviewers also record the time they spent in the self-review.
- Relevant checklists, guidelines, and standards may be used while reviewing.
- The sample self review log is as shown below -

Project Name:

Code For:

ID:

Reviewer Name:

Effort Spent(in Hrs):

Defect List

Name of defect	Description	Location	Status (minor, major, critical)

- Ideally, the self review should be done in one continuous time span. The recommended time is less than two hours.
- This process ends when all the reviewers prepare their own self review log.

6.2.3 Group Review Meeting

- The basic purpose of group review meeting is to prepare a final defect list based on the defected enlisted in the self review log of all the reviewers. There are chances that new important defects might get discovered during the meeting.
- The main outputs of this phase are the defect log and the defect summary report.
- The moderator examines the effort and defect data in the self review logs to confirm that sufficient time and attention has gone into the preparation. When preparation is not adequate, the group review is deferred until all participants are fully prepared.
- If everything is ready then the meeting is held.
- The moderator is the in-charge of this meeting
- The meeting is conducted as follows -

The team member called reader goes through work product line by line. At any line the reviewer may raise some issues or can find new issues while listening to other reviews. The discussion takes place on these issues. If the issue is a defect then moderator accepts it or otherwise explains why that issue is not a defect.

After the discussion the agreement is reached and a team member called scribe records all the identified defects in the defect log. At the end of the meeting, the scribe reads out the defects recorded in the defect log for a final review by the team members. Note that only defects are identified and not the solution. The final defect log is prepared which is the official record of the defects identified in the inspection and may also be used to track the defects.

For analysing the effectiveness of the review the information is summarised and a summary report is prepared. The summary report describes the work product, the total effort spent, different review process activities, total number of defects found for each category and size.

The sample summary report is as shown below -

Project Name	XYZ
Work Product	Code
Name	AAA
Size of the product	30 pages

Review Team	R1,R2,R3,R4,R5
Effort	
Preparation	10 hours
Group Review Meeting	5 hours
Total Effort	15 hours
Defects	
Number of Minor defects	10
Number of Major defects	5
Number of Critical defects	2
Total Number of defects	17
Review Status	Accepted
Recommendation for Next Phase	
Comments	

Review Question

1. Explain the process of code review. **GTU : Summer-2018, Marks 4; Summer-2019, Marks 3**

6.3 Software Documentation

Various kinds of software documents need to be developed during software development process. The software document includes source code, executable files, user manual, Software Requirements Specification(SRS), Design document, test document, installation manual and so on.

Purpose

Good documents are very useful and serve following purpose -

1. Documents enhance the **readability, understandability** and **maintainability** of software product.
2. They **reduce the effort and time required** for maintenance.
3. They are useful in **exploring the system** effectively.
4. They are used to keep track of the **progress** of the project.
5. Good documents also help in **overcoming the manpower turnover** problem. Newcomer can easily and quickly grasp the new project.

Types of Documents

There are two types of software documents -

1. **Internal Documentation** : The internal documentation is a part of source code. Insertion of meaningful **comment statements** at appropriate place in the code, use of meaningful module name and **variable name**, use of appropriate data types, use of enumerated types, are the techniques of internal documentation.
2. **External Documentation** : External documentation can be done by creating various supporting documents such as **SRS, Design Document, Test plan** and so on. These documents are normally created during the software development life cycle activities.

Consistency is the most important and desired feature of software documentation.

6.4 Introduction to Software Testing

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.

6.4.1 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.

6.4.2 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.

6.4.3 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.

Review Question

1. *What is importance of testing practices ? What are the principles of testing practices ?*

6.5 Testing Strategies

GTU : Summer-2012, 2016, 2018, 2019, Winter-2019, Marks 7

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.

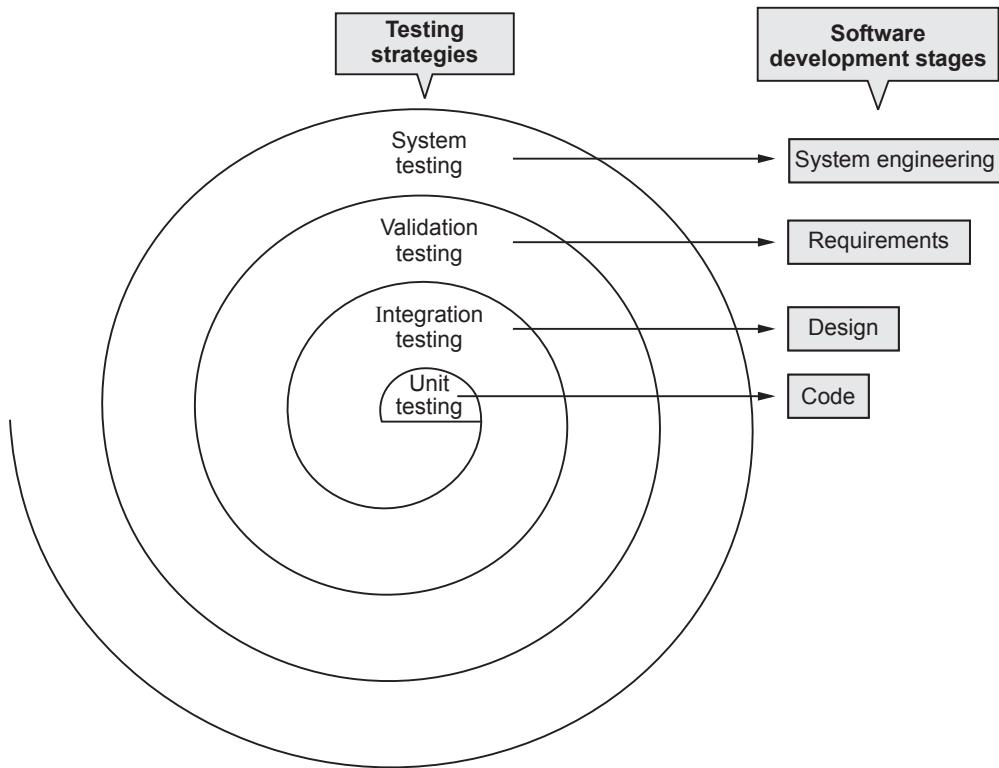
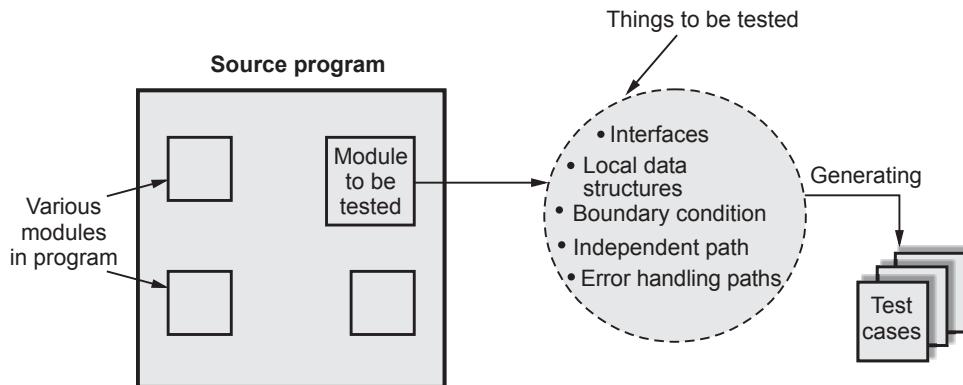


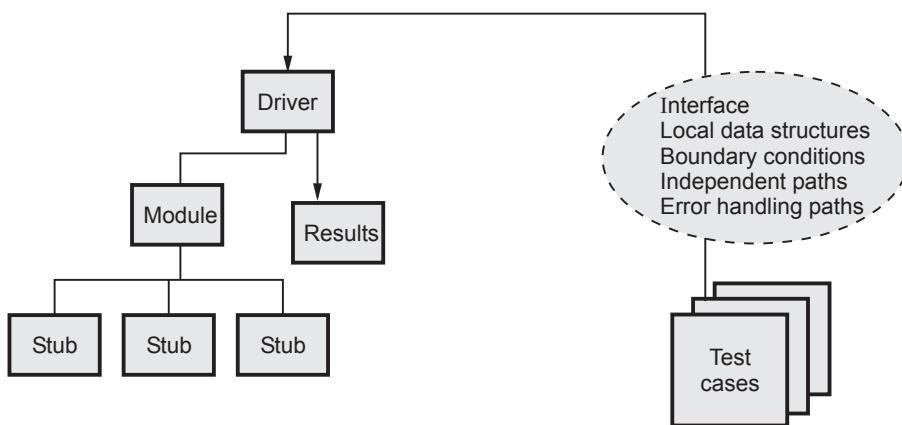
Fig. 6.5.1 Testing strategy

6.5.1 Unit Testing

- 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. 6.5.2 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. 6.5.3.

**Fig. 6.5.3 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.

6.5.2 Integration Testing

- 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

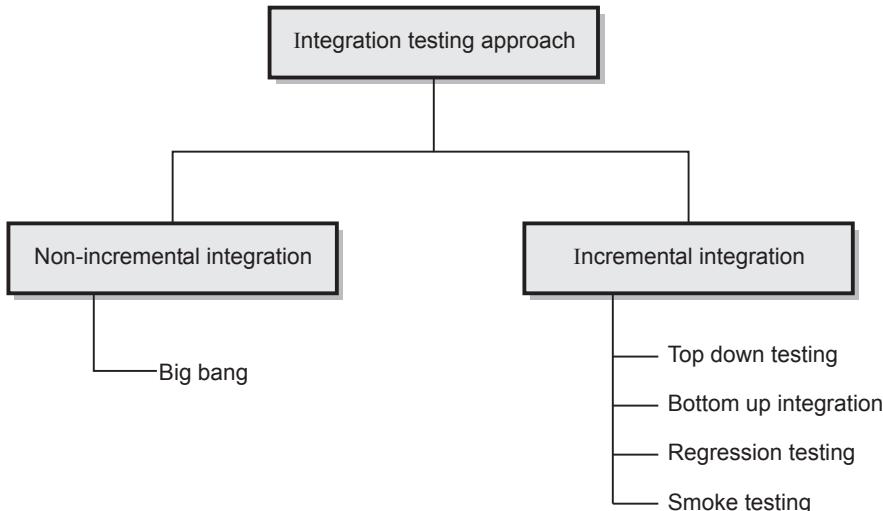


Fig. 6.5.4 Integration testing approach

- 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

6.5.2.1 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 :

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.

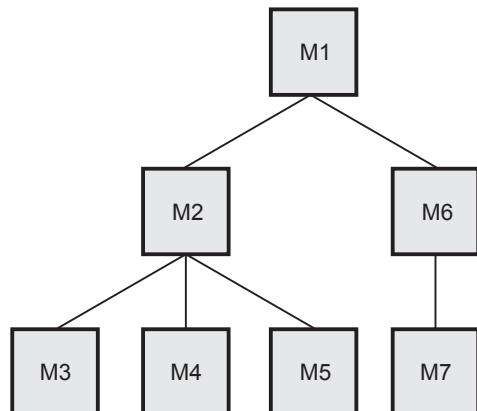


Fig. 6.5.5 Program structure

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 :

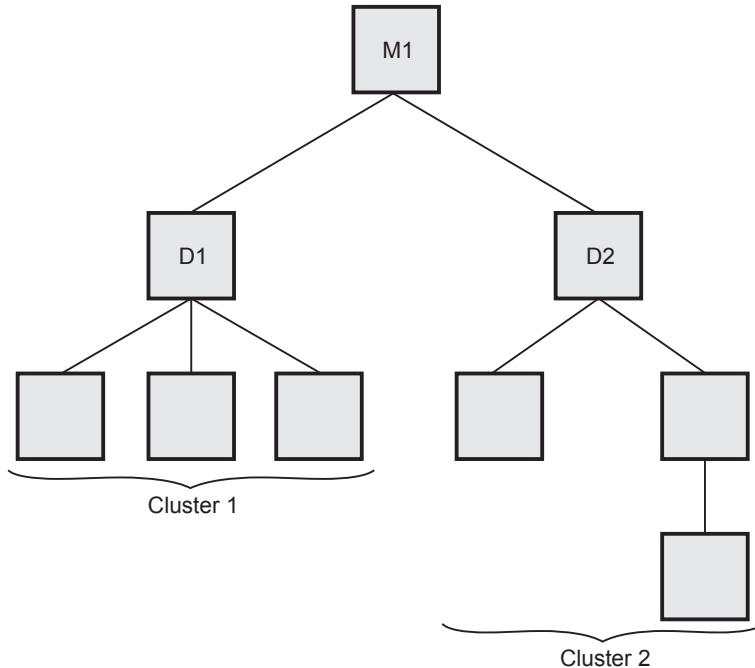


Fig. 6.5.6 Bottom up integration testing

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.

6.5.2.3 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.

6.5.2.4 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.

6.5.3 Validation Testing

- 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.

6.5.3.1 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.
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.

Difference between Alpha and Beta Testing

Alpha testing	Beta testing
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.
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.

6.5.4 System Testing

The system test is a series of tests conducted to 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.

6.5.4.1 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.

6.5.4.2 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.

6.5.4.3 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.

6.5.4.4 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 Questions

1. Explain testing strategy for conventional software architecture. Draw the spiral diagram showing testing strategies with phases of software development. **GTU : Summer-2012, Marks 7**

2. What are the different levels of testing ? Explain any one with suitable example. **GTU : Summer-2016, Marks 7**

3. Explain integration testing. **GTU : Summer-2018, 2019, Marks 7**

4. What are the different testing strategies? Explain any one with suitable example. **GTU : Winter-2019, Marks 7**

5. Compare and contrast alpha and beta testing. **GTU : Summer-2018, Marks 3**

6. What are the different levels of testing ? Briefly discuss the goal of each level. **GTU : Summer-2018, Marks 3**

6.6 Testing Conventional Applications

There are two general approaches for the software 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.

6.7 White Box Testing

GTU : Winter-2017, 2018, Summer-2012, 2019, Marks 7

6.7.1 Basis Path Testing

In this method the procedural design using basis set of execution path is tested. This basis set ensures that every execution path will be tested at least once.

6.7.1.1 Flow Graph Notation

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 (See Fig. 6.7.1) -

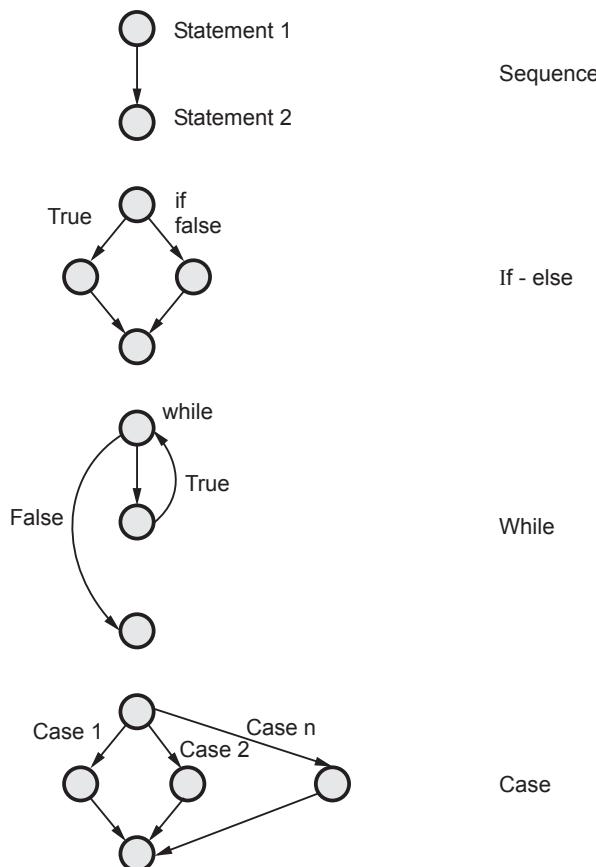


Fig. 6.7.1

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)     {
7)       printf ("Element is present");
8)     }
9)     else
10)    if (a [mid] < key)
11)      bottom = mid + 1;
12)    else
13)      top = mid - 1;
14)  } // end of while
15) } // end of search
```

The flow graph will be

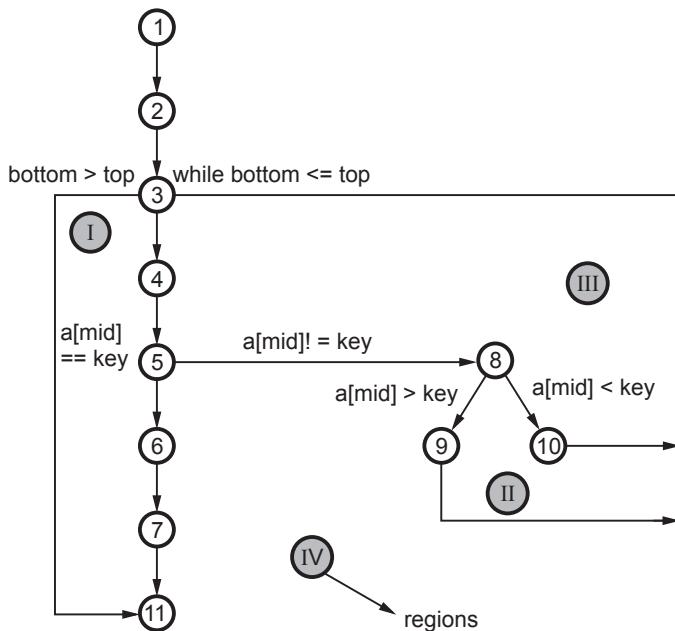


Fig. 6.7.2

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'.(See Table on next page)

Test case id	Test case name	Test case description	Test steps		Test case status	Test status (P / F)	Test priority	Defect severity
			Step	Expected				
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 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			

Example 6.7.1 Draw the flow graph for finding maximum of three numbers and derive the testcases using cyclomatic complexity.

Solution : The flow graph for given program is - (From Fig. 6.7.3)

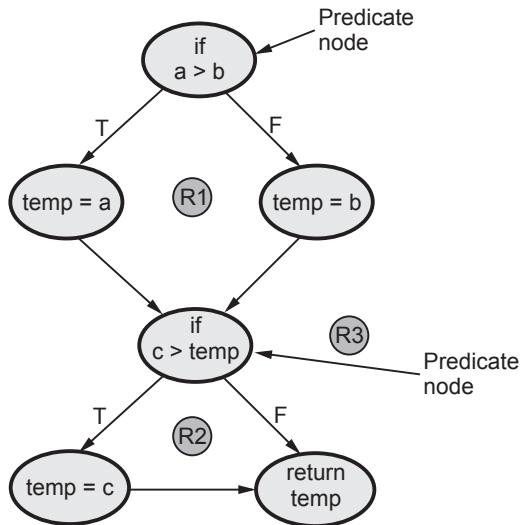


Fig. 6.7.3 Flow graph for finding maximum of three numbers

$$\text{Cyclomatic complexity} = E - N + 2 = 7 - 6 + 2 = 3$$

$$\text{Cyclomatic complexity} = P + 1 = 2 + 1 = 3$$

$$\text{Cyclomatic complexity} = \text{Regions encountered} = 3$$

Hence cyclomatic complexity of given program is 3.

6.7.1.2 Graph Matrices

Definition : Graph matrix is a square matrix whose size is equal to number of nodes of the flow graph.

For example consider a flow graph -

The graph matrix will be

For computing the cyclomatic complexity. Following steps are adopted -

Step 1 : Create a graph matrix. Mark the corresponding entry as 1 if node A is connected to node B.

Step 2 : Count total number of 1's from each row and subtract 1 from each corresponding row.

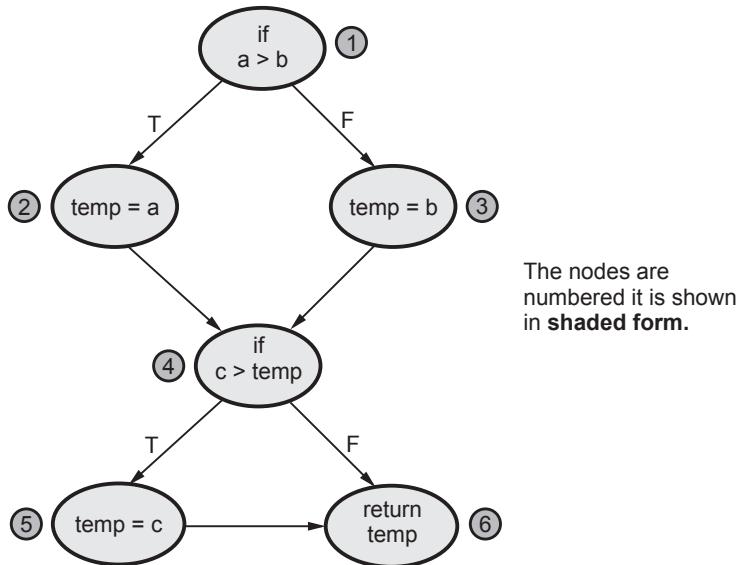


Fig. 6.7.4 Flow graph

Step 3 : Add cyclometric complexity.

	1	2	3	4	5	6	
1		1	1				$2 - 1 = 1$
2				1			$1 - 1 = 0$
3				1			$1 - 1 = 0$
4					1	1	$2 - 1 = 1$
5						1	$1 - 1 = 0$
6							$2 + 1 = 3$

the results of each row and add 1 to it. The resultant value is the cyclomatic complexity.

How to extend graph matrix for use in testing ?

Following properties indicate how to extend graph matrix for use in testing.

- i) The number of times of link between the nodes get executed.
- ii) The processing time spent in traversal of a link.
- iii) The number of resources required.
- iv) The amount of memory required to traverse the link.

6.7.2 Control Structure Testing

- The structural testing is sometime called as white box testing.

- In structural testing derivation of test cases is according to program structure. Hence knowledge of the program is used to identify additional test cases.
- Objective of structural testing is to exercise all program statements.

6.7.2.1 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.

6.7.2.2 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.

where

- $n = 0$ that means skip the loop completely.
- $n = 1$ that means one pass through the loop is tested.
- $n = 2$ that means two passes through the loop is tested.
- $n = m$ that means testing is done when there are m passes where $m < n$.
- Perform the testing when number of passes are $n - 1$, $n, n + 1$.

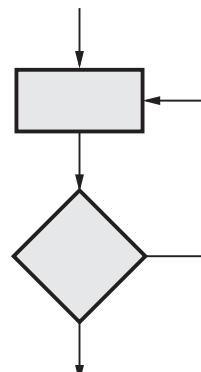


Fig. 6.7.5 Simple loop

2. Nested loops :

The nested loop can be tested as follows.

- i) Testing begins from the innermost loop first. At the same time set all the other loops to minimum values.
- ii) The simple loop test for innermost loop is done.
- iii) Conduct the loop testing for the next loop by keeping the outer loops at minimum values and other nested loops at some specified value.
- iv) This testing process is continued until all the loops have been tested.

3. Concatenated loops :

The concatenated loops can be tested in the same manner as simple loop tests. (Refer Fig. 6.7.7)

4. Unstructured loops :

The testing cannot be effectively conducted for unstructured loops. Hence these types of loops needs to be redesigned. (Refer Fig. 6.7.8)

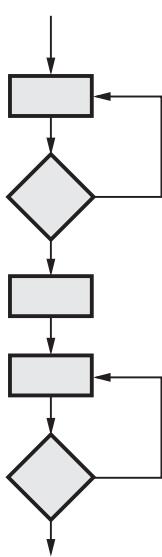


Fig. 6.7.7 Concatenated loops

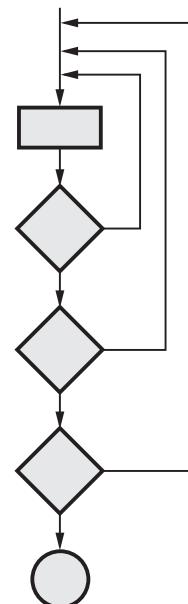


Fig. 6.7.6 Nested loops

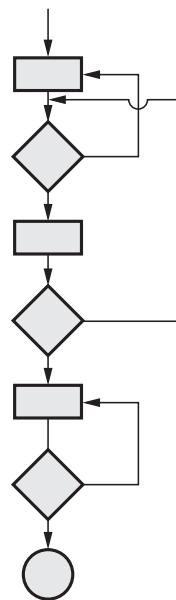


Fig. 6.7.8 Unstructured loops

6.7.2.3 Data Flow Testing

- The testing based on data flow mechanism performs testing on definitions and uses of variables in the program.
- In this method of testing, definition and use chain (DU chain) is required. The DU chain is obtained by identifying the def and use pairs from the program structure. This strategy of testing is also called as DU testing strategy.
- Set **DEF(n)** contains variables that are defined at node n.
- Set **USE(n)** contains variables that are read or used at node n.

For example :

```

1.   s:=0;
2.   a:=0;
3.   while(a<b) {
4.       a:= a+2;
5.       b=b - 4;
6.       if(a+b<20)
7.           s:=s+a+b;
     else
8.           s:=s+a-b;
}

```

For above given programming lines the DU chain will be -

```

DEF(1) = {s} USE(1) = {ϕ}
DEF(2) = {a} USE(2) = {ϕ}
DEF(3) = {ϕ} USE(3) = {a, b}
DEF(4) = {a} USE(4) = {a}
DEF(5) = {b} USE(5) = {b}
DEF(6) = {ϕ} USE(6) = {a, b}
DEF(7) = {s} USE(7) = {s, a, b}
DEF(8) = {s} USE(8) = {s, a, b}
DEF(9) = {ϕ} USE(9) = {ϕ}
DEF(10) = {ϕ} USE(10) = {ϕ}

```

- Identify all DU pairs and construct test cases that cover these pairs.
- Identify all DU paths : That means for each DU pair (n₁, n₂) for variable a, exercise all possible paths n₁...n₂ that are clear of definitions of a.

Identify all uses : That means for each DU pair (n₁, n₂) for a, exercise atleast one path n₁...n₂ that is clear of definitions of a.

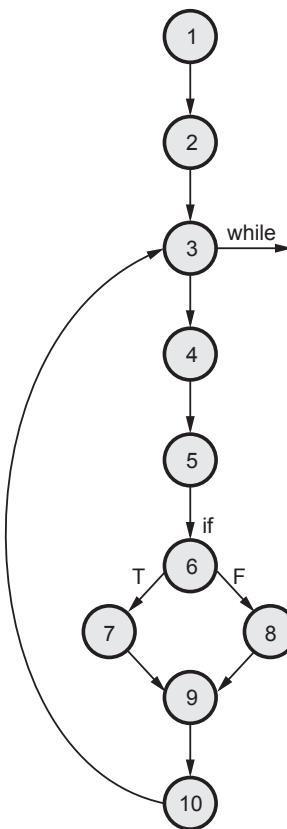


Fig. 6.7.9 Flow graph

Review Questions

1. Explain the following : i) Condition testing ii) Loop testing
 2. What is basis path testing ? What is cyclomatic complexity ? How is it determined for a flow graph ? Illustrate with an example.
 3. Basis path testing covers all statements in a program module. Justify with example.
 4. Explain the graph matrix and loop testing methods.
 5. What is cyclomatic complexity ? How is it determined for a flow graph ? Illustrate with example.
 6. What is cyclomatic complexity? Define steps to find the cyclomatic complexity using flow graph
- GTU : Summer-2012, Marks 7**
7. Explain white box testing with an example. **GTU : Winter-2017, Summer-2019, Marks 7**
 8. What is cyclomatic complexity ? Define steps to find cyclomatic complexity using flow graph.
- GTU : Winter-2017, 2018, Marks 3**

6.8 Black-Box Testing**GTU : Winter-2013, Marks 7**

- 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

6.8.1 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.

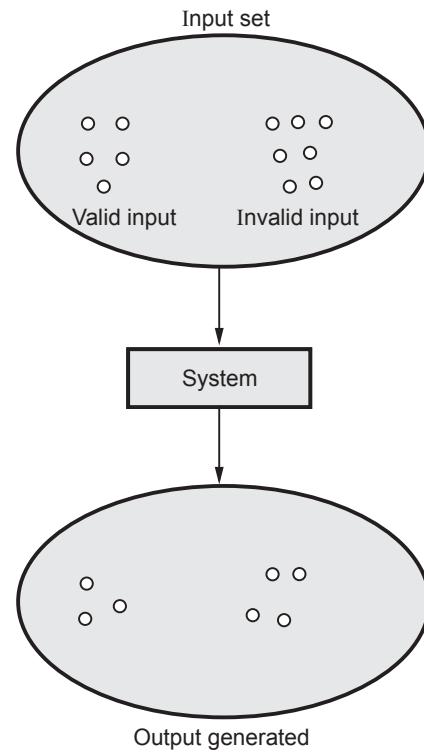


Fig. 6.8.1

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.

6.8.2 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

6.8.3 Graph based Testing

Graph based testing :

- In the graph based testing, a graph of objects present in the system is created.
- The graph is basically a collection of nodes and links. Each node represents the object that is participating in the software system and links represent the relationship among these objects.
- The node weight represents the properties of object and link weight represents the properties or characteristics of the relationship of the objects.
- After creating the graph, important objects and their relationships are tested.

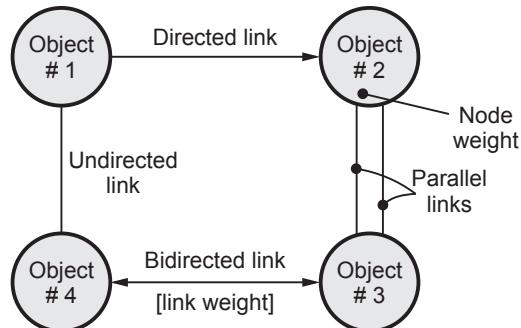


Fig. 6.8.2 Graph notations

6.8.4 Orthogonal Array Testing

There are many applications for which very **small number of input** is needed and values that each input requires might be **bounded**. In such a situation the number of **test cases** is relatively **small** and can be **manageable**. But if the number of input gets increased then it will increase there will be large number of test cases. And testing may become impractical or impossible. **Orthogonal array testing** is a kind of testing method which can be applied to the applications in which **input domain is relatively small** but there could be large number of test cases. Using orthogonal array testing method, only faulty regions can be tested and thus the number of test cases can be reduced.

Orthogonal arrays are two dimensional arrays of numbers which possess the interesting quality that by choosing any two columns in the array you receive an even distribution of all the pair-wise combinations of values in the array. Following are some important terminologies used in orthogonal testing methods -

Runs

It denotes the number of rows in the array. These can be directly translated to the test cases.

Factors

It denotes the number of columns in the array. These can be directly translated to maximum number of variables that can be handled by the array.

Level

This number denotes the maximum number of values that a single factor(column) can take.

L9 orthogonal array

This array is used to generate the test cases. This array has a **balancing** property. That means the testing can be done uniformly by executing the test cases generated by the L9 orthogonal array.

Example

Consider that we want to develop an application which has three sections - top, bottom and middle.

- Associated with each section we will consider one variable. That means now we have to analyse only three variables.
- These variables can be assigned with Boolean values and hence the values can be true or false.
- If we decide to test it completely then there would be $2^3 = 8$ test cases
- By orthogonal array testing method, mapping the values to L9 orthogonal array would be -

	factor1	factor2	factor3
Run1	false	false	false
Run2	true	false	false
Run3	false	true	false
Run4	false	false	true

Variable from 1st section is true Variable from second section is true Variable from third section is true

- Thus the values are left to all the levels.
- Now the test cases would be

1. Display the application without any section.
2. Display the application with top section only.
3. Display the application with the middle section only.
4. Display the application with the bottom section only.

Thus there are only four test cases in all and no need to test all the combinations. In fact we have considered this test cases by considering the single parameter. This kind of testing is called **single mode fault detection**. By considering these test case the faults can be fixed.

Double mode fault detection - In this method, two parameters can be considered at a time for determining the number of test cases.

Advantage

The orthogonal array testing approach enable the developer to provide good test coverage with very few test cases.

Review Question

1. *What is black box testing? Explain any one technique to carry out black box testing.*

GTU : Winter-2013, Marks 7

6.9 Comparison between Black Box and White Box Testing

GTU : Winter-2012, Summer-2015, Marks 7

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. Explain black box and white box testing

GTU : Summer-2015, Marks 7

2. What is software testing? What is the role of tester? Compare Black box and white box testing.

GTU : Winter-2012, Marks 7

6.10 Testing Technique and Test Case

GTU : Winter-2019, Marks 3

- Test cases are used to determine the **presence of fault** in the program. Sometimes even if there is **some fault** in our program the correct output can be obtained for some inputs. Hence it is necessary to exercise those set of inputs for which faults (if any) can be exposed off.
- Executing test cases require money because - i) machine time is required to execute test cases ii) human efforts are involved in executing test cases. Hence in the project testing minimum number of test cases should be there as far as possible.
- The testing activity should involve two goals -
 - i) Maximize the number of errors detected.
 - ii) Minimize the number of test cases.
- The selection of test case should be such that faulty module or program segment must be exercised by at least one test case.
- Selection of test cases is determined by some criteria which is called as **test selection criterion**. Hence the test selection criterion T can be defined as the set of conditions that must be satisfied by the set of test cases.
- Testing criterion are based on two fundamental properties - **reliability** and **validity**.
- A test criterion is reliable if all the test cases detect same set of errors.
- A test criterion is valid if for any error in the program there is some set which causes error in the program.
- For testing criteria there is an important theorem - "if testing criterion is valid and reliable if a set satisfying testing criterion succeeds then that means program contains no errors".
- Generating test cases to satisfy criteria is complex task.
- The test case specification records the results of the testing, the conditions used for testing particular unit. It also specifies the expected test results. It records the outcome of test cases (Pass/Fail). The sample structure of a test case specification is as given below -

Precondition :

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								
3								
4								

- Test case specification is the major activity in the testing process. Careful selection of test cases will help in conducting proper testing.
- There are two basic reasons why test case specification should be before using them for testing - Firstly it will assist the tester to reveal as many errors as possible from the program and secondly the high quality code can be produced.

6.10.1 Test Case Execution

- After test case specification the next step is to execute the test cases.
- Execution of test cases requires **driver** and **stubs**.
- The test case execution procedure is sometimes called as **test procedure specification**. This specification consists of special requirements required for setting up the test environment. It also describes the **format of reports** of testing results.
- The most common report produced during testing is test summary report and **error report**.
- The **test summary report** describes the execution of entire test cases and is used in project management. The **error report** specifies the errors and defects encountered during testing.
- During the execution of test cases it is important to estimate the **test efforts**. **Testing effort** is the total effort actually spent by the team in testing activities, and is an indicator of whether or not sufficient testing is being performed.

Review Question

1. Explain concept of test case.

GTU : Winter-2019, Marks 3

6.11 Test Suites Design

- Test suite is a collection of various tests that help the tester in executing and reporting the test case execution.
- Various test cases can be added to test suites.
- The test suite can be generated from the test plan. Refer Fig. 6.11.1.

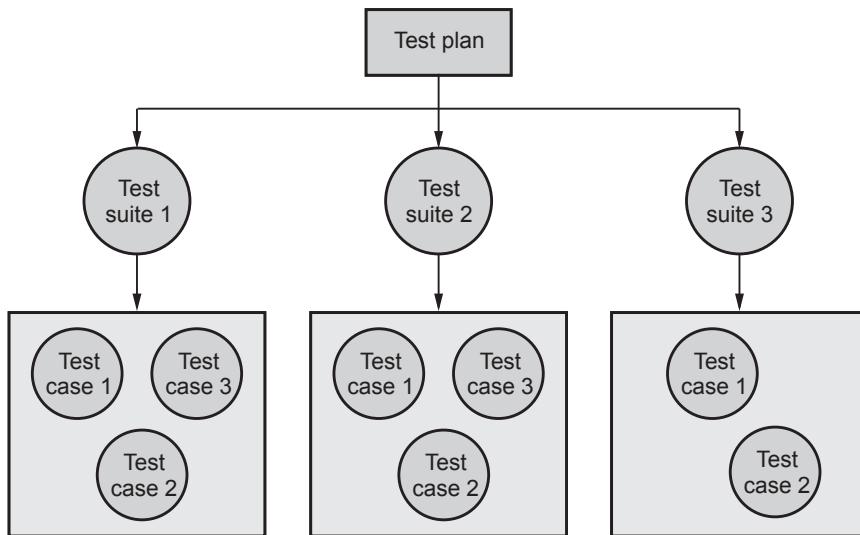


Fig. 6.11.1 Test suite preparation

- The test suite often also contains more detailed instructions or goals for each collection of test cases.
- The test suit document contains the fields such as - a test case ID, test step or order of execution number, related requirement(s), depth, test category, author, and check boxes for whether the test is manual or and has been automated.
- Each test case represents some scenario. For example a test suite for online purchase contain various test cases as -
 - Test Case#1: Login
 - Test Case#2: Search for Product
 - Test Case#3: Add Product to Cart
 - Test Case#4: Checkout
- "Test suites can identify gaps in a testing effort where the successful completion of one test case must occur before you begin the next test case."

6.12 Testing Object Oriented Applications

The object oriented architecture is a layered subsystem that contains various **collaborating classes**. The collaborating classes perform various functions to achieve the system requirements.

In this section we will discuss various object oriented testing methods.

6.12.1 Conventional Test Case Design Methods

White box testing : The white box testing methods can be applied to operations defined within the class. Various testing methods such as loop testing, basis path testing are useful in testing every programming statements.

Black box testing : The black box testing methods are also useful for OO system testing. The use cases in object oriented design provide the input to the black box testing method.

6.12.2 Fault based Testing

- The fault based testing method is a kind of testing method in which **plausible faults** (apparently valid) are identified.
- The **integration testing** detects three types of faults and those are wrong operation used, unexpected result, incorrect invocation.
- For determining the plausible faults various operations are invoked.
- The integration testing is applied to data attributes and operations of the class

6.12.3 Scenario based Testing

- The **drawback of fault based system** is that it does not focus on two main areas and those are 1. What are the users' requirements 2. interactions among various subsystems.
- Scenario based subsystem focuses on what the user wants from the system (users expectations).
- Many interaction errors are uncovered by building the scenarios. Hence test cases that are created for scenario based testing are **complex**.
- Scenario based testing focuses on **multiple subsystems using a single test**.

Example - Let us take an example of a customer who has a credit card. Suppose he lost his credit card. So informs bank about it. Being the customer of that bank, his history information must be stored in the banks computer. Ideally, by simply changing the status of his card as a 'renewal of card' should make his history available. But many times the complete information of that customer has to be entered.

So if scenario testing would have been done for this test scenario this bug would have been found well in advance.

6.13 Object-Oriented Testing Strategies

- The primary objective of testing for object oriented software is to **uncover as much errors as possible** with manageable amount of efforts in realistic time span.
- The object oriented testing strategy is identical to conventional testing strategy. The strategy is start **testing in small** and work outward to **testing in large**. The basic unit of testing is **class** that contains attributes and operations. These classes **integrate** to form object oriented architecture. These are called **collaborating classes**.

6.13.1 Unit Testing in OO Context

- **Class** is an encapsulation of data attributes and corresponding set of operations.
- The **object** is an instance of a class. Hence objects also specify some data attributes and operations.
- In object oriented software the focus of unit testing is considered as classes or objects.
- However, operations within the classes are also the testable units. In fact in OO context the operation can not be tested as single isolated unit because one operation can be defined in one particular class and can be used in multiple classes at the same time. For example consider the operation **display()**. This operation is defined as super class and at the same time it can be used by the can be multiple derived classes. Hence it is not possible to test the operation as single module.
- Thus class testing for OO software is equivalent to unit testing for conventional software. In conventional software system, the algorithmic details and data that flow are units of testing but in OO software the encapsulated classes and operations (that are encapsulated within the class and state behavior of class) are the unit of testing.

6.13.2 Integration Testing in OO Context

- There are two strategies used for integration testing and those are -
 1. Thread based testing
 2. Use-based testing.
- The **thread based testing** integrates all the classes that respond to one input or event for the system. Each thread is then tested individually.

- In **use-based testing** the independent classes and dependent classes are tested. The **independent classes** are those classes that uses the server classes and **dependant classes** are those classes that use the independent classes. In use based testing the independent classes are tested at the beginning and the testing proceeds towards testing of dependent classes.

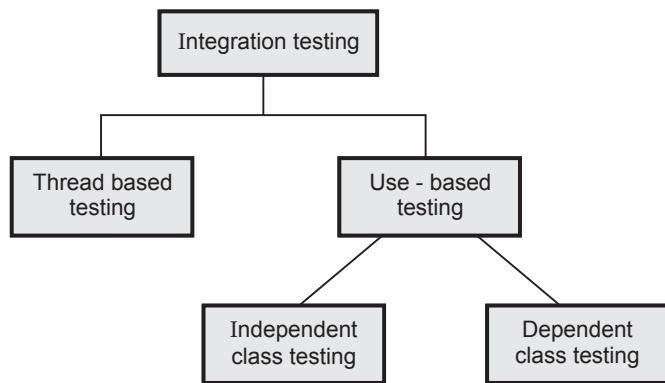


Fig. 6.13.1 Integration testing in OO context

- Drivers and stub :** In OO context the driver can be used to test operations at lowest level and for testing whole group of classes. The stub can be used in the situations in which collaborations of the classes is required and one collaborating class is not fully implemented.
- The **cluster testing** is one step in integration testing in OO context. In this step all the collaborating classes are tested in order to uncover the errors.

6.13.3 Difference between OO Testing Strategy and Conventional Testing Strategy

- The testing strategies applied to conventional software are based on **unit testing**, **integration testing**, **validation testing** and **system testing**. But in object oriented testing, the unit testing loses some of its meaning and integration testing changes significantly.
- The object oriented testing strategy is also based on the principle - "test in small" and then work "in the large".
 - Testing in the small** involves **class attributes and operations**; the main focus is on communication and collaboration within the **class**.
 - Testing in the large** involves a series of regression tests to uncover errors due to **communication and collaboration among classes**

Finally, the system as a whole is tested to detect errors in fulfilling requirements.

Review Questions

- Explain the object oriented testing strategies.
- Differentiate between OO testing strategy and conventional strategy.

6.14 Testing Web Applications

The process of testing begins with content testing. Fig. 6.14.1 shows the design pyramid with associated testing strategies. The testing flow proceeds from left to right and from top to bottom.

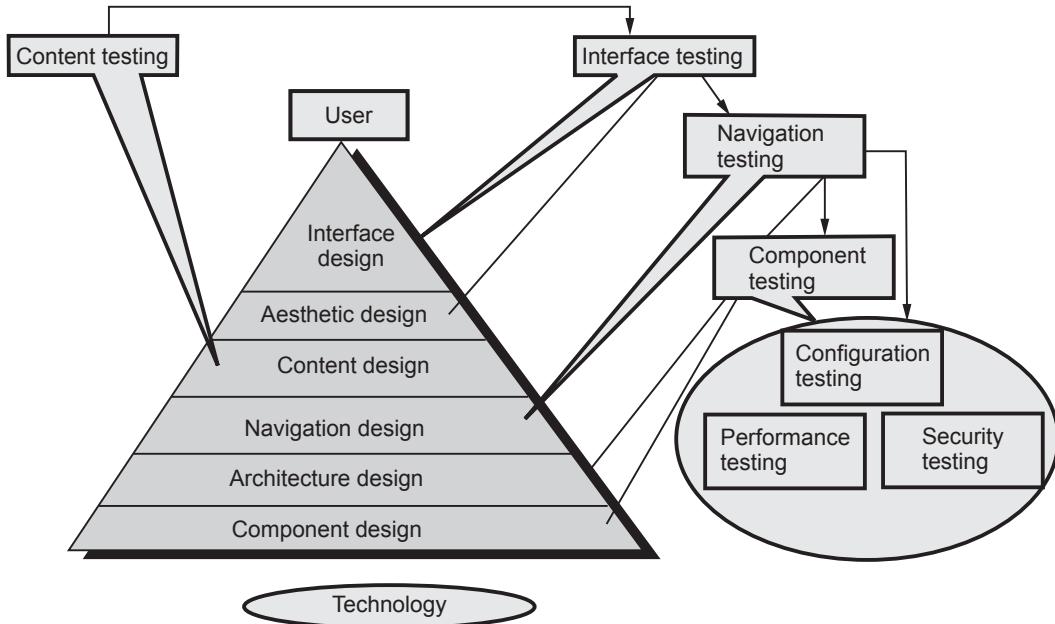


Fig. 6.14.1 Testing process

Content testing

Content testing is for uncovering the errors from the web document. Such testing is similar to proof editing of the document. The typographical error, grammatical mistakes, errors in cross referencing, errors in graphics can be exposed off.

User interface testing

In this testing, the interaction mechanism and aesthetic aspects of the user interface are tested.

Poorly implemented interaction mechanisms, inconsistencies and ambiguities caused in newly introduced interfaces - all such errors are uncovered in user interface testing.

Navigation testing

Navigation testing is concerned with use cases. The test cases for navigation testing are designed to exercise usage scenario against navigation design. Navigation mechanisms implemented within the interface layout are tested against use-cases and NSUs. In this kind of testing the use cases are identified correctly and are corrected.

Component testing

In component testing the **content and functional units** are tested. For web application the term unit refers to the **web page** (containing web contents, navigation links and processing elements) or it may be a **functional component** that provides the services directly to the user or it can be infrastructure component. The testing of each functional unit is done in the same manner as an individual module tested in conventional software. This testing is mainly of **black box** testing but sometimes if the processing is complex then the white box testing may also be used. After constructing the web architecture, the navigation and component testing are used as integration test. For integration testing, the web architecture is considered. If the web architecture is linear, grid or simple hierarchical structure then all the web pages can be integrated and tested in the same manner as a conventional software.

Regression testing is applied to ensure that no side effects occur. Cluster testing integrate all the collaborating web pages and those are tested for uncovering the errors in collaboration.

Configuration testing

Configuration testing uncovers the errors particularly from client server environment. A cross reference matrix is used in this testing. All probable browsers, operating systems, hardware platforms communication protocols are used for testing compatibility the web application. Tests are conducted for all the above mentioned configuration and errors are uncovered.

Security testing

This testing is useful to expose the vulnerabilities in web applications and its environment.

Performance testing

In performance testing following things are assessed -

- How is the response time and reliability of a web application when user traffic gets increased ?
- Which are those components responsible for degradation of performance of the web application ?
- What is the impact of performance degradation on overall objective and requirement of web application ?

6.15 Testing Mobile Applications

Mobile testing or mobile application testing is a process by which application software developed for handheld mobile devices is tested for its functionality, usability

and consistency. There is change in behavior and usage pattern of mobile applications. Testing mobile application is more complex and time consuming as compared to traditional desktop and web application.s Hence there are some major challenges in mobile application testing

Challenges in Mobile Testing

1. There is a **huge number of mobile devices** available ranging from handsets, to smart phones, to tabs, to ipads and wearable tech provides a huge diversity of environments which the mobile application faces.
2. There is **diversity in mobile network operator** such as CDMA, GSM and so on. The mobile application has to work in these diverse environment.
3. A mobile app can be a native app, a web app or a hybrid app which has both contents. Testing of each such **app type is different** than another as their implementation is quite different from one another.
4. There are **different operating systems** for mobile such as Android, ios, Windows and so on. Each operating system has its own limitation. Testing the application to work on diverse operating system is a real challenge.
5. **Choosing the test method** for mobile application is also one challenge. The mobile application can be tested on emulators or on actual devices. Some issues can not be tested on emulators and trying out different types of devices is time and cost consuming.

Criteria used for Mobile Testing

1. Testing the mobile application to ensure that it is working as per the requirements.
2. Testing the mobile application to work on different OS platforms, devices and networking environment.
3. Testing the mobile application during the interrupts such as incoming calls, SMS, notifications, data cable insertion or removal and so on.
4. Testing the mobile application for installation, un-installation, updation on the devices.
5. Testing the mobile application for security i.e. to check for attacks, authentication and data security.

Review Question

1. *What is mobile testing ? Mention the challenges in mobile testing.*

6.16 Testing Tools

Automated software testing is becoming more and more important for many software projects in order to automatically verify key functionality.

Various **areas** in which automated **testing tools are used** are -

1. The automated testing tools can be used for reviews, walkthroughs and inspection.
2. The tools can be used for various activities such as regression testing, defect management, test management and so on for a stable system.
3. Configuration tools can be used for configuration control and version management.

Advantages of using Automated Testing Tools

1. The testing becomes reliable and there is no chance of human error.
2. The testing tools are reusable.
3. The testing becomes efficient due to use of automated tools.
4. The cost involved in testing gets reduced due to use of automation.

Disadvantages of using Automated Testing Tools

1. The high investment is needed in training the people to use the tool.
2. The people involvement is required for test preparation.
3. Many times a lot of testing areas left uncovered.

Let us discuss some examples of testing tools

6.16.1 Win Runner

WinRunner is Mercury Interactive's enterprise automation tool. It is the most commonly used automated testing tool.

Features

- 1. Automatic Recovery :** It is possible for the user to set up various operations that will become activated if an exception event appears. The recovery manager will give you a wizard that will allow you to set up a scenario for recovery.
- 2. Silent Installation :** The unattended installation mode can be set for WinRunner.
- 3. Support For Various Environments :** WinRunner includes support for Internet Explorer 6.x and Netscape 6.x, Windows XP and Sybase's Power Builder 8, in addition to 30+ environments.
- 4. Cost Effective :** WinRunner provides the most powerful, productive and cost-effective solution for verifying enterprise application functionality.

5. **Support Multiple Data Combination :** WinRunner has an ability to use numerous data combinations for one test. The DataDriver Wizard has been designed for automatic processing of large amounts of data.
6. **Multiple Verification :** It can offer checkpoints for text, URL, GUI and databases, and this will give the testers the ability to compare the expected outcomes with the outcomes that occur. In addition to this, it is able to find specific problems with various objects that are GUI based.

6.16.2 Load Runner

HP Load runner is a software testing tool from Hewlett-Packard. It is used to test applications, measuring system behaviour and performance under load. HP Load runner can simulate thousands of users concurrently using application software, recording and later analysing the performance of key components of the application.

Features

1. It has excellent **monitoring and analysis interface** where tester can see reports in easy to understand colored charts and graphics.
2. It uses C as a default programming language. However, it also **supports** other languages like **Java and Visual Basic**.
3. **No need to install** it on the server under test. It uses native monitors.
4. It has a support for most of the **commonly used protocols**.
5. It has **GUI generated scripts** which can be modified as per the requirements.
6. This tool can quickly point out the effect of the wide area network (WAN) on **application reliability, performance, and response time**.

Review Questions

1. Explain the concept of automated testing tool along with advantages and disadvantages of it.
2. Write a note on - 1) WinRunner 2) LoadRunner.



7

Quality Assurance and Management

Syllabus

Quality concepts and software quality assurance, Software reviews (Formal technical reviews), Software reliability, The quality standards : ISO 9000, CMM, Six sigma for SE, SQA plan.

Contents

7.1 Quality Concepts	Summer-2019,	Marks 3
7.2 Software Quality Assurance (SQA)	Winter-2011, 2013, 2018, Summer-2012, 2014, 2016, 2018,	Marks 7
7.3 Software Reviews	Winter-2018, 2019, Summer-2019,	Marks 3
7.4 Software Reliability		
7.5 Quality Standards	Summer-2014, Winter-2017, 2018,	Marks 7
7.6 SQA Plan		

7.1 Quality Concepts

GTU : Summer-2019, Marks 3

There are many definitions of software quality. In simple words, the software quality means 'how well the software works'. Furthermore we can also state that controlling the variation or differences is the **key** to high quality software product. Let us see "*what is software quality ?*"

7.1.1 Quality

Software quality can be defined as "the **conformance** to explicitly stated **functional** and **performance requirements**, explicitly **documented development standards** and **implicit characteristics** that are expected of all **professionally developed software**".

- There are two kinds of quality

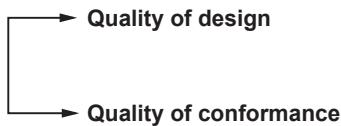


Fig. 7.1.1

Quality of design is the characteristics of the item which is specified for the designer. For example, if a temperature control system is designed, then it should display the temperature with maximum limit of 100 degree centigrade. This is what the basic characteristic of that system is, and at the time of design of the product this issue must be focused.

Quality of conformance is the degree to which the design specifications are followed during manufacturing. If the degree of conformance is **more** then it indicates **higher quality**.

Thus in software development process **quality of design** is concerned towards requirements, specifications and design of the system and **quality of conformance** is concerned with implementation.

- Along with quality of design and quality of conformance, **customer's satisfaction** is very important factor in any software product.
- According to **Robert Glass** - an authority in software field,

$$\text{User satisfaction} = \text{Compliant product} + \text{Good quality} + \text{Delivery within budget.}$$

7.1.2 Quality Control

- Quality control is a process in which activities are conducted in order to maintain the quality of product. These activities are series of inspections, reviews and tests

used throughout the software process. These activities ensure whether each work product is satisfying the requirements imposed on it.

- While applying the quality control there should be a **feedback loop** to the process which generates the work product. With the help of such feedback we can tune the process if it does not satisfy the requirements. The feedback loop helps in minimizing the defects in the software product.
- The quality control activities can be fully automated or it can be completely manual or it can be a combination of automated tools and manual procedures.

7.1.3 Quality Assurance

Definition of quality assurance : *It is planned and systematic pattern of activities necessary to provide a high degree of confidence in the quality of a product. It provides quality assessment of the quality control activities and determines the validity of the data or procedures for determining quality.*

- The quality assurance consists of set of reporting and auditing functions.
- These functions are useful for assessing and controlling the effectiveness and completeness of quality control activities.
- The goal of quality assurance is to ensure the management of data which is important for product quality.

Compare Quality Control with Quality assurance

Sr. No.	Quality Control	Quality Assurance
1.	This is an activity with the primary goal as to prevent the defects.	This is an activity with the primary goal as to identify and correct the defects.
2.	This process is intended to provide the assurance that the quality request will be achieved.	This process is intended to focus on quality being requested.
3.	This method is to manage the quality verification .	This method is for quality validation .
4.	It does not involve executing of the program.	During this method the program is executed .
5.	It is a preventive technique.	It is a corrective technique.
6.	It is a proactive measure	It is a reactive measure .
7.	It involves the full software development life cycle .	It involves testing phase of software development life cycle.
8.	It's main goal is to prevent defects in the system. It is less time consuming activity.	It's main goal is to correct the defects in the system. Hence it is more time consuming activity.

7.1.4 Cost of Quality

The cost of quality can be defined as the total cost required to obtain the quality in the product and to conduct the quality related activities.

The cost of quality has various components such as

1. Prevention cost - This is the cost of quality required for conducting quality planning, formal technical reviews, test equipments and training.
2. Appraisal cost - This is the cost of quality required for gaining the insight into the product. It includes the cost required for in-process and inter process inspection, maintenance and testing.
3. Failure cost - Failure cost means the cost required to remove the defects in the software product before delivering it to customer. There are two types of failure costs.
 - a. *Internal failure cost* - Internal failure is nothing but the cost of defects occurred in the product before delivering it to customer e.g. repair in networking, repairing of communication network.
 - b. *External failure cost* - External failure is the cost of defects occurred in the product after delivering it to the customer e.g. product repair/replace, complaint processing, warranty work.

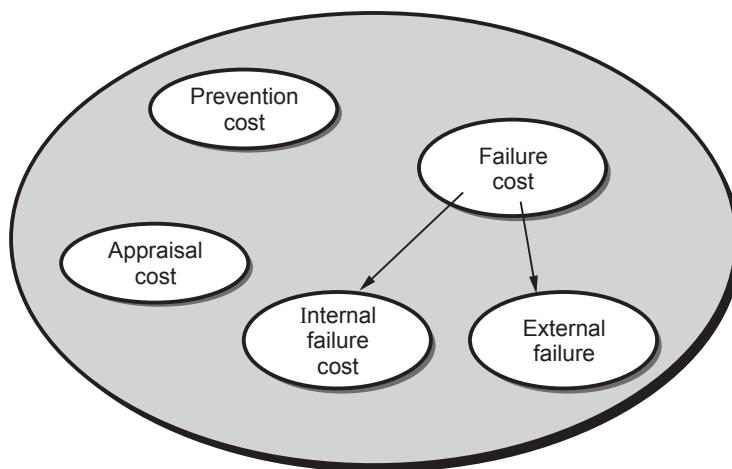


Fig. 7.1.2 Cost of quality

Review Question

1. Compare **quality control** with **quality assurance**.

GTU : Summer-2019, Marks 3

7.2 Software Quality Assurance (SQA)

GTU : Winter-2011, 2013, 2018, Summer-2012, 2014, 2016, 2018, Marks 7

We have already discussed the definition of software quality in section 7.1.1 which says that quality is the conformance to functional and non functional requirements of the product.

There are three main reasons for *why software quality gets failed ?*

1. Software requirements must be well understood before the software development process begins.
2. Similar to explicit requirements it is also essential to understand the implicit requirements of the software. If the software confirms the explicit requirements but not satisfying the implicit requirements then surely quality of software being developed is poor.
3. The set of development criteria has to be decided in order to specify the standards of the product. This will ultimately help the software engineer during development. If such a criteria is not been fixed then definitely the software product will lack the quality. Software quality assurance is the process in which conformance to the requirements of the product is made.

7.2.1 Software Quality Assurance (SQA) Activities

Let us list out the SQA activities conducted by SQA group.

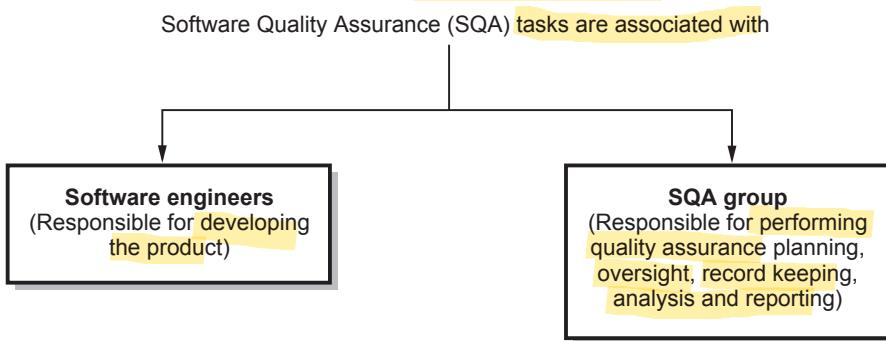


Fig. 7.2.1

1. Create a SQA plan.

A SQA plan is developed while planning the project. Quality assurance activities are conducted that are indicated in this plan. This plan basically

- Identifies evaluations to be performed.
- Audits and reviews to be performed, standards that should be adopted for the project.
- Procedures for error reporting and tracking.

- It also specifies documents to be produced by SQA group.
- Amount of feedback provided to the software project team.

2. Participates in description of software process.

The process selected by the software team is reviewed by the SQA group. This review is for

- Process description to ensure that it follows the organizational policy.
- Internal software standards.
- Some standards that are adopted by the organization.

3. Reviews software engineering activities.

The SQA group identifies and documents the processes. The group also verifies the correctness of software process.

4. Authenticate designated software work products.

The SQA group performs following tasks -

- Reviews selected work product
- Identifies the process
- Documents them
- Tracks deviations
- Verifies the correctness made in the processes
- Regular reporting of results of its work to the project manager.

5. Ensure the deviations in software work. Document work products

The deviations in software work are identified from project plan. These processes are identified and handled according to documented procedure.

6. Identify any noncompliance and reports to senior management.

Non compliance items are identified and pursued until they get resolved. The periodic reporting about it is done to project manager.

Review Questions

1. List software related activities.

GTU : Winter-2011, Marks 7

2. Define quality for software. List and explain SQA activities.

GTU : Summer-2012, 2018, Winter-2018, Marks 7

3. Explain importance of SQA.

GTU : Winter-2013, Marks 7

4. Explain software quality assurance techniques.

GTU : Summer-2014, Marks 7

5. What is the importance of SQA ? Explain the SQA activities.

GTU : Summer-2016, Marks 7

7.3 Software Reviews

GTU : Winter-2018, 2019, Summer-2019, Marks 3

Software reviews are filter to software engineering process. Such reviews are applied at various points during software development life cycle. The objective of software reviews is to uncover errors and defects that can be removed. The software reviews are conducted for **following reasons** -

Reasons for Reviews

- Point out needed improvements in the product of a single person or team.
- Confirm those parts of the product in which improvement is not desired.
- Achieve technical work of more uniform quality than can be achieved without reviews, in order to make technical work more manageable.

There are three different ways by which software review can be conducted.

1. **Informal meeting** an informal meeting can be conducted outside the working environment and an informal discussion of technical issues can be held.
2. **Formal presentations** can be conducted for customer, management and technical staff.
3. **Formal Technical Reviews (FTR)** sometimes called as **walkthrough** or an inspection is the most effective way of software review. This helps a lot for uncovering the software errors and to improve the quality of software.

Benefit of formal technical reviews or walkthrough is that errors can be discovered in early stage before they become defects in the next release of software.

7.3.1 Formal Technical Reviews (FTR)

Formal Technical Review is a **software quality assurance activity** performed by **software engineer**.

Objectives of FTR

1. FTR is useful to **uncover errors** in **logic**, **function** and **implementation** for any **representation** of the software.
2. The **purpose** of FTR is to **ensure** that software **meets specified requirements**.
3. It also ensures that the **software is represented** according to **predefined standards**.
4. It helps to **review** the **uniformity** in software development process.
5. It makes the project more **manageable**.

Besides the above mentioned objectives, the purpose of **FTR** is to **enable** junior engineers to **observe** the **analysis, design, coding and testing approaches** more closely.

Each FTR is conducted as a **meeting** and is **considered successful** only if it is properly **planned, controlled and attended**.

7.3.1.1 The Review Meeting

- Every review meeting should be conducted by considering the following constraints -
 - **Involvement of people** - Between 3 and 5 people should be involved in the review.
 - **Advance preparation** - Advance preparation should occur but it should be very short i.e. at the most 2 hours of work for each person can be spent in this preparation.
 - **Short duration** - The duration of the review meeting should be less than 2 hours.
- Rather than attempting to review the entire design, **walkthroughs** are conducted for modules or for small groups of modules.
- The **focus** of the FTR is on a **work product** (a software component to be reviewed).
- The review meeting is attended by the review leader, all reviewers and the producer.
- The *review leader* is responsible for evaluating the product for its readiness. The copies of product material is then distributed to reviewers.
- The *producer* organizes a “walkthrough” the product, explaining the material, while the *reviewers* raise issues based on their advance preparation.
- One of the reviewers becomes *recorder* who records all the important issues raised during the review. When errors are discovered, the recorder notes each.
- At the end of the review, the attendees decide whether to accept the product or not, with or without modifications.

7.3.1.2 Review Reporting and Record Keeping

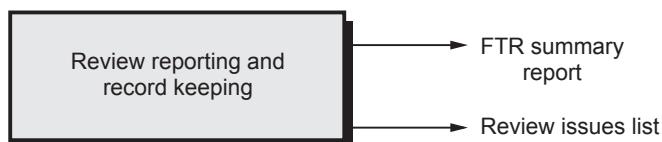


Fig. 7.3.1

During the FTR, the reviewer actively records all issues that have been raised. At the end of meeting these all raised issues are consolidated and *review issues list* is prepared. Finally, a *formal technical review summary report* is produced.

What is a purpose of producing review issues list ?

1. It helps in identifying problematic areas within the product.
2. From this issue list, a check list can be prepared which guides the producer for making the corrections.

The review issue list is normally attached to formal technical summary report.

7.3.1.3 Review Guidelines

Guidelines for the conducting of formal technical reviews must be established in advance. This guideline must be distributed to all reviewers, agreed upon, and then followed. For example - Guideline for review may include following things.

1. Concentrate on work product only. That means review the product, not the producer.
2. Set an agenda of review and maintain it.
3. When certain issues are raised then debate or arguments should be limited. Reviews should not ultimately result in some hard-feelings.
4. Find out problem areas, but don't attempt to solve every problem noted.
5. Take written notes (it is for the recorder).
6. Limit the number of participants and insist upon advance preparation.
7. Develop a checklist for each product that is likely to be reviewed.
8. Allocate resources and time schedule for FTRs in order to maintain time schedule.
9. Conduct meaningful trainings for all reviewers in order to make the reviews effective.
10. Review earlier reviews which servers as the base for the current review being conducted.

Review Question

1. Explain formal technical review.

GTU : Winter-2018, 2019, Summer-2019, Marks 3

7.4 Software Reliability

Software reliability is defined as the probability of failure free operation of a computer program in a specified environment for a specified time.

The software reliability can be measured, directed and estimated.

7.4.1 Measure of Reliability and Availability

Normally there are two measures of software reliability.

1. **MTBF** : mean-time-between-failure is a simple measure of software reliability which can be calculated as

$$\text{MTBF} = \text{MTTF} + \text{MTTR}$$

where MTTF means mean-time-to-failure

and MTTR stands for mean-time-to-repair.

Many software researchers feel that MTBF is more useful measure of software reliability than defects/KLOC or defects/FP.

2. **Availability** : It's another measure of software reliability software availability is defined as the probability that the program is working according to the requirements at a given points in time. It is measured as

$$\text{Availability} = (\text{MTTF}/(\text{MTTF}+\text{MTTR})) * 100 \%$$

MTBF is equally sensitive to MTTF and MTTR but *availability* is more sensitive to MTTR.

7.4.2 Software Safety

Software safety is a quality assurance activity in which **potential hazards** are identified and assessed. These hazards may bring the total failure of the system. If such hazards are identified and specified in early stage of software development then such hazards can be eliminated or controlled in order to make the software safe. Modeling and analysis process is conducted as a part of software safety.

For example : In a computer based automobile system software hazards are

1. Uncontrolled acceleration that cannot be stopped.
2. Does not respond to slow the system when breaks are applied.
3. Slowly gains the speed.

How to handle the system level hazards?

Following are the steps that can be applied to preserve the software safety.

Step 1 : The hazards are identified.

Step 2 : Analysis techniques are used to assign severity of these hazards. The probability of occurrence of such hazards is also analyzed with the help of analysis

techniques. The commonly used analysis techniques are fault-tree analysis, real-time logic and Petri-net models. These techniques basically predict the chain of events that can cause hazards.

Step 3 : Once hazards are identified, safety related requirements can be specified for the software. This specification basically includes list of undesirable events and desired system response. The template for safety related requirement specification is given below -

Safety related requirement specification

Identified hazards		
Name of the event that can cause hazard	Severity of hazard	Probability of occurrence
1.		
2.		
3.		
...		
1.		
2.		
3.		
4.		
....		
Undesired event		Desired system response
1.		
2.		
3.		
...		

Step 4 : Finally the role of software in managing undesirable event is specified.

What is the difference between software reliability and software safety?

- Software reliability and software safety are closely related to each other. However, the difference between them lies in degree and not the type.
- Software reliability uses statistical analysis made to determine the occurrence of software failure. These failures will cause simply dissatisfaction of customer

requirements. But the software safety examines the ways in which failure results in conditions that can lead to hazards.

- Software reliability does not detect the failures in depth. But the software safety detects the failures in context of an entire computer based system.

Review Question

1. Write a note on - Software Reliability.

7.5 Quality Standards

GTU : Summer-2014, Winter-2017, 2018, Marks 7

7.5.1 ISO 9000

In order to bring quality in the product and service, many organizations are adopting the quality assurance system. The quality assurance systems are the organizational structures that are used to bring quality in responsibilities, procedures, processes and resources.

ISO 9000 is a family of quality assurance system. It can be applied to all types of organizations. It doesn't matter what size they are or what they do. It can help both product and service oriented organizations to achieve standards of quality. ISO 9000 is maintained by ISO, the International Organization for Standardization and is administered by accreditation and certification bodies. In ISO 9000, company's quality system and operations are scrutinized by third-party auditors for a compliance to the standard and effective operation. This process is called registration to ISO 9000. On successful registration, the company gets a certification from accreditation bodies of ISO. Such a company is then called "ISO certified company".

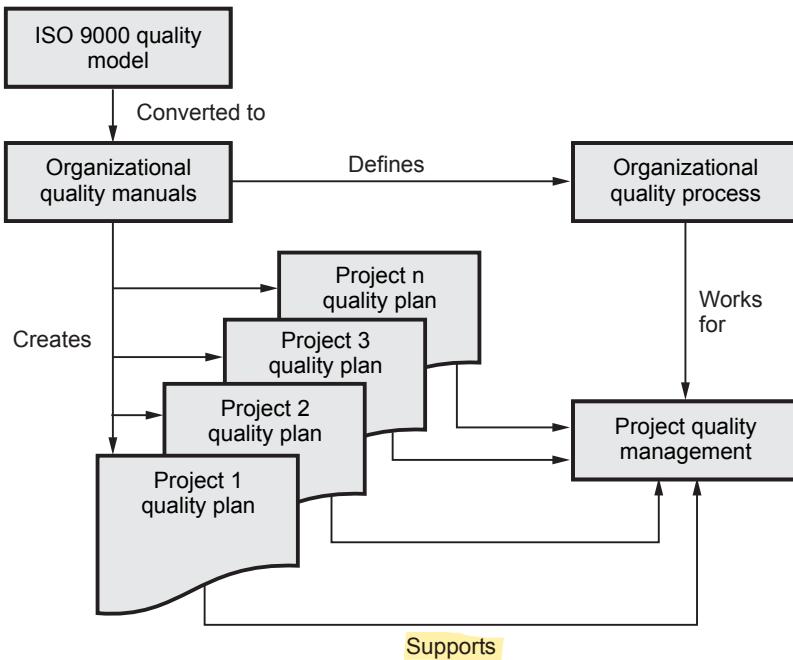
ISO 9001:2000 is a quality assurance standard which is applied to software engineering systems. It focuses on process flows, customer satisfaction, and the continual improvement of quality management systems. ISO 9001:2000 specifies requirements for a quality system that can be applied to any size or type of organization. The guideline steps for ISO 9001:2000 are

- Establish quality management system - Identify and manage the processes in the quality management system.
- Document the quality management system
- Support the quality
- Satisfy the customers
- Establish quality policy
- Conduct quality planning
- Control quality systems

- Perform management reviews
- Provide quality resources
- Provide quality personnel
- Provide quality infrastructure
- Provide quality environment
- Control realization planning
- Control customer processes
- Control product development
- Control purchasing functions
- Control operational activities
- Control monitoring devices
- Control non confirming products
- Analyze quality information
- Make quality improvement

The ISO 9000 helps in creating organisational quality manuals. These quality manuals identify the organisational quality processes. Using these quality manuals, the project quality plan can be prepared for every individual project. Thus project quality management can be done.

This is illustrated by following figure -



7.5.2 CMM capability maturity model

- 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.

7.5.3 Six Sigma

Six sigma is widely used statistical software quality assurance strategy. It is a business driven approach to process improvement, reduced costs and increased profit. The word "six sigma" is derived from six standard deviations - 3.4 defects per million occurrences. Six Sigma originated at Motorola in the early 1980s.

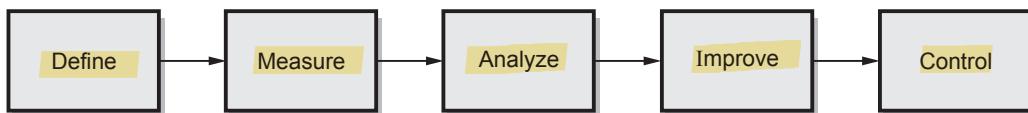


Fig. 7.5.1 Six sigma framework

There are three core steps in six sigma method -

Define - The customer requirements, project goals and deliverables are defined by communicating the customers.

Measure - The existing process and its output is measured in order to determine current quality performance.

Analyze - In this phase defect metrics are analyzed in order to determine the few causes.

If an improvement is needed to an existing software then there are additional two methods in six sigma -

Improve - By eliminating the root causes of defects the process can be improved.

Control - The process can be controlled in such a way that the causes of defects can not be reintroduced.

These steps can sometimes be referred as DMAIC.

For a newly developing software, some organizations are suggesting following two alternating steps -

Design - In this step avoid root causes of defects and meet the customer requirements.

Verify - To verify the process, avoid defects and meet customer requirements.

These steps can sometimes be referred as DMADV.

Review Questions

1. Explain six sigma method.
2. Write a note on ISO 9000.
3. Explain different quality standards.
4. List quality standards. Explain any one.

GTU : Summer-2014, Marks 7

GTU : Winter-2017, 2018, Marks 3

7.6 SQA Plan

The SQA plan is a document created for summarizing all the SQA activities conducted for the software project. The SQA plan specifies the goal and tasks that can be performed in order to conduct all the SQA activities. Such plan should be developed by SQA group. The standard for this plan is published by IEEE standard. The template for this plan is as given below –

SQA Plan

1. Purpose and scope of the plan
2. Description of work product
3. Applicable standards
4. SQA activities
5. Tools and methods/standards used

- 6. SCM procedures for managing change
- 7. Methods for maintaining SQA related records
- 8. Organizational roles and responsibilities

The SQA plan is a document aimed to give confidence to developers and customers that the specified requirements will be met and final product will be a quality product.

Review Question

- 1. *Give the template for SQA Plan.*



8

Software Maintenance and Configuration Management

Syllabus

Types of software maintenance, Re-engineering, Reverse engineering, Forward engineering, The SCM process, Identification of objects in the software configuration, Version control and change control.

Contents

8.1 Software Maintenance	Summer-2018,	Marks 7
8.2 Re-Engineering	Summer-2016,	
	Winter-2017, 2018,	Marks 7
8.3 Reverse Engineering	Summer-2013, 2018, 2019,	
	Winter-2017, 2018,	Marks 7
8.4 Forward Engineering	Summer-2019, Winter-2019, Marks 4	
8.5 Introduction to Software Configuration Management (SCM)		
	Summer-2016, Winter-2018, Marks 7	
8.6 Software Configuration Items	Winter-2011,	Marks 7
8.7 SCM Process	Summer-2013, 2018, 2019,	
	Winter-2019,	Marks 7

8.1 Software Maintenance

GTU : Summer-2018, Marks 7

- 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.

8.1.1 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.
- Not decrease the performance of the software**

8.1.2 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.

Review Question

1. Explain software maintenance.

GTU : Summer-2018, Marks 7

8.2 Re-Engineering

GTU : Summer-2016, Winter-2017, 2018, Marks 7

Software re-engineering means re-structuring or re-writing part or all of the software engineering system.

The software re-engineering is needed for the applications which require frequent maintenance.

Advantages of software re-engineering

1. **Reduced risk** - The re-engineering allows the developer to eliminate certain constraints on the system. This helps in reducing the risks of failures.
2. **Reduced cost** - The cost of re-engineering is often significantly less than the costs of developing new software.

Re-engineering process activities

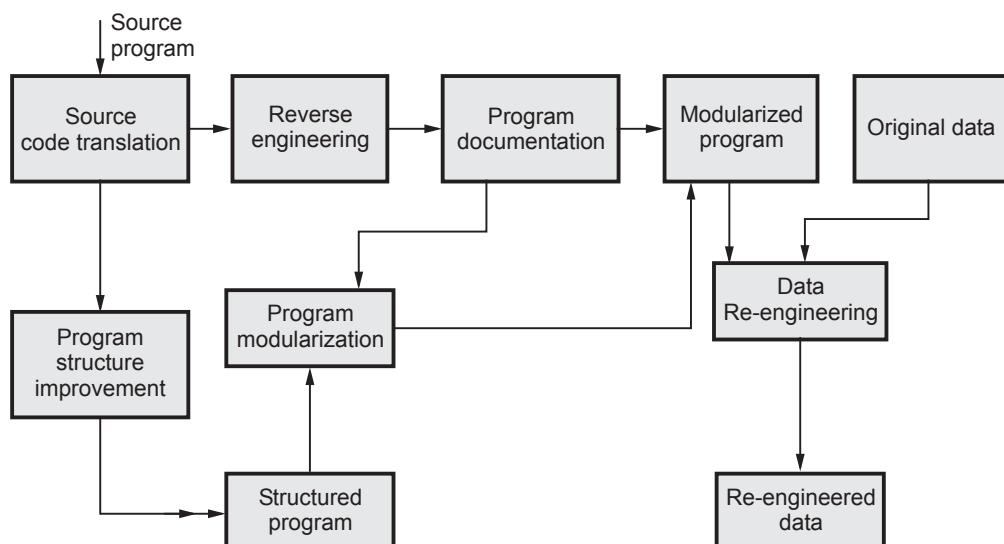


Fig. 8.2.1 Re-engineering process activities

- Source code translation : In this phase the code is converted to a new language.
- Reverse engineering : Under this activity the program is analysed and understood thoroughly.
- Program structure improvement : Restructure automatically for understandability.
- Program modularization : The program structure is reorganized.
- Data re-engineering : Finally clean-up and restructure system data.

Review Questions

1. Explain the software re-engineering activities.
2. Explain software re-engineering process model.
3. Write short notes on re-engineering.

GTU : Summer-2016, Marks 7

GTU : Winter-2017, 2018, Marks 3

8.3 Reverse Engineering

GTU : Summer-2013, 2018, 2019, Winter-2017, 2018, Marks 7

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.

detailing the abstract level

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.

8.3.1 Reverse Engineering Process

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. 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.

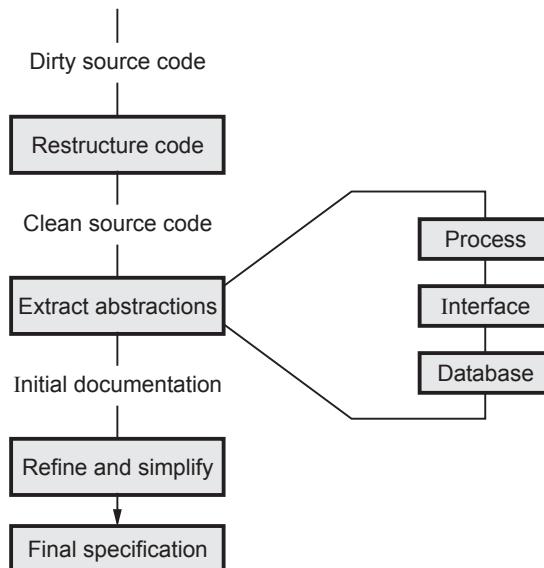


Fig. 8.3.1 Reverse engineering

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.

Review Questions

1. Write short note on : Reverse engineering

GTU : Summer-2013, Marks 7

2. Write short notes on reverse engineering.

GTU : Winter-2017, 2018, Summer-2019, Marks 4

3. Explain the following term in brief - 1) Re-Engineering, 2) Reverse Engineering.

GTU : Summer-2018, Marks 4

8.4 Forward Engineering

GTU : Summer-2019, Winter-2019, Marks 4

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.

8.4.1 Forward Engineering for Client Server Architectures

- During re-engineering process, many mainframe applications are modified to accommodate the client server architecture.
- Following are the features of this re-engineering process -
 1. The application functionality is transferred to each client computer from centralized resources.
 2. The new GUI are developed for each client workstation.

3. The database functionalities are handed over to servers.
 4. The responsibility of some specialized functionality can also be handled by Servers.
 5. New communication techniques are incorporated in the architecture.
 6. New security mechanisms are also established both at client and server side.
- The forward engineering for mainframe to client server architecture requires both business and software re-engineering activities.
 - The database transactions and queries are handled by server applications. At the same time these transactions must be controlled within the context of business rules.
 - That means, make sure that these transactions are executed in consistent manner such that all updates are performed by authorized users.
 - During forward engineering, there exists a business layer present at both client and server. The task of this layer is to control and coordinate the tasks of transactions and query handling. The communication among desktop applications is controlled by business rules layer.
 - There is client applications layer which implements the business functions that are required by specific group of end users.

8.4.2 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 Questions

1. Difference between reverse engineering and forward engineering.

GTU : Summer-2019, Marks 4

2. Write short note on - Forward Engineering.

GTU : Winter-2019, Marks 4

8.5 Introduction to Software Configuration Management (SCM)

GTU : Summer-2016, Winter-2018, Marks 7

Definition : Software configuration management is a set of activities carried out for identifying, organizing and controlling changes throughout the lifecycle of computer software.

During the development of software change must be managed and controlled in order to improve quality and reduce error.

8.5.1 Need for SCM

The software configuration management is concerned with managing the changes in the evolving software. If the changes are not controlled at all then this stream of uncontrolled change can cause the well-running software project into chaos. Hence it is essential to perform following activities -

- i) Identify these changes
- ii) Control the changes
- iii) Ensure that the changes are properly implemented and
- iv) Report these changes to others.

The software configuration management may be seen as part of quality management process.

Review Questions

1. Explain software configuration management.

GTU : Summer-2016, Marks 7

2. Discuss software configuration management in detail.

GTU : Winter-2018, Marks 7

8.6 Software Configuration Items Not did

GTU : Winter-2011, Marks 7

A Software Configuration Item (SCI) is information that is created as part of the software engineering process.

Examples of Software Configuration Items are

- Computer programs
 - Source programs
 - Executable programs

- Documents describing the programs
 - Technical manual
 - Users manual
- Data
 - Program components or functions
 - External data
 - File structure

For each type of item, there may be a large number of different individual items produced. For instance there may be many documents for a software specification such as project plan, quality plan, test plan, design documents, programs, test reports, review reports.

These SCI or items will be produced during the project, stored, retrieved, changed, stored again, and so on.

Each configuration item must have a unique name, and a description or specification which distinguishes it from other items of the same type.

Review Question

1. What do you mean by software configuration ? What is meant by software configuration management ?

GTU : Winter-2011, Marks 7

8.7 SCM Process

GTU : Summer-2013, 2018, 2019, Winter-2019, Marks 7

The primary objectives of Software Configuration Management process (SCM) are -

1. **Configuration Identification** : Identify the items that define the software configuration.
2. **Change Control** : Manage changes to one or more items.
3. **Version Control** : Facilitate to create different versions of the application.
4. **Configuration Authentication** : To ensure that the quality of the software is maintained as the configuration evolves over the time.

The SCM process must be developed in such a way that the software team must answer the following set of questions -

1. How does the software team identify the software configuration items ?
2. How does the software team control the changes in the software before and after delivering it to the customer ?

3. How does the software team manage the versions of the programs in the software package ?
4. How does team get ensured that the changes are made properly ?
5. Who is responsible for approving the changes in the software ?

The answers to these questions lead the definition of five tasks of SCM and those are - Identification, change control, version control and configuration audit and status reporting.

8.7.1 Identification of Objects in Software Configuration

- The software configuration items must be separately named and identified as object.
- These objects must be arranged using object oriented approach.
- There are two categories of objects - basic objects and aggregate objects.
- The basic object is unit of information created during requirements analysis, design, coding or testing. For example basic object can be part of source code.
- Aggregate object is a collection of basic objects and other aggregate objects. For example SRS or data model can be aggregate object.
- Each object can be uniquely identified because it has got -
 1. Name : The name of the object is nothing but the collection of characters(string) or some text. It is unique.
 2. Description : For describing the object, the object description can be given. This description contains document, program or some other description such as project identifier or version information.
 3. List of resources : The resources are the entities that are used for accessing, referencing and processing of objects. The data types and functionalities can serve as a resource.
 4. Realization or identification : It is pointer to object.
- The configuration object identification can also consider relationships that exist between the named objects.
- If a change is made to one configuration object it is possible to determine which other configuration objects in the repository are affected by the change.
- Basically object evolve throughout the software process. During the process of object identification the evolution of objects along with its process must be identified.
- Major modifications in the object must be noted.

8.7.2 Change Control

Changes in any software projects are vital. Sometimes, introducing small changes in the system may lead to big problems in product. Similarly, introducing some changes may enhance the capabilities of the system. According to James Bach too little changes may create some problems and too big changes may create another problems.

For a large software engineering project, uncontrolled change creates lot of chaos. For managing such changes, human procedures or automated tools can be used.

The change control process is shown by following Fig. 8.7.1. (See Fig. 8.7.1 on next page.)

Step 1 : First of all there arises a need for the change.

Step 2 : The change request is then submitted by the user

Step 3 : Developers evaluate this request to assess technical merits, potential side effects, and overall impact on system functions and cost of the project.

Step 4 : A change report is then generated and presented to the Change Control Authority (**CCA**).

Step 5 : The change control authority is a person or a group of people who makes a final decision on status or priority of the change.

Step 6 : An Engineering Change Order (**ECO**) is generated when the change gets approved. In ECO the change is described, the restrictions and criteria for review and audit are mentioned.

Step 7 : The object that needs to be changed is checked out of the project database.

Step 8 : The changes are then made on the corresponding object and appropriate SQA activities are then applied.

Step 9 : The changed object is then checked in to the database and appropriate version control is made to create new version.

The checked in and checked out mechanisms require two important elements -

- Access control
- Synchronization control

The **access control** mechanism gives the authority to the software engineer to access and modify the specific configuring object. The **synchronization control** mechanism allows to make parallel changes or the changes made by two different people without overwriting each other's work.

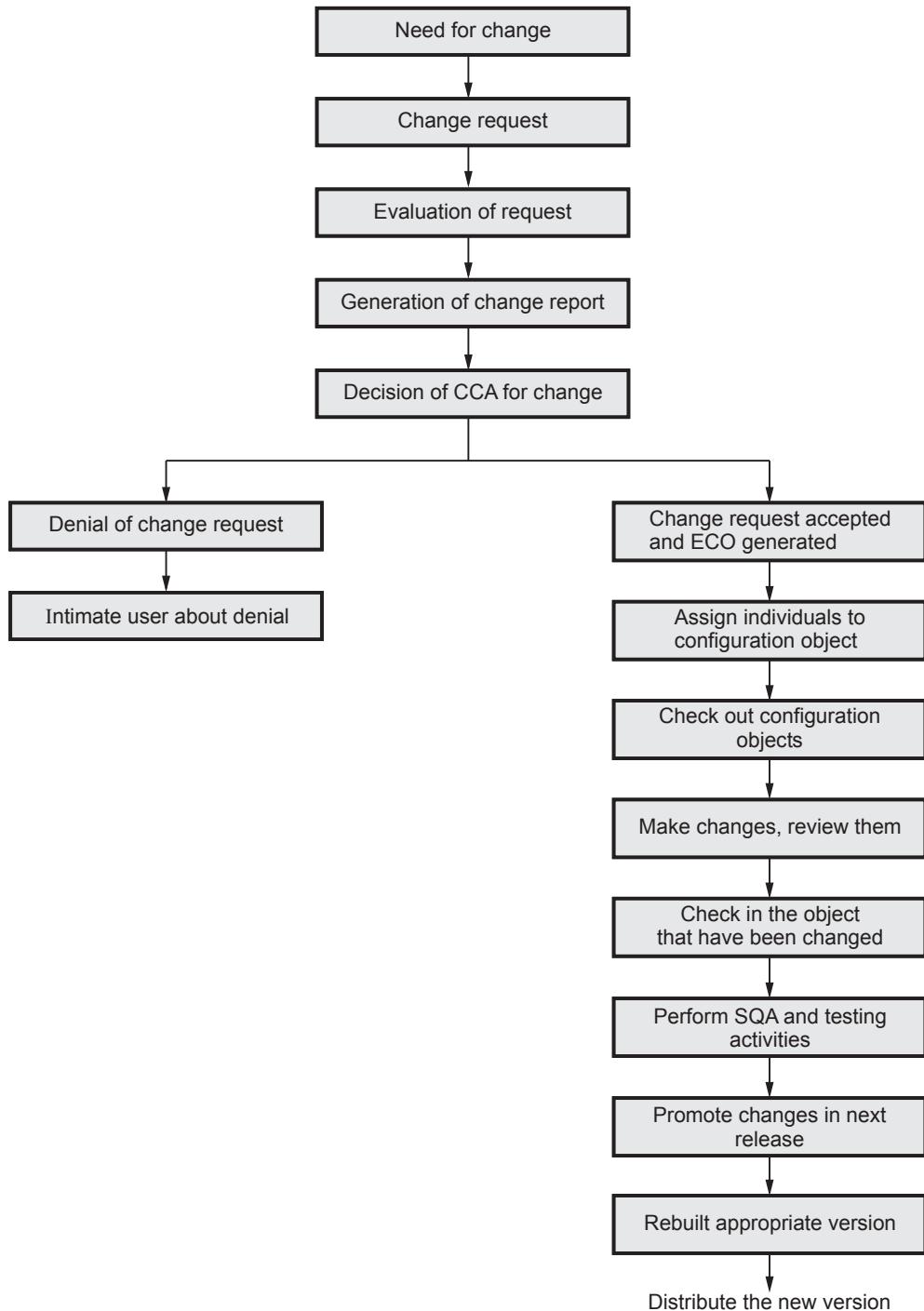


Fig. 8.7.1 Change control process

8.7.3 Version Control

Version is an instance of a system which is functionally distinct in some way from other system instances.

Version control works to help manage different versions of configuration items during the development process.

The configuration management allows a user to specify the alternative configurations of the software system by selecting appropriate version.

Certain attributes are associated with each software version. These attributes are useful in identifying the version. For example : The attribute can be 'date', 'creator', 'customer', 'status'.

In practice the version needs an associated name for easy reference.

Different versions of a system can be shown by an evolution graph as

Each version of software system is a collection of software configuration items.

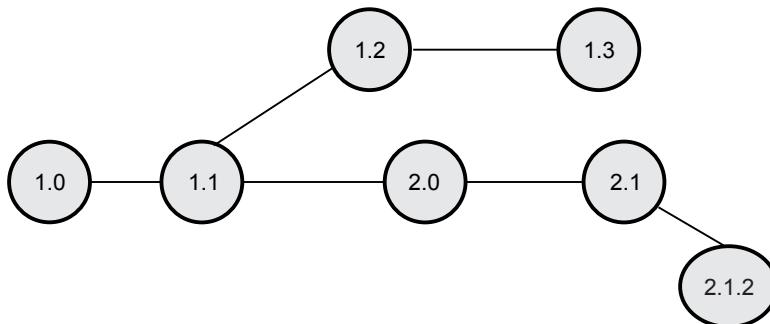


Fig. 8.7.2 Version numbering in evolution graph

8.7.4 Configuration Audit

- In order to ensure that the change has been properly implemented or not two activities are carried out.
 1. Formal Technical Review (FTR)
 2. Software Configuration Audit
- In Formal Technical Review, the correctness of configuration object is identified and corrected. It is conducted by technical reviewer.
- The software configuration audit assess the configuration object for the characteristics that are not reviewed in formal technical review. It is conducted by software quality assurance group.
- Following are some primary questions that are asked during configuration audit -
 1. Whether FTR is conducted to assess the technical correctness ?
 2. Whether or not the change specified by ECO has been made ?

3. If additional changes need to be made or not ?
 4. Whether the software engineering standards are properly followed ?
 5. Do the attributes of configuration objects reflect the change ?
 6. Whether all the SCI are updated properly ?
 7. Whether the SCM process(object identification, change and version control, configuration audit and status reporting) are properly followed ?
- The above questions can be asked as a part of formal technical review.

8.7.5 Status Reporting

The status reporting focuses on communication of changes to all people in an organization that involve with changes.

During status reporting following type of questions were asked.

1. **What happened ?** : What are the changes that are required ?
2. **Who did it ?** : Who will be handling these changes ?
3. **When did it happen ?** : The time at which these changes are arised.
4. **What else will be affected ?** : The objects or part of the software that might be reflected due to these changes.

Review Questions

1. Define software configuration management. Explain change control management and version control management.
2. Explain software configuration management and change control management in detail.
3. Explain "How to manage the different versions that get created and how to maintain the quality of code under changing conditions ?". GTU : Summer-2013, Marks 7
4. Explain version and change control management. GTU : Summer-2018, 2019, Winter-2019, Marks 4
5. Explain SCM process in details. GTU : Summer-2019, Marks 7



9

DevOps

Syllabus

Product lifetime : Independent product Vs. continues, Improvement, Software as a service, SaaS architecture.

Contents

- 9.1 Overview
- 9.2 Problem Case Definition
- 9.3 Benefits of Fixing Application Development Challenges
- 9.4 DevOps Adoption Approach through Assessment
- 9.5 Solution Dimensions
- 9.6 What is DevOps?
- 9.7 DevOps Importance and Benefits
- 9.8 DevOps Principles and Practices
- 9.9 The 7 C's of DevOps Lifecycle for Business Agility
- 9.10 DevOps and Continuous Testing
- 9.11 How to Choose Right DevOps Tools
- 9.12 Challenges with DevOps Implementation
- 9.13 Must Do Things for DevOps
- 9.14 Mapping My App to DevOps
- 9.15 Assessment, Definition, Implementation, Measure and Feedback

9.1 Overview

- DevOps encourages the development, IT operations, quality engineering and security activities to be performed in coordination and collaboration to produce better, more reliable products. By adopting a DevOps culture along with DevOps practices and tools, teams gain the ability to better respond to customer needs, increase confidence in the applications they build and achieve business goals faster.
- DevOps is a culture which promotes collaboration between Development and Operations Team to deploy code to production faster in an automated and repeatable way.
- Teams that adopt DevOps culture, practices and tools become high-performing, building better products faster for greater customer satisfaction.

9.1.1 Difference between DevOps and Agile

Sr. No.	Agile	Devops
1.	The idea in agile is to develop software in small iterations and be thus able to adapt to the changing customer needs.	Devops is to deliver technology to business units in a timely fashion and ensure the technology runs without interruption or disruption.
2.	It adopts rapid development approach	This is not a rapid development approach.
3.	The focus of agile development is merely on software development and release.	The focus of Devops is not only on software development, its release but on its safest deployment in working environment.
4.	In agile development, every team member has a skill of design, development and coding. Any available team member should be able to do what's required for progress.	Devops, on the other hand, assumes there will be development teams and operational teams, the two will be separate. These teams can communicate between themselves on frequent and regular basis.
5.	Communication in agile development is informal and in the form of daily meetings.	In Devops communication, specifications, documents are involved and it is formal. It does not occur on daily basis.
6.	The team is small in nature.	Large team size and multiple teams are required in Devops.
7.	Agile is about software development.	Devops is about software development and management.
8.	Documentation is not much important in agile development.	Documentation is very much important in devops.

9.	Agile development teams, may choose to use certain automation tools. But there are no specific tools required for an agile team.	Devops absolutely depends on automated development to make everything happen smoothly and reliably. Certain tools are an integral part of devops.
10.	Agile is less flexible.	Devops is more flexible.
11.	Agile has limited scope.	Devops has broader scope.

Review Question

1. Differentiate between DevOps and agile Development.

9.2 Problem Case Definition

In this section we will consider some real world problem for application development. Suppose we want to design online share trading application. This application can be developed using following stages -

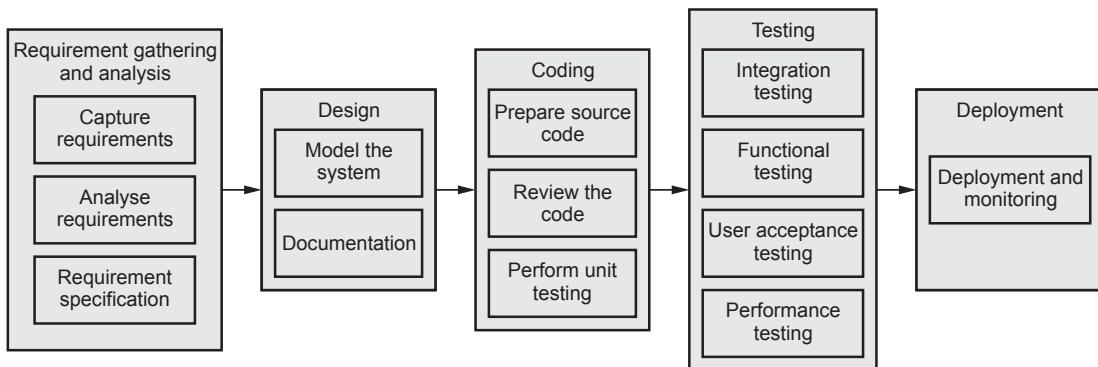


Fig. 9.2.1 Application development Life Cycle

Consider that the agile process is adopted as a delivery process by the development team.

1. **Requirement analysis :** During this phase, the requirements of the system are identified. Both the functional and non functional requirements are gathered, analyzed and presented to the development team in specific format.
2. **Design :** Design is an activity to outline the processes to be conducted. The using diagrammatic representation, the activities are modeled.
3. **Coding :** Once the design of the system is done, the coding process starts. With the help of suitable programming language, the codes are written, and reviewed by the development team. Some testing is also performed with the coding. This type of testing is usually unit testing.

4. Testing : Before deploying the product in the working system, four types of testing are carried out -

- 1) Integration testing 2) Functional testing
- 3) User acceptance testing 4) Performance testing

Multiple teams perform testing. The operational team ensures that product is working accurately in working environment.

5. Deployment : This is a stage in which the product is packaged, deployed and monitored for correct functioning.

9.2.1 Challenges in Application Development

The challenges during application development life cycle are -

- 1) For execution of certain activities, during application development there exists total dependency on individuals in the team.
- 2) Quality is an important concern during development stages.
- 3) For testing the application, the provision of working environment must be made. It may cause delay in testing process.
- 4) Tracing the requirements during the development stage is a tedious job.
- 5) Testing can be costly, if repeated testing is required.

9.3 Benefits of Fixing Application Development Challenges

Following are the key benefits of fixing application development challenges -

- 1) **Time :** User requirements are changing many times. The ability of IT system to align with the business creates huge impact.
- 2) **Cost :** Cost of the software is an important factor. The software cost can be reduced by reducing waste in the development process.
- 3) **Quality :** By fixing the application development challenge the product quality can be improved. The delivery team should be able to quantify the product quality in early stage of development.

Review Question

1. *What are the challenges in application development ?*

9.4 DevOps Adoption Approach through Assessment

Definition of Assessment : The process of reviewing the application and related processes to identify the status of DevOps challenges is called assessment or maturity assessment.

There are five steps to be followed to identify and fix the application challenges. These steps are -

Step 1 : Assess to Identify Gaps : This step identifies the gap in process, technology, tools and automation in software development process.

Step 2 : Define Solution : In this step, the solution are created for the issues that are identified in the step 1. The solutions are prepared as per the DevOps strategy.

Step 3 : Implement Solution : The solution is implemented by taking maximum benefit.

Step 4 : Measure Benefit : The benefit is measured understand the effectiveness of the adopted strategy.

Step 5 : Feedback Benefits and Challenges : The feedback is important to refine the solution the solution with better quality and in few iterations.

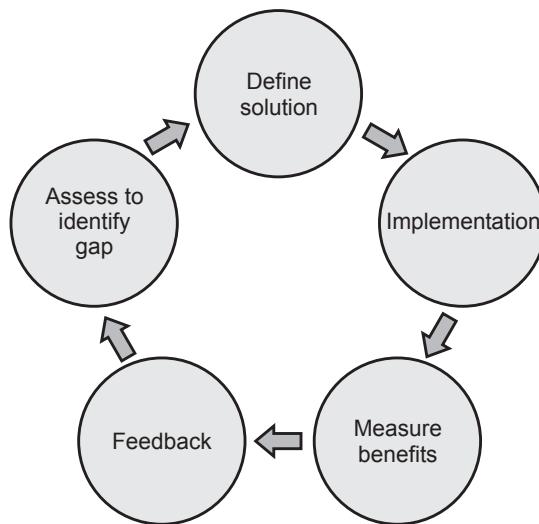


Fig. 9.4.1

9.5 Solution Dimensions

There are three key dimensions that can be used to solve issues in the application development in an integrated way.

1) Process : After requirements gathering the methodology of development is decided. This method can be waterfall model, spiral model, agile development process model or any other suitable process model. These process methods must be reviewed to fix the issues in application development.

2) Tools : The tools are used in application development process to automate the manual activities. The solution must focus on effectiveness of the tools and the manner in which the tool can be used. The solution must identify if additional toolset is required or not.

3) Architecture and technology : The solution must evaluate underlying architecture. It must be checked if parallel processing is possible or not. It must also be evaluated if the automated processes can be executed on this architecture or not.

Review Question

1. *What is solution Dimension in solving the issues of typical application development process.*

9.6 What is DevOps?

Definition : Devops is a practice in which development and operation engineers participate together in entire lifecycle activities of system development from design, implementation to product support.

- The term Devops is derived from "Software DEvelopment" and "information technology OPerationS".
- Devops promotes a set of processes and methods from the three department **Development, IT operations and Quality assurance** that communicate and collaborate together for development of software system.

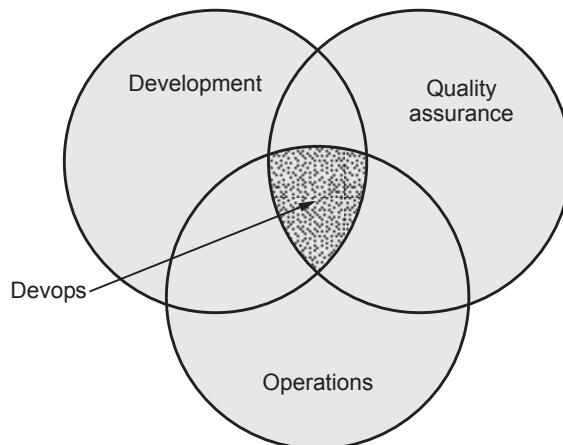


Fig. 9.6.1

Review Question

1. *What is DevOps ?*

9.7 DevOps Importance and Benefits

Importance

- Devops enhances the organization's performance, improves the productivity and efficiency of development and operations teams.

- Bringing the two teams together centralizes the responsibility on the entire team and not specific individuals working.
- Devops is more than just a tool or a process change. It inherently requires an organizational culture shift.
- This cultural change is especially difficult, because of the conflicting nature of departmental roles :
 1. Operations - seeks organizational stability;
 2. Developers - seek change;
 3. Testers - seek risk reduction.
- Adoption of Devops is driven by various factors. These factors are -
 1. Demand for an increased rate of production releases - from application and business unit stakeholders.
 2. Increased usage of data center automation and configuration management tools.
 3. Use of agile and other development processes and methods.
 4. Increased focus on test automation and continuous integration methods.
 5. Wide availability of virtualized and cloud infrastructure.

Benefits

Various benefits of Devops are -

- **Technical benefits**
 1. Continuous software delivery is possible
 2. There is less complexity to manage the project.
 3. The problems in the project gets resolved faster.
- **Cultural benefits**
 1. The productivity of teams get increased.
 2. There is higher employee engagement.
 3. There arise greater professional development opportunities.
- **Business benefits**
 1. The faster delivery of the product is possible.
 2. The operating environment becomes stable.
 3. The communication and collaboration gets improved among the teams members and customer.
 4. More time is available for innovation rather than fixing and maintaining.

Review Question

1. Explain the importance and benefits of DevOps.

9.8 DevOps Principles and Practices

DevOps principles and practices are enlisted as given below -

Principles

- 1) **Customer centric action** : Continuous feedback of end user or real customer should get reflected in all the development activities. The product and services must be developed in such a manner that the customer satisfaction is always protected.
- 2) **Focus on end result** : The developers and engineers must work by keeping the complete picture of the end product in mind.
- 3) **End-to-End responsibility** : All the services required to develop, maintain and monitor must be done by the same team. This helps in improving the quality of end product.
- 4) **Cross functional autonomous teams** : If same team works on all the phases such as development, maintenance and monitoring then it helps in improving the quality of end product. The members of team having multiple skill set on various technologies and use of variety of testing tools help to make the team self-sufficient.
- 5) **Continuous improvement** : The continuous improvement is normally done for optimizing the speed of processing, reduced cost and minimized waste. The objective of continuous improvement is to bring best quality product.
- 6) **Use of automated tools and techniques** : The use of automated tools help in improving the speed and reducing the development efforts. Hence automate the things wherever possible.

Practices

- 1) **Perspective consideration** : The difference between the developers and participants perspective must be considered.
- 2) **Flexibility** : The organization must have flexibility to switch between the delivery of the product using DevOps values and without DevOps values.
- 3) **Integrate changes** : It must be possible to accommodate the required changes in the system.
- 4) **Agility** : The process as well as tool selection needs to be agile according the project needs.
- 5) **Transparency** : There must be transparency about the goals, delivery plan and activities between developers and operational engineers.

9.9 The 7 C's of DevOps Lifecycle for Business Agility

The 7Cs of DevOps are as follows -

1) Continuous business planning : During this phase the planning for identifying skills, resources required and outcome is made.

2) Continuous development : In this phase, development sketch plan is prepared and programming techniques are identified. Before continuous integration, development teams would write a bunch of code for three to four months. Then those teams would merge their code in order to release it.

3) Continuous integration : Continuous integration is the practice of quickly integrating newly developed code with the main body of code that is to be released. Continuous integration saves a lot of time when the team is ready to release the code.

The continuous integration process from a DevOps perspective involves checking your code in, compiling it into usable (often binary executable) code and running some basic validation testing.

4) Continuous deployment : It is the practice of deploying all the way into production without any human intervention. Teams that utilize continuous delivery don't deploy untested code; instead, newly created code runs through automated testing before it gets pushed out to production. The code release typically only goes to a small percentage of users and there's an automated feedback loop that monitors quality and usage before the code is propagated further.

5) Continuous testing : Continuous testing (CT) is the process of executing automated test cases as part of the software delivery pipeline. Its goal is to obtain immediate and continuous feedback on the business risks associated with a software release candidate.

6) Continuous delivery and monitoring : With continuous delivery, every code change is built, tested, and then pushed to a non-production testing or staging environment. There can be multiple, parallel test stages before a production deployment.

Continuous monitoring is a process of monitoring the application continuously to check its service, performance and security. The continuous monitoring can be done using different tools. This monitoring process involves generation of reports.

7) Continuous feedback : This is a process which allows for an immediate response from your customers for your product and its features and helps you modify accordingly.

Review Question

1. Explain 7 C's of DevOps lifecycle for business agility.

9.10 DevOps and Continuous Testing

- Continuous testing is an important aspect in DevOps.
- **Definition of continuous testing :** Continuous testing is defined as software testing that involves the process of testing early, testing frequently and automate the test.
- Test automation is a mandatory step in continuous testing.
- Continuous testing is a vital aspect that bridges continuous integration and continuous delivery.

9.10.1 Continuous Testing Process in DevOps

- In DevOps processes the a software change is continuously moving from development to testing to deployment. That means the code is continuously developed, delivered, tested and then deployed.
- During continuous testing, there is an execution of various types of tests, continuously on the code base and on different environments that it gets deployed on to, as predefined and designed in the continuous delivery pipeline.

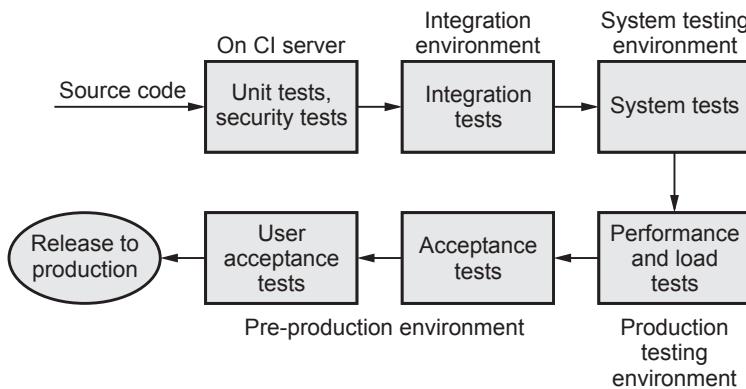


Fig. 9.10.1 Continuous testing architecture in DevOps

- The unit tests happen on the Continuous Integration(CI) server itself, which tests each unit of the system in isolation.
- Integration tests happen on Integration environment which basically verifies the components integrated together.
- System tests in the system testing environment where the BIG system with all the integrated components and interfaces are tested through system-level scenarios in a system testing environment and so on.
- And the depth of testing often progresses as the simulation of environment gets closer to production.

- Then goes to the 'Acceptance testing' which basically includes the automated site acceptance test cases and then finally on to the 'User Acceptance Testing' which could be a manual execution and includes end-user participation to carry out the tests and a this will be a kind of final sign off on the product or a feature,
- Here the continuous testing would be mainly running the automated test cases automatically with a trigger.

9.10.2 Advantages of Continuous Testing

- 1) Quality of the product gets improved.
- 2) Speed of execution gets increased by involvement of automated testing techniques
- 3) Faster feedback on code.
- 4) The continuous testing process boosts the confidence of the team and encourages them to improve continuously.

9.10.3 Challenges in Continuous Testing

1) Tool early stage testing : Early testing is normally done at requirements analysis phase. In this phase requirements can get changed. There may be lack of test case coverage or there can be multiple overlapping test cases. Poorly defined or incomplete requirements can not be tested completely in this phase.

2) Test data management : Another challenge in continuous delivery is locating proper test data. Hence proper test data management must be done consistently. Following are the difficulties in test data management -

- (i) extraction of test data spread across multiple databases
- (ii) limited access to production systems
- (iii) maintenance of multiple dataset versions for different tests
- (iv) creating test data without copying the production data

3) Availability of testing environment : Within continuous delivery, tests run in parallel in the development phase and integration phase. Because of that, the team needs to have many environments available for different purposes. The lack of test environments is one of the biggest challenges to achieve continuous testing.

4) Lack of availability of proper automated testing tool : Most of the legacy testing tools are unfit for continuous testing in DevOps, hence use of appropriate automated tools at every stage of development life cycle is a big challenge.

Review Question

1. Explain continuous integration process in DevOps.

9.11 How to Choose Right DevOps Tools

Following are the 7 steps to be followed to choose right DevOps tools -

Step 1 : Understand the collaboration and shared tools strategy

DevOps teams need to come up with a common tools strategy that lets them collaborate across development, testing, and deployment. The common strategy is based on -

- Processes
- Communications and collaboration planning
- Continuous development tools
- Continuous integration tools
- Continuous testing tools
- Continuous deployment tools

Step 2 : Use tools to capture all the requests

DevOps tooling should capture every request for new or changed software. DevOps provides the ability to automate the acceptance of change requests that come in either from the business or from other parts of the DevOps teams.

Step 3 : Use agile Kanban project management for automation

Kanban is a framework used to implement agile development that matches the amount of work in progress to the team's capacity. It gives teams more flexible planning options, faster output, clear focus, and transparency throughout the development cycle. Kanban tools provide the ability to see all the items in context with each other.

Step 4 : Use tools to log metrics on both manual and automated processes

Select tools that can help you understand the productivity of your DevOps processes, both automated and manual.

Step 5 : Implement test automation and test data provisioning tooling

With DevOps, testing must be continuous. There can be thousands of test cases that need to be executed during application development. Thus test automation and test data provision tools are essential.

Step 6 : Perform acceptance tests for each deployment tooling

For the tool set selected, those charged with DevOps testing processes should spend time defining the acceptance tests, and ensuring that the tests meet with the acceptance criteria selected. These tests may be changed at any time by development or operations.

Step 7 : Ensure continuous feedback between the teams

There must be feedback loops to automate communication between tests that spot issues, and tests that process needs to be supported by your chosen tool. Hence tools that monitor software in production, identify issues in software in an automated or manual way, link the issues to deployable components.

9.12 Challenges with DevOps Implementation

1) Cultural change : Any organization must promote the collaborative culture for effective implementation of DevOps. The leaders should bring transparency in the work process. There must be positive atmosphere in an organization.

2) Bringing silos and sectors together : There is a tendency of maintaining silos and sectors within the team. While developers are constantly writing pieces of code in order to build a system, testers perform a thorough analysis to ensure product stability before the final delivery to the customer. Due to this practice, there lies a big gap between teams. Both Dev and Ops work in silos, leading to a lack of transparency and poor teamwork.

3) Giving up legacy systems : Organizations must give up the old or outdated systems and must adopt modern and efficient systems. Handling new systems along with the old existing systems in the organization can be challenging many times.

4) Tool selection confusion : There are number of tools available in the market which tempt the DevOp developers to choose different tools. This leads to changing and updating the tools frequently. Changing and updating the tools becomes difficult with change in strategy. Hence instead of using ever changing tools and increasing the cost, the team should adopt well organized long term strategy to use specific tool.

5) Different metrics : During product development process each team measure their performance using different metrics. When the projects is to be implemented using DevOps technology, there must be a common metric to be used to measure the performance. Accepting a common measure of performance is sometimes challenging for different teams.

6) Resistance to change : The teams are normally unwilling to accept the changes. They are resistance to change their preferred working style. They do not open up the silos to other teams. This makes it difficult to other teams to work and may create an unhealthy environment.

7) Process challenges : DevOps strategy does not define specific rules to implement the process or to use the tools. The only restriction is to follow the project goal. Although this gives lot of flexibility and chances to use innovative methodologies , it may lead to challenging situations like confusion and disputes among the team members of some strategies.

9.13 Must Do Things for DevOps

Following is a list of things that are most required for DevOps to implement.

1) Find DevOps driver : A DevOps driver is a resource, process or condition that will help to stay focused in software development activities, security and data management.

2) Adopt with DevOps culture : Adopting DevOps culture means instead of having traditional software development approach, the team should work in collaboration with the other teams. It should have an involvement in all the activities from development to deployment of the product and chase for the quality in work.

3) Move in right direction : For moving the team in right direction coaching in both Agile and DevOps help the team members to embrace the changes in culture and practices. Team members should also be trained to use new tools and techniques.

4) Make the customer happy : By creating user friendly product and high quality service within the stipulated time- makes your customer happy.

5) Follow certain principles : Some of the commonly followed principles are -

(i) Automate repetitive tasks.

(ii) Keep it simple

(iii) Every must be responsible.

(iv) Get continuous feedback and work accordingly.

6) Ensure security : Adopting DevOps practices introduce complications for implementing standard security practices. Team must think about making the product and service more secure.

7) Make use of right tools : No tool is perfect. But the team should adopt well organized long term strategy to use specific tool.

8) Use key technologies : DevOps implementation require frequent deployment and continuous feedback. Due to which new features are getting added frequently. Hence make use of key technologies and practices instead of making use of traditional, monolithic development.

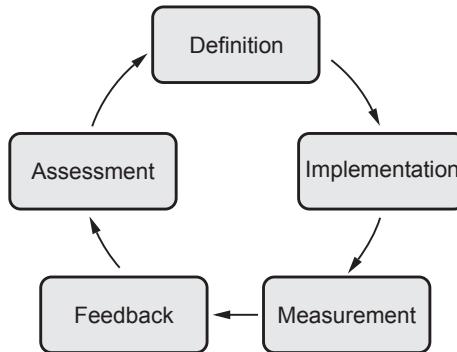
9.14 Mapping My App to DevOps

Mapping My Application to DevOps is called adoption of DevOps. The DevOps adoption is a continuous iterative process. It consists of following steps -

- 1) Assessment
- 2) Definition
- 3) Implementation

4) Measurement

5) Feedback

**Fig. 9.14.1**

DevOps strategy can be adopted to the entire end-to-end development process or it can be applied to particular life cycle stage (For instance - coding).

9.15 Assessment, Definition, Implementation, Measure and Feedback

1) Assessment :

- This is the first step of mapping an application to DevOps.
- This step is basically called as **DevOps maturity assessment**.
- There are a set of parameters to be measured at every stage of DevOps Maturity Model to confirm an organization's level of DevOps maturity.
- Assessment the Devops maturity is based on the various parameters such as process, people, tools, culture, measurement and reporting.
- The DevOps capability is measured in terms of Crawl, Walk, Run and Sprint or equivalent levels measuring maturity in increasing order.
- Method of assessment is arranging brain-storming sessions with different stakeholders. This is an iterative activity in which stakeholders put their views and ideas and the application is refined further.
- Typically set of predefined questionnaire is prepared for measuring the maturity of application in activity of development lifecycle.
- Finally a detailed report about DevOps maturity is prepared.

2) Definition :

- Definition is the second phase in which solution is defined. That means the actions are identified which help the application to reach to next level of maturity.
- The input to definition phase are - 1) Objectives of the project 2) findings from assessment phase and 3) Enterprise DevOps strategy.

- Enterprise DevOps strategy consists of set of guidelines for individual project about adopting DevOps strategy. For instance - the guideline may consist of list of approved tools for the project.
- Enterprise DevOps strategy must be flexible enough by accepting contributions from individual project, even if it is not implemented earlier in the organization.
- Hence Enterprise DevOps strategy must be evolving for betterment.
- At the end of definition phase, a blueprint of project of DevOps is prepared with action items that are obtained in assessment report.

3) Implementation :

- This is a phase in which the items which are identified during the definition phase are implemented.
- Due to adoption of DevOps strategy, use of automated tools saves lot of resources during each phase of development life cycle. This enhances the efficiency of application development process.
- Following Fig. 9.15.1 and 9.15.2 illustrates the difference between pre DevOps implementation and post DevOps implementation

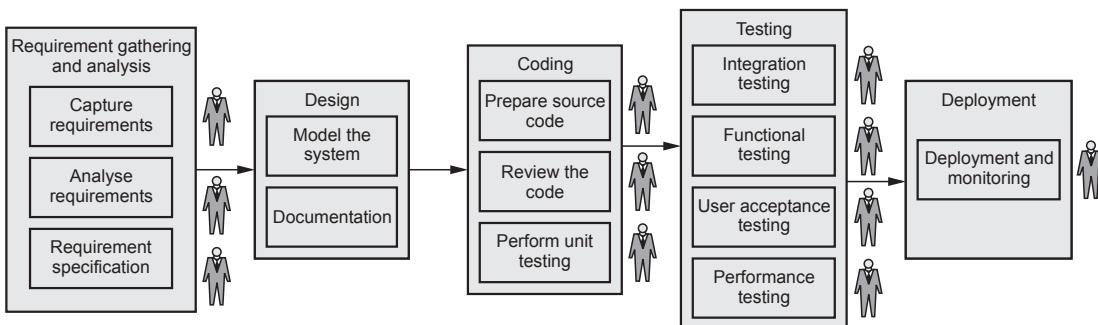


Fig. 9.15.1 Pre DevOps development

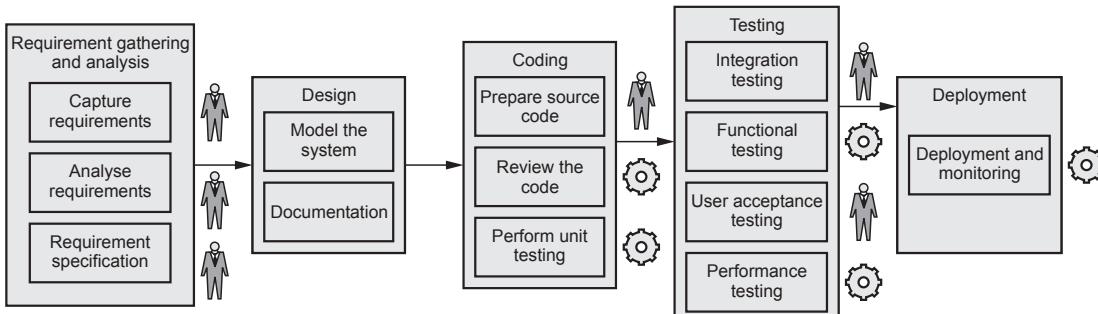


Fig. 9.15.2 Post DevOps implementation

Note that during the use of automated tools at various stages of development life cycle makes the things simplified. For instance at the coding phase, code review can be possible by the tool 'SonarQube', and for unit testing one can use the tool such as Junit. Similarly different tools can be used at testing and deployment phases.

4) Measure and Feedback :

- This is a final important step in which the success of the project is measured.
- After implementation and deployment of the application in the working environment, this step must appear so that the quality of implemented application can be judged.
- Following are the **Key Performer Indicators(KPI)** that measures the benefits of tools and process implementation -
 - **Failure rate** : What is the frequency of occurring failures in the operations ?
 - **Deployment time** : Does the duration of deployment time get shorten ?
 - **Mean time to recovery** : How quickly the application can recover from failure ?
- In addition to these KPIs, the **dashboards** can be provided to measure the success of the application implementation. These dashboards are provided by the tools to monitor the individual tools parameters. Hence it is possible to measure testing time, productivity and so on.
- Thus the ultimate goal of mapping the application to DevOps is to reduce the overall cost of the implementation and to enhance the quality of project.

Review Question

1. Explain Assessment, Definition, Implementation, Measure and Feedback in DevOps.



Notes

10

Advanced Topics in Software Engineering

Syllabus

Component-based software engineering, Client/Server software engineering, Web engineering, Reengineering, Computer-aided software engineering, Software process improvement, Emerging trends in software engineering.

Contents

- 10.1 Component Based Software Engineering (CBSE)
 - 10.2 Client Server Software Engineering **Summer-2018, Winter-2019**, Marks 3
 - 10.3 Web Engineering **Summer-2018**, Marks 7
 - 10.4 Computer Aided Software Engineering (CASE)
 - **Winter-2012, 2013, 2014, 2017, 2019**,
 - **Summer-2011, 2014, 2015, 2016**, Marks 7
 - 10.5 Software Process Improvement. **Summer-2013, 2016**,
 - **Winter-2018**, Marks 7
 - 10.6 Emerging Trends in Software Engineering

10.1 Component Based Software Engineering (CBSE)

Definition : The CBSE is an approach of defining, implementing, integrating loosely coupled independent components into system.

CBSE is basically a reuse based approach.

Benefits

1. By CBSE, it becomes easy to construct understandable and maintainable software.
2. Components are independent entities and they do not interfere in other component's operation.
3. Component implementation is hidden.
4. Communication among the components is through well defined interfaces.
5. Component platform can be shared and ultimately it reduces the cost of development

Drawbacks

1. It is difficult to verify the trustworthiness of the component without source code.
2. The quality of component can not be verified.
3. It is not possible to predict the emergent properties of component compositions.
4. It is difficult to make the trade-offs between the features of various components.

10.1.1 Component and Component Models

According to Council and Heinmann

"Software component is a software element conforms to a component model and can be independently deployed and composed without modification according to a composition standard."

Various characteristics of component are -

1. **Standardized :** The components confirms to some standardized component model. These standards are for defining component interfaces, metadata, composition, deployment and documentation.
2. **Independent :** The component must not depend upon other component while deployment is made.
3. **Deployable :** The component must have an ability to operate as a standalone entity and it should be self-contained.
4. **Composable :** For a component to be composable, all external interactions must take place through publicly defined interfaces.

- 5. Documented :** The component must be fully documented so that any user can decide whether particular component is useful to meet the system requirements.

10.1.2 Component Models

The component model specifies how interfaces should be defined, what are the elements to be included in an interface definition.

Examples of component model are -

1. EJB Model
2. COM Model
3. CORBA Model

10.1.3 CBSE Process

Component composition is the process of 'wiring' components together to create a system. When choosing compositions, you have to consider required functionality, non-functional requirements and system evolution.

The steps for CBSE process are as follows -

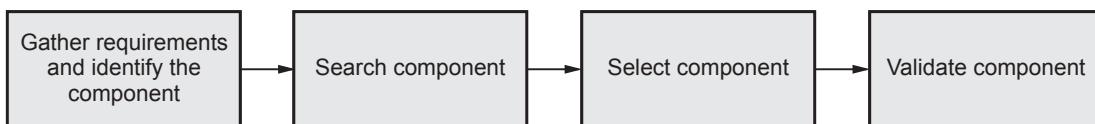


Fig. 10.1.1 CBSE Process

1. Gather the user requirements. Because, using complete set of requirements one can identify as many components as possible for reuse.
2. Refine the requirements depending upon the availability of component. If user requirements can not be satisfied from available components, then sometimes the requirements need to be modified or search for appropriate component is made.
3. During the development process, the discovered component is integrated with software being developed.
4. Then the component validation process is carried out. In this process the compatibility with the interfaces is tested. The selected component must behave as per the requirement. Component validation involves developing a set of test cases for component.

10.2 Client Server Software Engineering

GTU : Summer-2018, Winter-2019, Marks 3

The client-server architecture models the application in such a way that the server consists of a set of services which are demanded by the clients. That means clients demand for the services and servers provide these services to the clients. The clients and servers are the separate processes. The architecture is as shown in the Fig. 10.2.1.

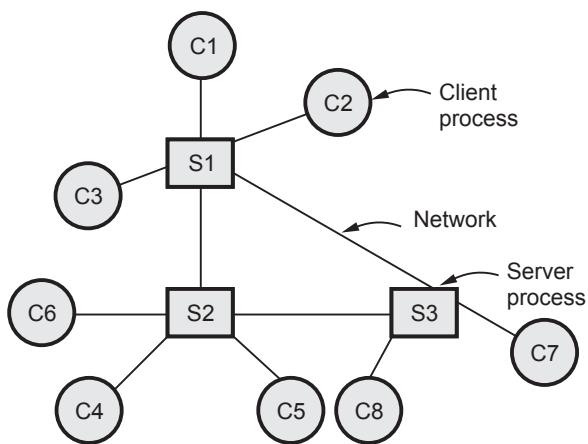


Fig. 10.2.1 Client-server architecture

In client-server architecture, one server might be connected more than one client. The simplest client-server architecture is called two-tier client server architecture in which the application executes on two layers client layer and server layer. The two-tier architecture is of following types -

1. Thin client model

In this model the data management and application logic is implemented on the server and the client is responsible for running the presentation software.

Example

The compiler application makes use of thin client model.

Advantage

- This model is used in simple applications.

Disadvantages

- There is a heavy load on both the server and the network. The server is responsible for all the computations. This ultimately results in heavy network traffic between client and server.
- There is a lot of processing available in modern computing devices. Executing simply the presentation software on clients mean not utilizing the power of these computing devices.

2. Fat client model

In this model, the server is responsible for only data management. The application logic and presentation software is executed on the client itself.

Example

The automatic teller machine (ATM) is an example of fat client model.

The ATM is connected to the server. User operates the ATMs and the information is processed at the client side. The data management part is handled by the server.

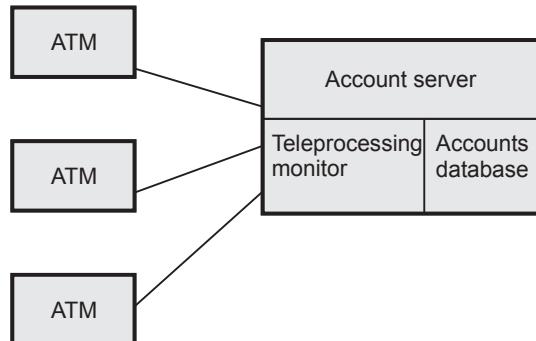


Fig. 10.2.2 ATM system (Fat client model)

Advantages

- This model makes a proper utilization of power computing devices by executing application logic on the clients. Thus processing is distributed effectively in fat client model.
- The application functionality is spread across many computers and thereby efficiency of the overall system gets increased.

Disadvantages

- In this model, the system management is more complex.
- If the application software has to be changed then this involves reinstallation of the software in every client. This process becomes costly.

10.2.2 Three Tier Architecture

In this architecture, the presentation, application processing and data management are logically separate processes and execute on different processors. The architecture is as shown in Fig. 10.2.3.

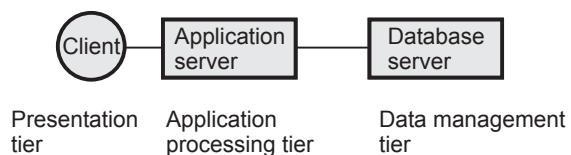


Fig. 10.2.3 Three tier architecture

Example

Internet banking is an example of three tier architecture. In this application the client browses the web page and requests for banking transaction (This is presentation tier). Then the application processes the request and communicates the database server to verify the request for transaction. (This is application processing tier). The database

server stores the bank database and executes the queries. The results of these queries are returned to the application server.

Review Question

1. Explain client / server software engineering.

GTU : Summer-2018, Winter-2019, Marks 3

10.3 Web Engineering

GTU : Summer-2018, Marks 7

Murugesan has defined web engineering as -

Definition : Web engineering is a sound scientific, engineering and management principles and disciplines. It is also systematic and disciplined approaches to the successful development, deployment and maintenance of high quality web based systems and applications.

10.3.1 Attributes of Web Based Applications

Web applications are evolving continuously. Hence attributes of web applications are based on the nature of it. Following attributes are encountered in majority of applications -

1. Concurrency

Large number of users can access the same web application concurrently. Sometimes pattern of usage of these web applications among the users may vary greatly.

2. Performance

If a user has to wait for a long time to access the desired web page then he may lose the interest in accessing it.

3. Network intensiveness

The particular web application may be residing on the internet or it may be present on intranet or may be on the extranet. That means the environment in which the web application is residing may vary.

4. Unpredictable load

The load for accessing particular web application varies greatly. That means, some day 10 users might be accessing the web application and immediately on the next day 10000 users may access the same.

5. Availability

100 % availability of particular web application is just impossible. But the application which is used very frequently should be available to its user almost all the time.

6. Data driven

The primary contents on any web application are text, graphics, sound and video. Some web applications may present information to its user using the databases and such databases might be **on remote machines**.

7. Content sensitivity

It is an important aspect of any web application. The information present of the web application must be authentic. Similarly the contents must be neatly arranged on the web page. The quality of web application is dependant upon this attribute.

8. Continuous evolution

Web applications evolve continuously. There are some web applications that **evolve continuously minute to minute**. For example web systems showing the share market position.

9. Security

The **strong security** is the **topmost demand** for any web application. Sometimes the sensitive information must be conveyed to limited number of users. In such case strong security measure must be applied.

10. Immediacy

Building a web application quickly is today's need. Hence web engineers must use methods for planning, analysis, design, implementation and testing for preparing the target web application within given schedule.

11. Aesthetics

Aesthetic means **something related to the look and feel**. Aesthetic is one of the important attribute of web application. That means the **look and feel** of any web application should be sophisticated and appealing.

10.3.2 Design Model for Web Based Applications

We have already discussed the Web Engineering design model in section 5.8.

Review Questions

1. Define the term **web engineering**. What are the attributes of web based applications ?
2. Explain **web engineering**.

GTU : Summer-2018, Marks 7

10.4 Computer Aided Software Engineering (CASE)

GTU : Winter-2012, 2013, 2014, 2017, 2019, Summer-2011, 2014, 2015, 2016, Marks 7

Importance of CASE Tools

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

10.4.1 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. 10.4.1 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.

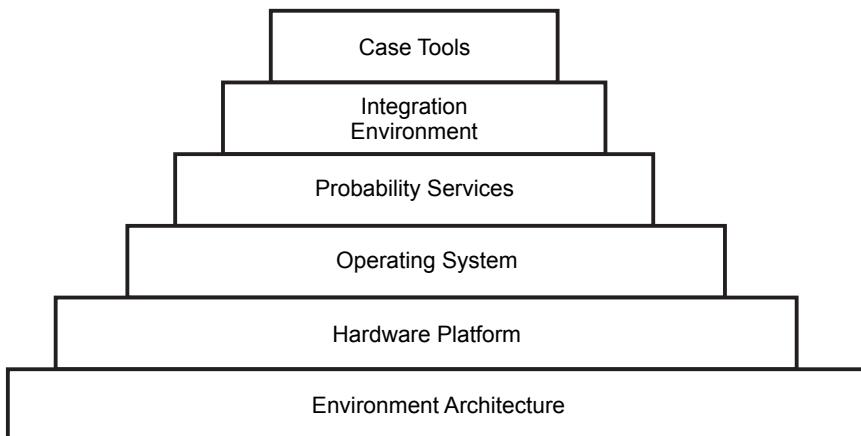


Fig. 10.4.1 Building block for CASE Tools

- 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.

10.4.1.1 Taxonomy of CASE Tools

- 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 help 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.

10.4.2 Integrated CASE Environment

- 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. 10.4.2 shows the simple model for integrated CASE environment. This environment consists of various levels.
- 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).

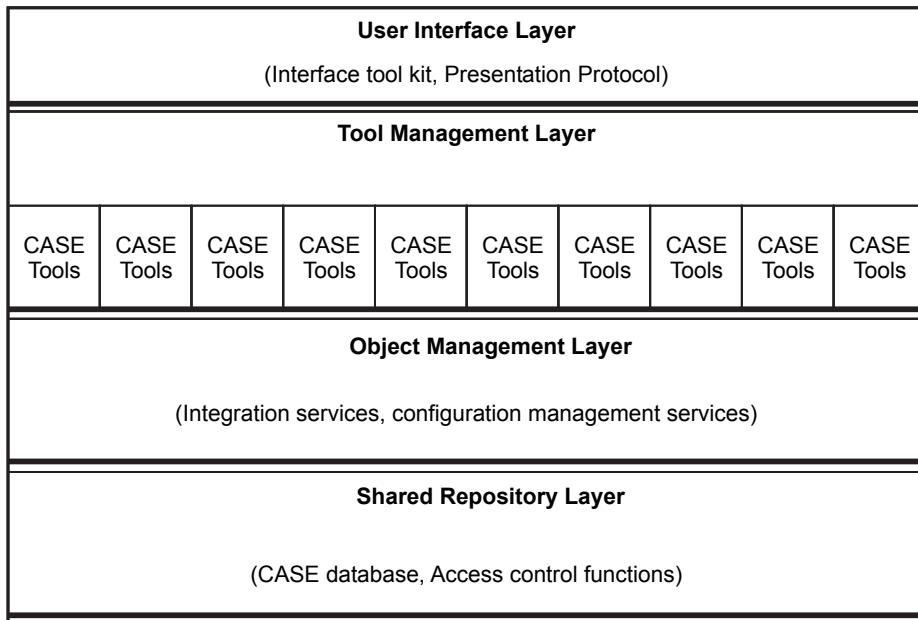


Fig. 10.4.2 Model for integrated CASE environment

- 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.

Review Questions

1. *Describe integrated CASE environment.* **GTU : Summer-2011,Winter-2013, Marks 7**
2. *What does CASE stand for ? Explain all CASE components.* **GTU : Winter-2014, Marks 7**
3. *Explain CASE and building blocks of CASE.* **GTU : Summer-2014, Marks 7**
4. *Explain CASE tools and its use/importance in software engineering.* **GTU : Winter-2012, Summer-2015, Marks 7**
5. *Write a short note on : CASE.* **GTU : Summer-2016, Winter-2019, Marks 7**
6. *Explain CASE tools and its use in software engineering.* **GTU : Winter-2017, Marks 3**

10.5 Software Process Improvement

GTU : Summer-2013, 2016, Winter-2018, Marks 7

Process improvement means understanding the existing process and changing these processes to increase the product quality, to reduce the cost and/or to reduce the development time in order to accelerate the project.

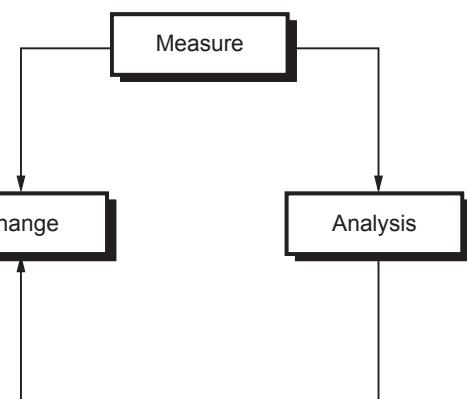
Following are process attributes focusing on the concept of process improvement -

1. **Understandability** : "Is the process definition easy to understand ?" - This aspect is focused for understandability.
2. **Visibility** : "Do the process activities happen in such a manner that the progress of process is visible ?" - This aspect is focused for visibility.
3. **Reliability** : "Is the process design in such a manner that the process errors are avoided before getting introduced in the product as the product error ?" - This aspect is focused for reliability.
4. **Supportability** : "To what extent the CASE tools support the process activities ?" - This aspect is focused.
5. **Robustness** : "Will the system continue to work even if some unexpected errors occur ?" - This aspect is focused.
6. **Acceptability** : "Is the desired process acceptable for producing the software product ?" - This aspect is focused.
7. **Rapidity** : "How fast the processes can be delivered into the system ?" - This aspect is focused.
8. **Maintainability** : "Can the process evolve if any changes or modifications occur in the system ?" - This aspect is focused.

10.5.1 SPI Model

Process improvement is the crucial activity and involves following stages -

- 1) **Process Improvement** : Current project attributes are measured. Then identify the process that needs to be improved.
- 2) **Process Analysis** : The identified process is analysed to obtain the bottlenecks and weaknesses.
- 3) **Process Change** : After analysis phase some processes need to get changed. In this phase the required changes are incorporated in the process.



Review Questions

1. Explain software process improvement. Explain various elements of SPI framework and maturity model.

GTU : Summer-2013, Marks 7

2. Explain software process improvement with various elements of SPI framework.

GTU : Summer-2016, Winter-2018, Marks 7

10.6 Emerging Trends in Software Engineering

Software engineering is a continuously changing stream. Software Intensive systems have become foundation of virtually every modern technology. The software must be demonstrably safe, secure and reliable. Requirements will emerge as systems evolve. Thus the world is demanding for better, more reliable software. Hence new techniques and trends are used in software engineering. Let us learn few emerging trends in software engineering.

10.6.1 Process Trends

The fundamental unit of business, organizational and cultural trends is **process**.

The **six process trends** suggested by Conradi and Fuggetta are as follows -

1. In a rapid software development the focus will be on short term goals that have product innovation.
2. Software engineers have a good sense that where the process is weak. Therefore the process change will be driven by this knowledge.
3. Automation software process technology is used only within those processes which will get benefited most by such automation.
4. Emphasis of process development will be on return of investment of software development activities.
5. As time passes the software community will understand that the software development has a greater impact on sociology and anthropology.
6. New modes of learning will facilitate the effective software processes.

10.6.2 Collaborative Development

- Today, software engineers collaborate across the international boundaries, and every one of them shares the information.
- The challenges over next decade will be to develop methods and tools that facilitate the collaboration for software development.
- Number of success factors that lead to successful collaborative efforts are -

1. **Shared Goals** : The project goals must be clearly specified and all the stakeholders must understand and agree with them.
2. **Shared Culture** : The cultural difference should be clearly defined and proper communication and educational approach must be adopted.
3. **Shared Process** : Process is the basic unit of collaborative project. All the team members work on it to create a successful working system.
4. **Shared Responsibilities** : Every team member must recognize the needs of the customer and should work to give best possible solution.

10.6.3 Model Driven Development

- During the software design phase, architectural and component level abstractions of the system are represented and assessed.
- In the subsequent phases of software development, these design components must be translated into programming language representations. Thus high level abstraction is transformed into low level abstractions. This low level abstract model should work in a specific computing environment.
- **Model driven software development** is an approach of software development in which Domain Specific Modeling Language (DSML) is combined with transformation engines and generators in such a way that high level abstraction is converted into low level abstraction.
- **Domain specific modeling language** represents the application structure, behavior and requirements within particular application domain.
- The domain specific modeling language describes the relationships among various domains, their semantics and the constraints associated with these domains.

10.6.4 Test Driven Development

Test Driven software Development is software development technique in which there occurs very short development cycle followed by repetitive tests. A Test First Design (TFD) approach is adopted in TDD. That means, the first step is always to add the test. If test is passed then add another test. But if test is failed then make some corrections in the code, run the automated test until all the tests get success.

The test driven development cycle

Following are the basic steps followed in TDD cycle.

- **Add a Test**

In the test driven development the development cycle begins by writing a test. To write these tests, developer must understand the feature and requirements clearly.

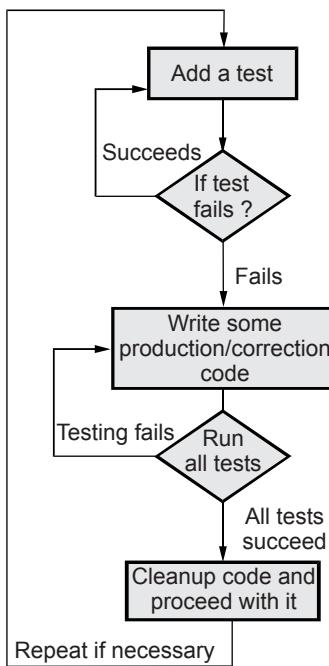


Fig. 10.6.1 Test driven development cycle

- **Run tests**

Execute the test for existing code. If the tests succeed then iteratively add new test.

- **Write correcting code**

If the tests fail then correct the existing code and send it for testing. This will increase the confidence in the code.

- **Run automated tests**

If all the test cases pass in this manner then programmer can be confident that code meets all the tested requirements.

- **Repeat**

Starting with another new test case the cycle will be repeated to push forward the functionality.

Finally the code can be cleaned up. By re-running the test cases the developer is re-factoring the code, testing is without affecting the its functionality.

Review Questions

1. Explain various emerging trends in software engineering.
2. Write a short note on - Test driven development.



Notes