

Software Engineering and Project Management

(Code : 310243)

Semester V - Computer Engineering
(Savitribai Phule Pune University)

**Strictly as per the New Credit System Syllabus (2015 Course)
Savitribai Phule Pune University w.e.f. academic year 2017-2018**

Dr. Sachin D. Babar

Ph.D (Computer Engineering), Professor & Head,
Department of Computer Engineering,
Sinhgad Institute of Technology, Lonavala,
Pune - 410401

M. A. Ansari

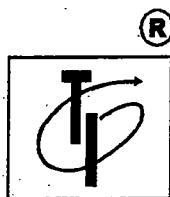
M-Tech (CSE)
Smt. Kashibai Navale College of Engineering,
Vadgaon Budruk, Off Sinhgad Road, Pune - 411041

Dr. Shafi K. Pathan

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

Nalini A. Mhetre

ME(CSE-IT)
Smt. Kashibai Navale College of Engineering,
Vadgaon Budruk, Off Sinhgad Road, Pune - 411041



Tech-Max
Publications, Pune
Innovation Throughout
Engineering Division

P0266A



Software Engineering and Project Management

Dr. Sachin D. Babar, Dr. Shafi K. Pathan, M. A. Ansari, Nalini A. Mhetre

Semester V - Computer Engineering (Savitribai Phule Pune University)

Copyright © by Tech-Max Publications. All rights reserved. No part of this publication may be reproduced, copied, or stored in a retrieval system, distributed or transmitted in any form or by any means, including photocopy, recording, or other electronic or mechanical methods, without the prior written permission of the publisher.

This book is sold subject to the condition that it shall not, by the way of trade or otherwise, be lent, resold, hired out, or otherwise circulated without the publisher's prior written consent in any form of binding or cover other than which it is published and without a similar condition including this condition being imposed on the subsequent purchaser and without limiting the rights under copyright reserved above.

First Printed in India : August 2006 (Pune University)

First Edition : June 2017

This edition is for sale in India, Bangladesh, Bhutan, Maldives, Nepal, Pakistan, Sri Lanka and designated countries in South-East Asia. Sale and purchase of this book outside of these countries is unauthorized by the publisher.

Printed at : Image Offset, Dugane Ind. Area, Survey No. 28/25, Dhayari, Near Pari Company,
Pune - 41, Maharashtra State, India. E-mail : rahulshahimage@gmail.com

ISBN 978-93-5224-578-9

Published by

Tech-Max Publications

Head Office : B/5, First floor, Maniratna Complex, Taware Colony, Aranyeshwar Corner,

Pune - 411 009. Maharashtra State, India.

Ph : 91-20-24225065, 91-20-24217965. Fax 020-24228978. Email : info@techmaxbooks.com,

Website : www.techmaxbooks.com

[310243] (FID : TP465) (Book Code : PO266A)

Syllabus

Savitribai Phule Pune University

Third Year of Computer Engineering (2015 Course)

310243 : Software Engineering and Project Management

Teaching Scheme	Credits	Examination Scheme
TH : 03 Hours/Week	03	In-Sem (Paper) : 30 Marks
		End-Sem (Paper) : 70 Marks

Prerequisite Courses : Fundamentals of Programming Languages (110003, 110011)

Course Objectives

- To learn and understand the principles of Software Engineering.
- To be acquainted with methods of capturing, specifying, visualizing and analyzing software requirements.
- To apply Design and Testing principles to S/W project development.
- To understand project management through life cycle of the project.
- To understand software quality attributes.

Course Outcomes

On completion of the course, student will be able to

- Decide on a process model for developing a software project.
- Classify software applications and Identify unique features of various domains.
- Design test cases of a software system.
- Understand basics of IT Project management.
- Plan, schedule and execute a project considering the risk management.
- Apply quality attributes in software development life cycle.

Course Contents

Unit I - Introduction to Software Engineering, Software Process Models (07 Hours)

Software Engineering Fundamentals : Nature of Software, Software Engineering Principles, The Software Process, Software Myths. **Process Models :** A Generic Process Model, **Prescriptive Process Models :** The Waterfall, Incremental Process (RAD), Evolutionary Process, Unified Process, Concurrent. **Advanced Process Models and Tools :** Agile software development : Agile methods, Plan-driven and agile development, Extreme programming Practices, Testing in XP, Pair programming. Introduction to agile tools : JIRA, Kanban, Case Studies : An information system (mental health-care system), wilderness weather system.

(Refer Chapters 1, 2 and 3)

Unit II - Software Requirements Engineering and Analysis (08 Hours)

Requirements Engineering : User and system requirements, Functional and non-functional requirements, Types and Metrics, A spiral view of the requirements engineering process. **Software Requirements Specification (SRS) :** The software requirements Specification document, The structure of SRS, Ways of writing a SRS, structured and tabular SRS for an insulin pump case study, **Requirements elicitation and Analysis :** Process, Requirements validation, Requirements management. **Case Studies :** The information system. Case study - Mental health care patient management system (MHC-PMS). (Refer Chapter 4)

Unit III - Design Engineering**(08 Hours)**

Design Process and quality, Design Concepts, The design Model, Pattern-based Software Design, **Architectural Design** : Design Decisions, Views, Patterns, Application Architectures, **Modeling Component level Design** : Component, Designing class based components, conducting component-level design, **User Interface Design** : The golden rules, Interface Design steps and Analysis, Design Evaluation, **Case Study** : Web App Interface Design.

(Refer Chapters 5, 6 and 7)

Unit IV - Project Management : Process, Metrics, Estimations and Risks**(08 Hours)**

Project Management Concepts : The Management Spectrum, People, Product, Process, Project, The W5HH Principle, Metrics in the Process and Project Domains, Software Measurement : size and function oriented metrics (FP and LOC), Metrics for Project and Software Quality, **Project Estimation** : Observations on Estimation, Project Planning Process, Software Scope and feasibility, Resources : Human Resources, Reusable software, Environmental Resources. Software Project Estimation, Decomposition Techniques, Empirical Estimation Models : Structure, COCOMO II, Estimation of Object-oriented Projects, Specialized Estimation **Case Study** : Software Tools for Estimation, **Project Scheduling** : Basic Concepts, Defining a Task Set for the Software Project, Defining Task Network, Scheduling with time-line charts, Schedule tracking Tools : Microsoft Project, Daily Activity Reporting and Tracking (DART).

(Refer Chapters 8 and 9)

Unit V - Project Management : Risk Management, Configuration Management, Maintenance and Reengineering**(07 Hours)**

Project Risk Management : Risk Analysis and Management : Reactive versus Proactive Risk Strategies, Software Risks, Risk Identification, Risk Projection, Risk Refinement, Risk Mitigation, Risks Monitoring and Management, The RMMM plan for case study project.

Software Configuration Management : The SCM repository, SCM process, Configuration management for WebApps, **Case study** : CVS and Subversion Tools, Visual Source Safe from Microsoft and Clear Case. **Maintenance and Reengineering** : Software Maintenance, Software Supportability, Reengineering, Business Process Reengineering, Software Reengineering, Reverse Engineering, Restructuring, Forward Engineering.

(Refer Chapters 10, 11 and 12)

Unit VI - Software Testing**(07 Hours)**

Introduction to Software Testing, Principles of Testing, Testing Life Cycle, Phases of Testing, Types of Testing, Verification and Validation, Defect Management, Defect Life Cycle, Bug Reporting, GUI Testing, Test Management and Automation.

(Refer Chapter 13)



**UNIT I**

Syllabus : Software Engineering Fundamentals : Nature of Software, Software Engineering Principles, The Software Process, Software Myths. Process Models : A Generic Process Model, Prescriptive Process Models : The Waterfall, Incremental Process (RAD), Evolutionary Process, Unified Process, Concurrent Advanced Process Models and Tools : Agile software development : Agile methods, Plan-driven and agile development, Extreme programming Practices, Testing in XP, Pair programming. Introduction to agile tools : JIRA, Kanban, Case Studies : An information system (mental health-care system), wilderness weather system.

Chapter 1 : Software Engineering Fundamentals**1-1 to 1-11**

✓	Syllabus Topic : Software Engineering Fundamentals (Introduction).....	1-1
1.1	Introduction	1-1
✓	Syllabus Topic : Nature of Software	1-1
1.2	Nature of Software	1-1
1.2.1	Absence of Fundamental Theory	1-2
1.2.2	Ease of Change	1-2
1.2.3	Rapid Evolution of Technologies.....	1-2
1.2.4	Low Manufacturing Cost	1-2
1.3	Software Engineering : A Layered Technology (SPPU - May 13, May 14, Dec. 14, 6 Marks).....	1-3
1.3.1	Quality Focus	1-3
1.3.2	Process	1-3
1.3.3	Methods.....	1-3
1.3.4	Tools.....	1-3
1.3.5	The Characteristics of Software (SPPU - May 14, Dec. 16, Apr. 17).....	1-4
1.3.6	Software Crisis (SPPU - May 13).....	1-4
1.3.7	Legacy Software (SPPU - May 13).....	1-4
✓	Syllabus Topic : Software Engineering Principles	1-5
1.4	Software Engineering Principles	1-5
✓	Syllabus Topic : The Software Process.....	1-7
1.5	The Software Process (SPPU – May 12, May 15)	1-7
1.5.1	Umbrella Activities (SPPU – Feb. 15, May 16)	1-7
✓	Syllabus Topic : Software Myths	1-8
1.6	Software Myths (SPPU - May 12, May 13, Dec. 13, Dec. 14, Feb. 15) ...	1-8
1.6.1	Management Level Myths (or Manager Level Myths) (SPPU - May 14).....	1-9
1.6.2	Customer Level Myths (SPPU – May 14)	1-10
1.6.3	Practitioner Level Myths (or Developer Level Myths) (SPPU – Dec. 12, Dec. 13, Feb. 16, Apr. 17)	1-10
1.7	Importance of Software Engineering (SPPU - Dec. 14)	1-11

Chapter 2 : Process Models**2-1 to 2-16**

✓	Syllabus Topic : A Generic Process Model.....	2-1
2.1	A Generic Process Model (or Generic Process Framework (SPPU - Dec. 12)	2-1
2.1.1	Communication	2-2
2.1.2	Planning	2-2
2.1.3	Modeling.....	2-2
2.1.4	Construction	2-2
2.1.5	Deployment.....	2-2
✓	Syllabus Topic : Prescriptive Process Models.....	2-2
2.2	Prescriptive Process Models.....	2-2
✓	Syllabus Topic : The Waterfall Model	2-3
2.2.1	The Waterfall Model (SPPU – Dec. 12, Dec. 13, Dec. 14)	2-3
2.2.1.1	V-Model (Software Development) (SPPU - Dec. 13).....	2-4
✓	Syllabus Topic : Incremental Process Models.....	2-5
2.2.2	Incremental Process Models.....	2-5
✓	Syllabus Topic : The Incremental Model.....	2-5
2.2.2.1	The Incremental Model (SPPU - May 14, Dec. 14)	2-5
✓	Syllabus Topic : The RAD Model.....	2-6
2.2.2.2	The RAD Model.....	2-6
✓	Syllabus Topic : Evolutionary Process Models	2-7
2.2.3	Evolutionary Process Models (SPPU - May 12).....	2-7
2.2.3.1	The Prototyping Paradigm (SPPU – Dec. 13)	2-8
2.2.3.2	The Spiral Model (SPPU - Dec. 13).....	2-9
✓	Syllabus Topic : Concurrent Model.....	2-10
2.2.3.3	The Concurrent Development Model	2-10
2.2.3.4	Differentiation between Prescriptive and Evolutionary Process Models.....	2-11
2.2.4	The Specialized Process Models.....	2-11
2.2.4.1	Component-Based Development Models (SPPU - Dec. 13, Dec. 14)	2-12
2.2.4.2	The Formal Methods Model	2-13
2.2.4.3	Aspect-Oriented Software Development.....	2-13
2.2.4.4	Need of Process Models	2-14
✓	Syllabus Topic : The Unified Process	2-14
2.3	The Unified Process (SPPU - Dec 16).....	2-14
2.3.1	The Phases of Unified Process (SPPU – Dec. 12)....	2-15
2.3.2	The Iteration among Four Phases	2-16

Chapter 3 : Advanced Process Models and Tools**3-1 to 3-29**

3.1	Agile Process Model	3-1
3.1.1	Comparison between the Agile and Evolutionary Process Models (SPPU - May 13, Feb. 16).....	3-1
✓	Syllabus Topic : Agile Software Development.....	3-2
3.2	Agile Software Development.....	3-2
✓	Syllabus Topic : Agile Methods.....	3-3
3.2.1	Agile methods	3-3
3.2.2	Agile Manifesto.....	3-3
3.2.3	Agility Principles (SPPU – Apr. 17)	3-3
✓	Syllabus Topic : Plan-Driven and Agile Development .	3-4



3.3	Plan-Driven and Agile Development.....	3-4
3.3.1	Comparison between Plan-driven and Agile Development	3-5
✓	Syllabus Topic : Extreme Programming Practices	3-5
3.4	Extreme Programming Practices (SPPU –Dec. 15, Feb. 16).....	3-5
3.4.1	XP Values	3-6
3.4.2	The XP Process (SPPU - Dec. 14, Dec. 15).....	3-6
3.4.3	Scrum.....	3-7
3.4.3.1	Process Flow	3-7
3.4.4	Scrum Roles	3-8
3.4.5	Scrum Cycle Description	3-9
3.4.6	Product Backlog.....	3-10
3.4.7	Sprint Planning Meeting.....	3-11
3.4.8	Sprint Backlog.....	3-11
3.4.9	Sprint Execution.....	3-12
3.4.10	Daily Scrum Meeting.....	3-12
3.4.11	Maintaining Sprint Backlog and Burn-Down Chart	3-13
3.4.12	Sprint Review and Retrospective	3-14
✓	Syllabus Topic : Testing in XP	3-14
3.5	Testing in XP	3-14
3.5.1	Exploratory Testing Versus Scripted Testing	3-14
3.6	Agile Practices	3-15
✓	Syllabus Topic : Pair Programming	3-15
3.6.1	Pair Programming.....	3-15
3.6.2	Refactoring	3-16
3.6.3	Test Driven Development	3-17
3.6.4	Continuous Integration.....	3-18
3.7	Agile Modeling (AM)	3-19
✓	Syllabus Topic : Introduction to Agile Tools : JIRA.....	3-20
3.8	Introduction to Agile Tools : JIRA.....	3-20
3.8.1	JIRA platform	3-21
3.8.2	JIRA Architecture.....	3-21
3.8.3	JIRA Database Schema	3-21
3.8.4	JIRA Mobile Connect.....	3-22
3.8.5	JIRA Applications.....	3-22
3.8.6	JIRA APIs	3-22
✓	Syllabus Topic : Introduction to Agile Tools : Kanban	3-22
3.9	Introduction to Agile Tools : Kanban.....	3-22
3.9.1	Kanban Boards.....	3-23
3.9.2	Kanban Cards.....	3-23
3.9.3	The Benefits of Kanban	3-23
3.9.4	Comparison between Kanban and Scrum.....	3-26
✓	Syllabus Topic : Case Study : An Information System(Mental Health-care System)	3-26
3.10	Mental Health-Care System.....	3-26
✓	Syllabus Topic : Case Study : Wilderness Weather System.....	3-28
3.11	Wilderness Weather System	3-28

UNIT II

Syllabus : Requirements Engineering : User and system requirements, Functional and non-functional requirements, Types and Metrics, A spiral view of the requirements engineering process. Software Requirements Specification (SRS) : The software requirements Specification document, The structure of SRS, Ways of writing a SRS, structured and tabular SRS for an insulin pump case study, Requirements elicitation and Analysis : Process, Requirements validation, Requirements management. Case Studies : The information system. Case study - Mental health care patient management system (MHC-PMS).

Chapter 4 : Requirement Engineering 4-1 to 4-18

✓	Syllabus Topic : Requirements Engineering.....	4-1
4.1	Introduction	4-1
4.2	Requirement Engineering	4-1
4.2.1	Inception (SPPU - Feb. 16).....	4-2
4.2.2	Elicitation (SPPU - Feb. 16).....	4-2
4.2.3	Elaboration	4-2
4.2.4	Negotiation (SPPU - Feb. 15).....	4-3
4.2.5	Specification.....	4-3
4.2.6	Validation (SPPU – Feb.15, Feb. 16).....	4-3
4.2.7	Requirement Management	4-3
4.2.8	Initiating the Requirement Engineering Process.....	4-3
✓	Syllabus Topic : User and System Requirements	4-5
4.3	User and System Requirements	4-5
✓	Syllabus Topic : Functional and Non - Functional Requirements.....	4-6
4.4	Functional and Non- Functional Requirements (SPPU - Feb. 15, May 16).....	4-6
4.4.1	Functional Requirements	4-6
4.4.2	Non-functional Requirements	4-6
✓	Syllabus Topic : Types and Metrics	4-7
4.5	Types and Metrics	4-7
4.5.1	Metrics for Specifying Non-functional Requirements.....	4-7
✓	Syllabus Topic : A Spiral View of the Requirements Engineering Process	4-8
4.6	A Spiral View of the Requirements Engineering Process	4-8
✓	Syllabus Topic : Software Requirements Specification (SRS), The Structure of SRS	4-8
4.7	Software Requirements Specification (SRS)	4-8
✓	Syllabus Topic : Ways of Writing a SRS	4-9
4.7.1	Writing Software Requirements Specifications	4-9
4.7.2	What is a Software Requirements Specification?	4-9
4.7.3	What Kind of Information Should an SRS Include?	4-10
4.7.4	SRS Template	4-10
4.7.5	Characteristics of an SRS (SPPU - Feb. 15, Feb. 16).....	4-10



✓	Syllabus Topic : Structured SRS for an Insulin Pump Case Study.....	4-11	5.4.1	Abstraction	5-3
4.7.6	Structured Specifications for an Insulin Pump Case Study.....	4-11	5.4.2	Architecture (SPPU – May 13).....	5-3
✓	Syllabus Topic : Tabular SRS for an Insulin Pump Case Study	4-12	5.4.3	Patterns.....	5-3
4.7.7	Tabular Specifications for an Insulin Pump Case Study	4-12	5.4.4	Modularity (SPPU – May 13).....	5-3
✓	Syllabus Topic : Requirements Elicitation and Analysis : Process	4-12	5.4.5	Information Hiding.....	5-5
4.8	Requirements Elicitation : Process (SPPU - Dec. 12).....	4-12	5.4.6	Functional Independence (SPPU – Dec. 16).....	5-5
4.8.1	Collaborative Requirements Gathering	4-12	5.4.7	Refinement (SPPU - May 13).....	5-7
4.8.2	Quality Function Deployment.....	4-13	5.4.8	Refactoring (SPPU - May 15, Dec. 15).....	5-8
4.8.3	Usage Scenarios	4-13	5.4.8.1	Importance of refactoring (SPPU - May 15, Dec. 15)....	5-8
4.8.4	Elicitation Workproduct (SPPU - May 13)	4-13	5.4.9	Design Classes	5-8
4.8.5	Elicitation Techniques.....	4-14	5.4.10	Differentiation between Abstraction and Refinement (SPPU - Dec. 12).....	5-8
4.8.6	Developing Use Cases	4-14	✓	Syllabus Topic : The Design Model	5-9
4.9	Requirements Analysis.....	4-15	5.5	The Design Model	5-9
4.9.1	Analysis Rules of Thumb.....	4-16	5.5.1	Data Design Elements	5-10
4.9.2	Domain Analysis	4-16	5.5.2	Architectural Design Elements.....	5-10
4.9.3	Requirements Modeling Approaches	4-17	5.5.3	Interface Design Elements.....	5-10
✓	Syllabus Topic : Requirements Validation.....	4-17	5.5.4	Component-Level Design Elements	5-11
4.10	Requirements Validation (SPPU - Dec. 15, Dec. 16)	4-17	5.5.5	Deployment-Level Design Elements.....	5-11
✓	Syllabus Topic : Requirements Management.....	4-17	5.5.6	Translating Requirements Model to Design Model.....	5-12
4.11	Requirements Management (SPPU - May 14).....	4-17	5.5.7	Guidelines for the Data Design	5-12
✓	Syllabus Topic : Case Studies : MHC-PMS	4-18	✓	Syllabus Topic : Pattern-Based Software Design.....	5-13
4.12	Case Studies : Mental Health Care Patient Management System (MHC-PMS)	4-18	5.6	Pattern-Based Software Design.....	5-13
			5.6.1	Describing a Design Pattern	5-13
			5.6.2	Using Patterns in Design	5-13
			5.6.3	Frameworks	5-14

UNIT III

Syllabus : Design Process and quality, Design Concepts, The design Model, Pattern-based Software Design. Architectural Design : Design Decisions, Views, Patterns, Application Architectures, Modeling Component level Design : Component, Designing class based components, conducting component-level design, User Interface Design : The golden rules, Interface Design steps and Analysis, Design Evaluation, Case Study : Web App Interface Design.

Chapter 5 : Design Engineering 5-1 to 5-14

✓	Syllabus Topic : Design Engineering.....	5-1
5.1	Introduction to Design Engineering	5-1
✓	Syllabus Topic : Design Process	5-1
5.2	Design Process.....	5-1
✓	Syllabus Topic : Design Quality.....	5-1
5.3	Design Quality	5-1
5.3.1	Quality of Design Guidelines	5-2
5.3.2	The Quality Attributes (SPPU - Dec. 12)	5-2
✓	Syllabus Topic : Design Concepts.....	5-3
5.4	Design Concepts	5-3

Chapter 6 : Architectural Design 6-1 to 6-12

✓	Syllabus Topic : Architectural Design	6-1
6.1	Introduction to Architectural Design	6-1
✓	Syllabus Topic : Design Decisions.....	6-3
6.2	Architectural Design Decisions	6-3
✓	Syllabus Topic : Views.....	6-4
6.3	Architectural Views.....	6-4
✓	Syllabus Topic : Patterns	6-5
6.4	Architectural Patterns (SPPU - Dec. 16).....	6-5
6.4.1	Software Architecture (SPPU - Feb. 16).....	6-6
✓	Syllabus Topic : Application Architectures.....	6-6
6.5	Application Architectures	6-6
6.5.1	Transaction Processing Systems.....	6-7
6.5.2	Language Processing Systems	6-8
✓	Syllabus Topic : Modeling Component Level Design : Component.....	6-9
6.6	Modeling Component level Design	6-9
✓	Syllabus Topic : Designing Class based Components	6-11
6.7	Class-based Components.....	6-11
6.7.1	Basic Design Principles.....	6-12
✓	Syllabus Topic : Conducting Component-level Design	6-12
6.7.2	Conducting Component-Level Design	6-12



Chapter 7 : User Interface Design	7-1 to 7-13	Chapter 8 : Project Management Concepts 8-1 to 8-29	
✓ Syllabus Topic : User Interface Design	7-1	✓ Syllabus Topic : The Management Spectrum.....	8-1
7.1 User Interface Design.....	7-1	8.1 The Management Spectrum	
7.1.1 Type of User Interface	7-2	(SPPU - May 12, May 13, May 15).....	8-1
7.1.2 Characteristics of Good User Interface	7-3	✓ Syllabus Topic : People	8-1
7.1.3 Benefits of Good Interface Design.....	7-4	8.1.1 The People (SPPU - May 12)	8-1
✓ Syllabus Topic : The Golden Rules	7-4	8.1.1.1 Stake Holders (SPPU - May 13, May 14).....	8-2
7.2 The Golden Rules (SPPU – Dec. 13, May 14).....	7-4	8.1.1.2 Team Leaders.....	8-2
7.2.1 Place the user in Control	7-4	8.1.1.3 Software Team.....	8-3
7.2.2 Reduce the User's Memory Load (SPPU – Dec. 12)	7-5	8.1.1.4 Agile Teams	8-3
7.2.3 Make the Interface Consistent.....	7-6	8.1.1.5 Co-ordination and Communication Issues.....	8-4
7.2.4 Necessity of a Good User Interface (SPPU - May 12) ..	7-6	✓ Syllabus Topic : Product	8-4
7.3 Shneiderman's 8 Golden Rules for UI Analysis (SPPU – May 12).....	7-7	8.1.2 The Product (SPPU - May 12)	8-4
7.4 Interface Analysis and Design Models	7-8	✓ Syllabus Topic : Process	8-5
7.4.1 Interface Analysis and Design Models	7-8	8.1.3 The Process.....	8-5
7.4.2 User Interface Design Process (SPPU – Dec. 13, Dec. 14, Feb. 16, May 16, Dec. 16).....	7-8	✓ Syllabus Topic : Project	8-5
✓ Syllabus Topic : Interface Design Steps and Analysis	7-9	8.1.4 The Project.....	8-5
7.5 Interface Design Steps and Analysis (SPPU – Dec. 12, Dec.16)	7-9	✓ Syllabus Topic : The W5HH Principle.....	8-5
7.5.1 Applying Interface Design Steps	7-10	8.2 The W5HH Principle.....	8-5
7.5.2 User Interface Design Patterns	7-10	✓ Syllabus Topic : Metrics in the Process and Project Domains	8-6
7.5.3 Interface Design Issues (SPPU – Dec. 14, Apr. 17) ...	7-10	8.3 Metrics in the Process and Project Domains (SPPU - May 15, Dec. 15)	8-6
7.5.4 Interface Design Evaluation.....	7-11	8.3.1 Process Metrics.....	8-6
✓ Syllabus Topic : Design Evaluation	7-12	✓ Syllabus Topic : Metrics for Project	8-7
7.6 Design Evaluation.....	7-12	8.3.2 Project Metrics	8-7
✓ Syllabus Topic : Case Study : WebApp Interface Design	7-12	✓ Syllabus Topic : Software Measurement	8-7
7.7 WebApp Interface Design.....	7-12	8.4 Software Measurement.....	8-7
7.7.1 WebApp Design Principles (SPPU – May 12, Apr. 17)	7-12	✓ Syllabus Topic : Software Measurement : Size and Function Oriented Metrics (FP and LOC)	8-7
UNIT IV		8.4.1 Size-Oriented Metrics (SPPU- May 14)	8-7
Syllabus : Project Management Concepts : The Management Spectrum, People, Product, Process, Project, The W5HH Principle, Metrics in the Process and Project Domains, Software Measurement : size and function oriented metrics (FP and LOC), Metrics for Project and Software Quality, Project Estimation : Observations on Estimation, Project Planning Process, Software Scope and feasibility, Resources : Human Resources, Reusable software, Environmental Resources. Software Project Estimation, Decomposition Techniques, Empirical Estimation Models : Structure, COCOMO II, Estimation of Object-oriented Projects, Specialized Estimation		8.4.2 Function-Oriented Metrics (FP and LOC)	8-8
Case Study : Software Tools for Estimation, Project Scheduling : Basic Concepts, Defining a Task Set for the Software Project, Defining Task Network, Scheduling with time-line charts, Schedule tracking Tools : Microsoft Project, Daily Activity Reporting and Tracking (DART).		8.4.3 Reconciling LOC and FP Metrics	8-8
		8.4.4 Object-Oriented Metrics	8-9
		8.4.5 Integrating Metrics within the Software Process	8-9
		✓ Syllabus Topic : Software Quality	8-10
		8.4.6 Software Quality (SPPU - May 14, Dec. 14, May 15, Dec. 15, May 16)	8-10
		8.4.6.1 McCall's Quality Factors (SPPU - Dec. 15, May 16)	8-11
		8.4.6.2 ISO 9126 Quality Factors (SPPU - May 12, Dec. 16)	8-12
		8.4.7 Software Reliability (SPPU - May 15, May 16)	8-13
		8.4.7.1 Measures of Reliability and Availability (SPPU - May 15)	8-13
		8.4.7.2 Software Safety.....	8-13
		✓ Syllabus Topic : Project Estimation : Observations on Estimation	8-14
		8.5 Observations on Estimation	8-14
		8.5.1 Software Sizing	8-14
		8.5.2 Problem-Based Estimation	8-15
		8.5.3 An Example of LOC-Based Estimation	8-15
		8.5.4 An Example of FP-Based Estimation	8-16



8.5.5	Process-Based Estimation.....	8-16
8.5.6	An Example of Process-Based Estimation	8-17
8.5.7	Estimation with Use-Cases.....	8-18
8.5.8	An Example of Use-Case Based Estimation	8-18
8.5.9	Reconciling Estimates	8-19
✓	Syllabus Topic : Software Scope and Feasibility.....	8-19
8.5.10	Software Scope and Feasibility	8-19
8.5.10.1	Obtaining Information Necessary for Scope	8-20
8.5.10.2	Feasibility	8-20
8.5.10.3	A Scoping Example.....	8-20
✓	Syllabus Topic : Resources.....	8-20
8.6	Resources (SPPU - Dec. 16).....	8-20
✓	Syllabus Topic : Human Resources.....	8-21
8.6.1	Human Resources	8-21
✓	Syllabus Topic : Reusable Software.....	8-22
8.6.2	Reusable Software Resources	8-22
✓	Syllabus Topic : Environmental Resources.....	8-22
8.6.3	Environmental Resources.....	8-22
✓	Syllabus Topic : Software Project Estimation.....	8-22
8.7	Software Project Estimation.....	8-22
✓	Syllabus Topic : Decomposition Techniques.....	8-23
8.8	Decomposition Techniques	8-23
8.8.1	Problem Decomposition.....	8-23
8.8.2	Process Decomposition	8-23
✓	Syllabus Topic : Empirical Estimation Models.....	8-24
8.9	Empirical Estimation Models.....	8-24
✓	Syllabus Topic : Empirical Estimation Model : Structure	8-25
8.9.1	The Structure of Estimation Models	8-25
✓	Syllabus Topic : Empirical Estimation Model : COCOMO II	8-25
8.9.2	The COCOMO II Model	8-25
	(SPPU - Dec. 13, Dec. 15, May 16, Dec. 16)	8-25
8.9.3	The Software Equation	8-27
✓	Syllabus Topic : Estimation of Object-Oriented Projects	8-27
8.10	Estimation of Object-Oriented Projects	8-27
✓	Syllabus Topic : Specialized Estimation.....	8-28
8.11	Specialized Estimation.....	8-28
8.11.1	Estimation for Agile Development	8-28
8.11.2	Estimation for Web Engineering Projects	8-28
✓	Syllabus Topic : Case Study : Software Tools for Estimation	8-29
8.12	Case Study : Software Tools for Estimation	8-29

Chapter 9 : Project Scheduling 9-1 to 9-6

✓	Syllabus Topic : Project Scheduling - Basic Concepts.....	9-1
9.1	Project Scheduling (SPPU - May 12, Dec. 16)	9-1
✓	Syllabus Topic : Defining a Task Set for the Software Project.....	9-2
9.1.1	Defining a Task Set for the Software Project.....	9-2
9.1.2	Scheduling	9-2

✓	Syllabus Topic : Scheduling with Time-line Charts (SPPU - Dec. 12)	9-2
9.1.3	Tracking the Schedule	9-3
9.2	Earned Value Analysis (SPPU - May 16)	9-3
✓	Syllabus Topic : Schedule Tracking Tools	9-4
9.3	Schedule Tracking Tools	9-4
✓	Syllabus Topic : Schedule Tracking Tools - Microsoft Project	9-4
9.3.1	Microsoft Project	9-4
✓	Syllabus Topic : Schedule Tracking Tools : Daily Activity Reporting & Tracking(DART)	9-5
9.3.2	Daily Activity Reporting and Tracking (DART)	9-5

UNIT V

Syllabus : Project Risk Management : Risk Analysis and Management : Reactive versus Proactive Risk Strategies, Software Risks, Risk Identification, Risk Projection, Risk Refinement, Risk Mitigation, Risks Monitoring and Management, The RMMM plan for case study project.

Software Configuration Management : The SCM repository, SCM process, Configuration management for WebApps, Case study : CVS and Subversion Tools, Visual Source Safe from Microsoft and Clear Case. Maintenance and Reengineering : Software Maintenance, Software Supportability, Reengineering, Business Process Reengineering, Software Reengineering, Reverse Engineering, Restructuring, Forward Engineering.

Chapter 10 : Project Risk Management 10-1 to 10-13

✓	Syllabus Topic : Risk Analysis and Management.....	10-1
10.1	Risk Analysis and Management	10-1
	(SPPU – May 14, May 15, Dec. 15, May 16)	10-1
10.1.1	Software Risks	10-1
10.1.2	Reactive Versus Proactive Risk Strategies.....	10-2
✓	Syllabus Topic : Risk Identification	10-2
10.2	Risk Identification	10-2
	(SPPU – Dec. 12, May 13, Dec. 13, Dec. 16)	10-2
10.2.1	Assessing Overall Project Risk	10-3
10.2.2	Risk Components and Drivers	10-4
✓	Syllabus Topic : Risk Projection	10-5
10.3	Risk Projection	10-5
10.3.1	Developing a Risk Table (SPPU - Dec. 13)	10-5
10.3.2	Assessing Risk	10-6
10.3.3	Project Plan	10-7
✓	Syllabus Topic : Risk Refinement	10-7
10.4	Risk Refinement	10-7
✓	Syllabus Topic : Risk Mitigation, Risk Monitoring and Risk Management (RMMM)	10-8
10.5	Risk Mitigation, Risk Monitoring and Risk Management (RMMM) (SPPU - May 12, May 14)	10-8
10.5.1	The RMMM Plan	10-8



✓	Syllabus Topic : The RMMM Plan for Case Study Project.....	10-11
10.6	The RMMM Plan for Case Study Project.....	10-11
10.6.1	The general overview of RMMM Plan for WMITS	10-11
10.6.2	The Description of Risk for WMITS	10-11
10.6.3	Risk Mitigation, Monitoring and Management for WMITS.....	10-12

Chapter 11 : Software Configuration Management (SCM)**11-1 to 11-12**

✓	Syllabus Topic : Software Configuration Management (SCM).....	11-1
11.1	Software Configuration Management (SCM).....	11-1
	(SPPU - May 12, May 16)	
11.1.1	SCM Basics (Configuration Management System Elements).....	11-2
11.1.2	Baselines	11-2
11.1.3	Software Configuration Items	11-2
✓	Syllabus Topic : The SCM Repository.....	11-3
11.2	The SCM Repository (SPPU - May 12, Dec. 14)	11-3
11.2.1	The Role of the Repository.....	11-3
11.2.2	General Features and Content.....	11-4
11.2.3	SCM Features.....	11-5
✓	Syllabus Topic : The SCM Process.....	11-6
11.3	The SCM Process.....	11-6
	(SPPU - May 12, Dec. 12, May 13, Dec. 13)	
11.3.1	Identification of Objects in the Software Configuration	11-7
11.3.2	Version Control.....	11-7
11.3.3	Change Control (SPPU - Dec. 12, Dec. 13, May 14, May 16).....	11-7
11.3.4	Configuration Audit	11-8
11.3.5	Status Reporting	11-9
✓	Syllabus Topic : Configuration Management for WebApps	11-9
11.4	Configuration Management for WebApps	11-9
✓	Syllabus Topic : Case Study : CVS and Subversion Tools.....	11-9
11.5	CVS and Subversion (SVN) Tools.....	11-9
11.5.1	Comparison between CVS and Subversion (SVN) Tools	11-10
✓	Syllabus Topic : Case Study : Visual Source Safe from Microsoft and Clear Case	11-11
11.6	Visual Source Safe from Microsoft and Clear Case ..	11-11

Chapter 12 : Software Maintenance and Reengineering**12-1 to 12-9**

✓	Syllabus Topic : Software Maintenance	12-1
12.1	Software Maintenance.....	12-1
12.1.1	Modifiability	12-1
12.1.2	Types of Maintenance	12-2
12.1.2.1	Corrective Maintenance.....	12-2
12.1.2.2	Adaptive Maintenance	12-2

12.1.2.3	Perfective Maintenance	12-2
12.1.2.4	Preventive Maintenance	12-3
12.1.3	Need of Maintenance.....	12-3
✓	Syllabus Topic : Software Supportability	12-3
12.2	Software Supportability	12-3
✓	Syllabus Topic : Reengineering	12-3
12.3	Reengineering.....	12-3
12.3.1	Re-Engineering Process Model	12-4
✓	Syllabus Topic : Business Process Reengineering ...	12-5
12.4	Business Process Reengineering	12-5
✓	Syllabus Topic : Software Reengineering.....	12-6
12.5	Software Reengineering	12-6
✓	Syllabus Topic : Reverse Engineering.....	12-7
12.6	Reverse Engineering.....	12-7
12.6.1	Abstraction Level.....	12-7
12.6.2	Completeness	12-7
12.6.3	Directionality.....	12-7
✓	Syllabus Topic : Restructuring.....	12-8
12.7	Restructuring.....	12-8
✓	Syllabus Topic : Forward Engineering.....	12-9
12.8	Forward Engineering.....	12-9

UNIT VI

Syllabus : Introduction to Software Testing, Principles of Testing, Testing Life Cycle, Phases of Testing, Types of Testing, Verification and Validation, Defect Management, Defect Life Cycle, Bug Reporting, GUI Testing, Test Management and Automation.

Chapter 13 : Software Testing**13-1 to 13-37**

✓	Syllabus Topic : Introduction to Software Testing	13-1
13.1	Introduction to Software Testing (SPPU - Dec. 13, Dec. 14)	13-1
13.2	Software Testing Fundamentals	13-2
13.2.1	Test Characteristics (Attributes of good test).....	13-3
✓	Syllabus Topic : Principles of Testing	13-3
13.3	Principles of Testing (SPPU - Dec. 15).....	13-3
✓	Syllabus Topic : Testing Life Cycles	13-4
13.4	Testing Life Cycles.....	13-4
13.4.1	Requirement Analysis	13-4
13.4.2	Test Planning	13-5
13.4.3	Test Case Development	13-5
13.4.4	Test Execution	13-5
13.4.5	Test Cycle Closure	13-5
✓	Syllabus Topic : Phases of Testing	13-5
13.5	Phases of Testing	13-5
13.5.1	Unit Testing (SPPU - Dec. 12).....	13-5
13.5.2	Integration Testing (SPPU – Dec. 13, Dec. 14, Dec. 15, May 16, Dec. 16)	13-6
13.5.3	Functional Testing	13-8
13.5.4	System Testing	13-9
13.5.5	Recovery Testing (SPPU – Dec. 13)	13-9



13.5.6 Security Testing	13-9	13.10.4 Smoke Testing (SPPU – May 12, May 13, May 14, Dec. 16)	13-25
13.5.7 Stress Testing	13-10	13.10.5 Comments on Integration Testing.....	13-25
13.5.8 Performance Testing (SPPU – Dec. 13, Dec. 14).....	13-10	13.10.6 Integration Test Documentation.....	13-25
13.5.9 Verification and Validation	13-10	13.10.7 Difference between Regression and Smoke Testing (SPPU – May 15)	13-25
13.5.10 Regression Testing (SPPU – May 13)	13-10	13.11 Test strategies for Object-Oriented software	13-26
13.5.11 User acceptance Testing (SPPU – May 13, May 14, Dec. 14)	13-11	13.11.1 Unit Testing in the OO Context.....	13-26
13.5.12 Alpha and Beta Testing (SPPU – Dec. 16)	13-11	13.11.2 Integration Testing in the OO Context	13-26
✓ Syllabus Topic : Types of Testing	13-12	13.12 Test strategies for WebApps.....	13-27
13.6 Types of Testing	13-12	✓ Syllabus Topic : Defect Management	13-27
13.7 White-Box Testing (SPPU – Dec. 16)	13-12	13.13 Defect Management.....	13-27
13.7.1 Basis Path Testing (SPPU – May 12, Dec. 12, May 14, Dec. 14, May 15, Dec. 16)	13-12	13.13.1 Defect Management Process.....	13-27
13.7.1.1 Flow Graph Notation.....	13-13	13.13.2 Defect Removal Efficiency	13-28
13.7.1.2 Independent Program Paths.....	13-14	✓ Syllabus Topic : Defect Life Cycle	13-28
13.7.1.3 Deriving Test Cases	13-15	13.14 Defect Life Cycle	13-28
13.7.1.4 Graph Matrices (SPPU – May 13).....	13-15	✓ Syllabus Topic : Bug Reporting	13-29
13.7.2 Control Structure Testing (SPPU - May 16)	13-16	13.15 Bug Reporting (The art of debugging)	13-29
13.7.2.1 Condition Testing	13-16	13.15.1 The Debugging Process (SPPU – May 13, Dec. 13) 13-30	
13.7.2.2 Data Flow Testing.....	13-16	13.15.2 Psychological Considerations	13-31
13.7.2.3 Loop Testing (SPPU – May 13).....	13-16	13.15.3 Debugging Approaches	13-31
13.8 Black-Box Testing.....	13-17	✓ Syllabus Topic : GUI Testing	13-33
13.8.1 Graph-Based Testing Method.....	13-18	13.16 GUI Testing	13-33
13.8.2 Equivalence Partitioning	13-18	13.16.1 Interface Testing Strategy	13-33
13.8.3 Boundary Value Analysis (SPPU – Dec. 16).....	13-19	13.16.2 Testing Interface Strategy Mechanisms.....	13-33
13.8.4 Orthogonal Array Testing (SPPU – Dec. 16).....	13-19	13.16.3 Usability Tests.....	13-33
13.8.5 Differentiation between White-box and Black-box Testing (SPPU – May 12, Dec. 12, May 15)	13-20	13.16.4 Compatibility Tests	13-34
✓ Syllabus Topic : Verification and Validation	13-21	13.16.5 Test Plan	13-34
13.9 Verification and Validation (SPPU – May 14).....	13-21	13.16.6 Positive Testing	13-35
13.9.1 Difference between Verification and Validation (SPPU – May 12, Dec. 13, Dec. 15, May 16)	13-21	13.16.7 Negative Testing	13-35
13.10 Test Strategies for Conventional Software	13-22	✓ Syllabus Topic : Test Management and Automation	13-35
13.10.1 Unit Testing (SPPU – May 12)	13-22	13.17 Test Management and Automation.....	13-35
13.10.2 Integration Testing	13-23	13.17.1 Motivation	13-36
13.10.3 Regression Testing (SPPU – Dec. 16).....	13-25	13.17.2 Test Driven Development (TDD).....	13-36

Software Engineering Fundamentals

Syllabus

Software Engineering Fundamentals: Nature of Software, Software Engineering Principles, The Software Process, Software Myths.

Syllabus Topic : Software Engineering Fundamentals (Introduction)

1.1 Introduction

Review Questions

- Q. Why Computer Software is important ?
- Q. Explain the term Computer Software in brief.

- Computer software has become an integral part of our daily lives. It helps in all sorts of businesses and decision makings in business. The application of computer software includes : Telecommunication, transportation, military, medical sciences, online shopping, entertainment industry, office products, education industry, construction business, IT industry, banking sector and many more.
- The software applications have a great impact on our social life and cultural life as well. Due to its widespread use, it is the need of time to develop technologies to produce high quality, user friendly, efficient and economical software.
- Computer software is actually a product developed by software engineers by making use of various software engineering processes and activities.

- Software consists of data, programs and the related documents. All these elements build a configuration that is created as a part of the software engineering process. The main motive behind software engineering is to give a framework for building software with better quality.
- We can say that the software has become the key element in all computer based systems and products.

Syllabus Topic : Nature of Software

1.2 Nature of Software

Review Questions

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

- The nature of software has great impact on software engineering systems. The general nature of software may describe the important characteristic :
- **Reliability :** If we consider the use of software in air traffic control system and space shuttle, then there should not be any chances of failure of software.



- In these examples, if reliability is not taken into consideration, then the human life is at risk.
- In addition to this, there are four other important characteristics that describe the nature of software :

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

1.2.1 Absence of Fundamental Theory

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

1.2.2 Ease of Change

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

1.2.3 Rapid Evolution of Technologies

- In the modern age, the software development technologies and development environments are changing rapidly with an extreme speed.

- It becomes the need of the time to keep the software engineers updated and armed with latest technologies and skills.
- The software engineering standards must also be revised with the technology evolutions.

1.2.4 Low Manufacturing Cost

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

1.3 Software Engineering : A Layered Technology

SPPU - May 13, May 14, Dec. 14

University Questions

- Q. Explain the layered approach to software engineering. (May 2013, 4 Marks)
- Q. Define Software Engineering. What are the various categories of software? (May 2014, 4 Marks)
- Q. Define Software Engineering. (Dec. 2014, 6 Marks)

Review Questions

- Q. Define the term Software Engineering.
- Q. Define Software Engineering as Layered Technology.
- Q. How Software Characteristics are different from Hardware Characteristics ?
- Q. Define the terms Software Crisis.
- Q. Define the terms Legacy Software.
- Q. Define software engineering. What are the characteristics of software which make its Engineering different from Industrial Manufacturing ?

Q. What are the problems associated with development process ?
 (Hint : Refer Software Crisis)

- The term software engineering is defined as :
- "By using the principles of sound engineering and its establishment, software is developed that should be economical and should work efficiently on real machines."
- Software engineering is a systematic and disciplined approach towards developments of software. The approach must be systematic for operation of the software and the maintenance of it too.
- Software engineering is considered as a layered technology. These layers includes :

1. Quality focus
2. Process
3. Methods
4. Tools

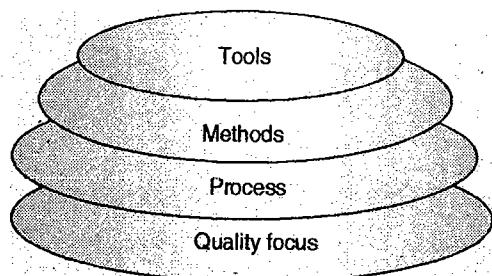


Fig. 1.3.1 : Software engineering layers

1.3.1 Quality Focus

Quality is nothing but 'degree of goodness'. Software quality cannot be measured directly. Good quality software has following characteristics :

1. **Correctness** is degree to which software performs its required function.
2. **Maintainability** is an ease with which software is maintained. Maintenance of software implies change in software. If software is easy to maintain, then the quality of that software is supposed to be good.
3. **Integrity** is a security provided so that unauthorized user can not access data or information. e.g. password authentication.

4. **Usability** is the efforts required to use or operate the software.

1.3.2 Process

Software process defines a framework that includes different activities and tasks. In short, process defines following 'what' activities and tasks for software development :

1. What activities are to be carried out ?
2. What actions will be taken ?
3. What tasks are to be carried out in a given action ?

1.3.3 Methods

Method provides technical way to implement the software i.e. 'how to' implement. Methods consist of collection of tasks that include :

1. **Communication** : Between customer and developer.
2. **Requirement analysis** : To state requirements in detail.
3. **Analysis and design modelling** : To build a prototyping model of software to exhibit the requirements clearly.
4. **Program construction** : Implementation of requirements using conventional programming languages or automated tools.
5. **Testing and support** : Test for errors and expected results.

1.3.4 Tools

Software tool is an automated support for software development.

For example

1. Microsoft front page or Microsoft Publisher can be used as web designing tool.

2. Rational Rose can be used as object oriented analysis and design tool.

1.3.5 The Characteristics of Software

SPPU - May 14, Dec. 16, Apr. 17

University Question

Q. What are the software characteristics?
(May 2014, Dec. 2016, Apr. 2017, 4 Marks)

- Software is having some characteristics, those are totally different from hardware characteristics or the industrial manufacturing
- Software is developed or engineered and it is not manufactured like other hardware products.
- There exist some similarities between development of software and manufacturing of hardware
- In both the cases quality is achieved by good design but manufacturing phase of hardware can introduce quality problems that are absent in software.
- Both activities depend on people but relationship between people applied and work done is different in both the cases.
- Both the cases require the 'construction of product', but approaches are different.

Software does not wear out.

- o Note that, wear out means process of losing the material.
- o Hardware components are affected by dust, temperature and other environmental maladies. Stated simply, hardware can wear out. However software does not influenced by such environmental means.
- o When hardware component wear out, it can be replaced by spare part. But there are no spare parts for software. Any software error indicates error in design or coding of that software.

- o Hence software maintenance involves more complexity than hardware maintenance.
- o Mostly software is custom built rather than assembled from existing components.
- o Computer hardware can be assembled from existing components as per our requirements. But that is not the case for software. Software is developed according to customer's need.

1.3.6 Software Crisis

SPPU - May 13

University Question

Q. Explain the term : Software crisis.
(May 2013, 2 Marks)

- Most of software engineers refer the problems associated with software development as "Software crisis".
- The causes of software crisis are linked to all the problems and complexities associated with development process. Following are some most encountered problems associated with development process :
 1. Running out of time i.e. deadline crossed
 2. Over budget
 3. Software inefficient
 4. Low quality
 5. Not up to expectations of customers
 6. No enough development team

1.3.7 Legacy Software

SPPU - May 13

University Question

Q. Explain the term : Legacy software.
(May 2013, 2 Marks)

- The term legacy software refers to older software developments that were poorly designed and documented. It had supported



- for many years. The legacy systems may or may not remain in use.
- Even if it is no longer used, it may continue to impact the organization due to its historical role. Historic data may not have been converted into the new system format and may exist within the new system.

Syllabus Topic : Software Engineering Principles

1.4 Software Engineering Principles

Review Questions

- Q. What are seven Software Engineering Principles ? Explain each in detail.
- Q. Why continuous validation is required ?
- Q. Why it is required to maintain Disciplined Product Control ?

- There are seven Software Engineering Principles that have been determined which are actually supposed to be the independent and complete set.
- Following are the list of seven Software Engineering Principles :
 1. Manage the software using the phrased life cycle.
 2. During the development process, perform continuous validation.
 3. Maintain a disciplined control on the flow of product development.
 4. In development process, use all the innovative and modern programming practices.
 5. Maintain all the accountability of results.
 6. Use the best team.
 7. Always maintain a commitment to improve the overall process.

Principle 1 : Manage the software using the phrased life cycle

- Planned and controlled software projects are conducted for one and only reason

- because it is the only way known to manage complexity.
- Project failure rate remains higher than it should be even though the success rate for software projects has improved. One of the reasons is that the customers lose interest quickly (because what they have requested was not really as important as they first thought), and the projects are cancelled.
- The software engineers and the software project manager must follow certain guidelines for project plans in order to avoid project failure :
 1. Carefully design a set of common warning signs.
 2. Understand the critical success factors that lead to good project management, and
 3. Develop a common sense approach for planning, monitoring, and controlling the project.
- A description of the products to be developed in each phase, and of the associated development activities and schedules. Major products of each phase should be identified as discrete milestones, in such a way that there can be no ambiguity about whether or not a milestone has been achieved.
- The component activities in each phase should be identified in some detail.
- Following are the six phases in every Software development life cycle model:
 1. Requirement gathering and analysis
 2. Design
 3. Implementation or coding
 4. Testing
 5. Deployment
 6. Maintenance

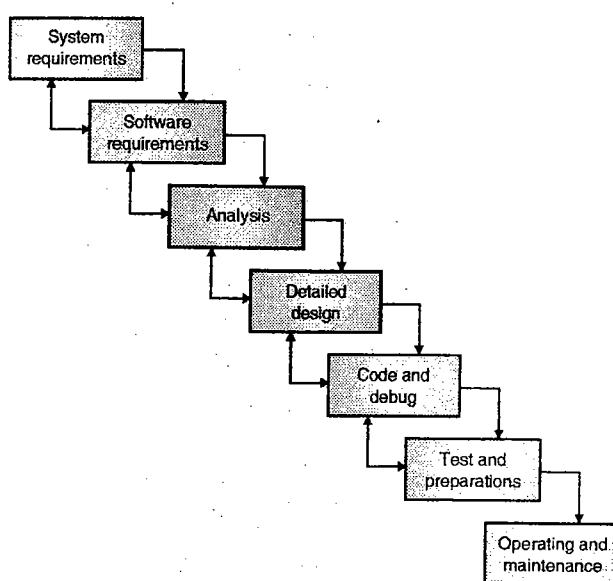


Fig. 1.4.1 : Phases in Software development life cycle

Principle 2 : Continuous validation

- One of the most common mistakes (that proves costly later in software projects today) is to defer the activity of detecting and correcting software problems until late in the project, i.e. in the "test and validation" phase after the code has been developed.
- There are two main reasons why this is a mistake:
 1. Most of the errors have already been made before coding begins and
 2. The later an error is detected and corrected, it is a expensive affair.
- Thus there must be continuous validation throughout the development process.

Principle 3 : Maintain disciplined product control

- In fact, any good-sized project must accommodate changes in requirements throughout the development cycle.
- Some changes may be due to external environment like:
 1. New government reporting regulations
 2. Improvements in technology
 3. User organizational changes or changes

in the overall system like bank, refinery and retail store etc. where this software product is used.

- All these changes have good impact on the system development. Thus it becomes very easy that different versions of the documentation and the code to be reproduced frequently.
- The tester or the user observes that the system is different from the system that was proposed actually.
- Thus, it is necessary to maintain a disciplined product control activity throughout the system life-cycle to avoid such mismatches.

Principle 4 : Use of latest programming practices

- The use of modern programming practices like top-down structured programming or object oriented programming practices help to deal with more visibility into software development process.
- Enhancement in modern practices has good features and functionalities.

Principle 5 : Maintain accountability for results

- Each team member in the project team should have a clear understanding of the results for which he or his group are accountable, and a clear understanding that his future rewards depend on how well he does in producing those results.
- For a project to be able to do this, one additional thing is needed and that is adequate visibility into each team member's individual performance.

Principle 6 : Use the best team

- Even if all the principles from 1 to 5 are observed carefully, still there are chances of project failure.
- This failure can be caused due to incompetent individual team member.



- Thus it is very important to recruit a staff of a good repute and a good caliber.

Principle 7 : Always maintain a commitment to improve the overall process

- All the above principles are good enough to produce a good software product. But still it's a good practice to improve the regular processes and practices.
- It is significant in terms of the need for planning and understanding your organization.

Syllabus Topic : The Software Process

1.5 The Software Process

SPPU – May 12, May 15

University Questions

- Q. What is a software engineering process ?
(May 2012, 3 Marks)
- Q. What is Software process framework? Explain in detail.
(May 2015, 7 Marks)

Review Questions

- Q. What are various Umbrella Activities associated in the Development Process ?
- Q. Describe Common Process Framework with the help of diagram.

- A software process can be illustrated by the following Fig. 1.5.1. In the Fig. 1.5.1, a common process framework is exhibited. This framework is established by dividing overall framework activities into small number of framework activities.

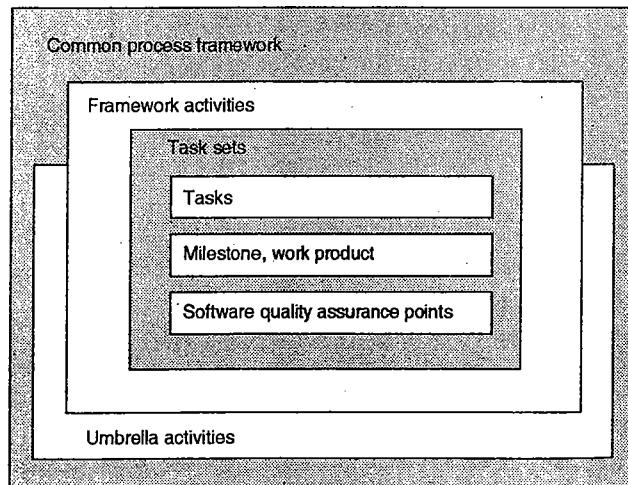


Fig. 1.5.1 : A software process

- And these small activities are applicable to the entire project irrespective of its size and complexity. A framework is a collection of task sets and these task sets consists of :

- o Collection of small work tasks
- o Project milestones, deliverable i.e. actual work product and
- o Software quality assurance points

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

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

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

1.5.1 Umbrella Activities

SPPU – Feb. 15, May 16

University Question

- Q. What are various umbrella activities applied throughout a software project?

(Feb. 2015, May 2016, 7 Marks)

The framework described in generic view of software engineering (Fig. 1.5.1 : A software process) is complemented by number of Umbrella activities. Typical **umbrella activities** are :

1. Software project tracking and control

- o Developing team has to assess project plan and compare with predefined schedule.



- o If project plan doesn't match with predefined schedule, necessary actions are taken to maintain the schedule.

2. Risk management

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

3. Software Quality Assurance (SQA)

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

4. Formal Technical Reviews (FTR)

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

5. Measurement

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

6. Software Configuration Management (SCM)

It manages the effects of change throughout the software process.

7. Reusability management

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

8. Work product preparation and production

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

Syllabus Topic : Software Myths

1.6 Software Myths

SPPU - May 12, May 13, Dec. 13, Dec. 14, Feb. 15

University Questions

- Q. State a software myth each which customers, developers and project manager believe. What is the reality in each case ?
(May 2012, 5 Marks)
- Q. Explain the term : Software myth.
(May 2013, 2 Marks)
- Q. What do you mean by software myths?
(Dec. 2013, 3 Marks)
- Q. What is software myth ? Give an example.
(Dec. 2014, 4 Marks)
- Q. What are different software myths that the customers and practitioners believe in and what are their corresponding realities ?
(Feb. 2015, 4 Marks)

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



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

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

1.6.1 Management Level Myths (or Manager Level Myths) SPPU - May 14

University Question

Q. What are the management myths?
(May 2014, 4 Marks)

➤ Myth

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

➤ Reality

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

➤ Myth

We can buy software tools and use them.

➤ Reality

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

➤ Myth

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

➤ Reality

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

➤ Myth

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

➤ Reality

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

➤ Myth

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

➤ Reality

The rule book exists. But the questions arise :

- o Is it actually used by the developers ?
- o Are software developers aware of this type of book of standards and procedures ?
- o Does it contain all the modern engineering practices ?
- o Is it foolproof ?
- o Does it focus on quality ?

- o Does it streamlined to timely delivery ?
- o In most of the cases, the simple answer to all these queries is a big "NO".

➤ Myth

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

➤ Reality

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

1.6.2 Customer Level Myths SPPU – May 14

University Question

Q. State and explain customer's myth.
(May 2014, 4 Marks)

➤ Myth

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

➤ Reality

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

➤ Myth

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

➤ Reality

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

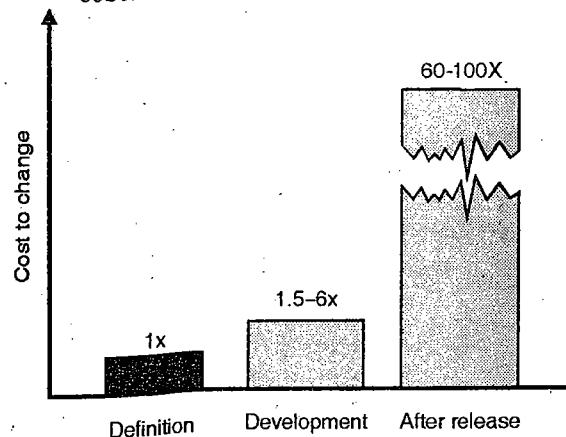


Fig. 1.6.1: The impact of change

1.6.3 Practitioner Level Myths (or Developer Level Myths)

SPPU – Dec. 12, Dec. 13, Feb. 16, Apr. 17

University Questions

- Q. State the myths of the practitioner and the realities. (Dec. 2012, 6 Marks)
- Q. State the myths and facts for software developer. (Dec. 2013, 3 Marks)
- Q. List and explain practitioner's myths and what are their corresponding realities? (Feb. 2016, 4 Marks)
- Q. Briefly explain the myths and reality associated with practitioner and management in software project. (Apr. 2017, 5 Marks)

**➤ Myth**

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

➤ Reality

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

➤ Myth

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

➤ Reality

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

➤ Myth

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

➤ Reality

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

➤ Myth

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

➤ Reality

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

1.7 Importance of Software Engineering

SPPU - Dec. 14

University Question

Q. Explain the importance of Software Engineering. (Dec. 2014, 4 Marks)

- Software engineering is the study and application of engineering to the design, development and maintenance of software that needed in all the aspects of our daily lives.
- In every field of work, specific software is needed. Since software is developed and embedded in machines so that it could meet the requirement of the user. This is possible because of software engineering.
- Using software engineering concepts reduces the complexity in developing the software application.
- It also reduces the software cost.
- Development time is reduced considerably.



Process Models

Syllabus

Process Models : A Generic Process Model, Prescriptive Process Models: The Waterfall, Incremental Process(RAD), Evolutionary Process, Unified Process, Concurrent.

Syllabus Topic : A Generic Process Model

2.1 A Generic Process Model (or Generic Process Framework)

SPPU - Dec. 12

University Question

Q. State the generic process framework activities.
(Dec. 2012, 4 Marks)

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

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

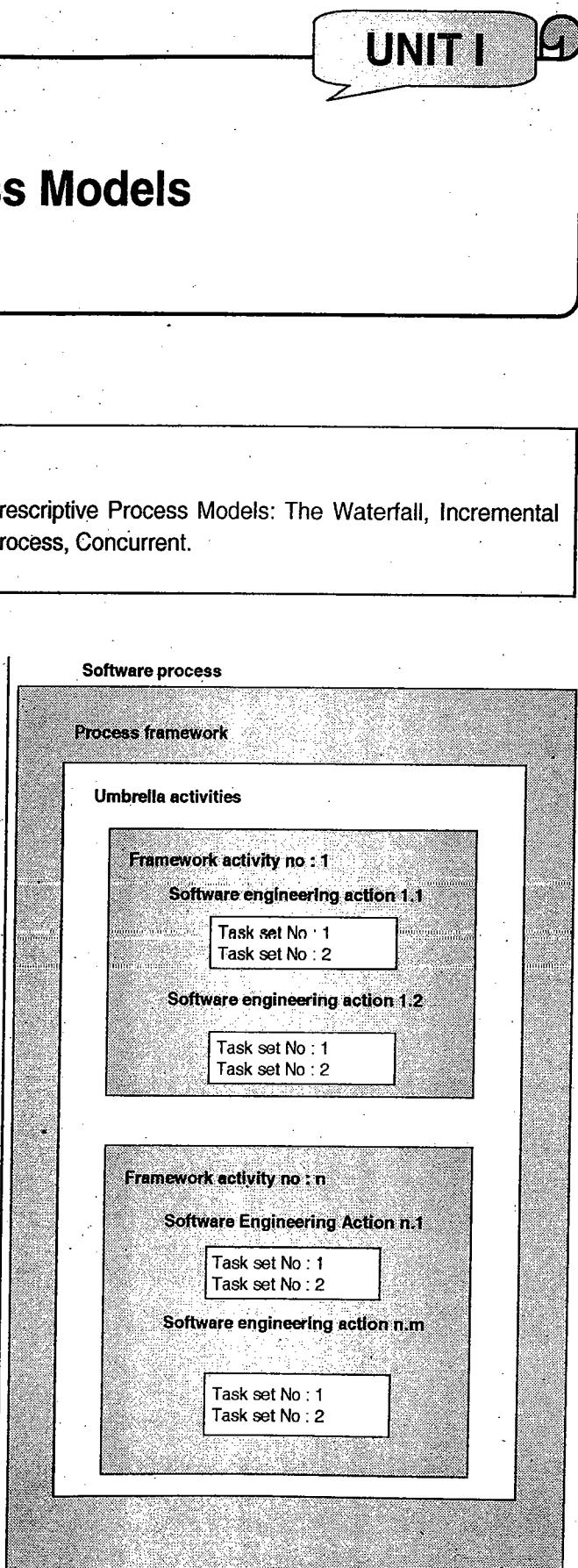


Fig. 2.1.1 : A software process framework

2.1.1 Communication

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

2.1.2 Planning

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

2.1.3 Modeling

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

2.1.4 Construction

- It includes coding and testing steps :
- (i) **Coding** : Design details are implemented using appropriate programming language.
 - (ii) **Testing** : Testing is carried out for the following reasons :
 - o To check whether the flow of coding is correct or not.
 - o To check out the errors of program e.g. in C program just by pressing F7 key we check step by step execution of program or by using "Add-Watch" we add the variables and watch the values of variables.
 - o To check whether program is giving expected output as per input specifications.

2.1.5 Deployment

- It includes software delivery, support and feedback from customer.

- If customer suggest some corrections, or demands additional capabilities, then changes are required for such corrections or enhancement.
- These five generic framework activities can be used for :
 - o Development of small programs
 - o Creation of large programs
 - o Development of large system based programs
- In addition to these five generic framework activities, various umbrella activities are also associated throughout the development process.

Syllabus Topic : Prescriptive Process Models

2.2 Prescriptive Process Models

- All the software organizations describe a unique set of framework activities for their own development processes. In general, it should adopt all the generic framework activities and the set of tasks defined in Section 1.5 i.e. generic process model. Whatever process model is chosen by the organization, but it should encompass the following framework activities :
 - 1. Communication 2. Planning
 - 3. Modeling 4. Construction
 - 5. Deployment
- The name 'prescriptive' is given since the model prescribes set of activities, actions, tasks, quality assurance and change control mechanism for every project. Each of the prescriptive models also prescribes a workflow.
- Workflow is defined as the flow of process elements and the manner in which they are interrelated. All the generic framework activities are defined earlier, but each of the prescriptive models put different emphasis to these generic framework activities and give different workflow.

- In the following section, we describe some of the prescriptive process models :

1. The waterfall model
2. Incremental process models
3. Evolutionary process models
4. The specialized process models

Syllabus Topic : The Waterfall Model

2.2.1 The Waterfall Model

SPPU – Dec. 12, Dec. 13, Dec. 14

University Questions

- Q. Explain waterfall model with its advantages and disadvantages. (Dec. 2012, 6 Marks)
- Q. Explain the waterfall model with V model. (Dec. 2013, 6 Marks)
- Q. What are the disadvantages of the waterfall model ? (Dec. 2014, 2 Marks)

Review Question

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

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

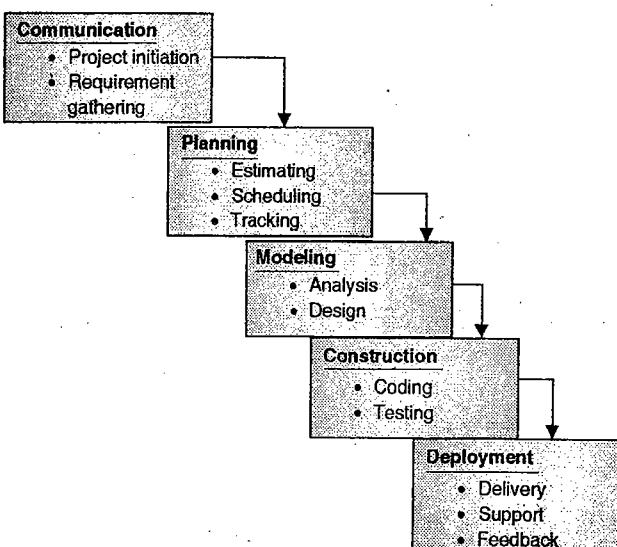


Fig. 2.2.1 : The Waterfall model

- An alternative design for 'Linear Sequential Model' is as shown in Fig. 2.2.2.

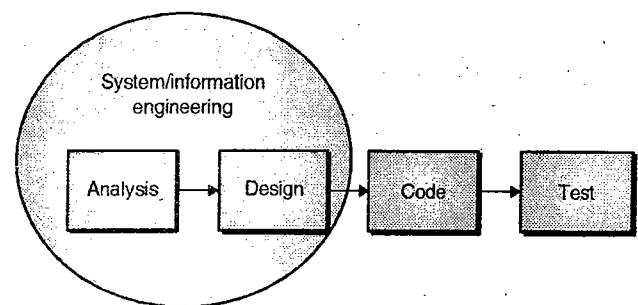


Fig. 2.2.2 : The linear sequential model

1. Communication

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

2. Planning

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

3. Modeling

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

4. Construction

It includes coding and testing steps :

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

5. Deployment

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

Merits / Advantages

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

Demerits / Disadvantages

1. Problems in this model remain uncovered until software testing.
2. One of the disadvantages of this approach is 'blocking states'. In blocking state, the members of development team waits for other members to complete the dependent tasks. Due to this, huge amount of time spent in waiting rather than spending time on some productive work. In today's world, the software work is fast paced and thus waterfall model is not useful.
3. According to this model customer must state all his requirements at the beginning stage of development, which is difficult for the customer.

4. This model is step-by-step systematic sequential approach. Ultimately, customer gets the working version of software too late. Hence customer requires patience.
5. This approach is actually not realistic and does not match with real projects.
6. Also it does not incorporate the risk assessment.
7. Finally it is useful for smaller projects only where requirements are well understood in the beginning only.

2.2.1.1 V-Model (Software Development)

SPPU - Dec. 13

University Question

Q. Explain the waterfall model with V model.
(Dec. 2013, 6 Marks)

- In software engineering development process, the V-model represents a development process that may be considered as an extension of the waterfall model.
- In more general V-model, instead of moving down in a linear way, the process steps are bent upwards after the coding phase as shown in Fig. 2.2.3.

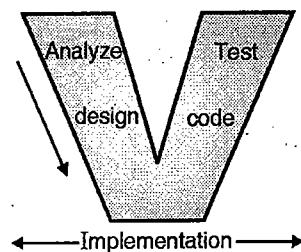


Fig. 2.2.3 : V-model

- The V-model demonstrates the relationships between each phase of the development life cycle.

Syllabus Topic : Incremental Process Models

2.2.2 Incremental Process Models

- Using these models a limited set of customer's requirements are implemented quickly and are delivered to customer.
- Then modified and expanded requirements are implemented step by step.
- Actually in this approach, the overall functionalities are split to smaller modules and these modules are quickly developed and delivered. Then refinements are possible in later increments or versions.
- In this approach, the initial requirements are very well defined. Each increment is passed through the overall development cycle i.e. phases of classic life cycle discussed earlier.
- The customer's feedback is very important after each of the increments (or releases) and next increment is developed. This process continues till the product is finished i.e. all the requirements are satisfied.
- Following are the two types of incremental process models :
 1. The incremental model 2. The RAD model

Syllabus Topic : The Incremental Model

2.2.2.1 The Incremental Model

SPPU - May 14, Dec. 14

University Question

- Q. Explain the incremental model with advantages and disadvantages ?

(May 2014, Dec. 2014, 6 Marks)

- The incremental model combines the elements of waterfall model applied in an interactive fashion. The first increment is generally a core product.
- Each increment produces the product, which is submitted to the customer. The customer suggests some modifications.

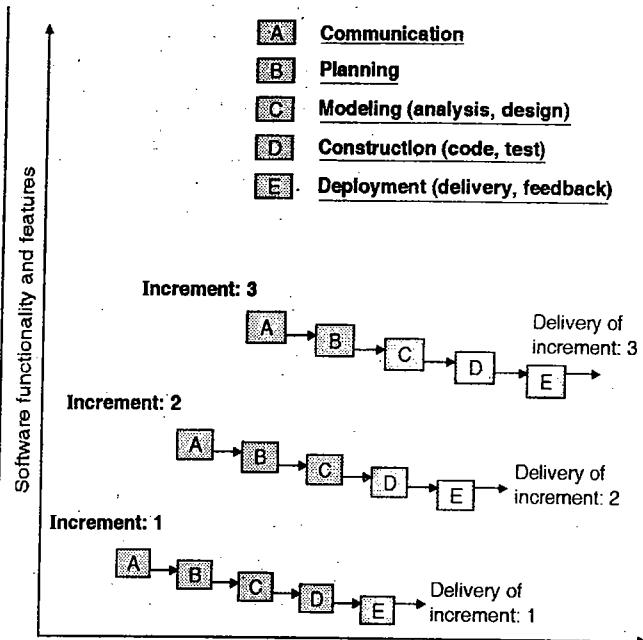


Fig. 2.2.4 : Project calendar time

- The next increment implements customer's suggestions and some additional requirements in previous increment. The process is repeated until the product is finished.
- Consider the development of word processing software (like MS Word or Word Star) using incremental model.

Increment – 1

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

Increment – 2

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

Increment – 3

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

Increment – 4

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

Merits / Advantages

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

Demerits / Disadvantages

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

Syllabus Topic : The RAD Model

2.2.2.2 The RAD Model

Review Question

Q. Explain the RAD model with advantages and disadvantages?

- RAD (Rapid Action Development) model is high-speed adaptation of waterfall model. Using RAD model, software product can be developed within a very short period of time i.e. almost within 60 to 90 days.
- The initial activity starts with communication between customer and developer. Depending upon initial requirements the project planning is done.

Now the requirements are divided into different groups and each requirement group is assigned to different teams.

- Like other approaches, in RAD approach also all the generic framework activities are carried out. These generic framework activities are already discussed earlier.
 - When all teams are ready with their final products, the product of each team is integrated i.e. combined to form a product as a whole.
 - In RAD model each team carries out the following steps :
1. **Communication :** It understands the business problem and information for software.
 2. **Planning :** Since various teams work together on different modules simultaneously, planning is important part of development process.
 3. **Modeling :** It includes :

- (i) **Business Modeling :** It includes information flow among different functions in the project e.g., what information will be produced by each function, which functions handles that information etc.
- (ii) **Data Modeling :** It includes different data objects used in software and relationship among different objects.
- (iii) **Process Modeling :** During process modeling, process descriptions are created. e.g. add, modify, delete etc.

4. **Construction :** It includes :

(i) Component reuse

Reusable components means the software components i.e. modules or procedures that are developed for specific application can be reused in development of other application.

(ii) Automatic code generation

The software producing the codes after providing proper inputs or

selecting readymade options, e.g. Microsoft FrontPage used in Web development, Rational Rose used for Object Oriented analysis and designing.

(iii) Testing

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

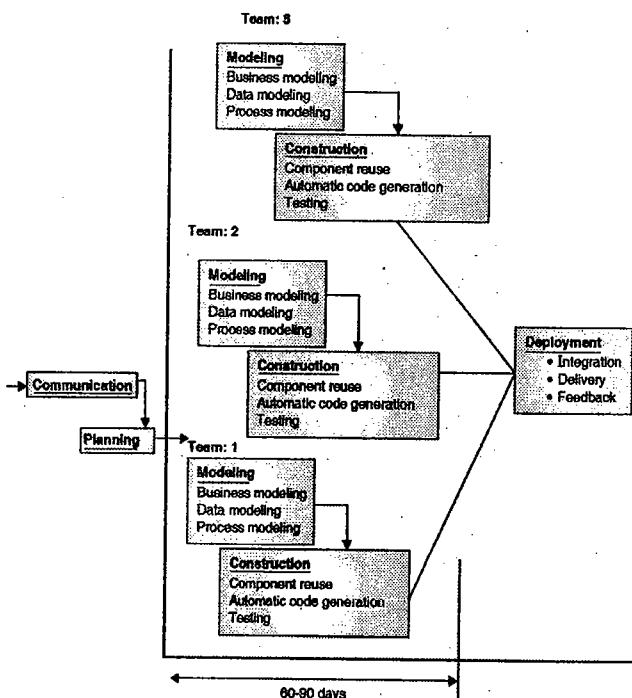


Fig. 2.2.5 : The RAD Model

Merits / Advantages

1. Using the RAD approach, a product can be developed within very short period of time.
2. It supports increased reusability of the components.
3. It results in minimal code writing as it supports automatic code generation.
4. It encourages the customer feedback.
5. In this approach, quick initial reviews are possible.

6. In this approach, modules integration is done from the beginning thus resolving number of integration issues.

Demerits / Disadvantages

1. It can be used, if sufficient number of staff is available. Thus, it requires sufficient man power to create number of teams.
2. In RAD approach, many teams work parallel to implement different functions of same project. Hence all teams must work with equal speed. If one team lags behind the schedule, overall project delivery will be late.
3. If system can not be modularized properly, then building the components for RAD model will be problematic.
4. The RAD model is not appropriate when technical risks are high. (e.g. a new application makes heavy use of new technology).
5. This approach is useful for only larger projects.
6. The RAD model had two primary disadvantages as follows :
 - o **Reduced scalability** : It occurs because a RAD application evolves from a prototype to a finished application. Thus there is less scope for scalability.
 - o **Reduced features** : It occurs due to time boxing. Time boxing is a time management technique in which the task is to be completed in a given frame of time called time boxes. Thus the features are pushed to be completed to the later releases due to short amount of time.

Syllabus Topic : Evolutionary Process Models

2.2.3 Evolutionary Process Models

SPPU - May 12

University Question

- Q. Explain evolutionary process models mentioning the types of projects for which they are suitable. (May 2012, 6 Marks)

Review Question

Q. What do you mean by evolutionary process models ? Explain spiral model as an evolutionary process model ?

- Evolutionary process models are iterative type models. Using these models the developer can develop increasingly more complete versions of the software.
- As the requirements change very often, the end product may be unrealistic and a complete version of the product is not possible. To overcome this drawback, the concept of limited version product is introduced and this version is gradually completed over the period of time.
- Each version is refined based on the feedbacks till it is completed.
- Following are the examples of evolutionary process models :

1. The Prototyping paradigm
2. The Spiral model
3. The Concurrent development model

2.2.3.1 The Prototyping Paradigm

SPPU - Dec. 13

University Question

Q. Explain the prototyping model with its advantages and disadvantages.

(Dec. 2013, 6 Marks)

Review Questions

- Q. What do you mean by working and throwaway prototypes? Explain how they are used in prototyping model ?
- Q. Explain advantages and disadvantages of prototyping model ?

- The **prototyping paradigm** offers best approach for human machine interaction. Ideally prototype serves as a mechanism for identifying software requirements.
- The prototyping approach is often called as **throw-away prototyping**. By using this approach, there is a rough demonstration of the customer's requirements i.e. the

prototype used in demonstration is just a throwaway prototype.

If working prototype is built, developer can use software tools if required (e.g. report generators)

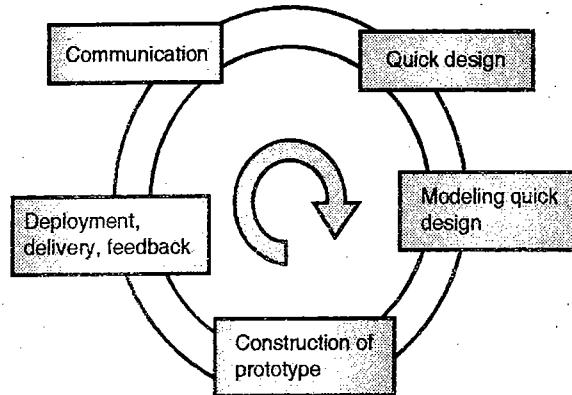


Fig. 2.2.6 : The prototyping model

This produces working program quickly. Thus prototype can be served as 'first system'.

Different phases of prototyping model are :

1. Communication

- o The software prototyping paradigm starts with the communication between the developer and the customers.
- o The software engineer and the customer meet regularly to define the overall objectives of the desired software and to identify its requirements.

2. Quick design

- o Quick design focuses on those aspects of software that are visible to the customer.
- o It includes clear input, output formats and human machine interfaces.

3. Modeling quick design

The model of software is now built that gives clear idea of the software to be developed. This enables the developer to better understand the exact requirements.

4. Construction of prototype

The concept of modeling the quick design

leads to the development of prototype. This construction of prototype is deployed by the customer and it is evaluated by the customer itself.

5. Deployment, Delivery, Feedback

- o As picture of software to be built is very clear, customer can give his suggestions as a feedback.
- o If result is satisfactory, the model is implemented otherwise process is repeated to satisfy customers all requirements.

Merit / Advantage

Prototyping makes requirements more clear and system more transparent.

Demerits / Disadvantages

- The software developer generally compromises in the quality to get the working prototype quickly.
- Due to this reason, sometimes inappropriate operating system or programming language may be selected. He may use and implement inefficient algorithm.

2.2.3.2 The Spiral Model

SPPU - Dec. 13

University Question

- Q. State the activities of spiral model.
(Dec. 2013, 4 Marks)

Review Question

- Q. What are the advantages and disadvantages of spiral model ?

- The **Spiral model** is combination of well known waterfall model and iterative prototyping. It yields rapid development of more complete version of software.
- Using spiral model software is developed as series of evolutionary releases. During the initial releases, it may be just paperwork or prototype. But during later releases the version goes towards more completed stage.

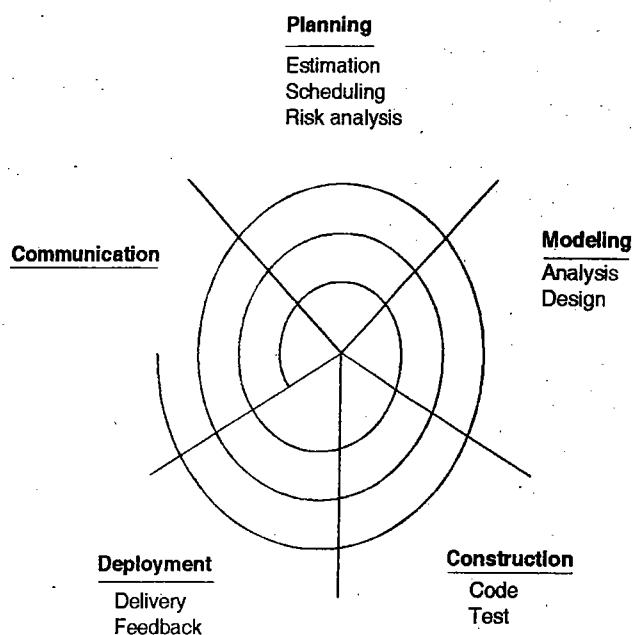


Fig. 2.2.7 : The Spiral Model

- The spiral model can be adopted to apply throughout the entire lifecycle of the application from concept development to maintenance. The spiral model is divided into set of framework activities defined by software engineer team. Each framework activity represents one segment of spiral as shown in Fig. 2.2.7.
- The initial activity is shown from centre of circle and developed in clockwise direction. Each spiral of the model includes following steps :

1. Communication

The software development process starts with communication between customer and developer.

2. Planning

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

3. Modelling

- o It includes detail requirement analysis and project design (algorithm, flowchart etc).

- o Flowchart shows complete pictorial flow of program whereas algorithm is step by step solution of problem.

4. Construction

It includes coding and testing steps :

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

- (ii) **Testing** : Testing is carried out.

5. Deployment

- o It includes software delivery, support and feedback from customer. If customer suggest some corrections, or demands additional capabilities then changes are required for such corrections or enhancement.
- o Note that after customer evaluation, next spiral implements, 'customer's suggestions' plus 'enhancement plan'. Thus, each of iteration around the spiral leads to more completed version of software.

Merits / Advantages

1. In this approach, project monitoring is very easy and more effective compared to other models.
2. It reduces the number of risk in software development before they become serious problems.
3. It is suitable for very high risk projects.
4. Project estimates i.e. schedule and cost is more realistic.
5. Risk management is in-built feature of spiral model.
6. Changes can be accommodated in the later stages of development.

Demerits / Disadvantages

1. If major risk is not discovered in early iteration of spiral, it may become a major risk in later stages.

2. Each iteration around the spiral leads to more completed version of software. But it's difficult to convince (especially in contract situation) to the customer that the model is controllable.
3. Cost of this approach is usually high.
4. It is not suitable for low risk projects.
5. Rules and protocols must be followed very strictly to implement the approach.

Syllabus Topic : Concurrent Model

2.2.3.3 The Concurrent Development Model

Review Question

- Q. Explain concurrent development model with advantages and disadvantages ?

- The **concurrent development model** is also called as **concurrent model**. This model is more appropriate for system engineering projects where different engineering teams are involved. In system engineering stage, the project requirements are considered at system level.
- Diagrammatically it can be represented as a series of framework activities, task, actions and their associated states.
- Fig. 2.2.8 exhibits one element of the concurrent process model :

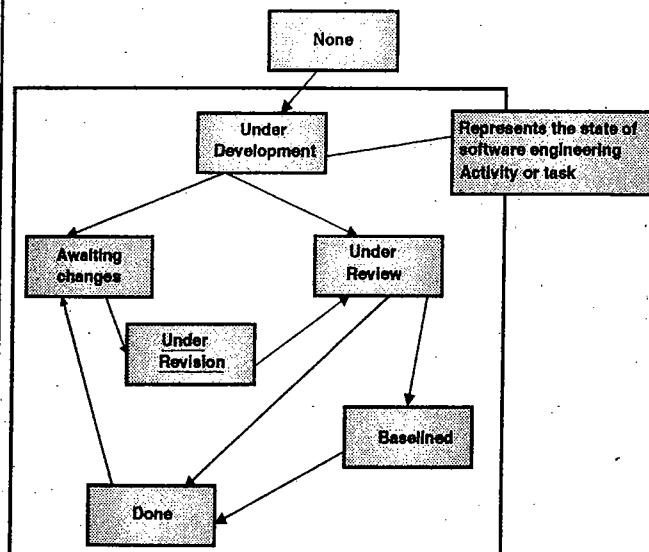


Fig. 2.2.8 : One element of concurrent process model (i.e. Modeling activity)



- The **modelling activity** is one of the states and other activities like **communication or construction** can also be represented in an analogous manner.
- Let the **communication activity** has completed its first iteration and available in **awaiting changes state**. The **modelling activity** has completed its initial communication and ready to move to **under development state** from none state.
- During these transitions, if customer indicates some modification in the requirement, then the **modelling activity** transits to **awaiting changes state** from **under development state**.
- Instead of considering activities, actions and tasks as sequence of events, it defines network of activities.
- The concurrent process model activities transits from one state to another state and this model is used in all types of software development and it provides very clear idea of the current state of the project.

Merits / Advantages

1. The concurrent development model is applicable to all types of software development processes.
2. It gives very clear picture of current state of the project.
3. It is easy to use and easy to understand.
4. This approach is flexible and number of releases determined by the development team.
5. New functionalities as elicited by the client can be accommodated late in the project.
6. It has immediate feedback from testing.

Demerits / Disadvantages

1. Since all the stages in the concurrent development model works concurrently, any change in the requirement from the client may halt the progress.

This may happen due to dependent components among different stages and it lead to more longer development cycles as compared the planned cycles.

2. It requires excellent and updated communication between the team members. This may not be achieved all the time.
3. The SRS must be updated at regular intervals to reflect the changes.

2.2.3.4 Differentiation between Prescriptive and Evolutionary Process Models

Review Question

Q. Differentiate between prescriptive and evolutionary process models.

Sr. No.	Prescriptive process models	Evolutionary process models
1.	Developed to bring order and structure to the software development process.	Evolutionary software processes do not establish the maximum speed of the evolution. Due to this development process becomes slow.
2.	Defines a distinct set of activities, actions, tasks, milestones, and work products that are required to engineer high-quality software	Evolutionary process models lack flexibility, extensibility, and high quality.
3.	It is more popular.	It is less popular.
4.	It provides complete and full developed systems.	Time does not allow a full and complete system to be developed.
5.	Example : Water fall model, Incremental models.	Example : Prototyping, Spiral and Concurrent models.

2.2.4 The Specialized Process Models

- Specialized process models have many characteristics of one or more of the conventional models described in earlier section.

- Specialized models are applied when a narrowly defined engineering approach is chosen.
- Following are some specialized process models :

1. Component-based development models
2. The formal methods model
3. Aspect-oriented software development

2.2.4.1 Component-Based Development Models SPPU - Dec. 13, Dec. 14

University Questions

- Q. Explain the component development model with activities. (Dec. 2013, 6 Marks)
- Q. Explain how component based development reduces the time of software development ? (Dec. 2014, 6 Marks)

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

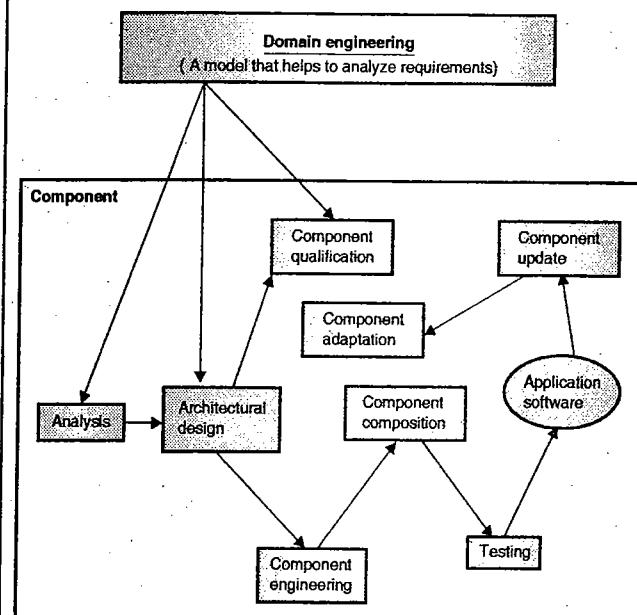


Fig. 2.2.9 : A component-based development model

- Component based development includes following steps :
 1. Available components-based products are researched and evaluated for application domain.
 2. Component integration issues are also considered.
 3. Software architecture is designed to accommodate the components.
 4. Components are integrated i.e. combined into architecture.
 5. Testing is carried out to check the functionality.
- Important feature of Component Based Development is that, it leads to 'software reuse'. Reusable components means the components that are developed for specific application can be reused in development of other application projects also. Reusability provides many benefits for software development.

Merits / Advantages

1. The component based development model supports reusability.
2. Due to reusability, there is reduction in development cycle time.

3. Ultimately the cost of overall development reduces substantially.
4. There is significant increase in productivity.

Demerits / Disadvantages

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

2.2.4.2 The Formal Methods Model

- The formal methods model includes set of activities that leads to formal mathematical specification of computer software.
- Using formal methods, a software engineer can specify, develop and verify computer based system by applying mathematical notations.

Following are some examples of formal methods model :

A Symbol table

- A program is used to maintain a symbol table. Such tables can be used in many different types of applications. For example, a table that includes a collection of items without any duplication.
- A typical example is names of users of a system without duplicate entries. If such table is examined during execution of the system any time, it will never show duplicate names for use 'Raina'.

1. Virat
2. Dhoni
3. Shikhar
4. Yuvraj
5. Raina

(An example symbol table of users used in formal methods model)

Merits / Advantages

1. When formal methods are used, many problems get eliminated those are difficult to overcome using other software engineering paradigms. For example Ambiguity, incompleteness and inconsistency can be discovered and corrected more easily using mathematical analysis.
2. The formal methods used in the design process provide the basis for program verification. They also enable the software developer to uncover the undetected errors and then fix them.
3. Using this method it's possible to develop accurate and defect free software.

Demerits / Disadvantages

1. This model is just time consuming and more expensive with compared to other methods.
2. As formal method models requires that software developers must have knowledge about formal methods. Otherwise training about formal methods must be given to developers.
3. It is difficult to use the model as a communication mechanism for technically unsophisticated customers.

2.2.4.3 Aspect-Oriented Software Development

- Irrespective of the software process that is chosen, the builder of complex software implements a set of localized features, functions and information content.

- These localized software characteristics are models as components (e.g. Object Oriented Classes) and then constructed within the context of system architecture.
- As modern computer based systems become more sophisticated, certain "concerns" (i.e. customer required properties of technical interest-span the entire architecture). Some concerns are high-level properties of the system (e.g. security). Other concern affects functions (e.g. application of business rules), while others are systematic (e.g. task synchronization or memory management).
- In most of the large systems the relationship between different program components and their requirements are complex. One single requirement can be implemented by various components and vice versa that means one single component may be involved in various requirements.
- To address this particular problem and make programs easier and maintain good reusability, Aspect Oriented Software Engineering (AOSE) is used.
- AOSE implements system functionality at various places in the program. An important characteristic of AOSE is that it includes a definition of an aspect for its use in a proper place in a program.
- The important benefit of this approach is that it supports the separation of concerns into independent elements. For example, user authentication can be represented as an aspect that requests a login name and a password. This login name and password can be used in a program wherever authentication is required.
- The aspect oriented process is likely to adopt characteristics of spiral models and concurrent process models. The spiral model's evolutionary nature is best the aspects since in these models, aspects are

identified and after that they are constructed.

- The concurrent development model has parallel nature and it is necessary, because aspects are developed independently by using local software components. Still aspects have a direct impact on these software components.

2.2.4.4 Need of Process Models

Workflow is defined as the flow of process elements and the manner in which they are interrelated. All the generic framework activities are defined earlier, but each of the prescriptive models put different emphasis to these generic framework activities and gives different workflow. This is the reason why different process models are required in software development. Each model has its own importance and significance.

Syllabus Topic : The Unified Process

2.3 The Unified Process

SPPU - Dec 16

University Question

- Q. Explain in detail the Unified process indicating workflows and process phases. What are the advantages of iterative development ?
(Dec. 2016, 5 Marks)

- The object oriented methods and programming languages are widely used in software development. Hence Object Oriented Analysis (OOA) and Object Oriented Design (OOD) methods are proposed for Object Oriented software development.
- Object Oriented Process Model similar to that of evolutionary models are used as new paradigm for software development using object oriented Languages.
- During 1990 Rumbaugh, Booch and Jacobson combined, presented a Unified Modeling Language (UML) that includes

- notations for modeling and development of OO systems. By 1997 UML became industry standard for Object Oriented software development and Rational corporation developed automated tools to support UML methods (i.e. Rational Rose software).
- UML provides the necessary technology to support object oriented software engineering practice, but doesn't provide the process framework to guide project teams in their application of the technology. Hence, Rumbaugh, Booch and Jacobson developed the Unified Process, a framework for object oriented software engineering using UML.

2.3.1 The Phases of Unified Process

SPPU – Dec. 12

University Question

Q. Explain the phases of unified process model.

(Dec. 2012, 6 Marks)

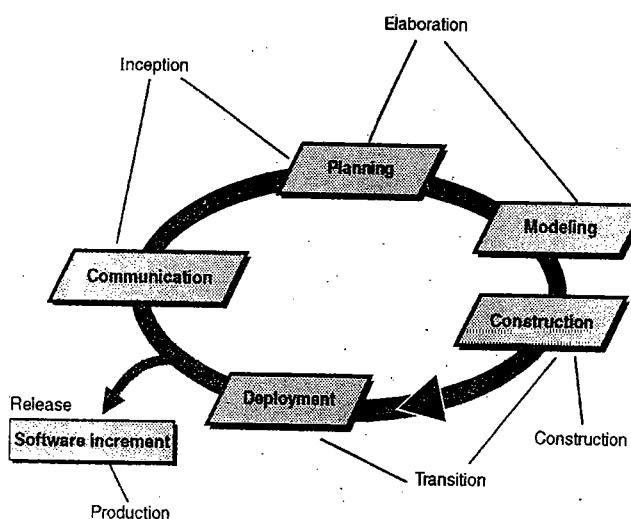


Fig. 2.3.1 : The unified process

The unified process includes following phases :

1. Inception phase

- o It includes customer communication and planning activities.

- o Here with customer communication requirements are identified and architecture for the system is planned.
- o To understand the requirements better, the Use case diagram is developed.
- o The Use case diagram shows the relationship between actors (e.g. person, machine, another system, etc.) and use cases (sequence of actions i.e. procedures /functions). Fig. 2.3.2 shows a use case diagram for sales management system.

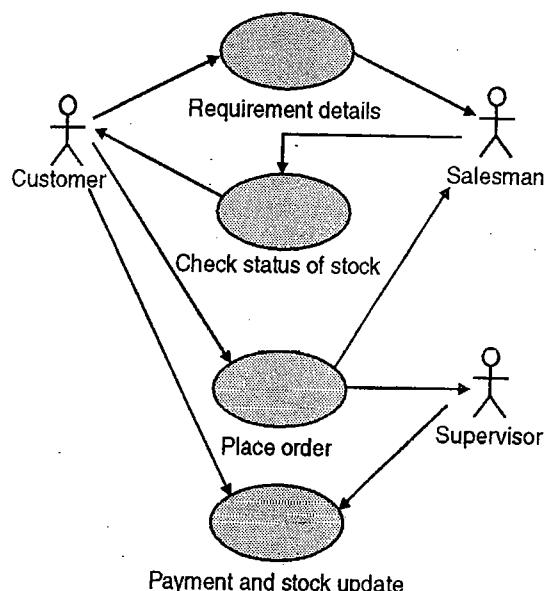


Fig. 2.3.2 : Use Case diagram for Sales Management System

2. Elaboration phase

It includes customer communication and modeling activities of generic process model. Elaboration refines and expands the use cases developed in inception phase and expands architectural representation to include five different views of software.

- Use case model
- Analysis model
- Design model
- Implementation model
- Deployment model

3. Construction phase

- This phase is equivalent to construction activity for generic software process.
- Using the architectural model as input the construction phase develops software components that will make use of each use case diagram.
- To acquire this, analysis and design models that were started during elaboration phase are completed to reflect the final version of software increment.

4. Transition phase

- During this step the software is given to end user for beta testing.
- Now user can report the defects and necessary changes.
- Developing team also creates all necessary information regarding developed software.
- For example user manuals, troubleshooting guides (Guidelines if some trouble occur during handling of the software), and installation procedures. As a conclusion of this phase software is now ready to release.

5. Production phase

It is equivalent to deployment activity of generic process.

During this phase

- Ongoing use of software is monitored.

- Support for operating environment is provided.
- Defect reports and requests for changes are submitted and evaluated.
- During the time of construction, transition and production phases, preparation of next increment is started.

2.3.2 The Iteration among Four Phases

- A phase is said to be the period of time between two processes. As shown in Fig. 2.3.1, a software development life cycle consists of mainly four phases :

- Inception
- Elaboration
- Construction and
- Transition

Between all these four phases, one important element is present i.e. iteration. Iteration actually separates all these phases. Iteration consists of different set of activities in the project plan and the criteria for evaluation necessary for future releases.

This software development life cycle is revolves around all these four phases and software development activities can be divided into these four phases. This development process is highly dependent on continuous stream of releases. This is the reason why all the four phases of the unified process supports incremental and iterative development.

Advanced Process Models and Tools

Syllabus

Agile software development: Agile methods, Plan-driven and agile development, Extreme programming Practices, Testing in XP, Pair programming. Introduction to agile tools: JIRA, Kanban, Case Studies: An information system (mental health-care system), wilderness weather system.

3.1 Agile Process Model

Review Question

Q. Explain Agile process model. What are the different agility principles?

- An agile process model includes the concept of development along with a set of guidelines necessary for the development process. The conceptual design is necessary for the satisfaction of the customer. The concept is also necessary for the incremental delivery of the product. The concept or the philosophy makes development task simple. The agility team must be highly motivated, updated with latest skill sets and should focus on development simplicity.
- We can say that agile development is an alternative approach to the conventional development in certain projects.
- The development guidelines emphasize on analysis and design activities and continuous communication between developers and customers. An agile team quickly responds to changes. The changes may be in development, changes in the team members, and changes due to new

technology. The agility can be applied to any software process. It emphasizes rapid delivery of operational software.

- All the agile software processes should address three important assumptions :
 - o Difficult to predict in advance software requirements
 - o Design and construction are interleaved in most of the projects, it is difficult to predict design before construction.
 - o Analysis, design, construction and testing are not much predictable.
- To address these assumptions of unpredictability, the agile development process must be adaptable. So an agile process must adapt incrementally.

3.1.1 Comparison between the Agile and Evolutionary Process Models

SPPU - May 13, Feb. 16

University Questions

- Q. Compare the agile and evolutionary process models with a specific process model for each. (May 2013, 8 Marks)
- Q. Compare the process between the Agile and evolutionary model. (Feb. 2016, 4 Marks)



Sr. No.	Agile process model	Evolutionary process model
1.	These models satisfy customer through early and continuous delivery.	Using these models the developer can develop increasingly more complete versions of the software.
2.	It can accommodate changing requirements.	It yields rapid development of more complete version of software rather accommodating continuous changing requirements.
3.	In this development process customer, business people and developers must work together daily.	Daily meetings are not required. After one evolution of design, customer, business people and developers can meet and discuss.
4.	The messages are conveyed orally i.e. face-to-face. Conversation is essential.	Oral messages are not so essential.
5.	Technical excellence and good design is achieved.	The developer often compromises in order to get working prototype quickly. Thus inappropriate operating system or programming language may be used simply because it's available and known. Thus inefficient algorithm may be implemented.

Syllabus Topic : Agile Software Development

3.2 Agile Software Development

Review Question

Q. Explain agile development. What are the key factors that are considered for agile development ?

- In today's modern world, businesses spread across the globe in each sphere of life. It has become the global phenomenon. Due to this changing environment, the client must respond to new opportunities and to the volatile market conditions, and to the arrival of new products and services.
- The software application is actually the heart of all the business operations. So according to the rapid changing rules and business ideas, it should respond quickly.

- Thus the developer is supposed to develop the new software quickly to meet the desired requirements in the stipulated time.
- In fact, in such rapidly changing market condition, rapid development of the software and its urgent delivery is the need of the time and it is supposed to be the most critical requirement of any software system.
- Normally most of the clients are even ready to compromise the quality of software and the requirements at the cost of faster development of the software and its quick delivery.
- Since all the businesses usually operate in the rapidly changing environment, it becomes highly impossible to gather complete set of requirements. The requirements elicited by the customers in the beginning are always changed since the customers find it difficult to predict the actual working of the software and the challenges to come across.
- Thus understanding the requirements quickly and accommodate the requirements change narrated by the customer quickly is the need of the time. Otherwise, delayed delivery of the software may make it unusable and out of date.
- Thus waiting for the complete requirements gathering from the customer will not work for the rapid development. Since the requirement of the customer changes with the progress of the software development process.
- Due to this reason, the conventional approaches like waterfall model or specification based processes will delay the final delivery of the software system.
- But in some cases like critical control system, the complete analysis of the requirement is mandatory else it may cost

- life. For such type of safety systems, plan-driven approach is always the best choice.
- Rapid software development processes are the ultimate choice for the development of useful products in quick time span. In this approach the software is developed as a series of increments.
 - Agile methods are incremental development methods in which the increments are usually small the new versions of the system are created rapidly and handed over to the customers within the span of 15 to 20 days and their feedback is received for the next versions.

Syllabus Topic : Agile Methods

3.2.1 Agile methods

- In the early days of software development, developer used to plan the project carefully, focus on quality assurance and employ the analysis and design methods supported by various CASE tools.
- This was the general practice of the software developer community that was responsible for the development of larger projects such as government projects etc.
- This larger projects use larger team and the team members may spread geographically across the world and they worked on the software for the longer period of time.
- Therefore, the development period may extend up to the 10 years from the initial specification to the final delivery of the product.
- Such a planed-driven approach will involve significant overheads in all the stages of the development processes like requirement specification, planning, designing, coding and documentation.
- In order to overcome this heavy weight plan-driven approach, the concept of agile methods was proposed. This methodology mainly focused on the product instead of

focusing on the design and documentation part.

- All the agile methods trust on the incremental approach instead of the conventional waterfall approach. The incremental approach is the best approach where the customer requirements and the software requirements change rapidly or frequently.
- The philosophy behind agile methods is also observed in **Agile Manifesto** that is accepted by most of the leading software developers. The Agile Manifesto is discussed in the following section.

3.2.2 Agile Manifesto

- The Agile Manifesto states that :

“We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value,

 - o **Individuals and interactions** work over processes and tools
 - o **Working software** takes responsibility of comprehensive documentation
 - o **Customer collaboration** look after contract negotiation
 - o **Responding to change** after having a good plan.”
- Even though these agile methods are based on the concept of incremental development, quicker delivery and faster deployment, the agile manifesto gives different processes to achieve this.
- But most of the set of principles used by various experts that are based on agile manifesto are very much common.

3.2.3 Agility Principles

SPPU – Apr. 17

University Question

- Q. List the principles of Agility.

(April 2017, 3 Marks)



- Agile software development describes a set of principles for software development under which requirements and solutions evolve through the collaborative effort of self-organizing cross-functional teams.
- It advocates adaptive planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change.
- The agile alliance has given twelve agility principles as follows :
1. Satisfy customer through early and continuous delivery.
 2. Accommodate changing requirements.
 3. Deliver the working software frequently in shorter time span.
 4. The customer, business people and developers must work together on daily basis during entire development.
 5. Complete the task with motivated developers and facilitate healthy and friendly environment.
 6. Convey the message orally, face-to-face conversation.
 7. Working software is the primary measure of progress.
 8. Agile process promotes sustainable development.
 9. Achieve technical excellence and good design.
 10. There must be simplicity in development.
 11. The best architecture, requirements and design emerge from self-organizing teams.
- 12. Every team should think how to become more effective. It should review regularly and adjust its behaviour accordingly.
- Thus the principles of agility may be applied to any software process. To achieve agile development, the team must obey all the principles mentioned above and conduct proper planning.
- All the agile software processes should address three important assumptions :
- o Difficult to predict in advance software requirements
 - o Design and construction are interleaved in most of the projects, it is difficult to predict design before construction.
 - o Analysis, design, construction and testing are not much predictable.
- To address these assumptions of unpredictability, the agile development process must be adaptable. So an agile process must adapt incrementally.

Syllabus Topic : Plan-Driven and Agile Development

3.3 Plan-Driven and Agile Development

- In agile development approaches, the design and implementation are considered as the main activity in the software development process.
- In addition to that, they also use activities like requirement specification, testing, debugging and maintenance.
- But in a plan-driven approach, output associated with stage is also identified. The output obtained in one stage is used as the basis of planning for the later stages of the software process.

- Fig. 3.3.1 exhibits comparison between plan-driven and agile approaches to the requirement specification :

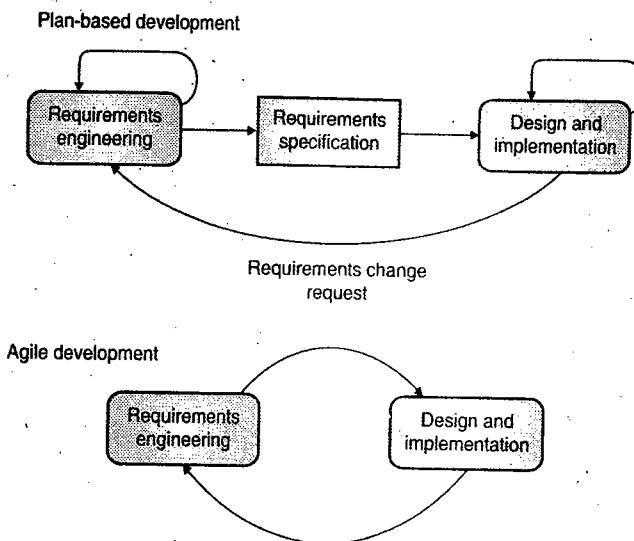


Fig. 3.3.1 : Distinction between plan-driven and agile approaches to the requirement specification

- In **plan-driven approach**, the iteration may occur within each of the activities and they are communicated with the help of formal documentation. After this formal communication, final “**Requirement Specification**” will be produced.
- In contrast, for an **agile approach** the iteration occurs across different activities. Observe the Fig. 3.3.1, the requirement specification and design are developed together to produce final product.
- As discussed earlier, the agile approach employs incremental development and delivery. Thus there is a series of incremental deliveries or we can say that series of versions for the software application under consideration.
- In general practice, so many software organizations have used agile methods and also use agile practices and they have integrated these with their plan-driven processes.

3.3.1 Comparison between Plan-driven and Agile Development

Sr. No.	Plan-driven development	Agile development
1.	In a plan-driven approach, output associated with stage is also identified. The output obtained in one stage is used as the basis of planning for the later stages of the software process.	In agile development approaches, the design and implementation are considered as the main activity in the software development process.
2.	In plan driven development, iteration occurs within each of the activities.	In agile development, iteration occurs across different activities.
3.	Communication between different activities done with the help of formal documentation and after this communication, final “Requirement Specification” will be produced	The requirement specification and design are developed together to produce final product.
4.	Plan-driven software development uses structure to control risk.	Agile software development uses flexibility to control risk.
5.	Plan-driven emphasizes formal communications and control.	Agile emphasizes continual informal communications and an ability to react to changes and uncertainty.
6.	It is less adaptive in nature.	It is more adaptive in nature.
7.	Examples are : Traditional project management, Unified Process.	Examples are : SCRUM, Extreme Programming

Syllabus Topic : Extreme Programming Practices

3.4 Extreme Programming Practices

SPPU -Dec. 15, Feb. 16

University Questions

Q. What is Extreme Programming ?

(Dec 2015, 7 Marks)

Q. What are the different key strategies used for Xtreme Programming approach to software development ?

(Feb. 2016, 6 Marks)

- The Extreme Programming is one of the most commonly used agile process models.

- All the agile process models obey the principles of agility and the manifesto of agile software development.
- The XP uses the concept of object oriented programming. This approach is preferred development paradigm.
- As in conventional approach, a developer focuses on the framework activities like planning, design, coding and testing, the XP also has set of rules and practices.
- Following Fig. 3.4.1 shows the Extreme Programming process and all the key XP activities are summarized :

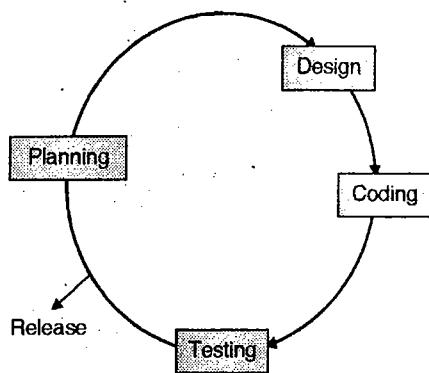


Fig. 3.4.1 : Extreme programming process

3.4.1 XP Values

- Following are a set of five values that establish a foundation for all work performed in context with XP.
 - Communication
 - Simplicity
 - Feedback
 - Courage
 - Respect
- For any development process, there must be a regular meeting of developer and the customer. There should be a proper communication for requirement gathering and discussion of the concepts.
- The simple design can always be easily implemented in code.
- The feedback is an important activity which leads to the discipline in the development process.

- In every development projects, there is always a pressure situation. The courage or the discipline will definitely make the task easy.
- In addition to all the XP values, the agile should inculcate respect among all the team members, between other stake holder and with customers.

3.4.2 The XP Process

SPPU - Dec. 14, Dec. 15

University Question

Q. Explain the Extreme Programming Process. (Dec 2014, Dec. 2015, 6 Marks)

- The XP process encompasses four framework activities :

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



3.4.3 Scrum

Review Questions

- Q. What is concept of SCRUM? Explain ?
- Q. Explain scrum process flow ?
- Q. What are different roles in scrum ?
- Q. Explain scrum cycle.
- Q. What is product backlog ?
- Q. Explain sprint planning meeting with diagram.
- Q. What do you mean by sprint backlog ?
- Q. Explain daily scrum meeting.
- Q. What do you mean by sprint review and retrospective?

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

- o **Product backlog** : List of priority wise requirement.
- o **Sprint backlog** : List of task to be completed which are prioritized.
- o **Sprint** : Period in which development occurs.
- o **Spike** : Period used to research concept.
- o **Tracer bullet** : It is spike with current architecture, technology, best practices, etc.
- o **Tasks** : Items added to sprint backlog at the beginning of sprint which are broken into hours.
- o **Definition of done** : Exit criteria which decides product backlog items are completed or not.
- o **Velocity** : Total efforts a team is capable of in a sprint.
- o **Scrum-but** : It is exception to scrum methodology.

3.4.3.1 Process Flow

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

➤ Setup

- o Setup environment contains the following :
- o Remove any customization from existing application.
- o Activate scrum process pack plug-in.
- o Assign scrum roles to users.

➤ Create a product

- o Product represents the functionality which is identified by owner.
- o Product contains different features which are needed for enhancement which are useful for customer satisfaction.

- It can have few features or many features.

➤ Create user stories

- These are product requirement created by product owner.
- User stories are written in plain language. Less technical jargons are used.
- It contains specific requirement that product should have.
- Stories cannot be created without associating with product.

➤ Create release

It has start and end date in which number of development iterations are completed.

➤ Create sprint

- It is basic unit of time in development process.
- It can have any length. Typically it takes one to four weeks to complete.
- Scrum master creates one or more sprints.
- Sprints should release in given start and end dates.
- Scrum team need to complete all the stories which are committed.
- Scrum master expects all the stories are fully implemented and ready to release.

➤ Plan sprint

- Team and scrum master need to decide on which stories they can commit to complete within a sprint.
- Scrum master make sure that capability of sprint team matches the requirement of stories.
- If capacity of the stories exceeds then scrum master add team members, remove stories or add sprint as and when needed.

- Velocity chart is used in estimation process.
- Velocity chart shows the historical record of number of completed points.
- With the help of velocity chart scrum master get the idea of capacity of team.
- Velocity chart is important tool in planning sprint.

➤ Track sprint progress

- Scrum master is responsible for checking the progress.
- He provides the progress report of the team.
- Team members frequently update task and records.

➤ Generate charts

- Progress of a sprint can be tracked with the help of different charts.
- Chart is one of the important tools of scrum team.
- Burn down chart compares ideal progress in sprint against the actual progress.
- It is done on daily basis.

3.4.4 Scrum Roles

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

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

1. Product owner

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

- The scrum process needs to be one product owner. Sometimes they are part of development team.

Role of the product owners in product requirement are as follows :

- Important role of product owner is communication with development team.
- Identifying important requirement and communicating is important task of product owner.
- They generally reduce the communication gap between team and stakeholders.
- They demonstrate the solution to stakeholders.
- They announce the different release of the product.
- They communicate the team status.
- Organizes the milestone review.
- They educate the stakeholders in development process.
- They negotiate priorities, scope, funding and schedule.

2. Development team

- They are responsible for delivering the product.
- Team generally consists of people of different skills.

- Team size can be from between 3 to 9.
- They includes member like analyst, designer, developer, tester, technical communicator, document writing, etc.
- Development team is self organizing.

3. Scrum master

- Scrum is facilitated by scrum master.
- He is accountable for product goals and deliverables.
- He is not a team lead or manager. He acts as a buffer between development team and distracting influences.
- He is the enforcer of the scrum process.
- He generally involves in key meeting and challenges the team to improve.
- Project manager is responsible for people management but scrum master is not related to people management.

3.4.5 Scrum Cycle Description

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

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

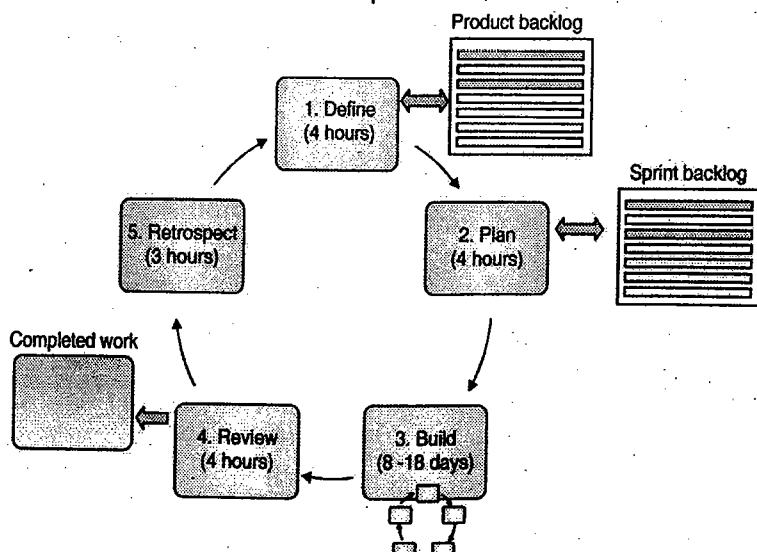


Fig. 3.4.2 : SCRUM cycle

- 1. Define :** During defining the requirement generally scrum team, scrum master and product owner sits together. They decide the priority of team for duration of sprint. It is useful in high level direction for the team.
 - 2. Plan :** It is about selecting items from product backlog. It also evaluates value of different items and estimate the efforts needed by scrum team. Generally items are selected from product backlog which can be considered as sprint backlog.
 - 3. Build :** Task from the sprint backlog is selected for development. They go on selecting the items till the items from the sprint backlog get completed.
 - 4. Review :** Review is presented by scrum team to product owner about the implementation of the items.
 - 5. Retrospect :** It is final steps of the retrospection. It has few objectives. It focuses on identifying the improvement area by the scrum team. They also discuss on the success of earlier iterations.
- Product backlogs are written in story format.
- It mainly contains what exactly need to be delivered.
- It also contains the order in which sequence the requirement should be delivered.
- Requirement listing and ordering that requirement is solely done by product owner.
- It contains assessment of business values by the product owner.
- Development team's assessment is also part of product backlog.
- Different methods are used by product owner to decide the priority of requirement which includes Fibonacci sequence and other methods.
- If some requirements are urgent then product owner can request to scrum team about prior requirement.
- Sizes of backlog items are determined by the development team.
- It does not contain only user stories, but it contains other information also.
- Product owner is responsible for maximizing the value of the product. He is also responsible for maximizing the work of development team.
- Product backlog is used for :
- o Taking request for product modification.
 - o It includes adding new facility, replace old facility, remove some unwanted features.
 - o Ensure delivery team is given work which maximizes the business benefits to the owner of the product.

3.4.6 Product Backlog

- It is ordered list of requirements which are maintained for product.
- It contains different details like features, bug fixes and non functional requirements.
- It contains whatever is needed for the success of the product.
- Items from the product backlog are ordered by product owner using different factors which includes date needed to complete the task, business values, risk, dependencies, etc.

3.4.7 Sprint Planning Meeting

- Product owner, development team and scrum master is involved in the sprint planning meeting.
- If required outside beneficiary can be invited for sprint planning meeting.
- In this meeting product owner describes the highest priority work and communicate to development team.
- Development team ask the question if any doubts, to the product owner.
- This meet is held at the beginning of sprint cycle.
- They generally select what work need to be done.
- They prepare sprint backlog which gives details of time taken to complete the work.
- They discuss how much work should be done during the current sprint cycle.
- Maximum time limit is 8 hours.
- Within 8 hours, in the first four hour entire team discusses about prioritizing the product backlog.
- In second 4 hours, development team plan for sprint which comes with sprint backlog.
- Sample agenda for sprint planning meeting is as follows :
 - o Product roadmap
 - o Name of the iterations
 - o Capacity of team
 - o Review
 - o Stories
 - o Commit
 - o Status of development
 - o Velocity of previous iterations
 - o Concerns
 - o Updates
 - o Tasking out

- Sprint planning meeting template is shown in Fig. 3.4.3.

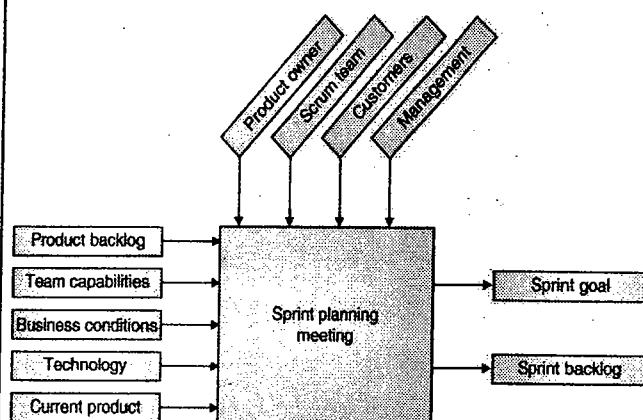


Fig. 3.4.3 : Sprint planning meeting

3.4.8 Sprint Backlog

- It is nothing but list of work development team need to complete in the sprint.
- This list is derived from product backlog.
- It contains list of user stories in sprint in the order of priority.
- It also contains the relative effort estimate.
- The task needed to develop every story is also maintained in the sprint backlog.
- In sprint planning meeting team selects some number of product backlog items.
- Product backlog items are selected in user story format.
- They also identifies user stories need to complete.
- Size of sprint backlog is selected by team.
- Sprint backlog can be maintained using spreadsheet.
- Example of sprint backlog is shown in Table 3.4.1.

Table 3.4.1

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

3.4.9 Sprint Execution

- Sprint is basic unit of development in scrum development.
- Scrum make progress is series of sprint.
- During the sprint every day project status meeting occurs. This meeting is called as daily scrum meeting.

- Sprint execution is nothing but the work which is performed by scrum team to meet sprint goals.
 - All the work which is necessary to deliver is performed.
 - It accounts for majority of time during a sprint.
 - It begins after sprint planning.
 - It ends when sprint review starts.
- 3.4.10 Daily Scrum Meeting**
- Every day during sprint project meeting occurs.
 - This meeting is called as daily scrum meeting.
 - It is helpful for schedule coming day's work.
 - Scrum meeting has following guidelines :
 - o All members presents for meeting with updates.
 - o Usually meeting starts on time even if some members are absent.
 - o Usual meeting time is at the morning.
 - o Meeting time and venue are same every day.
 - o Approximate meeting time is 15 minutes.
 - o Generally core team member speak in the meeting.
 - Different questions are answered by team. Some of the questions are as follows :
 - o What have been done since yesterday?
 - o What is today's planning?
 - o Any impediments or stumbling blocks?
 - Sample scrum meeting is shown in Table 3.4.2.

Table 3.4.2

Monday	Tuesday	Wednesday	Thursday	Friday
		Sprint 1 planning 1 pm - 5 pm.	Daily scrum 9:15 - 9:30	Day scrum 9:15 - 0:30
Daily scrum 9:15 - 9:30	Daily scrum 9:15 - 9:30 Backlog grooming 1 pm - 2 pm	Daily scrum 9:15 - 9:30	Daily scrum 9:15 - 9:30 Backlog grooming 1 pm - 2 pm	Daily scrum 9:15 - 9:30
Daily scrum 9:15 - 9:30	Daily scrum 9:15 - 9:30 Dry run and prep for sprint review (optional) 2 pm - 3 pm	Daily scrum 9:15 - 9:30 Sprint 1 review 10 am - 11 am Sprint 1 retrospective 11 am - 12 pm Celebration lunch 12 pm - 1 pm		

Monday	Tuesday	Wednesday	Thursday	Friday
		Sprint 2 planning 1 pm - 5 pm		

3.4.11 Maintaining Sprint Backlog and Burn-Down Chart

- Burn down is graphical representation of work left versus time.
- Pending work is on vertical axis and time is at horizontal axis.
- It is nothing but the run chart of outstanding work.
- It is useful in predicting when the work will be completed.
- It is used in agile software development particularly in scrum process.
- It can be applicable to any project.
- Sample burn down chart is shown in Fig. 3.4.4.

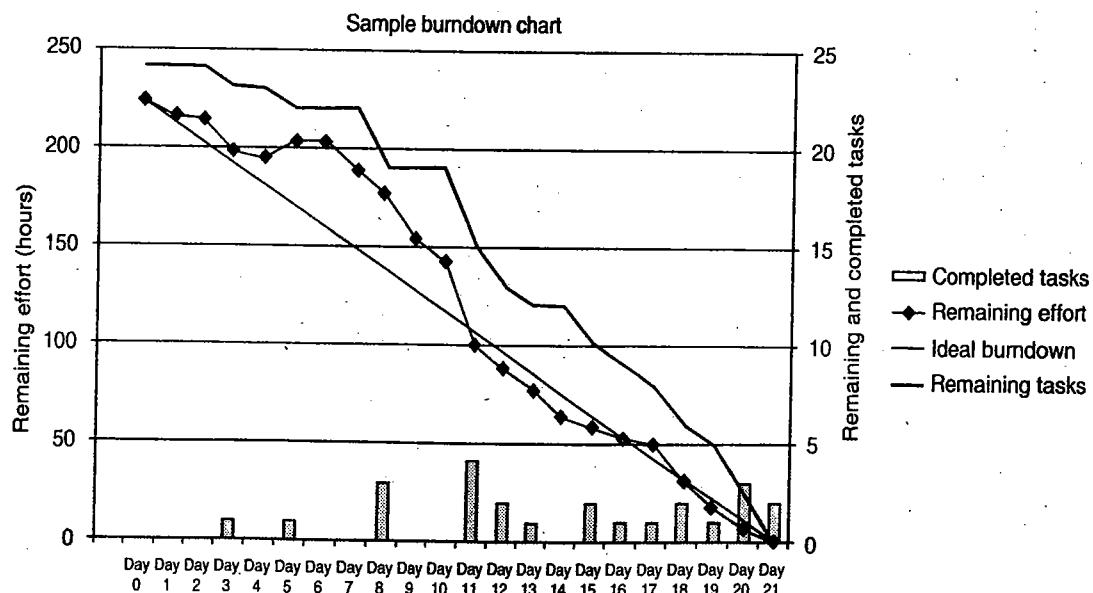


Fig. 3.4.4 : burn down chart

3.4.12 Sprint Review and Retrospective

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

Syllabus Topic : Testing in XP

3.5 Testing in XP

- Before the start of coding, the unit test is created in XP approach. This creation of unit test is the key element of XP approach.
- This unit test must be implemented by using some framework. This framework enables them to become automated.
- All the individual unit tests are organized into a 'universal testing suite'. Integration

and validation testing of the system can occur on daily basis.

- XP acceptances test, also called customer tests are specified by customer and focus on overall system features and functionality.

3.5.1 Exploratory Testing Versus Scripted Testing

Review Question

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

- Scripted testing considers two roles namely test designer and tester. Test designer is usually designs the test cases. He is high skilled specialist. Tester executes the test case which is designed by test designer.
- In exploratory testing different roles are not considered. Test designing and actual testing is done by same person. Tester designs the test cases and executes it. According to previous results he may create more test case and execute it.
- Learning, test design and execution all are done by tester in exploratory testing.
- In script testing team can be formed of different employees where one of the members would be very high skilled one and other members can beginners. Cost of testing can be saved with script testing.
- In exploratory testing depends upon skill of the employees which results in high project cost.
- With scripted testing high planning is possible. We can predict for desired outcome with scripted testing. As scenarios are ready we execute the test cases precisely.
- In exploratory testing planning is impossible or very difficult. We cannot guarantee all the test cases will be executed or not.

- With scripted testing well documentation can be created. Test designer can document the test scenarios whereas tester can records the status of executed test. With exploratory very less documentation is used.
- Exploratory testing is flexible whereas script testing is not.
- If some changes happen in the requirement then test designer need to change the scenarios. When scenarios are changed then only tester can executes the test cases.
- Exploratory testing is interesting than that of scripted testing.

3.6 Agile Practices

- Agile development is supported by number of practices. It includes the different area like requirement analysis, design, coding, testing, project management, process, quality, etc. Some of the important agile practices are as follows :
 - o Pair Programming
 - o Refactoring
 - o Test Driven Development
 - o Continuous Integration

Syllabus Topic : Pair Programming

3.6.1 Pair Programming

Review Questions

- Q. Explain Pair programming in the context of agile software development.
- Q. What are various benefits of pair programming ?
- Q. What do you mean by Remote pair programming ?

- It is one of agile practice is agile software development.
- In this method two programmer work together on one machine.

- One programmer types the code. Another programmer observes point and suggests the changes.
- Sometimes programmer can switch their role to each other.
- Observer programmer can suggest improvement in code.
- With this method programmer who writes the code can concentrate on efficient coding. Observer programming can think and suggest improvements.

Benefits of pair programming

There are numerous advantages of pair programming.

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



- o **Team building and communication:**
It allows to share the problem with each other and able to find the solution on it. With this practice communication among team member increase with benefits in greater team building.

Remote pair programming

- o It is part of pair programming.
- o It allows programmer to be geographically apart from each other but connected through the network.
- o It is also called as virtual pair programming or distributed pair programming.
- o They generally work using shared editors, shared desktop, remote pair programming tools.
- o Sometimes this method creates the problems which are not present in face to face programming.
- o Some of the problem in remote pair programming are delay in communication, communication link failure problem, lack of coordination, etc.
- o Different tools are provided for remote pair programming. Some of the tools are as follows :
 - o Microsoft Lync
 - o Screenhero
 - o Terminal multiplexers
(tmux a, screen -x)
 - o Saros
 - o Floobits
 - o Cloud9
 - o VNC
 - o Skype
 - o Gobby
 - o Xpirtise
 - o Visual studio anywhere

3.6.2 Refactoring

Review Questions

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

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

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

- With code maintainability, we can easily identify the bugs in the software as the code is properly documented.
- With code extensibility, new features can be added very easily.
- We need automatic unit test before applying code refactoring.

List of refactoring techniques

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

Techniques allowing more abstraction

- o Code encapsulation where different fields can be accessed with getter and setter methods.
- o Code sharing can be achieved by generalizing types. If we make generalize code then chances of sharing that code increases.
- o Replace type checking code with states.
- o Use polymorphism.

Techniques for breaking code apart into more logical pieces

- o Break the code into different modules. Modules increase the code reusability.
- o Extracting class results in moving code to new code.
- o Extracting methods results into moving methods to new methods.

Techniques for improving name and location of code

- o Move methods or field to appropriate class.

- o Change the name of methods or fields to user friendly name. Name should signify the purpose of method or variable.
- o Use concept of superclass from object oriented programming.
- o Use concept of subclass from object oriented programming.

Automated code refactoring

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

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

3.6.3 Test Driven Development

Review Question

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

- It is one of the agile practices in agile development.
 - This software development process relies on repetition of short development cycles.
 - Changes in software are quick and easy.
 - Short development cycle includes writing automated test case which defines desired improvement or new function. Minimum coding is done to pass the test. Afterwards code are redesigned as per the standard.
 - It is related to extreme programming.
- Test driven development is shown in Fig. 3.6.1.

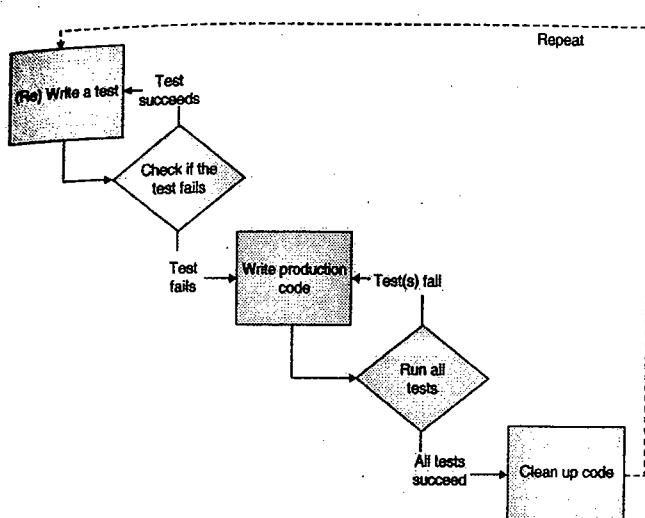


Fig. 3.6.1 : Test driven development

Different phases of test driven development are as follows.

(a) Create test

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

(b) Run test and see it fails

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

(c) Writing some code

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

(d) Run test

We run the test on created code.

(e) Refactor code

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

(f) Repeat

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

3.6.4 Continuous Integration

Review Questions

- | | |
|----|---|
| Q. | Explain Continuous integration in the context of agile software development. What are various principles of Continuous integration. |
| Q. | List different advantages of Continuous integration ? |
| Q. | List Software tools which supports continuous integration ? |

- It is method of merging all developer copies of work together frequently.
- It is part of extreme programming.
- Main aim is to prevent integration problems.
- It is useful in iterative software development.
- Earlier it was used in coordination with test driven development.

Principles of continuous integration

- **Maintain code repository :** All the project source code should be properly stored in code repository. It is useful in revision control system.

- **Automate the build :** There should be facility for building the integration automatically. Single command should be able to do all the activities. Different build tools like make can be used. Automated build includes integration.
- **Make build self testing :** After making the integration software should work as per specification. Testing facility after integration should be automated one.
- **Everyone commits to baseline every day :** With this facility every committer can reduce conflicting changes.
- **Every commit should be built :** System should build commits to current working version.
- **Keep the build fast :** Building process should be rapid.
- Other principles are as follows.

Test in clone of the production environment

Make it easy to get the latest deliverables

Everyone can see the results of the latest build

Automate deployment

Advantages

It has many advantages which are as follows.

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

- Creating modules can be possible with continuous integration.

Software tools which supports continuous integration

- o Apache Continuum
- o Apache Gump
- o AutomatedQA Automated Build Studio
- o Atlassian Software Systems Bamboo
- o Buildbot
- o BuildHive
- o CABIE
- o Cascade
- o CruiseControl
- o CruiseControl.NET
- o CruiseControl.NET
- o Electric Cloud
- o FinalBuilder Server
- o Hudson
- o Jenkins
- o IBM Rational Team Concert
- o IBM Rational Software SCLM
- o TeamCity
- o TinderboxTravis CI
- o Xcode 5

3.7 Agile Modeling (AM)

Review Question

- Q. Discuss the significance of Agile Modeling (AM). What are different modeling principles that make agile modeling unique?

- There are many situations where the software developer build very large software systems used for some business applications and other critical software system. The software developer should model the complexity of such system so that he should be able to define the scope of the system. The software developer do so for the following reasons :

All the component of the system should be understood properly

- o The actual problem can be divided into various partition to solve and merge it into a single unit
- o The quality of the software is evaluated at each step during its construction.

- In the last three decades, various modelling technique have been put forwarded for analysis and design at both architectural level and component level.
- All these methods have profound impact and are difficult to apply. This method once applied is difficult to sustain over the life of software project.
 - In order to handle this deficiency, some rigid techniques are required so that the projects can be handled intellectually.
 - The best alternative to this approach is agile modelling. The agile modelling is defined as the practice based methodology that is very effective in modelling documentation of various software systems.
 - The agile modelling is actually a collection of design principles and practices for modelling and it is applied to software system.
 - In actual practice, the agile models are more effective and lightweight as compared to traditional models.
 - In addition to agile manifesto the agile modelling suggest following techniques :
 - o **Model with a purpose :** A software developer using agile modelling must have a proper and specific goal in his mind before he start developing a project .
 - o **Use multiple model :** The software developer is free to use to various model and notation to explain a software in a better way. The agile modelling suggests different aspect of system and provides the value to the intended end-users.
 - o **Travel light :** Agile modelling also suggest to use the models that provide long terms values and must be able to accommodate to the changes occurring at the later stages i.e. the developer must used light weight model.

- o **Content is more important than presentation :** In agile modelling the contents are very important for the intended users. Based on presentation only, the work can not be accomplished
- o **Know the model and the tools you are using to create the software :** While using agile methodology a software developer must know the strength and weaknesses of each of the models and tools.
- o **Adapt locally :** The agile Modelling approach must be adaptable to the needs of the team.

Syllabus Topic : Introduction to Agile Tools : JIRA

3.8 Introduction to Agile Tools : JIRA

- JIRA is an agile tool used for issue tracking and project management by over 25,000 customers in 122 countries in the world.
- JIRA lets you prioritize, assign, track, report and audit your issues from software bugs and help-desk tickets to project tasks and change requests.
- JIRA is offered in three packages :
 - o JIRA Core includes the base software.
 - o JIRA Software is intended for use by software development teams and includes JIRA Core and JIRA Agile.
 - o JIRA Service Desk is intended for use by IT or business service desks.
- The main features of JIRA for agile software development are the functionality to plan development iterations, the iteration reports and the bug tracking functionality.
- JIRA is a commercial software product that can be licensed for running on-premises or available as a hosted application. Pricing depends on the maximum number of users.

3.8.1 JIRA platform

- Every JIRA application (JIRA Software, JIRA Service Desk, and JIRA Core) is built on the JIRA platform. The JIRA platform provides a set of base functionality that is shared across all JIRA applications, like issues, workflows, search, email, and more.
- You can extend and modify this functionality via the integration points provided by the JIRA platform, including the JIRA REST APIs, webhooks, plugin modules, etc.
- The pages in this section will help you learn about the JIRA platform and how to develop for it. You'll find information on JIRA's architecture and JIRA add-ons, guides to developing for different parts of JIRA, JIRA Data Centre, and more.

3.8.2 JIRA Architecture

- JIRA is a web application written in Java. It is deployed as a standard Java WAR file into a java Servlet Container such as Tomcat.
- JIRA uses Open Symphony's WebWork 1 to process web requests submitted by users. Please note that WebWork 1, not 2, is used. WebWork 1 is a MVC framework similar to Struts. Each request is handled by a WebWork action which usually uses other objects, such as utility and Manager classes to accomplish a task.
- JIRA uses JSP for the View layer. So most of HTML that is served to the user as the response to their web request is generated by a JSP. Therefore, to generate a response the WebWork action uses a JSP.

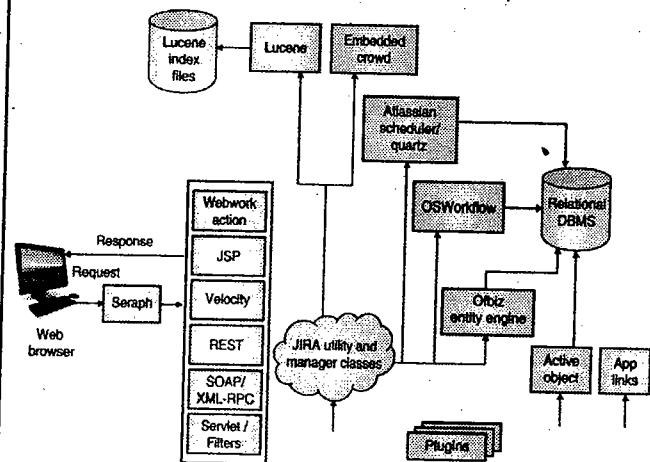


Fig. 3.8.1 : JIRA architecture

3.8.3 JIRA Database Schema

- To generate schema information for the JIRA database, e.g. PDF file, follow the instructions below. You can generate schema information in pdf, txt and dot formats. Note, if you want to generate the schema in PDF format, you need to have Graph viz installed.
 - (1) Download the attached plugin:
 - o For JIRA 5: JIRA-schema-diagram-generator-plugin-1.0.jar
 - o For JIRA 6: JIRA-schema-diagram-generator-plugin-1.0.1.jar
 - o For JIRA 7: JIRA-schema-diagram-generator-plugin-1.1.0.jar
 - (2) Install the plugin in your JIRA instance by following the instructions on Managing JIRA's Plugins.
 - (3) Database Schema
 - o Go to the JIRA administration console and navigate to System > Troubleshooting and Support > Generate Schema Diagram
 - o (tick)Keyboard shortcut: g + g + start typing generate
 - o Enter the tables/columns to omit from the generated schema information, if desired.

- o If you want to generate a pdf, enter the path to the Graph viz executable.
- o Click Generate Schema.
- o The 'Database Schema' page will be displayed with links to the schema file in txt, dot and pdf format.

3.8.4 JIRA Mobile Connect

- JIRA Mobile Connect (JMC) is an iOS library that can be embedded into any iOS app to provide following extra functionality:
 1. **Real-time crash reporting**, have users or testers submit crash reports directly to your JIRA instance.
 2. **User or tester feedback views** that allow users or testers to create bug reports within your app.
 3. **Rich data input**, users can attach and annotate screenshots, leave a voice message, and have their location sent.
 4. **Two-way communication with users**, thank your users or testers for providing feedback on your app.

3.8.5 JIRA Applications

- The JIRA family of applications currently consists of JIRA Software, JIRA Service Desk, and JIRA Core. A JIRA application is built on the JIRA platform, and extends and modifies the base functionality provided by the JIRA platform.
- For example, the JIRA Software application provides software development features that are not part of the JIRA platform, such as agile boards, linked development tool information (e.g. commits, builds, etc), and software project templates.

3.8.6 JIRA APIs

- The JIRA platform provides both Java APIs and REST APIs that you can use to interact with JIRA programmatically. These APIs are common to all JIRA applications.

- In addition, JIRA Software and JIRA Service Desk provide APIs for application-specific functionality. For example, the JIRA Software REST API has methods for creating sprints, creating boards, retrieving epics, etc.

Syllabus Topic : Introduction to Agile Tools - Kanban

3.9 Introduction to Agile Tools : Kanban

- Kanban is a popular framework used by software teams practicing agile software development. It is enormously prominent among today's agile software teams, but the kanban methodology of work dates back more than 50 years.
- In the late 1940s Toyota began optimizing its engineering processes based on the same model that supermarkets were using to stock their shelves.
- Supermarkets stock just enough products to meet consumer demand, a practice that optimizes the flow between the supermarket and the consumer.
- Because inventory levels match consumption patterns, the supermarket gains significant efficiency in inventory management by decreasing the amount of excess stock it must hold at any given time.
- Meanwhile, the supermarket can still ensure that the given product a consumer needs is always in stock.
- Agile software development teams today are able to leverage these same JIT principles by matching the amount of work in progress (WIP) to the team's capacity.
- This gives teams more flexible planning options, faster output, clearer focus, and transparency throughout the development cycle.

- While the core principles of the framework are timeless and applicable to almost any industry, software development teams have found particular success with the agile practice.
- In part, this is because software teams can begin practicing with little to no overhead once they understand the basic principles.
- Unlike implementing kanban on a factory floor, which would involve changes to physical processes and the addition of substantial materials, the only physical things a software teams need are a board and cards, and even those can be virtual.

3.9.1 Kanban Boards

- The work of all kanban teams revolves around a kanban board, a tool used to visualize work and optimize the flow of the work among the team.
- While physical boards are popular among some teams, virtual boards are a crucial feature in any agile software development tool for their traceability, easier collaboration, and accessibility from multiple locations.
- Regardless of whether a team's board is physical or digital, their function is to ensure the team's work is visualized, their workflow is standardized, and all blockers and dependencies are immediately identified and resolved.
- A basic kanban board has a three-step workflow: To Do, In Progress, and Done. However, depending on a team's size, structure, and objectives, the workflow can be mapped to meet the unique process of any particular team.
- The kanban methodology relies upon full transparency of work and real-time communication of capacity, therefore the kanban board should be seen as the single source of truth for the team's work.

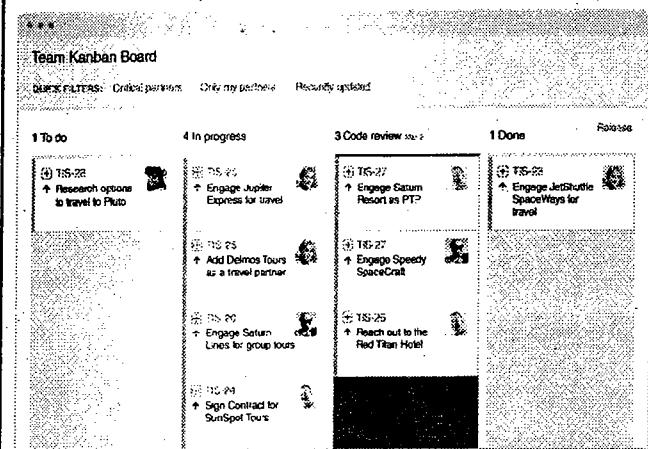


Fig.3.9.1 : Kanban boards

3.9.2 Kanban Cards

- In Japanese, kanban literally translates to "visual signal." For kanban teams, every work item is represented as a separate card on the board.
- The main purpose of representing work as a card on the kanban board is to allow team members to track the progress of work through its workflow in a highly visual manner.
- Kanban cards feature critical information about that particular work item, giving the entire team full visibility into who is responsible for that item of work, a brief description of the job being done, how long that piece of work is estimated to take, and so on.
- Cards on virtual kanban boards will often also feature screenshots and other technical details that is valuable to the assignee.
- Allowing team members to see the state of every work item at any given point in time, as well as all of the associated details, ensures increased focus, full traceability, and fast identification of blockers and dependencies.

3.9.3 The Benefits of Kanban

- Kanban is one of the most popular software development methodologies adopted by agile teams today. Kanban offers several

- additional advantages to task planning and throughput for teams of all sizes.
- **Planning flexibility :** A kanban team is only focused on the work that's actively in progress. Once the team completes a work item, they pluck the next work item off the top of the backlog.
- The product owner is free to reprioritize work in the backlog without disrupting the team, because any changes outside the current work items don't impact the team.
- As long as the product owner keeps the most important work items on top of the backlog, the development team is assured they are delivering maximum value back to the business. So there's no need for the fixed-length iterations you find in scrum.
- **Shortened cycle times :** Cycle time is a key metric for kanban teams. Cycle time is the amount of time it takes for a unit of work to travel through the team's workflow—from the moment work starts to the moment it ships.
- By optimizing cycle time, the team can confidently forecast the delivery of future work.
- Overlapping skill sets lead to smaller cycle times. When only one person holds a skill set, that person becomes a bottleneck in the workflow. So teams employ basic best practices like code review and mentoring help to spread knowledge.
- Shared skills mean that team members can take on heterogeneous work, which further optimizes cycle time. It also means that if there is a backup of work, the entire team can swarm on it to get the process flowing smoothly again.
- For instance, testing isn't only done by QA engineers. Developers pitch in, too.
- In a kanban framework, it's the entire team's responsibility to ensure work is moving smoothly through the process.
- **Fewer bottlenecks :** Multitasking kills efficiency. The more work items in flight at any given time, the more context switching, which hinders their path to completion.
- That's why a key tenant of kanban is to limit the amount of work in progress (WIP). Work-in-progress limits highlight bottlenecks and backups in the team's process due to lack of focus, people, or skill sets.
- For example, a typical software team might have four workflow states: To Do, In Progress, Code Review, and Done. They could choose to set a WIP limit of 2 for the code review state.
- That might seem like a low limit, but there's good reason for it: developers often prefer to write new code, rather than spend time reviewing someone else's work.
- A low limit encourages the team to pay special attention to issues in the review state, and to review others work before raising their own code reviews. This ultimately reduces the overall cycle time.
- **Visual metrics :** One of the core values is a strong focus on continually improving team efficiency and effectiveness with every iteration of work.
- Charts provide a visual mechanism for teams to ensure they're continuing to improve. When the team can see data, it's easier to spot bottlenecks in the process (and remove them).
- Two common reports kanban teams use are control charts and cumulative flow diagrams.
- A control chart shows the cycle time for each issue as well as a rolling average for the team.

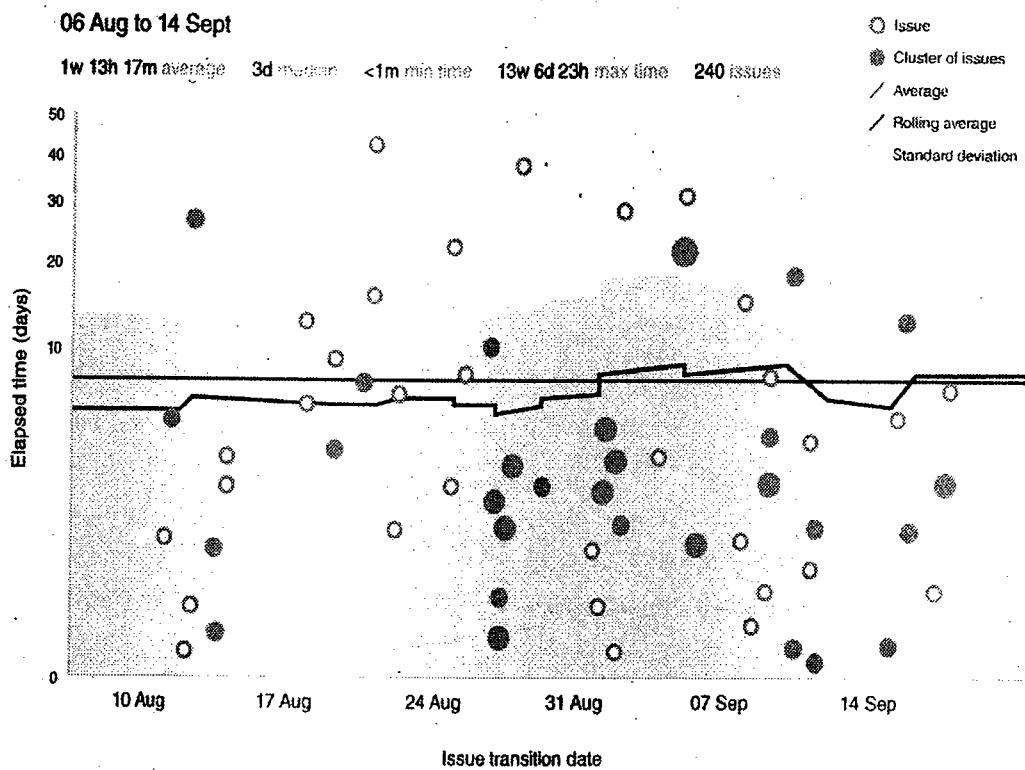


Fig. 3.9.2 : Control chart

- A cumulative flow diagram shows the number of issues in each state. The team can easily spot blockages by seeing the number of issues increase in any given state. Issues in intermediate states such as "In Progress" or "In Review" are not yet shipped to customers, and a blockage in these states can increase the likelihood of massive integration conflicts when the work does get merged upstream.

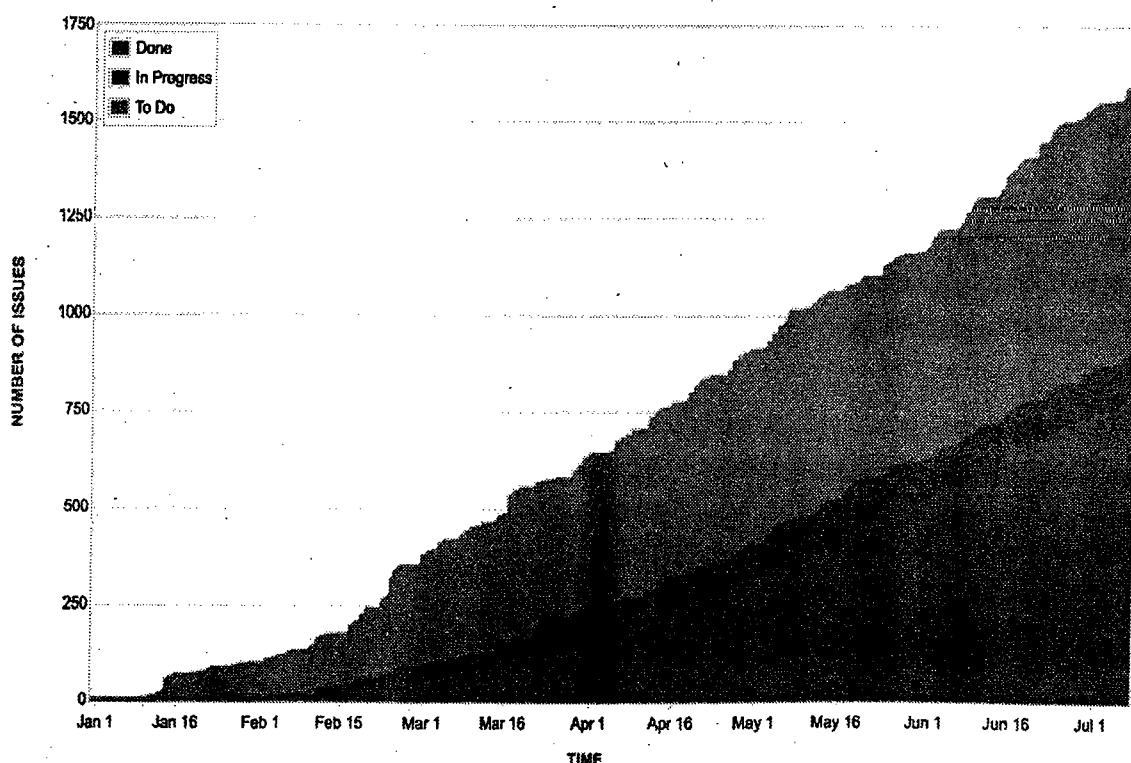


Fig. 3.9.3 : Cumulative flow diagram

- **Continuous delivery** : We know that continuous integration—the practice of automatically building and testing code incrementally throughout the day—is essential for maintaining quality. Now it's time to meet continuous delivery (CD). CD is the practice of releasing work to customers frequently—even daily or hourly. Kanban and CD beautifully complement each other because both techniques focus on the just-in-time (and one-at-a-time) delivery of value.
- The faster a team can deliver innovation to market, the more competitive their product will be in the marketplace. And kanban teams focus on exactly that: optimizing the flow of work out to customers.

3.9.4 Comparison between Kanban and Scrum

Kanban	Scrum
In Kanban, there is continuous flow.	In Scrum approach, we have regular fixed length sprints (i.e. 2 weeks)
Continuous delivery or at the team's discretion.	Delivery at the end of each sprint if approved by the product owner.
Cycle time is the key metrics in Kanban approach.	Velocity is the key metrics.
Changes can happen at any time.	Teams should strive to not make changes to the sprint forecast during the sprint. Doing so compromises learning around estimation.
Roles : No existing roles. Some teams enlist the help of an agile coach.	Roles : Product owner, scrum master, development team etc.

Syllabus Topic : Case Study : An Information System(Mental Health-care System)

3.10 Mental Health-Care System

- A patient information system to support mental health care is a medical information

system that maintains information about patients suffering from mental health problems and the treatments that they have received. Most mental health patients do not require dedicated hospital treatment but need to attend specialist clinics regularly where they can meet a doctor who has detailed knowledge of their problems. To make it easier for patients to attend, these clinics are not just run in hospitals. They may also be held in local medical practices or community centers.

- The MHC-PMS (Mental Health Care-Patient Management System) is an information system that is intended for use in clinics. It makes use of a centralized database of patient information but has also been designed to run on a PC, so that it may be accessed and used from sites that do not have secure network connectivity. When the local systems have secure network access, they use patient information in the database but they can download and use local copies of patient records when they are disconnected. The system is not a complete medical records system so does not maintain information about other medical conditions. However, it may interact and exchange data with other clinical information systems. Figure 1.6 illustrates the organization of the MHC-PMS.

- The MHC-PMS has two overall goals:
 1. To generate management information that allows health service managers to assess performance against local and government targets.
 2. To provide medical staff with timely information to support the treatment of patients.
- The nature of mental health problems is such that patients are often disorganized so may miss appointments, deliberately or accidentally lose prescriptions and

medication, forget instructions, and make unreasonable demands on medical staff. They may drop in on clinics unexpectedly. In a minority of cases, they may be a danger to themselves or to other people. They may regularly change address or may be homeless on a long-term or short-term basis. Where patients are dangerous, they may need to be 'sectioned'—confined to a secure hospital for treatment and observation.

- Users of the system include clinical staff such as doctors, nurses, and health visitors (nurses who visit people at home to check on their treatment). Nonmedical users include receptionists who make appointments, medical records staff who maintain the records system, and administrative staff who generate reports.
- The system is used to record information about patients (name, address, age, next of kin, etc.), consultations (date, doctor seen, subjective impressions of the patient, etc.), conditions, and treatments. Reports are generated at regular intervals for medical staff and health authority managers. Typically, reports for medical staff focus on information about individual patients whereas management reports are anonymized and are concerned with conditions, costs of treatment, etc.
- The key features of the system are :

1. **Individual care management :** Clinicians can create records for patients, edit the information in the system, view patient history, etc. The system supports data summaries so that doctors who have not previously met a patient can quickly learn about the key problems and treatments that have been prescribed.
2. **Patient monitoring :** The system regularly monitors the records of patients that are involved in treatment

and issues warnings if possible problems are detected. Therefore, if a patient has not seen a doctor for some time, a warning may be issued. One of the most important elements of the monitoring system is to keep track of patients who have been sectioned and to ensure that the legally required checks are carried out at the right time.

3. **Administrative reporting :** The system generates monthly management reports showing the number of patients treated at each clinic, the number of patients who have entered and left the care system, number of patients sectioned, the drugs prescribed and their costs, etc.

Two different laws affect the system. These are laws on data protection that govern the confidentiality of personal information and mental health laws that govern the compulsory detention of patients deemed to be a danger to themselves or others. Mental health is unique in this respect as it is the only medical speciality that can recommend the detention of patients against their will. This is subject to very strict legislative safeguards. One of the aims of the MHC-PMS is to ensure that staff always acts in accordance with the law and that their decisions are recorded for judicial review if necessary.

As in all medical systems, privacy is a critical system requirement. It is essential that patient information is confidential and is never disclosed to anyone apart from authorized medical staff and the patient themselves. The MHC-PMS is also a safety-critical system. Some mental illnesses cause patients to become suicidal or a danger to other people. Wherever possible, the system should warn medical staff about potentially suicidal or dangerous patients.

- The overall design of the system has to take into account privacy and safety requirements. The system must be available when needed otherwise safety may be compromised and it may be impossible to prescribe the correct medication to patients. There is a potential conflict here, privacy is easiest to maintain when there is only a single copy of the system data. However, to ensure availability in the event of server failure or when disconnected from a network, multiple copies of the data should be maintained.

Syllabus Topic : Case Study : Wilderness Weather System

3.11 Wilderness Weather System

- To help monitor climate change and to improve the accuracy of weather forecasts in remote areas, the government of a country with large areas of wilderness decides to deploy several hundred weather stations in remote areas.

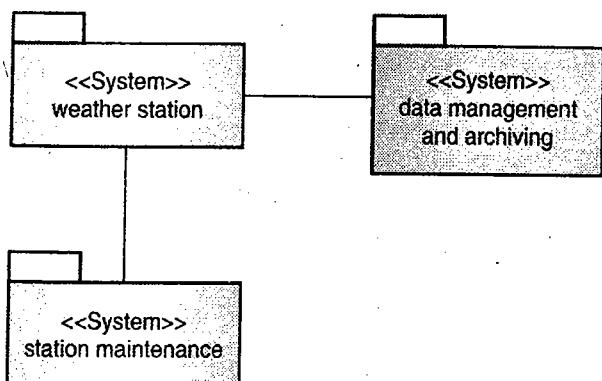


Fig. 3.11.1 : The weather station's environment

- These weather stations collect data from a set of instruments that measure temperature and pressure, sunshine, rainfall, wind speed, and wind direction.
- Wilderness weather stations are part of a larger system (Fig. 3.11.1), which is a weather information system that collects data from weather stations and makes it

available to other systems for processing. The systems in Fig. 3.11.1 are:

- The weather station system :** This is responsible for collecting weather data, carrying out some initial data processing, and transmitting it to the data management system.
- The data management and archiving system :** This system collects the data from all of the wilderness weather stations, carries out data processing and analysis, and archives the data in a form that can be retrieved by other systems, such as weather forecasting systems.
- The station maintenance system :** This system can communicate by satellite with all wilderness weather stations to monitor the health of these systems and provide reports of problems. It can update the embedded software in these systems. In the event of system problems, this system can also be used to remotely control a wilderness weather system.

In Fig. 3.11.1, I have used the UML package symbol to indicate that each system is a collection of components and have identified the separate systems, using the UML stereotype «system». The associations between the packages indicate there is an exchange of information but, at this stage, there is no need to define them in any more detail.

- Each weather station includes a number of instruments that measure weather parameters such as the wind speed and direction, the ground and air temperatures, the barometric pressure, and the rainfall over a 24-hour period. Each of these instruments is controlled by a software system that takes parameter readings periodically and manages the data collected from the instruments.

- The weather station system operates by collecting weather observations at frequent intervals—for example, temperatures are measured every minute. However, because the bandwidth to the satellite is relatively narrow, the weather station carries out some local processing and aggregation of the data. It then transmits this aggregated data when requested by the data collection system. If, for whatever reason, it is impossible to make a connection, then the weather station maintains the data locally until communication can be resumed.
- Each weather station is battery-powered and must be entirely self-contained—there are no external power or network cables available. All communications are through a relatively slow-speed satellite link and the weather station must include some mechanism (solar or wind power) to charge its batteries. As they are deployed in wilderness areas, they are exposed to severe environmental conditions and may be damaged by animals. The station

software is therefore not just concerned with data collection. It must also :

1. Monitor the instruments, power, and communication hardware and report faults to the management system.
2. Manage the system power, ensuring that batteries are charged whenever the environmental conditions permit but also that generators are shut down in potentially damaging weather conditions, such as high wind.
3. Allow for dynamic reconfiguration where parts of the software are replaced with new versions and where backup instruments are switched into the system in the event of system failure.

Because weather stations have to be self-contained and unattended, this means that the software installed is complex, even though the data collection functionality is fairly simple.



Requirement Engineering

Syllabus

Requirements Engineering: User and system requirements, Functional and non-functional requirements, Types & Metrics, A spiral view of the requirements engineering process. Software Requirements Specification (SRS): The software requirements Specification document, The structure of SRS, Ways of writing a SRS, structured & tabular SRS for an insulin pump case study, Requirements elicitation & Analysis: Process, Requirements validation, Requirements management. Case Studies: The information system. Case study - Mental health care patient management system (MHC-PMS).

Syllabus Topic : Requirements Engineering

4.1 Introduction

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

analysis is wrong, then design will be wrong.

Ultimately coding will be wrong. Finally, software expectations will not match with outcomes. Hence requirements engineering should be carried out carefully.

4.2 Requirement Engineering

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

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

4.2.1 Inception

SPPU - Feb. 16

University Question

Q. What do you mean by requirement inception ?

(Feb. 2016, 6 Marks)

- Inception is beginning and it is usually said that, 'well beginning is half done'. But it is always problematic for the developer that what should be the starting point i.e. 'from where to start'.
- The customer and developer meet and they decide the overall scope and nature of the problem statement. The aim is :
 - o To have the basic understanding of the problem.
 - o To know the people who will use the software.
 - o To know exact nature of problem that is expected from customer's side.
 - o To maintain effectiveness of preliminary communication.
 - o To have collaboration between customer and developer.
- The requirements engineering is a 'communication intensive' activity because a requirement gathering is an initial step for design. Hence exact requirements are gathered with customer communication.
- The developer must ask following questions:
 - o Who is behind the request for this work?
 - o Who will use the solution ?
 - o What will be the economical benefits of a successful solution ?
 - o Is there another source of solution for the same problem ?

4.2.2 Elicitation

SPPU - Feb. 16

University Question

Q. What do you mean by requirement elicitation ?

(Feb. 2016, 6 Marks)

- Elicitation means, 'to draw out the truth or reply from anybody'. In relation with

requirements engineering, elicitation is a task that helps the customer to define what is required.

- To know the objectives of the system or the project to be developed is a critical job. Why it is difficult to get a clear understanding of customer wants? To answer this question, we have a number of problems like :

Problems of scope

Many times customer states unnecessary technical details. The unnecessary details may confuse developer instead of giving clarity of overall system objectives.

Problems of understanding

Sometimes both customer as well as developer has poor understanding of :

- What is needed ?
- Capabilities and limitations of the computing environment.
- Understanding of problem domain.
- Specify requirements those conflict with other customers and users.

Problems of volatility

Volatility means 'change from one state to another'. The customer's requirements may change time to time. This is also a major problem while deciding fixed set of requirements.

4.2.3 Elaboration

- Elaboration means 'to work out in detail'. The information obtained during inception and elicitation is expanded and modified during elaboration.
- Now requirements engineering activity focuses on developing the technical model of the software that will include :
 - o Functions
 - o Features.
 - o Constraints

- Thus, elaboration is an 'analysis modeling' action. This model focuses on 'how the end user will interact with the system'.

4.2.4 Negotiation

SPPU - Feb. 15

University Question

Q. What do you mean by requirement negotiation? (Feb. 2015, 2 Marks)

- Here, 'Negotiation' means 'discussion on financial and other commercial issues'.
- In this step customer, user and other stakeholders discuss to decide :
 - o To rank the requirements
 - o To decide priorities
 - o To decide risks
 - o To finalize the project cost
 - o The impact of above issues on project cost and delivery date.

4.2.5 Specification

- The specification is the final work product produced by requirement engineer. The specification may take different forms :
 - o A written document
 - o A set of graphical model
 - o A formal mathematical model
 - o A collection of scenarios
 - o A prototype
 - o Any combination of above
- The specification is considered as the foundation of all subsequent software engineering activities.
- The specification describes the function and performance of the computer based systems. The specification also describes the constraints are necessary in its development.

4.2.6 Validation

SPPU – Feb.15, Feb. 16

University Question

Q. What do you mean by requirement validation? (Feb. 2015, Feb. 2016, 6 Marks)

- All previous work completed will be just useless and meaningless if it is not validated against the customers expectations.
- The requirement validation checklist includes :
 - o All requirements are stated clearly ?
 - o Are the requirements misinterpreted ?
 - o Does the requirements violate any system domain constrains?
 - o Is the system requirement traceable to the system model that is created?
 - o Are the requirements associated with performance, behaviour and operational characteristics ?

4.2.7 Requirement Management

- Requirement management is a software engineering task that helps the project team members to identify the requirements, control the requirements, track the requirements and changes to requirements at any time as the project exceeds.
- The requirement management task starts with identification and each of the requirements is assigned a unique identifier.
- After the requirements finalization, the traceability table is developed. Traceability table relates the requirements to one or more aspects of the system or its environment. The traceability tables are used as requirements database and are useful in understanding how change in one particular requirement will affect other parts or aspects of the system.

4.2.8 Initiating the Requirement Engineering Process

- In requirements engineering process, following are considerable issues :



- o Customers may be located at different cities or countries.
 - o Customers do not have clear idea of product requirements.
 - o Different customers may have different and conflicting opinions about the system to be built.
 - o Customers may have limited technical knowledge.
 - o Customers may have only limited time to interact with requirement engineer.
- Hence, considering all these issues, following are the steps required to initiate the requirements engineering process :

Identifying the Stakeholders

- Stakeholder is anyone who has direct interest in or benefits from the system that is to be developed. Hence, we can list following people as stakeholders :
 - o Business operations manager
 - o Product managers
 - o Marketing people
 - o Internal and external customers
 - o End users
 - o Consultants
 - o Product engineers
 - o Software engineers
 - o Support and maintenance engineers
 - o Others
- All these people have different view of the system regarding the system that is to be developed.

Recognizing Multiple Viewpoints

As many stakeholders exist, they all have different views regarding the system that is to be developed.

For example

- Marketing people are interested in functions and features having potential market.

- Business people are interested in functions and features those can be set within minimum budget.
- End users are interested in functions and features those will be easy to understand and use.
- Software engineers are interested in functions and features those support their existing infrastructure.
- Support engineer may focus on the maintainability of the software.
- It's a duty of software engineer to consider all these viewpoints of all the stakeholders and find the consistent and balanced set of requirements.

Working towards collaboration

Customers must collaborate with themselves and software engineering practitioners to get successful system. The job of requirement engineer is to find :

1. **Common areas** : The requirements for which all stakeholders agree.
2. **Conflict areas** : The requirements that are proposed by one stakeholder but opposed by another stakeholder.

Now, the higher authorities like business manager or senior technologist can take the decision for the requirements either to forward or to reject these requirements.

Asking the First Questions

The requirements engineering is a 'communication intensive' activity because 'requirements gathering' is an initial step for design. Hence following three sets of questions are used to gain to understand the requirements :

1. First set : The context free questions

- o Who is behind the request for this work ?
- o Who will use the solution ?
- o What will be the economical benefits of a successful solution ?
- o Is there another source of solution for the same problem ?

2. Second set : Questions to gain preliminary understanding of problem

- o How will you characterize good output that will be generated by successful solution ?
- o What are the probable problems for this solution ?
- o What is the business environment in which this solution will be used ?
- o What are special performance issues and constraints that will affect the solution ?

3. Third set : Questions those focus on effectiveness of the communication activity

- o Are the questions official ?
- o Are the questions related to the problem ?
- o Are the questions 'too many' in quantity ?
- o Can anyone else provide additional information ?
- o Are some questions left to ask ?

Syllabus Topic : User and System Requirements

4.3 User and System Requirements

- The fundamental definition of the requirements for a system is :

"The set of descriptions listed for the system to accomplish the required services and the constraints on its operation."

- These requirements reflect the needs of customers for a system. The process of identifying the customers needs, analyzing, documenting and checking the services and constraints that a system should provide, is called requirements engineering (RE).

- Most of the time, the term 'Requirement' is not used in software organizations. But the

requirement is illustrated as 'high-level, abstract statement of services that a system should provide for its users'.

The problem exists when we talk about distinction between different levels of description. Primarily the requirements specification can divided into two important classes as follows :

- o User requirements
- o System requirements

- The **User requirements** are the statements narrated by the customer in simple natural language and with some useful diagrams.

- The customer describes what services the system should provide and the constraints on the operations and functions of the system.

- The **System requirements** are more detailed descriptions of the software system's functions, services, and operational constraints.

- The system requirements document also called as functional specification, should define exactly what is to be implemented. It is also the contract between the system buyer and the software developers. All these requirements are well documented on paper for future references.

- Following figure exhibits the readers of different types of requirements specification :

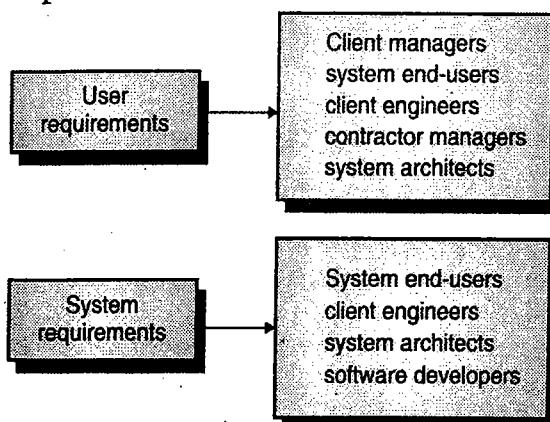


Fig. 4.3.1 Readers of different types of requirements specification



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

Syllabus Topic : Functional and Non - Functional Requirements

4.4 Functional and Non- Functional Requirements

SPPU - Feb. 15, May 16

University Question

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

(Feb. 2015, May 2016, 7 Marks)

- The system requirements are classified into following two types :
 - o Functional requirements
 - o Non-functional requirements
- **The functional requirements** are the statements of the services that a system should provide and how the system must react to a given input and the system reacts in some situations. The functional requirement should also mention that a system should do nothing in some situations.
- **The non-functional requirements** are treated as some constraints that a system should behave in some situations. Also the non-functional requirements are the explicit conditions that are put on the services and the functionalities of the system to behave. The non-functional requirements are applied on the whole

system and not on the individual system features. .

4.4.1 Functional Requirements

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

4.4.2 Non-functional Requirements

- The non-functional requirements are not directly related to the functions, services of the system and the desired outputs.

- The non-functional requirements are related to the system properties that are listed below :
 - o Reliability
 - o Speed of responses i.e. response time,
 - o System performance,
 - o System security,
 - o System availability,
 - o Portability,
 - o Robustness,
 - o Ease of use etc.
- In contrast to functional requirements, the non-functional requirements are more critical. Therefore, if functional requirements are failed then the whole system may be failed and become unusable.
- For example if a air system does not provide reliability, then it can not be used in actual practice. It will not be called as a safe system.
- The failure of an individual non-functional requirement may lead to failure of the whole system.

Syllabus Topic : Types and Metrics

4.5 Types and Metrics

- In actual practice, the classification of different **Types of requirement** is not very much clear according to the definitions of requirements described in earlier sections.
- A user requirement which is directly connected to the parameters like security, authorizations is supposed to be the nonfunctional requirement.
- But when such systems are developed in more detail, this requirement may generate other requirements that are clearly

functional, such as the need to include user authentication facilities in the system etc. This exhibits that requirements are not independent and that one requirement often generates or constrains other requirements. Therefore the system requirements not only specify the services or the features of the system that are desired, but they also specify the necessary functionality to ensure that these services or the features are delivered properly.

4.5.1 Metrics for Specifying Non-functional Requirements

When we write non-functional requirements, we can test it. Following are some of the metrics that can be used to specify non-functional properties of the system. We can easily measure these properties or characteristics when the software is under testing.

- o **Speed** : While measuring the speed property, we observe the speed of the processed transactions. We also see event response time and event refresh time.
- o **Robustness** : Time to recover after failure occurred, Probability of data corruption on failure and Percentage of events causing failure.
- o **Portability** : Number of target systems and Percentage of target dependent statements.
- o **Size** : Measurement of read only memory (ROM) chips available in the system.
- o **Ease of use** : Number of help frames available and training time required.
- o **Reliability** : It is the mean time to failure (MTTF), or Probability of unavailability, Rate of failure occurrence etc.

- If the non-functional requirements are given separately from the functional requirements, then the relationships between them may be difficult to understand. Non-functional requirements such as reliability, safety, and confidentiality requirements are particularly important for critical systems.

Syllabus Topic : A Spiral View of the Requirements Engineering Process

4.6 A Spiral View of the Requirements Engineering Process

- Basically requirements engineering processes includes following four high-level activities :
 - o **Feasibility study** : To check whether the system is useful for the business.
 - o **Requirements elicitation and analysis** : Finding the requirements from the user and conduct analysis on them.
 - o **Requirements specification** : Requirements received in some natural language and in the form of some diagrams and convert them in some standard form.
 - o **Requirements validation** : It is actually the checking of systems functioning. Whether it is in function according to the requirement elicited by the user.
- All these activities constitute requirements engineering process and can be organized as an iterative process to form a spiral view. In actual practice, all these activities are interleaved in one another. The spiral view of the activities involved in the requirements engineering process as exhibited as follows :

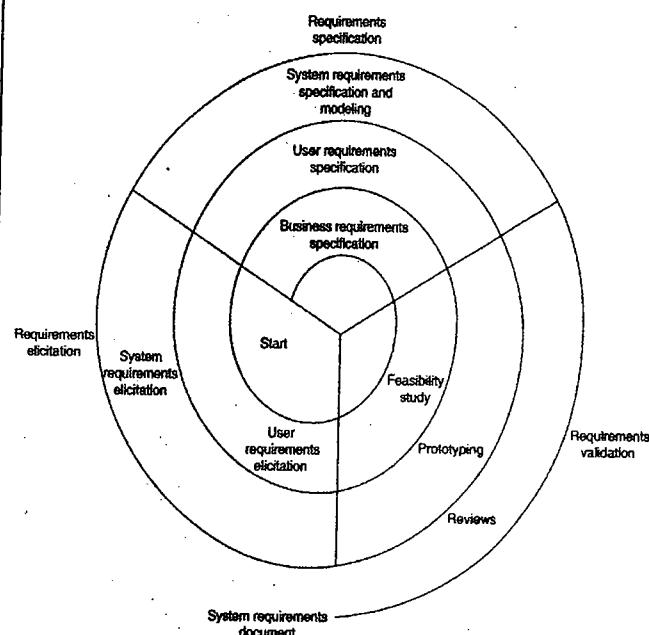


Fig. 4.6.1: Spiral view of the activities involved in the requirements engineering process

Syllabus Topic : Software Requirements Specification (SRS), The Structure of SRS

4.7 Software Requirements Specification (SRS)

Review Question

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

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

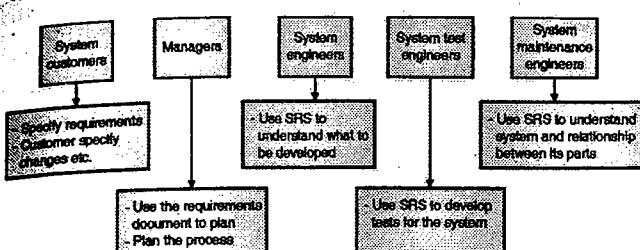


Fig. 4.7.1 : Users of SRS

- The details which are included in SRS depend on the type of system that is being developed and the type of the development process.
- A number of large organisations, such as IEEE have defined standards for software requirements document.
- The most widely used standard is IEEE/ANSI 830-1998. This standard suggests the following structure for requirements document :
 - 1. Introduction
 - o Purpose of SRS
 - o Scope of product.
 - o Definitions, acronyms, abbreviations.
 - o References
 - o Overview
 - 2. General description
 - o Product perspective
 - o Product Functions
 - o User characteristics
 - o General constraints
 - o Assumptions and dependencies
 - 3. Specific requirements
 - 4. Appendices
 - 5. Index.
- SRS is very useful when an outside contractor is developing the software system.
- For business system, where requirements are unstable, SRS play an important role.

Syllabus Topic : Ways of Writing a SRS

4.7.1 Writing Software Requirements Specifications

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

4.7.2 What is a Software Requirements Specification?

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



- It contains functional and non functional requirements.

Good SRS have accomplishes following goals

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

4.7.3 What Kind of Information Should an SRS Include?

Following things are part of SRS :

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

4.7.4 SRS Template

- Different existing SRS templates can be used in writing SRS documents.
- We can select the template which matches our requirement.
- Template will just acts as guiding principles for writing template. Proper changes need to be done whenever necessary in template.
- Table 4.7.1 shows SRS outline :

Table 4.7.1 : A sample of a basic SRS outline

1. **Introduction** : It contains different details like purpose of software, conventions used, target audiences, extra information, SRS team members, references.

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

4.7.5 Characteristics of an SRS

SPPU - Feb. 15, Feb. 16

University Question

- Q. Explain four desirable characteristics of a good Software Requirements Specification(SRS) document. (Feb. 2015, Feb. 2016, 6 Marks)

Review Question

- Q. Explain different characteristics of Software Requirement Specification ?

- **Complete** : All the project functionalities should be recorded by SRS.
- **Consistent** : SRS should be consistent
- **Accurate** : SRS defines systems functionality in real world. We need to record requirement very carefully.

- **Modifiable** : Logical and hierarchical modification should be allowed in SRS.
- **Ranked** : Requirement should be ranked using different factors like stability, security, ease or difficulty.
- **Testable** : The requirement should be realistic and able to implement.
- **Traceable** : Every requirement we must be able to uniquely identify.
- **Unambiguous** : Statement in the SRS document should have only one meaning.
- **Valid** : All the requirements should be valid.

Syllabus Topic : Structured SRS for an Insulin Pump Case Study

4.7.6 Structured Specifications for an Insulin Pump Case Study

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

requirements, supporting materials, and so on. This is similar to the approach used in the example of a structured specification shown in Table 4.7.2.

Table 4.7.2 : A structured specification of a requirement for an insulin pump

Function	Compute insulin dose: Safe sugar level.
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.
Inputs	Current sugar reading (r_2), the previous two readings (r_0 and r_1).
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose—the dose in insulin to be delivered.
Destination	Main control loop.
Action	CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.
Requirements	Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin.
Post-condition	r_0 is replaced by r_1 then r_1 is replaced by r_2 .
Side effects	None.

- To use a structured approach to specifying system requirements, you define one or more standard templates for requirements and represent these templates as structured forms. The specification may be structured around the objects manipulated by the system, the functions performed by the system, or the events processed by the system. An example of a form-based specification, in this case, one that defines how to calculate the dose of insulin to be delivered when the blood sugar is within a safe band, is shown in Table 4.7.2.



Syllabus Topic : Tabular SRS for an Insulin Pump Case Study

4.7.7 Tabular Specifications for an Insulin Pump Case Study

- Using structured specifications removes some of the problems of natural language specification. Variability in the specification is reduced and requirements are organized more effectively. However, it is still sometimes difficult to write requirements in a clear and unambiguous way, particularly when complex computations (e.g., how to calculate the insulin dose) are to be specified.
- To address this problem, you can add extra information to natural language requirements, for example, by using tables or graphical models of the system. These can show how computations proceed, how the system state changes, how users interact with the system, and how sequences of actions are performed.
- Tables are particularly useful when there are a number of possible alternative situations and you need to describe the actions to be taken for each of these. The insulin pump bases its computations of the insulin requirement on the rate of change of blood sugar levels. The rates of change are computed using the current and previous readings. Table 4.7.3 is a tabular description of how the rate of change of blood sugar is used to calculate the amount of insulin to be delivered.

Table 4.7.3 : Tabular specification of computation for an insulin pump

Condition	Action
Sugar level falling ($r_2 < r_1$)	$CompDose = 0$
Sugar level stable ($r_2 = r_1$)	$CompDose = 0$
Sugar level increasing and rate of increase	$CompDose = 0$
Decreasing ($((r_2 - r_1) < (r_1 - r_0))$)	
Sugar level increasing and rate	$CompDose = \text{round}$

Condition	Action
of increase stable or increasing ($((r_2 - r_1) \geq (r_1 - r_0))$)	$((r_2 - r_1)/4)$ If rounded result = 0 then $CompDose = \text{MinimumDose}$

Syllabus Topic : Requirements Elicitation and Analysis : Process

4.8 Requirements Elicitation : Process

SPPU - Dec. 12

University Question

Q. State and explain the methods for eliciting requirements. (Dec. 2012, 8 Marks)

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

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

4.8.1 Collaborative Requirements Gathering

- Gathering software requirements is team oriented activity. All the software team members, software engineering manager, members of marketing, and product engineering representatives all work together. The aim of this activity is :
 - o To identify the problem.
 - o To suggest the solution.
 - o To negotiate different approaches.
 - o To specify the preliminary set of solution requirements.
- The meeting for 'collaborative requirements gathering' is conducted to discuss all above issues.
- The basic guidelines for conducting a collaborative requirements gathering meeting are :

- o Meeting is conducted and attended by both software engineers as well as customers.
- o An agenda is suggested that is formal enough to cover all points those are to be discussed in the meeting.
- o A 'facilitator' may be customer, developer or outsider controls the meeting.
- o A definition mechanism including work sheets, charts, wall stickers, chat rooms, projectors is used.

4.8.2 Quality Function Deployment

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

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

1. Normal requirements

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

For example

- o Graphic displays.
- o Specific system functions.
- o Specified output formats.

2. Expected requirements

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

For example

- The developed system must provide easy human machine interaction.

- The system should be menu driven,
- All hot key buttons help should be provided.
- The system should be 'user friendly'.
- The system should be easy to install.

3. Exciting requirements

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

4.8.3 Usage Scenarios

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

4.8.4 Elicitation Workproduct

SPPU - May 13

University Question

Q. What are the workproducts of elicitation ?

(May 2013, 6 Marks)

- The work products produced by requirement elicitation depend upon the size of the system or the system to be built.
- The information produced as a consequence of requirements gathering includes :
 - o A statement of need and feasibility.
 - o A statement of scope for the system or product.

- A list of customers, users and other stakeholders who participated in requirement elicitation.
- Description of system's technical environment.
- The list of requirements and domain constraints.
- The set of scenarios.
- Any prototype developed to define requirements clearly.

4.8.5 Elicitation Techniques

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

4.8.6 Developing Use Cases

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

plays just one role in the context of use case.

For example

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

The software for control machine requires four different modes for interaction : Programming mode, test mode, monitoring mode, troubleshooting mode.

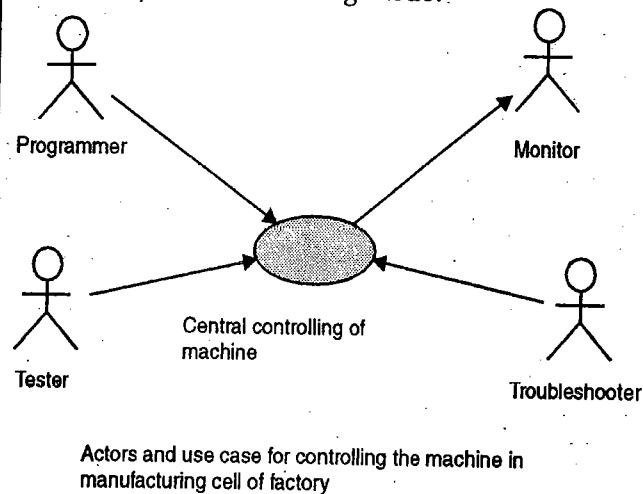


Fig. 4.8.1

- Hence four actors can be defined : programmer, tester, monitor and trouble shooter.

There are two types of actors :

1. Primary actors
2. Secondary actors

1. Primary actors

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

2. Secondary actors

- These actors support the system so that primary actors can do their work. After identifying the actors the use cases can

be developed. Jacobson suggests the number of questions those are answered by the use cases.

- o What are primary and secondary actors?
- o What are the goals of actors (i.e. primary and secondary actors)?
- o What are the preconditions that exist before the development begins?
- o What are tasks and functions that are performed by actors?
- o What are considerable exceptions?
- o What are the possible variations in primary and secondary actor's interaction?
- o What are the system information that a actor can acquire, produce?
- o What are the system information that a actor can modify?
- o Is it necessary for a actor to inform the system, the possible changes in the external environment?
- o What is the desired information of a system that an actor want to know?
- o Is it necessary for a system to inform the actor about unexpected changes?

4.9 Requirements Analysis

Review Question

Q. Explain requirement analysis with example.

- Analysis model uses a combination of text and diagrammatic forms to depict requirements of data, functions and behaviour. So that it will be easy to understand overall requirements of the software to be built.
- The 'Software Engineer' also called as 'Analyst' builds the model using requirements stated by the customer.

- Analysis model validates the software requirements and represent the requirements in multiple dimensions.

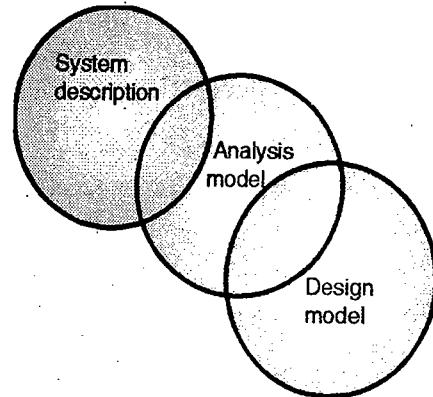


Fig. 4.9.1 : Analysis model, bridge in system description and design model

- Basic aim of analysis modelling is to create the model that represents the information, functions and behaviour of the system to be built.
- These information, functions and behaviour of the system are translated into architectural, interface and component level designs in design modeling.
- Information, functions and behaviour of the system are represented using number of different diagrammatic formats.
- As stated previously, the 'analysis model' acts as a bridge between 'system description' and the 'design model'.
- The system description describes overall system functionality of the system including software, hardware, databases, human interfaces and other system elements. And the software design mainly focuses on application architectural, user interface and component level designs.
- Thus, all elements of analysis model are directly traceable to the parts of design model. Hence, the analysis model must have following three objectives :
 - o To state clearly what customer wants exactly.



- To establish the basis of the 'design model'. The information, functions and behaviour of the system defined by 'analysis model' are translated into architectural, interface and component level designs in 'design modeling'.
- To bridge the gap between a system level description and software design.

4.9.1 Analysis Rules of Thumb

Review Question

Q. Explain Thumb rule for requirement analysis.

These rules are

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

designing the 'design model'. End user can use same model for testing.

The analysis model must be as simple as possible.

Only simple model will help the easy understanding of software requirements. Hence analysis model should be simple enough.

4.9.2 Domain Analysis

Review Question

Q. Explain Domain analysis. Also discuss its advantages.

By definition, software domain analysis is "the identification, analysis and specification of common requirements from a specific application domain." These common software domains can be reused for multiple projects within that application domain.

In short, software domain analysis leads to 'reusable software components' in software development. Here, in specific application domain common objects, common classes, common frameworks can be identified and can be reused.

Role of 'domain analyst' is same as job of toolsmith. The job of toolsmith is to design the tools those are used by many people for similar works. The role of domain analyst is to define reusable objects, classes, frameworks those can be used by many people in similar applications.

For example : The specific application domain may be 'bus reservation system'. The software domains of 'bus reservation system' can be used for 'railway reservation system'.

Advantages

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

4.9.3 Requirements Modeling Approaches

Review Question

Q. What are different requirement approaches ?

Following are different requirement modeling approaches :

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

Syllabus Topic : Requirements Validation

4.10 Requirements Validation

SPPU - Dec. 15, Dec. 16

University Questions

- Q.** Explain how do we negotiate and validate requirement analysis process.
(Dec. 2015, 7 Marks)
- Q.** What do you understand with the Validating requirements ?
(Dec. 2016, 5 Marks)

- Once requirement model is built, it is checked for inconsistencies and ambiguities. Then the requirements are optimized.

- During the review of the requirements model, following questions arise :

- o Are all the requirements consistent ?
- o Whether requirements followed proper level of abstraction ?
- o Is the requirement really necessary ?
- o Are there any requirements which conflicts with one another ?
- o Is each requirement testable, once implemented ?
- o Does the requirement model properly reflect the information, function and behaviour of the system to be built ?

- o Does the requirements model use the requirements pattern to simplify the model ?

- These questions are answered to make sure that the requirements model is as per the customer's exact needs.

Syllabus Topic : Requirements Management

4.11 Requirements Management

SPPU - May 14

University Question

- Q.** Draw and explain the traceability table for requirement management. (May 2014, 6 Marks)

Review Question

- Q.** Explain traceability table. How it is requirement management ?

- Requirement management is a software engineering task that helps the project team members to identify the requirements, control the requirements, track the requirements and changes to requirements at any time as the project exceeds.

- The requirement management task starts with identification and each of the requirements is assigned a unique identifier.

- After the requirements finalization, the traceability table is developed. Traceability table relates the requirements to one or more aspects of the system or its environment. The traceability tables are used as requirements database and are useful in understanding how change in one particular requirement will affect other parts or aspects of the system.

- Following are some examples of traceability table :

- o **Features traceability table** observes product features and make sure that

- how it is closely related to customer requirement.
- **Source traceability table** is used to identify the source of each of the requirement.
 - **Dependency traceability table** observes the relationships among requirements.
 - **Subsystem traceability table** classifies the requirements as per the subsystem they govern.
 - **Interface traceability table** indicates the internal and external system interface relate as per the given requirement.

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

Table 4.11.1 : Generic traceability table

Syllabus Topic : Case Studies : MHC-PMS

4.12 Case Studies : Mental Health Care Patient Management System (MHC-PMS)

- In case of Mental health care patient management system (MHC-PMS), we illustrate following requirements :
 - Product requirements
 - Organizational requirements, and

- External requirements

- In **product requirements**, we can say that MHC-PMS will be available only on the normal working hours from Monday to Friday. Downtime for the system should not exceed more than one minute or 30 seconds.

- In **Organizational requirements**, the users of the MHC-PMS should pass through some authentication process in order to limit the access to the system. The authorization may be in the form of some passwords or some clinic chip-enabled identity cards.

- In **External requirements**, the system must provide some privacy to the patients.

- For the case of MHC-PMS, functional requirements can be listed as follows :

1. A user (in this case, it is patient), shall be able to search the appointments lists for all hospitals.
2. The MHC-PMS shall generate each day, for each hospital, a list of patients who are expected to attend appointments for that day.
3. Each staff member using the system shall be identified by his or her employee number. This employee code must be unique so that system can recognize him or her.

- All these functional requirements the facilities provided by the system. All these requirements are taken from requirements document.

- In actual practice, it is highly impossible to achieve consistency and completeness in requirements.

Design Engineering

Syllabus

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

Syllabus Topic : Design Engineering

5.1 Introduction to Design Engineering

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

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

Syllabus Topic : Design Process

5.2 Design Process

- It is process of designing the software to achieve the intended goal of the software.

Syllabus Topic : Design Quality

5.3 Design Quality

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

5.3.1 Quality of Design Guidelines

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

5.3.2 The Quality Attributes

SPPU – Dec. 12

University Question

Q. Explain the quality attributes, considered in software design. (Dec. 2012, 8 Marks)

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

- **Functionality** is an important aspect of any software system. It is generally evaluated by the feature set and capabilities of that software.
- Following are general features that a system is supposed to deliver.
- **Usability** is best evaluated by considering following important factors :

- o Human factors
- o Overall aesthetics
- o Consistency and documentation.

Reliability is assessed by measuring following parameters :

- o Frequency and severity of failure
- o Accuracy of output results
- o Mean-time-to-failure (MTTF)
- o Recovery from failure and
- o Predictability of the program

Performance is evaluated by considering following characteristics :

- o Speed
- o Response time
- o Resource consumption
- o Throughput and
- o Efficiency

Supportability collectively combines following three important attributes :

- o Extensibility
- o Adaptability
- o Serviceability

Most commonly, above three attributes can be better defined by the term maintainability. In addition to these attributes, following are some more attributes with which a system can be installed easily, and the problems can be found out easily :

- o Testability,
- o Compatibility, and
- o Configurability,

It also contains more attributes which are as follows :

- | | |
|-------------------|-----------------|
| o Compatibility | o Extensibility |
| o Fault tolerance | o Modularity |
| o Reusability | o Robustness |
| o Security | o Portability |
| o Scalability | |

Syllabus Topic : Design Concepts

5.4 Design Concepts

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

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

5.4.1 Abstraction

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

5.4.2 Architecture

SPPU – May 13

University Question

- Q. Explain the following design concept :
Architecture. (May 2013, 2 Marks)

- Software architecture means the complete structure of the software. In number of ways, this structure provides basis for conceptual integrity for a system.
- In its simplest form, the architecture is the complete structure or arrangement of the program components in such a way that they interact with each other. The data structures may be used by these

components. All these components are the building blocks of the overall structure of the application being developed.

- One goal of software design is to derive an architectural framework of a system. The architectural design can be represented using one or more of a number of different models.

5.4.3 Patterns

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

5.4.4 Modularity

SPPU – May 13

University Question

- Q. Explain the following design concept :
Modularity. (May 2013, 2 Marks)

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

Let there are two problems namely p_1 and p_2 , if

$$C(p_1) > C(p_2) \quad \dots(5.4.1)$$

it follows that,

$$E(p_1) > E(p_2) \quad \dots(5.4.2)$$

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

$$C(p_1 + p_2) > C(p_1) + C(p_2) \quad \dots(5.4.3)$$

- The Equation (5.4.3) shows that the complexity of a problem that combines two problems p_1 and p_2 is greater than the complexity when the problems are considered separately. Now consider the Equation (5.4.3) and the condition implied by Equations (5.4.1) and (5.4.2), it follows that :

$$E(p_1 + p_2) > E(p_1) + E(p_2) \quad \dots(5.4.4)$$

- This equation leads to a strategy called divide and conquer. It is always easy to solve a complex problem when it is divided in small sub-problems that are easily manageable..
- The result expressed in Equation (5.4.4) has important implications with regard to modularity and software. It is an argument for modularity. It is possible to conclude from Equation (5.4.4) that, if we subdivide software into smaller modules then the effort required to develop it will become considerably small.
- Referring to Fig 5.4.1 the effort (cost) to develop an individual software module does decrease as the total number of modules increases.
- For a given the set of requirements, the modules that can be broken into smaller manageable size. When the size of the problem is large, the number of modules

will be increased and the effort (cost) associated with integrating the modules also increases.

These characteristics are exhibited by using the following curve i.e. a total cost or effort curve shown in the Fig. 5.4.1. The M is number of modules that would result in minimum development cost, but there is no any proper and sophisticated method to predict M with assurance.

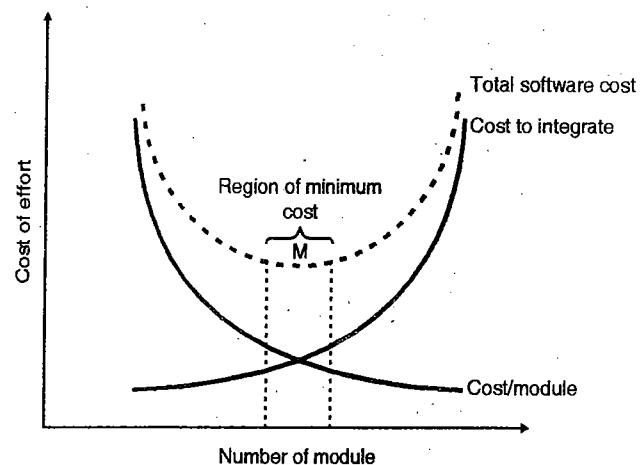


Fig. 5.4.1 : Modularity

- The curves shown in Fig. 5.4.1 provide some useful guidance when modularity is taken into consideration.
- We should divide the problem into modules, but utmost care must be taken to stay in the vicinity of M . It is always better to avoid under-modularity or over-modularity.
- There are five criteria defined for an effective modular system :
 1. **Modular decomposability** : A complex problem may be solved very easily by decomposing into smaller sub problems. The concept of modular solutions makes it very easy and effective.
 2. **Modular composability** : Different modules and components are integrated to produce a new system is called as modular composability.
 3. **Modular understandability** : While integrating a system, if each of the individual modules or components are

understood well, then it becomes very easy to build the system once again.

4. Modular continuity : It is always better to make changes module-wise instead of making changes system-wise. By doing this, the side effect of the changes can be minimized.

5. Modular protection

- o For any abnormal condition within a module, the effects are within that module only. Thus the side effects are minimized and problems will not spread outside that module. This phenomenon is called as modular protection.
- o After providing security to the modular design, it is also important to focus on rigidity in its implementation.
- o Any overhead caused by the sub programs is not acceptable. This overhead will affect the performance of the system.
- o In these situations, the products are designed using modularity and the coding of system is in-line.
- o The source code may not be seen modular initially but in-line code philosophy will be useful in the modular system.

5.4.5 Information Hiding

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

5.4.6 Functional Independence SPPU – Dec. 16

University Question

Q. What do you mean by the term cohesion and coupling in the context of software design? How are these concepts useful in arriving at a good design of a system ? (Dec. 2016, 2 Marks)

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

1. Cohesion
2. Coupling

- Cohesion is a relative functional strength of a module and coupling is the relative interdependence among modules.

1. Cohesion

- o Cohesion is an extension of the concept of information hiding. In cohesion, a module performs only one task within a software procedure i.e. requires only a little interaction with procedures that are in other parts of a program.
- o Cohesion can be represented as a "spectrum". High cohesion is always recommended but mid-range of the spectrum is often acceptable. The cohesion scale is nonlinear.
- o That is, low-end cohesiveness is worse than middle range, which is nearly same high-end cohesion.

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

2. Coupling

- o Coupling is a measure of inter-relationship among various modules in a software structure. Coupling always depends on the interface complexity among modules i.e. the point at which reference or entry is made to a module. It also depends on what data is passed across the interface among modules.

- o In software design, we look for minimum possible coupling. If the connectivity between different modules of software is made simple then it is bit easier to understand.
- o The simple connectivity will also avoid "ripple effect" up to certain extent. The ripple effect is introduced when the errors occurring at one particular location in the system and propagating through a system.

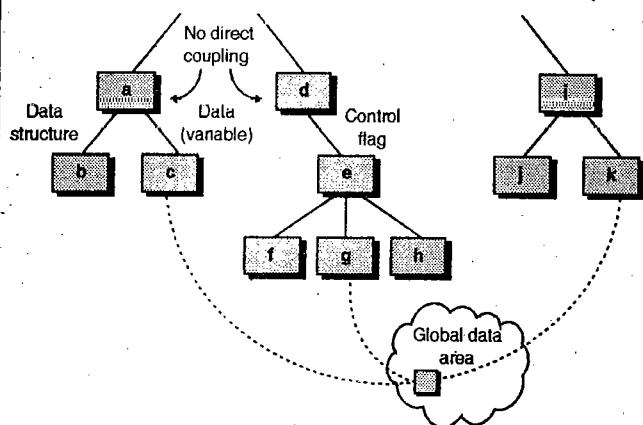


Fig. 5.4.2 : Types of coupling

- o In the Fig. 5.4.2, various examples of different types of module coupling are illustrated :
 - o Modules **a** and **d** are children of different modules. Each of them is not related to each other and thus there is no direct coupling occurs.
 - o Module **c** is child of module **a**. It is accessed through a conventional argument list. All the data are passed through this argument list.
 - o When the argument list is simple, the data can be passed and there is one to one correspondence between the items exist. The low data coupling or simply low coupling is demonstrated through this structure.



- When a part of data structure apart from simple argument, is passed through a module interface, then stamp coupling is exhibited. This is actually a variation of data coupling only. It is illustrated in the Fig. 5.4.2 and it occurs between **b** and **a**.
- The coupling is characterized by passage of control between modules at moderate levels. In most of the software designs, control coupling is common and is exhibited in Fig. 5.4.2 where a "control flag" is passed between modules **d** and **e**.
- When modules are tied to some environments that are external to software, then the high levels of coupling may occur. Like an I/O can couple a module to some devices, formats, and different communication protocols. This is external coupling and is very essential. It must be limited to only a small number of modules in a structure.
- Like low coupling, high coupling may also occurs when different modules reference a global data area. This is called common coupling also, and is shown in Fig. 5.4.2 modules **c**, **g**, and **k** share their data item in a global data area. The module **c** initializes the items.
- After that the module **g** recomputes and later on updates these items. Now assume that an error has been occurred and module **g** updates the item incorrectly.
- In processing the module, **k** reads the wrong item and during its processing, it fails and ultimately the software is aborted.
- The reason behind this abort is module **k** and the actual cause is module **g** since it has updated the item incorrectly.
- Diagnosing the problems is time consuming and difficult. The diagnosis in structures is always with the common coupling. The use of global data is not always bad, but the developer may use them in coupling. The designer should know well the impacts of these couplings and care against them to protect.
- The coupling occurs when a module uses the data present in some another module. It can use control information also during the coupling.
- The concept of content coupling used when all the subroutines or data structures within a module are compulsory asked to involve in coupling. The content coupling must be avoided.
- The content coupling modes may occur due to design decisions made during development of structure. The variants of the coupling discussed, may be introduced during coding phase.
- **For example :** Compiler coupling ties source code to specific (and often nonstandard) attributes of a compiler; Operating System (OS) coupling ties design and resultant code to operating system "hooks" that can create havoc when OS changes occur.

5.4.7 Refinement

SPPU - May 13

University Question

Q. Explain the following design concept :
Refinement. (May 2013, 3 Marks)

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

5.4.8 Refactoring

SPPU - May 15, Dec. 15

University Questions

- Q. What do you understand by refactoring ?
 (May 2015, 3 Marks)
- Q. Explain refactoring. (Dec. 2015, 3 Marks)

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

5.4.8.1 Importance of refactoring

SPPU - May 15, Dec. 15

University Question

- Q. Give the importance of refactoring in improving quality of software.
 (May 2015, Dec. 2015, 3 Marks)

- The refactoring is used in improving the quality of software.
- Following are some points that prove the importance of refactoring :
 - o By improving the quality of the code, working becomes easier. By using code refactoring, the addition of new code and maintenance of the code becomes easy.

- o The programmer do not write perfect codes, hence refactoring improves the code over the period of time.
- o Refactoring play important role in splitting long functions into reasonably small sub functions.
- o It helps in creating reusable codes.
- o Error handling is much easier and within control.

5.4.9 Design Classes

- Each of the software team should define a set of design classes. All of these classes should describe the elements of problem domain and that should focus the customer requirement.
- A design class should :
 - o Refine the analysis classes that are implemented.
 - o Create a new set of design classes that must support the business solutions.

5.4.10 Differentiation between Abstraction and Refinement

SPPU - Dec. 12

University Question

- Q. Differentiation between abstraction and refinement
 (Dec. 2012, 4 Marks)

Sr. No.	Abstraction	Refinement
1.	A description of something that omits some details that are not relevant called abstraction.	A detailed description that is even not relevant.
2.	It provides compact design process.	It allows more flexible design process.
3.	When we consider a modular solution to any problem, many levels of abstraction can be posed.	Stepwise refinement is a top-down design strategy.

Sr. No.	Abstraction	Refinement
4.	The highest level of abstraction generally states a solution by using the language of the problem environment.	A program is usually created by refining the levels of procedural detail.
5.	Similarly the lower levels of abstraction gives more detailed description of the solution. A data abstraction is nothing but a collection of data that explains a data object.	The hierarchy in refinement is created by partitioning a macroscopic statement of function i.e. a procedural abstraction. This partitioning is done stepwise till all the programming language statements are covered.

Syllabus Topic : The Design Model

5.5 The Design Model

- The design model can be viewed in two different dimensions as illustrated in Fig. 5.5.1.
- The process dimension indicates the evolution of the design model as design tasks are executed as part of the software process.
- The abstraction dimension represents the level of detail as each element of the analysis model is transformed into a design equivalent and then refined iteratively.
- In the Fig. 5.5.1, the dashed line indicates the boundary between the analysis and design models.
- The elements of the design model use many of the UML diagrams. These diagrams are refined and elaborated as the part of design.

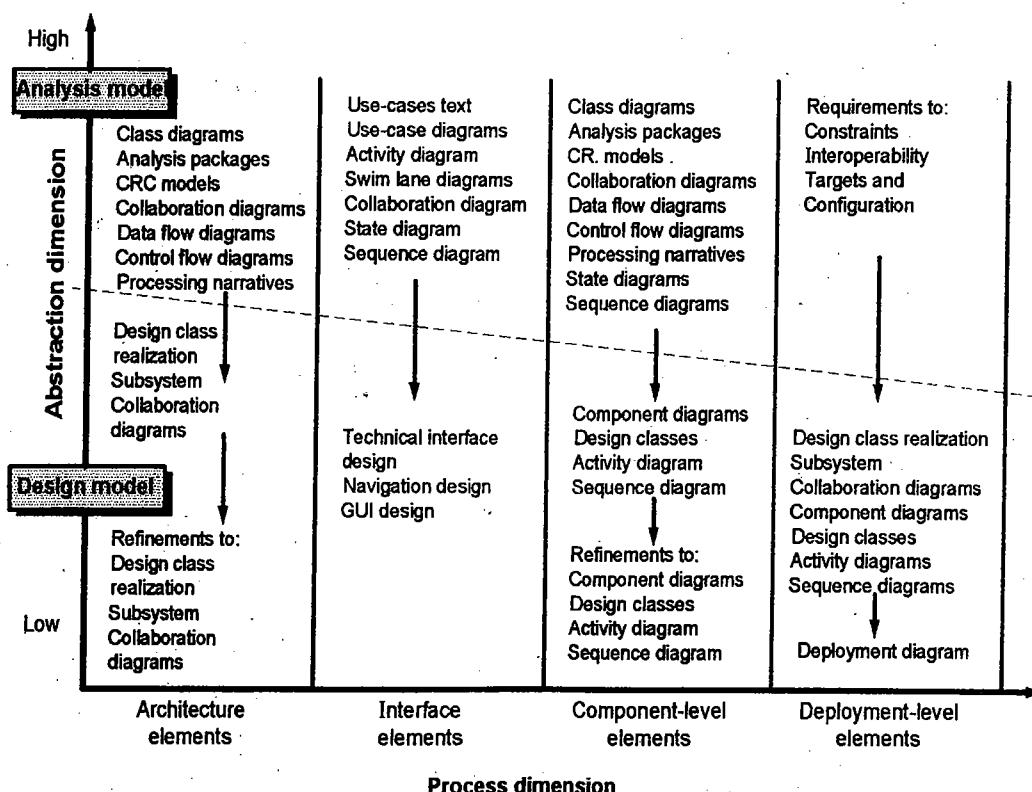


Fig. 5.5.1 : Dimensions of the design model

5.5.1 Data Design Elements

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

5.5.2 Architectural Design Elements

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

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

5.5.3 Interface Design Elements

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

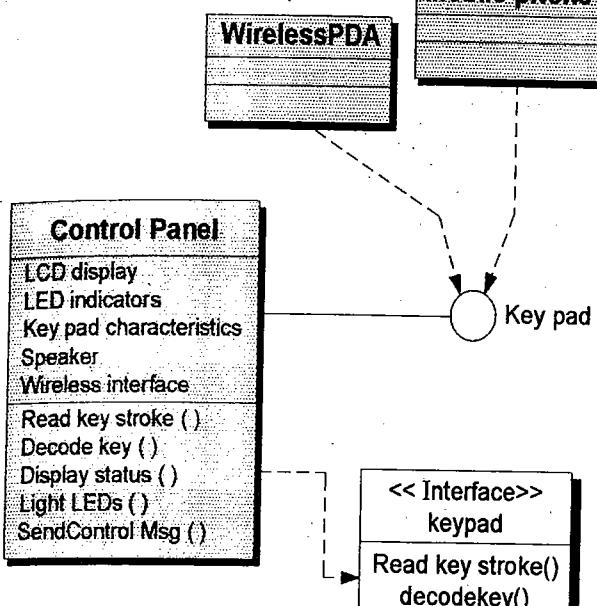


Fig. 5.5.2 : UML interface representation for control panel

5.5.4 Component-Level Design Elements

- If we consider the component-level design for software, then it is equivalent to the detailed drawings along with its specifications for each of rooms separately in a building.
- These drawings show various connections like electrical wiring and plumbing in each of the room separately. The location of switch board and other accessories in the room.
- The detailed design also describes which flooring is to be used, and which modeling is to be applied, and all the details related to room. Thus the component-level design for software completely explains the internal detail of each and every component of the software.
- In order to accomplish this, the component-level design gives the data structures for all local data objects along with algorithmic detail for all these processing that occurs within a component. It also elaborates the interface used to access to all component operations i.e. behaviours.



Fig. 5.5.3 : UML component diagram for sensor management

5.5.5 Deployment-Level Design Elements

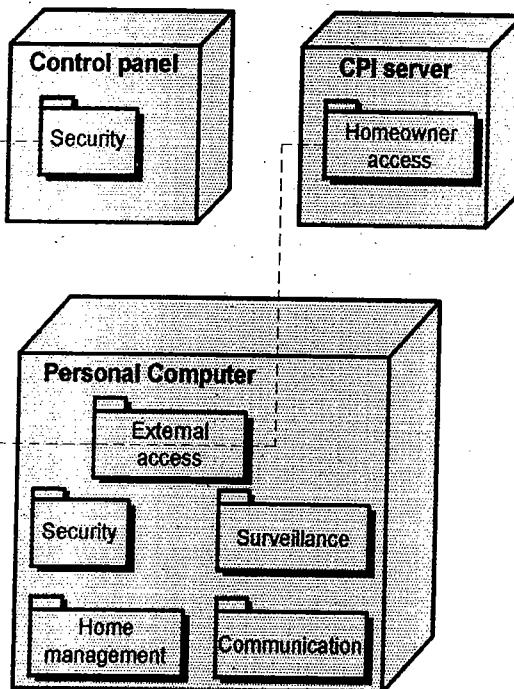


Fig. 5.5.4 : UML deployment diagram

- Deployment-level design elements indicate how software functionality and subsystems will be allocated within the physical computing environment that will support the software.
- During this design a UML deployment diagram is created as shown in the Fig. 5.5.4.
- It has three computing environment as shown below :
 - o The **personal computer** that implements security, observation and management.
 - o The **CPI server** manages the access rights.
 - o The **control panel** manages the security.

5.5.6 Translating Requirements Model to Design Model

- The software design is residing actually at the technical kernel of software development and engineering and it is applied irrespective of the software process model that is employed.
- Starting with the requirement analysis and modeling, the software design is the last activity or the last action and it is actually the platform for coding and testing.
- The elements of the analysis model give the information essential to create the four design models that are required for a full specification of design.
- The flow of information during software design is illustrated in Fig. 5.5.5. The design produces data or class design, an interface design, an architectural design, and a component design.
- The data or class design is translated from analysis class models to design class realizations and the appropriate data structures required to implement the software.
- Here more detailed class design is created as each of the software components is developed.
- The relationship between major structural elements of the software is defined by the architectural design. The design patterns

and the architectural styles can be used to achieve the requirements defined for the system.

- The interface design also describes the way software communicates with systems and with the end users who use it. An interface shows a flow of information.
- The component-level design translates structural components of the software architecture into a procedural description of software components.

5.5.7 Guidelines for the Data Design

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

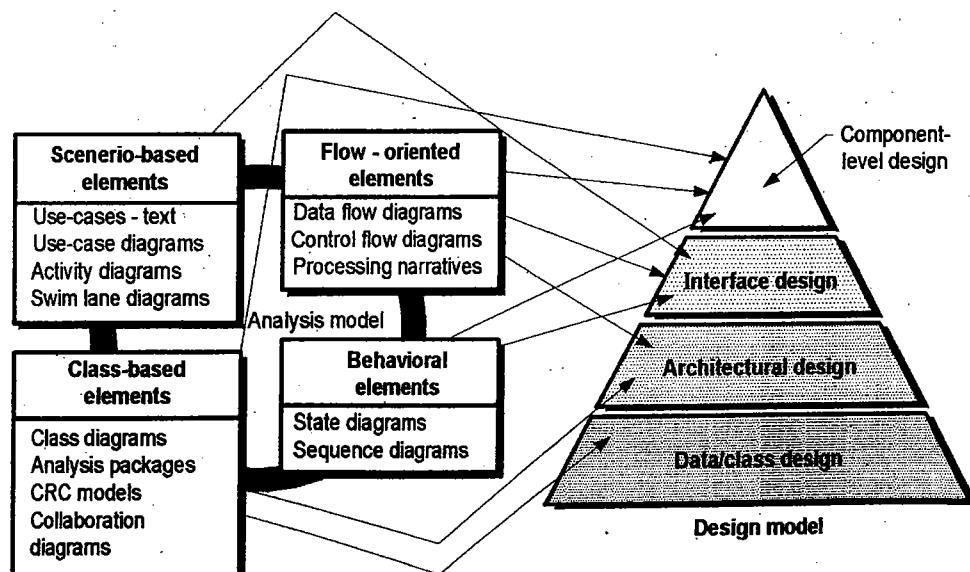


Fig. 5.5.5 : Translating the analysis model into the design model

Syllabus Topic : Pattern-Based Software Design

5.6 Pattern-Based Software Design

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

5.6.1 Describing a Design Pattern

- In software engineering, developer uses thousands of design patterns. For example an electronics engineer uses some extremely complex integrated circuit (IC) to design some new circuit. Also a mechanical engineer can use pulley-based design pattern to engineer some new machine.
- The design pattern can be described using some standard template. This template can be described using following attributes :
 - o **Pattern name** : The name should short, simple and effective to describe the design pattern, so that software developer should be able to use it quickly.
 - o **Synonyms** : It should provide list of synonyms.
 - o **Intent** : The intent of the design must be clear, what actually it does.

- o **Examples illustrated** : to motivate the use, illustrate the pattern by example.
- o **Structure** : It shows the classes required to implement the pattern.
- o **Responsibilities** : The responsibilities of the classes must be described.
- These are some examples of descriptions for a design pattern template. In addition to these attributes, collaborations, consequences, applicability etc. can also be mentioned.
- All these attributes actually represent the characteristics of the design pattern and any design pattern can be searched from the database using these attributes.

5.6.2 Using Patterns in Design

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

- o **Design patterns** : It focuses on some specific element of design such as aggregation of components, in order to solve the design problems or the relationships among different components. It also focuses on component to component communication.
- Each of the above mentioned patterns differ in the level of abstraction and differ in degree to which it provides the guidance for coding in this example.

5.6.3 Frameworks

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



Architectural Design

Syllabus

Architectural Design : Design Decisions, Views, Patterns, Application Architectures. Modeling Component level Design: component, Designing class based components, conducting component-level design

Syllabus Topic : Architectural Design

6.1 Introduction to Architectural Design

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

- In order to understand the system architecture, Fig. 6.1.1 provides a simple model of architecture for a remote control car :

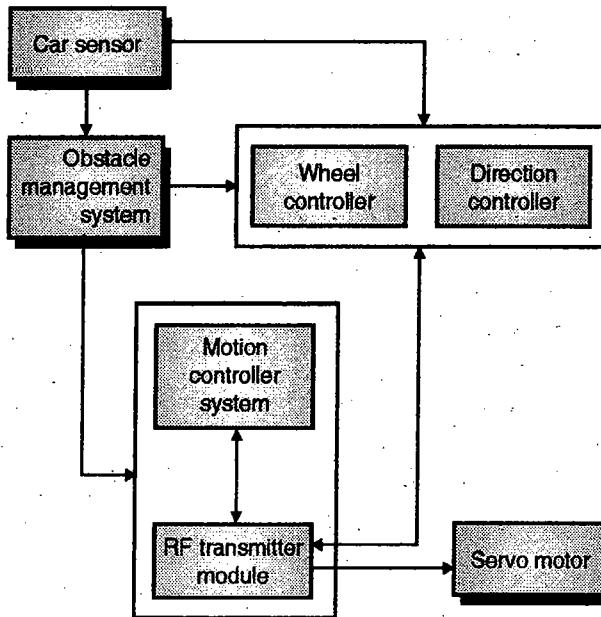


Fig. 6.1.1 : The architecture of a remote control car

The Fig. 6.1.1 exhibits the abstract model of architecture of a remote control car system that will help to understand the functioning of the car.

In the Fig. 6.1.1 shown, the interconnection between the components used to develop the remote control car system.

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

- o **Reusability at large-scale :** A model or the system architecture is considered as compact and manageable description of the system. For all the systems having similar requirements uses same system architecture and this is the reason why it can support reusability at large scale.

Syllabus Topic : Design Decisions

6.2 Architectural Design Decisions

- Architectural design is assumed to be an intellectual process and a software organization tries its level best to satisfy the functional requirements and non-functional requirements as well.
- All the activities present within the process depends on the type of the system under construction and specific requirement elicited by the customers. It also depends on the system architect previous experiences.
- Therefore the architectural design is not a sequence of activities rather it is assumed to be the series of decisions made.
- Thus the architectural design process requires number of structural decisions that have impact on the system and its development processes.
- Based on the past experiences of architectural designers, they have various questions about the system and are listed below :
 - o Is there any generic architecture present, so that it can be used as template for the system development ?
 - o How the components of the architecture can be decomposed into subcomponents?
 - o What architectural organization is best for delivering the non-functional requirements of the system?

- o Which strategy will be best suited to control the operation of various components of the system ?
 - o How the evaluation of architectural design is done ?
 - o How documentation is carried out ?
 - o What will be the basic methodology to design the system ?
 - o What are the architectural patterns that can be used ?
 - o How distributability feature is implemented ?
- Even if every software system is unique one, but the system with the same domain have similar design architectures.
- Therefore it is evident that software developer or the software designer decides what parts of the architectural design can be reused during the development process.
- For embedded systems and systems designed for personal computers, there is usually only a single processor and you will not have to design a distributed architecture for the system. However, most large systems are now distributed systems in which the system software is distributed across many different computers.
- The architecture of a software system may be based on a particular architectural pattern or style. An architectural pattern is a description of a system organization, such as a client-server organization or a layered architecture.
- Since there is a close relationship between non-functional requirements for a software system, the software architecture, architectural style and structure depends on the non-functional system requirements. We observe following non-functional requirements :
- o Performance

- Security
 - Safety
 - Availability
 - Maintainability
- Evaluating an architectural design is difficult because the true test of architecture is how well the system meets its functional and non-functional requirements when it is in use.
- However, you can do some evaluation by comparing your design against reference architectures or generic architectural patterns. The description of the non-functional characteristics of architectural patterns can also be used to help with architectural evaluation.

Syllabus Topic : Views

6.3 Architectural Views

- The architectural model of the system focuses on the functional and non-functional requirements and design. Also the architectural design is well documented for further design of any software system and implementation of the system in future.
- The architectural design can also be used as the foundation for the evolution of the system. Following are two important issues that are related to the architectural design :
 1. What views of the architectural design are actually useful in designing and documenting the system's architecture?
 2. What are the notations that will be used to describe the architectural model or architectural design of the system ?
- It is highly impossible to describe all the information about the architecture of the system in one single model since each of the model represent or show only one view of the system.

- Some of the models represent how a system is decomposed into different modules, some of them show how different processes interact each other during the run time, some of them might represent that how system components can spread across the network using distributed system.
- In all of the above case, all these views are extremely useful in different situations. Hence both the design and documentation must be represented using multiple views of the software architecture.
- There are different opinions as to what views are required in a well-known 4+1 view model of software architecture which suggests that there should be four fundamental architectural views, which are helpful in different scenarios.
- Software architecture takes into account the design and implementation of the software structure.
- From different architectural elements, the architecture is chosen to satisfy functionality and performance requirements. The non-functional requirements are also taken into consideration like scalability, reliability, availability and portability.
- Software architecture also deals with the style and aesthetics. In the Fig. 6.3.1 software architecture is illustrated as a model presenting five main views.

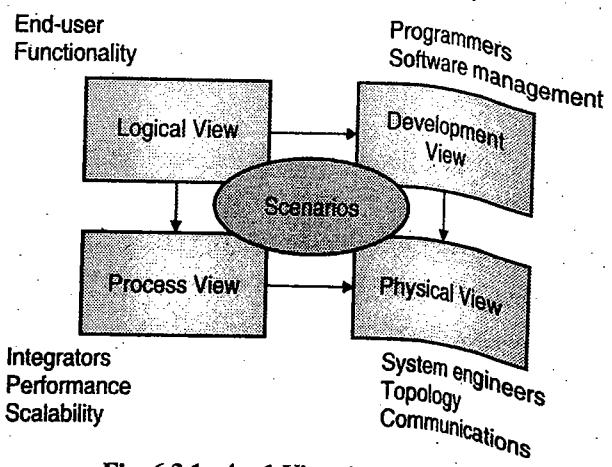


Fig. 6.3.1 : 4 + 1 View Architecture

- **The logical view**, which is the object model of the design (when an object-oriented design method is used),
- **The process view**, which captures the concurrency and synchronization aspects of the design,
- **The physical view**, which describes the mapping(s) of the software onto the hardware and reflects its distributed aspect,
- **The development view**, which describes the static organization of the software in its development environment.
- The description of architecture, the decisions made can be organized around these four views, and then illustrated by a few selected use cases, or scenarios which become a fifth view. The architecture is in fact partially evolved from these scenarios as we will see later.
- The “4 + 1” view model is quite “generic”: other notations and tools, other design methods can be used, especially for the logical and process decompositions, but we have indicated the ones we have used with success.
- In actual practice, the conceptual views are developed during the design process and are very much useful in supporting the architectural decision making.
- There is some form of communication in a system to various stakeholders of the software system. During the design process, as different scenarios come across, multiple views for those different scenarios or for different aspects of the system may also be developed.
- It is not possible at all to cover all the aspects of the system in a single description but it is possible to use architectural patterns or architectural styles for different views of a system.
- A software architect can also use the UML for architectural description. The UML was designed for describing object-oriented systems and, at the architectural design

stage, you often want to describe systems at a higher level of abstraction. Object classes are too close to the implementation to be useful for architectural description.

- Many researchers have proposed the use of more specialized architectural description languages (ADLs) in order to describe system architectures. The basic elements of ADLs are components and connectors, and they include rules and guidelines for well-formed architectures. However, because of their specialized nature, domain and application specialists find it hard to understand and use ADLs.
- Users of agile methods claim that detailed design documentation is mostly unused. It is, therefore, a waste of time and money to develop it. Most of the developers agree with this view and they think that, for most systems, it is not worth developing a detailed architectural description from these four perspectives.
- We should develop the views that are useful for communication and not worry about whether or not your architectural documentation is complete. However, an exception to this is when you are developing critical systems, when you need to make a detailed dependability analysis of the system. We may need to convince external regulators that your system conforms to their regulations and so complete architectural documentation may be required.

Syllabus Topic : Patterns

6.4 Architectural Patterns SPPU - Dec. 16

University Question

Q. Discuss architectural patterns in details.

(Dec. 2016, 6 Marks)

- The concept of patterns is well known for presenting, sharing, and reusing knowledge



about software systems. This is more commonly used nowadays. Architectural patterns were proposed earlier in the 1990s.

- An architectural pattern can be defined as a stylized, abstract description of good practices, which has been tried and tested on various systems and in different environments.
- Therefore, an architectural pattern describes a system organization that was already proved successful in various past systems.
- The knowledge of strengths and weaknesses of pattern must be documented properly and also there should be a proper documentation that in which situation, it is not proper to use the said pattern.

6.4.1 Software Architecture SPPU - Feb. 16

University Question

Q. What do you mean by software architecture ?
(Feb. 2016, 2 Marks)

- When we discuss the architecture of a building, it is the manner in which the various components of the building are integrated to form a complete structure.
- It is the way in which the building fits into its environment and meshes with other buildings in its vicinity.
- Also visual impact of the building, and the way textures, colours and materials are combined to create the external appearance and the internal living environment.
- Similarly we can define the software architecture concept. The software architecture of a program or computing system is actually the structures of the system that consists of following attributes:
 - o Software components,
 - o The externally visible properties of software components, and

- o The relationships among them.

- The architecture is basically not the operational software. But, it is a representation only that enables a software engineer to :

- o Analyze the effectiveness of the design.
- o Consider architectural alternatives, and
- o Reduce the associated risks.

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

Syllabus Topic : Application Architectures

6.5 Application Architectures

- Any software application is developed to satisfy the business or organizational needs. If we observe, then we conclude that all the businesses are common in their functioning.
- For example:
 - o Any business needs people to run it.
 - o It generates invoices for the sales.
 - o It keeps track of the accounts.
 - o It keeps track of the delivery of the products ordered.
 - o Facilitates online payments.
 - o Provides facility to accept the returns etc.
- The list is endless.
- Now consider the example of mobile phone companies. They provide following functionalities :

- o Connect to calls
 - o Manage the mobile towers and their networks.
 - o Generate bills for the customers.
- If we observe these two businesses, then we conclude that both the businesses have many things in common. This commonness actually triggers the development of software architectures that describes the structure and organization of particular types of software systems.
- The application architecture includes basic the characteristics of these systems or the class of the system and thus the common architectural structure can be reused when developing new systems of the same type.
 - The application architecture may be reimplemented when developing new systems but, for many business systems, application reuse is possible without reimplementation. For example, Enterprise Resource Planning (ERP) systems from companies such as SAP and Oracle, and vertical software packages (COTS) for specialized applications etc., in different areas of business. In these systems, a generic system is configured and adapted to create a specific business application.
 - Consider the example of supply chain management system providing different types of products. Any business involved in supply chain management system with some different products can use the same ERP for their business.
 - Thus a software designer can use the application model without any hesitation for the systems which share common business rules and policies.
 - There are some application systems that look very different but many of these superficially dissimilar applications actually have lot of common functionalities in them. Thus these types of dissimilar businesses can adapt the same application architecture.

- Following are good examples of such types of applications, we describe their architectures :

- o Transaction processing systems
- o Language processing systems

6.5.1 Transaction Processing Systems

- Transaction processing systems are developed to facilitate the transaction executed by the customers. It process customer requests for information from a database, or requests to update a database (e.g. **online banking systems**).
- Basically, a database transaction is a series of operations performed by the user and it is considered as one single transaction. All of the operations in a transaction have to be completed before the database changes are made permanent.
- The atomicity property of database system ensures that failure of the system or the failure of transaction should not lead to any inconsistencies in the database.
- Consider the very famous example of withdrawing money from ATM banking system that uses the following sequence of the operations :

 - o Get details of the customer's account from the database
 - o Check account balance
 - o Update the balance after withdrawal
 - o Send the command to release the requested cash

- Until and unless all these steps are completed successfully, the transaction is supposed to be incomplete and the account balance remains same as it was before the start of the transaction.
- Usually all transaction processing systems are interactive in nature where users make asynchronous requests for the services.

- Fig. 6.5.1 exhibits the conceptual architectural structure of transaction processing system for ATM banking system.

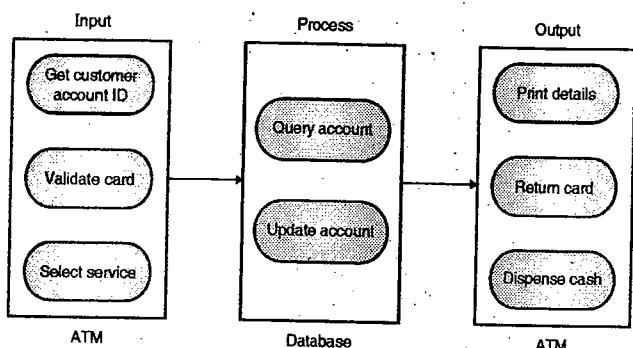


Fig. 6.5.1 : The software architecture of an ATM banking system

- The sequence of operations is illustrated in the Fig. 6.5.1 for the software architecture of an ATM banking system.
- The input and output components are implemented as software in the ATM and the processing component is part of the bank's database server. The architecture of this system shown, illustrates the functions of the input, process, and output components.

6.5.2 Language Processing Systems

- The main task of "language processing systems" translates a natural or artificial language into another language. Also programming languages may also process the sequence of instructions and produce the code.
- In software engineering, compilers translate an artificial programming language into machine code. Other language-processing systems may translate an XML data description into commands to query a database or to an alternative XML representation.
- Natural language processing systems may also translate one natural language to

another natural language e.g., English to Marathi or Hindi.

- The architecture for a language processing system for a programming language is explained in the Fig. 6.5.2. The source language instructions define the program to be executed and a translator converts these into instructions for an abstract machine.

The instructions are then interpreted by another component that fetches the instructions for execution and executes them using data from the environment. The output of the process is the result of interpreting the instructions on the input data.

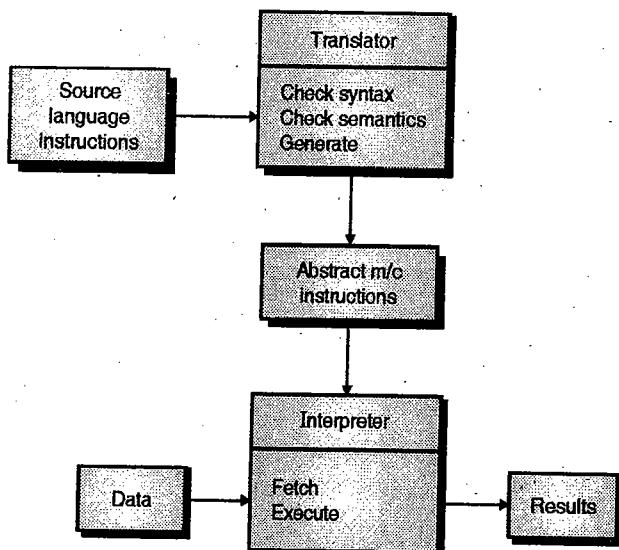


Fig. 6.5.2 : The architecture of a language processing system

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

of the language and builds a syntax tree.

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

The generic architecture for the compilers used in the programming languages shown in the Fig. 6.5.3 :

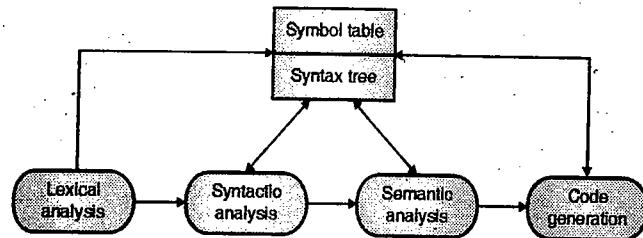


Fig. 6.5.3 : A pipe and filter compiler architecture

There are alternative architectural patterns that may be used in a language processing system. Compilers can be implemented using a composite of a repository and a pipe and filter model.

In compiler architecture, the symbol table is a repository for shared data. The phases of lexical, syntactic, and semantic analysis are organized sequentially, as shown in Fig. 6.5.3, and communicate through the shared symbol table.

This pipe and filter model of language compilation is effective in batch environments where programs are compiled and executed without user interaction; for example, in the translation of one XML document to another.

It is less effective when a compiler is integrated with other language processing tools such as a structured editing system,

an interactive debugger. In this situation, changes from one component need to be reflected immediately in other components. It is better, therefore, to organize the system around a repository, as shown in Fig. 6.5.4.

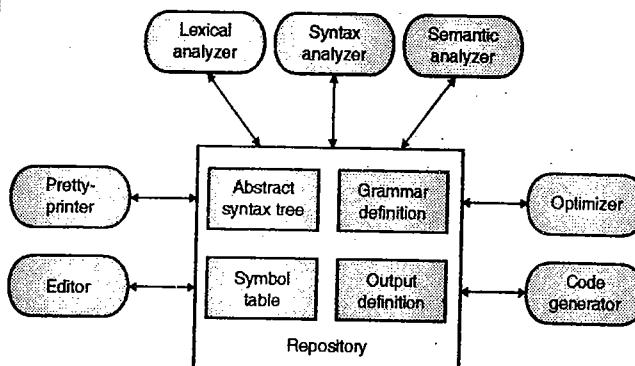


Fig. 6.5.4 : A repository architecture for a language processing system

This Fig. 6.5.4 illustrates how a language processing system can be part of an integrated set of programming support tools. In this example, the symbol table and syntax tree act as a central information repository. Tools or tool fragments communicate through it.

Other information that is sometimes embedded in tools, such as the grammar definition and the definition of the output format for the program, have been taken out of the tools and put into the repository.

Syllabus Topic : Modeling Component Level Design : Component

6.6 Modeling Component level Design

Component level design comes after architectural design is completed. It is possible to represent the component level design by using some programming languages these programs can be created by using the architectural design model.

- In fact component level design is an alternative approach to the architectural design approach.
- A component is the basic building block of a software system, it refers the OMG specifications and is defined as a modular and deployable part of a computer system and it consists of different sets of interfaces.
- Following are different views of a component design :

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

1. An object oriented view

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

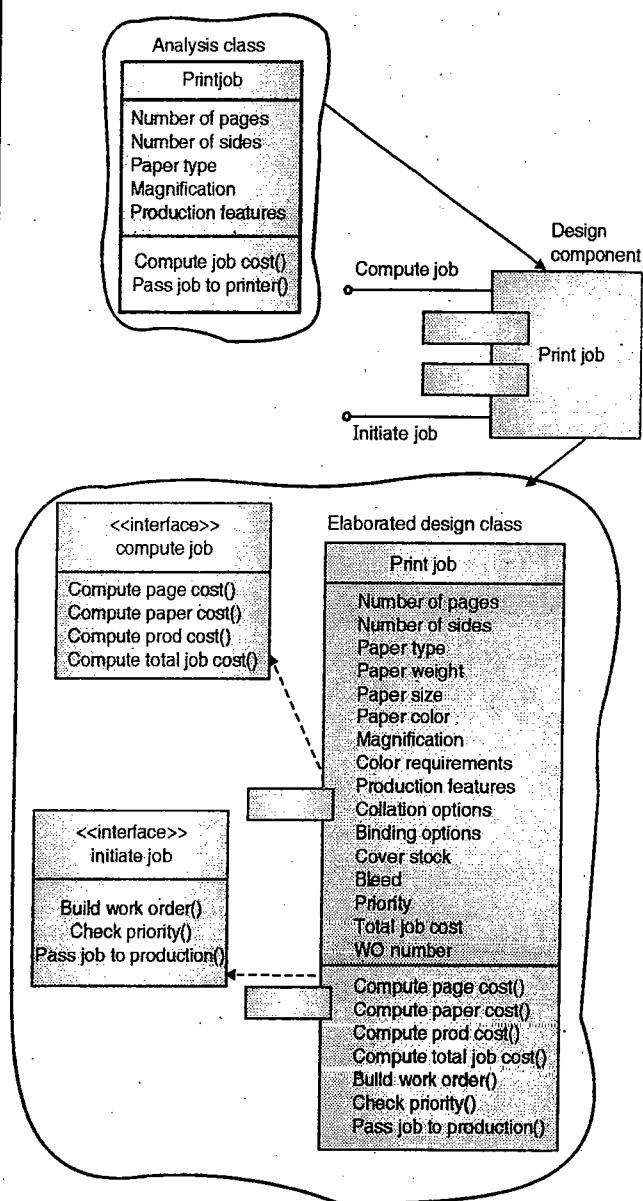


Fig. 6.6.1 : Elaboration of design component

2. The conventional view

- o In conventional method for software engineering, a component consist of a functional element of a program or a module that has processing logic, data structures required to implement the interfaces between the components.
- o To explain the process of design for conventional components we consider a sophisticated photo copying of analysis modeling discussed in architectural design.
- o In component level design each subsystem is illustrated in the Fig. 6.6.2.

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

3. The process-related view

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

Syllabus Topic : Designing Class based Components

6.7 Class-based Components

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

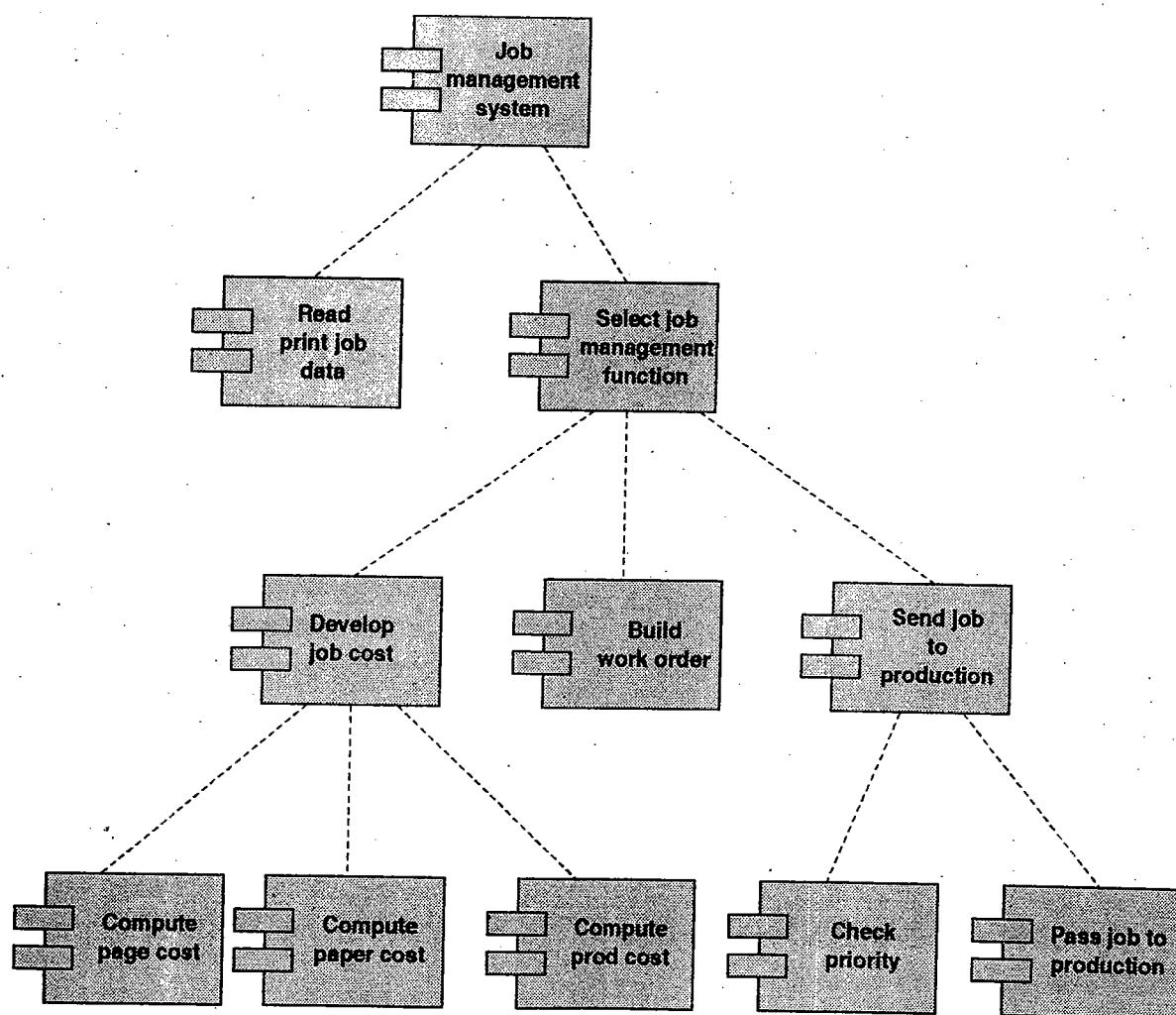


Fig. 6.6.2 : Structure chart for a conventional system



6.7.1 Basic Design Principles

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

1. The Open-Closed Principle (OCP) :

In OCP any module must be available for extension and modification. The designer should specify the component in such a way that it can be extended without any modification in the internal structure of the component.

2. The Liskov Substitution Principle (LSP) :

In this principle the subclasses should be substitutable for the base classes. This particular principle is suggested by Liskov. The LSP suggests that any class derived from the base class must imply the contract between the base classes and their components.

3. Dependency Inversion Principle (DIP) :

In DIP the design depends on abstractions and not on concretion. The abstractions are the place where design is allowed to extend without any further complications.

4. The Interface Segregation Principle (ISP) :

In this principle many client specific interfaces are better than general purpose interfaces. The component level design are organised into sub-systems or packages and suggests additional packaging principle for each component.

5. The Release Reused Equivalency Principle (REP) :

In this the reuse of each module is actually the release of module. The classes or components are designed for reuse. It is always better to make a group of reusable classes i.e. packages that are easy to manage.

6. The Common Closure Principle (CCP) :

In this principle the classes that changed together belongs together i.e. there is a cohesion between the classes.

7. The Common Reused Principle (CRP) :

In CRP the classes that are not reused together, they should not be group together also.

Syllabus Topic : Conducting Component-level Design

6.7.2 Conducting Component-Level Design

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

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

User Interface Design

Syllabus

User Interface Design : The golden rules, Interface Design steps & Analysis, Design Evaluation, Case Study : Web App Interface Design.

Syllabus Topic : User Interface Design

7.1 User Interface Design

- Success of most of the software depends upon user interface design.
- It is part of human computer interaction. In user interface human controls the software.
- As two different entities are trying to communicate with each other, one is human being and other is computer.
- We need to understand nature of human being designing user interface.
- Many factors are considered before designing user interface which includes type of user, age of user, education level, purpose of software, etc.
- User interface design is created by considering human type also like child, grown up person, senior citizen, educated person, non educated persons, etc.
- Good interface design is user friendly and user can communicate very easily with that software.
- User interface can be at the hardware and software level also.
- At hardware level, type of button provided can be considered as user interface. E.g. in washing machine, button provided, display provided can be considered as hardware user interface.
- At software level we can provide user interface using programming. Different menus, options, radio button, list boxes, text boxes, button, etc are provided at the software for user interface.
- If the software is used by illiterate person then user interface should be in terms of pictures and some visual effects.
- If user is disabled then we need to design interface accordingly. E.g. for blind users we need to provide interface using sound effects. Different biometric factors can be used for user interface design for disabled users.
- User convenience is given at most priority in user interface design.
- If interface is designed for mobile users then also we need to consider different factors like type of mobile, type of user, internet bandwidth, battery life, processing power, screen size, type of input, etc.
- It needs good understanding of user needs.

- In this topic we focus on user interface design of software and not of the hardware.
- Various aspects of interface design has been discussed. Different principles, methods are discussed.
- Plenty of examples are taken for better understanding.

7.1.1 Type of User Interface

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

- 1. Command Interpreter
- 2. Graphical User Interfaces

1. Command Interpreter

- o In command interpreter, user communicates with computer system with the help of commands.
- o In Unix/Linux system the command interpreter is known as shell. Shell is nothing but the interface between user and operating system.
- o All the shells give similar facility with only minor changes. Selecting the particular shell totally depends on user's requirement. Command interpreter just accept the commands, call the corresponding program and execute it.

- o When we type the "cd" command, operating system search the program for the "cd" command and get it executed. Many commands are provided by the operating system. The kind of command supported is depends upon the operating system.
- o The commands are implemented in different manner. In the first approach, the command interpreter itself contains the code for the command.
- o Whenever we type any command it will execute directly. The size of command interpreter depends upon the number of command supported by the command interpreter.
- o In the second approach separate program is created for each and every command supported by the system. Whenever we need to execute any command operating system search the program for it, load it into memory and get it executed.
- o The program for commands is stored on certain locations so that operating system can search it quickly. The approach is used in Linux system.
- o In Linux system all the programs for the command are stored in the "/bin" directory. In short system provides program for each and every command provided.
- o Executing the command means executing the corresponding program.

2. Graphical User Interfaces (GUI)

- o Another approach to interface with the computer system is through a user friendly graphical user interface or GUI.

- Instead of directly entering commands through a command-line interface, a GUI allows a mouse-based, window-and-menu based system as an interface.
- GUI gives the desktop environment where mouse and keyboard can be used. Mouse can be moved to certain location and when we click on some location operating system calls the corresponding program. Depending on the mouse pointer location, mouse click will invoke the program to execute. For every mouse click the corresponding program starts executing.
- GUI was initially used in the Xerox Alto computer. But the popularity of GUI is after 1980 in apple Macintosh operating system.
- Macintosh operating system had many changes in the initial GUI.
- Microsoft's first version of Windows-version 1.0 was based upon a cursor interface to the MS-DOS operating system.
- Two kinds of interfaces are available in UNIX system which is X-Windows systems and Common Desktop Environment (CDE).
- Various open source projects have made a significant development in GUI environment. The popular open source projects are K Desktop Environment (or KDE) and the GNOME desktop by the GNU project. The KDE or GNOME works on UNIX and Linux systems.
- Which method of interface to use is totally depends on the user preferences. Generally Unix/Linux user uses the command line interface as it provides very powerful collection of shell commands. Most of Windows user uses

the GUI environment which is very much user friendly one and even they are not aware about command based interface.

More interface types

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

7.1.2 Characteristics of Good User Interface

There are different characteristics of good user interface :

- Clarity
- Concision
- Familiarity
- Responsiveness
- Consistency
- Aesthetics
- Efficiency
- Attractiveness
- Forgiveness



7.1.3 Benefits of Good Interface Design

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

Syllabus Topic : The Golden Rules

7.2 The Golden Rules

SPPU – Dec. 13, May 14

University Questions

- Q. State and explain any four principles.
(Dec. 2013, 6 Marks)
- Q. What are the interface design principles and guidelines ?
(May 2014, 8 Marks)

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

7.2.1 Place the user in Control

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

- Define the interaction modes in such a way that the user is not forced to perform unnecessary or undesired actions :

- o Consider the example of word processor for spell check. If user selects spell check in its current interaction, then a word-processor menu should move it to a spell-checking mode.
- o The software should not force the user to remain in spell-checking mode if the user wishes to make some small text edit along the way. There should be less effort in entering and exiting spell check mode.

The interaction should be flexible

- o For different users, different interaction preferences must be provided. For example, the software should allow a user to interact with the system with different input devices like keyboard, mouse, a digitizer pen, or voice recognition commands.
- o It is not necessary that all the actions are supported by all means of input devices, but some actions supports only specific mechanism only. Consider drawing a shape using keyboard will be too difficult, but mouse will be a good option.

User interaction should be interruptible and undoable

- o Even for a long sequence of actions, the user must be able to interrupt the sequence and can perform some another task without losing any data or work.
- o There must be undo options available in the software.



- **The different levels of difficulty in interaction for different classes of users must be customized**
 - o The users find that they perform some sequence of interactions repeatedly. Instead of writing the same sequence of interaction again and again, it is always recommended to use a macro mechanism.
 - o The software should provide such a mechanism to facilitate interaction.
- **Hide complexities from the casual user**
 - o The main benefit of the user interface is that it takes its user to a virtual world while interacting. The software must support this feature.
 - o The design complexities like use of operating system, functions in file management, or other hidden complexities in computing technology.
 - o The interface should never require a user to interact at a level which is inside the machine. For example, the user never asked to type operating system commands in the application software.
- **The direct interaction with objects that appear on the screen**
 - o The user should have feel of using an object like a physical object. i.e. a design must be in such a way that user is able to perform all the function and do manipulation as he do in a physical object or physical thing.
 - o For example, an application interface allows its users to stretch or resize an object as it is a physical object i.e. scaling should be a direct implementation of direct manipulation.

7.2.2 Reduce the User's Memory Load

SPPU – Dec. 12

University Question

Q. What are the design principles for reducing the user's memory load in user interface design?
(Dec. 2012, 6 Marks)

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

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

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



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

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

7.2.3 Make the Interface Consistent

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

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

- **The system should maintain the consistency within a family of applications**

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

7.2.4 Necessity of a Good User Interface

SPPU - May 12

University Question

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

(May 2012, 3 Marks)

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

7.3 Shneiderman's 8 Golden Rules for UI Analysis

SPPU – May 12

University Question

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

(May 2012, 3 Marks)

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

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

1. Strive for consistency

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

2. Enable frequent users to use shortcuts

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

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

3. Offer informative feedback

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

4. Design dialog to yield closure

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

5. Offer simple error handling

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

6. Permit easy reversal of actions

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

contents should be easily able to retrieve. If some action is happened by mistake then undo option should be there.

7. Support internal locus of control

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

8. Reduce short-term memory load

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

7.4 Interface Analysis and Design Models

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

7.4.1 Interface Analysis and Design Models

- Following are four different models generated while analyzing and designing user interface :

- o **User model** is established by a human engineer or the software engineer,
- o **Design model** is created by the software engineer,
- o **User's mental model or the system perception model** created by end-user.
- o **Implementation model** is created by the implementers of the system.

- But each of these models may differ significantly in actual practice.

- This is the role of an interface designer is to patch these differences and derive a consistent representation of the interface.

- The user model establishes the profile of end-users of the system. In order to build a good user interface, the design must start with an understanding of intended users of the system and considering their profiles including age, sex, education, cultural, physical abilities, ethnic background, motivation, goals and personality etc. In addition to this, the users can be categorized as follows :

- o **Naive Users** : User having no technical and syntactical skills.
- o **Knowledgeable and intermittent users** : The users having semantic knowledge of the application but relatively weak in syntactic information.
- o **Knowledgeable and frequent users** : Users having good semantic and good syntactic knowledge.

A design model of the complete system includes the data, architectural interface, and detailed procedural representations of the software. The requirements specification establishes different constraints that will help to define the user.

7.4.2 User Interface Design Process

SPPU – Dec. 13, Dec. 14, Feb. 16, May 16, Dec. 16

University Questions

Q. Explain the user interface design process.

(Dec. 2013, Dec. 2014, 8 Marks)

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

(Feb. 2016, May 2016, 6 Marks)

Q. Describe the User Interface analysis and design process with diagram and explain interface design element. (Dec. 2016, 5 Marks)

- The analysis and design process for user interfaces is iterative and can be represented using a spiral model.
- Referring to Fig. 7.4.1, the user interface analysis and design process encompasses four distinct framework activities :
 1. User, task, and environment analysis and modeling.
 2. Interface design.
 3. Interface construction (implementation).
 4. Interface validation.

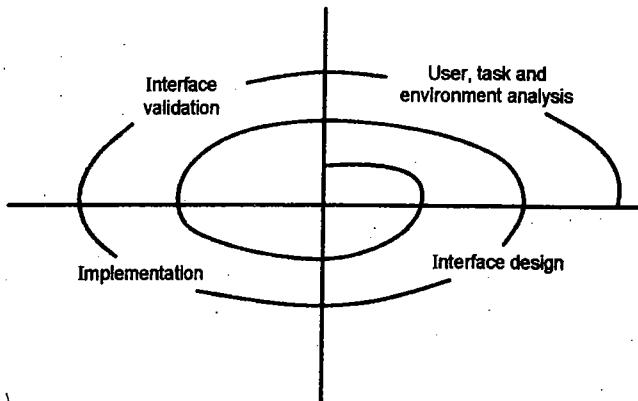


Fig. 7.4.1 : The user interface design process

- The spiral shown in Fig. 7.4.1 implies that each of these tasks will occur repeatedly along with each of the passes of spiral to represent the additional explanation of requirements and the final design.
- In many cases, the construction activity includes the prototyping that is the only practical way to validate the design.
- Interface analysis always focuses on the profile of the users. These are categorized as : skilled users, business level users etc and the class of users having general capabilities are recorded.

- Once general requirements have been defined, a more detailed task analysis is conducted. These tasks are identified and are performed by the user to reach the goals of the system.
- The analysis of the user environment focuses on the physical work environment.
- The information gathered as part of the analysis activity is used to create an analysis model for the interface. Using this model as a foundation, the design activity starts.
- The ultimate goal of interface design is to define interface objects and their actions that will help user to perform all defined tasks.

Syllabus Topic : Interface Design Steps and Analysis

7.5 Interface Design Steps and Analysis

SPPU – Dec. 12, Dec.16

University Questions

Q. Explain the user interface design steps. (Dec. 2012, 8 Marks)

Q. Describe the User Interface analysis and design process with diagram and explain interface design element. (Dec. 2016, 5 Marks)

- Once the interface analysis is completed, all the tasks (or objects and actions) required by the end-user have to be identified in detail, and the interface design activity starts. Interface design, is an iterative process like all other software engineering design processes.
- Each of the user interface design step occurs number of times, each elaborating and refining information developed in the previous step.
- However so many user interface design models have been proposed, all of them suggest some combination of the following steps :



1. Using information developed during interface analysis; define interface objects and actions (operations).
 2. Define events (user actions) that will cause the state of the user interface to change. Model this behaviour.
 3. Depict each interface state, as it will actually look to the end-user.
 4. Indicate how the user interprets the state of the system from information provided through the interface.
- In some cases, the interface designer may begin with sketches of each interface state (i.e., what the user interface looks like under various circumstances) and then work backward to define important design information like objects, actions etc.
 - Regardless of the sequence of design tasks, the designer must always follow the rules for user interface design.

7.5.1 Applying Interface Design Steps

- The most important step in interface design is defining the interface objects and the actions that are applied to them. To fulfil this use-cases are parsed.
- That is, a description of a use-case is written. Nouns (objects) and verbs (actions) are isolated to create a list of objects and actions.
- Once the objects and actions are defined and elaborated iteratively, they are classified by their types. Later on the target objects, source objects, and application objects are identified.
- The source object is dragged and dropped onto a target object. For an example the report icon is dragged and dropped onto a printer icon. The outcome of this action is to create a hard copy of report.

The application objects represent the data that are not directly modified as part of screen interaction. For example, in a mailing list, the names are stored for mailing. The list can be sorted or merged but it can not be dragged and dropped through user interaction.

When all the objects and actions are defined then the screen layout is performed. The screen layout is an interactive process in that graphical design, icons and definition of descriptive screen text, titles of windows and specification is conducted.

7.5.2 User Interface Design Patterns

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

7.5.3 Interface Design Issues

SPPU – Dec. 14, Apr. 17

University Questions

- Q. What do you mean by application accessibility and internationalization in user interface design? (Dec. 2014, 4 Marks)
- Q. Explain in detail the user interface design issues. (Apr. 2017, 5 Marks)

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

- **Response time :** Response time is one of the performance attributes of all the systems. This is the main complaint for many interactive applications. Generally, the system response time is measured from the point when user presses the enter key to the point when the software responds with the expected output or expected action.
- **Help facilities :** The users of the system require help every now and then. In short, in all of the software system, the help facility or the user manual is essential for the smooth use of software.
- Some of the software provides on-line help facilities also to get the problem solved without closing the application or the interface.
- **Error handling :** The large number of error messages and warnings are actually irritating and produces unnecessary hindrance in the operation. Until and unless the error messages or warnings are very critical, it should not be popped up. The critical messages like "Do you want to delete all data ?" is essential to be included. Thus increased use of error messages and warnings will increase the frustration of users.
- **Menu and command labelling :** Earlier the typed command were very common but now-a-days, the common mode of interaction point and pick interfaces. It has reduced the trust on typed commands. In most of the software, in addition to typed command, menu labels are used as a common mode of interaction.
- **Application accessibility :** The computing applications are used everywhere and the developer should take care in designing the user interface and it should include easy access for all the needs of users. The physically disabled people

should also be able to use the application in an easy way.

Internationalization : The software developers should take into consideration the design in such a way that it is useful for end-users from all around the world. The system must support the languages spoken by all these users. The user interface must be designed for generic use.

7.5.4 Interface Design Evaluation

- After development of an operational user interface prototype, the evaluation process to assess whether the system satisfies the need of the user.
- Evaluation can be an informal "test drive," in that a user gives feedback to a formally designed study that uses statistical methods for the evaluation of questionnaires completed by a population of end-users.
- In the following Fig. 7.5.1, the interface evaluation cycle is exhibited :

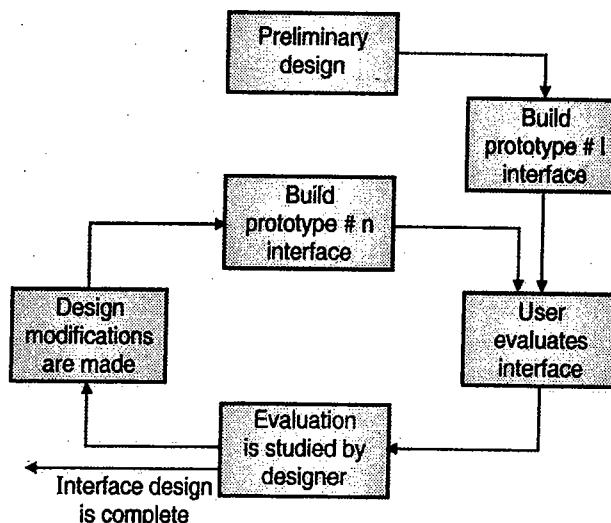


Fig. 7.5.1 : The interface design evaluation cycle

- After completion of the design model, a first-level prototype is created. This prototype is tested by the user, who provides the designer with direct feedback about the power of the interface.
- The prototyping approach is effective, but it is not possible to evaluate the quality of a user interface before a prototype is built.

- If potential problems can be detected and fixed in the early stages, the number of loops through the evaluation cycle will be minimized and development is faster.
- Once a design model of the interface is ready, the evaluation criteria can be applied.
- After development of the first prototype, the developer collects a variety of qualitative and quantitative data that will help in assessing the interface.
- Questionnaires should be distributed to users to collect qualitative data related to prototype.

Syllabus Topic : Design Evaluation

7.6 Design Evaluation

- In design evaluation, a prototyping model is created and it is handed over to user to determine whether it is developed as per the requirement given by the client.
- The end user can only evaluate the product and check that it meets the requirement.
- After receiving the feedback from the user, it is studied well by the developer and if the feedback is valid then the modifications are done by the developer.
- This user interface evaluation is illustrated by the Fig. 7.6.1 :

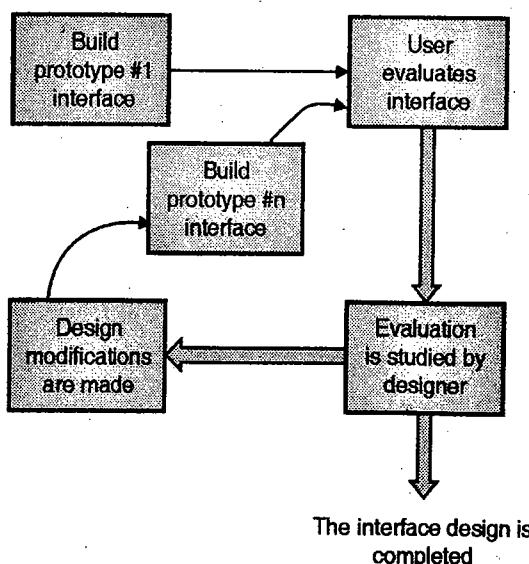


Fig. 7.6.1 : The interface design evaluation cycle

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

Syllabus Topic : Case Study : WebApp Interface Design

7.7 WebApp Interface Design

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

7.7.1 WebApp Design Principles

SPPU – May 12, Apr. 17

University Questions

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

(May 2012, 5 Marks)

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

(Apr. 2017, 5 Marks)

Following are some WebApp design principles :

- Anticipation means it should let the user know its next step or next move.
- Communication let the user know the status of any activity initiated by him through some text message or through some image.

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

□□□

Project Management Concepts

Syllabus

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

Estimation : Observations on Estimation, Project Planning Process, Software Scope and feasibility, Resources: Human Resources, Reusable software, Environmental Resources. Software Project Estimation, Decomposition Techniques, Empirical Estimation Models: Structure, COCOMO II, Estimation of Object-oriented Projects, Specialized Estimation

Case Study : Software Tools for Estimation.

Syllabus Topic : The Management Spectrum

8.1 The Management Spectrum

SPPU - May 12, May 13, May 15

University Question

Q. Explain the role of people, product and process in project management.

(May 2012, May 2013, May 2015, 8 Marks)

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

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

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

Syllabus Topic : People

8.1.1 The People

SPPU - May 12

University Question

Q. Explain the term the people of management spectrum. (May 2012, 4 Marks)

- Software engineering institute has developed model named the People Management Capability Maturity Model (PM-CMM). The PM-CMM has been developed to deploy the talent needed to improve their software development capability.
- Not only to deploy talent but also to enhance the readiness of software organizations to undertake increasingly complex applications by helping to grow, attract, motivate the talent needed to improve their software development capability and to continue to hold the talent to improve the development capabilities.

- Also helping to retain the talents that are needed to improve the development capabilities.
 - The PM-CMM defines the following key practice areas for software people :
 - o Recruiting
 - o Selection
 - o Performance management
 - o Training
 - o Compensation
 - o Career development
 - o Organization and work design
 - o Team or culture development
 - The PM-CMM is very close to the software capability maturity model integration. It guides organization in creation of a mature software process.
 - There are various groups that are involved for the most needed communication and co-ordination issues required for the effective software. All these groups can be categorized as under :

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

8.1.1.1 Stake Holders

SPPU- May 13,May 14

University Questions

- Q.** Explain the term in brief: Stakeholders.
(May 2013, 2 Marks)

Q. What are the categories of stakeholders? What are the characteristics of effective project manager?
(May 2014, 8 Marks)

maximizes each person's skills and capabilities. This is the job of the team leader that looks into each and every corner of the process. The stakeholders can be categorized into one of five categories required :

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

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

8.1.1.2 Team Leaders

- Competent practitioners often make poor team leaders because project management is a people-intensive activity. But unfortunately in many cases individuals just fall into a project manager role and become the project managers accidentally.

- The **MOI** (Model of Leadership) is used by the project managers that has the following characteristics to make a project manager very effective:

- **Motivation** : The ability to encourage the team to produce their best ability.

Stake holders are the people who are directly or indirectly involved in the software process and software project. To make the effective development process, the project team must be organized in such a way that

- **Organization :** The ability to organize the ongoing processes in such a way that will help the initial concept to mould into a final product.
- **Ideas or innovation :** The ability to encourage the team to create the innovative ideas let them feel to be more creative in their tasks.
- All the successful project leaders apply a problem solving management style i.e. a software project manager should be able to concentrate on understanding the problem and finding the solution, managing the flow of ideas and thoughts. At the same time, manager should let everyone in the team to know that quality is important parameter and can not be compromised.
- One other view of the characteristics that defines an effective project manager should emphasize following four key factors :
 - **Problem solving :** A project manager should find out technical and organizational issues.
 - **Managerial identity :** A project manager should take the charge of the project. He should have overall control and should allow only good technical team members to do their own styles.
 - **Achievement :** To enhance the productivity of a project team, he should take initiative and demonstrate the function through his own actions.
 - **Influence and team building :** A project manager should have the ability to read the faces and capability of his team members i.e. face reading.

8.1.1.3 Software Team

- The people those are directly involved in a software project are within the project manager's scope. The structure of a good team depends on the management methodology adopted in the organization,

the number of people who will inhabit the team and their skill levels, and the overall complexity of the question.

- Following are seven project factors that must be considered when planning the structure of software engineering teams :

- The ability to solve the difficulty of the problem.
 - The overall size of the final programs in lines of code.
 - The amount of time that the team will stay together.
 - The degree of modularity.
 - The quality and reliability of the system.
 - The delivery date rigidity.
 - The degree of communication required for the project.
- To achieve a high-performance team :
- Team members must have trust in one another.
 - The distribution of skills must be appropriate to the problem.
 - An unorthodox or independent-minded person may have to be excluded from the team, if team unity is to be maintained.

- The aim and idea for every project manager is to create a team that exhibits unity among the team members.

8.1.1.4 Agile Teams

The term agile means very active. As a cure to many of the problems that have afflicted software project work, in recent years, agile software development has been put forward. The agile philosophy provides the following :

- It encourages customer satisfaction by early incremental delivery of software.

- It provides small highly motivated project teams.
 - It consists of informal methods.
 - It also provides minimal software engineering work products.
 - It results in overall development simplicity.
- The small sized and highly motivated project team, also called an agile team, adopts many of the characteristics of successful software project teams.

8.1.1.5 Co-ordination and Communication Issues

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

1. Scale

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

2. Uncertainty

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

3. Interoperability

- Interoperability is one of the important characteristics of many software applications. The new software should be able to communicate with the existing system and satisfy the predefined constraints and conditions implemented by the software.
- The following important characteristics of software are the facts of life :
 1. Scale
 2. Uncertainty
 3. Interoperability

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

Syllabus Topic : Product

8.1.2 The Product

SPPU - May 12

University Question

Q. Explain the term the product of management spectrum. (May 2012, 4 Marks)

- The product objectives and scope must be established before planning a project. All the technical and management constraints and difficulties should be identified and their alternative solutions should be considered.
- Without this information, it is impossible to define reasonable and accurate estimation of the cost, and conduct an effective assessment of risk and a realistic breakdown of project tasks.
- In order to define the product objectives and its scope, there should be proper coordination between the software developer and customer and they must meet.
- In most of the cases, this particular activity begins as part of the system engineering or business process engineering and continues

as the first step in software requirements engineering.

- Objectives identify the overall goals for the product. These objectives do not take into consideration, how these goals will be achieved. The scope identifies the primary data and functions and the behaviours that characterize the product, and more importantly, attempts to bound these characteristics in a quantitative manner.

Syllabus Topic : Process

8.1.3 The Process

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

Syllabus Topic : Project

8.1.4 The Project

- Planned and controlled software projects are conducted for one and only reason because it is the only way known to manage complexity.
- Project failure rate remains higher than it should be even though the success rate for software projects has improved. One of the reasons is that the customers lose interest quickly (because what they have requested was not really as important as they first thought), and the projects are cancelled.
- The software engineers and the software project manager must follow certain guidelines in order to avoid project failure:
 - o Heed a set of common warning signs
 - o Understand the critical success factors that lead to good project management and
 - o Develop a common sense approach for planning, monitoring, and controlling the project.

Syllabus Topic : The W5HH Principle

8.2 The W5HH Principle

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

Why is the system being developed ?

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

What will be done ?

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

When will it be done ?

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

Who is responsible for a function ?

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

Where they are organizationally located ?

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

How will the job be done technically and managerially ?

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

How much of each resource is needed ?

The answer to this question is derived by developing estimates.

W5HH principle is applicable regardless of the size or complexity of a software project. The questions noted provide an excellent planning outline for the project manager and the software team.

Syllabus Topic : Metrics in the Process and Project Domains

8.3 Metrics in the Process and Project Domains

SPPU - May 15, Dec. 15

University Question

Q. Explain in detail software process and project metrics. (May 2015, Dec. 2015, 9 Marks)

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

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

8.3.1 Process Metrics

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

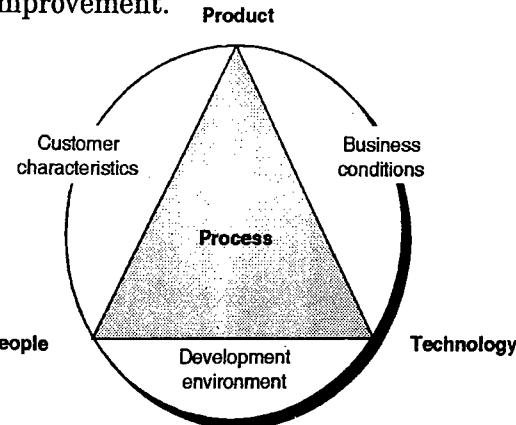


Fig. 8.3.1 : Determinants for software quality and organizational effectiveness



- In the Fig. 8.3.1, the process resides at the center of a triangle. The three corners of the triangle are actually the factors that have a profound impact on software quality and the organizational performance. They are:
 - o Product
 - o People
 - o Technology
- The skill set and motivation of the team members are shown to be the most significant factor in quality and performance. The other parameters like complexity of the product, also has a greater impact on quality and team performance. And finally the technology will also have some impact.

Syllabus Topic : Metrics for Project

8.3.2 Project Metrics

- Actually the project metrics are tactical ones i.e. the project metrics and the indicators derived from them are generally used by a project manager and a software team in order to adapt workflow and various activities.
- The very first application of metrics on various software projects is observed as estimation.
- With the progress of a project, the measures of effort and actual time spent are compared to the original estimates and schedule.
- As soon as technical work begins, other significant project metrics may also be taken into consideration.
- In addition to these project metrics, the errors detected during each task are also tracked.
- The intent of project metrics is twofold :
 - o First, to avoid delays and mitigate potential problems and risks.

- o Secondly, the project metrics are used to assess the product quality on an ongoing basis and whenever necessary can modify the technical approach to improve quality.

Syllabus Topic : Software Measurement

8.4 Software Measurement

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

Syllabus Topic : Software Measurement : Size and Function Oriented Metrics (FP and LOC)

8.4.1 Size-Oriented Metrics

SPPU- May 14

University Question

Q. Explain size oriented metrics.

(May 2014, 4 Marks)

- Size-oriented software metrics are derived by normalizing quality and productivity measures. This metric also consider the size of the software product. If any software organization keeps the simple records of software development process, then a table

containing the size-oriented measures can be created. The example of such a table is shown in Fig. 8.4.1.

Project	LOC	Effort	\$ (000)	Pp. doc.	Errors	Defects	People
Alpha	12,100	24	168	355	134	28	3
Beta	27,200	62	440	1224	321	66	5
Gamma	20,200	43	314	1050	258	64	6
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Fig. 8.4.1 : Size-oriented metrics

- This table has the list of each software development projects that are already completed over the past few years and it keeps the record of their corresponding measures for project.
- To develop metrics for a project that has the similar metrics compared to other projects, the developer can select the lines of codes as a standard normalization value.
- In fact, the size-oriented metrics are not considered as foolproof metrics and it is globally not accepted as the best metrics for software process.

8.4.2 Function-Oriented Metrics (FP and LOC)

- The second important metric is function-oriented software metrics that use an amount of the work delivered by the application as a standard normalization value. The most commonly used function-oriented metric is Function Point (FP). The computation of the function point depends on the information domain and complexity of the software project.
- FP is function oriented software metric that is programming language independent. It is used in various applications that are using conventional and nonprocedural languages. It depends on the data known in the

beginning of the project development or early evolution of the project. These are the reasons why FP is a good and attractive estimation approach.

- The computation of FP depends on subjective data rather than objective data. Later on, it may be difficult to keep the counts of the information domain and FP will be a simple number and it would have no physical meaning.
- LOC metric is a software metric used to measure the size of a computer program by counting the number of lines in the text of program's source code.
- LOC is typically used to predict the amount of effort that will be required to develop a program.

8.4.3 Reconciling LOC and FP Metrics

- The Lines of Code (LOC) and Function Points and their relationship are based on the programming language used to implement the software. The relationship also depends on the quality of the design. Various researches have been observed in the past that tried to relate LOC and FP measures.
- Table 8.4.1 provides a rough estimates of LOC required to develop a FP in different programming languages.

Table 8.4.1 : LOC per function point

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

Programming language	Average	Medium	Low	High
FoxPro	32	35	25	35
Informix	42	31	24	57
Java	63	53	77	-
JavaScript	58	63	42	75
JSP	59	-	-	-
Lotus Notes	21	22	15	25
Oracle	30	35	4	217
PeopleSoft	33	32	30	40
Perl	60	-	-	-
Powerbuilder	32	31	11	105
Smalltalk	26	19	10	55
SQL	40	37	7	110
Visual Basic	47	42	16	158

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

8.4.4 Object-Oriented Metrics

- The above discussed conventional software project metrics (i.e. LOC or FP) are also used to estimate the object-oriented software projects. The problem with these approaches is that they do not provide schedule, effort and adjustments required.
- Following are the sets of object-oriented metrics for the object-oriented projects suggested :
 - o **Number of scenario scripts** : The scenario script is actually a detailed sequence of steps used in interaction between the end-user and the product.

- o **Number of key classes** : The key classes are highly independent components defined in object-oriented analysis. These key classes sit in the centre of the problem domain.
- o **Number of support classes** : The support classes are used in system implementation but they are not directly involved in the problem domain. The examples of support classes are : User interface classes, database access classes and database manipulation classes.
- o **Average number of support classes per key class** : Normally the key classes are known in the beginning of the project itself. The support classes are defined throughout the development process. If the average number of support classes per key class is known in a problem domain then the estimation would be much simpler.
- o **Number of subsystems** : Actually a subsystem is an aggregation of keys class and its support class to define a function that is visible to the user. Once these subsystems are accurately identified, then it would be very easy to make a schedule and the related work among project staff.

8.4.5 Integrating Metrics within the Software Process

- It is a survey that most of the software developers do not go for software measurement. Here we present certain important points that will depict the significance of software measurement.
- If developer do not measure, then it becomes too difficult to find out whether there is an improvement or not. Actually there is no any way to find out the

- improvement except software measurement. If there is no improvement, then the progress of the project is lost.
- During the development phase of a project and at technical level, software metrics provides lot of benefits.
 - Once the development is completed then developer would like to find out answers for various questions like :
 - o Which of the requirements may change?
 - o How much testing should be done for each component ?
 - o Which of the components are prone to errors ? - If a developer has done software measurement in the beginning of development process, then it is very easy to find out the answers for all of these questions.
 - Thus we can say that software metrics acts as a technical guide for development process. It is always better to integrate metrics within the software process.
 - Also by establishing a baseline for metrics, the immediate benefits can be obtained from process, project, products i.e. at technical levels.
 - The **metrics baseline** consists of the history or the past data of various software development projects as shown in Fig. 8.4.1 (Size-oriented metrics).
 - The metric baseline must have following attributes for effective process improvements :
 - o All the data must be accurate and there should not be any blind guesses.

- o If possible, collect the data from various projects so that comparison becomes easy.
- o All the measures must be consistent.
- o The application under development should be similar to application estimated.

The process of establishing the metric baseline is exhibited in the Fig. 8.4.2.

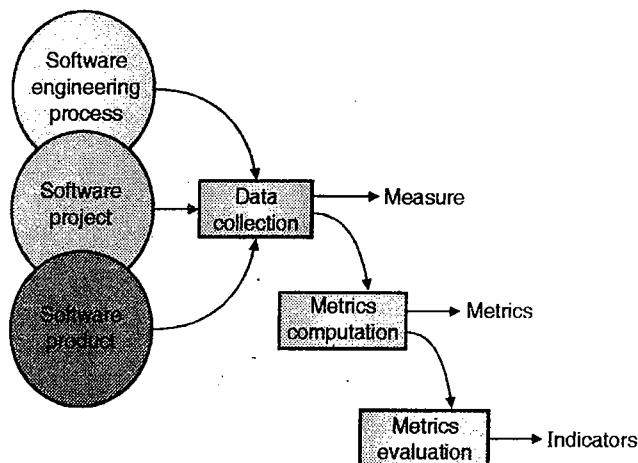


Fig. 8.4.2 : Establishing a baseline

Syllabus Topic : Software Quality

8.4.6 Software Quality

SPPU - May 14, Dec. 14, May 15, Dec. 15, May 16

University Questions

- Q. What are the software quality factors ? Explain any four. **(May 2014, 12 Marks)**
- Q. Explain any four quality measures of software. **(Dec. 2014, 8 Marks)**
- Q. What is the need for Software Quality ? **(Dec. 2015, May 2016, 4 Marks)**
- Q. What is software quality ? **(May 2015, Dec. 2015, 4 Marks)**

- The most software developers will agree that high-quality software is an important goal. In the most general sense, software quality is conformance to explicitly stated, functional and performance requirements, explicitly documented development

- standards, and implicit characteristics that are expected of all professionally developed software.
- The definition emphasizes three important points :
1. Software requirements are the basis for software quality measurement. If requirements are not satisfied then obviously it will yield low quality software.
 2. For the development processes, some standards are specified that define the development criteria. These development criteria will guide software engineering process. If the criteria are not followed properly then the quality of the product will be quite low.
 3. There are two types of requirements :
 - o Implicit requirements and
 - o Explicit requirementsEven if the explicit requirements are considered, the quality may be degraded if it fails to meet implicit requirements.
- Software quality is a complex mix of factors that will vary across different applications and the customers who request them.
- Software quality factors are identified and the human activities required to achieve them are described as follows :
1. McCall's Quality Factors
 2. ISO 9126 Quality Factors
- #### 8.4.6.1 McCall's Quality Factors
- SPPU - Dec. 15, May 16
- University Question**

Q. Explain different McCall's quality factors.

(Dec. 2015, May 2016, 4 Marks)
- The factors that affect software quality can be categorized in two broad groups :

- o Factors that can be directly measured (e.g. defects uncovered during testing) and
 - o Factors that can be measured only indirectly (e.g. usability or maintainability)
- The categorization of factors that affect software quality shown in Fig. 8.4.3, Focus on three important aspects of a software product :
1. Its operational characteristics,
 2. Its ability to undergo change and
 3. Its adaptability to new environments

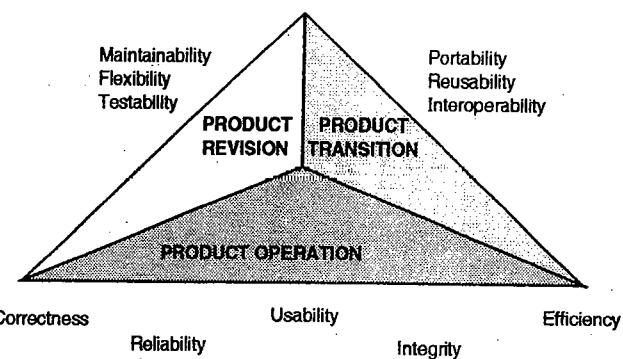


Fig. 8.4.3 : McCall's software quality factors

- Referring to the factors noted in Fig. 8.4.3, provide the following descriptions :

1. **Correctness** : The correctness is the quality factor that assesses the software for its correct functioning according to the specification and customer's requirements.
2. **Reliability** : It is the quality attribute that evaluates the software for its working according to the desired precision.
3. **Usability** : The software usability is defined as the degree to which the application software is easy to use. It defines the efforts taken to operate and learn, submit input and interpret output.

4. **Efficiency :** It is defined as the amount of various resources required to execute a given program. The computing resources are memory devices, processing devices and input output devices etc.
5. **Maintainability :** It is defined as the effort needed to uncover the errors and the problems in functioning of a program and resolve these errors.
6. **Integrity :** The integrity of a system is defined as the efforts taken to manage access authorization. The integrity of a system can be ensured by controlling all the accesses to it.
7. **Flexibility :** The ease with which a program can be modified and extended in future.
8. **Testability :** Amount of time and the efforts taken to test a program.
9. **Portability :** Shifting a program from one environment to another.
10. **Reusability :** How existing codes can be reused in future developments according to the scope of the software.
11. **Interoperability :** Ability to connect one system to the another system.

8.4.6.2 ISO 9126 Quality Factors

SPPU - May 12, Dec. 16

University Questions

- Q. Explain the following quality factors:
i) Maintainability ii) Reusability
(May 2012, 4 Marks)
- Q. Explain ISO 9126 Quality Factors.
(Dec. 2016, 5 Marks)

The ISO 9126 standard was developed in an attempt to identify quality attributes for computer software. The standard identifies six key quality attributes :

1. **Reliability :** The software should be available for the use and should satisfy following key attributes :

- o Maturity
 - o Fault tolerance i.e. ability to withstand against failures
 - o Recoverability i.e. the system should regain its original state once the failure occurs. It should also ensure the data integrity and consistency while recovering from failure states.
2. **Portability :** Portability means the system should be able to work in different hardware and software environments according to the following attributes :
 - o Installability
 - o Adaptability
 - o Conformance
 - o Replaceability
 3. **Efficiency :** Efficiency means making the optimal use of the system resources like memory devices, processing devices and input output devices etc. It should take into consideration following attributes :
 - o Time behaviour and
 - o Resource behaviour.
 4. **Maintainability :** Maintainability is defined as the ease with which the software may be repaired and ready to use once again. The maintainability should take into consideration following attributes :
 - o Changeability
 - o Testability
 - o Stability and
 - o Analyzability
 5. **Functionality :** The degree to which the application software satisfies the customer's needs and requirements according to the following attributes :
 - o Accuracy of working model
 - o Suitability of the product



- Interoperability (i.e. the product can be connected to any system),
- Compliance and
- Security

6. Usability : The software usability is defined as the degree to which the application software is easy to use according to the following usability attributes :

- Understandability of the software by the end-users
 - Learnability means learning the use of the product and
 - Operability.
- The ISO 9126 factors do not support direct measurement of quality, but provides indirect measurement.
 - It provides a very good checklist for evaluating the quality.

8.4.7 Software Reliability

SPPU - May 15, May 16

University Question

Q. What is the concept of Software Reliability ?
(May 2015, May 2016, 4 Marks)

- Unlike other quality factors, software reliability can be measured directly by using historical data. The software reliability is an important quality factor used by most of the developers.
- The **software reliability** is defined as the probability of failure free program in a specified environment.
- When we discuss failure, one question comes into mind. What is failure?
- Failure can be defined as non conformance to the software requirement i.e. the software application does not produce the desired output as per the requirements and these results in failures.

8.4.7.1 Measures of Reliability and Availability

SPPU - May 15

University Question

Q. Explain different measures of software reliability and availability.(May 2015, 4 Marks)

- If we talk about hardware reliability model then we predict failure due to wear rather than failure due to design defects i.e. physical wear due to effect temperature, corrosion, shock etc.
- But when we discuss software reliability then the failure is related to design defects and the failure can be traced to implementation problems.
- Consider a computer based system in that the measure of reliability is Mean-Time-Between-Failure (MTBF) where

$$\text{MTBF} = \text{MTTF} + \text{MTTR}$$

Where, MTTF = Mean-Time-To-Failure

$$\text{MTTR} = \text{Mean-Time-To-Repair}$$

- In addition the software availability is defined as the probability that a program is running as per the requirement at a given point of time.

$$\text{Availability} = [\text{MTTF} / (\text{MTTF} + \text{MTTR})] \times 100\%$$

- The availability measure is more sensitive to MTTR and it is an indirect measure of maintainability of software.

8.4.7.2 Software Safety

- Software safety is an activity of Software Quality Assurance (SQA) and it focuses on finding and assessing various hazards. These hazards can have the worst effect on the entire system and can halt the system from working.
- If these hazards are identified in the beginning of the process then it is easy to eliminate and control the hazards.



- Once these hazards are identified at system level then analysis techniques are used to find the probability of occurrence and the safety measures can be taken.
- Even if software reliability and software safety are closely related it is very important to understand the significant differences between them.
- Software reliability uses historical data to find the probability of the occurrence of software failure.
- While software safety is a way in that also failures occurs but these failures lead to accidents that are dangerous to life.

Syllabus Topic : Project Estimation : Observations on Estimation

8.5 Observations on Estimation

- The software cost estimation and the efforts estimation can never be the exact. Because there various factors affecting the estimation. **For example :** Manpower, development environment, political environment etc. These factors will affect the overall cost of the project and the efforts taken to develop the project.
- Following are some options available for achieving reliable software project estimation :
 - o Try to delay the estimation until the late in the project
 - o Use baseline metrics theory i.e. collect accurate estimates of past similar projects.
 - o Use simple decomposition techniques to obtain cost and effort estimates.
 - o Use more than one estimation models.
- Software project estimation is a form of problem solving, and in most cases, the problem to be solved is too complex. For this reason, we decompose the problem, re-

characterizing it as a set of smaller problems.

8.5.1 Software Sizing

- The accuracy of a software project estimate is predicated on a number of things :
 - o The degree estimation for the size of the software to be developed.
 - o The ability to translate the estimation for the size into the human efforts, time, and money.
 - o The abilities of the development team reflected in the project plan, and
 - o The product stability requirements and the environment that supports the software development efforts.

- Four different approaches to the sizing problem have been suggested :

- o **Fuzzy logic sizing :** In order to apply the fuzzy logic approach, the planner should identify the type of software and should establish its magnitude on a qualitative scale. Later on, this magnitude is refined within the original range.
- o **Function point sizing :** In this, the planner establishes the estimates of the information domain characteristics.
- o **Standard component sizing :** In standard component sizing approach, the software consist of different standard components. Examples of the standard components for an information system are :
 1. Subsystems
 2. Modules
 3. Screens
 4. Reports
 5. Interactive programs,
 6. Batch programs, files LOC
 7. Object-level instructions



- The project planner should estimate the number of occurrences of each of the standard components.
- **Change sizing :** In this approach, the planner should estimate different modifications that is accomplished. This particular approach is generally used when a project uses some existing project and requires its modification.
- Results of each of these sizing approaches should be combined statistically to create a three-point or expected-value estimate.

8.5.2 Problem-Based Estimation

- The LOC and FP data are generally used during software project estimation the following two ways are required :
 - (1) As an estimation variable that will enable the sizing of each element of the software and
 - (2) Secondly, as a baseline metrics used to develop cost and effort estimates.
- The project planner starts with a software scope that is a bounded statement, and tries to decompose the software into problem functions. These problem functions can be estimated individually. Later the LOC or FP is estimated for each function.
- Following are baseline productivity metrics :
 - LOC/person-month and
 - FP/person-month
- These baseline productivity metrics are applied to the appropriate estimation variable and from it the cost or effort for the function is derived. Finally the function estimates are integrated to produce one complete estimate for the whole project.
- The expected-value or three-point value can be computed for the estimation variable size S , as a weighted average of the optimistic size (S_{opt}), most likely size (S_m), and pessimistic size (S_{pess}) estimates. Consider the following example,

$$S = (S_{opt} + 4S_m + S_{pess}) / 6 \quad \dots(8.5.1)$$

The above example gives the emphasis on "most likely" estimate and it follows a beta probability distribution.

8.5.3 An Example of LOC-Based Estimation

As an example of LOC and FP problem-based estimation techniques, consider a software package to be developed for a computer-aided design application for mechanical components. The software is to execute on an engineering workstation and must interface with various peripherals including a mouse, digitizer, high-resolution colour display, and laser printer. A preliminary statement of software scope can be developed.

Table 8.5.1 : Estimation Tables for the LOC Methods

Function	Estimated LOC
User Interface and Control Facilities (UICF)	2,300
Two-Dimensional Geometric Analysis (2DGA)	5,300
Three-Dimensional Geometric Analysis (3DGA)	6,800
Database Management (DBM)	3,350
Computer Graphics Display Facilities (CGDF)	4,950
Peripheral Control Function (PCF)	2,100
Design Analysis Modules (DAM)	8,400
Estimated lines of code	33,200

For our purposes, we assume that further refinement has occurred and that the major software functions listed in Table 8.5.1 are identified. The decomposition technique for LOC, an estimation table, shown in Table 8.5.1, is developed. A range of LOC estimates is developed for each function.

For example : The range of LOC estimates for the 3D geometric analysis function is optimistic 4600 LOC, most likely 6900 LOC, and pessimistic 8600 LOC. Applying Equation (8.5.1), the expected value for the 3D geometric analysis function is 6800 LOC.

8.5.4 An Example of FP-Based Estimation

Table 8.5.2 : Estimating information domain values

Information domain value	Opt.	Likely	Pess.	Est. count	Weight	FP count
Number of external inputs	20	24	30	24	4	97
Number of external outputs	12	15	22	16	5	78
Number of external inquiries	16	22	28	22	5	88
Number of internal logical files	4	4	5	4	10	42
Number of external interface files	2	2	3	2	7	15
Count total						320

- Decomposition for FP-based estimation focuses on information domain values rather than software functions.
 - Referring to the table presented in Table 8.5.2, the project planner estimates external inputs, external outputs, external inquiries, internal logical files, and external interface files for the CAD software.
 - FP are computed using the technique discussed. For the purposes of this estimate, the complexity weighting factor is assumed to be average. Table 8.5.2 presents the results of this estimate.

Each of the complexity weighting factors is estimated and the value adjustment factor is computed.

Factor	Value
1. Backup and recovery	4

	Factor	Value
2.	Data communications	2
3.	Distributed processing	0
4.	Performance critical	4
5.	Existing operating environment	3
6.	On-line data entry	4
7.	Input transaction over multiple screens	5
8.	ILFs updated online	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
	Value adjustment factor	1.17

Finally, the estimated number of FP is derived:

$$FP_{estimated} = \text{count} - \text{total} \times [0.65 + 0.01 \sum(F_i)]$$

$$FP_{estimated} = 375$$

The organizational average productivity for systems of this type is 6.5 FP/pm.

8.5.5 Process-Based Estimation

- The most common technique for estimating a project is to base the estimate on the process that will be used. That is, the process is decomposed into a relatively small set of tasks and the effort required to accomplish each task is estimated.

Table 8.5.3 : Process-based estimation table



Activity →	CC	Planning	Risk analysis	Engineering		Construction release		CE	Totals
↓									
UICF				0.50	2.50	0.40	5.00	n/a	8.40
2 DGA				0.75	4.00	0.60	2.00	n/a	7.35
3DGA				0.50	4.00	1.00	3.00	n/a	8.50
CGDF				0.50	3.00	1.00	1.50	n/a	6.00
DBM				0.50	3.00	0.75	1.50	n/a	5.75
PCF				0.25	2.00	0.50	1.50	n/a	4.25
DAM				0.50	2.00	0.50	2.00	n/a	5.00
Totals	0.25	0.25	0.25	3.50	20.50	4.50	16.50		46.00
% effort	1%	1%	1%	8%	45%	10%	36%		

CC = Customer Communication,

CE = Customer Evaluation

- A series of framework activities must be performed for each function. Functions and related framework activities may be represented as part of a table similar to the one presented in Table 8.5.3.
- After the problem functions and development process activities are declared, the planner may begin estimating the effort like person-months. This estimation is necessary to accomplish each of the software process activity for each of software function. All these data constructs the central matrix of the table as shown in Table 8.5.3. The average labour rates like cost/unit effort are applied to the effort estimated for each of the process activity.

8.5.6 An Example of Process-Based Estimation

- To explain the use of process-based estimation, we again consider the CAD software. Refer the completed process-based table shown in Table 8.5.3, estimates of effort (in person-months) for each

software engineering activity are provided for each CAD software function. The engineering and construction release activities are subdivided into the major software engineering tasks.

- Generally the gross estimates of effort are prepared for communication with the customers, planning and risk analysis. It must be noted that nearly 53 percent of the efforts is spent on front-end engineering tasks like requirements analysis and design. It actually indicates the significance and importance of this work.

For example : Let the average burdened labour rate of \$8,000 per month, then the total estimated project cost is computed as \$368,000, and the estimated effort is approximately 46 person-months. In this case, the labour rates could be associated with each framework activity, if desired. It can also be associated with each of the software engineering task and calculated separately.

8.5.7 Estimation with Use-Cases

- Use-cases give software team members to see into insight of the software scope and requirements. But use-cases estimation approach can be problematic due to following reasons :
 - o Use-cases may be written by using various formats and styles. So far, there is no standard format.
 - o They represent the external view of the software and may be described at various levels of abstraction.
 - o They do not look at the complexity of the functions and features.
 - o And use-cases do not explain complex behaviour also.
 - o In contrast to a LOC or a FP computations, the efforts required to implement one person's "use-case" and some other person's "use-case" "may vary drastically. That is, one person may require months and another person may require a day or two.
- To illustrate how this computation might be made, consider the following relationship :

$$\text{LOC}_{\text{estimate}} = N \times \text{LOC}_{\text{avg}} + [(S_a/S_h - 1) + (P_a/P_h - 1)] \times \text{LOC}_{\text{adjust}} \quad \dots(8.5.2)$$

where, N = Actual number of use-cases

LOC_{avg} = Historical average LOC per use-case for this type of subsystem.

$\text{LOC}_{\text{adjust}}$ = Represents an adjustment based on of LOC_{avg}

S_a = Actual scenarios per use-case

S_h = Average scenarios per use-case for this type of subsystem

P_a = Actual pages per use-case

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

- The Equation (8.5.2) is useful in developing the rough estimate of the number based on

LOC and the actual number of use-cases used by several scenarios along with the page length of the use-cases. Thus the adjustment shows the historical average LOC per use case.

8.5.8 An Example of Use-Case Based Estimation

- The CAD software is composed of three subsystem groups :
 - 1. User interface subsystem
 - 2. Engineering subsystem group
 - 3. Infrastructure subsystem group
- Consider an example in that six use-cases describe the **user interface subsystem**. Each of the use case is written by approximately 10 scenarios and it has an average length of six pages.
- The **engineering subsystem group** is illustrated by approximately 10 use-cases. And each of these use-cases may have 20 scenarios associated with it and it has an average length of nearly eight pages.
- Finally, the **infrastructure subsystem group** is covered by five use-cases with an average of only six scenarios and an average length of five pages.

Table 8.5.4 : Use-case estimation

	Use-cases	Scenarios	Pages	Scenarios	Pages	LOC	LOC estimate
User interface subsystem	6	10	6	12	5	560	3,366
Engineering subsystem group	10	20	8	16	8	3100	31,233
Infrastructure subsystem group	5	6	5	10	6	1650	7,970
Total LOC estimate							42,568

- Using the relationship noted in Equation (8.5.2), the table shown in Table 8.5.4 is developed. Hence the LOC estimate for the user interface subsystem is computed using Equation (8.5.2). Using the same approach, estimates are made for both the engineering and infrastructure subsystem groups. Table 8.5.4 summarizes the estimates and indicates that the overall size of the CAD software is estimated at 42,500 LOC.

8.5.9 Reconciling Estimates

- The estimation techniques discussed in the preceding sections result in multiple estimates which must be reconciled to produce a single estimate of effort, project duration, or cost. To illustrate this reconciliation procedure, we again consider the CAD software.
- Total estimated effort for the CAD software range from a low of 46 person-months (derived using a process-based estimation approach) to a high of 68 person-months (derived with use-case estimation). The average estimate is 56 person-months. The variation from the average estimate is approximately 18 percent on the low side and 21 percent on the high side. The planner must determine the cause of divergence and then reconcile the estimates.

Syllabus Topic : Software Scope and Feasibility

8.5.10 Software Scope and Feasibility

- A software project manager is normally confused with a condition in the beginning of a software engineering project i.e. quantitative estimates and an organized plan. They are required and necessary but proper information is not available.
- A detailed analysis of software requirements may provide all the necessary

information for estimates, but analysis often takes too much time to complete, even weeks or months. Requirements may be changing regularly as the project proceeds.

- Therefore, the developer and the manager should examine the product and the problem. It is intended to solve at the beginning of the project itself. At a minimum, the scope of the product must be established and bounded.

- The first software project management endeavor is the finding out of software scope. Scope can be defined by answering the following simple questions :

- o **Context** : How the software system can be developed to fit into a large system or some business context and what are the constraints that should be considered as a result of the context considered ?
- o **Information objectives** : What data objects are produced as output through the software developed from customer's point of view ? And what data objects will be used for the input ?
- o **Function and performance** : What are the functions that the software should perform in order to translate input into output ? Is there any special performance characteristic that is supposed to be addressed ?

- There should be a clear and comprehensive software project scope defined and that should be understandable at all the levels (i.e. the management and technical levels).

- Generally the software scope describes the following :

- o Performance
- o Function
- o The data and control used to define the constraints
- o Reliability and
- o Interfaces.



- Functions are described in the statement of scope are assessed and evaluated to provide details in estimation. The cost and schedule are two important parameters and they are functionality oriented.

8.5.10.1 Obtaining Information Necessary for Scope

- In start of a development process, it is always good and necessary to define the scope of the project. The frequent communication between the customer and developer builds the foundation for defining scope properly. The frequent meetings between the customer and developer are very important to bridge the gap in all sorts of communication.
- Following are the set of questions that focuses on the goals and objectives :
 - o Know the person behind the request for this work.
 - o See who will use the solution?
 - o What will be the commercial benefit of a solution?
 - o Is there any other resource of solution?
- These questions clear the understanding of problems and customers can raise their doubts.

8.5.10.2 Feasibility

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

8.5.10.3 A Scoping Example

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

Syllabus Topic : Resources

8.6 Resources

SPPU - Dec. 16

University Question

- Q. Explain the quality attributes, considered in software design. (Dec. 2016, 5 Marks)

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

- Functionality is an important aspect of any software system. It is generally evaluated by the feature set and capabilities of that software.
- Following are general features that a system is supposed to deliver.

- **Usability** is best evaluated by considering following important factors :
 - o Human factors
 - o Overall aesthetics
 - o Consistency and documentation.
- **Reliability** is assessed by measuring following parameters :
 - o Frequency and severity of failure
 - o Accuracy of output results
 - o Mean-time-to-failure (MTTF)
 - o Recovery from failure and
 - o Predictability of the program
- **Performance** is evaluated by considering following characteristics :
 - o Speed
 - o Response time
 - o Resource consumption
 - o Throughput and
 - o Efficiency
- **Supportability** collectively combines following three important attributes :
 - o Extensibility
 - o Adaptability
 - o Serviceability
- Most commonly, above three attributes can be better defined by the term maintainability. In addition to these attributes, following are some more attributes with which a system can be installed easily, and the problems can be found out easily :
 - o Testability,
 - o Compatibility, and
 - o Configurability,
- It also contains more attributes which are as follows :
 - o Compatibility o Extensibility
 - o Fault tolerance o Modularity

- o Reusability
- o Security
- o Scalability
- o Robustness
- o Portability

Syllabus Topic : Human Resources

8.6.1 Human Resources

- The second planning task is estimation of the resources required to accomplish the software development effort.
- The Fig. 8.6.1 depicts the three major categories of software engineering resources : people, reusable software components, and the development environment.
- Each resource is specified with 4 characteristics :
 - o description of the resource;
 - o a statement of availability;
 - o time when the resource will be required;
 - o duration of time that resource will be applied.

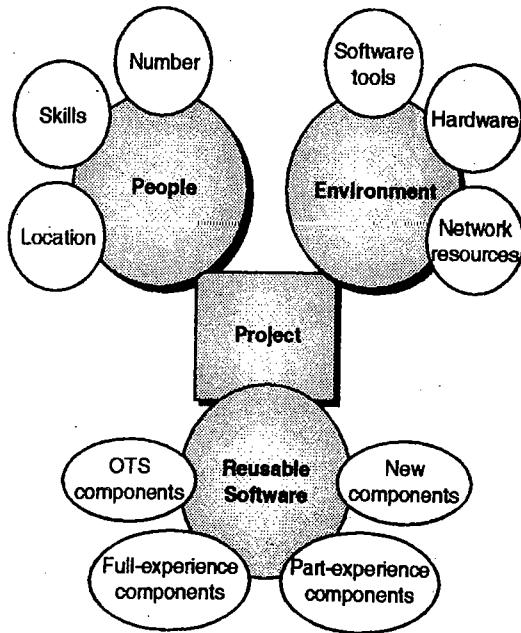


Fig. 8.6.1 : Project Resources

Syllabus Topic : Reusable Software

8.6.2 Reusable Software Resources

- Reusability is an important software building block that must be catalogued for easy reference, standardized for easy application, and validated for easy integration. Four software resources have been suggested as categories that should be considered as planning proceeds :
 - o **Off-the-shelf components** : The reusability is an important feature and thus the existing software can be purchased from a third party to use it in the current project. Usually the COTS i.e. "commercial off the shelf" are purchased. They can be completely validated after use in the current project.
 - o **Full-experience components** : The existing components had undergone through all phases of software development life cycle i.e. specifications, designs, code, and testing. Thus these full-experienced components can be used in the current project.
 - o **Partial-experience components** : Some components can be used directly, but they need some modification before they can be used. They are referred as partial-experience components. They had also gone through all phases of software development life cycle i.e. specifications, designs, code, and testing. Still they require modification.
 - o **New components** : In addition to all these components, some new components may also be built as per the demands of the current project.
- Thus the evaluation of all these alternatives is conducted.

Syllabus Topic : Environmental Resources

8.6.3 Environmental Resources

- The software engineering environment (i.e. SEE) supports the software project. The SEE normally supports all the engineering activities of the development process.
- Similarly the hardware also provides a platform that supports all the software tools needed to complete an application by using some good engineering practices.
- While designing and developing a computer based system, the development team uses all the hardware elements designed by other development team. All these resources are referred as environmental resources.

Syllabus Topic : Software Project Estimation

8.7 Software Project Estimation

- Software cost and effort estimation will never be an exact science. Too many variables are there like human, technical, environmental, political that can affect the ultimate cost of software and effort applied to develop it.
- However, software project estimation can be transformed from a black art to a series of systematic steps that provide estimates with acceptable risk.
- To achieve reliable cost and effort estimates, a number of options arise :
 - o Delay estimation until late in the project (obviously, we can achieve 100% accurate estimates after the project is complete!).

- Base estimates on similar projects that have already been completed.
- Use relatively simple decomposition techniques to generate project cost and effort estimates.
- Use one or more empirical models for software cost and effort estimation.

Syllabus Topic : Decomposition Techniques

8.8 Decomposition Techniques

- Software project estimation is a form of problem solving, and in most cases, the problem to be solved (i.e., developing a cost and effort estimate for a software project) is too complex to be considered in one piece.
- For this reason, we decompose the problem, recharacterizing it as a set of smaller (and hopefully, more manageable) problems.
- We use following two types of Decomposition Techniques :
 - Problem decomposition, and
 - Process decomposition

8.8.1 Problem Decomposition

- Problem decomposition is an activity that is present during software requirements analysis. The problem decomposition, sometimes called as **partitioning** or **problem elaboration**.
- During defining the scope of software, the problem is not fully decomposed. The decomposition activity is applied in following two important areas :
 - The functionality that is expected to be delivered and
 - The process that helps to deliver it.
- Generally all the human beings prefer a divide-and-conquer scheme when they face any complex problem. It can be stated in this way that a complex problem is divided

into smaller sub problems that can be managed very easily.

- This technique is often applied in the beginning of the project planning.
- During the evolution of the statement of scope, the first level of partitioning occurs naturally. For example, the project team members learn that the marketing team has communicate with potential customers and found that the following functions should be the part of automatic copy editing required :
 - Spell checking
 - Sentence grammar checking
 - Reference checking for large documents
 - Section and chapter reference validation for large documents.

- All these features represent a sub-function that is supposed to be implemented in software. Each feature can be further refined if the decomposition will make planning easier.

8.8.2 Process Decomposition

- An outline for project planning is ascertained by the process framework. It is adapted by allocating a task set that is appropriate to the project. The framework activities that characterize the software process are applicable to all software projects. The problem is to select the process model that is appropriate for the software to be engineered by a project team.
- The project manager should decide that which of the process model is most appropriate for :
 - The customers who request the product and the people who will do the work
 - The characteristics of the software product
 - The project environment in which the software team works

- When a process model is selected, the team then defines a preliminary project plan based on the set of process framework activities. After establishing the preliminary plan, the process decomposition can be started. A complete plan must be created, that reflects the work tasks required to populate the framework activities.
 - It is always recommended that a software team should be adaptable in selecting the software process model that is most appropriate for the project and all the software engineering tasks can be implemented once the process model is chosen.
 - Usually small projects can be completed using the linear sequential approach.
 - If time constraints are imposed very strictly then the problem might be heavily modularized into several small models and generally RAD model would be the correct choice.
 - If the deadline is to be followed very strictly then it is fact that full functionality may not be delivered in the first version and in this scenario, an incremental approach would be most suitable approach.
 - If projects encounter with following characteristics, then some other process models might be selected :
 - o Uncertain requirements
 - o Breakthrough technology
 - o Difficult customers
 - o Significant reuse potential
 - Once the process model has been finalized, the process framework is adapted to it. In all the cases, the generic framework activities like planning, communication, modelling, construction, and deployment can be used.
 - It will work for the following models :
 - o Linear models
 - o Iterative and incremental models
 - o Evolutionary models and
 - o Even for concurrent or component assembly models.
 - The process framework is invariant and serves as the basis for all software work performed by a software organization.
 - But the actual work tasks do always vary. The process decomposition begins when the project manager asks, "how do we accomplish this framework activity?"
 - **For example :** A small, relatively simple project may require the following work tasks for the communication activity :
 - o Develop list of clarification issues.
 - o Meet with customer to address clarification issues.
 - o Jointly develop a statement of scope.
 - o Review the statement of scope with all concerned.
 - o Modify the statement of scope as required.
 - These events may occur in less than 48 hours. These events represent that a process decomposition is appropriate for the small and relatively simple project. It should be noted that work tasks must be adapted to the specific needs of the project.
-
- Syllabus Topic : Empirical Estimation Models**
-
- ### 8.9 Empirical Estimation Models
-
- An estimation model for software products uses empirically derived formulas in order to predict the effort in terms of functions of LOC or FP. The values for LOC or FP can be estimated. Here, the resultant values for

- LOC or FP are used in the estimation model without the values using the tables.
- The empirical data are always supported in most of the estimation models. These empirical data are derived from some sample of projects. Due to this reason only, there is no estimation model suitable for all the classes of software and in all development environments. And hence the results obtained in such cases are used judiciously.
 - An estimation model must be calibrated in such a way that reflects its local conditions. The estimation model must be thoroughly tested by applying the data gathered from the past completed projects.

Syllabus Topic : Empirical Estimation Model : Structure

8.9.1 The Structure of Estimation Models

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

$$E = A + B \times (e_v)^c \quad \dots(8.9.1)$$

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

- In addition to the relationship described by Equation (8.6.1), most of estimation models use project adjustment component that enables the effort E. Generally the effort E is adjusted by the following project characteristics :
 - o Problem complexity,
 - o Staff experience,
 - o Development environment etc.

- Various LOC-oriented estimation models that have been proposed in the literature are as follows :

- 1) $E = 5.288 \times (\text{KLOC})^{1.047}$
(Doty model for KLOC > 9)
- 2) $E = 3.2 \times (\text{KLOC})^{1.05}$
(Boehm simple model)
- 3) $E = 5.2 \times (\text{KLOC})^{0.91}$
(Walston-Felix model)
- 4) $E = 5.5 + 0.73 \times (\text{KLOC})^{1.16}$
(Bailey-Basili model)

- Following are some FP-oriented models proposed :

- 1) $E = -37 + 0.96 \text{ FP}$ (Kemerer model)
- 2) $E = -91.4 + 0.355 \text{ FP}$
(Albrecht and Gaffney model)
- 3) $E = -12.88 + 0.405 \text{ FP}$
(small project regression model)

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

Syllabus Topic : Empirical Estimation Model : COCOMO II

8.9.2 The COCOMO II Model

SPPU - Dec. 13, Dec. 15, May 16, Dec. 16

University Questions

- Q. Explain COCOMO model for project cost estimation.
(Dec. 2013, Dec. 2015, May 2016, 8 Marks).
- Q. Explain COCOMO-II. (Dec. 2016, 5 Marks)

- Barry Boehm has devised a software estimation models hierarchy having the name as COCOMO i.e. COnstructive COst MOdel. The COCOMO model has been evolved into more comprehensive model called COCOMO II. It is actually having a hierarchy of estimation models that focus the following areas :

- **Application composition model :** It is used in the early stages of development, when user interfaces, system interaction, performance assessment and technology evaluation are prime important.
 - **Early design stage model :** Once the requirements are stabilized and basic architecture is constructed, then early design stage model is used.
 - **Post-architecture stage model :** It is used during development of the software.
- COCOMOII models also require sizing information like other estimation models for the software. Following three sizing options are available :
- Object points
 - Function points and
 - Lines of source code
- The COCOMO II model uses object points that are an indirect software measure. They are computed using the counts of :
- Screenshots taken at user interface
 - Reports and
 - Components required in developing the application.
- The screenshots or the reports are classified into any of the following three complexity levels :
- Simple
 - Medium
 - Difficult
- The client and server data tables are required to create the screenshots and reports. The complexity is defined as the function of these reports.

Table 8.9.1 : Complexity weighting for object types

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

After the determination of complexity, the number of screenshots, reports, and components object point are weighted as per the Table 8.9.1. In component-based development when software reuse is implemented, then the percent of reuse (% reuse) is estimated and the object point count is adjusted according to the following equation :

$$\text{NOP} = (\text{object points}) \times [(100 - \% \text{ reuse})/100]$$

where NOP → New Object Points.

Table 8.9.2 : Productivity rate for object points

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

- In order to derive the estimate of effort, a “productivity rate” should be derived first. They are based on the computed NOP value. The Table 8.6.2 shown above presents the productivity rate.
- These productivity rates are illustrated for various levels of developer’s experience as well as various levels of environment maturity.

$$\text{PROD} = \text{NOP}/\text{person-month}$$

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



Estimated effort = NOP/PROD

8.9.3 The Software Equation

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

$$E = [LOC \times B^{0.333}/P]^3 \times (1/t^4) \quad \dots(8.9.2)$$

Where,

E = Person-months effort or person-years efforts

t = Duration of the project in months or years

B = The special skills factor

P = Productivity parameter

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

- o An indication of project duration.

Syllabus Topic : Estimation of Object-Oriented Projects

8.10 Estimation of Object-Oriented Projects

- Estimation for the projects that has been designed especially for object oriented environment is appended with some extra guidelines.
- So object oriented estimation uses the same approach as it was used in conventional software cost estimation, but in addition following approach is suggested :
 - o Develop estimates using effort decomposition, FP analysis, and any other method that is applicable for conventional applications.
 - o Using object oriented analysis modeling, develop use case and determine a count
 - o From the analysis model, determine the number of classes key
 - o Categorize the type of interface for the application and develop a multiplier for support classes as shown below :

Interface Type	Multiplier
No GUI	3.0
Text based Interfaces	2.45
GUI	3.5
Complex GUI	4.0

Now multiply number of key classes obtained in step 3 by the multiplier depicted above to estimate support classes.

1. Multiply total number of classes i.e. key + support of work-units per class usually 15 to 20 person-days per class.

2. Finally cross check the class-based estimate by multiplying the average number of work-units per use case.

Syllabus Topic : Specialized Estimation

8.11 Specialized Estimation

- Various estimation techniques discussed so far can be used for all types of projects. But these estimation techniques are not useful for the projects having shortest development duration.
- We discuss two specialized estimation techniques as follows :
 - o Estimation for Agile Development
 - o Estimation for Web Engineering Projects

8.11.1 Estimation for Agile Development

- In terms of the agile development, the estimation process is an iterative one whereby the user stories in XP represent pieces of functionality to be estimated and this is done every 2 weeks.
- An overall expected time for each of these stories is estimated by the developers, and the customers then prioritize the stories based on these initial estimates and on the business value of each one.
- The nature of agile methods often results in fixed budgets and a fixed schedule, and it is the scope of the project that remains flexible throughout.
- On the other hand report that companies using agile methods usually lean towards “flexible contracts instead of fixed ones that predefine functionalities, price, and time”.
- The techniques used to estimate agile development projects have been typically expertise-based, where the developers look to past projects or iterations, and draw on their own experiences to produce estimates for the stories.

8.11.2 Estimation for Web Engineering Projects

- By using measurement principles to evaluate the quality and development of existing Web applications, we can obtain feedback that will help us to understand, control, improve, and make predictions about these products and their development processes.
- Prediction is a necessary part of an effective software process, whether it be authoring, design, testing, or Web development as a whole. As with any software project, having realistic estimates of the required effort early in a Web application’s life cycle. Lets project managers and development organizations manage resources effectively.
- The prediction process involves capturing data about past projects or past development phases within the same project, identifying size metrics and cost drivers and formulating theories about their relationship with effort, generating prediction models to apply to the project, and assessing the effectiveness of the prediction models.
- Design covers the methods used for generating the structure and functionality of the application, and typically doesn’t include aspects such as hypermedia application requirements analysis, feasibility consideration, and applications maintenance.
- In addition, our design phase also incorporates the application’s conceptual design, reflected in map diagrams showing documents and links. The data we used to generate our prediction models came from a quantitative case study evaluation, in which we measured a set of suggested size metrics and cost drivers for effort prediction.



- Some of our complexity metrics are adaptations from the literature of software engineering and multimedia. The metrics we propose characterize Web application size from two different perspectives : length and complexity.
- Our prediction models derive from statistical techniques, specifically linear regression and stepwise multiple regression. We also use Polynomial regression to study the effect of the predictors individually.

Syllabus Topic : Case Study : Software Tools for Estimation

8.12 Case Study : Software Tools for Estimation

- Estimating size or resources is one of the most important topics in software engineering and IT. You will not deliver according to expectations if you don't plan, and you cannot plan if you don't know the underlying dependencies and estimates.
- An estimate is a quantitative assessment of a future endeavor's likely cost or outcome. People typically use it to forecast a project's cost, size, resources, effort, or duration.
- Today's software market, with its increasing reliance on external components and adapted code, has led to new kinds of technologies for estimation, a practice that has moved from mere size-based approximations to functional and component estimates.
- Standards are starting to evolve as well, because these estimates play such a crucial role in business and enormous amounts of money are at stake.
- Unfortunately, people often confuse estimates with goals or plans. For instance, they schedule projects according to needs instead of feasibility, or commit to

something "very urgent and important" before checking how this "urgency" relates to current commitments and capacity planning.

- In fact, most failures in software projects come from belatedly understanding and considering the important difference between goals, estimates, and plans.
- Two of the most important ingredients for proper estimation are people and historical data. They are interrelated more than you might expect because most organizations lack historical data, thus they form estimates primarily through analogy and experience.
- It works if you have experienced people who periodically measure and put their estimates versus actual data into a historical database. Tools can help reducing the time and costs involved in data gathering, as well as supporting reports, risk management, and scenario analysis.
- Following are some Software Estimation Tools :
 - o **SLIM-Estimate** : It uses a proven top-down approach that minimizes the input information required to produce fact based, defensible estimates. In addition to software cost estimation, SLIM-Estimate's high level of configurability accommodates the many different design processes used by developers.
 - o **SystemStar** : It offers two types of models :
 - o COCOMO -- Software Cost Estimation
 - o COSYSMO -- Systems Engineering Cost Estimation

Project Scheduling

Syllabus

Project Scheduling : Basic Concepts, Defining a Task Set for the Software Project, Defining Task Network, Scheduling with time-line charts, Schedule tracking Tools:- Microsoft Project, Daily Activity Reporting & Tracking (DART)

Syllabus Topic : Project Scheduling - Basic Concepts

9.1 Project Scheduling

SPPU - May 12, Dec. 16

University Question

Q. What is project scheduling ? What are the basic principles of project scheduling ?

(May 2012, Dec. 2016, 8 Marks)

- Project scheduling involves separating the total work involved in a project into separate activities and judging the time required by these activities.
- Scheduling in software engineering can be considered from two views. First, end date for release has been established for a project and organization is responsible for distributing effort within prescribed time frame. Secondly, rough chronological wounds are discussed and end date is set by organisation.
- Thus, project scheduling is an activity that distributes estimated effort across the planned project duration by specific allocation of effort to the specific task in software engineering.

Basic principles for software project scheduling are :

1. Compartmentalization

Project is divided or compartmented into number of manageable activities, actions and tasks.

2. Interdependency

Interdependency between each compartmentalized activity, action task must be determined.

3. Time allocation

Each task is assigned some work unit i.e. effort by person in days.

4. Effort Validation

Every project has defined number of people and as time allocation occurs, project manager should check no more than allocated numbers of people have been scheduled at any given time.

5. Defined Responsibilities

Every task in schedule should be assigned to a specific team member.

6. Defined Outcomes

Every schedule task should have defined outcome i.e. work product or a part of work product.



7. Defined Milestones

Every task or group of tasks is associated with project milestone.

Syllabus Topic : Defining a Task Set for the Software Project

9.1.1 Defining a Task Set for the Software Project

- Regardless of model used, process model is populated by a set of tasks that enable a software team to define, develop and ultimately support computer software. But the same set of tasks is not appropriate for all types of projects.
- For larger projects, different set of tasks will be applicable. In the same way, for complex projects also, some different set of tasks will be used.
- For developing a project schedule, the entire task set should be divided on the project time line.
- For each project, the set of tasks will vary depending on the type of the project.
- Following are some types of projects :
 - o Concept development projects
 - o New project having certain application
 - o Application enhancement projects
 - o Maintenance projects
 - o Reengineering projects
- In a particular type of project, many factors influence the task set to be chosen.
- These factors include :
 - o Size of the project
 - o Number of users
 - o Stability requirements
 - o User friendliness
 - o Ease of communication between the application developer or user
 - o Performance
 - o Technology used

An example set of tasks

- Consider the example of concept development projects. Following are the sets of tasks those can be applied :
 - o Concept scope
 - o Concept planning
 - o Risk assessment and management
 - o Proof of concept
 - o Implementation
 - o Customer feedback and reaction
- These are certain activities or the set of tasks those can be applied for the concept development projects.

9.1.2 Scheduling

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

Syllabus Topic : Scheduling with Time-line Charts

Time-line charts

SPPU - Dec. 12

University Question

Q. What is time line chart ? How it is used in scheduling of software project ?

(Dec. 2012, 8 Marks)

- A time-line chart, also called Gantt chart, is generated on the basis of the start date inputs for each task.

- Fig. 9.1.1 shows an example Gantt chart. It elaborates the project schedule.
- In the left hand column of the Fig. 9.1.1, all the project tasks are listed. The horizontal lines indicate the duration of the task.
- For concurrent activities, multiple horizontal lines are used.

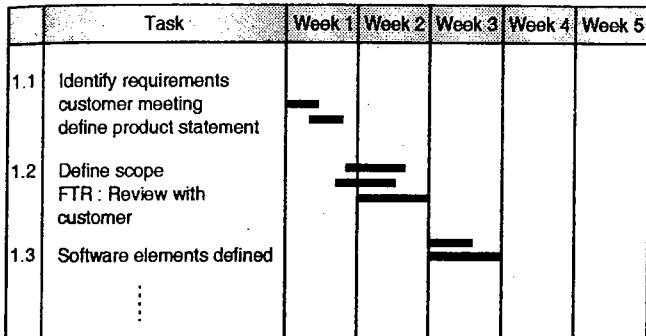


Fig. 9.1.1 : An example time-line chart

9.1.3 Tracking the Schedule

- The project schedule is nothing but a road map that defines the tasks and the milestones to be tracked.
- Tracking is achieved in different ways as follows :
 - o Conduct periodic meeting to find out the progress status and problems.
 - o Evaluate the process.
 - o Check whether all the activities are completed within the deadline or by the schedule date.
 - o Compare the planned date and actual start date for each activity.
 - o Meet practitioners informally to assess the progress.
- All these tracking techniques are used by the project managers.

9.2 Earned Value Analysis SPPU - May 16

University Question

Q. Explain the earned value analysis in project scheduling. (May 2016, 8 Marks)

- During the project tracking, each developer gives his project or task progress status to

manager. But this assessment of information is subjective.

- There should be some quantitative technique for assessing progress of project.
- Earned Value Analysis (EVA) is a technique used for quantitative analysis in the following manner :

The EVA provides a common value scale for every task. The time to complete total project is estimated and every task is given an earned value based on its estimated percentage of the total.

- To determine the earned value, following steps are performed :

1. The Budgeted Cost of Work Scheduled (BCWS) is calculated for each task.
2. The BCWS values for all work tasks are summed to calculate the final budget.

$$\therefore \text{BAC} = \sum (\text{BCWS}_k) \text{ for all tasks } k.$$

Where

BAC = Budget at Completion.

3. Now, the value for Budgeted Cost of Work Performed (BCWP) is computed. The value of BCWP is sum of the BCWS values for all work tasks completed by that point in time.

The progress indicators can be calculated as :

- o Scheduled Performance Index,

$$\text{SPI} = \frac{\text{BCWP}}{\text{BCWS}}$$

- o Scheduled Variance,

$$\text{SV} = \text{BCWP} - \text{BCWS}$$

- An SPI value close to 1.0 indicates efficient execution of the project schedule.

$$\text{Percent scheduled for completion} = \frac{\text{BCWS}}{\text{BAC}}$$

- It provides the percentage of work that should have been completed by time t.

$$\text{Percent complete} = \frac{\text{BCWP}}{\text{BAC}}$$

- It provides the percent of completeness of the project at a given point in time t.
- Also, the Actual Cost of Work Performed (ACWP) is the sum of the efforts actually expended on work tasks that have been completed by a point in time on the project schedule.
∴ Cost performance index,

$$CPI = \frac{BCWP}{ACWP}$$

Cost variance, $CV = BCWP - ACWP$

- A CPI value close to 1.0 provides a strong indication that the project is within its defined budget.

Syllabus Topic : Schedule Tracking Tools

9.3 Schedule Tracking Tools

- The project planning is an important activity performed by the project managers in order to estimate the following entities for their optimum use :
 - The resources required for the project under development.
 - The cost of the project, and
 - The schedule for in-time completion of the project
- Project Managers can use a range of tools and techniques to develop, monitor and control project schedules.
- These tracking tools can automatically produce critical path diagrams or Gantt charts. These tools also instantly update time plans as soon as new information is entered and produce automatic reports to monitor and control the project.

Project team tab ribbon

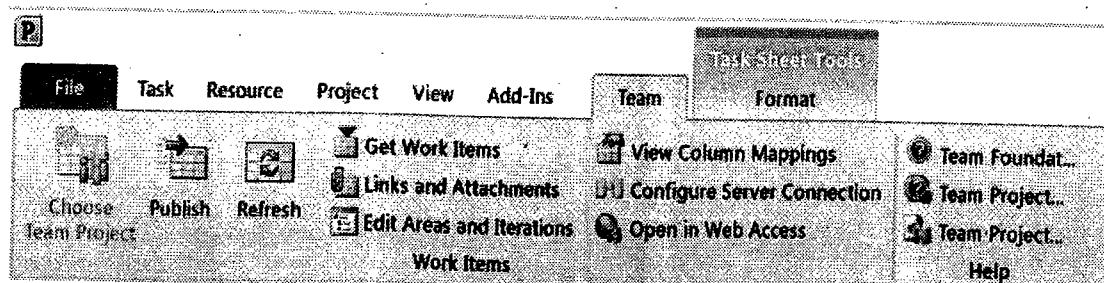


Fig. 9.3.1 : Project team tab ribbon

- These tracking tools can assist in planning and controlling the expenditures more effectively and improve communication of financial reporting.
- These tools also provide automatic report template with good GUI to effectively communicate with all the stakeholders of the project.
- These tools also look into work breakdown structure and risk management with greater accuracy and ease of updating and printing the reports.
- Following are some examples of Schedule Tracking Tools :
 - Microsoft Project
 - Daily Activity Reporting and Tracking (DART)

Syllabus Topic : Schedule Tracking Tools - Microsoft Project

9.3.1 Microsoft Project

- We can use Microsoft Project 2010 or later versions to plan team projects, schedule tasks, assign resources, and track changes to data that is stored in Team Services or TFS.
- By using Project, you can access many tools and functions through the simplified graphical menus and Office Ribbon. The Team tab menu, as shown in the Fig. 9.3.1, displays the same functions that are available from the Team tab in Excel.

- Project 2010 and later versions support several new project fields and functions. Depending on how you use Project to schedule team tasks, you may want to update the task work item form to display some of the new fields.
- To maintain new Project fields in both your project plan and in TFS, you have to customize the task work item type and the Microsoft Project Field mapping file for the team project.
- In Project 2010 and Project 2013, New task-related features are facilitated to schedule the tasks manually or automatically.
- By using Task Mode, which is accessed through the following Ribbon menu, you have more flexibility in the way you and team members schedule tasks.

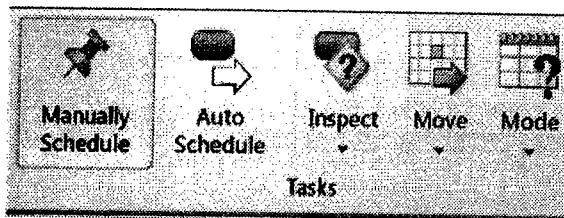


Fig. 9.3.2 : Task Mode

- Start and finish dates for auto scheduled tasks () are determined by the scheduling engine based on task dependencies and the project calendar, as in previous releases of Project.
- Project managers who are accustomed to automatic scheduling with past versions of Project can turn the new manual scheduling feature off for specific tasks or the entire project.
- Finally, it is quite simple to use Microsoft Projects for scheduling the projects. It is quite similar to other products of Microsoft office like Microsoft word, Microsoft excel and others.

Syllabus Topic : Schedule Tracking Tools : Daily Activity Reporting & Tracking(DART)

9.3.2 Daily Activity Reporting and Tracking (DART)

- DART – Daily Activity Reporting Tool, enables you to track the changes to record being made by users. The extension provides the ability to dispatch the email with summary of changes.
- After installation, in the tool we see various tabs. Here we provide brief introduction of these tabs and their usage.

Scheduling

- DART provides cron/modules/DART/DARTCron.service (php file) that can be configured through scheduler for execution at the end-of business day.
- The script will trigger the action to gather the changes and summarize it via email. The email configuration details can be updated in the file modules / DART / DART.Config.php.

Example

- Here is a CRON tab entry to schedule the script to be executed every day at 23:00 (i.e. 11:00 PM)

```
0 23 *** php -f /home/site/vtigercrm/cron  
/modules/DART/DARTCron.service
```

Module Views

- When you open DART it shows the list of changes tracked for the day. You can click on Refresh now link if you don't see it updated. The changes are updated when the CRON script is scheduled too.



NOTE: To receive email of the report you need to schedule the script cron/modules/DART/DARTCron.service, preferably at end-of-every day.

User	Action performed
admin	Leads : UPDATED: WilsonSusan Trouble Tickets : UPDATED: How to automatically add a lead from a web form to VTiger

vtiger CRM 6.1.0 © 2004-2010 vtiger.com | Read License | Privacy Policy

Fig. 9.3.3 : DART Snapshot

No activity notification

- DARTCron.service can send notification emails to group of users who have not used the system for the given day.
- To enable this, you need to uncomment the line in cron/modules/DART/DARTCron.service

```
//$dartCron->sendNoActivityUpdateEmail();
as
$dartCron->sendNoActivityUpdateEmail();
```

Project Risk Management

Syllabus

Project Risk Management : Risk Analysis & Management: Reactive versus Proactive Risk Strategies, Software Risks, Risk Identification, Risk Projection, Risk Refinement, Risk Mitigation, Risks Monitoring and Management, The RMMM plan for case study project

Syllabus Topic : Risk Analysis and Management

10.1 Risk Analysis and Management

SPPU – May 14, May 15, Dec. 15, May 16

University Questions

- Q. What are the types of risks ? Explain in brief.
(May 2014, 6 Marks)
- Q. Explain principles of risk management in detail.
(May 2015, 8 Marks)
- Q. What are the different categories of risk. Explain risk management process in detail.
(Dec. 2015, May 2016, 8 Marks)

- Whenever we start any business or any development process, we take into consideration the risks involved in accomplishing that task. Similarly when software development process is started, it is main job of a software manager to look into all types of possible risks involved in the entire process.
- It involves focusing on the possible risks that could affect the project development process :
 - o Schedule of the development process

- o Quality of the application under construction

- It is the responsibility of a project manager to look into the matter and take necessary action to avoid these risks. All the results of risk analysis and analysis of consequences of risk occurring should be well documented in the planning of the project itself.
- If the risk management is effective, then it becomes easier to handle all the problems and it is ensured that the project schedules and budget are within acceptable limits and there is no schedule slippage and ultimately no budget slippage.

10.1.1 Software Risks

- Software risk is a type of risk that always threatens the project development processes and the software under construction.
- Following are three important categories of risk :

1. Project risks
2. Product risks
3. Business risks

1. Project risks

- These are the risks that directly affect the schedule of the project and the resources involved in the development process.
- **Example of project loss :** Loss of an experienced developer and designer.

2. Product risks

- The product risks affect the quality and performance of the application built.
- **Example of product risks :** Failure of a purchased component to perform as per expectation.

3. Business risks

- The business risks are affecting the organization those develop and process the software.
- **Example of business risks :** A competitor of the organization introducing a new product.

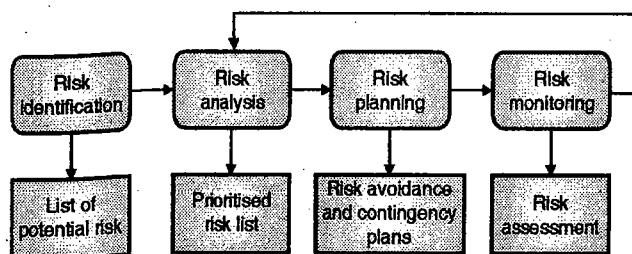


Fig. 10.1.1 : The Risks management process.

10.1.2 Reactive Versus Proactive Risk Strategies

1. Reactive risk strategy

- In this type of risk strategy, the project is monitored closely for the likely risks that may occur.
- Resources are monitored to guess the possible risks and deal with them so that they do not become the problems for budget slippage and cost slippage.

- Generally in reactive risk strategy, the development team members are directly not involved in the risks occurring.
- And after occurrence of the risks, the development team members do some rapid action to overcome the problems.
- This rapid action is called as “fire fighting mode”.
- After failure of all these actions, the concept of “crisis management” comes into picture and the project is in deep trouble and uncertainty happens.

2. Proactive risk strategy

- A proactive risk strategy is initiated when the actual technical work begins.
- The potential risks are identified and the probability and impact are assessed so that the development team members establish an appropriate plan to manage all these risk.
- The main goal to avoid the risks initially.
- But all the risks cannot be avoided, therefore the development team creates a contingency plan that will control the risks in an effective manner.

Syllabus Topic : Risk Identification

10.2 Risk Identification

SPPU – Dec. 12, May 13,
Dec. 13, Dec. 16

University Questions

- Q. Write short note for following : Risk identification. (Dec. 2012, 4 Marks)
- Q. Explain the risk identification. (May 2013, 8 Marks)
- Q. Explain the risk identification and assessment process for a software project ? (Dec. 2013, 8 Marks)

Q. What is risk identification? What are the different categories of risks?

(Dec. 2016, 7 Marks)

- Risk identification is related to discovering possible risks to the project. It is a systematic attempt to specify threats to project plan.
- There are two types of risks :

1. Generic risk
2. Product specific risk

1. Generic risk

They are potential threat to every software project.

2. Product specific risk

- o The project plan and software statement of scope are examined to find out threats due to special characteristics.
- o The check list can be created of risk and then focus is subset of known and predictable risks in following subcategories :

- (i) Product size
- (ii) Business impact
- (iii) Customer character
- (iv) Process definition
- (v) Development environment
- (vi) Technology to the limit
- (vii) Staff size and experience

(i) Product size

Risk involved in the overall size of the software.

(ii) Business impact

Risk involved in constraints imposed by management or the market.

(iii) Customer characteristics

Risk involved in the sophistication of the customer and ability of the developer to communicate the customer properly.

(iv) Process definition

Risk involved in defining the software process followed by software development organization.

(v) Development environment

Risk involved in availability and quality of the tools used in the development process.

(vi) Technology to the limit

Risk involved in complexity of system and risks associated with the new technology used.

(vii) Staff size and experience

Risk involved in overall development teams i.e. experience of developer, skill set of team members.

10.2.1 Assessing Overall Project Risk

- Following are the set of questions obtained from some previous successfully completed projects :
 - o Are the top software managers and customer managers are committed to help the project ?
 - o Are the end-users are involved in the development process with enthusiasm ?
 - o Are all the requirements elicited properly by the customers and are these questions well understood by the development team ?
 - o Have the customers involved completely in the development process with development team while defining the requirements ?
 - o Are all the end users mentioning their realistic expectations from the project ?
 - o Is the project scope stable or not ?

- Are there in the development team proper blend of skilled team members ?
- Are the requirements of the project stable ?
- Are the project team members well familiar with the latest technologies ?
- Are there enough team members in a team allotted for a project ?
- Are all the customers and users agree on the requirements of the project to be built ?
- If these questions are answered negatively, then the project may be at risk. In another words, we can say that the degree of risk is directly proportional the number of negative answers given.

10.2.2 Risk Components and Drivers

- In the project development process, it is essential for a project manager to identify the factors that affect the following **risk components** :

 - Performance
 - Cost
 - Support and
 - Schedule

- The factors that affect the risk components are also called as **risk drivers**. In the

context of risk management, the above components can be explained as follows :

- **Performance risk** : It is defined as the degree of uncertainty that an application satisfy the requirement expected by the customer.
- **Cost risk** : It is defined as the degree of uncertainty that a project does not exceed its budget.
- **Support risk** : It is defined as the degree of uncertainty that the final product will be adaptable and easy to maintain.
- **Schedule risk** : It is defined as the degree of uncertainty that the final product will be delivered on its deadline as mentioned in the documents.

The impact of the risk drivers on risk components are classified in four different classes as follows :

- Negligible impact
- Marginal impact
- Critical impact, and
- Catastrophic impact

We can illustrate all these risk components (i.e. performance, cost, support and schedule) in the following chart :

Table 10.2.1 : The table showing impact analysis

Risk component Class of impact	Performance	Cost	Support	Schedule
Negligible impact	If performance failure occurs, then it will cause inconvenience.	Errors can cause no serious impact on the project budget	The technical performance is not compromised. It can be supported with ease.	The delivery is before the deadline.



Risk component Class of impact	Performance	Cost	Support	Schedule
Marginal impact	There is some degradation in technical performance.	Some schedule slips can cause the cost within limits.	Some small degradation in performance.	The delivery is within the limits.
Critical impact	There is question mark in completion and success of the project.	The failure can cause project delays and thus financial overruns.	Some delay in project delivery due to modifications in the LOC.	There is schedule slippage due to LOC delays.
Catastrophic impact (Disastrous impact)	If performance failure occurs, then simply it is the project failure.	Failure can cause drastic cost overrun and project delays.	The final product is non-responsive and can not be supported.	Unachievable deadline due to increased cost and financial crunches.

Syllabus Topic : Risk Projection

10.3 Risk Projection

- Risk projection is interchangeably called as Risk estimation also. The risk projection rates the risk in following two ways :
 - o The probability of risk occurrence and
 - o The consequences of the risk occurred.
- Following are four important risk projection activities that every manager, developer should perform :
 - o Make a scale to measure the likelihood of the risk.
 - o Describe the consequences of the risks.

- o Estimate its impact and
- o Write down overall accuracy of the risk projection to avoid misunderstandings.

10.3.1 Developing a Risk Table

SPPU - Dec. 13

University Question

- Q. Write short notes for the following : Risk table
(Dec. 2013, 4 Marks)

- A risk table gives very useful technique to project managers for the risk projection. Following Table 10.3.1 is an example of risk table :

Table 10.3.1 A sample risk table

Risks	Category	Probability	Impact	RMMR
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End-users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirement	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	

- In the Table 10.3.1 the impact values are as follows :
 - o For the disastrous situations → 1
 - o For some critical situations → 2
 - o For the average situations → 3
 - o For the negligible situations → 4
- In the above table the categories used are as follows :
 - o PS used for project size risk
 - o BU used for business risk
 - o CU used loss of funds etc.
- The risk table is prepared right in the beginning of the development process. It is very much useful in analyzing the various types of risks associated. The risks are categorized in different categories as shown in Table 10.3.1.

- The probability of occurrence of particular risks is also estimated and associated impact values are listed in the above table. The project manager goes through this table and gives a cutoff line. All the risks that fall below the cutoff line are evaluated again. Similarly the risks that are above the cutoff line are properly managed by the project manager.

10.3.2 Assessing Risk

Following are three important factors that affect the result if risk occurs :

- **The nature of risk :** It is important to know the nature of the risk expected to occur. For example, if the hardware interface is not defined properly, then it will cause the problems during integration.
- **The scope of risk :** It looks into the severity of the risks and how the end-users will be affected by the occurrence of the risk.
- **The timing of risk :** It is important to note that for how long the impacts of risk will remain.
- Consider the following risk analysis approach. In order to determine the impact of risk, following steps are recommended :
 - o Find the average probability of occurrence of the risk for the risk components like performance, cost, support and schedule risk.
 - o Refer Table 10.2.1, and calculate the impact of each component.
 - o Complete the risk table and perform analysis on the result.
- Let RE is the overall risk exposure and it written as :

$$RE = P \times C$$

Where P→probability of occurrence of a risk
C→the cost incurred to the project, if risk occurs.

Consider the following example to understand above equation :

Example 1 : Let a software team describes the probable risk in the following manner :

(a) **Risk identification :** There are 75 % reusable components that can be integrated to the complete application. The remaining 25 % are the functionalities that are customized.

(b) **Risk probability :** Let there are 85 % probable risks.

(c) **Risk impact :** Let there are only 65 % reusable components planned out of 75 % used. Then in this scenario, nearly 15 components have to be developed from scratch, i.e. they are not reusable components. For calculation purpose, we consider 100 LOC on an average. Assume that \$20 are incurred for each of the LOC, the total cost to develop the complete project can be computed as :

$$15 \times 100 \times 20 = \$30,000.$$

Thus the **risk exposure** can be computed as follows :

$$\begin{aligned} RE &= 0.85 \times 30,000 \\ &= 25,500 (\text{Approx.}) \end{aligned}$$

In this way, the RE can be calculated for each of risk in the risk table.

Benefits using risk exposure

- The risk exposure is useful in adjusting the final cost of the project.
- It is also useful in cost estimation of the entire project.
- It is also useful in some situation where there is risk of schedule slippage. It can predict the probable risks.
- If the risk is predicted, then easily the organization can increase the manpower in that project to recover the schedule slippage.

10.3.3 Project Plan

- Planned and controlled software projects are conducted for one and only reason because it is the only way known to manage complexity.
- Project failure rate remains higher than it should be even though the success rate for software projects has improved. One of the reasons is that the customers lose interest quickly (because what they have requested was not really as important as they first thought), and the projects are cancelled.
- The software engineers and the software project manager must follow certain guidelines for project plans in order to avoid project failure :
 - o Carefully design a set of common warning signs,
 - o Understand the critical success factors that lead to good project management, and
 - o Develop a common sense approach for planning, monitoring, and controlling the project.

Syllabus Topic : Risk Refinement

10.4 Risk Refinement

- In the beginning of project planning, a risk may be quoted in general but with the progress of time, one can be able to understand the depth of the risk. So it is possible to refine the risk into a set of more detailed risks, and it would be more convenient to mitigate, monitor, and manage.
- CTC (condition-transition-consequence) format may be a good representation for the detailed risks. The example of CTC format is given below :
- Given that <condition> then there is concern that (possibly) <consequence>.



- This general condition can be refined in the following manner:
- **Sub condition 1 :** Certain reusable components were developed by a third party with no knowledge of internal design standards.
- **Sub condition 2 :** The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.
- **Sub condition 3 :** Certain reusable components have been implemented in a language that is not supported on the target environment.
- Even though we refine the risk into more detailed risks, but the consequences for the risks remains same. It only helps in separating the part of the risk for better understanding and analysis.

Syllabus Topic : Risk Mitigation, Risk Monitoring and Risk Management (RMMM)

10.5 Risk Mitigation, Risk Monitoring and Risk Management (RMMM)

SPPU - May 12, May 14

University Questions

- Q. What is Risk Mitigation Monitoring and Management (RMMM) ? Write a note on it.
(May 2012, 10 Marks)
- Q. Write short note on : RMMM.
(May 2014, 5 Marks)

- Risk analysis actually helps the project development team to build a strategy to handle all possible risks. Following are three important issues (or steps) that must be considered for developing effective strategies :
 - o Risk avoidance (i.e. Risk mitigation)
 - o Risk monitoring
 - o Risk management and planning.

- In order to avoid the risk, the best approach is proactive approach. In the proactive approach, the development process starts with risk mitigation plan and it helps in avoiding possible risks.
- The step is risk monitoring that begins from the start of the development process. The project manager is generally responsible for keeping vigilance on the factors that can cause risk to occur.
- The project manager starts monitoring with the information received from the risk mitigation step. He scans all the documents prepared in the risk mitigation step.
- The third and final step is risk management and planning which assumes that the risk has occurred and accordingly the manager has to take necessary action.
- The Risk Mitigation, Monitoring and Management (RMMM) steps incur extra project cost.

10.5.1 The RMMM Plan

- Risk management strategies should be included in the project plan itself. The risk management plan is usually added as a separate plan in the project plan. This is referred as risk mitigation, monitoring and management plan or RMMM plan.
- The RMMM plan is executed as a separate plan to evaluate the risk. Each of the risks are documented separately.
- The RMMM plan is well documented in the beginning of the project itself. Once the project begins, the mitigation and monitoring activities are started.
- The risk monitoring has main three objectives :
 - o Check whether the predicted risks actually occur.

- o All the risk assessment steps are properly followed, and
- o Collect all the necessary information that can be useful for future risk analysis.

Following are the list of probable risks that may occur during project development process. Here in this section, we explain each of the risks mentioned below discuss them keeping RMMM Plan in mind :

- o Computer Crash
- o Late Delivery
- o Technology will not Meet Expectations
- o End Users Resist System
- o Changes in Requirements
- o Lack of Development Experience
- o Lack of Database Stability
- o Poor Quality Documentation
- o Deviation from Software Engineering Standards
- o Poor Comments in Code

(1) Risk : Computer Crash

(a) Mitigation

- o The computer crash can cause the loss of important data and ultimately it will result in failure of the project.
- o It is always recommended that software development organization should keep multiple copies of the data at multiple locations to avoid loss.

(b) Monitoring

The project manager must keep a watch on the working infrastructure and working environment before the actual development work starts.

(c) Management

Any inconvenience in working environment must be noticed immediately and a proper action should be taken to make the system stable.

(2) Risk : Late Delivery

(a) Mitigation

- o The late delivery will result in approval from the customer. If the customer is reluctant to give the approval, then the development organization will get a bad image and it will affect the organization for getting future projects.
- o So the important steps and precautionary measure are taken to timely delivery of the project.

(b) Monitoring

Continuous monitoring is required during the entire development process. The development schedule must be observed strictly and if any slippage occurs, it be noted seriously and necessary action should be taken for the recovery.

(c) Management

The delayed project can move to critical state and ultimately can cause project failure. The project manager must look into the matter seriously and should negotiate with the customer for extending the deadline as a final solution.

(3) Risk : Technology Does Not Meet Specifications

(a) Mitigation

The formal and informal meeting must be conducted with the customer to avoid this risk. It guarantees that realistic products are being developed as per customer's need.

(b) Monitoring

The proper and frequent communication is needed to understand each other and ultimately the requirement.

(c) Management

- o If the idea of the project specification does not meet the requirement narrated by the customer, then the management team must take quick action to resolve this issue.

- o A meeting must be conducted to understand the problems.

(4) Risk : End Users Resist System

(a) Mitigation

The application must be developed keeping the end user in mind. The user interface must be user friendly and convenient to operate.

(b) Monitoring

Any mismatch must be monitored immediately and it should be brought to development team.

(c) Management

- o If the system is resisted by the user then the software must be evaluated completely and it is necessary to uncover the reasons and user interface should also be investigated properly.
- o After uncovering the reasons for customer resistance, the immediate actions are required.

(5) Risk : Changes in Requirements

(a) Mitigation

To avoid the changes in requirement by the customer, it always better to keep the record of requirements given by the customer. There must be formal and informal meetings conducted between the development team and the customer.

(b) Monitoring

The meetings between the customer and the development organization must be conducted to understand each other and the requirement also.

(c) Management

To rectify any notified problem, a meeting should be conducted to discuss at the issue and the resolution of the issue.

(6) Risk : Lack of Development Experience

(a) Mitigation

The development team members must be updated and they should be well experienced to use the development tools needed.

(b) Monitoring

The team members must look into other members also to find any weak link in the chain. If such cases are observed, then it should be brought to management team to handle.

(c) Management

The experienced members must help those who are proving a weak link.

(7) Risk : Database is not Stable

(a) Mitigation

The development team should communicate the database administrator to keep it stable. All the functions, procedures and database operation must be operated error free.

(b) Monitoring

Each of the team members must monitor continuously the functioning of all database operations. If any inconvenience is reported, it must be brought to attention.

(c) Management

Any noticed problem must be resolved instantly.

(8) Risk : Poor Quality Documentation

(a) Mitigation

All the stakeholders of the project development must conduct meeting to formulate a good documentation. Any suggestions must be incorporated to enhance the quality of the documentation.

(b) Monitoring

To keep the testing and development in normal condition, the documentation must be referred time-to-time.

(c) Management

Any good opinion must be included in the documentation and any unnecessary topic should be removed.

(9) Risk : Deviation from Software Engineering Standards**(a) Mitigation**

To avoid deviation from the standards, the development team must be familiar with the entire software engineering standards. They should have proper knowledge and understanding of the process.

(b) Monitoring

The technical reviews must be taken to determine any deviation.

(c) Management

If any deviation noticed, it should be addressed immediately and the management team must guide the development team to bring them on the right track.

(10) Risk : Poor Comments in Code**(a) Mitigation**

A writing standard must be followed while coding. All the functions and sub-functions must be commented properly for better understanding of the code.

(b) Monitoring

The codes must be reviewed on a regular basis to determine any poor comments.

(c) Management

If any poor comments are observed, action must be taken to minimize the poor commenting and refining comments as necessary.

Syllabus Topic : The RMMM Plan for Case Study Project**10.6 The RMMM Plan for Case Study Project**

- We consider the Case Study "Waste Management Inspection Tracking System (WMITS)" and present the RMMM plan in the following section.

10.6.1 The general overview of RMMM Plan for WMITS

- Our goal is to assist the project team in developing a strategy to deal with any risk. For this we will take a look at the possible risks, how to monitor them and how to manage the risk.
- For software development to avoid any risk both the developer and client have to work together. Client has to spend time with the developer in the beginning phase of the software development. If client decides to change the software, meaning if client wants to add some more functions into the software or to change the requirement, this will have major effect on the development of the software.

10.6.2 The Description of Risk for WMITS**Business Impact Risk**

- This is the risk where concern is that of the not being able to come up or produce the product that has impact on the client's business.
- If the software produced does not achieve its goals or if it fails to help the business of clients improve in special ways, the software development basically fails.

Customer Risks

- This is the risk where concern is client's motivation or willingness in helping the software development team.

- If the client fails to attend meeting regularly and fails to describe the real need of the business the produces software will not be one that helps the business.

Development Risks

- If client fails to provide all the necessary equipment for the development and execution of the software this will cause the software to become a failure. So in other words customer has to be able to provide time and resources for the software development team.
- If all the requested resources are not provided to the software development team odds for the software development to fail rises greatly.

Employee Risk

- This risk is totally dependent on the ability, experience and willingness of the software development team members to create the working product. If the team members are not experience enough to use the application necessary to develop the software it will keep pushing the development dates until it's too late to save the project.
- If one or more members of the software development team are not putting in all the effort required to finish the project it will cause the project to fail. Employee risk is one of the major risks to consider while designing the software.

Process Risks

- Process risk involves risks regarding product quality. If the product developed does not meet the standards set by the customer or the development team it is a failure.
- This can happen because of the customer's failure to describe the true business need or the failure of the software development team to understand the project and then to

use proper equipment and employees to finish the project.

Product Size

- This risk involves misjudgment on behalf of the customer and also the software development team. If the customer fails to provide the proper size of the product that is to be developed it will cause major problems for the completion of the project.
- If software development team misjudges the size and scope of the project, team may be too small or large for the project thus spending too much money on project or not finishing project at all because of shortage of finances.

Technology Risk

- Technology risk involves of using technology that already is or is soon to be obsolete in development of the software. Such software will only be functional for short period of time thus taking away resources from the customer.
- Since the technology changes rapidly these days it is important to pay importance to this risk. If customer request use of software that soon to be obsolete software development team must argue the call and have to pursue customer to keep-up with current technology.

10.6.3 Risk Mitigation, Monitoring and Management for WMITS

Risk Mitigation

- It is important to have **mitigation plan** to avoid risks once and for all. Goal is to attack the risk even before it comes into existence.
- The plan will help in identifying the possible risks and to monitor them.
- In this risk concern is of under or overestimation (mainly underestimation) of the number of Function Points (FP). If we

estimate too few LOC (line of code) necessary for the project we may get wrong cost figures which can prove fatal to the software development plan.

- To avoid this from happening we will use conservative figures to reduce the probability of the risk. This means we will overestimate the LOC a little.

Risk Monitoring

- To monitor the risk here, we will keep track of the amount of functions necessary for the program throughout the entire development cycle. This will tell us if the project may come across risk in future.
- As **monitoring step** in this risk we will setup user meetings to show them the work that has been completed and to get user input on the work.

- We will have meetings every other week to present the work that has been done from the time of the last meeting. This will help team in staying in touch with the customers and will also be very efficient way to derive customer's input on the progress made.

Risk Management

- If a mistake has been made, user input on the completed work will provide us with information to fix or improve the software. We have done many meeting with the clients and plan to do meeting every two weeks; this should clear any misunderstandings between the software development team and customers.
- This is the best way to go at since the work that is done on the project is revealed during the meetings and customer gets chance to make adjustments necessary.



Software Configuration Management (SCM)

Syllabus

Software Configuration Management: The SCM repository, SCM process, Configuration management for WebApps, Case study: CVS and Subversion Tools, Visual Source Safe from Microsoft and Clear Case.

Syllabus Topic : Software Configuration Management (SCM)

11.1 Software Configuration Management (SCM)

SPPU - May 12, May 16

University Questions

- Q. Define : SCM. (May 2012, 4 Marks)
 Q. What is software configuration management (SCM) ? (May 2016, 4 Marks)

Review Question

- Q. What do you mean by software configuration ? What is meant by software configuration management ?

- Basically the output of any software process contains the information based on various inputs. These output can be broadly divided into three important categories as follow :
 - o The computer programs
 - o The work products and
 - o The data that consist of the information produced.
- All these information parts collectively called as software configuration.

If each configuration item simply led to other items, little confusion would result. During the process, change takes place which may be unfortunate. But change can occur any time and for any undefined reason. The change is the only constant.

The first law of system engineering says that "No matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle".

Following are four basic sources of change :

- o Every new business conditions and market conditions may cause the change in product requirement and business rules definitions.
- o Customers may require change in data presented by the information system. The customers may also demand various functionality delivered by their product of interest. The customers seek some new services. All these changes are customer oriented.
- o As the business grows or its downsizing may cause various changes in priorities of the project and also restructuring of the team happens.

- The budget and time scheduling constraints are also important factors for change in system or product.
- SCM is a set of activities used for managing the change during the life cycle of computer software. The software configuration management can also be considered as a software quality assurance activity during the development process.

11.1.1 SCM Basics (Configuration Management System Elements)

Review Question

Q. What are configuration management system elements ?

Four basic elements that should exist when a configuration management system is developed :

1. **Component elements** : The tools in the file management system uses the software configuration item.
2. **Process elements** : The process elements or the procedures uses effective approach towards the change management in engineering and use of computer software.
3. **Construction elements** : The automated tools are used in construction or the development process and ensuring the validated components should be assembled.
4. **Human elements** : In order to make the effective use of SCM, the team makes the use of various tools and process feature.

These elements are not mutually exclusive. For example, component elements work in conjunction with construction elements as the software process evolves. Process elements guide many human activities that are related to SCM and might therefore be considered human elements as well.

11.1.2 Baselines

- The change is the only constant in software development life cycle. The customer want to modify the requirement as the model gets ready. Since in the beginning, even customer is not fully aware of the product requirement. As the development begins, customers need lot of changes in the requirement.
- Once customers modify their requirement, it is now manager who modifies the project strategy.
- Actually as time passes, all the people involved in the product development process come to know exact need, the approach and how it will be done in the prescribed amount of time.
- The additional knowledge is required to know the exact requirement. It is very difficult for most of the software engineers to accept this statement that most of the changes are justified.
- The baseline is SCM that help in development process without affecting much the schedule and control the changes.
- The IEEE provides a baseline as follows :
“A specification and requirement that is agreed upon between customer and developer is a basis for the product development and these requirements can be changed only through change control procedures”.

11.1.3 Software Configuration Items

Review Question

Q. What are SCI's ?

- Basically SCI i.e. software configuration item is the integral part of software engineering development process. It is a part of large specification or we can say that one test case among large suite of test cases.
- In fact the SCI is a document or the program component like C++ functions or a Java applet.

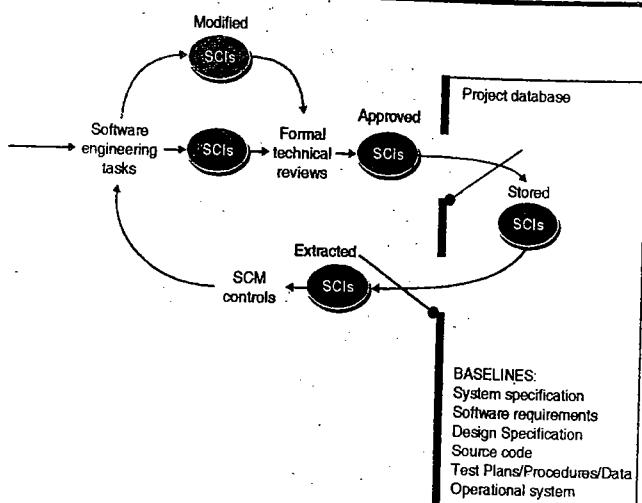


Fig. 11.1.1 : Baseline SCIs and the project database

- Most of the organizations use software tools under configuration control to help development process. In fact the editors, compilers, browsers and various automated tools are the integral part of software configuration.
- In fact the SCIs are catalogued in the project database with a single name and they form configuration objects. These objects are configuration object and it has a name, attribute and it has certain relationship with other configuration objects.
- Referring to Fig. 11.1.2, the configuration objects, Design Specification, Data Model, Component N, Source Code and Test Specification are each defined separately.

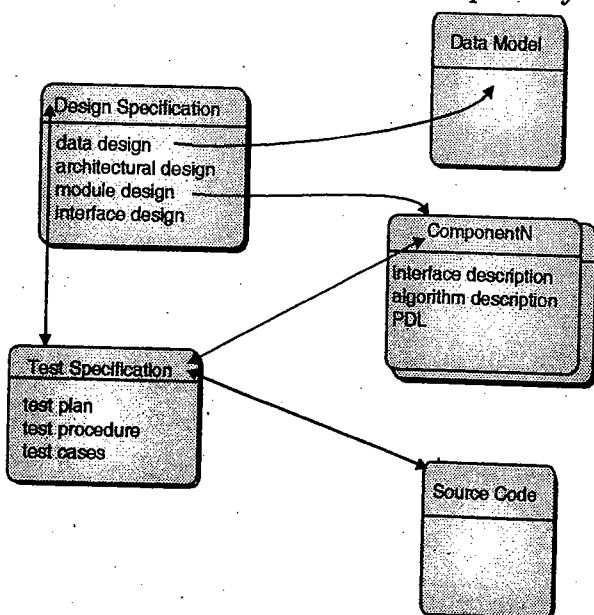


Fig. 11.1.2 : Configuration objects

Syllabus Topic : The SCM Repository

11.2 The SCM Repository

SPPU - May 12, Dec. 14

University Questions

- Q. What are the contents of SCM repository ?
(May 2012, 4 Marks)
- Q. Explain the following in brief – SCM Repository.
(Dec. 2014, 4 Marks)

Review Question

- Q. What is SCM Repository ?

- In the early days of software engineering, software configuration items were maintained as paper documents (or punched computer cards), placed in file folders or three-ring binders, and stored in metal cabinets.
- This approach was problematic for many reasons:
 - o To find a SCI is a difficult task when it is needed.
 - o To determine which items were changed and by whom. It is always challenging.
 - o To develop a new version from an existing program is prone to errors and time consuming too.
 - o To describe detailed relationship is actually impossible.
- As discussed earlier that SCIs are catalogued in the project database with a single name, they reside in the repository. The repository is a database that stores the software engineering information. The software developer or engineer interacts with the repository by using built in tools within repository.

11.2.1 The Role of the Repository

Review Question

- Q. Explain functions performed by SCM repository.

The SCM repository is the set of mechanisms and data structures that allow a software team to manage change in an effective manner. The repository will perform all the fundamental operations of database management system and in addition it will perform the following operations :

- **Data integrity :** Data integrity validates all the entries to the repository and make sure that the consistency among various objects intact and takes care of all the modifications takes place. It will also ensure cascading modifications i.e. change in one object causes change in a dependent object also.
- **Information sharing :** It is mechanism for sharing information among various developers and between various tools. These tools manage and control multi-user access to data, and locks or unlock objects to retain its consistency.
- **Tool integration :** Tool integration is a data model that can be used by many software engineering tools to control access to the data, and performs appropriate configuration management functions.
- **Data integration :** Data integration provides database functions that allow various SCM tasks to be performed on one or more SCIs.
- **Document standardization :** Document standardization is an important task for defining the objects. This standardization is a good approach for making software engineering documents.
- **Methodology enforcement :** Methodology enforcement defines an (E-R model) i.e. entity-relationship model available in the databases i.e. repository. This model may be used as a process model for software engineering. It is mandatory that the relationships and objects must define before building the contents of the repository.

To achieve these functions, the database is used as a meta-model. This meta-model exhibits the information and how this information is stored in the databases i.e. repository. This meta-model also checks data security and integrity.

11.2.2 General Features and Content

- The contents of databases and features of databases are considered from two perspective :
 - o What data is to be stored in the databases ?
 - o What services are provided by the databases ?
- A detailed breakdown of types of representations, documents, and work products that are stored in the repository is presented in Fig. 11.2.1.

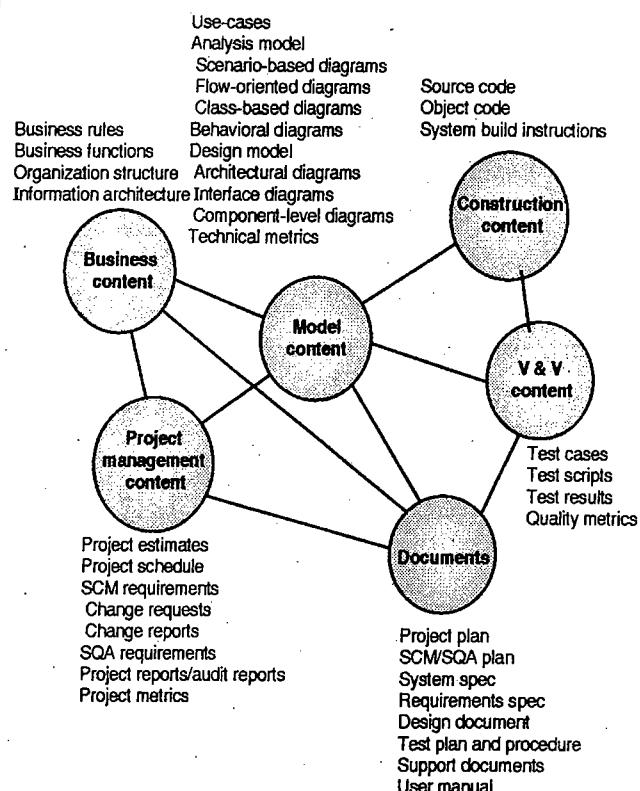


Fig. 11.2.1 : Content of the repository

A robust repository provides two different classes of services :

1. The same types of services that might be expected from any sophisticated database management system and

2. Services that is specific to the software engineering environment.
- A repository that serves a software engineering team should :
 - o Integrate with or directly support process management functions;
 - o Support specific rules that govern the SCM function and the data maintained within the repository;
 - o Provide an interface to other software engineering tools; and
 - o Accommodate storage of sophisticated data objects (e.g., text, graphics, video, audio).

11.2.3 SCM Features

Review Questions

- Q. Explain in detail SCM features.
- Q. Explain the following repository features with respect to software configuration management.
- (i) Versioning (ii) Dependency tracking
 - (iii) Requirement tracing
 - (iv) Configuration management
 - (v) Audit trails.

To support SCM, the repository must have a tool set that provides support for the following features :

1. Versioning
2. Dependency tracking and change management
3. Requirements tracing
4. Configuration management
5. Audit trails

1. Versioning

- o In the development process, as the project progresses, various versions of the product will be developed and the database will save all these versions.
- o The repository will keep track of these versions in order to make effective

management of the product delivery or the product releases.

- o The history of all releases will be used by the developers to make effective testing and debugging.

2. Dependency tracking and change management

- o The databases or the repository stores various relationships among the configuration objects.
- o The relationships may be between entities and processes, or between application design and component design and between all the design elements etc.
- o Some relationships are optional and some of the relationships are mandatory relationships that have various dependencies.
- o So to keep the track of previous history and relationships is very important for the consistency of the information present in the databases. The new releases of the final product are also dependent on the history stored in the repository. This will be useful in the improvement process.

3. Requirements tracing

- o The requirement tracing will provide the ability to trace all the design components and releases. This tracing results from a specific requirement called as forward tracing.
- o It will also be useful in finding that which requirements are fulfilled properly from the ready product. This is called as backward tracing.

4. Configuration management

- o The configuration management is a facility to keep the track of various configurations under development.

- o The series of configurations is used as the project milestones and future releases.

5. Audit trails

- o The audit trails keeps additional information (i.e. meta-data) like the changes made by whom and when. Also it stores the reasons why changes have been made.
- o The source of each change in the development must be stored in the repository.

Syllabus Topic : The SCM Process

11.3 The SCM Process

SPPU - May 12, Dec. 12, May 13, Dec. 13

University Questions

- Q. Which are the layers of SCM process ? Explain each in detail. **(May 2012, 6 Marks)**
- Q. Write short note on : Software configuration process. **(Dec. 2012, 4 Marks)**
- Q. Explain the software configuration management process. **(May 2013, Dec. 2013, 8 Marks)**

Review Question

- Q. What is the role of Software Maintenance in Software Product ?

- The SCM (Software Configuration Management) process consists of series of task to monitor the control on changes being occurred. The main objectives of these tasks are as follows :

1. To identify all individual items that can define software configuration collectively.
2. Manage the changes taking place in various individual items.
3. To handle different versions or releases of product.
4. To maintain the quality of the software under construction over the period of time.

- A process that achieves these objectives must be characterized in a manner that enables a software team to develop answers to a set of complex questions :

- o How does a software team identify the discrete elements of a software configuration?
- o How an organization manages the changes being done in existing release or the existing version? The modification should be incorporated efficiently.
- o How an organization keeps the track and control of new releases?
- o In an organization, who is responsible for authorizing all these changes?
- o The mechanism used to let others know the changes taking place and implemented?

- These questions lead us to the definition of five SCM tasks - identification, version control and change control, configuration auditing, and reporting, as illustrated in Fig. 11.3.1.

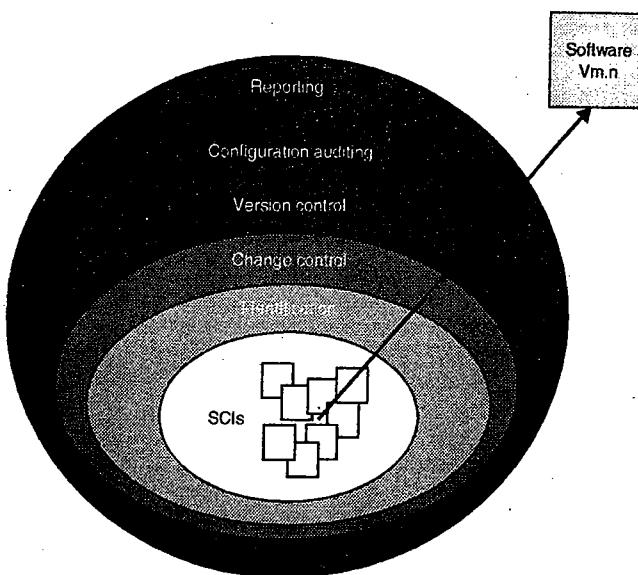


Fig. 11.3.1 : Layers of the SCM process



11.3.1 Identification of Objects in the Software Configuration

Review Question

Q. Write short note on "identification of objects in the software configuration".

- To control and manage software configuration items, each should be separately named and then organized using an object-oriented approach.
- Two types of objects can be identified :
 1. Basic objects and
 2. Aggregate objects

1. Basic object

A basic object is a unit of information that has been created by a software engineer during analysis, design, code, or test.

2. Aggregate object

An aggregate object is a collection of basic objects and other aggregate objects.

- Since each of the object in the product has some distinct features that make the object different from other objects. The object has a unique name, description and list of resources associated with it. The name of the object should be very clear and distinct.
- The description of the object should identify the type of software configuration item i.e. SCI represented by that object.

11.3.2 Version Control

Review Question

Q. Explain Version control with respect to software configuration management.

- The version control actually controls the new releases or new versions. It combines the procedures and tools in order to control various versions of configuration objects.

- Any version control management system has four major capabilities that is integrated in the version control system itself :

1. The repository will store all the related configuration objects.
2. The repository will store all the versions of configuration objects.
3. It has a facility to provide the related information about configuration objects so that a software engineer will construct a new version based on those information.
4. To track all the issues in development process by using a special tracking facility in the version control.

11.3.3 Change Control

SPPU - Dec. 12, Dec. 13, May 14, May 16

University Questions

- Q. Explain the change control process in software configuration process. (Dec. 2012, 8 Marks)
- Q. Write short note on : Change control process. (Dec. 2013, May 2014, 4 Marks)
- Q. Explain the change control mechanism in SCM. (May 2016, 4 Marks)

Review Question

Q. Explain Change control with respect to software configuration management.

- In a development of a larger software system, the changes may be uncontrolled and it leads to a complex situation. In such projects the change control is done partially by human and partially by some automated tools. In such a complex situation human intervention is very much necessary.
- The change control process is explained in the Fig. 11.3.2.
- The change request is first submitted and then evaluated by a technical support staff by taking into consideration its potential side effects and the overall impact on other objects in the product.

- The other parameters to be evaluated are system functions, the cost of project etc.
- Based on the result of the evaluation treated as a change report, the implementation is taken into consideration. This report is submitted by change control authority i.e. CCA.
- The CCA is a person or a group who has the final authority to take decision on any changes and their priority.
- After approval from CCA, a change order called ECO (engineering change order) is generated for each of the change.
- The object to be changed can be placed in a directory that is controlled solely by the software engineer making the change.
- These version control mechanisms, integrated within the change control process, implement two important elements of change management:
 - o Access control and
 - o Synchronization control.
- Before an SCI become a baseline, the changes should be applied. The developer will look after whether the changes are justified or not by project. The technical requirement must check properly.
- After approval from CCA, a baseline may be created and change control is implemented.
- Once the final product is released, the formal changes must be instituted as per the Fig. 11.3.2. This formal change is outlined.
- The CCA plays an active role in second and third layers of change control based on size of the project.

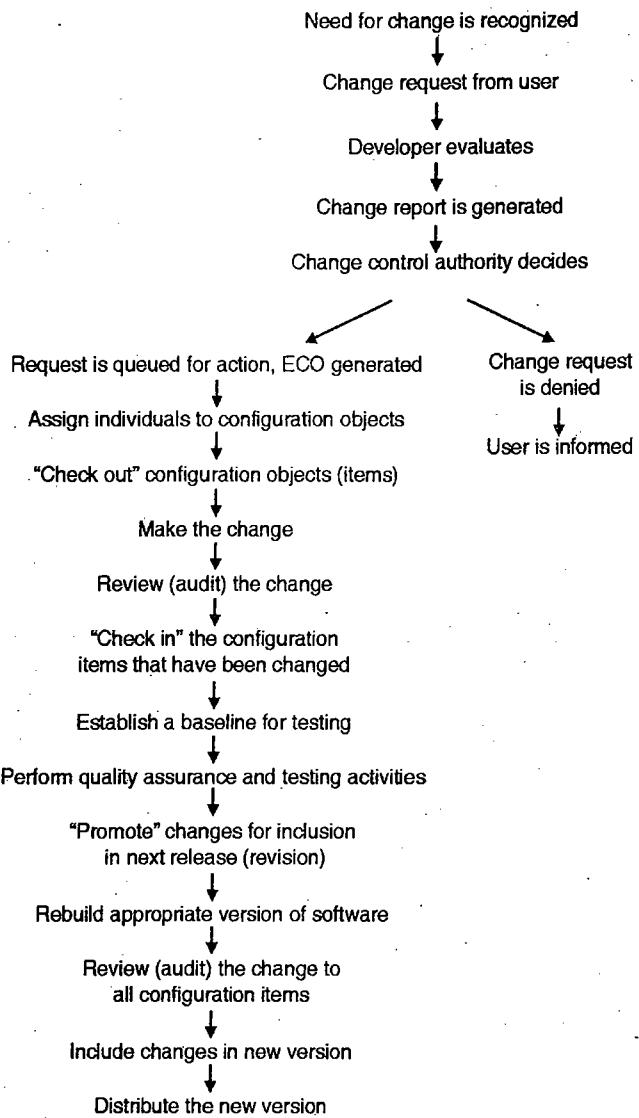


Fig. 11.3.2 : The change control process

11.3.4 Configuration Audit

Review Question

Q. Write short note on Configuration Audit.

- A software configuration audit addressees the following questions:
 - o Whether the change mentioned in the ECO applied or not ? Incorporated any additional modifications ?
 - o Whether the formal technical review conducted or not to assess technical correctness ?
 - o Whether the software process followed or not and software engineering standards properly applied ?



- Whether the changes are “highlighted” in the SCI ?
- Whether SCM procedures for noting the change, recording it, and reporting it been followed or not ?
- Whether all SCIs properly updated ?
- In some of the cases, the configuration audit questions are asked as part of a FTR (formal technical review). Still SCM is a formal activity. The SCM audit is conducted separately. These activities are performed by the quality assurance group.

11.3.5 Status Reporting

Review Question

Q. Write short note on Status Reporting.

- Configuration status reporting is a SCM task that has following questions to answer:
 - What had happened ? (Specify the changes made).
 - Who did it ? (Specify the responsible person or authority approving the changes).
 - When did it happen ? (Specify the time of occurrence of the change)
 - Anything else is affected based on the changes made?
- The CSR report is generated on regular basis to keep the developers aware of the changes made and the history of the changes made. It is very much useful in new releases or constructing new versions.

Syllabus Topic : Configuration Management for WebApps

11.4 Configuration Management for WebApps

- All the steps for the Configuration Management must be followed for the WebApps too. In addition following points must be remembered :

- Implement a versioning control system (VCS), such as SubVersion (SVN), CVS (Concurrent Versions System).
- Create a build and deployment process, even if it's just a single page.
- Block or disable FTP access to host servers,
- Always commit your files, including documents, to the project repository to maintain version control.
- Make use of version control systems deployment and version mechanisms.
- All builds and deployments to test, staging, production, or any platform should be built from the repository, create branches where specific modifications unique to a server or web app instance are needed.
- Choose a versioning control system that your IDE supports either inherently or through an add-on or plug-in.
- Do not dismiss following a formal development process as an unnecessary burden, even for small scale projects. Most development processes, such as RUP, will scale to your project size and contain invaluable disciplines.
- Tag release versions within the VCS, including beta or user acceptance testing versions, so a baseline can be established for these releases.
- Implement a backup procedure for your VCS repositories, preferably off-site, with a documented recovery plan.

Syllabus Topic : Case Study : CVS and Subversion Tools

11.5 CVS and Subversion (SVN) Tools

- CVS (Concurrent Versions System) and SVN (SubVersioN) are two version control tools that are popularly used by teams who are collaborating on a single project.

- These systems allow the collaborators to keep track of the changes that are made and know who is developing which and whether a branch should be applied to the main trunk or not.
- CVS is the much older and it has been the standard collaboration tool for a lot of people. SVN is much newer and it introduces a lot of improvements to address the demands of most people.
- Probably the biggest improvement to SVN is the addition of atomic commits. Atomic commits allow each commit to be applied in full or not at all. This can be quite useful when the server crashes in the middle of a commit.
- With SVN, the commit can be rolled back while CVS could not undo the partial commit. Another addition is the ability to cleanly rename and move the files in the repository. With SVN the files that have been renamed or removed still carry their revision history and metadata.
- CVS is also unable to push any new changes to parent repositories while it can be achieved in SVN with the use of some tools. These features are simply not supported by CVS or were not a part of its initial design and often cause a lot of problems for some people.
- SVN really works faster than CVS. It transmits less information through the network and supports more operations for offline mode. However, there is the reverse of the medal. Speed increasing is achieved basically at the expense of full backup of all work files on your computer.
- One important feature of CVS is that it allows to rollback any commit in the repository, even if this may require some time (each file should be processed independently). On the other hand, SVN does not allow rollback of commit.
- Authors suggest copy good repository state to the end of trunk to overwrite bad commit. However bad commit itself will remain in repository.

11.5.1 Comparison between CVS and Subversion (SVN) Tools

CVS	SVN
CVS is the much older and it has been the standard collaboration tool for a lot of people.	SVN is much newer and it introduces a lot of improvements to address the demands of most people.
Its slower.	SVN really works faster than CVS.
CVS is also unable to push any new changes to parent repositories while it can be achieved in SVN.	New changes can be pushed to parent repositories.
Atomic commits not available.	Biggest improvement to SVN is the addition of atomic commits. Atomic commits allow each commit to be applied in full or not at all. This can be quite useful when the server crashes in the middle of a commit.
One important feature of CVS is that it allows to rollback any commit in the repository, even if this may require some time (each file should be processed independently).	On the other hand, SVN does not allow rollback of commit.
CVS was initially intended for text data storage. That is why storage of other files (Binary, Unicode) is not trivial and requires special information, as well as adjustments on either server or client sides.	SVN manipulates all the file types and does not require your instructions.
Presently CVS is supported everywhere where you might need it.	SVN not yet so widely used, as the result there are places where it support still not implemented.



Syllabus Topic
Case Study : Visual Source Safe from Microsoft and Clear Case

11.6 Visual Source Safe from Microsoft and Clear Case

- In this section, we discuss the procedure to migrate data kept in Microsoft's Visual Source Safe (VSS) repositories into IBM Rational Clear Case repositories.
- Data is kept in VSS under a single directory tree, and all of the project paths start with a forward slash (/). Therefore, the typical project structure in VSS would look like that shown in Fig. 11.6.1.

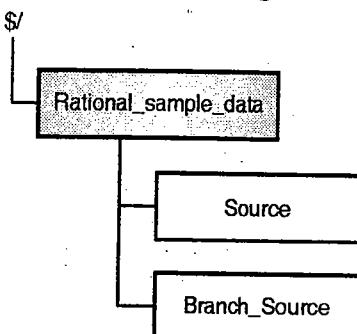


Fig. 11.6.1 : Typical project structure

- Rational_Sample_data is the product name, and it has different directories, which also include directories starting with the word "Branch." These directories are merely branched versions of the existing mainline directories.
- To elaborate, the Source directory is where mainline development takes place, and Branch_Source is the branch development of the Source directory. (This might be confusing if you are familiar with the ClearCase perspective, where you do not have separate physical directories for each branch.)
- The Source directory and the Branch_Source directory may or may not contain different data, depending upon the work being done on both of the development lines. However, the Branch_Source

directory will always exist from some version of the mainline development.

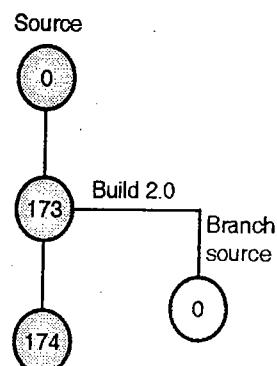


Fig. 11.6.2 : Sample development using branches

- You can use this kind of branching to fix some bugs in the released version of the product, while working in parallel with the development of the next release of the software. In Fig. 11.6.2, a label of Build.20 is applied to Version 173 of the Source directory, and a branch has been created to fix bugs on the Build 20-labeled version.
- Following are the steps involved in export the data from VSS and his can be imported in ClearCase format :

Step 1 : Export data from VSS

Perform these steps to export data from VSS:

1. Open a command prompt and ensure that the PATH variable includes the path of your VSS installation directory.
2. Map the VSS repository to a network drive, using proper authentication.
3. Set the following environment variables:
 - o set SSDIR=V: \ {mapped drive of the VSS repository}
 - o set TMP=c:\temp
 - o set SSUSER=<valid user name who has access to vss repository>
 - o set SSPWD = <VSS password of above user-id>
 - o Set the VSS project directory to this folder, which has to be exported: ss cp "\$/Article/DemoArticle1.0/Source"

- Verify the current project using this command: ss cp
4. Export the VSS directories into a text file by using this command:

```
clearexport_ssafe -r -o
```

```
c:\datafiles\vssexport_Source.txt
```

- There are certain switches available with the clearexport_ssafe command, which you can use here according to the requirements:
 - -p <date-time>: Process only versions that have been modified since date-time was specified with new metadata.
 - -s <date-time>: Process only versions that have been modified since date-time was specified.
 - -l <date-time>: Process Important versions only, but include versions created since that specified time. A version is important only if:
 - It is the most recent version
 - It has a label
5. To keep the same branching structure from VSS to ClearCase (as exhibited in Fig. 11.6.1), use the following command:

```
Clearexport_ssafe -r -s <date-time> -b
```

```
Branch_Source -v Build.20
```

```
-o c:\datafiles\vssexport_branch.
```

Where:

- Branch_Source is the Target branch in ClearCase

- Build.20 is the Label from which the Branch_Source branch has to be taken out.
- <date-time> is the Time when the Branch_Source branch was created in VSS, so that it processes only the required version on the branch.

6. If there are no errors, the output file should have the export data after successful completion of the export command.

Step 2 : Import data into ClearCase :

- In this step, you will import the data into ClearCase repositories with the help of the output data file that you created in Step 1.
- Now you can start the import process:
 1. Create the base ClearCase VOB CC_Migration_VOB (if it does not exist already).
 2. Create the base ClearCase view CC_Migration_View (if it does not exist already).
 3. Open the command prompt.
 4. Set the view context created in Step 2:

```
cleartool startview ClearCase_Migration_View
```

```
net use z:\ \\view\ClearCase_Migration_View
```

5. Browse through the VOB until you find the target location, and then execute the following command:

```
cd ClearCase_Migration_VOB
```

```
clearimport c:\datafiles\vssexport_branch.
```

Software Maintenance and Reengineering

Syllabus

Maintenance & Reengineering: Software Maintenance, Software Supportability, Reengineering, Business Process Reengineering, Software Reengineering, Reverse Engineering, Restructuring, Forward Engineering

Syllabus Topic : Software Maintenance

12.1 Software Maintenance

Review Question

Q. What is the role of Software Maintenance in Software Product ?

- The software maintenance is an activity that actually begins after the software product delivered at the client's end. In software maintenance, the modifications are carried out or the updates in the software are taken place.
- In software maintenance phase no major changes are implemented.
- In software maintenance, the changes are done in the existing program or the some small new functionality is added.
- Three decades ago, software maintenance was given the name as **iceberg**, where the potential problems reside inside and it is not visible to the clients. If maintenance is not done properly, it will sink the entire ship, as icebergs do.
- The maintenance of the software involves more than 60 % of all the efforts taken by the development team in developing the actual product.

12.1.1 Modifiability

Review Question

Q. Define Modifiability. What are types of maintenance ?

- The major cost of the software during its life cycle is the maintenance cost. When software is designed, the stakeholders want the software should be designed in such a way that future changes can easily be accommodated and implemented with less maintenance cost. If implementation is easy, then the cost of maintenance will automatically be decreased.
- Following are the queries that stakeholders come across during the early design stages :
 - o Is any alternative design possible that will reduce the cost ? Will future changes be economical ?
 - o What are types of efforts required for the early releases of the system ?
 - o What are the problems that can occur during maintenance phase ?
 - o What alternative design available that is significantly easier to implement the future modifications in the present design ?

- The modifiability means the ability of the software to be modified. The large number of definitions of quality exists. Following definitions are related to modifying system:
 - o Maintainability is defined as the ability of the software system to be modified. These modifications include : improvements, corrections, adaptability in changing environments and in changing requirements and the functional specifications.
 - o The other definition of modifiability is the ease with which a software system can be modified to accommodate changes in requirements, environments and functional specifications.
- There is difference between the maintainability and modifiability. The maintainability involves modifications in requirements as well as correction in the bugs whereas modifiability does not involve in correcting the bugs.

12.1.2 Types of Maintenance

Review Question

Q. What are types of maintenance ? Explain each in brief ?

- Following are four types of maintenance :
 1. Corrective maintenance
 2. Adaptive maintenance
 3. Perfective maintenance
 4. Preventive maintenance
- Fig. 12.1.1 illustrates the distribution of all activities into the above mentioned types of maintenance.

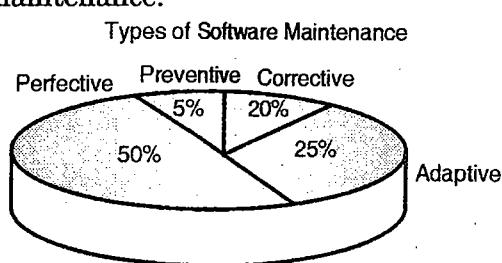


Fig. 12.1.1 Types of maintenance

12.1.2.1 Corrective Maintenance

- The corrective maintenance is the type of maintenance in which the errors are fixed when it is observed during the use of the software.
In this the errors may be caused due to faulty software design, incorrect logic and improper coding. All these errors are called as residual errors and they prevent the final specification.
- In case of system failure, the corrective maintenance approach is initiated to locate the original specification. Corrective maintenance normally used for more than 20% of all the maintenance activities.

12.1.2.2 Adaptive Maintenance

- The adaptive maintenance means the implementation of the modification in the system. The changes may be of hardware or the operating system environments.
- The examples of adaptive maintenance are:
 - o Business rules
 - o Work patterns
 - o Government policies that have substantial impact on the software.
- Adaptive maintenance normally used for more than 25% of all the maintenance activities.

12.1.2.3 Perfective Maintenance

- Perfective maintenance deals with the modified and changed user requirements. The functional enhancements are taken into consideration in Perfective maintenance.
- In Perfective maintenance, the function and efficiency of the code is continuously improved.
- Following are the examples of perfective maintenance :

- Modifying the payroll program to add new union settlement
- Adding a new report in the sales analysis system
- Perfective maintenance normally used for more than 50% of all the maintenance activities and it is the largest among all.

12.1.2.4 Preventive Maintenance

- Preventive maintenance is used to prevent the possible errors to occur. Thus in this activity, the complexity is minimized and the quality of the program is enhanced.
- Adaptive maintenance normally used for 5% of all the maintenance activities and it is the smallest among all.

12.1.3 Need of Maintenance

- **Corrective reasons :** Maintenance is done to resolve the existing errors.
- **Perfective reasons :** Developer undergo software maintenance to deals with the modifications in the requirements.
- **Adaptive reasons :** To accommodate the continuous changes.
- **Preventive reasons :** Software maintenance is done to prevent the possible errors.
- **Legal and Business reasons :** The changing business rules and changing government rules must be accommodated. This is the need for software maintenance.

Syllabus Topic : Software Supportability

12.2 Software Supportability

- If we talk about hardware reliability model then we predict failure due to wear rather than failure due to design defects i.e. physical wear due to effect temperature, corrosion, shock etc.

- But when we discuss software supportability, then the failure is related to design defects and the failure can be traced to implementation problems.
- Supportability combines the ability to extend the program (extensibility), adaptability and serviceability. All these three attributes represent a more common term, maintainability. In addition, testability, compatibility, configurability (the ability to organize and control elements of the software configuration).
- Consider a computer based system in that the measure of reliability is Mean-Time-Between-Failure (MTBF) where
$$\text{MTBF} = \text{MTTF} + \text{MTTR}$$

Where ,

$$\text{MTTF} = \text{Mean-Time-To-Failure}$$

$$\text{MTTR} = \text{Mean-Time-To-Repair}$$

- In addition the software availability is defined as the probability that a program is running as per the requirement at a given point of time.

$$\text{Availability} = [\text{MTTF} / (\text{MTTF} + \text{MTTR})] \times 100\%$$

- The availability measure is more sensitive to MTTR and it is an indirect measure of maintainability of software (i.e. Software supportability).

Syllabus Topic : Reengineering

12.3 Reengineering

Review Question

- Q. Write short note on : Re-Engineering.

- The business level reengineering is done by business experts whereas the process of reengineering software level is done by software engineers and software developers. The demands on business functions and the information technology that supports them are changing at rapid

- pace that puts huge amount of pressure on every organization.
- Both the business and the software that supports the business must be reengineered to keep pace. Business process reengineering (BPR) defines business goals, identifies and evaluates existing business processes, and creates revised business processes that better meet current goals.
 - The software reengineering process encompasses inventory analysis, document restructuring, reverse engineering, program and data restructuring, and forward engineering.
 - The intent of these activities is to create versions of existing programs that exhibit higher quality and better maintainability.
 - A variety of reengineering work products (e.g., analysis models, design models, test procedures) are produced. The final output is the reengineered business process and/or the reengineered software that supports it.
 - In order to ensure the correct approaches, use the same SQA practices that are applied in every software engineering process, formal technical reviews assess the analysis and design models, specialized reviews consider business applicability and compatibility, testing is applied to uncover errors in content, functionality, and interoperability.

12.3.1 Re-Engineering Process Model

- The business level reengineering is done by business experts whereas the process of reengineering software level is done by software engineers and software developers.
- The demands on business functions and the information technology that supports them are changing at rapid pace that puts huge amount of pressure on every organization.

- Both the business and the software that supports the business must be reengineered to keep pace. Business process reengineering (BPR) defines business goals, identifies and evaluates existing business processes, and creates revised business processes that better meet current goals.
- The software reengineering process encompasses inventory analysis, document restructuring, reverse engineering, program and data restructuring, and forward engineering.
- The intent of these activities is to create versions of existing programs that exhibit higher quality and better maintainability.
- A variety of reengineering work products (e.g., analysis models, design models, test procedures) are produced. The final output is the reengineered business process and/or the reengineered software that supports it.
- In order to ensure the correct approaches, use the same SQA practices that are applied in every software engineering process, formal technical reviews assess the analysis and design models, specialized reviews consider business applicability and compatibility, testing is applied to uncover errors in content, functionality, and interoperability.
- To a large extent, it involves maintenance activities:
 - o Understanding (predictive)
 - o Repairing (corrective)
 - o Improving (perfective)
 - o Evolving (adaptive)
- In the Fig. 12.3.1, we observe the Software Re-Engineering process model using Rapid Prototyping Process :

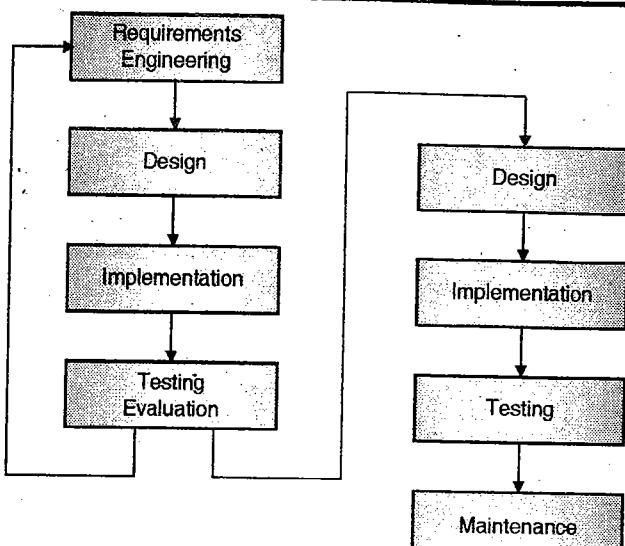


Fig. 12.3.1 : Rapid Prototyping Reengineering process

Syllabus Topic : Business Process Reengineering

12.4 Business Process Reengineering

- Business Process Reengineering involves changes in structures and in processes within the business environment. The entire technological, human, and organizational dimensions may be changed in BPR.
- Information Technology plays a major role in Business Process Reengineering as it provides office automation, it allows the business to be conducted in different locations, provides flexibility in manufacturing, permits quicker delivery to customers and supports rapid and paperless transactions.
- In general it allows an efficient and effective change in the manner in which work is performed.
- Business Process Reengineering is a management approach aiming at improvements by increasing efficiency and effectiveness of processes:
 - (1) Within public organizations
 - (2) Across public organizations
 - (3) From public organizations to businesses
 - (4) From public organizations to citizens

The Business process reengineering comprises of following steps:

Steps in Business Process Reengineering

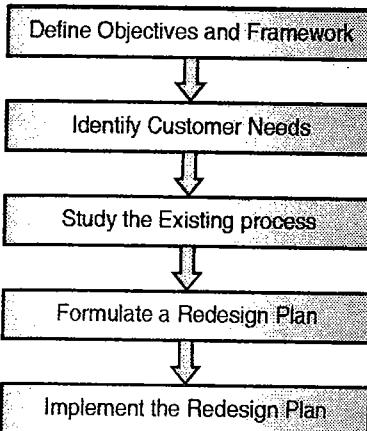


Fig. 12.4.1

(1) Define Objectives and Framework :

First of all, the objective of re-engineering must be defined in the quantitative and qualitative terms. The objectives are the end results that the management desires after the reengineering. Once the objectives are defined, the need for change should be well communicated to the employees because, the success of BPR depends on the readiness of the employees to accept the change.

(2) Identify Customer Needs : While, redesigning the business process the needs of the customers must be taken into prior consideration. The process shall be redesigned in such a way that it clearly provides the added value to the customer.

(3) Study the Existing Process : Before deciding on the changes to be made in the existing business process, one must analyze it carefully. The existing process provides a base for the new process and hence "what" and "why" of the new process can be well designed by studying the right and wrongs of the existing business plan.

(4) Formulate a Redesign Business Plan : Once the existing business process is studied thoroughly, the required changes are written down on a piece of paper and are converted into an ideal re-design process. Here, all the changes are chalked down, and the best among all the alternatives is selected.

(5) Implement the Redesign : Finally, the changes are implemented into the redesign plan to achieve the dramatic improvements. It is the responsibility of both the management and the designer to operationalise the new process and gain the support of all.

Syllabus Topic : Software Reengineering

12.5 Software Reengineering

- Over the period of time each of software application requires maintenance because some unexpected and serious side effects occur during the use.
- Therefore application must be evolved to handle such issues. In order to maintain the software, software engineering processes are reapplied using the concept of software maintenance discussed in the beginning of this chapter.
- This concept is known as software reengineering. Much of the software we use considerably old. Even when these programs were created using the best design and coding techniques known at the time, they were created when program size and storage space were principle concerns.
- They were then migrated to new platforms, adjusted for changes in machine and operating system technology and enhanced to meet new user needs, all without enough regard to overall architecture.
- The result is the poorly designed structures, poor coding, poor logic, and poor

documentation of the software systems we are now called on to keep running.

Reengineering takes time; it costs significant amounts of money; and it absorbs resources that might be otherwise occupied on immediate concerns. For all of these reasons, reengineering is not accomplished in a few months or even a few years.

Reengineering of information systems is an activity that will absorb information technology resources for many years. That's why every organization needs a pragmatic strategy for software reengineering.

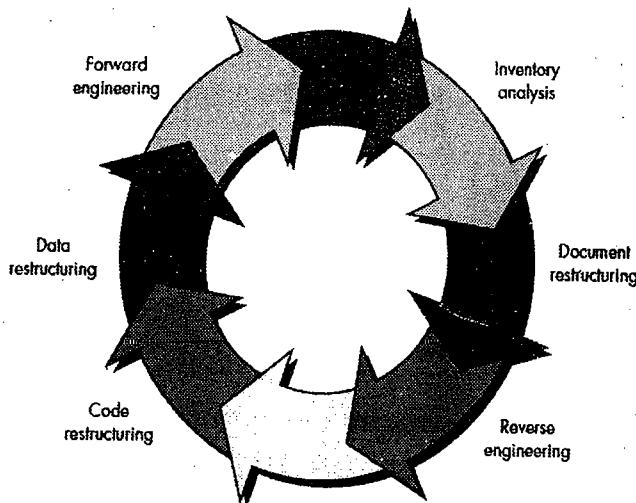


Fig.12.5.1 : A software reengineering process model

- The reengineering paradigm shown in the Fig. 12.5.1 is a cyclical model. This means that each of the activities presented as a part of the paradigm may be revisited. For any particular cycle, the process can terminate after any one of these activities.

Inventory Analysis

Every software organization should have an inventory of all applications. The inventory can be nothing more than a spreadsheet model containing information that provides a detailed description (e.g., size, age, business criticality) of every active application. It is important to note that the inventory should be revisited on a regular cycle.



Document restructuring

- Documentation must be updated, but we have limited resources. We'll use a "document when touched" approach. It may not be necessary to fully redocument an application. Rather, those portions of the system that are currently undergoing change are fully documented.

Reverse engineering

- The Reverse Engineering is a process where the system is analyzed at higher level of abstraction. It is also called as going backward through all the development cycles.

Code restructuring

- The code within the suspect modules can be restructured.

Data restructuring

- Data objects and attributes are identified, and existing data structures are reviewed for quality.

Forward engineering

- It applies to all the software engineering life cycle processes to re-create an existing application.

Syllabus Topic : Reverse Engineering

12.6 Reverse Engineering

Review Question

Q. Write short note on Reverse Engineering.

- The Reverse Engineering is the discipline of software engineering, where the knowledge and design information is extracted from the source code or it is reproduced.
- The Reverse Engineering is a process where the system is analyzed at higher level of abstraction. It is also called as going backward through all the development cycles.

- Following are some important purposes of Reverse Engineering :
 - o Security auditing
 - o Enabling additional features
 - o Used as learning tools
 - o Developing compatible products cheaper than that are currently available in the market.

- Following are three important parameters to be considered for of a reverse engineering process :

1. Abstraction Level
2. Completeness
3. Directionality

12.6.1 Abstraction Level

- In the abstraction level of a reverse engineering process, the design information is extracted from the source code. It is the highest level in the reverse engineering process. It is always expected that the abstraction level for any reverse engineering process must be high.
- When the level of abstraction is high, then it becomes easy for the software developer to understand the program.

12.6.2 Completeness

The completeness is nothing but the details available through the abstraction level of reverse engineering process. For example from the given source code it is very easy to develop the complete procedural design..

12.6.3 Directionality

- The directionality of a reverse engineering process is a method of extracting information from the source code and making it available to the software developers.

- The software developers can use this information during the maintenance activity.
- Fig. 12.6.1 exhibits the reverse engineering process.

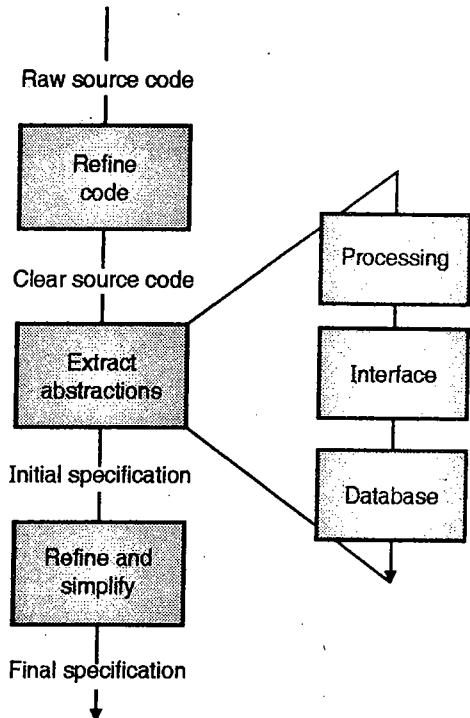


Fig. 12.6.1 : A Reverse Engineering Process

- The Reverse Engineering process include the following steps :
 - o The raw or dirty source code obtained from the abstraction level is taken as input.
 - o This code is refined or restructured. (Clean code is obtained)
 - o From the clean code, the abstraction is extracted.
 - o From this initial specification, the code is refined and simplified.
 - o Now, we get the final specification.
- Thus the final specification obtained from the reverse engineering process, is used as final specification by the software developers.

Syllabus Topic : Restructuring

12.7 Restructuring

Code restructuring

- The most common type of reengineering (actually, the use of the term reengineering is questionable in this case) is code restructuring.
- Some legacy systems have relatively solid program architecture, but individual modules were coded in a way that makes them difficult to understand, test, and maintain.
- In such cases, the code within the suspect modules can be restructured. To accomplish this activity, the source code is analyzed using a restructuring tool.
- Violations of structured programming constructs are noted and code is then restructured (this can be done automatically).
- The resultant restructured code is reviewed and tested to ensure that no anomalies have been introduced. Internal code documentation is updated.

Data restructuring

- A program with weak data architecture will be difficult to adapt and enhance. In fact, for many applications, data architecture has more to do with the long-term viability of a program than the source code itself.
- Unlike code restructuring, which occurs at a relatively low level of abstraction, data structuring is a full-scale reengineering activity.
- In most cases, data restructuring begins with a reverse engineering activity. Data objects and attributes are identified, and existing data structures are reviewed for quality.



- When data structure is weak (e.g., flat files are currently implemented, when a relational approach would greatly simplify processing), the data are reengineered.
- Because data architecture has a strong influence on program architecture and the algorithms that populate it, changes to the data will invariably result in either architectural or code-level changes.

Syllabus Topic : Forward Engineering

12.8 Forward Engineering

Review Question

Q. Write short note on : Forward Engineering.

- Consider a program with control flow that has modules with more than 2000 statements and few meaningful comment lines with suppose more than 3 Lac source statements. Let no other documentation must be modified for any requirement change.
- In this situation, we have the following options available with us :
 - o We can be lost with too many modifications and necessary design changes.

- o We can understand the working of modules and functions to make the modifications easy.
- o We can redesign the part of software that suffered with requirement modifications and recode them accordingly.
- o Or finally we can redesign from start and code again from beginning itself.

- Out of the above mentioned options no option is actually correct option.
- Whenever a software firm sells some product, preventive maintenance is taken into consideration for the new release of the application.
- The **forward engineering process** applies all the software engineering life cycle processes to re-create an existing application. In most of the cases, forward engineering is not the method to just create a new version of an older program but the new users and new technologies and new requirements are also integrated into the reengineering process.
- Thus the new software application has all the valid functionalities of the older software application and in addition the new capabilities.

Software Testing

Syllabus

Introduction to Software Testing, Principles of Testing, Testing Life Cycle, Phases of Testing, Types of Testing, Verification & Validation, Defect Management, Defect Life Cycle, Bug Reporting, GUI Testing, Test Management and Automation.

Syllabus Topic : Introduction to Software Testing

13.1 Introduction to Software Testing

SPPU – Dec. 13, Dec. 14

University Question

Q. What is software testing ?

(Dec. 2013, Dec. 2014, 4 Marks)

- Testing is an important activity in software development process. Before the start of the development, testing is planned and is conducted very systematically. In order to implement the testing, there are various testing strategies defined in the software engineering literature.
- All these strategies provide a testing template to the developers. The testing templates should possess following characteristics that is applicable to any software development process :
 - o For effective testing, formal technical reviews must be conducted by the development team.
 - o Frequent communication between developer and customer resolves most

of the complexities and confusion in the beginning itself. Thus before start of the testing process, number of errors may be uncovered and then fixed.

- o Testing starts from the component level and then finally complete computer based system is integrated at the end.
- o There are various testing techniques available. So at different point of time, suitable testing technique may be applied.
- o Testing may be conducted by the software developer itself for usually smaller projects. For the larger projects, an independent group (especially those people that are not the part of development team) may be hired for conducting an effective testing.
- o The members of independent testing group may be the member of some other team or may be outsourced.
- o Even though testing and debugging are two separate activities, debugging should be accommodated in testing strategies.

Any testing strategy should be able to conduct low level tests, since it verifies

small source code modules and can be easily implemented.

- It gives better precision. In addition to these low level tests, a testing strategy should also be able to conduct high level tests and it should be able to validate all the major functionalities as per customer's requirements.

13.2 Software Testing Fundamentals

Testability

- The software developer builds a software system or a product keeping **testability** in mind. The testability enables an individual to design the test cases with ease. He can design the effective test cases for effective testing.
- The software testability is defined as : "how easily a software can be tested effectively". It is a degree with which software supports testing. If the software testability is high, it means testing is easy.

Operability

- The meaning of **operability** is the characteristic of a system that makes it working.
- For software, it is the fact that if it works properly then the testing can be conducted more efficiently.
- The system may have some bugs, if it has no bugs, it will block the execution of tests.
- The product keep on evolving in its functionality i.e. simultaneous development and testing can be performed.

Observability

- **Observability** means an ability to see or observe. In software, the observability is an ability to see what is being tested. The

outputs are generated for each of the input values. The outputs are usually distinct.

- All the system states and variables are visible. It can be queried during the execution of the software. Also previous system states and variables are also visible and it can be done by maintaining the transaction logs.

- All the factors that affect the output are also visible to the developer. Any incorrect output can be easily identified by the developer. But the internal errors are automatically detected by using self testing mechanisms. These internal errors are also reported automatically and the source code is accessible.

Controllability

- If we can control the software properly then the testing can be automated and optimized very well. Various outputs can be generated by using some combination of input. Also the complete code is executed by using some combination of input.
- The states of software and hardware and variables used, can be controlled by the developer itself.
- The formats of input and output should be consistent and structured.
- All tests cases can be easily specified, automated, and reproduced.

Decomposability

- Controlling the scope of testing means, finding the cause of the problem quickly and once it is done, smarter retesting can be done.
- Since the software is developed from independent modules, these independent software modules can be tested separately.

Simplicity

- If amount of errors are less, then very quickly we can conduct and complete testing.



- The functional simplicity means :
 - o The functionality list provided by the customer is the minimum necessity to satisfy requirements.
- The structural simplicity means :
 - o Architecture is partitioned to control the propagation of new errors.
- The code simplicity means :
 - o Some coding standard is used for convenient and simple testing and maintenance.

Stability

- If changes or the modifications are less, then the disruptions to testing will be minimized.
- Modifications to the software are not very frequent.
- Modifications are controlled.
- Changes to the software also validate existing tests.
- The software always recovers from all types of failures.

Understandability

- If we have more information, then we can conduct tests in smart way.
- The design of the program is well understood by the user.
- All the dependencies and interrelationship among internal, external, and shared components are very well understood.
- All the modifications to the design are properly communicated.
- The technical documentation is important and it is easily accessible. It is well organized. Also it is specific and in details. It is accurate.
- Software engineer develops a software configuration i.e. programs, data, and documents. All these attributes are amenable to testing.

13.2.1 Test Characteristics (Attributes of good test)

Following are some attributes of a "good" test :

- A good test has a high probability of finding an error.
- A good test is not redundant. There is no point in conducting a test that has the same purpose as another test.
- A good test should be "best of breed", i.e. the test that has the highest likelihood of uncovering a whole class of errors should be used.
- A good test should be neither too simple nor too complex.

Syllabus Topic : Principles of Testing

13.3 Principles of Testing

SPPU - Dec. 15

University Question

Q. What are the principles of software testing ?
(Dec. 2015, 5 Marks)

A good software engineer must understand the basic principles for software testing before he can apply various methods to effective test cases. Following are the set of testing principles that have been suggested :

1. **All tests should be traceable to customer requirements :** Its single goal is to uncover faults by triggering failures. Any inference about quality is the responsibility of quality assurance but beyond the scope of testing. It follows that the most severe defects are those that cause the program to fail to meet its requirements.
2. **Tests should be planned long before testing begins :** Test planning can begin as soon as the requirements model is ready. All tests can be planned and designed before any code has been generated.

3. **The Pareto principle applies to software testing :** Pareto principle implies that 80 percent of all errors uncovered during testing will likely be traceable to 20 percent of all program components.
4. **Testing should begin “in the small” and progress toward testing “in the large.” :** The first tests planned and executed generally focus on individual components. As testing progresses, focus shifts in an attempt to find errors in integrated clusters of components and ultimately in the entire system.
5. **Exhaustive testing is not possible :** The number of path permutations for even a moderately sized program is exceptionally large. Therefore it is impossible to execute every combination of paths during testing.
6. **To be most effective, testing should be conducted by an independent third party :** The software engineer who created the system is not the best person to conduct all tests for the software. Therefore, to make the testing effective, it is advisable to conduct the testing from the third person who is not involved in the development process.

Syllabus Topic : Testing Life Cycles

13.4 Testing Life Cycles

- Software Testing Life Cycle (STLC) is the testing process which is executed in systematic and planned manner. In STLC process, different activities are carried out to improve the quality of the product. Let's quickly see what all stages are involved in typical Software Testing Life Cycle (STLC).
- Following steps are involved in Software Testing Life Cycle (STLC). Each of the steps has their own Entry Criteria and deliverable :
 - o Requirement Analysis

- o Test Planning
- o Test Case Development
- o Environment Setup
- o Test Execution
- o Test Cycle Closure

Ideally, the next step is based on previous step or we can say next step cannot be started unless and until previous step is completed. It is possible in the ideal situation, but practically it is not always true.

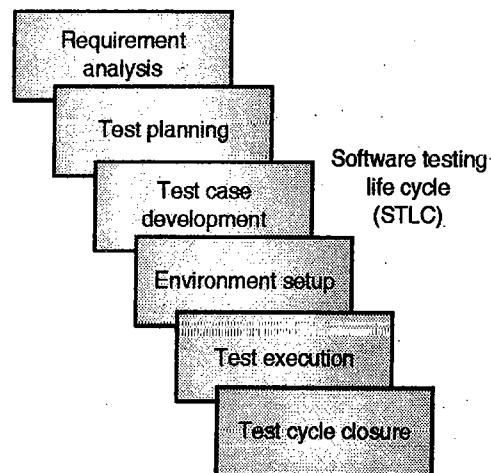


Fig. 13.4.1 : Software Testing Life Cycle (STLC)

13.4.1 Requirement Analysis

- Requirement Analysis is the very first step in **Software Testing Life Cycle (STLC)**. In this step Quality Assurance (QA) team understands the requirement in terms of what we will test & figure out the testable requirements. If any conflict, missing or not understood any requirement, then QA team follow up with the various stakeholders like Business Analyst, System Architecture, Client, Technical Manager/Lead etc. to better understand the detail knowledge of requirement.

From very first step QA involved in the where STLC which helps to prevent the introducing defects into Software under test. The requirements can be either Functional or Non-Functional like Performance, Security testing.

13.4.2 Test Planning

- Test Planning is most important phase of Software testing life cycle where all testing strategy is defined. This phase also called as Test Strategy phase.
- In this phase typically Test Manager (or Test Lead based on company to company) involved to determine the effort and cost estimates for entire project.

13.4.3 Test Case Development

- The test case development activity is started once the test planning activity is finished. This is the phase of STLC where testing team write down the detailed test cases.
- Along with test cases testing team also prepare the test data if any required for testing. Once the test cases are ready then these test cases are reviewed by peer members or QA lead.

13.4.4 Test Execution

- Once the preparation of Test Case Development and Test Environment setup is completed then test execution phase can be kicked off.
- In this phase testing team start executing test cases based on prepared test planning & prepared test cases in the prior step.

13.4.5 Test Cycle Closure

- Call out the testing team member meeting & evaluate cycle completion criteria based on Test coverage, Quality, Cost, Time, Critical Business Objectives, and Software.
- Discuss what all went good, which area needs to be improve & taking the lessons from current STLC as input to upcoming test cycles, which will help to improve bottleneck in the STLC process.

Syllabus Topic : Phases of Testing

13.5 Phases of Testing

- Each phase of testing has a different emphasis and purpose. Different phases of software testing help us to evaluate different work products of software starting at the lowest (smallest) possible item, and work towards testing the entire system as a whole.
- The following are the test phases that would allow verifying and validating lowest level structures of the software and moving towards system :
 - o Unit Testing
 - o Integration Testing
 - o Functional Testing
 - o System Testing
 - o Performance Testing
 - o Regression Testing
 - o User acceptance Testing

13.5.1 Unit Testing

SPPU - Dec. 12

University Question

Q. What is unit testing ? Explain the unit testing process. (Dec. 2012, 8 Marks)

In unit testing, the main focus is on assessment of smallest unit of the product design like component of the software or module. The unit testing has limited scope and it emphasizes on internal processing logic and data structures. Multiple modules and components can be tested in parallel.

Unit test considerations

- The unit testing is illustrated diagrammatically as in Fig. 13.5.1 :

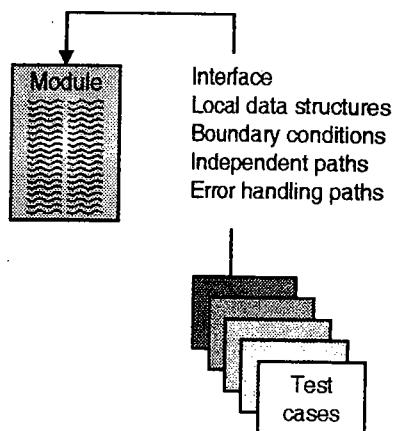


Fig. 13.5.1 : Unit test

- The test strategy is conducted on each of the **module interface** to assess the proper flow of input and output and its correctness as per the requirements.
- Next, the **local data structures** are assessed to ensure its integrity in the execution of the algorithm.
- All the **independent paths** (also called basis paths) are also evaluated through the control structure and it keeps track on the program and makes sure that at least each of the statements in a module should have been executed once.
- **Boundary conditions** are also tested so that the software works properly within the boundaries or within the limits. All the **error handling paths** are tested.
- Thus the **boundary testing** is one of the significant unit testing methodology.

Unit test procedures

- In parallel with coding step, usually unit testing is conducted. Before the start of coding, unit test can be conducted. This method is most commonly used in agile development process.
- The design information gives sufficient help for the developers to establish the test cases and uncover the errors. The developers always couple the set of results with the test cases.

Unit test environment

- The unit test environment is illustrated in Fig. 13.5.2.

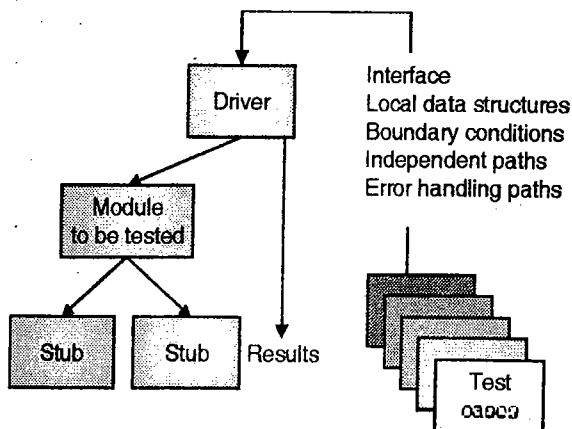


Fig. 13.5.2 : Unit test environment

- In most of the applications, a driver is considered as the "main program". This main program itself accepts all the test case data and pass that data to the component under test and the results are also printed side by side.
- In the above diagram, we observe that the stubs replace the modules of the program and are tested next to driver. The stub is also considered to be the subprogram. These subprograms make an interface with the main program to complete the test.
- Logically both the driver and stubs are the software that are written but not submitted to the customer and thus are considered as the overhead. It is always recommended to keep these overhead simple to reduce the cost. But overhead can not be made simple in all the cases and hence the unit testing can be postponed to the integration testing.

13.5.2 Integration Testing

SPPU – Dec. 13, Dec. 14, Dec. 15, May 16, Dec. 16

University Questions

- Q. Explain the integration testing approaches. (Dec. 2013, 8 Marks)**
- Q. Explain and compare incremental integration testing strategies. (Dec. 2014, 8 Marks)**

- Q.** What do you understand by the term integration testing ? Which types of defects are uncovered during integration testing ?
(Dec. 2015, 8 Marks, Dec. 16, 5 Marks)
- Q.** What do you understand by integration testing ? Explain objectives of integration testing.
(May 2016, 9 Marks)

- Integration testing is used for constructing the software architecture. It is a systematic approach for conducting tests to uncover interfacing errors. The main objective behind integration testing is to accept all the unit tested components and integrate into a program structure as given by the design.
- It is often observed that there is a tendency to attempt always non-incremental approaches. All the components are integrated in the beginning and the program is tested as a whole.
- In this approach a set of errors are encountered and correction seems to be very difficult due to isolation of causes and the complications by program since program is expanded over several modules. After correction of these errors, some new may appear and the process continues. It looks like an infinite loop.
- Small increments of the program are tested in an incremental approach. In incremental integration approach, the error detection and correction is quite simple. In this all the interfaces are tested completely and thus a systematic test strategy may be applied.
- Following are different **incremental integration strategies :**

(1) Top-down integration

SPPU - May 14, May 15

- Q.** What are the problems associated with top down approach of testing. **(May 2015, 4 Marks)**

- o Top-down integration testing is one of the incremental testing strategies for construction of the software architecture.
- o All the modules are combined by moving top to down direction through the control hierarchy. It begins with the main program or the main control module. The flow is from main module to its subordinate modules in depth-first or breadth-first manner.
- o Depth-first integration is illustrated in Fig. 13.5.3, and it integrates all the components on most of the control path of the program.

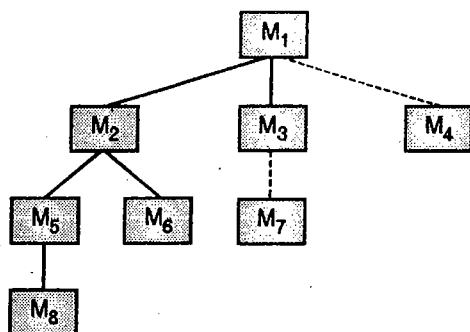


Fig. 13.5.3 : Top-down integration

Problems associated with Top down approach of testing

- In incremental integration testing approach, a set of errors are encountered and correction seems to be very difficult due to isolation of causes and the complications by program since program is expanded over several modules. After correction of these errors, some new may appear and the process continues. It looks like an infinite loop.
- Top-down integration testing is one of the incremental testing strategies for construction of the software architecture.
- The main problem with top down approach is that the test conditions are nearly impossible or they are extremely difficult to create.
- Stub modules are complicated.

University Questions

- Q.** Explain Top-down integration testing in detail.
(May 2014, 4 Marks)

(2) Bottom-up integration

SPPU – May 12, May 14

University Question

- Q. Explain bottom up integration testing strategy in detail.

(May 2012, 5 Marks, May 2014, 4 Marks)

- In bottom-up integration testing, the testing begins with the construction and components at the lowest level. These components are called as atomic modules.
- Since the components are integrated from the bottom to top, the required processing is always available for components subordinate in all the levels. Here in this approach the need of stub is completely eliminated. The use of stub was actually a disadvantage due to overhead.
- Following steps are required for implementation of bottom-up integration strategy :
 - (1) The low-level components are integrated to form clusters that can perform sub function of a specific software.
 - (2) A driver is needed to coordinate all the test case inputs and outputs.
 - (3) Later all the clusters are tested.
 - (4) Then, drivers are removed and all the clusters are integrated once again from bottom to top direction in the program.
- Integration follows the pattern illustrated in Fig. 13.5.4.

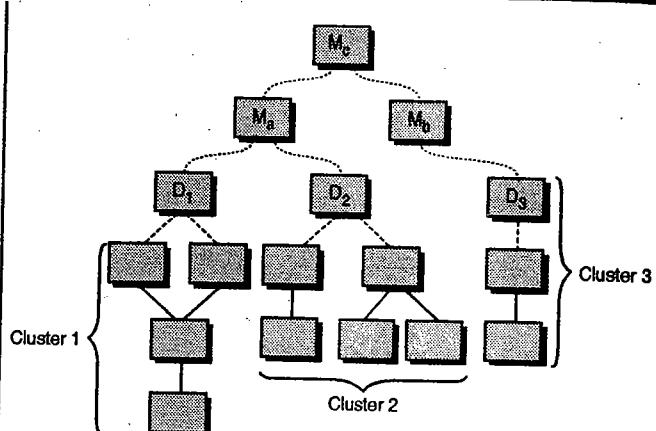


Fig. 13.5.4 : Bottom-up integration

- The components are integrated in such a fashion that they can form clusters 1, 2 and 3. These clusters are tested by using a driver (marked by a dashed block).
- The components in clusters 1 and 2 are subordinate to module Ma. The drivers D1 and D2 are then removed and the clusters are connected directly to Ma.
- In the same way, the driver D3 for the cluster 3 is removed and then cluster 3 is directly connected to module Mb. Here both the modules, Ma and Mb are finally integrated with the component Mc, and so on.

13.5.3 Functional Testing

- Functional Testing is a testing technique that is used to test the features/functionality of the system or Software, should cover all the scenarios including failure paths and boundary cases.
- Functional testing typically involves six steps :
 - The identification of functions that the software is expected to perform
 - The creation of input data based on the function's specifications
 - The determination of output based on the function's specifications
 - The execution of the test case



- o The comparison of actual and expected outputs
- o To check whether the application works as per the customer need.
- Functional testing does not imply that you are testing a function (method) of your module or class. Functional testing tests a slice of functionality of the whole system.
- Functional testing has many types :
 - o Smoke testing
 - o Sanity testing
 - o Regression testing
 - o Usability testing

13.5.4 System Testing

- The “**finger-pointing**” is a classic system testing problem. It occurs whenever an error is detected, and for each of the system element, the developers blame each other for the problem.
- **System testing** is a series of different tests and the main reason of these tests is to test the complete computer-based system. The purpose of each of the tests is to verify all the system elements properly and ensure they are integrated properly and perform required functions.

13.5.5 Recovery Testing

SPPU – Dec. 13

University Question

Q. What do you mean by recovery testing ?
(Dec. 2013, 3 Marks)

- The computer-based systems may recover from faults that are occurring and can resume back the normal processing within a stipulated time. But in some cases, the system should be fault tolerant i.e. it should continue to functioning despite failure.
- In some other cases, the system must recovered back from the failure and corrected within a stipulated period of time.

- If not done so, then some drastic economic damage will take place. For example online shopping sites, banking sites where the data is very sensitive and critical.
- In recovery testing, a system is forcefully sent to failed state in a variety of way and testing is done to recover from those states. The motive is to test that the system is recovering properly from the failure state.
- If it is an automatic recovery, then re-initialization, data recovery, restart and check-pointing mechanisms are evaluated for correcting the errors. If the recovery requires human intervention, then the Mean-Time-To-Repair (i.e. MTTR) is calculated to find whether it is in acceptable limits or not.

13.5.6 Security Testing

- All the computer-based systems that handles sensitive information or causes the actions that can harm or benefit any individuals is tested for illegal attacks to the system.
- The examples of illegal activities are :
 - o Hackers who try to attack systems for sport
 - o Detained employees who try to attack the system for revenge
 - o Some mischievous individuals who try attack system for illicit personal gain.
- Thus security testing verifies whether a protection mechanism is built or not. The mechanism should protect the system from illegal attacks and unauthorized access.
- In security testing, the tester may work as an intruder that tries to attack the system to check its security. The tester can do anything to check the system's security. For example the tester may try to obtain passwords and may attack the system to break down the defences that were constructed.

- Overload the system by series of requests to cause denial of service attack.
 - Purposely cause system errors.
 - Attack the system during recovery.
 - Browse through insecure data.
 - Find the key to enter into the system.
- Good security testing will definitely attack the system.

13.5.7 Stress Testing

- In stress testing, the test cases are conducted to bring the programs into abnormal situations purposely. Why this is done ? Because before the delivery, the attempt is to uncover every possible error by trying to fail the system by taking abrupt inputs.
- Stress testing is executed in such a way that the system demands resources to be in abnormal frequency, volume or quantity.

For example

- Input data rates are increased to determine how input will respond.
- Test cases that cause excessive searching of data on disk are created.
- Test cases that cause problems in memory management are created.
- Tests cases are created that generate interrupts.
- Test cases are created that require excessive memory or other excessive resources.

13.5.8 Performance Testing

SPPU – Dec. 13, Dec. 14

University Question

- Q. What do you mean by performance testing ?**
(Dec. 2013, 3 Marks, Dec. 2014, 4 Marks)

- Performance testing is generally conducted for real-time and embedded systems. These

are the software that can not compromise the performance requirements.

- If the software is not functioning as per the requirements, then it is unacceptable. Performance testing is conducted at the run-time and its performance is assessed in the context of an integrated system.
- Performance testing occurs throughout the testing process. Right from unit level testing, the performance testing is conducted. Each of individual modules is performance tested. In the end all the system elements are integrated into one complete system and the actual performance of the system is evaluated.

13.5.9 Verification and Validation

- Verification and validation is a set of activities that ensures that all customer's requirements are fulfilled by the software and all the functions are implemented and working correctly.
- The validation ensures that the software is developed and it is traceable to all the customer's requirements.

13.5.10 Regression Testing SPPU – May 13

University Question

- Q. Explain regression testing. (May 2013, 3 Marks)**

- Regression testing is incremental integration approach and in this every time when a new module is added as a part of integration testing, the software architecture changes.
- Every time new data flow paths are established and new I/O may appear and the new control logic is invoked. These frequent changes cause the problems with the functions that worked earlier without any error.
- In integration test strategy, regression testing may be the re-execution of some of the subset of tests.

- Thus already tested components may be tested again to ensure the errors are not having bad effects on the overall program structure.

13.5.11 User acceptance Testing

SPPU – May 13, May 14, Dec. 14

University Questions

- Q. What do you mean by acceptance testing ?
(May 2013, 4 Marks)
- Q. Explain acceptance testing.
(May 2014, 3 Marks)
- Q. What is acceptance testing ?
(Dec. 2014, 4 Marks)

- The acceptance test is conducted by the end-user rather than software developer. The need is to find the range of input values by the customer. Thus the acceptance test is well planned and is executed systematically. The series of acceptance test are usually conducted to validate the requirements.
- The acceptance testing is conducted over a period of weeks or months and thereby detecting the cumulative errors. These errors may degrade the system over time.
- If the software is built as a product that is to be used by different customers, then it is practically not possible to perform formal acceptance tests with each of the user.
- In short, when customized software is developed for one customer, then a series of acceptance tests are necessary to validate all the requirements by the customer.
- The acceptance testing can be classified as :
 - o Alpha testing
 - o Beta testing

13.5.12 Alpha and Beta Testing SPPU – Dec. 16

University Question

- Q. What is the difference between alpha testing and beta testing ?
(Dec. 2016, 5 Marks)

- It is very difficult for a software developer to visualize how the end user will actually use his program. The instructions for use may be read but not understood properly or some wrong meaning can be drawn.
- Also some combinations of input may be regularly used and the output that is correct for tester but it is not satisfied by the user in actual practice.
- When customized software is developed for one customer, then a series of acceptance tests are necessary to validate all the requirements by the customer.
- The acceptance test is conducted by the end-user rather than software developer. The need is to find the range of input values by the customer. Thus the acceptance test is well planned and is executed systematically. The series of acceptance test are usually conducted to validate the requirements.
- The acceptance testing is conducted over a period of weeks or months and thereby detecting the cumulative errors. These errors may degrade the system over time.
- If the software is built as a product that is to be used by different customers, then it is practically not possible to perform formal acceptance tests with each of the user.
- In such a scenario, most of the software developers use **alpha and beta testing** to find those errors that can be detected by the end-user only.
- The **alpha test** is conducted at developer's end by end-users. The end users work on the software and developer has a keen observation on all interactions of user with the software in an environment that is controlled by the developer. The developer keep on recording all the errors and problems encountered. Alpha tests are generally conducted in a controlled environment.



- The beta test is conducted at end-user place. Here the developer is generally not present. Hence the beta test is supposed to be the “live” application of the software.
- This environment is not controlled by the developer in contrast to the alpha testing that is done at developer’s end.
- In beta testing, the end-user will record all the problems (may be a real one or imaginary) that are encountered and will report to the developer at regular basis.
- After getting the reports from the end user, the developer will make modifications and then prepare the new release of the software and deliver to the customer.

Syllabus Topic : Types of Testing

13.6 Types of Testing

- Any of the software applications can be tested in the following two ways :
 - o It assures that all the functions specified by the customer are fully functional and again searching for more errors in each of the functions.
 - o All the internal operations are working as per the specifications. All the internal components have been traversed properly to uncover the errors.
- In the above two approaches, first test approach is called as black box testing and the second approach is known as white box testing. Thus we can categorize the testing approaches into the following two types :
 - o White-Box Testing
 - o Black-Box Testing

13.7 White-Box Testing

SPPU – Dec. 16

University Question

- Q. What do you understand by white box testing ?
(Dec. 2016, 7 Marks)

- White-box testing also called as glass-box testing. It employs the test case design concept that uses the control structures. These control structures are described as component-level design to derive the test cases.

- By using the white-box testing, the software developer derive test cases that has the following characteristics :

- o It guarantees the traversal of all independent paths in a module at least once. It refers the concept of traversal of tree i.e. each of the nodes in a tree must be traversed at least once.
- o Evaluate all logical decisions and find whether they are true or false.
- o Evaluate all the loops to check their boundaries and operational bounds, and
- o Evaluate all internal data structures to confirm their validity.

- Different white-box testing methods are :

1. Basis Path Testing
2. Control Structure Testing

13.7.1 Basis Path Testing

SPPU – May 12, Dec. 12, May 14,
Dec. 14, May 15, Dec. 16

University Questions

- Q. What is basis path testing ? What is cyclomatic complexity ? How is it determined for a flow graph ? Illustrate with an example.
(May 2012, 8 Marks)
- Q. Basis path testing covers all statement in program module. Justify with example.
(Dec. 2012, 8 Marks , Dec. 2016, 7 Marks)
- Q. What is cyclomatic complexity ? How is it determined for a flow graph ? Illustrate with an example.
(May 2014, 8 Marks)
- Q. How test cases are derived in basis path testing ?
(Dec. 2014, 6 Marks)
- Q. What are the main objective of basis path testing ? Explain in detail. (May 2015, 9 Marks)

- Basis path testing use the concept of white-box testing. It enables the developer to design test cases in order to derive a logical complexity measure of a procedural design. To define a basis set of execution paths, the developer uses this complexity measure as a guide.
- In this, test cases designed are guaranteed to execute all the statements in the program at least once.
- Following are the examples of basis path testing :

1. Flow graph notation
2. Independent program paths
3. Deriving test cases
4. Graph matrices

13.7.1.1 Flow Graph Notation

- The flow graph depicts logical control flow using the notation illustrated in Fig. 13.7.1.

- Each structured construct has a corresponding flow graph symbol.
- To illustrate the use of a flow graph, we consider the procedural design representation in the Fig. 13.7.2(a).
- In the Fig. 13.7.2, to show the program control structure, a flowchart is used. Fig. 13.7.2(b) there is mapping of flowchart into a flow graph.
- Refer Fig. 13.7.2(b), to show the flow graph node are represented by circles. These circles represent one or more procedural statements.
- A single node encompasses a sequence of process boxes and a decision diamond.
- The edges or links represent flow of control and are denoted by the arrows on the flow graph. These arrows are similar to flowchart arrows.

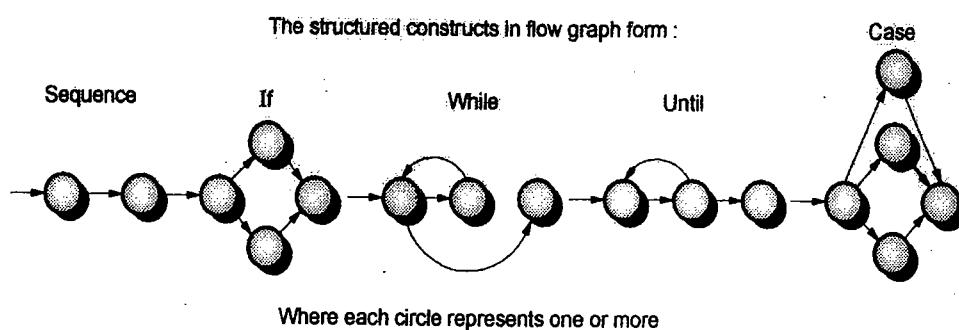
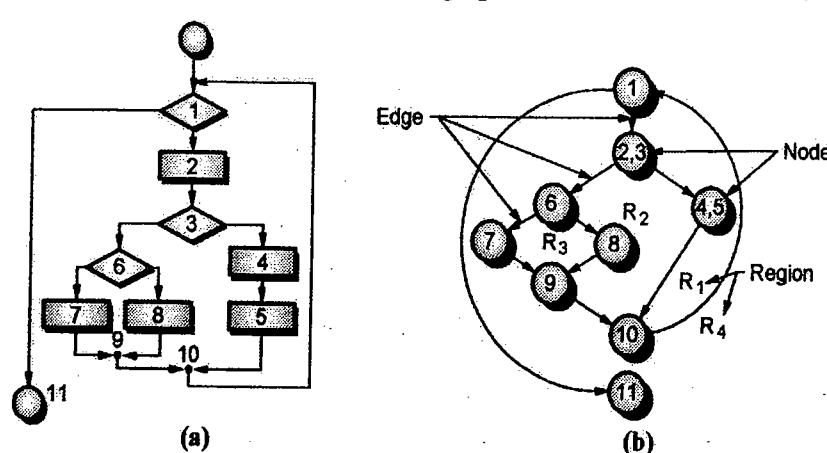


Fig. 13.7.1 : Flow graph notation



(a) Flowchart

(b) Flow Graph

Fig. 13.7.2

- Even if, a node does not represent any procedural statements, an edge must terminate at a node.
- The areas surrounded by edges and nodes are referred as regions. When we consider regions, we should include the area outside the graph as a region.
- Whenever in a procedural design, the compound conditions are encountered, the creation of a flow graph becomes more complex.
- In a conditional statement, a compound condition is encountered whenever one or more Boolean operators are present. The examples of Boolean operators are logical OR, AND, NAND, NOR etc.

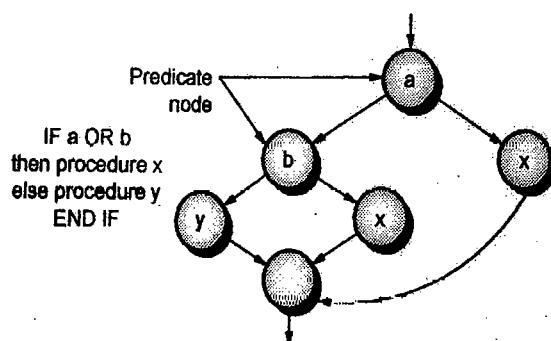


Fig. 13.7.3 : Compound logic

- The PDL (Program Design Language) segments are translated into the flow graph as shown in Fig. 13.7.3. For each of the conditions a and b, a separate node is created in the statement (i.e. IF a OR b).
- In the above flow graph, each node that contains a condition is known as predicate node. The predicate node is characterized by two or more edges coming from it (predicate node).

13.7.1.2 Independent Program Paths

- An independent path is defined as a path in the program that introduces at least one new set of processing statements or introduces a new condition.

- When an independent path is stated in terms of a flow graph, it must move along at least one edge and that edge should not have been traversed before defining the path.
- For example, A set of independent paths for the flow graph are illustrated in Fig.13.7.2 :

Path 1 : 1→1

Path 2 : 1→2→3→4→5→10→1→11

Path 3 : 1→2→3→6→8→9→10→1→11

Path 4 : 1→2→3→6→7→9→10→1→11

- We observe that, each of the new path introduces a new edge.

The path :

1→2→3→4→5→10→1→2→3→6→8→9→10→1→11

is not an independent path since it is a combination of some previously specified paths only. It has not traversed any new any new edge.

- The paths 1, 2, 3, and 4 are considered as a basis set for the flow graph in Fig. 13.7.2, i.e. if test cases are designed to execute these paths (i.e. a basis set), then each of the statements in the program will be executed at least once, and these condition will be executed to find whether they are true or false.
- The basis set is not unique. Actually, different basis sets can be derived for a given procedural design.
- **Cyclomatic complexity** provides quantitative measure of the logical complexity of the program. The **Cyclomatic complexity** is a software metrics.
- In the context of the basis path testing, the value of Cyclomatic complexity is calculated and they are the independent paths in the basis set. The Cyclomatic complexity provides an upper bound for the test cases

and it guarantees that all the statements are executed at least once.

- The Cyclomatic complexity is considered as a foundation in graph theory. It is calculated in one of the following three ways :

- o How many regions are related to the Cyclomatic complexity ?
- o Cyclomatic complexity $V(G)$ for a graph G is defined as :

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

- o In the above equation, E = the number of flow graph edges, and
 N = the number of flow graph nodes.
- o Cyclomatic complexity, $V(G)$, for a flow graph, G , is also defined as :

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

- o In the above equation, P = the number of predicate nodes contained in the flow graph G .

- Referring once more to the flow graph in Fig. 13.7.2, the Cyclomatic complexity can be computed using each of the algorithms just noted :

- (1) The flow graph has four regions.
- (2) $V(G) = 11 \text{ edges} - 9 \text{ nodes} + 2 = 4$.
- (3) $V(G) = 3 \text{ predicate nodes} + 1 = 4$.

13.7.1.3 Deriving Test Cases

- The basis path testing method can be applied to a procedural design or to source code. In this section, we present basis path testing as a series of steps.

- The following steps can be applied to derive the basis set :

- o Using the design or code as a foundation, draw a corresponding flow graph.

- o Determine the Cyclomatic complexity of the resultant flow graph.
- o Determine a basis set of linearly independent paths.
- o Prepare test cases that will force execution of each path in the basis set.

- One important point to remember is that some of the independent paths cannot be tested alone. But the combination of data is required to traverse the path. This can not be achieved in the normal flow of the program. In these cases, the independent paths are tested as part of another path test.

13.7.1.4 Graph Matrices

SPPU – May 13

University Question

Q. Explain graph matrix. (May 2013, 4 Marks)

- A graph matrix is a tool that supports basis path testing. A graph matrix is defined as a square matrix whose size is equal to the number of nodes on the flow graph. The size of matrix means the number of rows and columns of the square matrix.

- Each of the row and column of graph matrix corresponds to an identified node. The matrix entries as shown in Fig. 13.7.4 correspond to an edge between nodes (i.e. connections). An example of a simple flow graph and its corresponding graph matrix is shown Fig. 13.7.4.

- Each of the nodes on the flow graph is denoted by some numbers and the edges are represented by some letters as shown in following Fig. 13.7.4.

- **For example :** Node 3 in the following diagram is connected to the node 4 by the edge b. This can be represented in graph matrix by writing "b" in the corresponding cell.

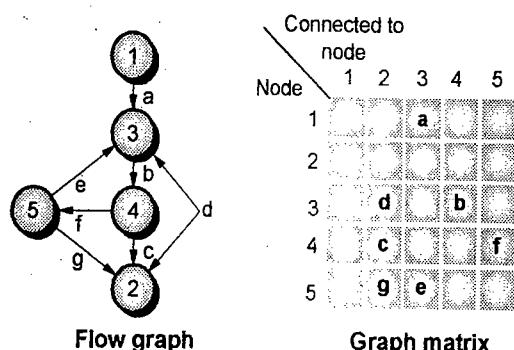


Fig. 13.7.4 : Graph Matrix

- The graph matrix is actually a tabular representation of a flow graph. By adding a link weight to each of the matrix entry, it becomes a powerful tool for testing program control structure.

13.7.2 Control Structure Testing

SPPU - May 16

University Question

- Q. Differentiate between condition and loop testing. (May 2016, 4 Marks)

- Basis path testing is one of the techniques for control structure testing.
- Since basis path testing is not sufficient, other variations on control structure testing are discussed in the following sections. The examples of control structure testing are :

1. Condition Testing
2. Data Flow Testing
3. Loop Testing

13.7.2.1 Condition Testing

- The condition testing is one of the test case design method. In condition testing, all the logical conditions in the program or the program module are tested.
- An example of a simple condition may be a Boolean variable or a relational expression. The expression may contain a single NOT operator.
- Any relational expression has the following simple form :

$$E_1 < \text{the relational operator} > E_2$$

Where, E_1 and E_2 are two arithmetic expressions and between them, any of the following relational operator may be present :

$<$ → Less than

\leq → less than or equal to

\neq → Non-equality

$>$ → Greater than

\geq → Greater than or equal to

A condition may also be a compound condition and it consists of two or more simple conditions, Boolean operators and parentheses.

13.7.2.2 Data Flow Testing

- In dataflow testing approach, a test paths of a program is selected based on the locations of definitions and uses of variables in the program.
- To explain the data flow testing method, consider that each of statement in a program is assigned a unique number. Also the functions and global variables do not modify their parameters.

- Consider a statement S (its unique statement number) as follows :

$$\text{DEFX}(S) = \{X \mid \text{statement } S \text{ contains a definition of } X\}$$

$$\text{USEX}(S) = \{X \mid \text{statement } S \text{ contains a use of } X\}$$

- If statement S is an "if" or "loop" statement, then its DEFX set is empty and its USEX set depends on the condition of statement S .
- The variable X at statement S is the live at statement S' , if there is a path from statement S to statement S' . Also it should not contain any other definition of X .

13.7.2.3 Loop Testing

SPPU – May 13

University Question

- Q. Explain Loop Testing methods.

(May 2013, 4 Marks)

- Loops are the important part of nearly all the programs in the software. Normally the developers pay very little attention this while conducting tests.
- The loop testing technique is in the category of a white-box testing. It focuses mainly on the validity of loop constructs. In the following section, we define four classes of loops :

1. Simple loops

Following tests can be applied to simple loops, where n is the maximum number of passes through the loop :

- o Skip the whole loop.
- o Only one pass allowed.
- o Only two passes allowed.
- o m passes allowed through the loop where $m < n$.
- o Also, the possibilities of $n - 1$, n and $n+1$ passes allowed.

2. Concatenated loops

This concatenated loop approach can be implemented by using the approach discussed in simple loops. In this each of the loops is independent of other. When loops are not independent, this approach is not suitable and instead nested loop approach is suggested as discussed in the next point.

3. Nested loops

The nested loop approach is an extension of simple loop approach. By nesting the loops, the number of testing levels will be increased automatically. This might result in number of tests which may not be practically possible. Following are some approaches that are suggested to reduce the number of tests :

- o Start with the inner most loops and set all the other loops at minimum levels.
- o Now conduct the simple loop approach to innermost loop while hold other with their minimum interaction value.
- o Move outward to conduct simple test to the next loop and keeping its out loop value at minimum.
- o Continue the same process till all the loops are exercised.

4. Unstructured loops (Fig. 13.7.5)

Whenever it is required, the loops should be redesigned by using structure using structured programming constructs.

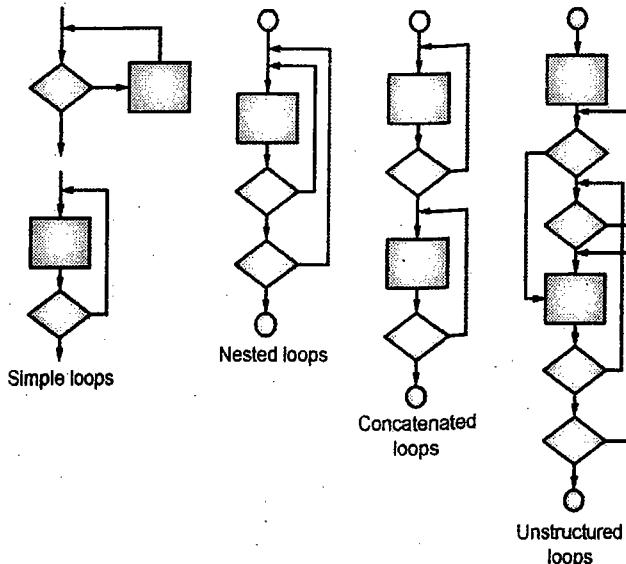


Fig. 13.7.5 : Classes of loops

13.8 Black-Box Testing

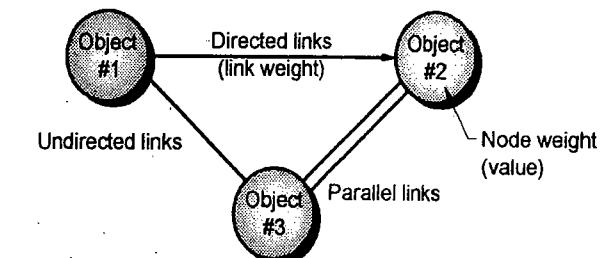
- Black-box testing is also known as behavioural testing. It focuses only on the functional requirements of the software.
- The black-box testing helps the software developer to derive the sets of input conditions. These input conditions exercise all the functional requirements for a program.
- Black-box testing is a different approach and it can not be the alternative or the replacement of white-box testing approach.

- It is a complementary testing method that detects a different class of errors compared to white-box testing methods.
- Following are categories of errors that can be detected by using black-box testing approach :
- o The performance errors or the behavioural errors
 - o Missing or incorrectly defined functions
 - o Data structures incorrectly defined and external data base access errors,
 - o Errors occurred during initialization and termination.
 - o Errors occurred during interface.
- In contrast to white-box testing, where testing begins early in the testing process, the black box testing is applied in the final stages of testing.
- Different black-box testing methods are :**
1. Graph-Based Testing Method
 2. Equivalence Partitioning
 3. Boundary Value Analysis
 4. Orthogonal Array Testing

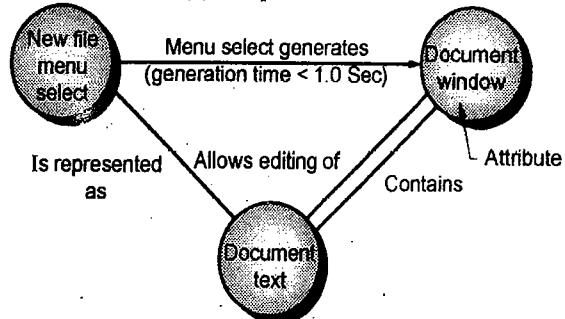
13.8.1 Graph-Based Testing Method

- The software testing starts with drawing the graphs of some important objects and their relationships. After this, the software developer performs the series of tests that cover the graph in such a way that each of the object and its relationship is covered and errors are detected.
- In order to complete these steps, the software developer starts with a graph that represents collection of objects and the relationships among them. The nodes of the graph are objects and links denotes relationships.

- In Fig. 13.8.1(a), a symbolic representation of a graph is exhibited. The circles in the graph are nodes and they are connected by links. There are different types of links as follows :
- o A **Directed link** denoted by an arrow, indicates that a relationship can moves in one direction only.
 - o A **bi-directional link**, it is also called as symmetric link i.e. the relationship is applied in both the directions.
 - o A **Parallel link**, is used in cases where various relationships exist among graph nodes.



(a) Graph notation



(b) Simple example

Fig. 13.8.1

13.8.2 Equivalence Partitioning

- Equivalence partitioning is a testing approach that employs black-box testing method. It divides the input domain of a program into various classes of data. Then these classes are used for deriving the test cases.
- In an ideal test case, this approach detects all the errors single-handedly from different classes of errors.



- The equivalence partitioning approach defines a test case that detects classes of errors and thus reducing the efforts of conducting more test cases.
- The equivalence classes that are generated by partitioning are defined according to the following guidelines :
 - o If the input conditions specify a range then one valid and two invalid equivalence classes can be defined.
 - o If an input condition requires some specific value then also one valid and two invalid equivalence classes are defined.
 - o If an input condition specifies a member of a set then in this case, one valid and one invalid equivalence class are defined.
 - o If an input condition is a Boolean value, then also one valid and one invalid class are defined.
- By applying these guidelines for the derivation of equivalence classes, test cases for each input domain data object can be developed and executed.

13.8.3 Boundary Value Analysis

SPPU - Dec. 16

University Question

Q. Explain boundary value analysis testing.

(Dec. 2016, 3 Marks)

- Generally large number of errors occurs at the boundaries of the input domain and not at the "center" of it.
- This is the reason why we require Boundary Value Analysis (BVA). It has been developed and used as a testing technique by the developers. Boundary value analysis selects and conducts different test cases to test the boundary values of input domain.

BVA is a test case design method in which the equivalence partitioning takes place. In spite of selecting some element of an equivalence class, the boundary value analysis actually selects the test cases at the "edges" of the class.

Here in boundary value analysis approach, the focus is not only on input conditions but also it derives and conducts the test cases from the output values.

Following are some guidelines for boundary value analysis approach :

- o If an input condition select a range between a and b then test cases should be designed beyond the values a and b. It means the input values may be just below and just above the values of a and b.
- o If an input condition selects the minimum and maximum numbers, then values just above and just below the minimum and maximum values are also tested.
- o The above two guidelines are applied to output conditions.
- o If internal program or its data structures prescribed some boundary values like an array of 100 numbers, then the care must be taken to conduct the test cases up to 100 entries only i.e. its boundary.

Most of the software developers perform boundary value analysis up to some degree only. After applying all these guidelines on boundary value analysis testing, then it will give more complete testing and higher probabilities of finding errors.

13.8.4 Orthogonal Array Testing

SPPU – Dec. 16

University Question

Q. Explain Orthogonal Array testing.

(Dec. 2016, 3 Marks)

- In various applications, the input domain is relatively limited and hence **orthogonal array testing** is a good option in such situations. In orthogonal array testing, the problems in which the input domain is small but it is large enough to accommodate the exhaustive testing.
- The orthogonal array testing approach is especially used in detecting errors that are caused by the faulty logic within a software component.
- To explain the comparison between orthogonal array testing and more conventional approaches, consider the following system in which there are three input values X, Y and Z.
- All these input values have three discrete values associated with each of it. The probability of test cases is $3^3 = 27$.
- In the following Fig. 13.8.2, a geometric view of the possible test cases associated are explained by considering the input values i.e. X, Y, and Z.

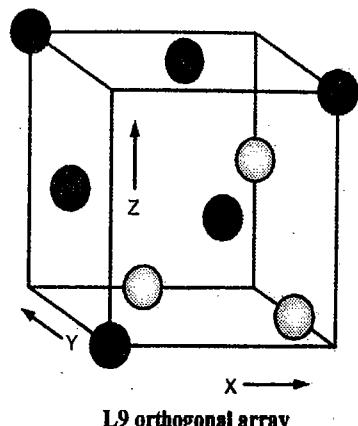
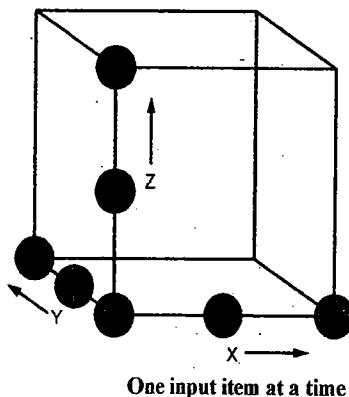


Fig. 13.8.2 : A geometric view of test cases

- Refer the Fig. 13.8.2, in that one input value at a time is changing in accordance with the input axis. The results obtained are generally having less coverage of the input values.
- An L9 orthogonal array of test cases is created whenever an orthogonal array testing is conducted. Generally a L9 orthogonal array has an important property called as **balancing property**. In this the test coverage more complete with the input domain.

13.8.5 Differentiation between White-box and Black-box Testing

SPPU – May 12, Dec. 12, May 15

University Question

- Q. Differentiate between white box and black box testing.

(May 2012, Dec. 2012, May 2015, 4 Marks)

Sr. No.	White box testing	Black box testing
1.	White-box testing also called as glass-box testing. It employs the test case design concept that uses the control structures. These control structures are described as component-level design to derive the test cases.	Black-box testing is also known as behavioural testing. It focuses only on the functional requirements of the software.
2.	It guarantees the traversal of all independent paths in a module at least once. It refers the concept of traversal of tree i.e. each of the nodes in a tree must be traversed at least once.	The black-box testing helps the software developer to derive the sets of input conditions. These input conditions exercise all the functional requirements for a program.
3.	It evaluates all logical decisions and finds whether they are true or false. It evaluates all the loops to check their boundaries and operational bounds. It evaluates all internal data structures to confirm their validity.	It evaluates performance errors or the behavioural errors. It evaluates missing or incorrectly defined functions. It evaluates incorrect data structures and external database errors. It evaluates errors occurred during initialization and termination and interface errors also.



Sr. No.	White box testing	Black box testing
4.	The white box testing begins early in the testing process.	The black box testing is applied in the final stages of testing
5.	Different white-box testing methods are : Basis Path Testing, Control Structure Testing	Different black-box testing methods are : Graph-Based Testing Method, Equivalence Partitioning, Boundary Value Analysis, Orthogonal Array Testing

Syllabus Topic : Verification and Validation

13.9 Verification and Validation

SPPU – May 14

University Question

Q. Explain validation testing. (May 2014, 3 Marks)

- Verification and validation is a set of activities that ensures that all customer's requirements are fulfilled by the software and all the functions are implemented and working correctly. The Verification and Validation (V&V) is basically one of the elements of software testing.
- The validation ensures that the software is developed and it is traceable to all the customer's requirements.
- The Boehm defines verification and validation in following way :
 - o Verification means evaluating whether the developer build product in a right way ?
 - o Validation means the evaluating whether developer built right product ?
- The V&V consists of different activities that are also the part of SQA (Software Quality Assurance).
- The software quality assurance consists of following important activities :

- o FTR (Formal Technical Reviews) : Frequent meetings between developer and customer to ensure effective communication on technical part of the software.
- o Performance monitoring.
- o Quality and configuration audits
- o Algorithm analysis
- o Development testing
- o Feasibility study
- o Database review
- o Simulation
- o Qualification testing and installation testing
- o Documentation review
- o Usability testing

13.9.1 Difference between Verification and Validation

SPPU – May 12, Dec. 13, Dec. 15, May 16

University Question

**Q. Differentiate between verification and validation.
(May 2012, Dec. 2013, 4 Marks, Dec. 2015, May 2016, 4 Marks)**

Sr. No.	Verification	Validation
1.	Verification is a set of activities that ensures that all customer's requirements are fulfilled by the software and all the functions are implemented and working correctly.	The validation ensures that the software is developed and it is traceable to all the customer's requirements.
2.	The Verification is basically one of the elements of software testing.	The Validation is also one of the elements of software testing.
3.	Verification means evaluating whether the developer build product in a right way ?	Validation means evaluating whether developer built right product ?
4.	Verification is static.	Validation is dynamic.

Sr. No.	Verification	Validation
5.	Verification evaluates all the documentation, the planning, the coding, requirements and specifications.	Validation means the evaluation of the complete product itself.
6.	Verification takes place before validation.	Not vice versa.
7.	The inputs for verification are : Technical meetings, various issues, review meeting etc.	The input for validation are : the complete testing of the product itself.
8.	The output of verification : Flawless documents, perfect plans and nearly completed specifications along with requirement.	The output of validation : The perfect final product.

13.10 Test Strategies for Conventional Software

- For the conventional software, there are various test strategies available for use. In first one, the team waits for a product development to be completed and then start conducting tests.
- In another approach, the developer conducts the test on a daily basis as the development proceeds and one part of the program is completed.
- These strategies are chosen by the development team based on the product and convenience of the team. The choice may among two approaches discussed above.
- The team starts with an incremental view of testing and test individual program units. Then moving ahead with integration of the units, and finalizing the tests of overall system.

Different types of tests are mentioned below :

1. Unit Testing
2. Integration Testing
3. Regression Testing
4. Smoke Testing

13.10.1 Unit Testing

SPPU – May 12

University Question

Q. Explain unit testing. (May 2012, 4 Marks)

In unit testing, the main focus is on assessment of smallest unit of the product design like component of the software or module. The unit testing has limited scope and it emphasizes on internal processing logic and data structures. Multiple modules and components can be tested in parallel.

Unit test considerations

- The unit testing is illustrated diagrammatically as follow in Fig. 13.10.1 :

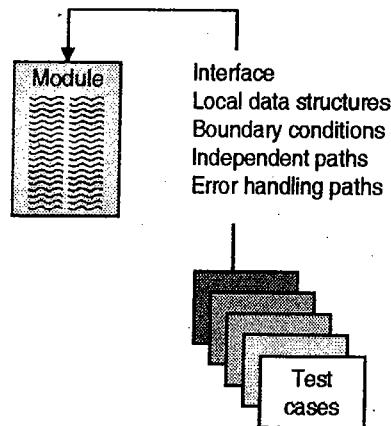


Fig. 13.10.1 : Unit test

- The test strategy is conducted on each of the **module interface** to assess the proper flow of input and output and its correctness as per the requirements.
- Next, the **local data structures** are assessed to ensure its integrity in the execution of the algorithm.
- All the **independent paths** (also called basis paths) are also evaluated through the control structure and it keeps track on the program and makes sure that at least each of the statements in a module should have been executed once.
- **Boundary conditions** are also tested so that the software works properly within the boundaries or within the limits. All the **error handling paths** are tested.

- Thus the **boundary testing** is one of the significant unit testing methodology.

Unit test procedures

- In parallel with coding step, usually unit testing is conducted. Before the start of coding, unit test can be conducted. This method is most commonly used in agile development process.
- The design information gives sufficient help for the developers to establish the test cases and uncover the errors. The developers always couple the set of results with the test cases.

Unit test environment

- The unit test environment is illustrated in Fig. 13.10.2.

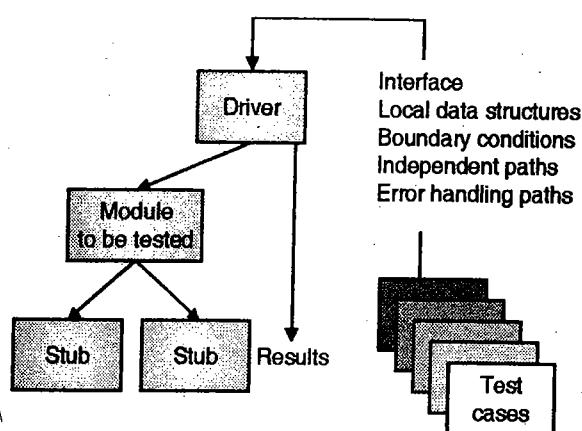


Fig. 13.10.2 : Unit test environment

- In most of the applications, a driver is considered as the "main program". This main program itself accepts all the test case data and pass that data to the component under test and the results are also printed side by side.
- In the above diagram, we observe that the stubs replace the modules of the program and are tested next to driver. The stub is also considered to be the subprogram. These subprograms make an interface with the main program to complete the test.
- Logically both the driver and stubs are the software that are written but not submitted

to the customer and thus are considered as the overhead. It is always recommended to keep these overhead simple to reduce the cost. But overhead can not be made simple in all the cases and hence the unit testing can be postponed to the integration testing.

13.10.2 Integration Testing

- Integration testing is used for constructing the software architecture. It is a systematic approach for conducting tests to uncover interfacing errors. The main objective behind integration testing is to accept all the unit tested components and integrate into a program structure as given by the design.
- It is often observed that there is a tendency to attempt always non-incremental approaches. All the components are integrated in the beginning and the program is tested as a whole.
In this approach a set of errors are encountered and correction seems to be very difficult due to isolation of causes and the complications by program since program is expanded over several modules. After correction of these errors, some new may appear and the process continues. It looks like an infinite loop.

- Small increments of the program are tested in an incremental approach. In incremental integration approach, the error detection and correction is quite simple. In this all the interfaces are tested completely and thus a systematic test strategy may be applied.
- Following are different incremental integration strategies :

(1) Top-down integration

- o Top-down integration testing is one of the incremental testing strategies for construction of the software architecture.

- All the modules are combined by moving top to down direction through the control hierarchy. It begins with the main program or the main control module. The flow is from main module to its subordinate modules in depth-first or breadth-first manner.
- Depth-first integration is illustrated in Fig. 13.10.3, and it integrates all the components on most of the control path of the program.

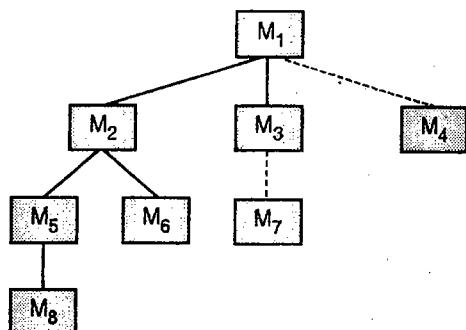


Fig. 13.10.3 : Top-down integration

(2) Bottom-up integration

- In bottom-up integration testing, the testing begins with the construction and components at the lowest level. These components are called as atomic modules.
- Since the components are integrated from the bottom to top, the required processing is always available for components subordinate in all the levels. Here in this approach the need of stub is completely eliminated. The use of stub was actually a disadvantage due to overhead.
- Following steps are required for implementation of bottom-up integration strategy :
- The low-level components are integrated to form clusters that can perform sub function of a specific software.

- A driver is needed to coordinate all the test case inputs and outputs.
- Later all the clusters are tested.
- Then, drivers are removed and all the clusters are integrated once again from bottom to top direction in the program.
- Integration follows the pattern illustrated in Fig. 13.10.4.

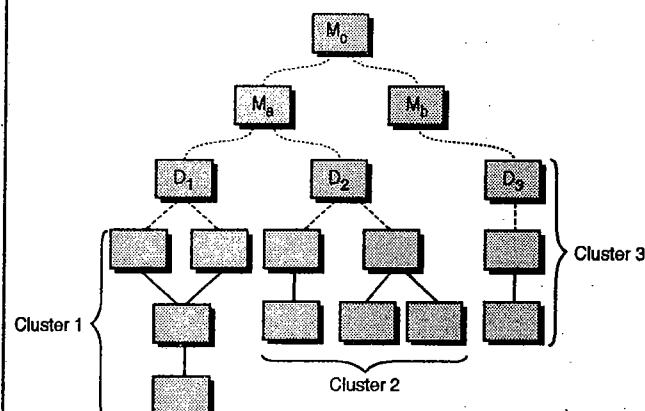


Fig. 13.10.4 : Bottom-up integration

- The components are integrated in such a fashion that they can form clusters 1, 2 and 3. These clusters are tested by using a driver (marked by a dashed block).
- The components in clusters 1 and 2 are subordinate to module M_a . The drivers D_1 and D_2 are then removed and the clusters are connected directly to M_a .
- In the same way, the driver D_3 for the cluster 3 is removed and then cluster 3 is directly connected to module M_b . Here both the modules, M_a and M_b are finally integrated with the component M_c , and so on.

Problems associated with Top down approach of testing

In incremental integration testing approach, a set of errors are encountered and correction seems to be very difficult due to isolation of causes and the



complications by program since program is expanded over several modules. After correction of these errors, some new may appear and the process continues. It looks like an infinite loop.

- Top-down integration testing is one of the incremental testing strategies for construction of the software architecture.
- The main problem with top down approach is that the test conditions are nearly impossible or they are extremely difficult to create.
- Stub modules are complicated.

13.10.3 Regression Testing SPPU – Dec. 16

University Question

Q. Explain regression testing. (Dec. 2016, 3 Marks)

- Regression testing is incremental integration approach and in this every time when a new module is added as a part of integration testing, the software architecture changes.
- Every time new data flow paths are established and new I/O may appear and the new control logic is invoked. These frequent changes cause the problems with the functions that worked earlier without any error.
- In integration test strategy, regression testing may be the re-execution of some of the subset of tests. Thus already tested components may be tested again to ensure the errors are not having bad effects on the overall program structure.

13.10.4 Smoke Testing

SPPU – May 12, May 13, May 14, Dec. 16

University Question

Q. Explain smoke testing.

(May 2012, 4 Marks, May 2013, May 2014,
Dec. 2016, 3 Marks)

- The smoke testing is incremental integration approach and most commonly used when the software products are developed.

- It is designed as a mechanism that increases the pace of time based projects. It allows the software development team to test their project on a regular basis. In the smoke testing approach, following activities are required :

1. All the software components that were developed, now translated into code and are combined to a complete product.
2. The series of tests is created to uncover errors that may cause the product to work incorrectly.
3. The product is integrated with other products and the complete product is smoke tested on daily basis. The incremental integration approach can be either top down or bottom up.

13.10.5 Comments on Integration Testing

- The selection of any integration strategy is based on the software characteristics or the project schedule.
- Generally a combined approach i.e. both the top-down and bottom-up are used. It is called as sandwich testing. The top-down tests can be implemented for upper levels and bottom-up tests are implemented for subordinate levels of the program. Actually the sandwich testing is the best compromise.

13.10.6 Integration Test Documentation

- The complete test plan for integration testing is documented in a test specification.
- This document encompasses a test plan and a test procedure that is actually a work product of the software process, and it becomes the part of the software configuration.

13.10.7 Difference between Regression and Smoke Testing

SPPU – May 15

University Question

Q. Differentiate between regression and smoke testing. (May 2015, 4 Marks)

Sr. No.	Regression Testing	Smoke Testing
1.	Regression testing is incremental integration approach and in this every time when a new module is added as a part of integration testing, the software architecture changes.	The smoke testing is incremental integration approach and most commonly used when the software products are developed.
2.	Every time new data flow paths are established and new I/O may appear and the new control logic is invoked. These frequent changes cause the problems with the functions that worked earlier without any error.	It is designed as a mechanism that increases the pace of time based projects. It allows the software development team to test their project on a regular basis.
3.	In integration test strategy, regression testing may be the re-execution of some of the subset of tests. Thus already tested components may be tested again to ensure the errors are not having bad effects on the overall program structure.	In the smoke testing approach, all the software components that were developed, are now translated into code and are combined to a complete product. The series of tests is created to uncover errors that may cause the product to work incorrectly. The product is integrated with other products and the complete product is smoke tested on daily basis. The incremental integration approach can be either top down or bottom up.
4.	Regression testing increases the cost of the project since it requires more manpower.	Smoke testing doesn't have effect on the budget of the project since it doesn't require extra manpower.
5.	The regression testing is conducted by the testers.	But smoke testing is conducted by the developer only just before the product is released.

13.11 Test strategies for Object-Oriented software

- The objective of any testing strategy is simply uncovering the maximum number of possible errors with optimum amount of efforts applied over a realistic time span.

- This basic idea remains same but the Object-Oriented(OO) software changes both testing strategy and testing tactics.

13.11.1 Unit Testing in the OO Context

- Whenever the object-oriented software is tested, the concept of the unit testing changes as compared to conventional approaches. Encapsulation is preferred in object oriented software and concept of classes is used.
- Each of classes and their instances are called as objects. The package attributes are the data and operations are nothing but the functions in object oriented context. An encapsulated class is the main focus of unit testing in object oriented context.
- Operations in each of the classes are the smallest units that can be unit tested.
- In conventional software, we call it as unit testing and in case of object oriented software, it is referred as class testing.

13.11.2 Integration Testing in the OO Context

- Since the object-oriented software does not have any fixed hierarchical control structure, the top-down and bottom-up integration strategies becomes meaningless.
- Also, the integrating operations one at a time into a class is impossible due to the direct and indirect interactions of the components of that class.
- Following are two different strategies for integration testing in OO context :

- 1. Thread-based testing
- 2. Use-based testing

- The **thread-based testing** combines the set of classes that are required to respond to the input for the system.
- The **use-based testing** start with the construction of the system and it tests those



classes only that uses very few server classes.

13.12 Test strategies for WebApps

- The testing strategy for Web App uses the basic principles for all software testing and also applies the strategies and tactics of object-oriented testing.
- Following steps summarize the approach :
 - o The content model for Web App is reviewed to uncover errors.
 - o The interface model is reviewed to ensure that all use cases can be accommodated.
 - o The design model for Web App is reviewed to uncover navigation errors.
 - o The user interface is tested to uncover errors in presentation or navigation mechanics.
 - o Each functional component is unit tested.
 - o Navigation throughout the architecture is tested.
 - o The Web App is implemented in a variety of different environmental configurations.
 - o Security tests are conducted.
 - o Performance tests are conducted.
 - o The Web App is tested by a controlled and monitored population of end users.

Syllabus Topic : Defect Management

13.13 Defect Management

- Software development teams and software testing teams have numerous choices of defect management tools to help support their software defect efforts. Selecting and

utilizing an effective tool is really only part of an overall defect management system.

- From a high-level view, defect management systems are made up of a combination of some defect management tools or tool and a defect management process.
- These two primary components work together to support each other. Ignore either one, and sub-optimal results can be expected.
- Following is an overview of a typical defect management process and the key features to look for in an effective defect management tool. All of this information will be useful to review.

13.13.1 Defect Management Process

High Level Steps in a typical defect management process

- The typical defect management process includes the following high-level process steps. When implemented inside of a specific organization, each of these high-level steps would have more detailed standard operating procedures along with policies to carry out the details of the process.
- 1. **Identification** - This step involves the discovery of a defect. Hopefully, the person discovering the defect is someone on the testing team. In the real world, it can be anyone including the other individuals on the project team, or on rare occasions even the end-customer.
- 2. **Categorization** - When a defect is reported, it is typically assigned to a designated team member to confirm that the defect is actually a defect as opposed to an enhancement, or other appropriate category as defined by the organization. Once categorized, the defect moves on in the process to the next step which is prioritization.

3. **Prioritization** - Prioritization is typically based on a combination of the severity of impact on the user, relative effort to fix, along with a comparison against other open defects. Depending on the size and structure of the organization, the prioritization is often handled by a formal change control board. The priority should be determined with representation from management, the customer, and the project team.
4. **Assignment** - Once a defect has been prioritized, it is then assigned to a developer or other technician to fix.
5. **Resolution** - The developer fixes (resolves) the defect and follows the organization's process to move the fix to the environment where the defect was originally identified.
6. **Verification** - Depending on the environment where the defect was found and the fix was applied, the software testing team or customer typically verifies that the fix actually resolved the defect.
7. **Closure** - Once a defect has been resolved and verified, the defect is marked as closed.
8. **Management Reporting** - Management reports are provided to appropriate individuals at regular intervals as defined reporting requirements. In addition, on-demand reports are provided on an as-needed basis.

13.13.2 Defect Removal Efficiency

- A quality metric that provides benefits at both the project and process level is defect removal efficiency (DRE). DRE is a measure of the filtering ability of quality assurance and control activities as they are applied throughout all process framework activities.

- DRE is defined in the following manner :

$$\text{DRE} = E/(E + D)$$

Where, E is the number of errors found before delivery of the software to the end-user, and D is the number of defects found after delivery.

- The ideal value for DRE is 1. That is, no defects are found in the software. Realistically, D will be greater than 0, but the value of DRE can still approach 1.
- A quality objective for a software team (or an individual software engineer) is to achieve DRE that approaches 1. That is, errors should be filtered out before they are passed on to the next activity.

Syllabus Topic : Defect Life Cycle

13.14 Defect Life Cycle

- Defect life cycle, also known as Bug Life cycle is the journey of a defect cycle, which a defect goes through during its lifetime. It varies from organization to organization and also from project to project as it is governed by the software testing process and also depends upon the tools used.
- Defect life cycle is a cycle which a defect goes through during its lifetime. It starts when defect is found and ends when a defect is closed, after ensuring it's not reproduced. Defect life cycle is related to the bug found during testing.
- The number of states that a defect goes through varies from project to project. Following lifecycle diagram, covers all possible states :

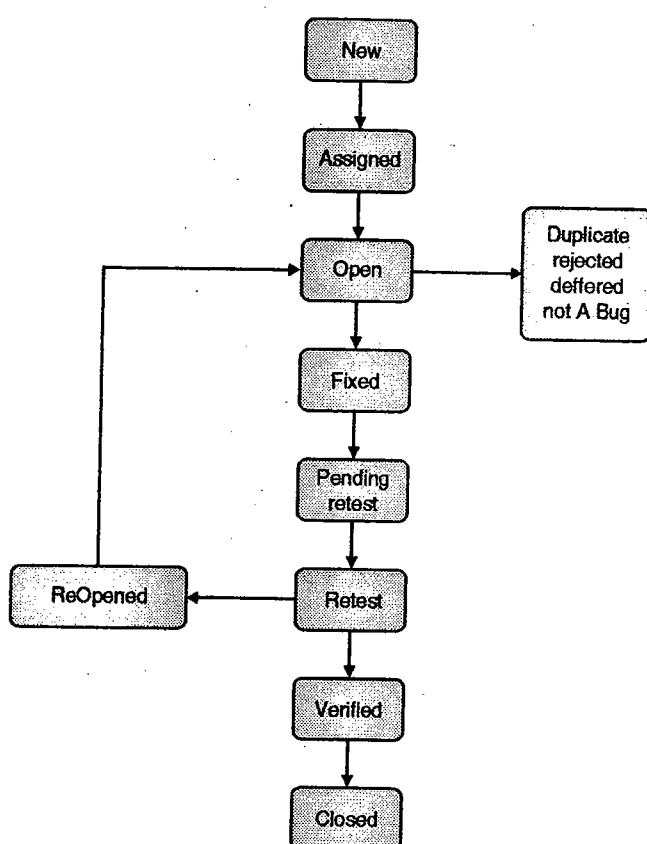


Fig. 13.14.1 : Defect Life Cycle

- **New** : When a new defect is logged and posted for the first time. It is assigned a status NEW.
- **Assigned** : Once the bug is posted by the tester, the lead of the tester approves the bug and assigns the bug to developer team.
- **Open** : The developer starts analyzing and works on the defect fix.
- **Fixed** : When developer makes necessary code change and verifies the change, he or she can make bug status as "Fixed."
- **Pending retest** : Once the defect is fixed the developer gives particular code for retesting the code to the tester. Since the testing remains pending from the testers end, the status assigned is "pending request."
- **Retest** : Tester does the retesting of the code at this stage to check whether the

defect is fixed by the developer or not and change the status to "Re-test."

- **Verified** : The tester re-tests the bug after it got fixed by the developer. If there is no bug detected in the software, then the bug is fixed and the status assigned is "verified."

- **Reopen** : If the bug persists even after the developer has fixed the bug, the tester changes the status to "reopened". Once again the bug goes through the life cycle.

- **Closed** : If the bug is no longer exists then tester assigns the status "Closed."

- **Duplicate** : If the defect is repeated twice or the defect corresponds the same concept of the bug, the status is changed to "duplicate."

- **Rejected** : If the developer feels the defect is not a genuine defect then it changes the defect to "rejected."

- **Deferred** : If the present bug is not of a prime priority and if it is expected to get fixed in the next release, then status "Deferred" is assigned to such bugs.

- **Not a bug** : If it does not affect the functionality of the application then the status assigned to a bug is "Not a bug".

Syllabus Topic : Bug Reporting

13.15 Bug Reporting (The art of debugging)

- Software testing process is a systematic and planned activity and it is an integral part of the development process.
- The test cases are conducted, and a strategy can be defined, and the results can be evaluated as per the requirements.
- Debugging occurs when a test case is successfully conducted. It means that the test case uncovers various errors, and the

debugging process can be started. The debugging process attempts to eliminate all the errors that were uncovered in the testing process.

- Even if debugging is an orderly process, it is considered to be an art.

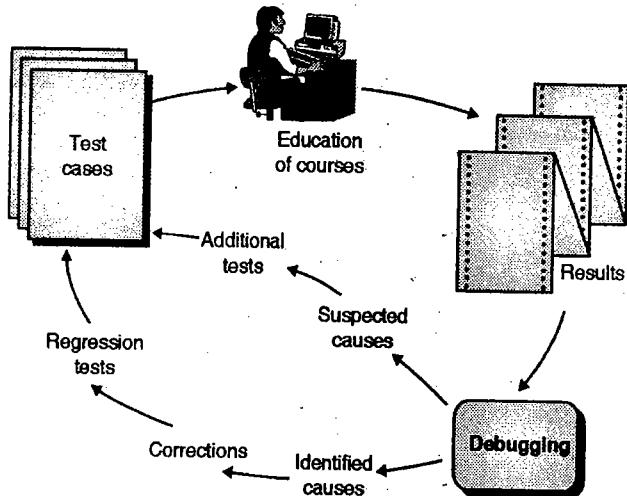


Fig. 13.15.1 : The debugging process

13.15.1 The Debugging Process

SPPU – May 13, Dec. 13

University Question

Q Explain the debugging process.

(May 2013, 8 Marks, Dec. 2013, 6 Marks)

- Debugging is not a testing process but it is the consequence of testing. In the Fig. 13.15.1, the debugging process starts with conduction of a test case.
- The results are evaluated and a difference between expected and actual performance is encountered.
- The debugging process has one of two outcomes as follows :
 - (1) The reason for causing error are found and corrected, or
 - (2) The reason for causing the error will not be found.
- In the case 2, the person performing debugging will be suspicious about a cause. Conducting the test cases will help to validate that suspicion, and the person can work to correct the errors in iterative fashion.

- It is observed that the debugging is a difficult process. Sometimes human psychology is required in addition to software technology. Following are some characteristics of bugs that will provide some clues to clear the doubts about debugging process :

- o The cause or the symptom can be geographically remote.
- o The symptom may be unavailable temporarily, while some other errors are corrected.
- o The symptom may also be caused by non-errors (e.g., round-off inaccuracies).
- o The symptom may be caused by some human error and that can not be easily traced.
- o The symptom may be due to timing problems rather than processing problems.
- o It is very difficult to reproduce input conditions accurately. For example real-time applications where order of the input values is indeterminate.
- o The symptom can also be intermittent i.e. not continuous. This scenario is very common in embedded systems.
- o The symptom can also be caused by number of tasks running on different processors in a distributed system scenario.

- In the debugging process, numerous error are encountered that may be mild annoying or catastrophic.

- The examples of mild annoying errors are :

- o Incorrect output format

- The examples of catastrophic errors are :

- o System fails
- o Physical damage
- o Economical losses

- In debugging process, finding the cause of the errors is more important than finding the errors itself. The software developer sometimes eliminates an error and at the same time, two more errors are introduced.

13.15.2 Psychological Considerations

- Debugging is considered to be the natural process and some people are good in debugging and some are not good in debugging. Thus there is large variance in the debugging process. It is evident from the various reports that the developers with the same education and same experiences have different debugging capabilities.
- Based on human abilities of debugging, Schneiderman gave following statements :
 - o Debugging is the most frustrating phase of development. It consists of problem solving and brain teasers, along with some annoying recognition that you made a mistake.
 - o The anxiety and unwillingness to admit the possibility of errors, usually increases the difficulty of the task. But finally all the bugs are corrected.
 - o It is quite difficult to "learn" debugging process as there are number of approaches exists for problem solving.

13.15.3 Debugging Approaches

- Regardless of the approach that is chosen, debugging has one simple objective i.e. to detect and correct the cause of error. The objective can be achieved by different evaluation techniques.
- Bradley has described the debugging approach in the following way :
 - o Debugging is an application of scientific method.
 - o In debugging, the basic idea is to locate the problem's source by partitioning it.

Consider the following non-software example

- o Suppose a lamp in a house is not working, it means that lamp is faulty. If everything in the house is not working, it means there is problem in the main circuit breaker. If I see in my neighbourhood, everything is blackout, then I can find there is problem in the main connection in my area and so on.
 - o In this scenario I have to start correcting the problems from top and reach to the specific location.
 - o This hypothesis may be useful in conducting the test.
- Following are three important categories for debugging approaches :

- (1) Brute force
 - (2) Backtracking
 - (3) Cause elimination

(1) Brute force

- o The Brute force is the most common and least efficient method for finding the cause of a software error.
- o This category of debugging i.e. Brute force debugging methods is applicable generally when all the other methods are failed. In this method, computer finds the errors itself since memory dumps are used and thus run-time traces are invoked. In this, the program is loaded with the WRITE statements.
- o In the complicated and confused pool of information, the developer is able to reach to the clue to find the cause of errors.
- o The mass information produced will lead to success finally. Sometimes it may lead to waste of time and effort. Therefore first, the thought should be expended.

(2) Backtracking

- Backtracking category of debugging is more common approach and is used successfully in especially small programs. In this approach, the hint or the point where errors are seen is selected as a starting point and the source code is traced backward. The backtracking is a manual process and continues till the cause of the error is found.
- In this category, the task is increased as compared to previous approaches. The number of source code lines are increased and sometimes the large number of backward paths are unmanageable.

(3) Cause elimination

- The third category of debugging use the concept of binary partitioning. The cause elimination approach is achieved by induction or deduction.
- Data that causes errors or somehow that data is related to the error occurrence, are organized properly to find the actual cause of potential errors. In this category of debugging, a "cause hypothesis" is used. In this, any previously mentioned data is taken to prove or disprove the hypothesis. In this, the list of causes is prepared and finally the tests are conducted to eliminate these errors.
- If tests prove the cause hypothesis then the data is refined to find the errors.
- These debugging approaches are supported by various debugging tools available to make it more effective.
- Various debugging compilers are applied and various dynamic debugging aids called as **tracers** are used.

- Also the automatic test case generators, cross-reference maps and memory dumps are also used to make this category of debugging more effective. But one should be remembered that these tools can not be the substitute of debugging process, they are only the supporting tools.
- As soon as the errors are found, they are eliminated, but during the correction of these errors, some new errors can be introduced. So instead of some good thing, more damage can be caused in this process.
- **Van Vleck** stated following three simple questions that must be answered before correction starts :
 1. Check whether the cause of error is reproduced in some another part of the program?
(A) In most of the cases, the program errors are caused by erring logic and this erroneous logic may also be reproduced in another part of the program. The explicit consideration should be taken for all the logical pattern. This consideration may discover the errors.
2. What will be the next error, if I fix one ?
(B) So before any correction is made, the tester should assess the source code to find the use of any logic in another part of the same program. A utmost care is taken in correction so that interrelated logic or coupling should not cause any new errors.
3. What precaution should be taken to prevent the error at the first place only?
(C) By using proper SQA (Software Quality Assurance) approach, this question can be answered. If the process and the product, both are corrected simultaneously then all the errors may be eliminated from the current programs as well as future programs.

Syllabus Topic : GUI Testing

13.16 GUI Testing

During the initial testing processes, the design model is reviewed to assess that all the customer's requirements are working and the quality attributes are achieved. In addition to all these testing, the final assessment is the usability assessment.

13.16.1 Interface Testing Strategy

The strategy for interface testing is to uncover errors in interface mechanism and also in the implementation of navigation and functionality and in the contents. In order to accomplish the strategy, following objectives must be achieved :

- Interface features must ensure the design rules, aesthetics (beautiful looks) and correct contents on the website. Features means : font size, font type, colours, images, frames, tables and all the concerned elements.
- Individual interface mechanism is tested to apply unit testing on each individual elements.
- Then the complete interface is also tested to uncover errors in the semantics of the interface by conducting a series of usability tests.
- The interface is also tested in different environments and different browsers to ensure the compatibility of the application.

13.16.2 Testing Interface Strategy Mechanisms

Whenever the end-users interact with the web-based applications, they interact through various interface mechanisms. In the following section, we elaborate each of the interface mechanism :

- **Links** : All the navigation links are tested to ensure proper flow of web-based

application and the web developer execute each of link individually to uncover errors.

- **Forms** : The tests are conducted to check each of the forms in the WebApp and all the components present on each form. It is ensured that no data should be lost in the transaction between client and server.
- **CGI scripts** : When the data is passed to the CGI scripts, black-box tests are conducted to ensure data integrity and data validation.
- **Cookies** : The testing is done on both the sides i.e. client side and server side to make sure that the cookies are properly constructed and they store proper data.
- **Dynamic HTML** : Each of the dynamic HTML pages is tested to uncover errors in display of dynamic data.
- **Pop-up windows** : The test cases are conducted to check pop-up windows are in proper size and position.
- **Client-side scripting** : The scripts at the client-side are executed to uncover the errors. The black-box testing is generally used in the script processing.

13.16.3 Usability Tests

- The usability tests are conducted to ensure that end-users interact with the WebApp effectively. Usability tests are designed by the web developers but they are tested by the end-users only.
- In usability test, following steps are applied:
 - o Define the set of test categories and identify their goals.
 - o Design tests to evaluate each category goal.
 - o List participants who will conduct these tests.

- Observe the participant's interaction with WebApp.
- Develop some mechanism to assess the usability of WebApp.
- Usability testing is conducted at different levels of abstraction as follows :
 - Usability of single interface mechanism can be evaluated
 - Usability of a complete web page can be assessed.
 - And finally usability of complete WebApp can be evaluated.
- While conducting usability testing, very first step is to identify the categories and their testing objectives. Following are the test categories and their respective objective that explain this approach :
 - **Interactivity** : The interface mechanism should be easy to understand and use.
 - **Time sensitivity** : It should respond in quick time.
 - **Layout** : The layout should be in such a way that end-users find it easy to use.
 - **Aesthetics** : The design and contents should look beautiful and user should feel comfortable.
 - **Display** : The resolution should be in such a way that it uses proper screen size.
 - **Personalization** : It should be adjustable to customer's specific needs.
 - **Accessibility** : It should be accessible physically disabled users.
 - **Readability** : All the textual contents should be in readable font and colour. The images must be visible and clear.
- Following Fig. 13.16.1 exhibits the set of test grades that can be chosen by the users.

These grades can be applied to all the levels of abstraction for usability testing.

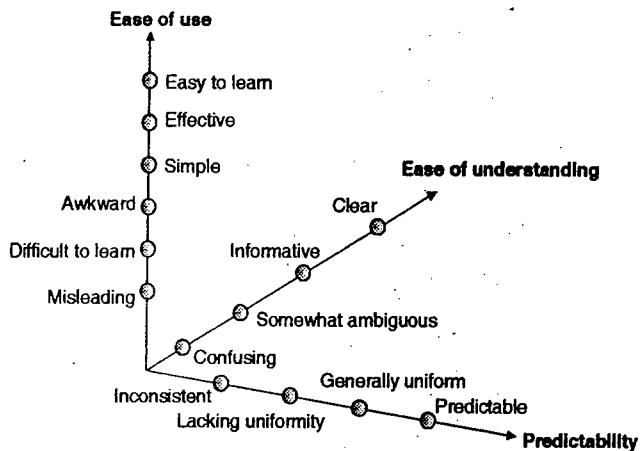


Fig. 13.16.1 : Qualitative assessment of usability

13.16.4 Compatibility Tests

- **Compatibility** testing means working of WebApp on various host configurations. The interface is tested in different environments and different browsers to ensure the compatibility of the application.
- Since WebApp is used on the network that encounters different computers with different configurations, different display devices, different browsers, different OS and variable communication and connection speeds, it should be designed in such a fashion that it works on these heterogeneous environments.

13.16.5 Test Plan

- A test plan documents is used to verify and ensure that software or a system satisfies its design specifications and all the requirements and the design strategies. Usually a test plan is written with some important inputs from test engineers or the developers.
- Based on the type of the product, and the responsibility of the organization, various test plans are selected and applied. A test plan generally includes a strategy for the following :

- **Design verification** - It is performed in the development or approval stages typically for small units.
- **Manufacturing test** - It is performed during assembly of the product or integration.
- **Acceptance test** - It is performed after delivery or installation.
- **Repair test** - It is performed over the service life of the product as per the contract.
- **Regression test** - It is performed on working product, to verify its functionality.
- Any complex system has a high level test plan to address the requirements and supporting test plans. The test plan document formats is varied according to the product and customer organization. Following are three major elements of a test plan :
 - o Test Coverage
 - o Test Methods, and
 - o Test Responsibilities.
- All these elements of test plan are also used in a formal test strategy

13.16.6 Positive Testing

- In positive testing, the system is validated against the valid input data. In this testing approach, the tester always tests only valid set of values. He checks the application is behaving as per expectation with the expected inputs.
- The main objective of this testing is that the application should work as per the requirements with valid set of inputs. This testing is conducted with positive frame of mind and to execute the positive scenario only.

- Positive Testing always prove that a given software satisfies the requirements and specifications.

13.16.7 Negative Testing

- In negative testing, the system is validated against the invalid input data. In this testing approach, the tester always tests only invalid set of values. He checks the application is behaving as per expectation with the invalid set of values.
- The main objective of this testing is that the application should work as per the requirements with invalid set of inputs. This testing is conducted with negative frame of mind and to execute the negative scenario only. Negative testing always proves that given software satisfies the requirements and specifications with different variety of incorrect validation data set.
- The main objective behind this type of testing is to check the stability of the system against different variety of incorrect validation data set.

Syllabus Topic : Test Management and Automation

13.17 Test Management and Automation

- In order to use test management and automation tools, we must remember following important points :
 - o Use automation when it advances the mission of testing.
 - o Select your automation strategy based on your context.
 - o A test tool is not a strategy.
 - o Test automation is a significant investment.
 - o Test automation is design and programming.

- Test management tools are used to store information on how testing is to be done, plan testing activities and report the status of quality assurance activities. The tools have different approaches to testing and thus have different sets of features.
 - Generally they are used to maintain and plan manual testing, run or gather execution data from automated tests, manage multiple environments and to enter information about found defects.
 - Test management tools offer the prospect of streamlining the testing process and allow quick access to data analysis, collaborative tools and easy communication across multiple project teams.
 - Many test management tools incorporate requirements management capabilities to streamline test case design from the requirements. Tracking of defects and project tasks are done within one application to further simplify the testing.
 - Test automation management tools are specific tools that provide a collaborative environment that is intended to make test automation efficient, traceable and clear for stakeholders.
 - Test automation is becoming a cross-discipline (i.e. a mix of both testing and development practices.)
- o Making transparent, meaningful and traceable reporting for all project stakeholders.
 - o Easing test debugging through test results analysis workflow.
 - o Providing valuable metrics and key performance indicators – both technical and business-wise (trend analysis, benchmarking, gap analysis, root cause analysis and risk point analysis).
 - o Grid benchmarking and comparison of test execution days reduces analysis and review effort.
 - o Clean traceability with other testing artifacts (test cases, data, issues, etc.).
 - o Organizing historical data.
 - o Post-project analysis and automation performance assessment. (Progress of test coverage shows the group performance.)

13.17.2 Test Driven Development (TDD)

- Test-driven development (TDD) utilizes test automation as the primary driver to rapid and high-quality software production. Concepts of green line and thoughtful design are supported with tests before actual coding, assuming there are special tools to track and analyze within TDD process.
- In the following section we exhibit test driven development (TDD) with block diagram and also explain different phases of TDD.
- It is one of the agile practices in agile development.
- This software development process relies on repetition of short development cycles.
- Changes in software are quick and easy.
- Short development cycle includes writing automated test case which defines desired improvement or new function. Minimum

13.17.1 Motivation

- Test automation systems usually lack reporting, analysis and meaningful information about project status. Test management systems target manual effort and do not give all the required information.
- Test automation management systems leverage automation effort towards efficient and continuous processes of delivering test execution and new working tests by :

coding is done to pass the test. Afterwards code are redesigned as per the standard.

- It is related to extreme programming.
- Test driven development is shown in Fig. 13.17.1.

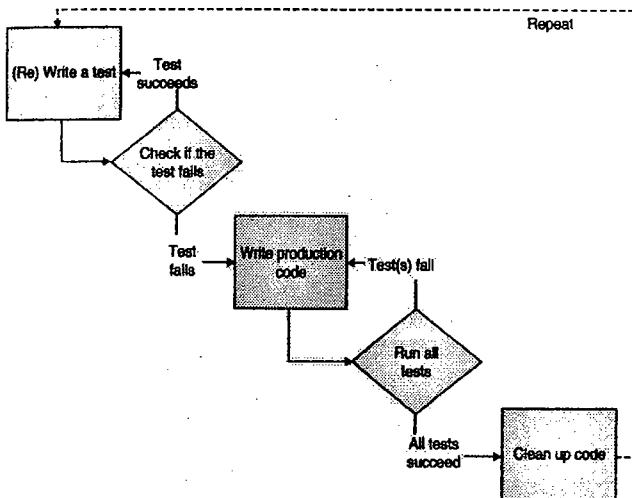


Fig. 13.17.1: Test driven development

- Different phases of test driven development are as follows.

(a) Create test

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

(b) Run test and see it fails

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

(c) Writing some code

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

(d) Run test

We run the test on created code.

(e) Refactor code

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

(f) Repeat

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

University In Sem Exam

Questions and Answers

Questions and Answers for In-Semester Examination

Chapter 1 : Software Engineering Fundamentals

**Q. 1.1 Discuss the Significance of Computer Software.
(3 Marks)**

Ans. :

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

Q. 1.2 List and explain the characteristics that describe the nature of software. (4 Marks)

Ans. :

The nature of software has great impact on software engineering systems. The general nature of software may describe the important characteristic:

- **Reliability :** If we consider the use of software in air traffic control system and space shuttle, then there should not be any chances of failure of software.
- In these examples, if reliability is not taken into consideration, then the human life is at risk.
- In addition to this, there are four other important characteristics that describe the nature of software :

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

Absence of Fundamental Theory

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

Ease of Change

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



Rapid Evolution of Technologies

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

Low Manufacturing Cost

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

Q. 1.3 Explain the layered approach to software engineering. (4 Marks)

Ans. :

- Software engineering is considered as a layered technology. These layers includes :

- | | |
|------------------|------------|
| 1. Quality focus | 2. Process |
| 3. Methods | 4. Tools |

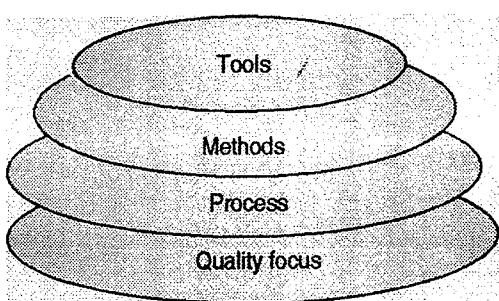


Fig. 1-Q.1.3 : Software engineering layers

Quality Focus

Quality is nothing but 'degree of goodness'. Software quality cannot be measured directly. Good quality software has following characteristics :

1. **Correctness** is degree to which software performs its required function.
2. **Maintainability** is an ease with which software is maintained. Maintenance of software implies change in software. If software is easy to maintain, then the quality of that software is supposed to be good.
3. **Integrity** is a security provided so that unauthorized user can not access data or information. e.g. password authentication.
4. **Usability** is the efforts required to use or operate the software.

Process

Software process defines a framework that includes different activities and tasks. In short, process defines following 'what' activities and tasks for software development :

1. What activities are to be carried out ?
2. What actions will be taken ?
3. What tasks are to be carried out in a given action ?

Methods

Method provides technical way to implement the software i.e. 'how to' implement. Methods consist of collection of tasks that include :

1. **Communication** : Between customer and developer.
2. **Requirement analysis** : To state requirements in detail.
3. **Analysis and design modelling** : To build a prototyping model of software to exhibit the requirements clearly.
4. **Program construction** : Implementation of requirements using conventional programming languages or automated tools.
5. **Testing and support** : Test for errors and expected results.

Tools

Software tool is an automated support for software development.

For example

1. Microsoft front page or Microsoft Publisher can be used as web designing tool.
2. Rational Rose can be used as object oriented analysis and design tool.

Q. 1.4 Define Software Engineering.. (3 Marks)

Ans. : The term software engineering is defined as :

- “By using the principles of sound engineering and its establishment, software is developed that should be economical and should work efficiently on real machines.”
- Software engineering is a systematic and disciplined approach towards developments of software. The approach must be systematic for operation of the software and the maintenance of it too.
- Software engineering is considered as a layered technology.

These layers includes :

1. Quality focus	2. Process
3. Methods	4. Tools

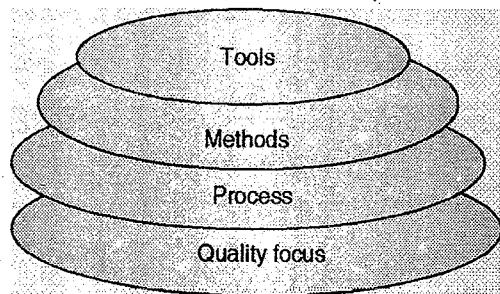


Fig. 1-Q.1.4 : Software engineering layers

Q. 1.5 What are the software characteristics? (3 Marks)

Ans. :

- Software is having some characteristics, those are totally different from hardware characteristics or the industrial manufacturing
- Software is developed or engineered and it is not manufactured like other hardware products.
- There exist some similarities between development of software and manufacturing of hardware
- In both the cases quality is achieved by good design but manufacturing phase of hardware can introduce quality problems that are absent in software.
- Both activities depend on people but relationship between people applied and work done is different in both the cases.
- Both the cases require the ‘construction of product’, but approaches are different.

- Software does not wear out.
- Note that, wear out means process of losing the material.
- Hardware components are affected by dust, temperature and other environmental maladies. Stated simply, hardware can wear out. However software does not influenced by such environmental means.
- When hardware component wear out, it can be replaced by spare part. But there are no spare parts for software. Any software error indicates error in design or coding of that software.
- Hence software maintenance involves more complexity than hardware maintenance.
- Mostly software is custom built rather than assembled from existing components.
- Computer hardware can be assembled from existing components as per our requirements. But that is not the case for software. Software is developed according to customer's need.

Q. 1.6 Explain the term : Software crisis. (3 Marks)

OR What are the problems associated with development process ? (3 Marks)

Ans. :

- Most of software engineers refer the problems associated with software development as “Software crisis”.
- The causes of software crisis are linked to all the problems and complexities associated with development process. Following are some most encountered problems associated with development process :

1. Running out of time i.e. deadline crossed
2. Over budget
3. Software inefficient
4. Low quality
5. Not up to expectations of customers
6. No enough development team

Q. 1.7 What is a software engineering process ?

(3 Marks)

Ans. :

- A software process can be illustrated by the following Fig. 1-Q.1.7 In the Fig. 1-Q.1.7, a common process framework is exhibited. This framework is established by dividing overall framework activities into small number of framework activities.

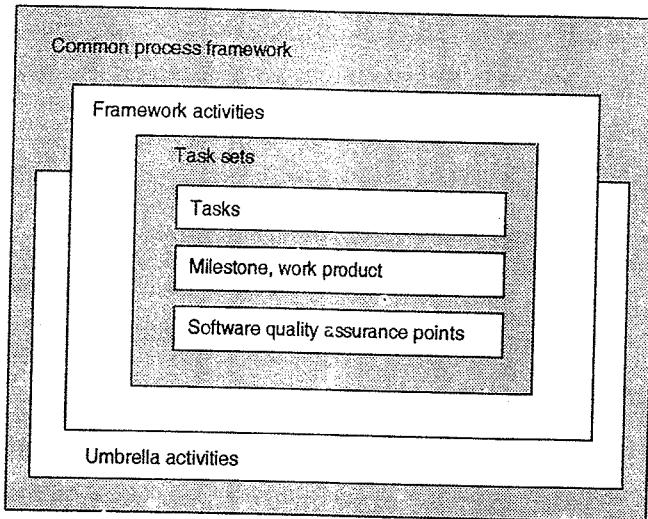


Fig. 1-Q.1.7 : A software process

- And these small activities are applicable to the entire project irrespective of its size and complexity. A framework is a collection of task sets and these task sets consists of :
 - o Collection of small work tasks
 - o Project milestones, deliverable i.e. actual work product and
 - o Software quality assurance points
- There are various activities called umbrella activities are also there and these activities are associated throughout the development process.

Q. 1.8 State a software myth each which customers, developers and project manager believe. What is the reality in each case? **(4 Marks Each Myth)**

Ans. :

- Dictionary meaning of myth is '**fable stories**'. It is a fiction, imagination or it is a thing or story whose existence is not verifiable. In relation with computer software myth is nothing but the misinformation, misunderstandings, or confusions propagated in software development field.
- In software development the myths can be considered as three level myths.

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

1. Management Level Myths (or Manager Level Myths)

Myth

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

Reality

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

Myth

We can buy software tools and use them.

Reality

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

Myth

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

Reality

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

Myth

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

Reality

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

Myth

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

Reality

The rule book exists. But the questions arise :

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

Myth:

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

Reality

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

2. Customer Level Myths**Myth**

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

Reality

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

Myth

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

Reality

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

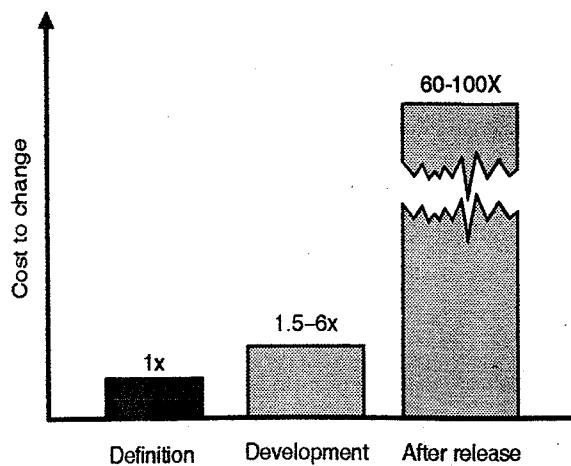


Fig. 1-Q.1.8 : The impact of change

3. Practitioner Level Myths (or Developer Level Myths)**Myth**

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

**Reality**

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

Myth

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

Reality

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

Myth

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

Reality

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

Myth

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

Reality

- Software engineering the process that creates the quality and it is not the process of only creating the documents.
- The good quality of the product will reduce the extra work involved in rework.

- And finally reducing the overhead of rework will lead to quicker development to complete the desired work on scheduled time.

Q. 1.9 Explain the importance of Software Engineering.

(4 Marks)

Ans. :

- Software engineering is the study and application of engineering to the design, development and maintenance of software that needed in all the aspects of our daily lives.
- In every field of work, specific software is needed. Since software is developed and embedded in machines so that it could meet the requirement of the user. This is possible because of software engineering.
- Using software engineering concepts reduces the complexity in developing the software application.
 1. It also reduces the software cost.
 2. Development time is reduced considerably

Chapter 2 : Process Models

Q. 2.1 State the generic process framework activities.

(4 Marks)

Ans. :

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

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

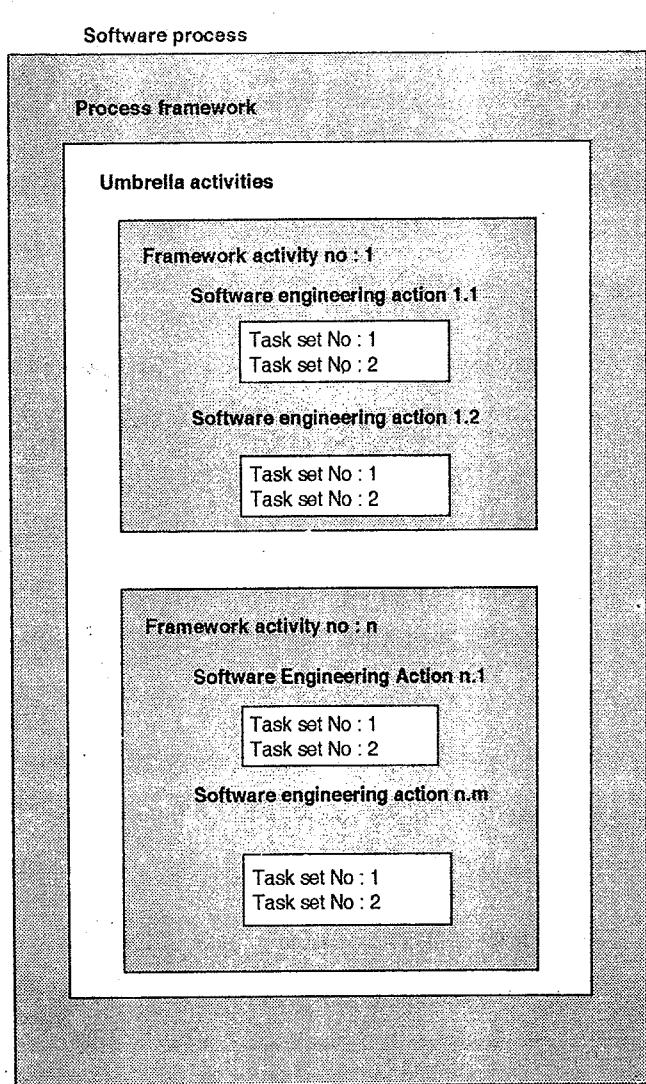


Fig. 1-Q.2.1 : A software process framework

Communication

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

Planning

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

Modeling

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

Construction

It includes coding and testing steps :

- i) **Coding** : Design details are implemented using appropriate programming language.
- ii) **Testing** : Testing is carried out for the following reasons :
 - To check whether the flow of coding is correct or not.
 - To check out the errors of program e.g. in C program just by pressing F7 key we check step by step execution of program or by using "Add-Watch" we add the variables and watch the values of variables.
 - To check whether program is giving expected output as per input specifications.

Deployment

- It includes software delivery, support and feedback from customer.
- If customer suggest some corrections, or demands additional capabilities, then changes are required for such corrections or enhancement.
- These five generic framework activities can be used for :
 - o Development of small programs
 - o Creation of large programs
 - o Development of large system based programs
- In addition to these five generic framework activities, various umbrella activities are also associated throughout the development process.

Q. 2.2 Explain the waterfall model with work products of each activity. (4 Marks)

Ans. :

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

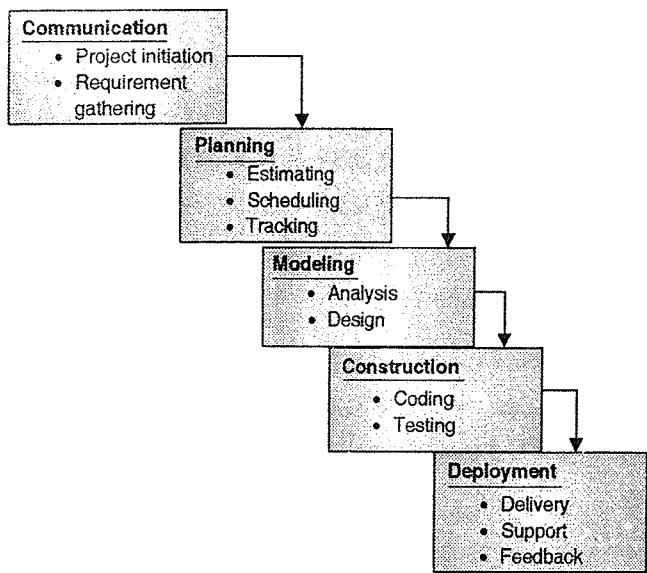


Fig. 1-Q.2.2 : The Waterfall model

- An alternative design for 'Linear Sequential Model' is as shown in Fig. 2-Q.2.2.

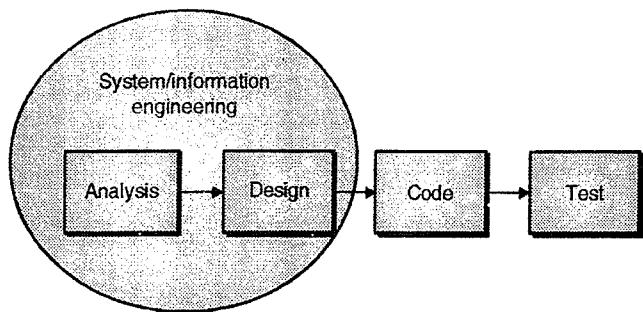


Fig. 2-Q.2.2 : The linear sequential model

1. Communication

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

2. Planning

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

3. Modeling

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

4. Construction

It includes coding and testing steps :

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

5. Deployment

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

Q. 2.3 Explain the waterfall model with V model.(3 Marks)

Ans. :

- In software engineering development process, the V-model represents a development process that may be considered as an extension of the waterfall model.
- In more general V-model, instead of moving down in a linear way, the process steps are bent upwards after the coding phase as shown in Fig. 1-Q.2.3.

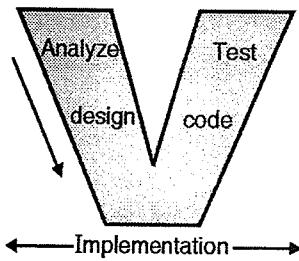


Fig. 1-Q.2.3 : V-model

- The V-model demonstrates the relationships between each phase of the development life cycle.

Q. 2.4 Explain the incremental model ? (4 Marks)

Ans. :

- The incremental model combines the elements of waterfall model applied in an interactive fashion. The first increment is generally a core product.



- Each increment produces the product, which is submitted to the customer. The customer suggests some modifications.

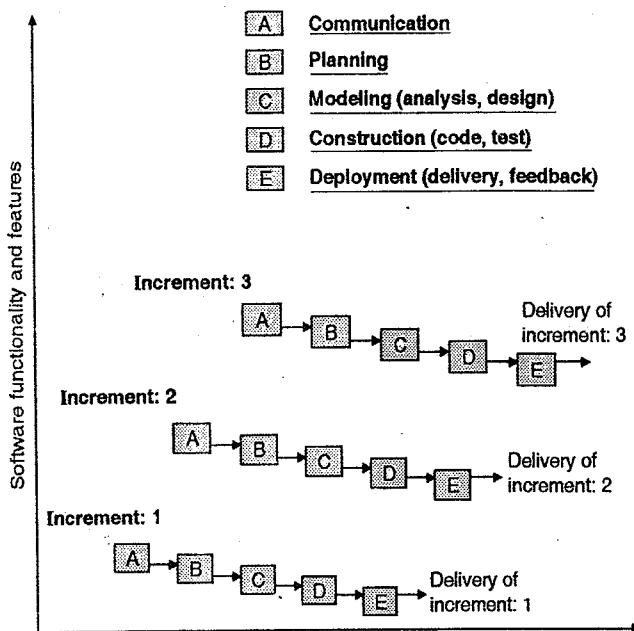


Fig. 1-Q.2.4 : Project calendar time

- The next increment implements customer's suggestions and some additional requirements in previous increment. The process is repeated until the product is finished.
- Consider the development of word processing software (like MS Word or Word Star) using incremental model.

Increment – 1

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

Increment – 2

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

Increment – 3

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

Increment – 4

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

Q. 2.5 Explain the RAD model. (4 Marks)

Ans. :

- RAD (Rapid Action Development) model is high-speed adaptation of waterfall model. Using RAD model, software

product can be developed within a very short period of time i.e. almost within 60 to 90 days.

The initial activity starts with communication between customer and developer. Depending upon initial requirements the project planning is done. Now the requirements are divided into different groups and each requirement group is assigned to different teams.

- Like other approaches, in RAD approach also all the generic framework activities are carried out. These generic framework activities are already discussed earlier.
- When all teams are ready with their final products, the product of each team is integrated i.e. combined to form a product as a whole.

In RAD model each team carries out the following steps :

- Communication :** It understands the business problem and information for software.
- Planning :** Since various teams work together on different modules simultaneously, planning is important part of development process.
- Modeling :** It includes :
 - Business Modeling :** It includes information flow among different functions in the project e.g., what information will be produced by each function, which functions handles that information etc.
 - Data Modeling :** It includes different data objects used in software and relationship among different objects.
 - Process Modeling :** During process modeling, process descriptions are created. e.g. add, modify, delete etc.
- Construction :** It includes :
 - Component reuse :** Reusable components means the software components i.e. modules or procedures that are developed for specific application can be reused in development of other application.
 - Automatic code generation :** The software producing the codes after providing proper inputs or selecting ready-made options, e.g. Microsoft



FrontPage used in Web development, Rational Rose used for Object Oriented analysis and designing.

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

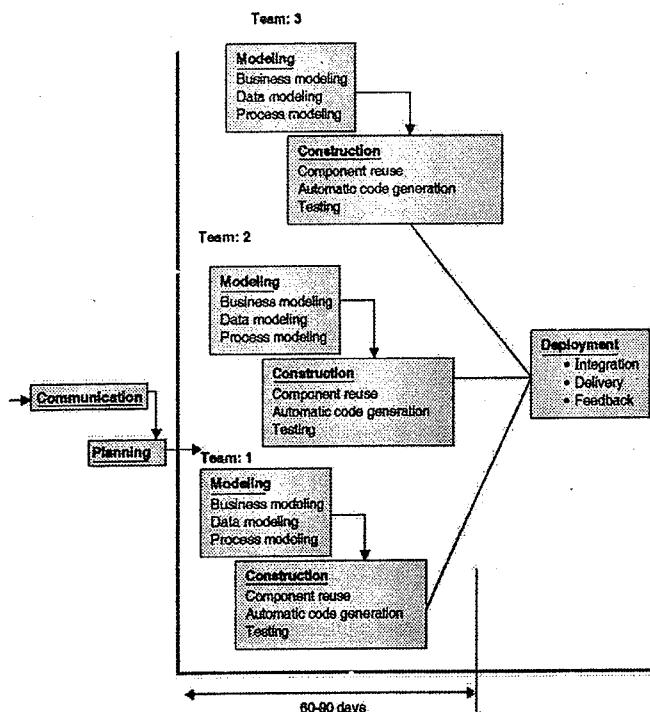


Fig. 1-Q.2.5 : The RAD Model

Q. 2.6 Explain the prototyping model. (4 Marks)

Ans. :

- The **prototyping paradigm** offers best approach for human machine interaction. Ideally prototype serves as a mechanism for identifying software requirements.
- The prototyping approach is often called as **throw-away prototyping**. By using this approach, there is a rough demonstration of the customer's requirements i.e. the prototype used in demonstration is just a throwaway prototype.
- If working prototype is built, developer can use software tools if required (e.g. report generators)

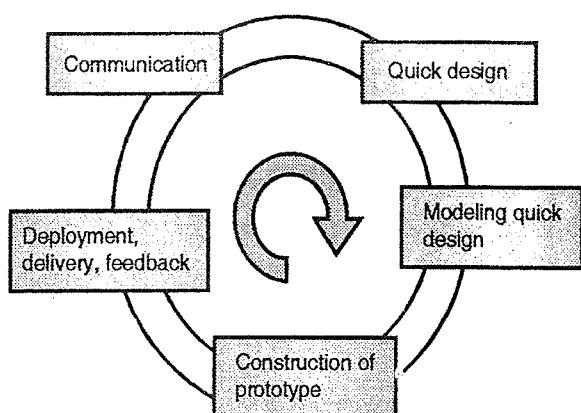


Fig. 1-Q.2.6 : The prototyping model

- This produces working program quickly. Thus prototype can be served as 'first system'.

- Different phases of prototyping model are :

1. Communication

- The software prototyping paradigm starts with the communication between the developer and the customers.
- The software engineer and the customer meet regularly to define the overall objectives of the desired software and to identify its requirements.

2. Quick design

- Quick design focuses on those aspects of software that are visible to the customer.
- It includes clear input, output formats and human machine interfaces.

3. Modeling quick design

The model of software is now built that gives clear idea of the software to be developed. This enables the developer to better understand the exact requirements.

4. Construction of prototype

The concept of modeling the quick design leads to the development of prototype. This construction of prototype is deployed by the customer and it is evaluated by the customer itself.

5. Deployment, Delivery, Feedback

- As picture of software to be built is very clear, customer can give his suggestions as a feedback.
- If result is satisfactory, the model is implemented otherwise process is repeated to satisfy customers all requirements.

Q. 2.7 State the activities of spiral model. (4 Marks)

Ans. :

- The **Spiral model** is combination of well known waterfall model and iterative prototyping. It yields rapid development of more complete version of software.
- Using spiral model software is developed as series of evolutionary releases. During the initial releases, it may be just paperwork or prototype. But during later releases the version goes towards more completed stage.

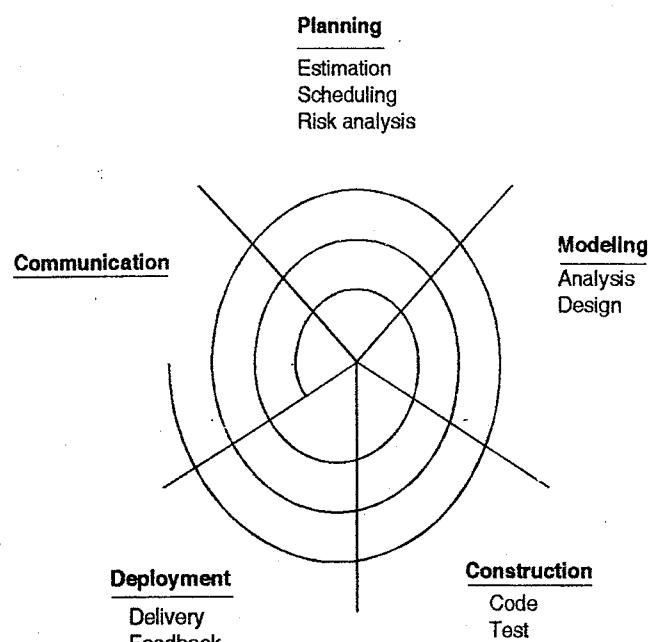


Fig. 1-Q.2.7 : The Spiral Model

- The spiral model can be adopted to apply throughout the entire lifecycle of the application from concept development to maintenance. The spiral model is divided into set of framework activities defined by software engineer team. Each framework activity represents one segment of spiral as shown in Fig. 1-Q.2.7.
- The initial activity is shown from centre of circle and developed in clockwise direction. Each spiral of the model includes following steps :

1. Communication

The software development process starts with communication between customer and developer.

2. Planning

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

3. Modelling

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

4. Construction

It includes coding and testing steps :

- (i) **Coding** : Design details are implemented using appropriate programming language.
- (ii) **Testing** : Testing is carried out.

5. Deployment

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

Q. 2.8 Explain concurrent development model. (4 Marks)

Ans. :

- The **concurrent development model** is also called as **concurrent model**. This model is more appropriate for system engineering projects where different engineering teams are involved. In system engineering stage, the project requirements are considered at system level.
- Diagrammatically it can be represented as a series of framework activities, task, actions and their associated states.
- Fig. 1-Q.2.8 exhibits one element of the concurrent process model :

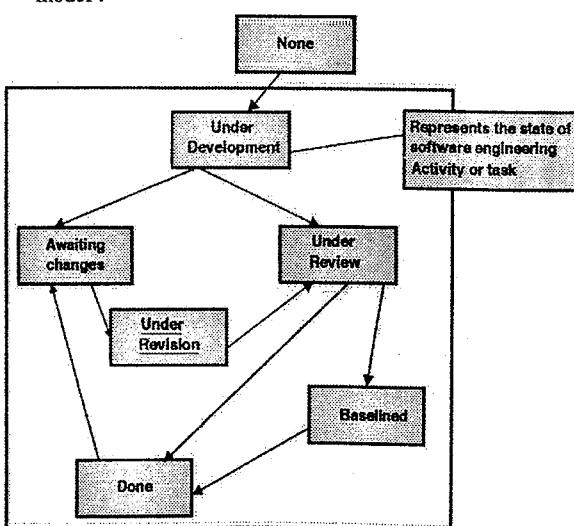


Fig. 1-Q.2.8 : One element of concurrent process model (i.e. Modeling activity)



- The **modelling activity** is one of the states and other activities like **communication** or **construction** can also be represented in an analogous manner.
- Let the **communication activity** has completed its first iteration and available in **awaiting changes state**. The **modelling activity** has completed its initial communication and ready to move to **under development state** from none state.
- During these transitions, if customer indicates some modification in the requirement, then the **modelling activity** transits to **awaiting changes state** from **under development state**.
- Instead of considering activities, actions and tasks as sequence of events, it defines network of activities.
- The concurrent process model activities transits from one state to another state and this model is used in all types of software development and it provides very clear idea of the current state of the project.

Chapter 3 : Advanced Process Models and Tools

Q. 3.1 Explain Agile process model. (3 Marks)

Ans. :

- An agile process model includes the concept of development along with a set of guidelines necessary for the development process. The conceptual design is necessary for the satisfaction of the customer. The concept is also necessary for the incremental delivery of the product. The concept or the philosophy makes development task simple. The agility team must be highly motivated, updated with latest skill sets and should focus on development simplicity.
- We can say that agile development is an alternative approach to the conventional development in certain projects.
- The development guidelines emphasize on analysis and design activities and continuous communication between developers and customers. An agile team quickly responds to changes. The changes may be in development, changes in the team members, and changes due to new technology. The agility can be applied to any software process. It emphasizes rapid delivery of operational software.

- All the agile software processes should address three important assumptions :
 - o Difficult to predict in advance software requirements
 - o Design and construction are interleaved in most of the projects, it is difficult to predict design before construction.
 - o Analysis, design, construction and testing are not much predictable.
- To address these assumptions of unpredictability, the agile development process must be adaptable. So an agile process must adapt incrementally.

Q. 3.2 . Discuss Agile manifesto. (3 Marks)

Ans. :

- The Agile Manifesto states that :

“We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value,

 - o **Individuals and interactions** work over processes and tools
 - o **Working software** takes responsibility of comprehensive documentation
 - o **Customer collaboration** look after contract negotiation
 - o **Responding to change** after having a good plan.”
- Even though these agile methods are based on the concept of incremental development, quicker delivery and faster deployment, the agile manifesto gives different processes to achieve this.
- But most of the set of principles used by various experts that are based on agile manifesto are very much common.

Q. 3.3 List the principles of Agility. (4 Marks)

Ans. :

- Agile software development describes a set of principles for software development under which requirements and solutions evolve through the collaborative effort of self-organizing cross-functional teams.
- It advocates adaptive planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change.



- The agile alliance has given twelve agility principles as follows :
 1. Satisfy customer through early and continuous delivery.
 2. Accommodate changing requirements.
 3. Deliver the working software frequently in shorter time span.
 4. The customer, business people and developers must work together on daily basis during entire development.
 5. Complete the task with motivated developers and facilitate healthy and friendly environment.
 6. Convey the message orally, face-to-face conversation.
 7. Working software is the primary measure of progress.
 8. Agile process promotes sustainable development.
 9. Achieve technical excellence and good design.
 10. There must be simplicity in development.
 11. The best architecture, requirements and design emerge from self-organizing teams.
 12. Every team should think how to become more effective. It should review regularly and adjust its behaviour accordingly.
- Thus the principles of agility may be applied to any software process. To achieve agile development, the team must obey all the principles mentioned above and conduct proper planning.
- All the agile software processes should address three important assumptions :
 - o Difficult to predict in advance software requirements
 - o Design and construction are interleaved in most of the projects, it is difficult to predict design before construction.
 - o Analysis, design, construction and testing are not much predictable.
- To address these assumptions of unpredictability, the agile development process must be adaptable. So an agile process must adapt incrementally.

Ans. :

- The Extreme Programming is one of the most commonly used agile process models. All the agile process models obey the principles of agility and the manifesto of agile software development.
- The XP uses the concept of object oriented programming. This approach is preferred development paradigm.
- As in conventional approach, a developer focuses on the framework activities like planning, design, coding, and testing, the XP also has set of rules and practices.
- Following Fig. 1-Q.3.4 shows the Extreme Programming process and all the key XP activities are summarized :

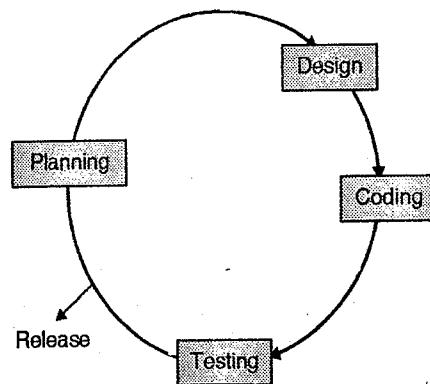


Fig. 1-Q.3.4 : Extreme programming process

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

Q. 3.4 What is Extreme Programming ? (4 Marks)

OR Explain the Extreme Programming Process.

(4 Marks)



Q. 3.5 What is concept of SCRUM? Explain. (4 Marks)

Ans. :

- It is one of the agile software development methods. It is an iterative and incremental software development framework. It gives holistic and flexible development strategies.
- It enables teams to self organize by online collaborations of team members. Its key principle is to recognize that during project development customers can change their requirement and requirements of customers are unpredictable.
- It adopts empirical approach. It assumes problem cannot be fully understood or defined but focusing on maximizing team ability to deliver quickly.
- Different terminologies used in SCRUM process is as follows :
 - o **SCRUM team** : It contains product owner, development team and scrum master.
 - o **Product owner** : Person responsible for product backlog.
 - o **SCRUM master** : Person responsible for scrum process.
 - o **Development team** : Team of people responsible for delivering the product.
 - o **Sprint burn down chart** : Daily sprint progress.
 - o **Release burn down chart** : Chart of completed product backlog.
 - o **Product backlog** : List of priority wise requirement.
 - o **Sprint backlog** : List of task to be completed which are prioritized.
 - o **Sprint** : Period in which development occurs.
 - o **Spike** : Period used to research concept.
 - o **Tracer bullet** : It is spike with current architecture, technology, best practices, etc.
 - o **Tasks** : Items added to sprint backlog at the beginning of sprint which are broken into hours.
 - o **Definition of done** : Exit criteria which decides product backlog items are completed or not.
 - o **Velocity** : Total efforts a team is capable of in a sprint.
 - o **Scrum-but** : It is exception to scrum methodology.

Q. 3.6 Compare Exploratory testing with scripted testing in the context of agile software development. (3 Marks)

Ans. :

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

Q. 3.7 What are the benefits of Pair Programming ? (3 Marks)

Ans. : There are numerous advantages of pair programming.

- **Economy** : It spends around 15% more time on programming. But designed code is efficient than that of



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

Q. 3.8 Explain Refactoring in the context of agile software development ? (4 Marks)

Ans. :

- It is one of the agile practices in agile software development.
 - It is process of restructuring the existing code.
 - The changes are made without affecting external behaviour.
- It does not change the functional requirement of the code.
- It changes the non-functional requirement of the code. Non functional requirements are useful in efficiency, performance, throughput, etc.
 - It has numerous advantages like improved readability, reducing complexity, reusability, etc.
 - It results in creating more expressive software architecture which results in extensibility.
 - Generally series of standards are applied for code refactoring.
 - Some standards are used which are designed by organization and some other standards also used in code refactoring.
 - They are generally identified by code smell.

- E.g. we have code which is very long or it is almost duplicate of another code. In this situation code refactoring can be applied where we can remove the code duplication. After refactoring the basic functionality of the code remain same.
- Code duplication can be removed by designing shared function. Shared by will be used whenever required instead of creating same copy of code again and again.
- Two main objectives of refactoring are maintainability and extensibility.
- With code maintainability, we can easily identify the bugs in the software as the code is properly documented.
- With code extensibility, new features can be added very easily.
- We need automatic unit test before applying code refactoring.

Chapter 4 : Requirement Engineering

Q. 4.1 Explain Requirements Engineering? (4 Marks)

Ans. :

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



- All these seven functions focus on the customer's needs and care must be taken to satisfy it. All these functions collectively form the strong base for software design and construction. These functions are :

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

Q. 4.2 What do you mean by requirement inception ?
(3 Marks)

Ans. :

- Inception is beginning and it is usually said that, 'well beginning is half done'. But it is always problematic for the developer that what should be the starting point i.e. 'from where to start'.
- The customer and developer meet and they decide the overall scope and nature of the problem statement. The aim is :
 - To have the basic understanding of the problem.
 - To know the people who will use the software.
 - To know exact nature of problem that is expected from customer's side.
 - To maintain effectiveness of preliminary communication.
 - To have collaboration between customer and developer.
- The requirements engineering is a 'communication intensive' activity because a requirement gathering is an initial step for design. Hence exact requirements are gathered with customer communication.
- The developer must ask following questions:
 - Who is behind the request for this work?
 - Who will use the solution ?
 - What will be the economical benefits of a successful solution ?
 - Is there another source of solution for the same problem ?

Q. 4.3 What do you mean by requirement elicitation ?
(3 Marks)

Ans. :

- Elicitation means, 'to draw out the truth or reply from anybody'. In relation with requirements engineering, elicitation is a task that helps the customer to define what is required.
- To know the objectives of the system or the project to be developed is a critical job. Why it is difficult to get a clear understanding of customer wants? To answer this question, we have a number of problems like :

Problems of scope

Many times customer states unnecessary technical details. The unnecessary details may confuse developer instead of giving clarity of overall system objectives.

Problems of understanding

Sometimes both customer as well as developer has poor understanding of :

- What is needed ?
- Capabilities and limitations of the computing environment.
- Understanding of problem domain.
- Specify requirements those conflict with other customers and users.

Problems of volatility

Volatility means 'change from one state to another'. The customer's requirements may change time to time. This is also a major problem while deciding fixed set of requirements.

Q. 4.4 What do you mean by requirement management ?
(3 Marks)

Ans. :

- Requirement management is a software engineering task that helps the project team members to identify the requirements, control the requirements, track the requirements and changes to requirements at any time as the project exceeds.
- The requirement management task starts with identification and each of the requirements is assigned a unique identifier.
- After the requirements finalization, the traceability table is developed. Traceability table relates the requirements to one

or more aspects of the system or its environment. The traceability tables are used as requirements database and are useful in understanding how change in one particular requirement will affect other parts or aspects of the system.

**Q. 4.5 Explain User and System Requirements ?
(4 Marks)**

Ans. :

- The fundamental definition of the requirements for a system is :

"The set of descriptions listed for the system to accomplish the required services and the constraints on its operation."

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

- Fig. 1.Q.4.5 exhibits the readers of different types of requirements specification :

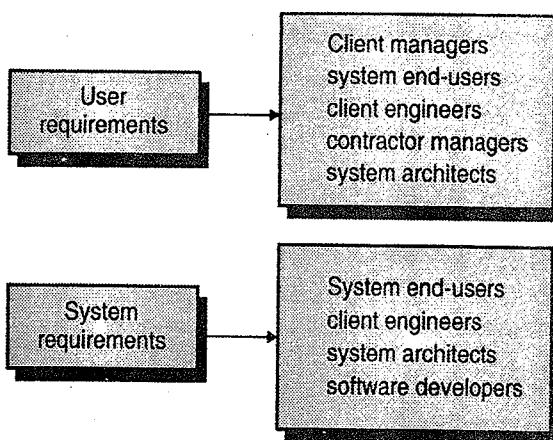


Fig. 1-Q. 4.5 : Readers of different types of requirements specification

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

**Q. 4.6 What are functional and non functional requirements of software?
(4 Marks)**

Ans. :

- The system requirements are classified into following two types :
 - o Functional requirements
 - o Non-functional requirements
- The **functional requirements** are the statements of the services that a system should provide and how the system must react to a given input and the system reacts in some situations. The functional requirement should also mention that a system should do nothing in some situations.
- The **non-functional requirements** are treated as some constraints that a system should behave in some situations. Also the non-functional requirements are the explicit conditions that are put on the services and the functionalities of the system to behave. The non-functional requirements are applied on the whole system and not on the individual system features.



Functional Requirements

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

Non-functional Requirements

- The non-functional requirements are not directly related to the functions, services of the system and the desired outputs.
- The non-functional requirements are related to the system properties that are listed below :
 - o Reliability
 - o Speed of responses i.e. response time,
 - o System performance,
 - o System security,
 - o System availability,

- o Portability,
- o Robustness,
- o Ease of use etc.

- In contrast to functional requirements, the non-functional requirements are more critical. Therefore, if functional requirements are failed then the whole system may be failed and become unusable.
- For example if a air system does not provide reliability, then it can not be used in actual practice. It will not be called as a safe system.
- The failure of an individual non-functional requirement may lead to failure of the whole system.

Q. 4.7 Explain Spiral View of the Requirements Engineering Process ? (3 Marks)

Ans. :

- Basically requirements engineering processes includes following four high-level activities :
 - o **Feasibility study** : To check whether the system is useful for the business.
 - o **Requirements elicitation and analysis** : Finding the requirements from the user and conduct analysis on them.
 - o **Requirements specification** : Requirements received in some natural language and in the form of some diagrams and convert them in some standard form.
 - o **Requirements validation** : It is actually the checking of systems functioning. Whether it is in function according to the requirement elicited by the user.
- All these activities constitute requirements engineering process and can be organized as an iterative process to form a spiral view. In actual practice, all these activities are interleaved in one another. The spiral view of the activities involved in the requirements engineering process as exhibited as follows :

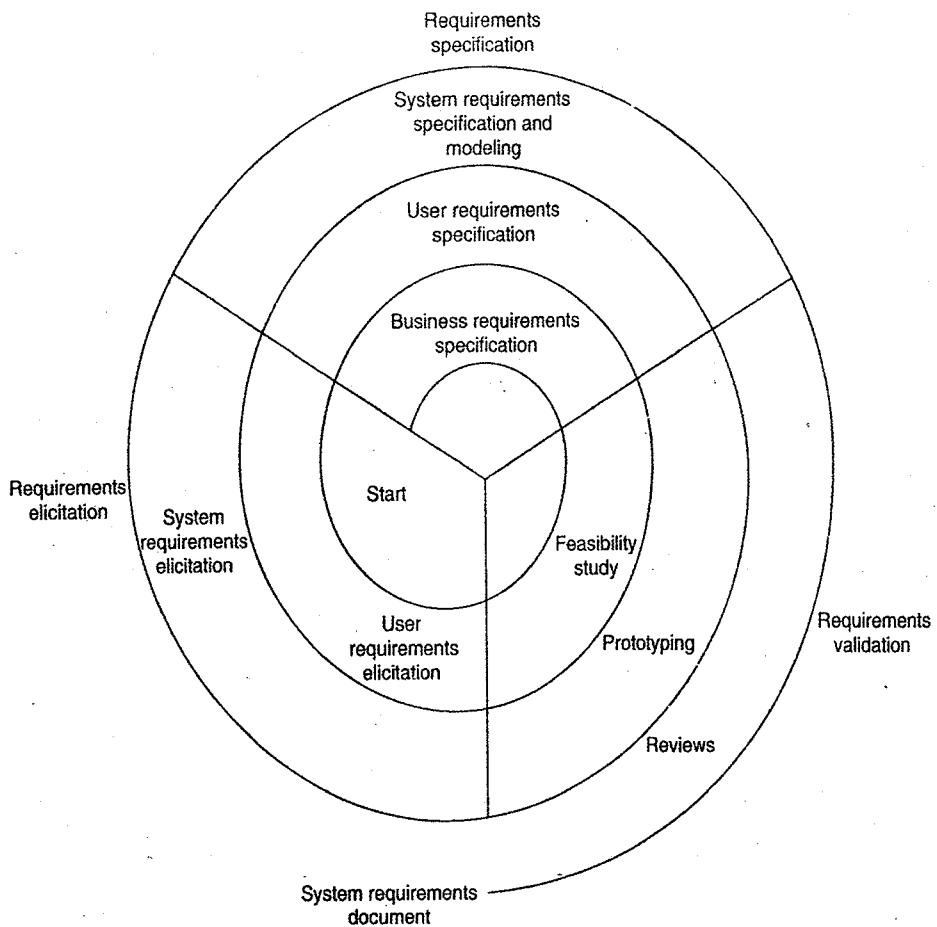


Fig. 1-Q. 4.7 : Spiral view of the activities involved in the requirements engineering process

Q. 4.8 Explain different desirable characteristics of a good Software Requirements Specification(SRS) document. (3 Marks)

Ans. :

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

- **Unambiguous** : Statement in the SRS document should have only one meaning.

- **Valid** : All the requirements should be valid.

Q. 4.9 Write SRS Template that can be used in writing SRS documents. (3 Marks)

Ans. :

- Different existing SRS templates can be used in writing SRS documents.

We can select the template which matches our requirement.

- Template will just acts as guiding principles for writing template. Proper changes need to be done whenever necessary in template.

- Table 1-Q.4.9 shows SRS outline:

Table 1-Q.4.9 : A sample of a basic SRS outline

- | |
|--|
| <p>1. Introduction : It contains different details like purpose of software, conventions used, target audiences, extra information, SRS team members, references.</p> |
|--|



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

Q. 4.10 Explain the structured SRS with case study of Insulin Pump. **(4 Marks)**

Ans. :

- Structured natural language is a way of writing system requirements where the freedom of the requirements writer is limited and all requirements are written in a standard way. This approach maintains most of the expressiveness and understand ability of natural language but ensures that some uniformity is imposed on the specification. Structured language notations use templates to specify system requirements. The specification may use programming language constructs to show alternatives and iteration, and may highlight key elements using shading or different fonts.
- The Robert sons, a well-known author, recommend that user requirements be initially written on cards, one requirement per card. They suggest a number of fields on each card, such as the requirements rationale, the dependencies on other requirements, the source of the requirements, supporting materials, and so on. This is similar to the approach used in the example of a structured specification shown in Table 1-Q. 4.10.

Table 1-Q. 4.10 : A structured specification of a requirement for an insulin pump

Function	Compute insulin dose: Safe sugar level.
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.
Inputs	Current sugar reading (r_2), the previous two readings (r_0 and r_1).
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose—the dose in insulin to be delivered.
Destination	Main control loop.
Action	CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.
Requirements	Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin.
Post-condition	r_0 is replaced by r_1 then r_1 is replaced by r_2 .
Side effects	None.

- To use a structured approach to specifying system requirements, you define one or more standard templates for requirements and represent these templates as structured forms. The specification may be structured around the objects manipulated by the system, the functions performed by the system, or the events processed by the system. An example of a form-based specification, in this case, one that defines how to calculate the dose of insulin to be delivered when the blood sugar is within a safe band, is shown in Table 4.7.2.



Q. 4.11 Explain the Quality Function Deployment.
(4 Marks)

Ans. :

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

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

1. Normal requirements

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

For example

- o Graphic displays.
- o Specific system functions.
- o Specified output formats.

2. Expected requirements

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

For example

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

3. Exciting requirements

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

Q. 4.12 What are the work products of elicitation ?
(3 Marks)

Ans. :

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

Q. 4.13 Explain Domain analysis. Also discuss its advantages.
(3 Marks)

Ans. :

- By definition, software domain analysis is "the identification, analysis and specification of common requirements from a specific application domain." These common software domains can be reused for multiple projects within that application domain.
- In short, software domain analysis leads to 'reusable software components' in software development. Here, in specific application domain common objects, common classes, common frameworks can be identified and can be reused.
- Role of 'domain analyst' is same as job of tool smith. The job of tool smith is to design the tools those are used by many people for similar works. The role of domain analyst is to define reusable objects, classes, frameworks those can be used by many people in similar applications.
- **For example :** 'The specific application domain' may be 'bus reservation system'. The software domains of 'Us reservation system' can be used for 'railway reservation system'.

Advantages

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



Q. 4.14 Draw and explain the traceability table for requirement management. (4 Marks)

Ans. :

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

Table 1-Q. 4.14 : Generic traceability table

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

Chapter 5: Design Engineering

Q. 5.1 Define Design Engineering and explain design process in short. (4 Marks)

Ans. : Design Engineering

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

Design Process

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

activity diagram, use case diagram and sequence diagram,
etc.

In software design process, the design patterns and architectural styles are also used.

Q. 5.2 Illustrate Design Quality Guidelines in short.
(3 Marks)

Ans. :

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

Quality of Design Guidelines

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

Q. 5.3 Explain the quality attributes, considered in software design. **(4 Marks)**

Ans. i

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

- **Functionality** is an important aspect of any software system. It is generally evaluated by the feature set and capabilities of that software.

- Following are general features that a system is supposed to deliver.
 - **Usability** is best evaluated by considering following important factors :
 - o Human factors
 - o Overall aesthetics
 - o Consistency and documentation.
 - **Reliability** is assessed by measuring following parameters :
 - o Frequency and severity of failure
 - o Accuracy of output results
 - o Mean-time-to-failure (MTTF)
 - o Recovery from failure and
 - o Predictability of the program
 - **Performance** is evaluated by considering following characteristics :
 - o Speed
 - o Response time
 - o Resource consumption
 - o Throughput and
 - o Efficiency
 - **Supportability** collectively combines following three important attributes :
 - o Extensibility
 - o Adaptability
 - o Serviceability
 - Most commonly, above three attributes can be better defined by the term maintainability. In addition to these attributes, following are some more attributes with which a system can be installed easily, and the problems can be found out easily :
 - o Testability,
 - o Compatibility, and
 - o Configurability,
 - It also contains more attributes which are as follows :
 - o Compatibility
 - o Extensibility
 - o Fault tolerance
 - o Modularity



- Reusability
- Robustness
- Security
- Portability
- Scalability

Q. 5.4 Explain the following Design concept :
1. Modularity 2. Architecture **(4 Marks)**

Ans. :

1. Modularity

- The modularity consists of software architecture and design patterns i.e. software is divided separately according to name and addresses of components and sometimes it is called as modules that are integrated to fulfil problem requirements.
- The modularity is defined as the modularization of a single attribute of software into number of small parts such that these parts can be easily managed.
- The number of variables, span of reference, number of control paths and overall complexity will make the understanding of the program nearly impossible.

2. Architecture

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

Q. 5.5 Explain the following Design concept :
1. Abstraction 2. Patterns **(4 Marks)**

Ans. :

1. Abstraction

- When we consider a modular solution to the problems, there are many levels of abstraction possible. In the highest level of abstraction, there is a solution that is stated in broad terms using the language of the problem environment.

- The lower levels of abstraction are used for detailed description and hence a more detailed description of the solution is provided at lower levels.

- A data abstraction is actually a collection of data that describes a data object.

2. Patterns

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

Q. 5.6 What do you mean by the term cohesion and coupling in the context of software design? **(4 Marks)**

Ans. :

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

- 1. Cohesion**
- 2. Coupling**

Cohesion is a relative functional strength of a module and coupling is the relative interdependence among modules.

1. Cohesion

- Cohesion is an extension of the concept of information hiding. In cohesion, a module performs only one task within a software procedure i.e. requires only a little interaction with procedures that are in other parts of a program.
- Cohesion can be represented as a "spectrum". High cohesion is always recommended but mid-range of the spectrum is often acceptable. The cohesion scale is nonlinear.
- That is, low-end cohesiveness is worse than middle range, which is nearly same high-end cohesion.

2. Coupling

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

Q. 5.7 Explain the following design concept : Refinement, Refactoring. (4 Marks)

Ans. : Refinement

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

Refactoring

- An important design activity, refactoring, is a reorganization technique that simplifies the design (or code) of a component without changing its function or behavior or in other words, refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code [design] yet improves its internal structure.
- The refactoring is used in improving the quality of software.
- Following are some points that prove the importance of refactoring :
 - o By improving the quality of the code, working becomes easier. By using code refactoring, the addition of new code and maintenance of the code becomes easy.
 - o The programmer do not write perfect codes, hence refactoring improves the code over the period of time.

- o Refactoring play important role in splitting long functions into reasonably small sub functions.
- o It helps in creating reusable codes.
- o Error handling is much easier and within control.

Chapter 6 : Architectural Design

Q. 6.1 Explain the Architectural Design. (4 Marks)

Ans. :

- Architectural design is backbone of any software system design and it is responsible for the overall structure of the system.
- In nearly all the models of software process, architectural design is considered as the first stage in the software design and development process.
- The output of the architectural design process is an architectural model that explains the overall structure of the system and also explains the working of the system and the communication among various components of the system.
- In case of agile development processes also, the first stage of development process is defining the system architecture. Incremental approach of the software development is not generally successful whereas the concept of refactoring of the components bit easy. But refactoring of the system architecture is expected to be expensive.
- In order to understand the system architecture, Fig. 1-Q. 6.1 provides a simple model of architecture for a remote control car :

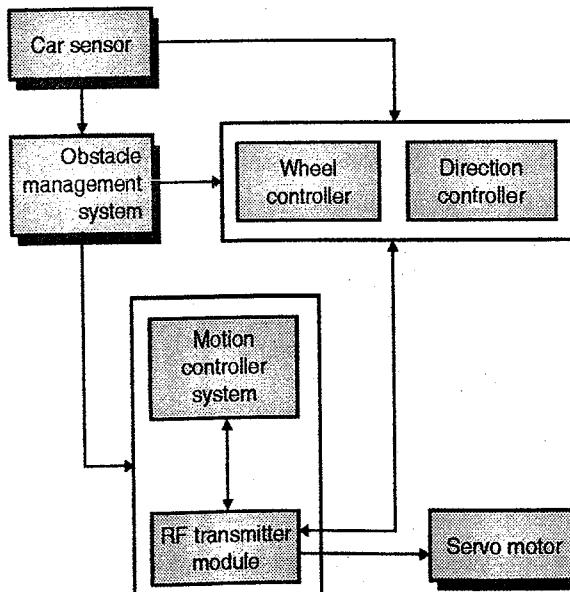


Fig. 1-Q.6.1 : The architecture of a remote control car



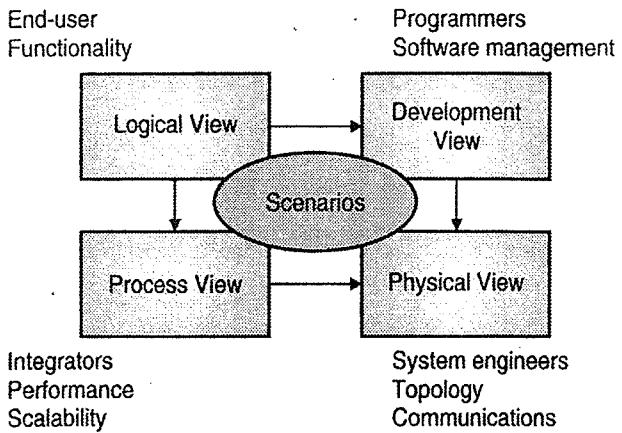
- The Fig. 1-Q.6.1 exhibits the abstract model of architecture of a remote control car system that will help to understand the functioning of the car.

Q. 6.2 Explain the Architectural Views. (4 Marks)**Ans. :**

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

The non-functional requirements are also taken into consideration like scalability, reliability, availability and portability.

- Software architecture also deals with the style and aesthetics. In the Fig. 1-Q.6.2 software architecture is illustrated as a model presenting five main views.

**Fig. 1-Q.6.2 : 4 + 1 View Architecture**

- The **logical view**, which is the object model of the design (when an object-oriented design method is used),
- The **process view**, which captures the concurrency and synchronization aspects of the design,
- The **physical view**, which describes the mapping(s) of the software onto the hardware and reflects its distributed aspect,
- The **development view**, which describes the static organization of the software in its development environment.

Q. 6.3 Discuss architectural patterns in details. (3 Marks)**Ans. :**

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

**Q. 6.4** Discuss Architectural Patterns in details. (3 Marks)**Ans. :**

- When we discuss the architecture of a building, it is the manner in which the various components of the building are integrated to form a complete structure.
- It is the way in which the building fits into its environment and meshes with other buildings in its vicinity.
- Also visual impact of the building, and the way textures, colours and materials are combined to create the external appearance and the internal living environment.
- Similarly we can define the software architecture concept. The software architecture of a program or computing system is actually the structures of the system that consists of following attributes:
 - o Software components,
 - o The externally visible properties of software components, and
 - o The relationships among them.
- The architecture is basically not the operational software. But, it is a representation only that enables a software engineer to :
 - o Analyze the effectiveness of the design.
 - o Consider architectural alternatives, and
 - o Reduce the associated risks.
- This definition of software architecture focuses on the role of "software components" in any architectural representation. A software component can be as simple as a program module or an object oriented class, in the context of architectural design. The architecture can also be extended to include databases and middleware that enables the configuration of a network having clients and servers.

Q. 6.5 Enlist basic design principles for class-based components. (4 Marks)**Ans. :**

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

1. **The Open-Closed Principle (OCP) :** In OCP any module must be available for extension and modification. The

designer should specify the component in such a way that it can be extended without any modification in the internal structure of the component.

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

Chapter 7 : User Interface Design

Q. 7.1 Explain User Interface Design. (4 Marks)**Ans. :**

- Success of most of the software depends upon user interface design.
- It is part of human computer interaction. In user interface human controls the software.
- As two different entities are trying to communicate with each other, one is human being and other is computer.



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

Q. 7.2 What are different Type of User Interface ?
(4 Marks)

Ans. :

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

- 1. Command Interpreter
- 2. Graphical User Interfaces

1. Command Interpreter

- In command interpreter, user communicates with computer system with the help of commands.
- In Unix/Linux system the command interpreter is known as shell. Shell is nothing but the interface between user and operating system.
- All the shells give similar facility with only minor changes. Selecting the particular shell totally depends on user's requirement. Command interpreter just accept the commands, call the corresponding program and execute it.
- When we type the "cd" command, operating system search the program for the "cd" command and get it executed. Many commands are provided by the operating system. The kind of command supported is depends upon the operating system.
- The commands are implemented in different manner. In the first approach, the command interpreter itself contains the code for the command.

2. Graphical User Interfaces (GUI)

- Another approach to interface with the computer system is through a user friendly graphical user interface or GUI.
- Instead of directly entering commands through a command-line interface, a GUI allows a mouse-based, window-and-menu based system as an interface.



- GUI gives the desktop environment where mouse and keyboard can be used. Mouse can be moved to certain location and when we click on some location operating system calls the corresponding program. Depending on the mouse pointer location, mouse click will invoke the program to execute. For every mouse click the corresponding program is start executing.

Q. 7.3 What are different Characteristics of Good User Interface ? (3 Marks)

Ans. :

There are different characteristics of good user interface :

- Clarity
- Concision
- Familiarity
- Responsiveness
- Consistency
- Aesthetics
- Efficiency
- Attractive
- Forgiveness

Benefits of Good Interface Design

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

Q. 7.4 State and explain Golden rules for GUI ? (4 Marks)

Ans. :

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

1. **Place the user in Control**
 - The main motive behind using interface constraints and restrictions are to simplify the mode of interaction.
 - In most of the cases, the developer may introduce constraints and limitations to simplify the implementation of the interface.
 - The ultimate result could be an interface that is easy to develop, but frustrating to use.
 - There are a number of design principles that allow the user to maintain control.
- Define the interaction modes in such a way that the user is not forced to perform unnecessary or undesired actions.
2. **Reduce the User's Memory Load**

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

Reduce the demand for short-term memory

 - When users are performing a complex task, then the demand of short-term memory might be significant.
 - The user interface must be designed in a way that reduces the requirement to remember its past actions and results. i.e. it should be able to reduce the short-term memory use.
 - This feature is achieved by providing visual clues that will help a user to remember past actions, rather than a need to recall them.
3. **Make the Interface Consistent**

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

The system should allow its users to put the ongoing task into a meaningful context

 - Most of the interfaces implement complex layers of interactions with dozens of screen images.
 - It is important to provide indicators like window titles, consistent colour codes and graphical icons that helps the user to know the context of the work at hand.
 - In addition, the user should be able to determine where he has come from and what alternatives exist for a transition to a new task.



- The system should maintain the consistency within a family of applications.
- A set of applications (or products) should all implement the same design rules so that consistency is maintained for all interaction.
- The new systems should not change any model that is created by the user's expectation, unless there is some genuine reasons.
- Once a particular interactive sequence has become a standard, whether it is right or wrong, the user community expects the same in all other applications. The best example is: Control S for saving a file.
- Any modification in use of Control S will lead to confusions. For example, if we use Control S for scaling, it leads to confusion.

Q. 7.5 What is the necessity of a good user interface ?
(3 Marks)

Ans. :

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

Q. 7.6 Explain the user interface design process.
(4 Marks)

Ans. :

- The analysis and design process for user interfaces is iterative and can be represented using a spiral model.
- Referring to Fig. 1-Q.7.6, the user interface analysis and design process encompasses four distinct framework activities :
 1. User, task, and environment analysis and modeling.

2. Interface design.
3. Interface construction (implementation).
4. Interface validation.

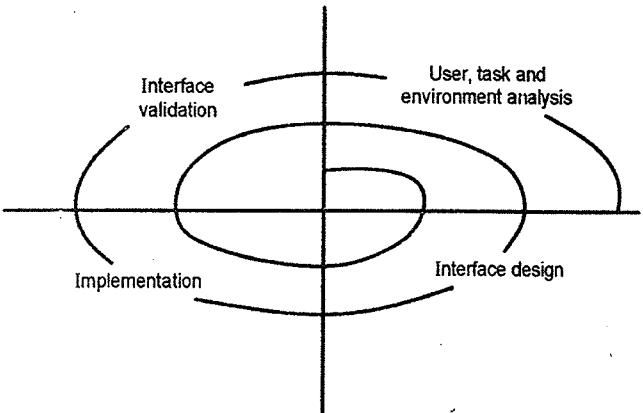


Fig. 1-Q.7.6 : The user interface design process

- The spiral shown in Fig. 1-Q.7.6 implies that each of these tasks will occur repeatedly along with each of the passes of spiral to represent the additional explanation of requirements and the final design.
- In many cases, the construction activity includes the prototyping that is the only practical way to validate the design.
- Interface analysis always focuses on the profile of the users. These are categorized as : skilled users, business level users etc and the class of users having general capabilities are recorded.
- Once general requirements have been defined, a more detailed task analysis is conducted. These tasks are identified and are performed by the user to reach the goals of the system.
- The analysis of the user environment focuses on the physical work environment.
- The information gathered as part of the analysis activity is used to create an analysis model for the interface. Using this model as a foundation, the design activity starts.
- The ultimate goal of interface design is to define interface objects and their actions that will help user to perform all defined tasks.

Q. 7.7 Explain the user interface design steps. **(3 Marks)**

Ans. :

- Once the interface analysis is completed, all the tasks (or objects and actions) required by the end-user have to be identified in detail, and the interface design activity starts. Interface design, is an iterative process like all other software



- engineering design processes.
- Each of the user interface design step occurs number of times, each elaborating and refining information developed in the previous step.
 - However so many user interface design models have been proposed, all of them suggest some combination of the following steps:
 1. Using information developed during interface analysis; define interface objects and actions (operations).
 2. Define events (user actions) that will cause the state of the user interface to change. Model this behaviour.
 3. Depict each interface state, as it will actually look to the end-user.
 4. Indicate how the user interprets the state of the system from information provided through the interface.
 - In some cases, the interface designer may begin with sketches of each interface state (i.e., what the user interface looks like under various circumstances) and then work backward to define important design information like objects, actions etc.
 - Regardless of the sequence of design tasks, the designer must always follow the rules for user interface design.

**Q. 7.8 Explain in detail the user interface design issues.
(3 Marks)**

Ans. :

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

- **Response time :** Response time is one of the performance attributes of all the systems. This is the main complaint for many interactive applications. Generally, the system response time is measured from the point when user presses the enter key to the point when the software responds with the expected output or expected action.
- **Help facilities :** The users of the system require help every now and then. In short, in all of the software system, the help facility or the user manual is essential for the smooth use of software.
- Some of the software provides on-line help facilities also to get the problem solved without closing the application or the interface.

- **Error handling :** The large number of error messages and warnings are actually irritating and produces unnecessary hindrance in the operation. Until and unless the error messages or warnings are very critical, it should not be popped up. The critical messages like "Do you want to delete all data ?" is essential to be included. Thus increased use of error messages and warnings will increase the frustration of users.
- **Menu and command labelling :** Earlier the typed command were very common but now-a-days, the common mode of interaction point and pick interfaces. It has reduced the trust on typed commands. In most of the software, in addition to typed command, menu labels are used as a common mode of interaction.
- **Application accessibility :** The computing applications are used everywhere and the developer should take care in designing the user interface and it should include easy access for all the needs of users. The physically disabled people should also be able to use the application in an easy way.
- **Internationalization :** The software developers should take into consideration the design in such a way that it is useful for end-users from all around the world. The system must support the languages spoken by all these users. The user interface must be designed for generic use.

**Q. 7.9 Enlist and explain the Webapp design principles in detail.
(3 Marks)**

Ans. :

Following are some WebApp design principles :

- **Anticipation** means it should let the user know its next step or next move.
- **Communication** let the user know the status of any activity initiated by him through some text message or through some image.
- **Consistency** must be maintained throughout the WebApp. For example navigation controls, icons or menus must be consistent. With respect to its colour, shape throughout the design.
- **Efficiency** : The design of the interface must optimize the end user's work efficiency.



- | | |
|--|---|
| <ul style="list-style-type: none">- Flexibility is the integral part of the interface design. The user may explore the application in some random fashion. The system must be flexible enough to accommodate the needs of the user.- Focus the WebApp interface should stay focus on the user tasks at hand.- Readability : Information presented should be readable to all the users. | <ul style="list-style-type: none">- Learnability : The application must be designed to minimize learning time.- Main the work integrity through the system.- Latency reduction should minimize the waiting time for user and use that waiting for some internal operation to complete. |
|--|---|

□□□

Note

Key Notes

(Exam Point of View)

Exam Oriented Key Points

Chapter 1 : Software Engineering Fundamentals

Characteristics of nature of software

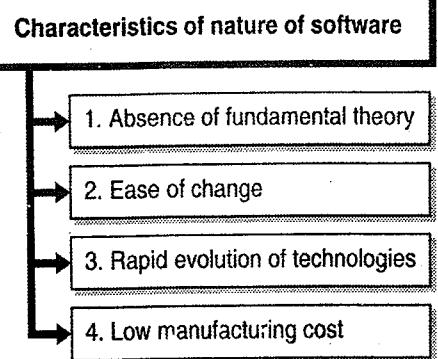


Fig. C1.1

Umbrella Activities

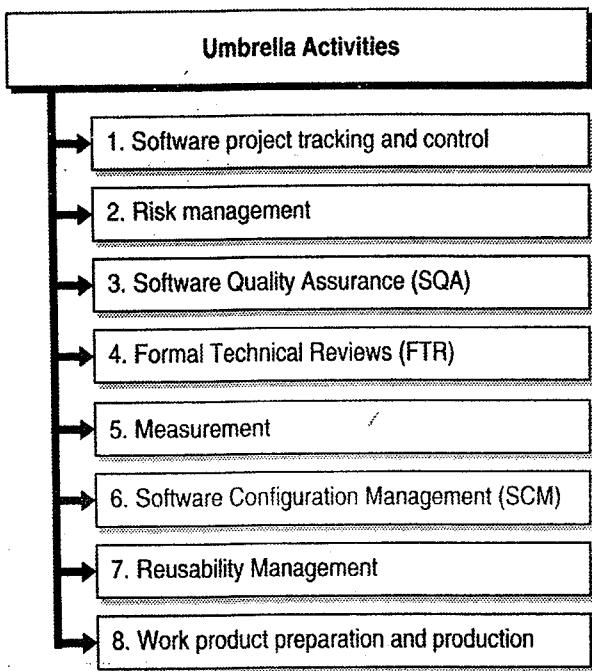


Fig. C1.2

Software Myths Level

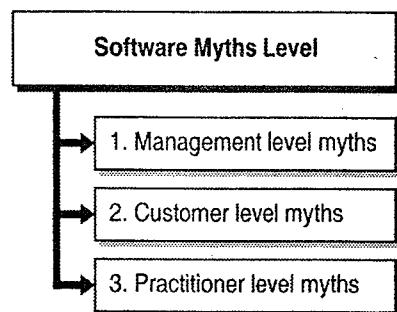


Fig. C1.3

Chapter 2 : Process Models

Generic Process Framework Activities

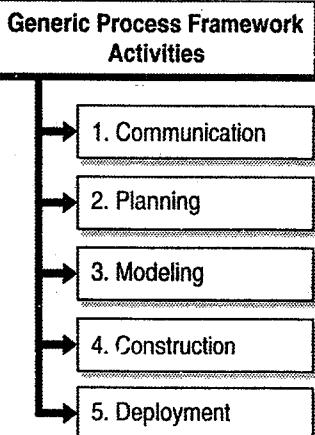


Fig. C2.1

Prescriptive Process Models

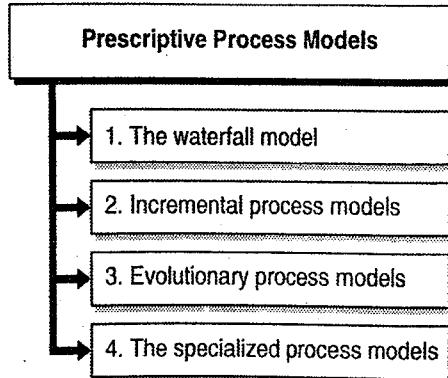


Fig. C2.2



☞ **Prescriptive Process Models**

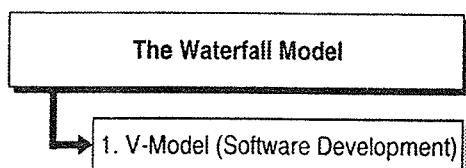


Fig. C2.3

☞ **Incremental Process Models**

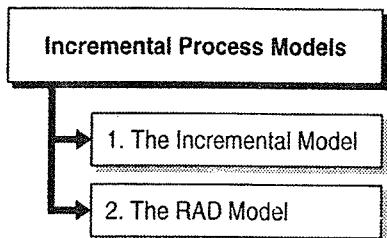


Fig. C2.4

☞ **Evolutionary Process Models**

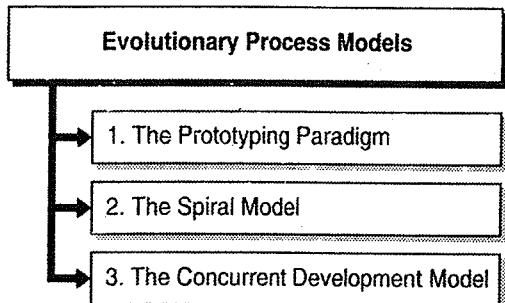


Fig. C2.5

☞ **The Specialized Process Models**

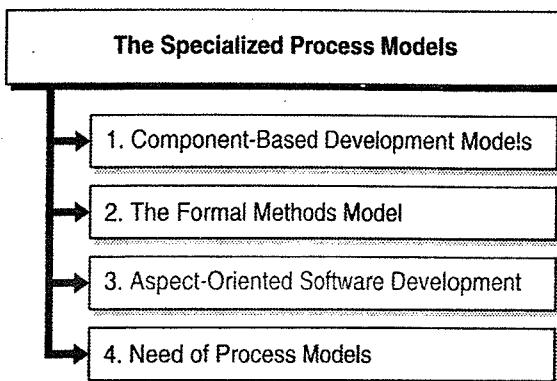


Fig. C2.6

Chapter 3 : Advanced Process Models and Tools

☞ **XP Framework Activities**

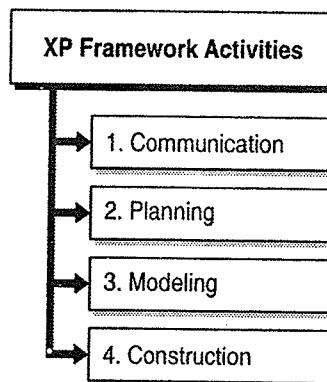


Fig. C3.1

☞ **Scrum Roles**

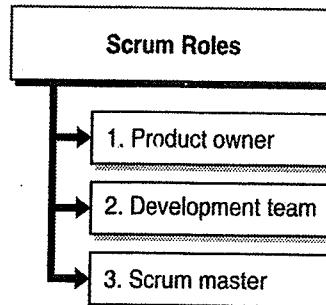


Fig. C3.2

☞ **Agile Practices**

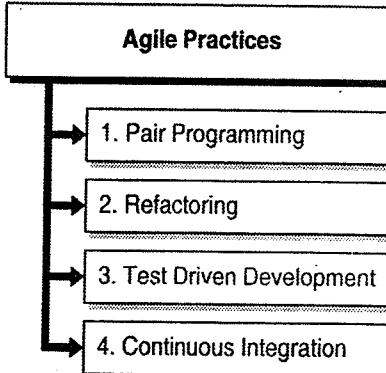


Fig. C3.3

Benefits of pair programming

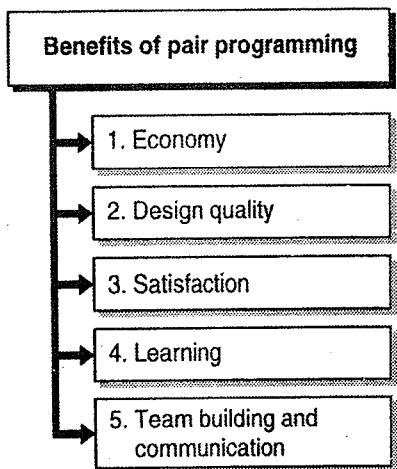


Fig. C3.4

Features of Mental Health-Care System

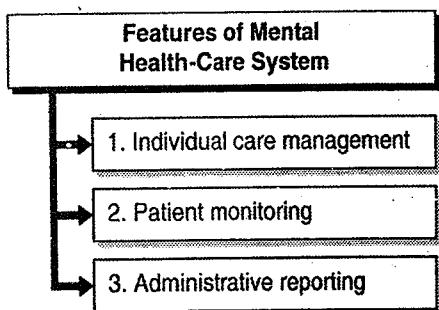


Fig. C3.5

Chapter 4 : Requirement Engineering

Requirement Engineering Functions

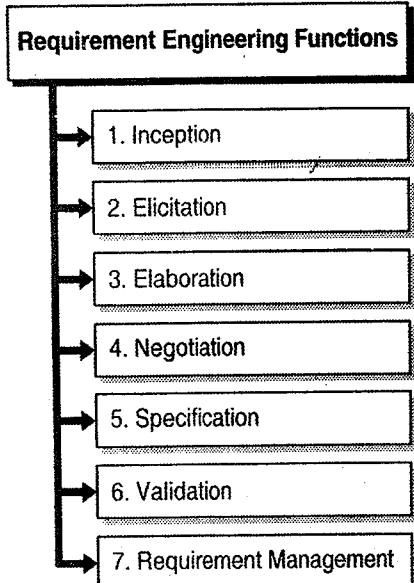


Fig. C4.1

SRS Outline

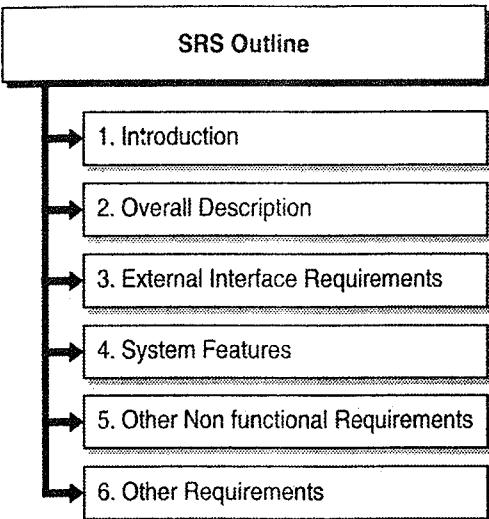


Fig. C4.2

Elicitation Requirements Methods

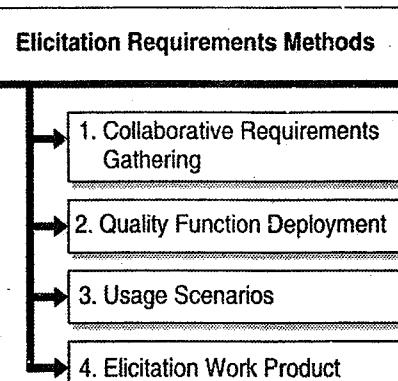


Fig. C4.3

Types of Requirements

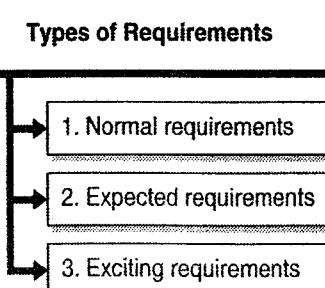


Fig. C4.4

Types of Actors

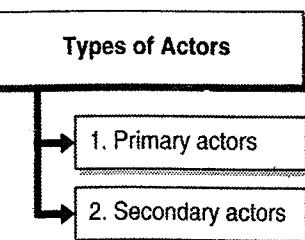


Fig. C4.5

Chapter 5 : Design Engineering

Design Concepts

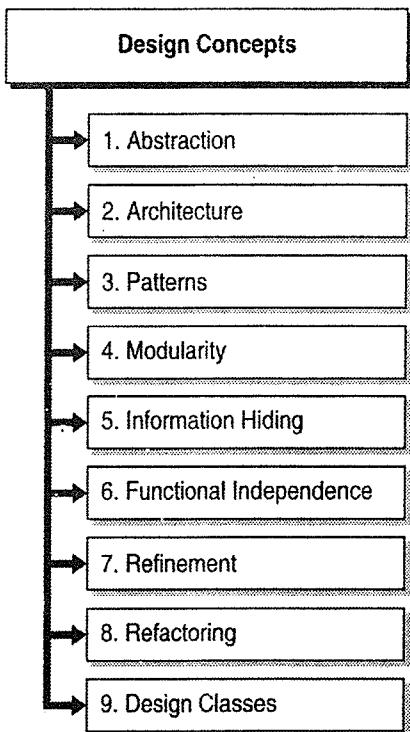


Fig. C5.1

Modular System Criteria

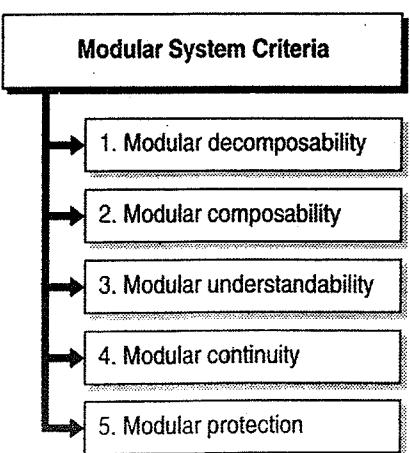


Fig. C5.2

Qualitative Criteria for Independence

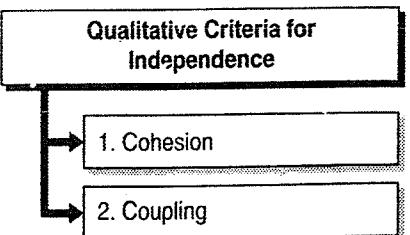


Fig. C5.3

Chapter 6 : Architectural Design

Application Architectures Example

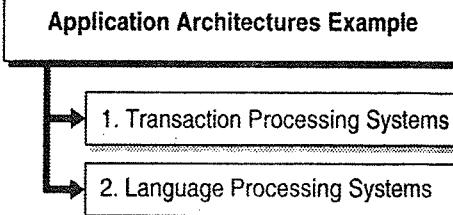


Fig. C6.1

Views of Component Design

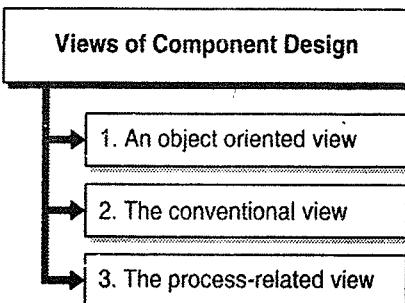


Fig. C6.2

Design Principles

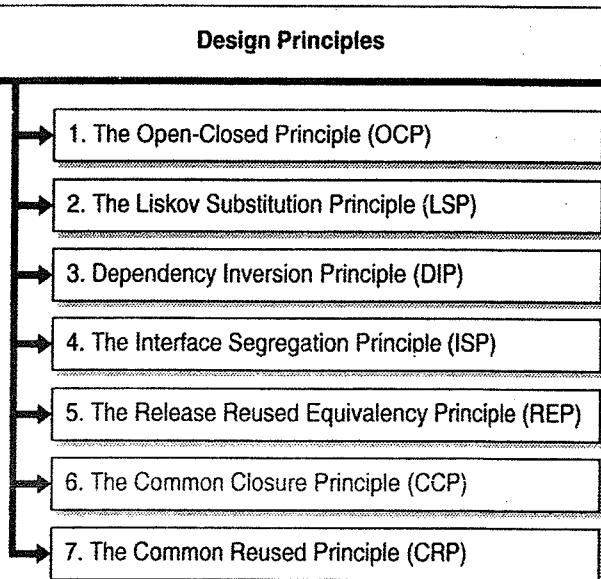


Fig. C6.3



Chapter 7 : User Interface Design

Type of User Interface

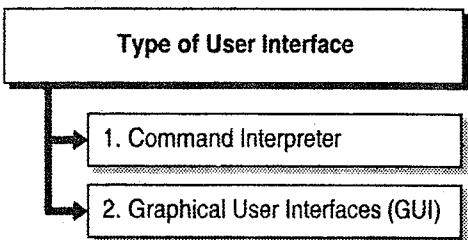


Fig. C7.1

Golden Rules

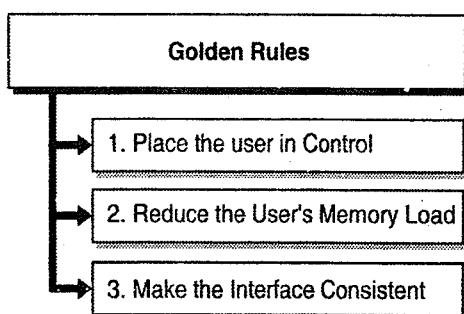


Fig. C7.2

Shneiderman's Golden Rules

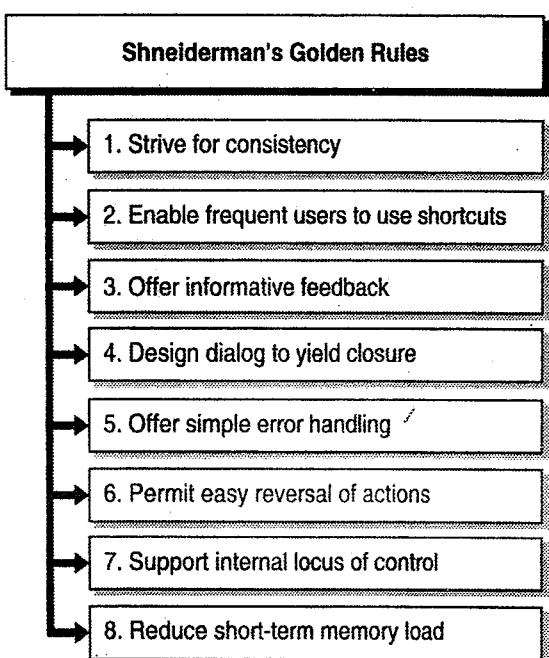


Fig. C7.3

Chapter 8 : Project Management Concepts

Four P's

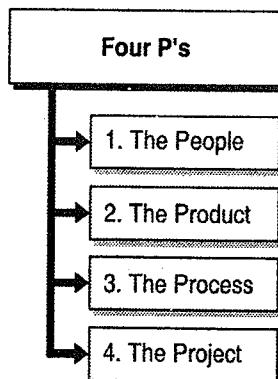


Fig. C8.1

Categories of Group

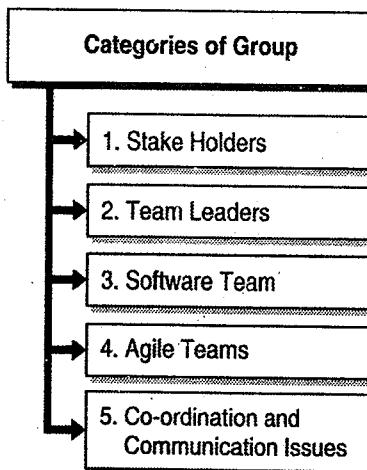


Fig. C8.2

Features of Modern Software

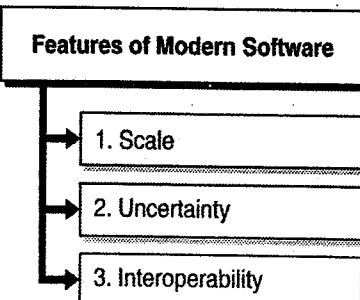


Fig. C8.3

Software Quality Factors

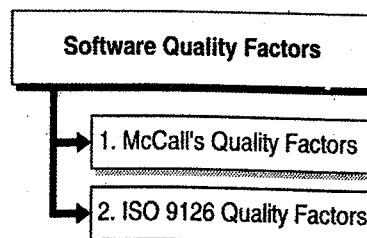


Fig. C8.4

Key Quality Attributes

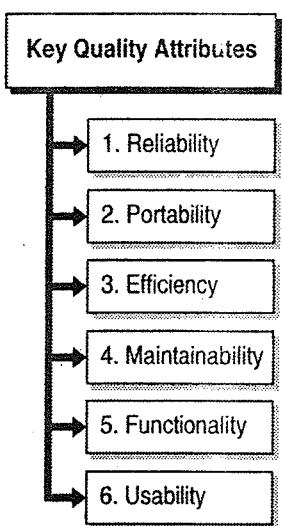


Fig. C8.5

Types of Decomposition Techniques

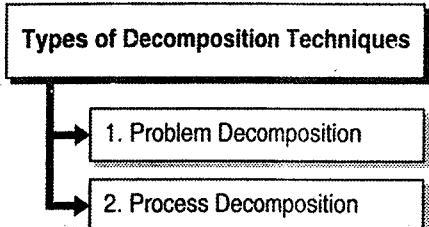


Fig. C8.6

Estimation Techniques

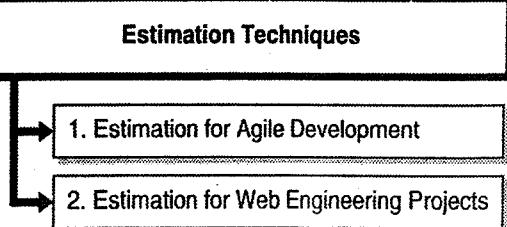


Fig. C8.7

Chapter 10 : Project Risk Management

Categories of Risk

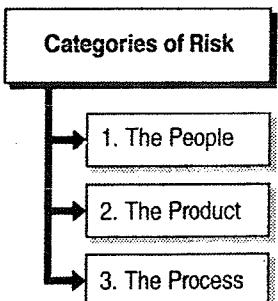


Fig. C10.1

Types of Risks

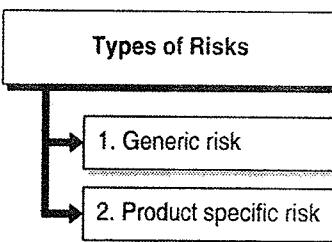


Fig. C10.2

Subcategories of Risks

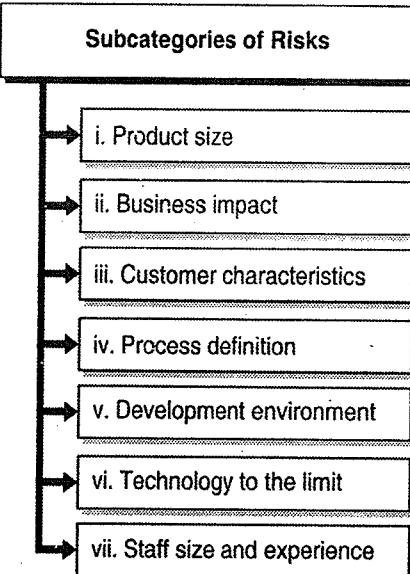


Fig. C10.3

Chapter 11 : Software Configuration Management (SCM)

Configuration Management Elements

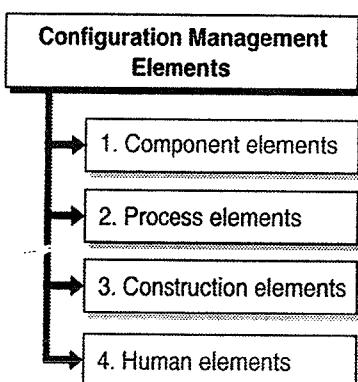


Fig. C11.1

SCM Features

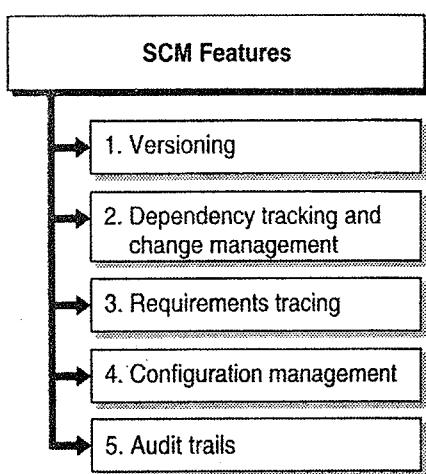


Fig. C11.2

Types of Objects

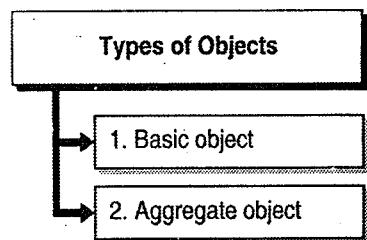


Fig. C11.3

Chapter 12 : Software Maintenance and Reengineering

Reverse Engineering Parameters

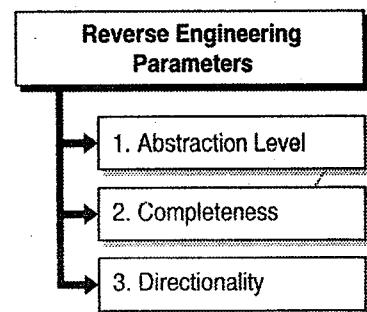


Fig. C12.1

Chapter 13 : Software Testing

Phases of Testing

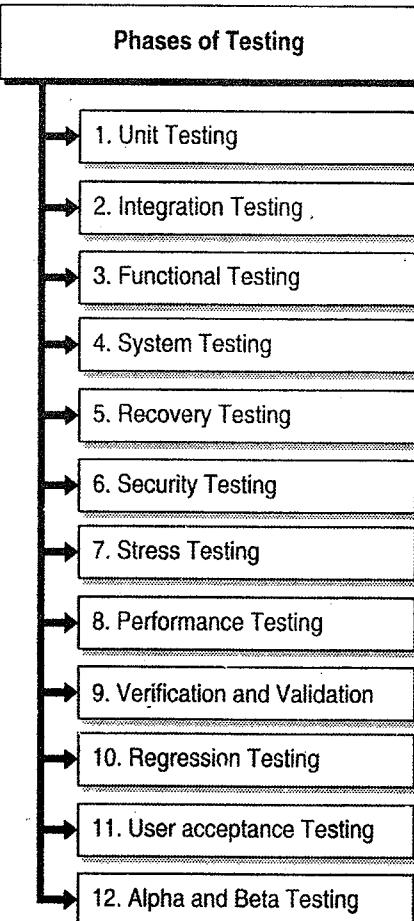


Fig. C13.1

Incremental Integration Strategies

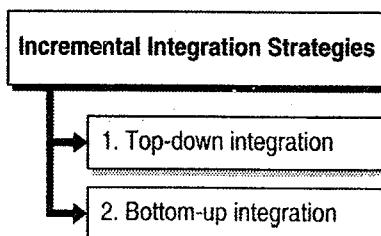


Fig. C13.2

Types of Testing

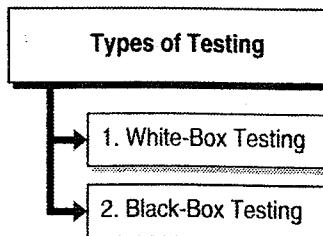


Fig. C13.3

☞ White-Box Testing Methods

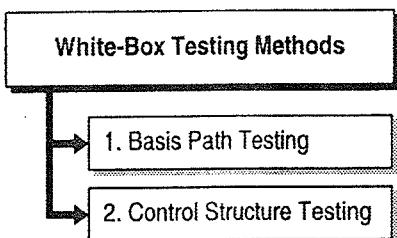


Fig. C13.4

☞ Basis Path Testing Examples

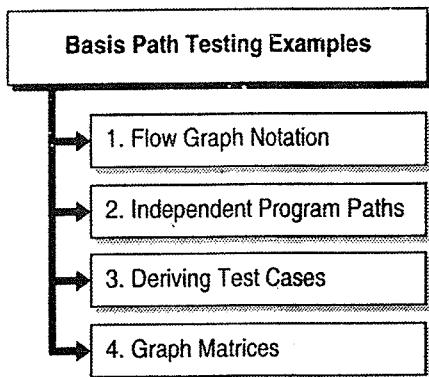


Fig. C13.5

☞ Control Structure Testing Examples

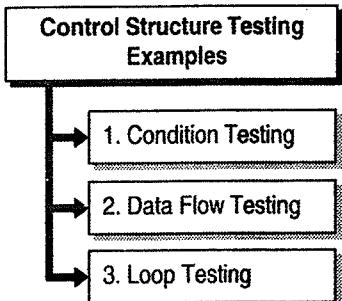


Fig. C13.6

☞ Classes of Loops

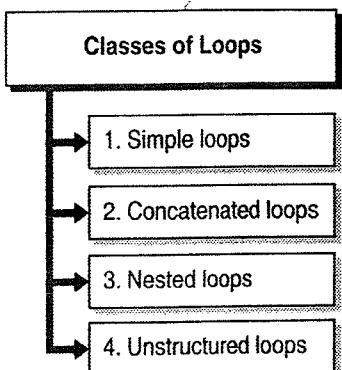


Fig. C13.7

☞ Black-Box Testing Methods

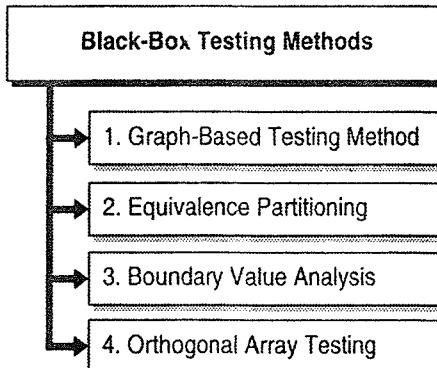


Fig. C13.8

☞ Types of Tests

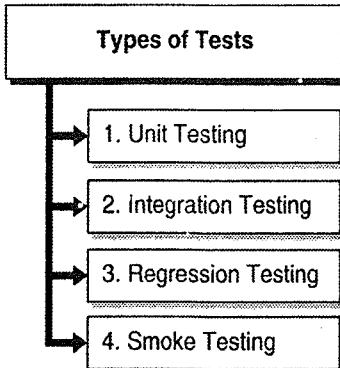


Fig. C13.9

☞ Incremental Integration Strategies

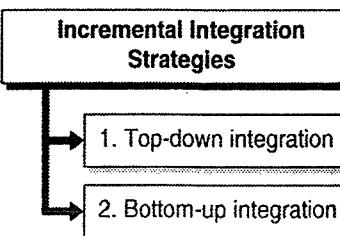


Fig. C13.10

Defect Management Process

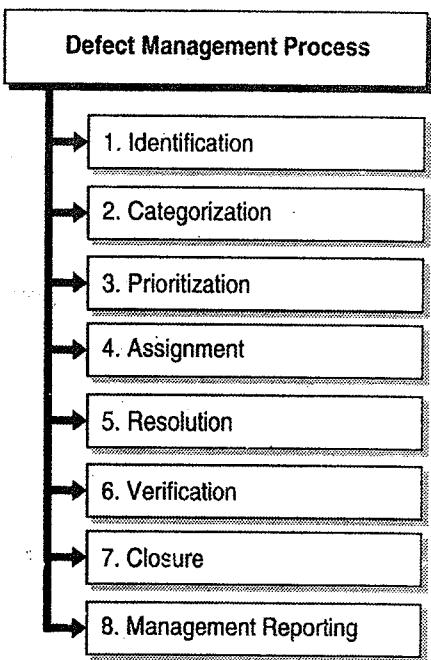


Fig. C13.11

Debugging Approaches Categories

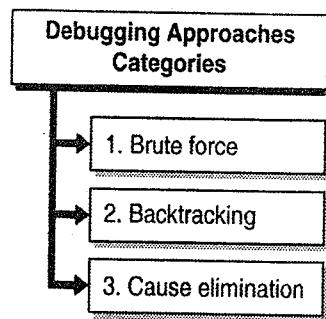


Fig. C13.12

Fully Solved University Question Papers

- Aug. 2017 (In Sem)**
- Dec. 2017**

August 2017 (In Sem.)

- Q.1(a)** Define terms 'Software' and 'Software Engineering'. "Software does not wear out". State whether this statement is true or false justify your answer.

(Ans. : Refer sections 1.1, 1.3, 1.3.5) (3 Marks)

Ans. :

- Computer software is actually a product developed by software engineers by making use of various software engineering processes and activities.
- Software engineering is a systematic and disciplined approach towards developments of software. The approach must be systematic for operation of the software and the maintenance of it too.

Software does not wear out.

- Note that, wear out means process of losing the material.
- Hardware components are affected by dust, temperature and other environmental maladies. Stated simply, hardware can wear out. However software does not influenced by such environmental means.
- When hardware component wear out, it can be replaced by spare part. But there are no spare parts for software. Any software error indicates error in design or coding of that software.
- Hence software maintenance involves more complexity than hardware maintenance.
- Mostly software is custom built rather than assembled from existing components.
- Computer hardware can be assembled from existing components as per our requirements. But that is not the case for software. Software is developed according to customer's need.

- Q. 1(b)** What are the customer myths ? Discuss the reality of these myths.

(Ans. : Refer section 1.6.2) (3 Marks)

Ans. :

Myth

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

Reality

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

Myth

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

Reality

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

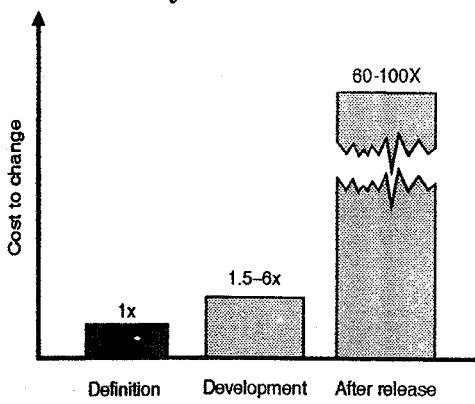


Fig. 1-Q.1(b) : The impact of change

- Q. 1(c)** Explain the following terms :

(i) Refactoring



- (ii) Pair programming
- (iii) Spike solution
- (iv) CRC

(Ans. : Refer sections 3.6.1 and 3.6.2)

(4 Marks)

Ans. :

(i) Refactoring

- It is one of the agile practices in agile software development.
- It is process of restructuring the existing code.
- The changes are made without affecting external behaviour.
- It does not change the functional requirement of the code.
- It changes the non-functional requirement of the code. Non functional requirements are useful in efficiency, performance, throughput, etc.

(ii) Pair programming

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

(iii) Spike Solution

Period used to research concept.

(iv) CRC

- A Class Responsibility Collaborator (CRC) model (Beck & Cunningham 1989; Wilkinson 1995; Ambler 1995) is a collection of standard index cards that have been divided into three sections.
- A class represents a collection of similar objects, a responsibility is something that a class knows or does, and a collaborator is another class that a class interacts with to fulfill its responsibilities.

Q. 2(a) Define Software Engineering. How software engineering is different from Hardware Engineering? Justify.

(Ans. : Refer section 1.3.5) (2 Marks)

Ans. :

- Software engineering is a systematic and disciplined approach towards developments of software. The approach must be systematic for operation of the software and the maintenance of it too.
- Software is having some characteristics, those are totally different from hardware characteristics or the industrial manufacturing.
- Software is developed or engineered and it is not manufactured like other hardware products.
- There exist some similarities between development of software and manufacturing of hardware.
- In both the cases quality is achieved by good design but manufacturing phase of hardware can introduce quality problems that are absent in software.
- Both activities depend on people but relationship between people applied and work done is different in both the cases.
- Both the cases require the 'construction of product', but approaches are different.
- Software does not wear out.



- o Note that, wear out means process of losing the material.
- o Hardware components are affected by dust, temperature and other environmental maladies. Stated simply, hardware can wear out. However software does not influenced by such environmental means.
- o When hardware component wear out, it can be replaced by spare part. But there are no spare parts for software. Any software error indicates error in design or coding of that software.
- o Hence software maintenance involves more complexity than hardware maintenance.
- o Mostly software is custom built rather than assembled from existing components.
- o Computer hardware can be assembled from existing components as per our requirements. But that is not the case for software. Software is developed according to customer's need.

Q. 2(b) Explain different umbrella activities in Software process framework.

(Ans. : Refer section 1.5.1) (4 Marks)

Ans. :

The framework described in generic view of software engineering is complemented by number of Umbrella activities. Typical **umbrella activities** are :

1. Software project tracking and control

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

2. Risk management

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

3. Software Quality Assurance (SQA)

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

4. Formal Technical Reviews (FTR)

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

5. Measurement

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

6. Software Configuration Management (SCM)

It manages the effects of change throughout the software process.

7. Reusability management

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

8. Work product preparation and production

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

Q. 2(c) Describe Agile Manifesto.

(Ans. : Refer section 3.2.2) (4 Marks)

Ans. :

- The Agile Manifesto states that :

"We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value,

- o **Individuals and interactions** work over processes and tools
- o **Working software** takes responsibility of comprehensive documentation
- o **Customer collaboration** look after contract negotiation
- o **Responding to change** after having a good plan."

- Even though these agile methods are based on the concept of incremental development, quicker delivery and faster deployment, the agile manifesto gives different processes to achieve this.
- But most of the set of principles used by various experts that are based on agile manifesto are very much common.

Q. 3(a) What is the difference between requirement inception and requirement elicitation? Why requirement elicitation difficult?

(Ans. : Refer sections 4.2.1 and 4.2.2)

(3 Marks)

Ans. :

- **Inception** is beginning and it is usually said that, 'well beginning is half done'. But it is always problematic for the developer that what should be the starting point i.e. 'from where to start'.
- **Elicitation** means, 'to draw out the truth or reply from anybody'. In relation with requirements engineering, elicitation is a task that helps the customer to define what is required.

- To know the objectives of the system or the project to be developed is a critical job. Why it is difficult to get a clear understanding of customer wants? To answer this question, we have a number of problems like :

Problems of scope

Many times customer states unnecessary technical details. The unnecessary details may confuse developer instead of giving clarity of overall system objectives.

Problems of understanding

Sometimes both customer as well as developer has poor understanding of :

- What is needed ?
- Capabilities and limitations of the computing environment.
- Understanding of problem domain.
- Specify requirements those conflict with other customers and users.

Problems of volatility

Volatility means 'change from one state to another'. The customer's requirements may change time to time. This is also a major problem while deciding fixed set of requirements.

Q. 3(b) What are functional and non-functional requirements of Software ?

(Ans. : Refer sections 4.4.1 and 4.4.2)

(3 Marks)

Ans. :

Functional Requirements

- Actually functional requirements state that how a system should behave and to work in a given situation.
- The functional requirements are often depended on the following parameters :
 - o Type of the software under construction,
 - o The users of the software,

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

Non-functional Requirements

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

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

Q. 3(c) Explain the structured SRS with case study of Insulin Pump.

(Ans. : Refer section 4.7.6) **(4 Marks)**

Ans. :

- Structured natural language is a way of writing system requirements where the freedom of the requirements writer is limited and all requirements are written in a standard way.
- This approach maintains most of the expressiveness and understandability of natural language but ensures that some uniformity is imposed on the specification. Structured language notations use templates to specify system requirements.
- The specification may use programming language constructs to show alternatives and iteration, and may highlight key elements using shading or different fonts.
- The Robertsons, a well known author, recommend that user requirements be initially written on cards, one requirement per card.
- They suggest a number of fields on each card, such as the requirements rationale, the dependencies on other requirements, the source of the requirements, supporting materials, and so on.
- This is similar to the approach used in the example of a structured specification shown in Table 1-Q. 3(c).



Table 1-Q. 3(c) : A structured specification of a requirement for an insulin pump

Function	Compute insulin dose: Safe sugar level.
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.
Inputs	Current sugar reading (r_2), the previous two readings (r_0 and r_1).
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose : the dose in insulin to be delivered.
Destination	Main control loop.
Action	CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.
Requirements	Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin.
Post-condition	r_0 is replaced by r_1 then r_1 is replaced by r_2 .
Side effects	None.

- To use a structured approach to specifying system requirements, you define one or more standard templates for requirements and represent these templates as structured forms.
- The specification may be structured around the objects manipulated by the system, the functions performed by the system, or the events processed by the system.
- An example of a form-based specification, in this case, one that defines how to calculate the dose of insulin to be delivered when the blood sugar is within a safe band, is shown in Table 1-Q. 3(c).

Q. 4(a) Explain the importance of requirements engineering and list the tasks involved.

(Ans. : Refer section 4.1) (3 Marks)

Ans. :

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

Q.4 (b) What are the characteristics of good SRS ?

(Ans. : Refer section 4.7.5) (3 Marks)

Ans. :

- **Complete** : All the project functionalities should be recorded by SRS.
- **Consistent** : SRS should be consistent
- **Accurate** : SRS defines systems functionality in real world. We need to record requirement very carefully.
- **Modifiable** : Logical and hierarchical modification should be allowed in SRS.



- **Ranked** : Requirement should be ranked using different factors like stability, security, ease or difficulty.
- **Testable** : The requirement should be realistic and able to implement.
- **Traceable** : Every requirement we must be able to uniquely identify.
- **Unambiguous** : Statement in the SRS document should have only one meaning.
- **Valid** : All the requirements should be valid.

Q. 4(c) State and explain the different methods of requirement elicitation.

(Ans. : Refer section 4.8.5) (4 Marks)

Ans. :

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

Following are different methods of requirement elicitation :

1) Prototyping

- Prototype is the representations or visualizations of the actual system parts.
- The prototype is designed in the early stages of the implementation of the project. It provides the General idea of the actual system functions and the work flow.
- Prototyping is used to gather the requirements from the users by presenting GUI based system functions.

2) Requirements reuse

- In the field of software engineering reusing the requirements of the existing system is common method of requirements elicitation.
- Using the existing Knowledge to develop the new product has many advantages that include low cost and less time.
- Though each product has their own type of stakeholders and users, there is still number of situations that the reusing of the requirements takes place

3) Scenarios

- The scenarios of the particular system will give the working method of different interaction sessions or the situations of the system.
- These scenarios are helpful for requirements elicitation in two ways :
 - (1) Analyzing the different sessions of the system gives the flexibility to find the requirements.
 - (2) The user response after interaction with the scenarios will give the flexibility to find the requirements.

4) Brainstorming

- Brainstorming is a techniques used to generate new ideas and find the solution to a specific issue. This is conducted as a conference with six to ten members.
- The members are from the different departments and domain experts are also included.
- This conference is headed by the organizer, who states the issue to be discussed. The conference is generally held in a round table fashion. Every member person is allotted with certain time interval to explain their ideas.

Q.5 (a) Explain the different Design concepts.

(Ans. : Refer section 5.4 to 5.4.9) (3 Marks)



Ans. :

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

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

Abstraction

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

Architecture

- Software architecture means the complete structure of the software. In number of ways, this structure provides basis for conceptual integrity for a system.

Patterns

- A pattern is a thing, which conveys the essence of a proven solution to a recurring problem within a certain context.

Modularity

- The modularity consists of software architecture and design patterns i.e. software is divided separately according to name and addresses of components and sometimes it is called as modules that are integrated to fulfil problem requirements.

Information Hiding

- The modules of the larger problem must be specified and designed properly so that information (i.e. algorithms and data) present within a module is not available to other modules that do not require such information.

Q.5(b) Enlist and explain the Webapp design principles in detail.

(Ans. : Refer section 7.7.1) **(3 Marks)**

Ans. :

Following are some WebApp design principles :

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



Q.5 (c) Define following design concepts.

- (i) Patterns
- (ii) Information hiding
- (iii) Architecture
- (iv) Refinement

(Ans. : Refer sections 5.4.3, 5.4.5, 5.4.2, 5.4.7) (4 Marks)

Ans. :

(i) Patterns

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

(ii) Information Hiding

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

(iii) Architecture

- Software architecture means the complete structure of the software. In number of ways, this structure provides basis for conceptual integrity for a system.
- In its simplest form, the architecture is the complete structure or arrangement of the program components in such a way that they interact with each other. The data structures may be used by these components. All these components are the

building blocks of the overall structure of the application being developed.

- One goal of software design is to derive an architectural framework of a system. The architectural design can be represented using one or more of a number of different models.

(iv) Refinement

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

Q.6 (a) What do you mean by the term cohesion and coupling in the context of Software Design? How are these concepts useful in arriving at a good design of a system?

(Ans. : Refer section 5.4.6) (3 Marks)

Ans. :

Cohesion

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

- Cohesion can be represented as a "spectrum". High cohesion is always recommended but mid-range of the spectrum is often acceptable. The cohesion scale is nonlinear.

That is, low-end cohesiveness is worse than middle range, which is nearly same high-end cohesion.

In a regular practice, designers are not required to be bothered about categorizing

cohesion in a particular module. Rather, the overall concept should be understood properly and low-end cohesion must be avoided when modules are designed.

- At the low-end of the spectrum which is actually not desirable, we encounter a module that performs different tasks that are loosely related to each other. These modules are termed as coincidentally cohesive.
- A module that performs tasks that are related logically is logically cohesive.
- When a module contains tasks that are related by the fact that all must be executed with the same span of time, the module exhibits temporal cohesion.
- Moderate levels of cohesion are close to each other in the degree of module independence. Procedural cohesion exists when processing elements of a module are related and are executed in a specific order. When all the processing elements do concentrate on only one area of a data structure, then communicational cohesion is available. High-end cohesion is characterized by a module that performs one single and unique procedural task.

2. Coupling

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

ripple effect is introduced when the errors occurring at one particular location in the system and propagating through a system.

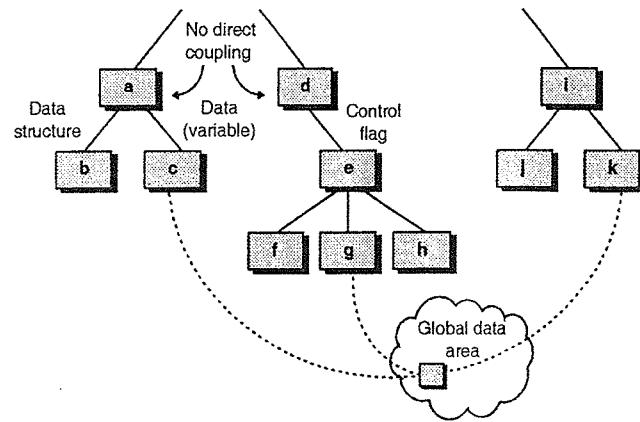


Fig. 1-Q. 6(a) : Types of coupling

- In the Fig. 1-Q. 6(a), various examples of different types of module coupling are illustrated:
 - o Modules **a** and **d** are children of different modules. Each of them is not related to each other and thus there is no direct coupling occurs.
 - o Module **c** is child of module **a**. It is accessed through a conventional argument list. All the data are passed through this argument list.
- When the argument list is simple, the data can be passed and there is one to one correspondence between the items exist. The low data coupling or simply low coupling is demonstrated through this structure.
- When a part of data structure apart from simple argument, is passed through a module interface, then stamp coupling is exhibited. This is actually a variation of data coupling only. It is illustrated in the Fig. 1-Q. 6(a) and it occurs between **b** and **a**.
- The coupling is characterized by passage of control between modules at moderate

- levels. In most of the software designs, control coupling is common and is exhibited in Fig. 1-Q. 6(a) where a "control flag" is passed between modules **d** and **e**.
- When modules are tied to some environments that are external to software, then the high levels of coupling may occur. Like an I/O can couple a module to some devices, formats, and different communication protocols. This is external coupling and is very essential. It must be limited to only a small number of modules in a structure.
 - Like low coupling, high coupling may also occurs when different modules reference a global data area. This is called common coupling also, and is shown in Fig. 1-Q. 6(a) modules **c**, **g**, and **k** share their data item in a global data area. The module **c** initializes the items.
 - After that the module **g** recomputes and later on updates these items. Now assume that an error has been occurred and module **g** updates the item incorrectly.
 - In processing the module, **k** reads the wrong item and during its processing, it fails and ultimately the software is aborted.
 - The reason behind this abort is module **k** and the actual cause is module **g** since it has updated the item incorrectly.
 - Diagnosing the problems is time consuming and difficult. The diagnosis in structures is always with the common coupling. The use of global data is not always bad, but the developer may use them in coupling. The designer should know well the impacts of these couplings and care against them to protect.
 - The coupling occurs when a module uses the data present in some another module. It can use control information also during the coupling.
- The concept of content coupling used when all the subroutines or data structures within a module are compulsory asked to involve in coupling. The content coupling must be avoided
 - The content coupling modes may occur due to design decisions made during development of structure. The variants of the coupling discussed, may be introduced during coding phase.
 - **For example :** Compiler coupling ties source code to specific (and often nonstandard) attributes of a compiler; Operating System (OS) coupling ties design and resultant code to operating system "hooks" that can create havoc when OS changes occur.
- Q.6 (b) What is Architecture? Explain the architecture context diagram.**
- (Ans. : Refer sections 5.4.2 and 6.1)(3 Marks)*
- Ans. :**
- ### Architecture
- Software architecture means the complete structure of the software. In number of ways, this structure provides basis for conceptual integrity for a system.
 - In its simplest form, the architecture is the complete structure or arrangement of the program components in such a way that they interact with each other. The data structures may be used by these components. All these components are the building blocks of the overall structure of the application being developed.
 - One goal of software design is to derive an architectural framework of a system. The architectural design can be represented using one or more of a number of different models.
 - Architectural design is backbone of any software system design and it is responsible for the overall structure of the system.

- In nearly all the models of software process, architectural design is considered as the first stage in the software design and development process.
- The output of the architectural design process is an architectural model that explains the overall structure of the system and also explains the working of the system and the communication among various components of the system.
- In case of agile development processes also, the first stage of development process is defining the system architecture. Incremental approach of the software development is not generally successful whereas the concept of refactoring of the components bit easy. But refactoring of the system architecture is expected to be expensive.
- In order to understand the system architecture, Fig. 1-Q.6(b) provides a simple model of architecture for a remote control car :

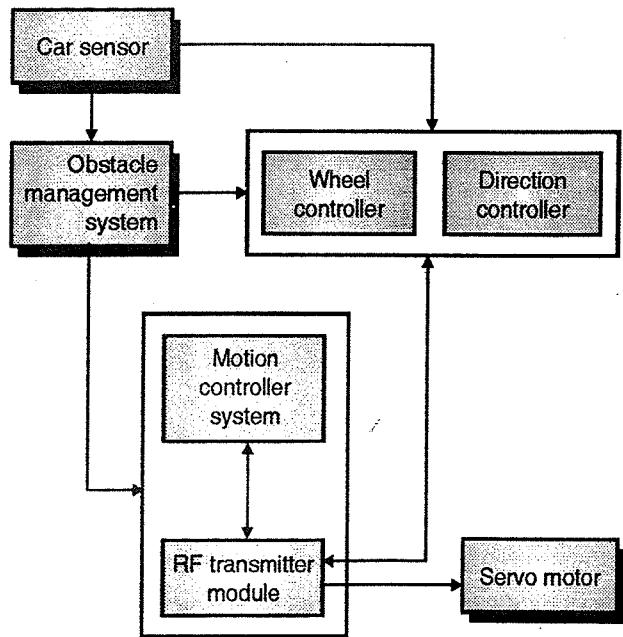


Fig. 1-Q. 6(b) : The architecture of a remote control car

- The Fig. 1-Q.6(b) exhibits the abstract model of architecture of a remote control car system that will help to understand the functioning of the car.

- In the Fig. 1-Q.6(b) shown, the interconnection between the components used to develop the remote control car system.
- It uses RF transmitter module to actually control the complete system. The sensor present in the car acts as RF receiver module which accepts the commands sent from the RF transmitter module.
- Various controllers listed below are actually used to control the functioning of the remote control car system :
 - o Wheel controller
 - o Direction controller
 - o Motion controller, and
 - o Obstacle management system that controls the obstacle and changes the direction to avoid the collision.
- The most important part of the remote control car system is "Servo Motor" that is responsible for overall functioning of the system.
- In actual practice, there is some similarities and overlap among the requirement engineering process and the system specification. The architectural design and system specification should avoid including the design information.
- Instead architectural decomposition is essential to organize the system specification and to provide the structure of the system.
- Therefore, as part of the requirements engineering process, the abstract system model is proposed so that various functions and feature is shown clearly. Therefore, the decomposition is used to show the requirements and features of the system with stakeholder (e.g. customer, developer, designer or the end users).



- Normally software architecture can be designed into two levels of abstraction as follows :
 - o Small software design architecture
 - o Large software design architecture
- **Architecture in the small** is related to the architecture of individual programs which may be small in size. This program is decomposed into smaller components. This type of architecture is related to mostly program architectures.
- At this level, we are concerned with the way that an individual program.
- **Architecture in the large** is related to architecture of complex systems that includes overall system, the entire program and the components of the program.
- **Software architecture** is an important characteristic since it is responsible for various features like performance, robustness, distributability, and maintainability of a system. The distributability feature is mainly the part of distributed systems over the networked lines.
- The non-functional system requirements are totally depend on the system architecture whereas functional system requirements are dependent on the individual components.
- Following are some of the advantages, if a software designer properly design, document and implement the software architecture :
 - o **Communication among various stakeholders** : Actually any architecture is considered as the highest level of abstraction and presentation of the system is important and it can be focused by discussion among various stakeholders of the system.
 - o **System analysis** : If we want to propose system architecture explicitly in the beginning only, then software development requires more analysis.
 - o Mostly the system architectural design decisions have greater impact on achieving the critical requirements like performance, reliability, and maintainability.
 - o **Reusability at large-scale** : A model or the system architecture is considered as compact and manageable description of the system. For all the systems having similar requirements uses same system architecture and this is the reason why it can support reusability at large scale.

Q. 6(c) Enlist the golden rules for User Interface Design.

(Ans. : Refer sections 7.2 to 7.2.4) (4 Marks)

Ans. :

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

Place the user in Control

- The main motive behind using interface constraints and restrictions are to simplify the mode of interaction.
- In most of the cases, the developer may introduce constraints and limitations to simplify the implementation of the interface.
- The ultimate result could be an interface that is easy to develop, but frustrating to use.



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

Reduce the User's Memory Load

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

- Reduce the demand for short-term memory

- o When users are performing a complex task, then the demand of short-term memory might be significant.
- o The user interface must be designed in a way that reduces the requirement to remember its past actions and results. i.e. it should be able to reduce the short-term memory use.
- o This feature is achieved by providing visual clues that will help a user to remember past actions, rather than a need to recall them.

- Establish meaningful defaults

- o The default values will be an added advantage especially for average users in the start of application. But the advanced user should have a provision to set their default individual preferences.
- o There should be reset option i.e. factory reset option available in mobile phones. So that user can once again obtain the default values.

- Define shortcuts that can easily remembered

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

It is easy to remember this shortcut since P is the first letter of print command

- The interface screens must be analogous to a real world object

A bill payment system should use a checkbook and check register metaphor to help its users to complete their transactions.

The information should be disclosed in a progressive fashion

- o The interface must be designed in such a way that it gives bubble tips about a task, an object at a high level of abstraction.
- o Then the detailed information should be given when user shows interest in that i.e. when user clicks on that bubble help icon.

Make the Interface Consistent

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

- The system should allow its users to put the ongoing task into a meaningful context

- o Most of the interfaces implement complex layers of interactions with dozens of screen images.
- o It is important to provide indicators like window titles, consistent colour codes and graphical icons that helps the user to know the context of the work at hand.
- o In addition, the user should be able to determine where he has come from and what alternatives exist for a transition to a new task.

The system should maintain the consistency within a family of applications

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

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

Dec. 2017

- Q.1(a)** What is objective of Personal Software Process(PSP)? What are the activities of PSP model?

(Ans. : Refer section 1.5 to 1.5.1) (5 Marks)

Ans. :

- A software process can be illustrated by the Fig. 1-Q.1(a). In the Fig. 1-Q.1(a), a common process framework is exhibited. This framework is established by dividing overall framework activities into small number of framework activities.

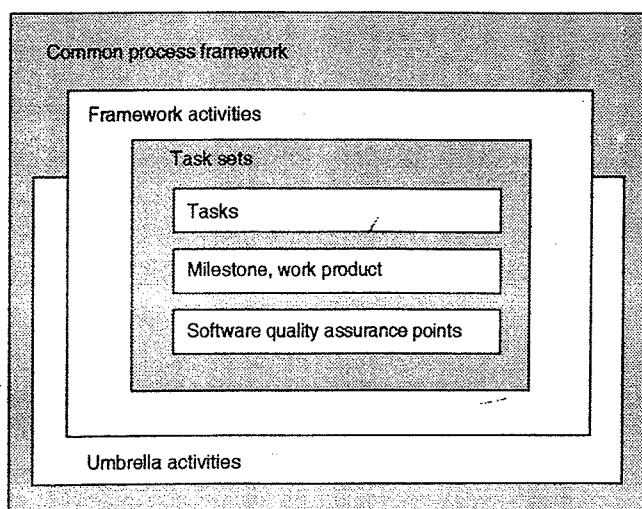


Fig. 1-Q. 1(a) : A software process

- And these small activities are applicable to the entire project irrespective of its size and complexity. A framework is a collection of task sets and these task sets consists of :

- o Collection of small work tasks
- o Project milestones, deliverable i.e. actual work product and
- o Software quality assurance points

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

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

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

Umbrella Activities

The framework described in generic view of software engineering (Fig. 1-Q.1(a) : A software process) is complemented by number of Umbrella activities. Typical **umbrella activities** are :

1. Software project tracking and control

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

**2. Risk management**

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

3. Software Quality Assurance (SQA)

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

4. Formal Technical Reviews (FTR)

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

5. Measurement

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

6. Software Configuration Management (SCM)

It manages the effects of change throughout the software process.

7. Reusability management

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

8. Work product preparation and production

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

Q.1(b) What is agility ? Explain about agility process model.

(Ans. : Refer section 3.1) (5 Marks)

Ans. :

Agility

- Agility means effective (rapid and adaptive) response to change, effective communication among all stockholder. Drawing the customer onto team and organizing a team so that it is in control of work performed.
- The Agile process, 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.
- An agile process model includes the concept of development along with a set of guidelines necessary for the development process. The conceptual design is necessary for the satisfaction of the customer. The concept is also necessary for the incremental delivery of the product. The concept or the philosophy makes



development task simple. The agility team must be highly motivated, updated with latest skill sets and should focus on development simplicity.

- We can say that agile development is an alternative approach to the conventional development in certain projects.
- The development guidelines emphasize on analysis and design activities and continuous communication between developers and customers. An agile team quickly responds to changes. The changes may be in development, changes in the team members, and changes due to new technology. The agility can be applied to any software process. It emphasizes rapid delivery of operational software.
- All the agile software processes should address three important assumptions :
 - o Difficult to predict in advance software requirements
 - o Design and construction are interleaved in most of the projects, it is difficult to predict design before construction.
 - o Analysis, design, construction and testing are not much predictable.
- To address these assumptions of unpredictability, the agile development process must be adaptable. So an agile process must adapt incrementally.

Q.2(a) What are the Practitioner's myths? Discuss the reality of three myths.

(Ans. : Refer section 1.6.3)

(5 Marks)

Ans. :

Myth

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

Reality

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

Myth

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

Reality

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

Myth

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

Reality

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

Myth

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

Reality

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



Q.2(b) What are requirements engineering tasks ? Explain in detail.

(Ans. : Refer section 4.1) (5 Marks)

Ans. :

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

Q.3(a) What is meant by feasibility study? Give general process models of the requirement elicitation and analysis process.

(Ans. : Refer sections 4.8 to 4.9) (5 Marks)

Ans. :

- **Feasibility study** : To check whether the system is useful for the business.
- Elicitation is a task that helps the customer to define what is required. 'Eliciting requirements' step is carried out by series of following steps :
 1. Collaborative requirements gathering
 2. Quality function deployment

3. User scenarios

4. Elicitation work product

Collaborative Requirements Gathering

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

Quality Function Deployment

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



1. Normal requirements

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

For example

- o Graphic displays.
- o Specific system functions.
- o Specified output formats.

2. Expected requirements

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

For example

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

3. Exciting requirements

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

- **Usage Scenarios :** It is just impossible to move into more technical software engineering activities until the software team understands how these functions and features will be used by different classes and end users.
- To understand this, the developer and users create a set of scenarios those will

identify all these issues. The scenario is called as 'Use Cases'.

4. Elicitation Workproduct

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

Elicitation Techniques

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

Developing Use Cases

- A Use case exhibits the behaviour of the system according to the response received from any of the stakeholders. Alternatively a use case explains how the users will operate the system under any specific circumstances.

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

For example

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

The software for control machine requires four different modes for interaction : Programming mode, test mode, monitoring mode, troubleshooting mode.

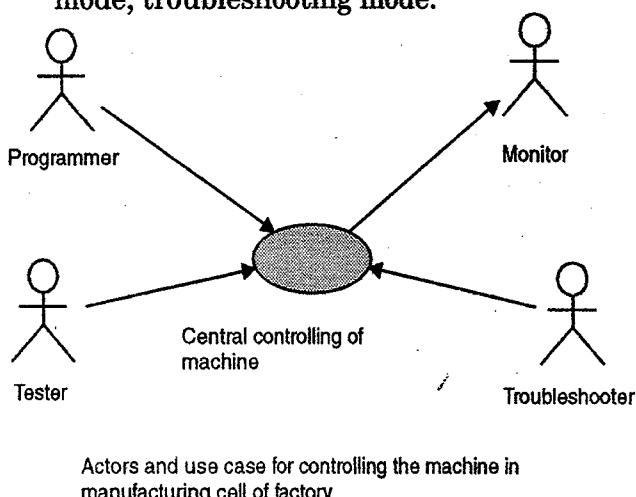


Fig. 1-Q. 3(a)

- Hence four actors can be defined : programmer, tester, monitor and trouble shooter.
- There are two types of actors :
 1. Primary actors
 2. Secondary actors

1. Primary actors

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

2. Secondary actors

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

Requirements Analysis

- Analysis model uses a combination of text and diagrammatic forms to depict requirements of data, functions and behaviour. So that it will be easy to

understand overall requirements of the software to be built.

- The 'Software Engineer' also called as 'Analyst' builds the model using requirements stated by the customer.
- Analysis model validates the software requirements and represent the requirements in multiple dimensions.

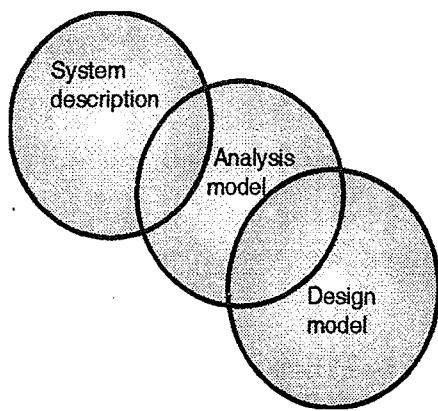


Fig. 2-Q. 3(a) : Analysis model, bridge in system description and design model

- Basic aim of analysis modelling is to create the model that represents the information, functions and behaviour of the system to be built.
- These information, functions and behaviour of the system are translated into architectural, interface and component level designs in design modeling.
- Information, functions and behaviour of the system are represented using number of different diagrammatic formats.
- As stated previously, the 'analysis model' acts as a bridge between 'system description' and the 'design model'.
- The system description describes overall system functionality of the system including software, hardware, databases, human interfaces and other system elements. And the software design mainly focuses on application architectural, user interface and component level designs.
- Thus, all elements of analysis model are directly traceable to the parts of design

model. Hence, the analysis model must have following three objectives :

- o To state clearly what customer wants exactly.
- o To establish the basis of the 'design model'. The information, functions and behaviour of the system defined by 'analysis model' are translated into architectural, interface and component level designs in 'design modeling'.
- o To bridge the gap between a system level description and software design.

Q.3(b) Explain layered architecture style with neat diagrams.

(Ans. : Refer sections 1.3 to 1.3.4) (5 Marks)

Ans. :

- The term software engineering is defined as :
- "By using the principles of sound engineering and its establishment, software is developed that should be economical and should work efficiently on real machines."
- Software engineering is a systematic and disciplined approach towards developments of software. The approach must be systematic for operation of the software and the maintenance of it too.
- Software engineering is considered as a layered technology. These layers includes :
 1. Quality focus
 2. Process
 3. Methods
 4. Tools

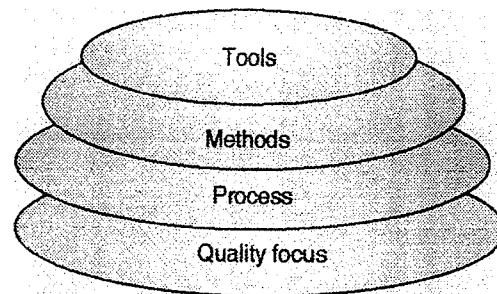


Fig. 1-Q. 3(b) : Software engineering layers

Quality Focus

Quality is nothing but 'degree of goodness'. Software quality cannot be measured directly. Good quality software has following characteristics :

1. **Correctness** is degree to which software performs its required function.
2. **Maintainability** is an ease with which software is maintained. Maintenance of software implies change in software. If software is easy to maintain, then the quality of that software is supposed to be good.
3. **Integrity** is a security provided so that unauthorized user can not access data or information. e.g. password authentication.
4. **Usability** is the efforts required to use or operate the software.

Process

Software process defines a framework that includes different activities and tasks. In short, process defines following 'what' activities and tasks for software development :

1. What activities are to be carried out ?
2. What actions will be taken ?
3. What tasks are to be carried out in a given action ?

Methods

Method provides technical way to implement the software i.e. 'how to' implement. Methods consist of collection of tasks that include :

1. **Communication** : Between customer and developer.
2. **Requirement analysis** : To state requirements in detail.

3. **Analysis and design modelling** : To build a prototyping model of software to exhibit the requirements clearly.
4. **Program construction** : Implementation of requirements using conventional programming languages or automated tools.
5. **Testing and support** : Test for errors and expected results.

Tools

Software tool is an automated support for software development.

For example

1. Microsoft front page or Microsoft Publisher can be used as web designing tool.
2. Rational Rose can be used as object oriented analysis and design tool.

Q.4 (a) Explain guidelines of component level design.

(Ans. : Refer section 6.6) (5 Marks)

Ans. :

- Component level design comes after architectural design is completed. It is possible to represent the component level design by using some programming languages these programs can be created by using the architectural design model.
- In fact component level design is an alternative approach to the architectural design approach.

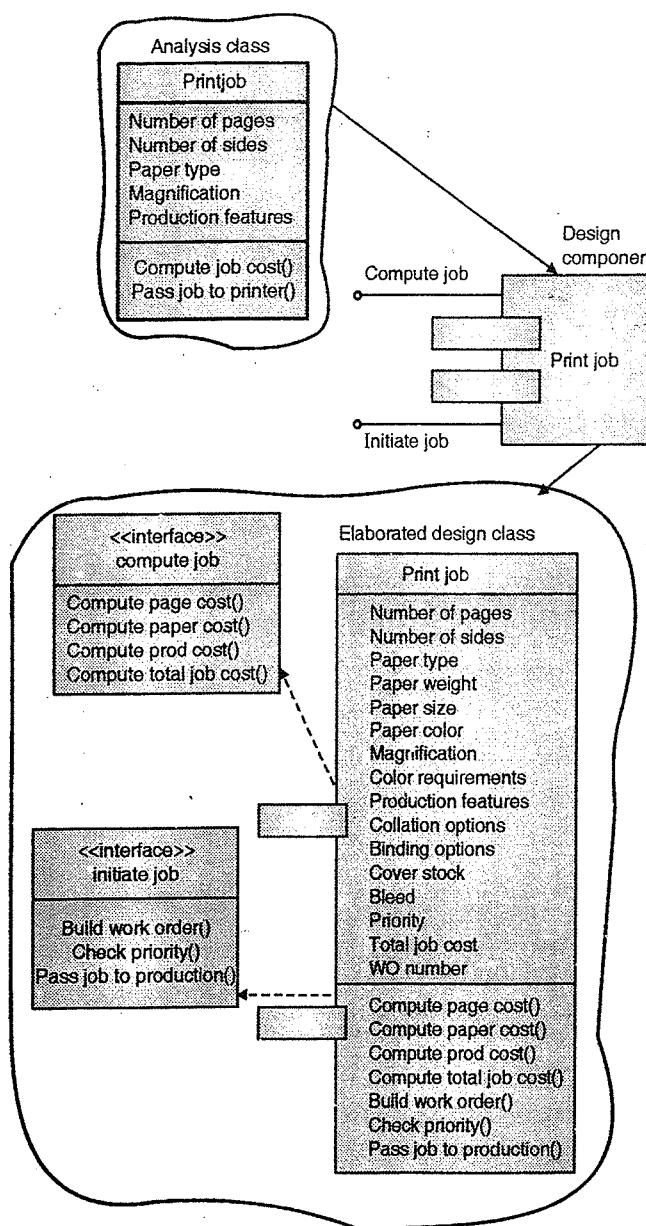


Fig. 1-Q. 4(a)

- A component is the basic building block of a software system, it refers the OMG specifications and is defined as a modular and deployable part of a computer system and it consists of different sets of interfaces.
- Following are different views of a component design :
 1. An object oriented view
 2. The conventional view
 3. The process related view

1. An object oriented view

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

2. The conventional view

- In conventional method for software engineering, a component consist of a functional element of a program or a module that has processing logic, data structures required to implement the interfaces between the components.
- To explain the process of design for conventional components, consider a sophisticated photo copying of analysis modeling which in architectural design.
- In component level design each subsystem is illustrated in the Fig. 1-Q. 4(a).
- The data flow and control object and their interfaces are illustrated very well in the diagram. The data structure is also explained properly and the algorithm allows to complete its intended function using stepwise refinement approach.



3. The process-related view

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

Q.4(b) Explain the user interface design principles.

(Ans. : Refer section 7.2 to 7.2.3) (5 Marks)

Ans. :

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

Place the user in Control

- The main motive behind using interface constraints and restrictions are to simplify the mode of interaction.
- In most of the cases, the developer may introduce constraints and limitations to simplify the implementation of the interface.
- The ultimate result could be an interface that is easy to develop, but frustrating to use.
- There are a number of design principles that allow the user to maintain control.
- **Define the interaction modes in such a way that the user is not forced to perform unnecessary or undesired actions :**

o Consider the example of word processor for spell check. If user selects spell check in its current interaction, then a word-processor menu should move it to a spell-checking mode.

- o The software should not force the user to remain in spell-checking mode if the user wishes to make some small text edit along the way. There should be less effort in entering and exiting spell check mode.

The interaction should be flexible

- o For different users, different interaction preferences must be provided. For example, the software should allow a user to interact with the system with different input devices like keyboard, mouse, a digitizer pen, or voice recognition commands.
- o It is not necessary that all the actions are supported by all means of input devices, but some actions supports only specific mechanism only. Consider drawing a shape using keyboard will be too difficult, but mouse will be a good option.

User interaction should be interruptible and undoable

- o Even for a long sequence of actions, the user must be able to interrupt the sequence and can perform some another task without losing any data or work.
- o There must be undo options available in the software.

The different levels of difficulty in interaction for different classes of users must be customized

- o The users find that they perform some sequence of interactions repeatedly. Instead of writing the same sequence of

interaction again and again, it is always recommended to use a macro mechanism.

- The software should provide such a mechanism to facilitate interaction.

- **Hide complexities from the casual user**

- The main benefit of the user interface is that it takes its user to a virtual world while interacting. The software must support this feature.
- The design complexities like use of operating system, functions in file management, or other hidden complexities in computing technology.
- The interface should never require a user to interact at a level which is inside the machine. For example, the user never asked to type operating system commands in the application software.

- **The direct interaction with objects that appear on the screen**

- The user should have feel of using an object like a physical object. i.e. a design must be in such a way that user is able to perform all the function and do manipulation as he do in a physical object or physical thing.
- For example, an application interface allows its users to stretch or resize an object as it is a physical object i.e. scaling should be a direct implementation of direct manipulation.

Reduce the User's Memory Load

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

- **Reduce the demand for short-term memory**

- When users are performing a complex task, then the demand of short-term memory might be significant.
- The user interface must be designed in a way that reduces the requirement to remember its past actions and results. i.e. it should be able to reduce the short-term memory use.
- This feature is achieved by providing visual clues that will help a user to remember past actions, rather than a need to recall them.

- **Establish meaningful defaults**

- The defaults values will be an added advantage especially for average users in the start of application. But the advanced user should have a provision to set their default individual preferences.
- There should be reset option i.e. factory reset option available in mobile phones. So that user can once again obtain the default values.

- **Define shortcuts that can easily remembered**

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

It is easy to remember this shortcut since P is the first letter of print command

- **The interface screens must be analogous to a real world object**

A bill payment system should use a checkbook and check register metaphor to help its users to complete their transactions.

- **The information should be disclosed in a progressive fashion**

- The interface must be designed in such a way that it gives bubble tips about a task, an object at a high level of abstraction.
- Then the detailed information should be given when user shows interest in that i.e. when user clicks on that bubble help icon.

Make the Interface Consistent

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

- **The system should allow its users to put the ongoing task into a meaningful context**

- Most of the interfaces implement complex layers of interactions with dozens of screen images.
- It is important to provide indicators like window titles, consistent colour codes and graphical icons that helps the user to know the context of the work at hand.
- In addition, the user should be able to determine where he has come from and what alternatives exist for a transition to a new task.

- **The system should maintain the consistency within a family of applications**

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

- **The new systems should not change any model that is created by the user's expectation, unless there is some genuine reasons**

- Once a particular interactive sequence has become a standard, whether it is right or wrong, the user community expects the same in all other applications. The best example is: Control S for saving a file.
- Any modification in use of Control S will lead to confusions. For example, if we use Control S for scaling, it leads to confusion.

Q. 5(a) What is need for defining a software scope ?
What are the categories of software engineering resources?

(Ans. : Refer sections 8.5.10 and 8.6.1)

(7 Marks)

Ans. :

- A software project manager is normally confused with a condition in the beginning of a software engineering project i.e. quantitative estimates and an organized plan. They are required and necessary but proper information is not available.
- A detailed analysis of software requirements may provide all the necessary information for estimates, but analysis often takes too much time to complete, even weeks or months. Requirements may be changing regularly as the project proceeds.
- Therefore, the developer and the manager should examine the product and the problem. It is intended to solve at the beginning of the project itself. At a minimum, the scope of the product must be established and bounded.
- The first software project management endeavour is the finding out of software scope. Scope can be defined by answering the following simple questions :
 - **Context :** How the software system can be developed to fit into a large system or some business context and what are the constraints that should be considered as a result of the context considered ?



- **Information objectives** : What data objects are produced as output through the software developed from customer's point of view ? And what data objects will be used for the input ?
- **Function and performance** : What are the functions that the software should perform in order to translate input into output ? Is there any special performance characteristic that is supposed to be addressed ?
- There should be a clear and comprehensive software project scope defined and that should be understandable at all the levels (i.e. the management and technical levels).
- Generally the software scope describes the following :
 - Performance
 - Function
 - The data and control used to define the constraints
 - Reliability and
 - Interfaces.
 - Functions are described in the statement of scope are assessed and evaluated to provide details in estimation. The cost and schedule are two important parameters and they are functionality oriented.
- The Fig. 1-Q. 5(a) depicts the three major categories of software engineering resources : people, reusable software components, and the development environment.
- Each resource is specified with 4 characteristics :
 - description of the resource;
 - a statement of availability;
 - time when the resource will be required;
 - duration of time that resource will be applied.

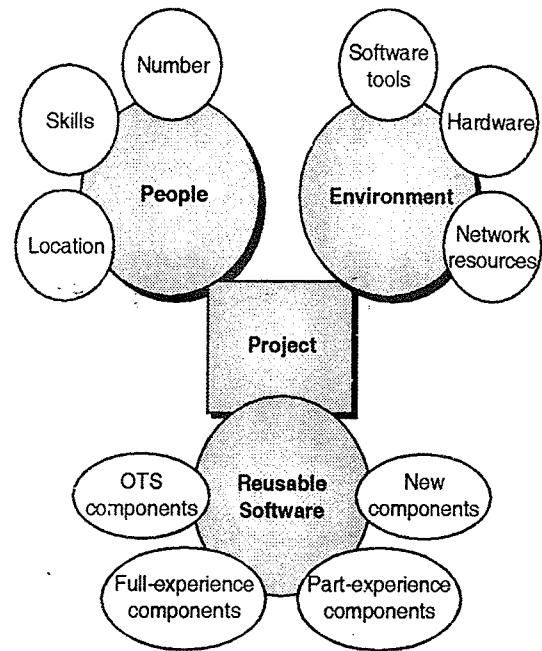


Fig. 1-Q. 5(a) : Project Resources

Q.5 (b) Compare software measurement and metric.
State the measurement principles.

(Ans. : Refer section 8.2 to 8.4) (6 Marks)

Ans. :

Metrics in the Process and Project Domains

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

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

Process Metrics

- The only logical and reasonable way to escalate any process is to measure specific

traits of the process, develop a set of understandable metrics based on these attributes, and then use the metrics to provide pointers that will lead to a plan for improvement.

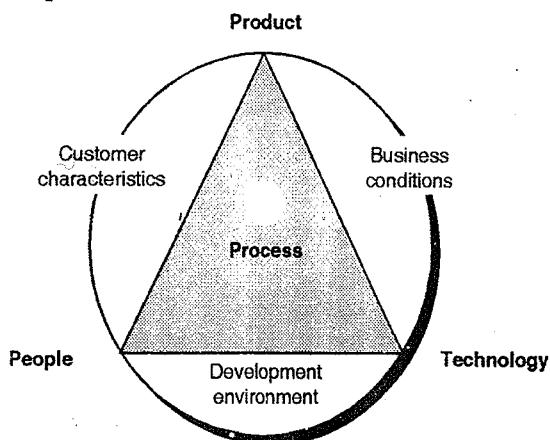


Fig. 1-Q. 5(b) : Determinants for software quality and organizational effectiveness

- In the Fig. 1-Q. 5(b), the process resides at the center of a triangle. The three corners of the triangle are actually the factors that have a profound impact on software quality and the organizational performance. They are :
 - o Product
 - o People
 - o Technology
- The skill set and motivation of the team members are shown to be the most significant factor in quality and performance. The other parameters like complexity of the product, also has a greater impact on quality and team performance. And finally the technology will also have some impact.

Software Measurement

- We have seen that software measurement can be classified in two ways:
 - o **Direct measures** of the software process and product (e.g. Lines of Code (LOC) produced).
 - o **Indirect measures** of the product that includes functionality, quality,

complexity, efficiency, reliability, maintainability, and many other abilities.

- Project metrics can be consolidated to create process metrics that are public to the software organization as a whole. However, if the measures are normalized then it is possible to create software metrics that enable comparison to broader organizational averages. Both size-oriented and function-oriented metrics are normalized in this manner.

Q. 5(c) Explain the reasons for software project failure.

(Ans. : Refer section 8.4.7 to 8.4.7.1)(4 Marks)

Ans. :

- The **software reliability** is defined as the probability of failure free program in a specified environment.
- When we discuss failure, one question comes into mind. What is failure?
- Failure can be defined as non conformance to the software requirement i.e. the software application does not produce the desired output as per the requirements and these results in failures.

Measures of Reliability and Availability

- If we talk about hardware reliability model then we predict failure due to wear rather than failure due to design defects i.e. physical wear due to effect temperature, corrosion, shock etc.
- But when we discuss software reliability then the failure is related to design defects and the failure can be traced to implementation problems.
- Consider a computer based system in that the measure of reliability is Mean-Time-Between-Failure (MTBF) where

$$\text{MTBF} = \text{MTTF} + \text{MTTR}$$



Where MTTF = Mean-Time-To-Failure

MTTR = Mean-Time-To-Repair

- In addition the software availability is defined as the probability that a program is running as per the requirement at a given point of time.

$$\text{Availability} = [\text{MTTF} / (\text{MTTF} + \text{MTTR})] \times 100\%$$

- The availability measure is more sensitive to MTTR and it is an indirect measure of maintainability of software.

Q.6 (a) Explain COCOMO Model for project estimation with suitable example.

(Ans. : Refer section 8.9.2) (7 Marks)

Ans. :

- Barry Boehm has devised a software estimation models hierarchy having the name as COCOMO i.e. COnstructive COst MOdel. The COCOMO model has been evolved into more comprehensive model called COCOMO II. It is actually having a hierarchy of estimation models that focus the following areas :

- o **Application composition model :** It is used in the early stages of development, when user interfaces, system interaction, performance assessment and technology evaluation are prime important.
- o **Early design stage model :** Once the requirements are stabilized and basic architecture is constructed, then early design stage model is used.
- o **Post-architecture stage model :** It is used during development of the software.
- COCOMO II models also require sizing information like other estimation models for the software. Following three sizing options are available :
 - o Object points

- o Function points and
- o Lines of source code

- The COCOMO II model uses object points that are an indirect software measure. They are computed using the counts of :

- o Screenshots taken at user interface
- o Reports and
- o Components required in developing the application.

- The screenshots or the reports are classified into any of the following three complexity levels :

- o Simple
- o Medium
- o Difficult

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

Table 1-Q. 6(a) : Complexity weighting for object types

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

- After the determination of complexity, the number of screenshots, reports, and components object point are weighted as per the Table 1-Q. 6(a). In component-based development when software reuse is implemented, then the percent of reuse (% reuse) is estimated and the object point count is adjusted according to the following equation :

$$\text{NOP} = (\text{object points}) \times [(100 - \% \text{ reuse})/100]$$

where NOP → New Object Points.



Table 2-Q. 6(a) : Productivity rate for object points

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

- In order to derive the estimate of effort, a "productivity rate" should be derived first. They are based on the computed NOP value. The Table 2-Q. 6(a) shown above presents the productivity rate.
- These productivity rates are illustrated for various levels of developer's experience as well as various levels of environment maturity.

$$\text{PROD} = \text{NOP}/\text{person-month}$$

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

$$\text{Estimated effort} = \text{NOP}/\text{PROD}$$

Q.6 (b) What is a task network in project scheduling? Explain with example.

(Ans. : Refer section 9.1.1) (6 Marks)

Ans. :

- Regardless of model used, process model is populated by a set of tasks that enable a software team to define, develop and ultimately support computer software. But the same set of tasks is not appropriate for all types of projects.
- For larger projects, different set of tasks will be applicable. In the same way, for complex projects also, some different set of tasks will be used.
- For developing a project schedule, the entire task set should be divided on the project time line.
- For each project, the set of tasks will vary depending on the type of the project.

- Following are some types of projects :
 - o Concept development projects
 - o New project having certain application
 - o Application enhancement projects
 - o Maintenance projects
 - o Reengineering projects
- In a particular type of project, many factors influence the task set to be chosen.
- These factors include :
 - o Size of the project
 - o Number of users
 - o Stability requirements
 - o User friendliness
 - o Ease of communication between the application developer or user
 - o Performance
 - o Technology used

An example set of tasks

- Consider the example of concept development projects. Following are the sets of tasks those can be applied :
 - o Concept scope
 - o Concept planning
 - o Risk assessment and management
 - o Proof of concept
 - o Implementation
 - o Customer feedback and reaction
- These are certain activities or the set of tasks those can be applied for the concept development projects.

Q.6(c) Explain various factors considered while forming software teams.

(Ans. : refer section 8.4.6 to 8.4.6.2) (4 Marks)

Ans. :

- The most software developers will agree that high-quality software is an important goal. In the most general sense, software quality is conformance to explicitly stated,

- functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.
- The definition emphasizes three important points :
 1. Software requirements are the basis for software quality measurement. If requirements are not satisfied then obviously it will yield low quality software.
 2. For the development processes, some standards are specified that define the development criteria. These development criteria will guide software engineering process. If the criteria are not followed properly then the quality of the product will be quite low.
 3. There are two types of requirements :
 1. Implicit requirements and
 2. Explicit requirements
 - Even if the explicit requirements are considered, the quality may be degraded if it fails to meet implicit requirements.
 - Software quality is a complex mix of factors that will vary across different applications and the customers who request them.
 - Software quality factors are identified and the human activities required to achieve them are described as follows :
 1. McCall's Quality Factors
 2. ISO 9126 Quality Factors

McCall's Quality Factors

- The factors that affect software quality can be categorized in two broad groups :
 - o Factors that can be directly measured (e.g. defects uncovered during testing) and

- o Factors that can be measured only indirectly (e.g. usability or maintainability)
- The categorization of factors that affect software quality shown in Fig. 1-Q.6(c), Focus on three important aspects of a software product :
 1. Its operational characteristics,
 2. Its ability to undergo change and
 3. Its adaptability to new environments

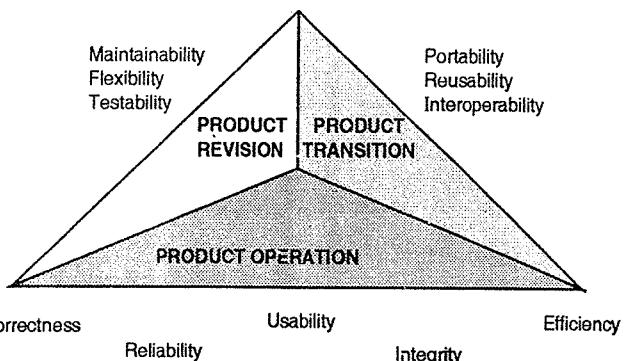


Fig. 1-Q. 6(c) : McCall's software quality factors

- Referring to the factors noted in Fig. 1-Q.6(c), provide the following descriptions :
 1. **Correctness** : The correctness is the quality factor that assesses the software for its correct functioning according to the specification and customer's requirements.
 2. **Reliability** : It is the quality attribute that evaluates the software for its working according to the desired precision.
 3. **Usability** : The software usability is defined as the degree to which the application software is easy to use. It defines the efforts taken to operate and learn, submit input and interpret output.
 4. **Efficiency** : It is defined as the amount of various resources required to execute a given program. The computing resources are memory devices, processing devices and input output devices etc.

5. **Maintainability** : It is defined as the effort needed to uncover the errors and the problems in functioning of a program and resolve these errors.
6. **Integrity** : The integrity of a system is defined as the efforts taken to manage access authorization. The integrity of a system can be ensured by controlling all the accesses to it.
7. **Flexibility** : The ease with which a program can be modified and extended in future.
8. **Testability** : Amount of time and the efforts taken to test a program.
9. **Portability** : Shifting a program from one environment to another.
10. **Reusability** : How existing codes can be reused in future developments according to the scope of the software.
11. **Interoperability** : Ability to connect one system to the another system.

ISO 9126 Quality Factors

The ISO 9126 standard was developed in an attempt to identify quality attributes for computer software. The standard identifies six key quality attributes :

1. **Reliability** : The software should be available for the use and should satisfy following key attributes :
 - o Maturity
 - o Fault tolerance i.e. ability to withstand against failures
 - o Recoverability i.e. the system should regain its original state once the failure occurs. It should also ensure the data integrity and consistency while recovering from failure states.
2. **Portability** : Portability means the system should be able to work in different hardware and software environments according to the following attributes :

- o Installability
 - o Adaptability
 - o Conformance
 - o Replaceability
3. **Efficiency** : Efficiency means making the optimal use of the system resources like memory devices, processing devices and input output devices etc. It should take into consideration following attributes :
 - o Time behaviour and
 - o Resource behaviour.
 4. **Maintainability** : Maintainability is defined as the ease with which the software may be repaired and ready to use once again. The maintainability should take into consideration following attributes :
 - o Changeability
 - o Testability
 - o Stability and
 - o Analyzability
 5. **Functionality** : The degree to which the application software satisfies the customer's needs and requirements according to the following attributes :
 - o Accuracy of working model
 - o Suitability of the product
 - o Interoperability (i.e. the product can be connected to any system),
 - o Compliance and
 - o Security
 6. **Usability** : The software usability is defined as the degree to which the application software is easy to use according to the following usability attributes :
 - o Understandability of the software by the end-uses
 - o Learnability means learning the use of the product and



- Operability.
- The ISO 9126 factors do not support direct measurement of quality, but provides indirect measurement.
- It provides a very good checklist for evaluating the quality.

Q.7 (a) What is software SCM repository? Explain the features of tool set supporting SCM-repository.

(Ans. : Refer section 11.2 to 11.2.2) (6 Marks)

Ans. :

- In the early days of software engineering, software configuration items were maintained as paper documents (or punched computer cards), placed in file folders or three-ring binders, and stored in metal cabinets.
- This approach was problematic for many reasons:
 - To find a SCI is a difficult task when it is needed.
 - To determine which items were changed and by whom. It is always challenging.
 - To develop a new version from an existing program is prone to errors and time consuming too.
 - To describe detailed relationship is actually impossible.
- SCIs are catalogued in the project database with a single name, they reside in the repository. The repository is a database that stores the software engineering information. The software developer or engineer interacts with the repository by using built in tools within repository.

The Role of the Repository : The SCM repository is the set of mechanisms and data structures that allow a software team to manage change in an effective manner. The repository will perform all the fundamental operations of database management system

and in addition it will perform the following operations :

- **Data integrity :** Data integrity validates all the entries to the repository and make sure that the consistency among various objects intact and takes care of all the modifications takes place. It will also ensure cascading modifications i.e. change in one object causes change in a dependent object also.
- **Information sharing :** It is mechanism for sharing information among various developers and between various tools. These tools manage and control multi-user access to data, and locks or unlock objects to retain its consistency.
- **Tool integration :** Tool integration is a data model that can be used by many software engineering tools to control access to the data, and performs appropriate configuration management functions.
- **Data integration :** Data integration provides database functions that allow various SCM tasks to be performed on one or more SCIs.
- **Document standardization :** Document standardization is an important task for defining the objects. This standardization is a good approach for making software engineering documents.
- **Methodology enforcement :** Methodology enforcement defines an (E-R model) i.e. entity-relationship model available in the databases i.e. repository. This model may be used as a process model for software engineering. It is mandatory that the relationships and objects must define before building the contents of the repository.
- To achieve these functions, the database is used as a meta-model. This meta-model exhibits the information and how this information is stored in the databases i.e. repository. This meta-model also checks data security and integrity.

General Features and Content

- The contents of databases and features of databases are considered from two perspective :
 - o What data is to be stored in the databases ?
 - o What services are provided by the databases ?
- A detailed breakdown of types of representations, documents, and work products that are stored in the repository is presented in Fig. 1-Q. 7(a).

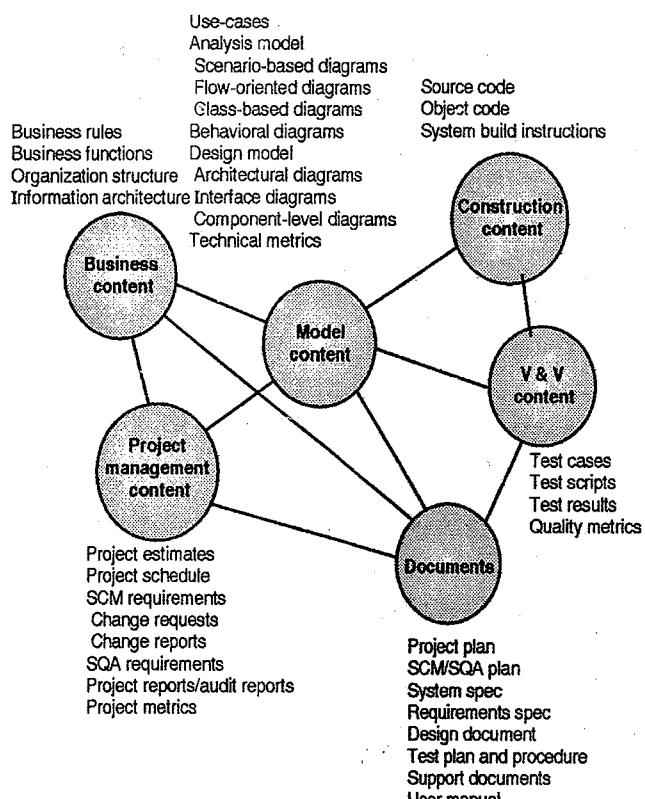


Fig. 1-Q. 7 (a) : Content of the repository

- A robust repository provides two different classes of services :
 1. The same types of services that might be expected from any sophisticated database management system and
 2. Services that is specific to the software engineering environment.
- A repository that serves a software engineering team should :
 - o Integrate with or directly support process management functions;

- o Support specific rules that govern the SCM function and the data maintained within the repository;
- o Provide an interface to other software engineering tools; and
- o Accommodate storage of sophisticated data objects (e.g., text, graphics, video, audio).

Q.7(b) In recent year, university has computerized its examination system by using various software applications. Find out Risk involved in implementation and administration you as software expert. Prepare RMMM Plan for the same.

(Ans. : Refer section 10.5 and 10.6) (6 Marks)

Ans. :

Project Risks:			
Risk	Probability	Effects	Risk planning strategy
The experience staff in the team leave the project before it is finish, or someone was ill	low	serious	Use more than one staff for each section, which might minimize this risk. Also, manager tries to increase salary for him.
The methodology to solve the problem can't work in a proper manner.	high	serious	Must be study more than one methodology to minimize this risk.
Budget does not enough or there is no budget.	low	catastrophic	Put a condition in the contract if there any more expenses, the funded side must be pay it. To avoid this risk.
HW requirement can't come in the time.	moderate	serious	See if there is any more time to delay the project or not. If there is no more time work by the team computers, to minimize this risk.



Product Risks:			
Risk	Probability	Effects	Risk planning strategy
Packages and Development tools does not enough.	high	serious	Put a condition in the contract to increase the time of project delivery depends on the problem occur. To avoid this risk.
Can't found the suitable components.	high	tolerable	Programmer must have professional programming skills to write a new code, which minimize this risk.

Business Risks:			
Risk	Probability	Effects	Risk planning strategy
Can't found the suitable place for meeting the team.	moderate	tolerable	Monitoring the work by E-mail every day. To avoid this risk.
Damage the electricity generator.	high	serious	There is a spare generator to avoid this risk.
Marketing the product system.	low	catastrophic	Distribution of advertisements, which minimize this risk.

Q. 7(c) What is forward engineering? Compare with reverse engineering.

(Ans. : Refer section 12.8) (4 Marks)

Ans. :

- Consider a program with control flow that has modules with more than 2000 statements and few meaningful comment lines with suppose more than 3 Lac source statements. Let no other documentation must be modified for any requirement change.
- In this situation, we have the following options available with us :
 - o We can be lost with too many modifications and necessary design changes.

- o We can understand the working of modules and functions to make the modifications easy.
 - o We can redesign the part of software that suffered with requirement modifications and recode them accordingly.
 - o Or finally we can redesign from start and code again from beginning itself.
 - Out of the above mentioned options no option is actually correct option.
 - Whenever a software firm sells some product, preventive maintenance is taken into consideration for the new release of the application.
 - The **forward engineering process** applies all the software engineering life cycle processes to re-create an existing application. In most of the cases, forward engineering is not the method to just create a new version of an older program but the new users and new technologies and new requirements are also integrated into the reengineering process.
 - Thus the new software application has all the valid functionalities of the older software application and in addition the new capabilities.
- ### Reverse Engineering
- The **Reverse Engineering** is the discipline of software engineering, where the knowledge and design information is extracted from the source code or it is reproduced.
 - The Reverse Engineering is a process where the system is analyzed at higher level of abstraction. It is also called as going backward through all the development cycles.
 - Following are some important purposes of Reverse Engineering :

- Security auditing
 - Enabling additional features
 - Used as learning tools
 - Developing compatible products cheaper than that are currently available in the market.
- Following are three important parameters to be considered for of a reverse engineering process :
- Abstraction Level
 - Completeness
 - Directionality

1. Abstraction Level

- In the abstraction level of a reverse engineering process, the design information is extracted from the source code. It is the highest level in the reverse engineering process. It is always expected that the abstraction level for any reverse engineering process must be high.
- When the level of abstraction is high, then it becomes easy for the software developer to understand the program.

2. Completeness

The completeness is nothing but the details available through the abstraction level of reverse engineering process. For example from the given source code it is very easy to develop the complete procedural design. .

3. Directionality

The directionality of a reverse engineering process is a method of extracting information from the source code and making it available to the software developers.

Q. 8(a) What are the elements that exists when an effective SCM system is implemented? Discuss each briefly.

(Ans. : Refer section 11.1.1) (6 Marks)

Ans. :

Four basic elements that should exist when a configuration management system is developed :

- 1. Component elements :** The tools in the file management system uses the software configuration item.
- 2. Process elements :** The process elements or the procedures uses effective approach towards the change management in engineering and use of computer software.
- 3. Construction elements :** The automated tools are used in construction or the development process and ensuring the validated components should be assembled.
- 4. Human elements :** In order to make the effective use of SCM, the team makes the use of various tools and process feature.

These elements are not mutually exclusive. For example, component elements work in conjunction with construction elements as the software process evolves. Process elements guide many human activities that are related to SCM and might therefore be considered human elements as well

Q. 8(b) Explain various risk associated with software project. How they are managed ?

(Ans. : Refer section 10.1.1) (6 Marks)

Ans. :

- Software risk is a type of risk that always threatens the project development processes and the software under construction.
- Following are three important categories of risk :
 1. Project risks
 2. Product risks
 3. Business risks

1. Project risks

- These are the risks that directly affect the schedule of the project and the resources involved in the development process.
- **Example of project loss :** Loss of an experienced developer and designer.

2. Product risks

- The product risks affect the quality and performance of the application built.
- **Example of product risks :** Failure of a purchased component to perform as per expectation.

3. Business risks

- The business risks are affecting the organization those develop and process the software.
- **Example of business risks :** A competitor of the organization introducing a new product.

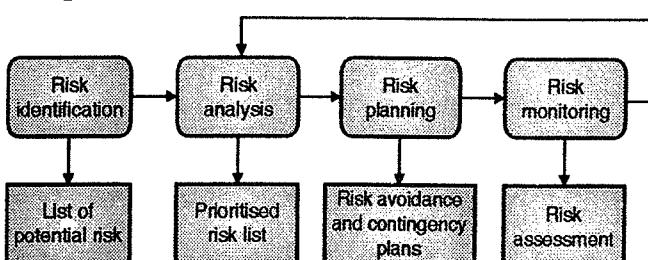


Fig. 1-Q. 8(b) : The Risks management process.

Q. 8(c) Explain Software Reengineering Process model in detail.

(Ans. : Refer section 12.5) (4 Marks)

Ans. :

- Over the period of time each of software application requires maintenance because some unexpected and serious side effects occur during the use.
- Therefore application must be evolved to handle such issues. In order to maintain the software, software engineering processes are reapplied using the concept of software maintenance.
- This concept is known as software

reengineering. Much of the software we use considerably old. Even when these programs were created using the best design and coding techniques known at the time, they were created when program size and storage space were principle concerns.

- They were then migrated to new platforms, adjusted for changes in machine and operating system technology and enhanced to meet new user needs, all without enough regard to overall architecture.
- The result is the poorly designed structures, poor coding, poor logic, and poor documentation of the software systems we are now called on to keep running.
- Reengineering takes time; it costs significant amounts of money; and it absorbs resources that might be otherwise occupied on immediate concerns. For all of these reasons, reengineering is not accomplished in a few months or even a few years.
- Reengineering of information systems is an activity that will absorb information technology resources for many years. That's why every organization needs a pragmatic strategy for software reengineering.

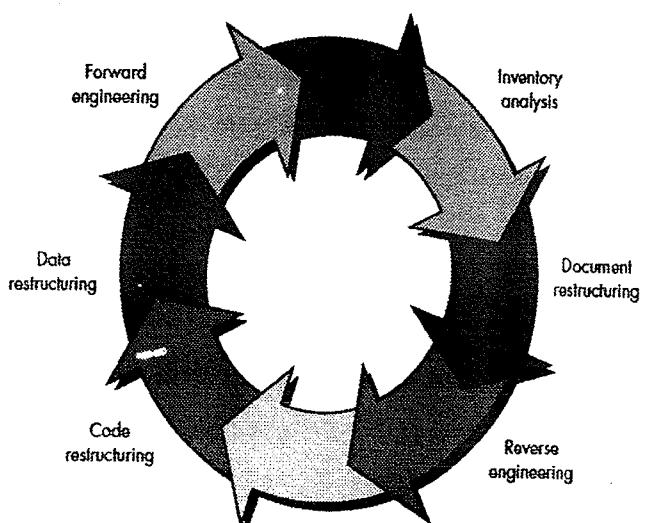


Fig.1-Q.8(c) : A software reengineering process model

- The reengineering paradigm shown in the Fig. 1-Q.8(c) is a cyclical model. This means that each of the activities presented as a



part of the paradigm may be revisited. For any particular cycle, the process can terminate after any one of these activities.

Inventory Analysis

- Every software organization should have an inventory of all applications. The inventory can be nothing more than a spreadsheet model containing information that provides a detailed description (e.g., size, age, business criticality) of every active application. It is important to note that the inventory should be revisited on a regular cycle.

Document restructuring

- Documentation must be updated, but we have limited resources. We'll use a "document when touched" approach. It may not be necessary to fully redocument an application. Rather, those portions of the system that are currently undergoing change are fully documented.

Reverse engineering

- The Reverse Engineering is a process where the system is analyzed at higher level of abstraction. It is also called as going backward through all the development cycles.

Code restructuring

- The code within the suspect modules can be restructured.

Data restructuring

- Data objects and attributes are identified, and existing data structures are reviewed for quality.

Forward engineering

- It applies to all the software engineering life cycle processes to re-create an existing application.

Q. 9(a) Explain in detail, basis path testing as a white box testing technique with following details:

- (i) Flow graph notation
- (ii) Cyclomatic complexity
- (iii) Test case derivation

(Ans. : Refer section 13.7.1 to 13.7.1.3)

(9 Marks)

Ans. :

- Basis path testing use the concept of white-box testing. It enables the developer to design test cases in order to derive a logical complexity measure of a procedural design. To define a basis set of execution paths, the developer uses this complexity measure as a guide.
- In this, test cases designed are guaranteed to execute all the statements in the program at least once.
- Following are the examples of basis path testing :

1. Flow graph notation
2. Independent program paths
3. Deriving test cases
4. Graph matrices

(i) Flow Graph Notation

- The flow graph depicts logical control flow using the notation illustrated in Fig. 1-Q.9(a). Each structured construct has a corresponding flow graph symbol.
- To illustrate the use of a flow graph, we consider the procedural design representation in the Fig. 2(a)-Q.9(a).
- In the Fig. 2-Q.9(a), to show the program control structure, a flowchart is used. Fig. 2(b)-Q.9(a) there is mapping of flowchart into a flow graph.
- Refer Fig. 2(b)-Q.9(a), to show the flow graph node are represented by circles. These circles represent one or more procedural statements.

- A single node encompasses a sequence of process boxes and a decision diamond.
- The edges or links represent flow of control and are denoted by the arrows on the flow graph. These arrows are similar to flowchart arrows.

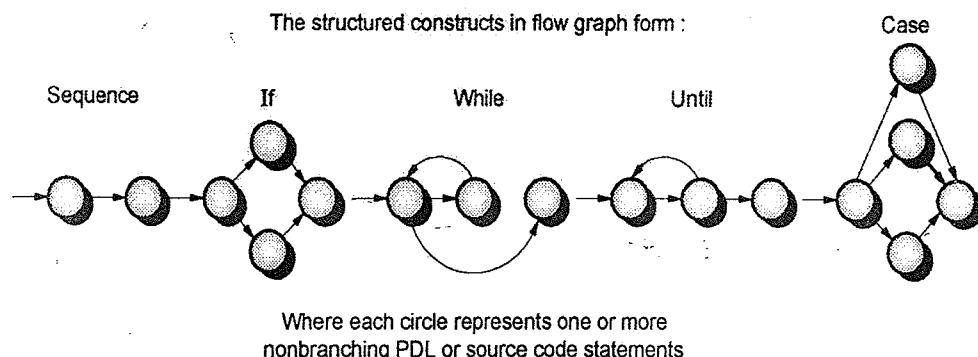
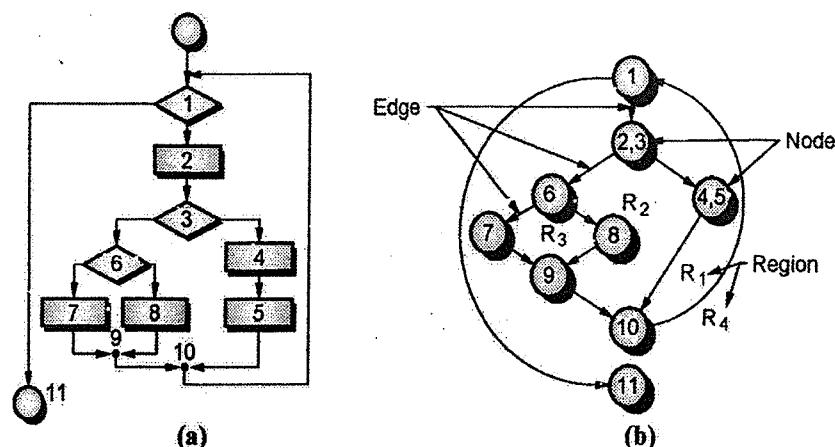


Fig. 1-Q. 9(a) : Flow graph notation



(a) Flowchart

(b) Flow Graph

Fig. 2-Q. 9(a)

- Even if, a node does not represent any procedural statements, an edge must terminate at a node.
- The areas surrounded by edges and nodes are referred as regions. When we consider regions, we should include the area outside the graph as a region.
- Whenever in a procedural design, the compound conditions are encountered, the creation of a flow graph becomes more complex.

In a conditional statement, a compound condition is encountered whenever one or more Boolean operators are present. The examples of Boolean operators are logical OR, AND, NAND, NOR etc.

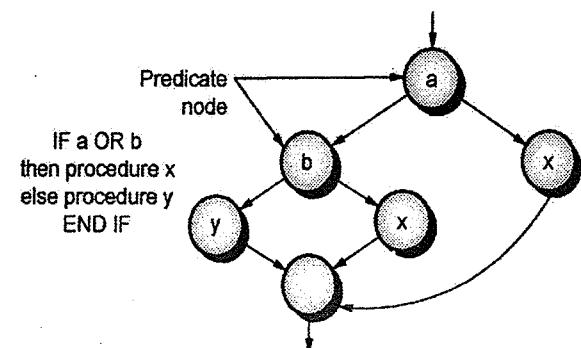


Fig. 3-Q. 9(a) : Compound logic

- The PDL (Program Design Language) segments are translated into the flow graph as shown in Fig. 3-Q.9(a). For each of the conditions a and b, a separate node is created in the statement (i.e. IF a OR b).
- In the above flow graph, each node that contains a condition is known as predicate



node. The predicate node is characterized by two or more edges coming from it (predicate node).

(ii) Independent Program Paths

- An independent path is defined as a path in the program that introduces at least one new set of processing statements or introduces a new condition.
- When an independent path is stated in terms of a flow graph, it must move along at least one edge and that edge should not have been traversed before defining the path.
- For example, A set of independent paths for the flow graph are illustrated in Fig. 2-Q. 9(a) :

Path 1 : $1 \rightarrow 1$

Path 2 : $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 10 \rightarrow 1 \rightarrow 11$

Path 3 : $1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 1 \rightarrow 11$

Path 4 : $1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 7 \rightarrow 9 \rightarrow 10 \rightarrow 1 \rightarrow 11$

- We observe that, each of the new path introduces a new edge.

The path :

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 10 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 1 \rightarrow 11$

is not an independent path since it is a combination of some previously specified paths only. It has not traversed any new any new edge.

- The paths 1, 2, 3, and 4 are considered as a basis set for the flow graph in Fig. 2-Q. 9(a), i.e. if test cases are designed to execute these paths (i.e. a basis set), then each of the statements in the program will be executed at least once, and these condition will be executed to find whether they are true or false.
- The basis set is not unique. Actually, different basis sets can be derived for a given procedural design.

- Cyclomatic complexity provides quantitative measure of the logical complexity of the program. The **Cyclomatic complexity** is a software metrics.

- In the context of the basis path testing, the value of Cyclomatic complexity is calculated and they are the independent paths in the basis set. The Cyclomatic complexity provides an upper bound for the test cases and it guarantees that all the statements are executed at least once.

- The Cyclomatic complexity is considered as a foundation in graph theory. It is calculated in one of the following three ways :

- o How many regions are related to the Cyclomatic complexity ?
- o Cyclomatic complexity $V(G)$ for a graph G is defined as :

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

- o In the above equation, E = the number of flow graph edges, and
 N = the number of flow graph nodes.
- o Cyclomatic complexity, $V(G)$, for a flow graph, G , is also defined as :

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

- o In the above equation, P = the number of predicate nodes contained in the flow graph G .

- Referring once more to the flow graph in Fig. 2-Q.9(a), the Cyclomatic complexity can be computed using each of the algorithms just noted :

- (1) The flow graph has four regions.
- (2) $V(G) = 11 \text{ edges} - 9 \text{ nodes} + 2 = 4$.
- (3) $V(G) = 3 \text{ predicate nodes} + 1 = 4$.

(iii) Deriving Test Cases

- The basis path testing method can be applied to a procedural design or to source code. In this section, we present basis path testing as a series of steps.



- The following steps can be applied to derive the basis set:
 - o Using the design or code as a foundation, draw a corresponding flow graph.
 - o Determine the Cyclomatic complexity of the resultant flow graph.
 - o Determine a basis set of linearly independent paths.
 - o Prepare test cases that will force execution of each path in the basis set.
- One important point to remember is that some of the independent paths cannot be tested alone. But the combination of data is required to traverse the path. This can not be achieved in the normal flow of the program. In these cases, the independent paths are tested as part of another path test.

Q. 9(b) What do you understand by System Testing?

What are the different kinds of system testing that are usually performed on large software testing.

(Ans. : Refer section 13.5.4 ,13.6 to 13.8.4)

(8 Marks)

Ans. :

- The “**finger-pointing**” is a classic system testing problem. It occurs whenever an error is detected, and for each of the system element, the developers blame each other for the problem.
- **System testing** is a series of different tests and the main reason of these tests is to test the complete computer-based system. The purpose of each of the tests is to verify all the system elements properly and ensure they are integrated properly and perform required functions.
- Any of the software applications can be tested in the following two ways :

- o It assures that all the functions specified by the customer are fully functional and again searching for more errors in each of the functions.
 - o All the internal operations are working as per the specifications. All the internal components have been traversed properly to uncover the errors.
- In the above two approaches, first test approach is called as black box testing and the second approach is known as white box testing. Thus we can categorize the testing approaches into the following two types :
- o White-Box Testing
 - o Black-Box Testing

White-Box Testing

- **White-box testing also called as glass-box testing.** It employs the test case design concept that uses the control structures. These control structures are described as component-level design to derive the test cases.
- By using the white-box testing, the software developer derive test cases that has the following characteristics :
 - o It guarantees the traversal of all independent paths in a module at least once. It refers the concept of traversal of tree i.e. each of the nodes in a tree must be traversed at least once.
 - o Evaluate all logical decisions and find whether they are true or false.
 - o Evaluate all the loops to check their boundaries and operational bounds, and
 - o Evaluate all internal data structures to confirm their validity.
- Different white-box testing methods are :
 1. Basis Path Testing
 2. Control Structure Testing

Basis Path Testing

- Basis path testing uses the concept of white-box testing. It enables the developer to design test cases in order to derive a logical complexity measure of a procedural design. To define a basis set of execution paths, the developer uses this complexity measure as a guide.
- In this, test cases designed are guaranteed to execute all the statements in the program at least once.
- Following are the examples of basis path testing :
 1. Flow graph notation
 2. Independent program paths
 3. Deriving test cases
 4. Graph matrices

Flow Graph Notation

- The flow graph depicts logical control flow

- using the notation illustrated in Fig. 1-Q. 9(b). Each structured construct has a corresponding flow graph symbol.
- To illustrate the use of a flow graph, we consider the procedural design representation in the Fig. 2(a)-Q. 9(b).
- In the Fig. 2-Q. 9(b), to show the program control structure, a flowchart is used. Fig. 2-Q. 9(b), there is mapping of flowchart into a flow graph.
- Refer Fig. 2(b)-Q. 9(b), to show the flow graph node are represented by circles. These circles represent one or more procedural statements.
- A single node encompasses a sequence of process boxes and a decision diamond.
- The edges or links represent flow of control and are denoted by the arrows on the flow graph. These arrows are similar to flowchart arrows.

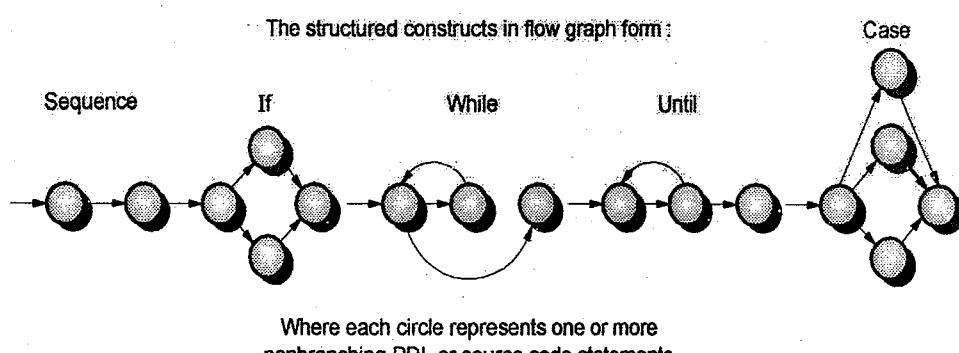
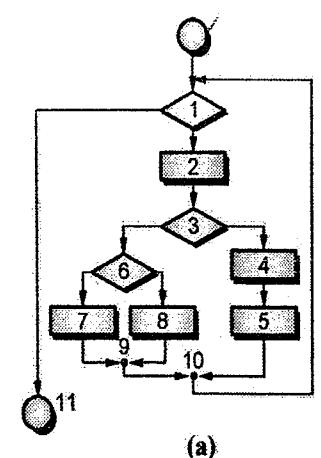
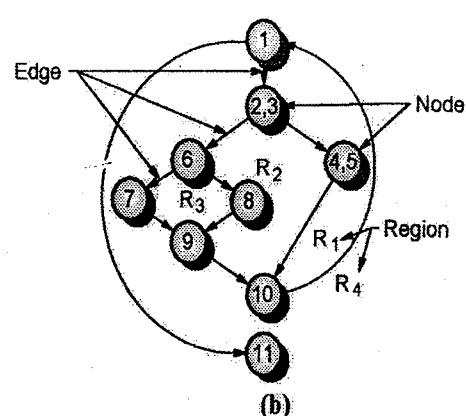


Fig. 1-Q.9(b) : Flow graph notation



(a) Flowchart



(b) Flow Graph

Fig. 2-Q. 9(b)



- Even if, a node does not represent any procedural statements, an edge must terminate at a node.
- The areas surrounded by edges and nodes are referred as regions. When we consider regions, we should include the area outside the graph as a region.
- Whenever in a procedural design, the compound conditions are encountered, the creation of a flow graph becomes more complex.
- In a conditional statement, a compound condition is encountered whenever one or more Boolean operators are present. The examples of Boolean operators are logical OR, AND, NAND, NOR etc.

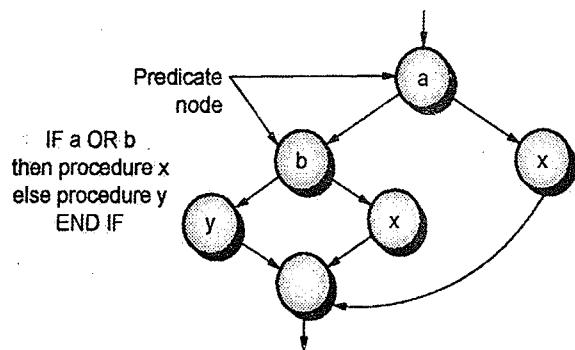


Fig. 3-Q. 9(b) : Compound logic

- The PDL (Program Design Language) segments are translated into the flow graph as shown in Fig. 3-Q. 9(b). For each of the conditions a and b, a separate node is created in the statement (i.e. IF a OR b).
- In the above flow graph, each node that contains a condition is known as predicate node. The predicate node is characterized by two or more edges coming from it (predicate node).

Independent Program Paths

- An independent path is defined as a path in the program that introduces at least one new set of processing statements or introduces a new condition.
- When an independent path is stated in terms of a flow graph, it must move along at least one edge and that edge should not have been traversed before defining the path.
- For example, A set of independent paths for the flow graph are illustrated in

Fig. 2-Q. 9(b) :

Path 1 : $1 \rightarrow 1$ Path 2 : $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 10 \rightarrow 1 \rightarrow 11$ Path 3 : $1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 1 \rightarrow 11$ Path 4 : $1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 7 \rightarrow 9 \rightarrow 10 \rightarrow 1 \rightarrow 11$

- We observe that, each of the new path introduces a new edge.

The path :

 $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 10 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 1 \rightarrow 11$

- is not an independent path since it is a combination of some previously specified paths only. It has not traversed any new edge.
- The paths 1, 2, 3, and 4 are considered as a basis set for the flow graph in Fig. 2-Q. 9(b), i.e. if test cases are designed to execute these paths (i.e. a basis set), then each of the statements in the program will be executed at least once, and these condition will be executed to find whether they are true or false.
- The basis set is not unique. Actually, different basis sets can be derived for a given procedural design.
- **Cyclomatic complexity** provides quantitative measure of the logical complexity of the program. The **Cyclomatic complexity** is a software metrics.
- In the context of the basis path testing, the value of Cyclomatic complexity is calculated



- and they are the independent paths in the basis set. The Cyclomatic complexity provides an upper bound for the test cases and it guarantees that all the statements are executed at least once.
- The Cyclomatic complexity is considered as a foundation in graph theory. It is calculated in one of the following three ways :
 - o How many regions are related to the Cyclomatic complexity ?
 - o Cyclomatic complexity $V(G)$ for a graph G is defined as :
$$V(G) = E - N + 2$$
 - o In the above equation, E = the number of flow graph edges, and
 N = the number of flow graph nodes.
 - o Cyclomatic complexity, $V(G)$, for a flow graph, G , is also defined as :
$$V(G) = P + 1$$
 - o In the above equation, P = the number of predicate nodes contained in the flow graph G .
 - Referring once more to the flow graph in Fig. 2-Q. 9(b), the Cyclomatic complexity can be computed using each of the algorithms just noted :
 - (1) The flow graph has four regions.
 - (2) $V(G) = 11 \text{ edges} - 9 \text{ nodes} + 2 = 4$.
 - (3) $V(G) = 3 \text{ predicate nodes} + 1 = 4$.

Deriving Test Cases

- The basis path testing method can be applied to a procedural design or to source code. In this section, we present basis path testing as a series of steps.
- The following steps can be applied to derive the basis set :

- o Using the design or code as a foundation, draw a corresponding flow graph.
- o Determine the Cyclomatic complexity of the resultant flow graph.
- o Determine a basis set of linearly independent paths.
- o Prepare test cases that will force execution of each path in the basis set.
- One important point to remember is that some of the independent paths cannot be tested alone. But the combination of data is required to traverse the path. This can not be achieved in the normal flow of the program. In these cases, the independent paths are tested as part of another path test.

Graph Matrices

- A graph matrix is a tool that supports basis path testing. A graph matrix is defined as a square matrix whose size is equal to the number of nodes on the flow graph. The size of matrix means the number of rows and columns of the square matrix.
- Each of the row and column of graph matrix corresponds to an identified node. The matrix entries as shown in Fig. 4-Q.9(b) correspond to an edge between nodes (i.e. connections). An example of a simple flow graph and its corresponding graph matrix is shown Fig. 4-Q.9(b).
- Each of the nodes on the flow graph is denoted by some numbers and the edges are represented by some letters as shown in Fig. 4-Q.9(b).
- **For example :** Node 3 in the Fig. 4-Q.9(b) is connected to the node 4 by the edge b. This can be represented in graph matrix by writing "b" in the corresponding cell.

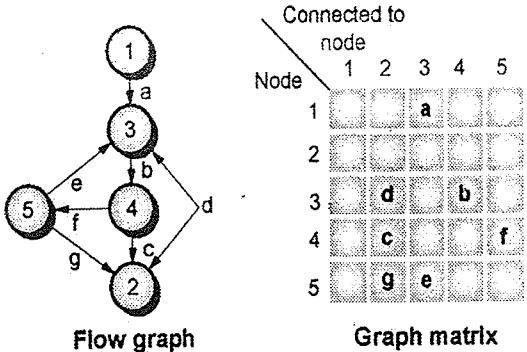


Fig. 4-Q. 9(b) : Graph Matrix

- The graph matrix is actually a tabular representation of a flow graph. By adding a link weight to each of the matrix entry, it becomes a powerful tool for testing program control structure.

Control Structure Testing

- Basis path testing is one of the techniques for control structure testing.
- Since basis path testing is not sufficient, other variations on control structure testing are discussed in the following sections. The examples of control structure testing are :
 1. Condition Testing
 2. Data Flow Testing
 3. Loop Testing

Condition Testing

- The condition testing is one of the test case design method. In condition testing, all the logical conditions in the program or the program module are tested.
- An example of a simple condition may be a Boolean variable or a relational expression. The expression may contain a single NOT operator.
- Any relational expression has the following simple form :

$E_1 <$ the relational operator $> E_2$

Where, E_1 and E_2 are two arithmetic expressions and between them, any of the following relational operator may be present :

- $< \rightarrow$ Less than
 $<= \rightarrow$ less than or equal to
 $\neq \rightarrow$ Non-equality
 $> \rightarrow$ Greater than
 $\geq \rightarrow$ Greater than or equal to

- A condition may also be a compound condition and it consists of two or more simple conditions, Boolean operators and parentheses.

Data Flow Testing

- In dataflow testing approach, a test paths of a program is selected based on the locations of definitions and uses of variables in the program.
- To explain the data flow testing method, consider that each of statement in a program is assigned a unique number. Also the functions and global variables do not modify their parameters.
- Consider a statement S (its unique statement number) as follows :
- $\text{DEFX}(S) = \{X \mid \text{statement } S \text{ contains a definition of } X\}$
- $\text{USEX}(S) = \{X \mid \text{statement } S \text{ contains a use of } X\}$
- If statement S is an "if" or "loop" statement, then its DEFX set is empty and its USEX set depends on the condition of statement S .
- The variable X at statement S is the live at statement S' , if there is a path from statement S to statement S' . Also it should not contain any other definition of X .

Loop Testing

- Loops are the important part of nearly all the programs in the software. Normally the developers pay very little attention this while conducting tests.
- The loop testing technique is in the category of a white-box testing. It focuses

mainly on the validity of loop constructs. In the following section, we define four classes of loops :

1. Simple loops

Following tests can be applied to simple loops, where n is the maximum number of passes through the loop :

- o Skip the whole loop.
- o Only one pass allowed.
- o Only two passes allowed.
- o m passes allowed through the loop where $m < n$.
- o Also, the possibilities of $n - 1$, n and $n+1$ passes allowed.

2. Concatenated loops

This concatenated loop approach can be implemented by using the approach discussed in simple loops. In this each of the loops is independent of other. When loops are not independent, this approach is not suitable and instead nested loop approach is suggested.

3. Nested loops

The nested loop approach is an extension of simple loop approach. By nesting the loops, the number of testing levels will be increased automatically. This might result in number of tests which may not be practically possible. Following are some approaches that are suggested to reduce the number of tests :

- o Start with the inner most loops and set all the other loops at minimum levels.
- o Now conduct the simple loop approach to innermost loop while hold other with their minimum interaction value.
- o Move outward to conduct simple test to the next loop and keeping its out loop value at minimum.

- o Continue the same process till all the loops are exercised.

4. Unstructured loops

Whenever it is required, the loops should be redesigned by using structure using structured programming constructs.

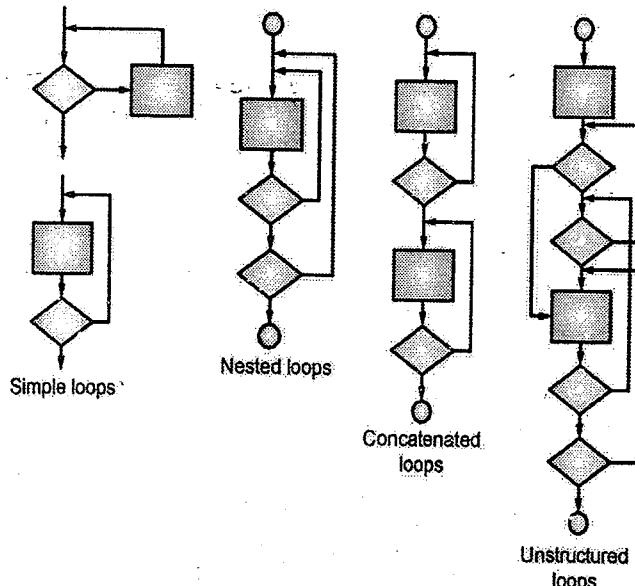


Fig. 5-Q. 9(b) : Classes of loops

Black-Box Testing

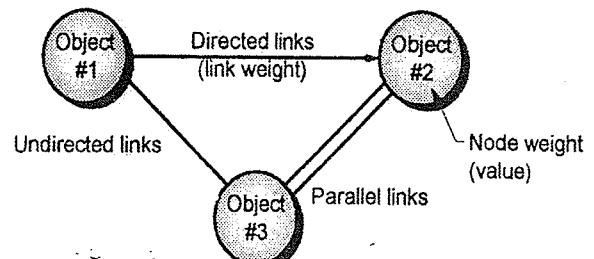
- Black-box testing is also known as behavioural testing. It focuses only on the functional requirements of the software.
- The black-box testing helps the software developer to derive the sets of input conditions. These input conditions exercise all the functional requirements for a program.
- Black-box testing is a different approach and it can not be the alternative or the replacement of white-box testing approach. It is a complementary testing method that detects a different class of errors compared to white-box testing methods.
- Following are categories of errors that can be detected by using black-box testing approach :
 - o The performance errors or the behavioural errors

- Missing or incorrectly defined functions
 - Data structures incorrectly defined and external data base access errors,
 - Errors occurred during initialization and termination.
 - Errors occurred during interface.
- In contrast to white-box testing, where testing begins early in the testing process, the black box testing is applied in the final stages of testing.
- **Different black-box testing methods are :**
1. Graph-Based Testing Method
 2. Equivalence Partitioning
 3. Boundary Value Analysis
 4. Orthogonal Array Testing

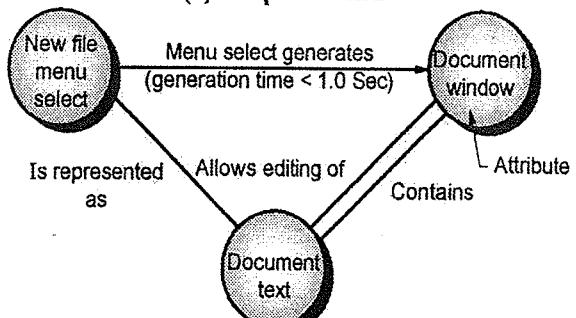
Graph-Based Testing Method

- The software testing starts with drawing the graphs of some important objects and their relationships. After this, the software developer performs the series of tests that cover the graph in such a way that each of the object and its relationship is covered and errors are detected.
- In order to complete these steps, the software developer starts with a graph that represents collection of objects and the relationships among them. The nodes of the graph are objects and links denotes relationships.
- In Fig. 6(a)-Q.9(b), a symbolic representation of a graph is exhibited. The circles in the graph are nodes and they are connected by links. There are different types of links as follows :
 - **A directed link** denoted by an arrow, indicates that a relationship can moves in one direction only.
 - **A bi-directional link**, it is also called as symmetric link i.e. the relationship is applied in both the directions.

- **A Parallel link**, is used in cases where various relationships exist among graph nodes.



(a) Graph notation



(b) Simple example

Fig.6-Q.9(b)

Equivalence Partitioning

- Equivalence partitioning is a testing approach that employs black-box testing method. It divides the input domain of a program into various classes of data. Then these classes are used for deriving the test cases.
- In an ideal test case, this approach detects all the errors single-handedly from different classes of errors.
- The equivalence partitioning approach defines a test case that detects classes of errors and thus reducing the efforts of conducting more test cases.
- The equivalence classes that are generated by partitioning are defined according to the following guidelines :
 - If the input conditions specify a range then one valid and two invalid equivalence classes can be defined.



- If an input condition requires some specific value then also one valid and two invalid equivalence classes are defined.
 - If an input condition specifies a member of a set then in this case, one valid and one invalid equivalence class are defined.
 - If an input condition is a Boolean value, then also one valid and one invalid class are defined.
- By applying these guidelines for the derivation of equivalence classes, test cases for each input domain data object can be developed and executed.

Boundary Value Analysis

- Generally large number of errors occurs at the boundaries of the input domain and not at the "center" of it.
- This is the reason why we require Boundary Value Analysis (BVA). It has been developed and used as a testing technique by the developers. Boundary value analysis selects and conducts different test cases to test the boundary values of input domain.
- BVA is a test case design method in which the equivalence partitioning takes place. In spite of selecting some element of an equivalence class, the boundary value analysis actually selects the test cases at the "edges" of the class.
- Here in boundary value analysis approach, the focus is not only on input conditions but also it derives and conducts the test cases from the output values.
- Following are some guidelines for boundary value analysis approach :
 - If an input condition select a range between a and b then test cases should be designed beyond the values a and b. It means the input values may be just

- below and just above the values of a and b.
 - If an input condition selects the minimum and maximum numbers, then values just above and just below the minimum and maximum values are also tested.
 - The above two guidelines are applied to output conditions.
 - If internal program or its data structures prescribed some boundary values like an array of 100 numbers, then the care must be taken to conduct the test cases up to 100 entries only i.e. its boundary.
- Most of the software developers perform boundary value analysis up to some degree only. After applying all these guidelines on boundary value analysis testing, then it will give more complete testing and higher probabilities of finding errors.
- ### Orthogonal Array Testing
- In various applications, the input domain is relatively limited and hence **orthogonal array testing** is a good option in such situations. In orthogonal array testing, the problems in which the input domain is small but it is large enough to accommodate the exhaustive testing.
 - The orthogonal array testing approach is especially used in detecting errors that are caused by the faulty logic within a software component.
 - To explain the comparison between orthogonal array testing and more conventional approaches, consider the following system in which there are three input values X, Y and Z.
 - All these input values have three discrete values associated with each of it. The probability of test cases is $3^3 = 27$.
 - In the Fig. 7-Q 9 (b) a geometric view of the possible test cases associated are explained by considering the input values i.e. X, Y, and Z.

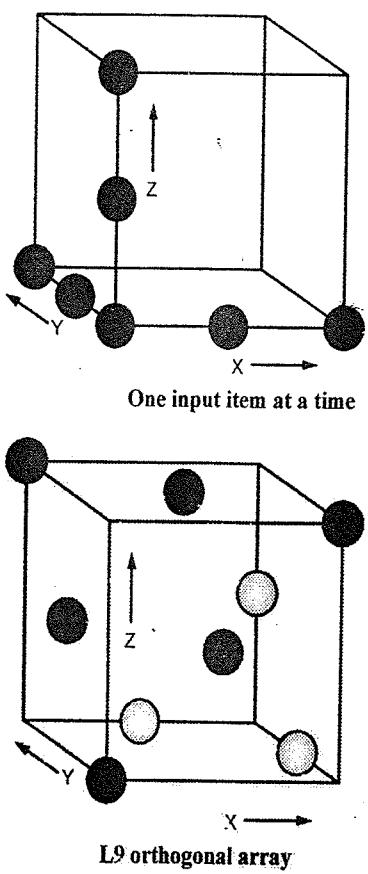


Fig. 7-Q.9(b) : A geometric view of test cases

- Refer the Fig. 7-Q.9(b), in that one input value at a time is changing in accordance with the input axis. The results obtained are generally having less coverage of the input values.
- An L9 orthogonal array of test cases is created whenever an orthogonal array testing is conducted. Generally a L9 orthogonal array has an important property called as **balancing property**. In this the test coverage more complete with the input domain.

Q. 10(a) Explain defect life cycle in detail.

(Ans. : Refer section 13.14) (7 Marks)

Ans. :

- Defect life cycle, also known as Bug Life cycle is the journey of a defect cycle, which a defect goes through during its lifetime. It varies from organization to organization and also from project to project as it is

governed by the software testing process and also depends upon the tools used.

- Defect life cycle is a cycle which a defect goes through during its lifetime. It starts when defect is found and ends when a defect is closed, after ensuring it's not reproduced. Defect life cycle is related to the bug found during testing.
- The number of states that a defect goes through varies from project to project. Following lifecycle diagram, covers all possible states :

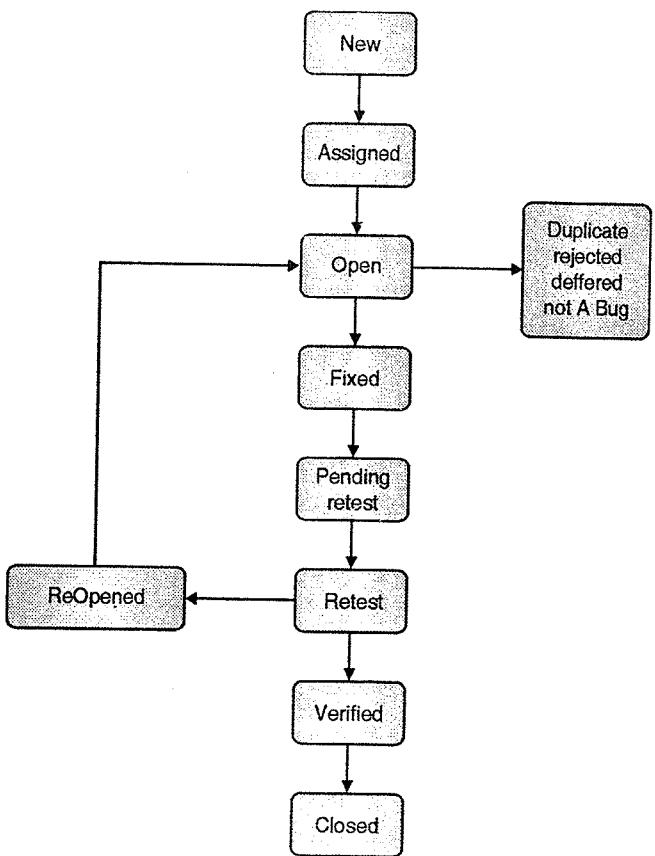


Fig. 1-Q.10(a) : Defect Life Cycle

- **New** : When a new defect is logged and posted for the first time. It is assigned a status NEW.
- **Assigned** : Once the bug is posted by the tester, the lead of the tester approves the bug and assigns the bug to developer team.
- **Open** : The developer starts analyzing and works on the defect fix.



- **Fixed** : When developer makes necessary code change and verifies the change, he or she can make bug status as "Fixed."
- **Pending retest** : Once the defect is fixed the developer gives particular code for retesting the code to the tester. Since the testing remains pending from the testers end, the status assigned is "pending request."
- **Retest** : Tester does the retesting of the code at this stage to check whether the defect is fixed by the developer or not and change the status to "Re-test."
- **Verified** : The tester re-tests the bug after it got fixed by the developer. If there is no bug detected in the software, then the bug is fixed and the status assigned is "verified."
- **Reopen** : If the bug persists even after the developer has fixed the bug, the tester changes the status to "reopened". Once again the bug goes through the life cycle.
- **Closed** : If the bug is no longer exists then tester assigns the status "Closed."
- **Duplicate** : If the defect is repeated twice or the defect corresponds the same concept of the bug, the status is changed to "duplicate."
- **Rejected** : If the developer feels the defect is not a genuine defect then it changes the defect to "rejected."
- **Deferred** : If the present bug is not of a prime priority and if it is expected to get fixed in the next release, then status "Deferred" is assigned to such bugs.
- **Not a bug** : If it does not affect the functionality of the application then the status assigned to a bug is "Not a bug".

Q. 10(b) How Top down and Bottom up integration is achieved ?

(Ans. : Refer section 13.5.2)

(6 Marks)

Ans. :

- Integration testing is used for constructing the software architecture. It is a systematic approach for conducting tests to uncover interfacing errors. The main objective behind integration testing is to accept all the unit tested components and integrate into a program structure as given by the design.
- It is often observed that there is a tendency to attempt always non-incremental approaches. All the components are integrated in the beginning and the program is tested as a whole.
 - In this approach a set of errors are encountered and correction seems to be very difficult due to isolation of causes and the complications by program since program is expanded over several modules. After correction of these errors, some new may appear and the process continues. It looks like an infinite loop.
 - Small increments of the program are tested in an incremental approach. In incremental integration approach, the error detection and correction is quite simple. In this all the interfaces are tested completely and thus a systematic test strategy may be applied.
 - Following are different **incremental integration** strategies :
 - **(1) Top-down integration**
 - Top-down integration testing is one of the incremental testing strategies for construction of the software architecture.
 - All the modules are combined by moving top to down direction through the control hierarchy. It begins with the main program or the main control module. The flow is from main module to its subordinate modules in depth-first or breadth-first manner.

- Depth-first integration is illustrated in Fig. 1-Q.10(b), and it integrates all the components on most of the control path of the program.

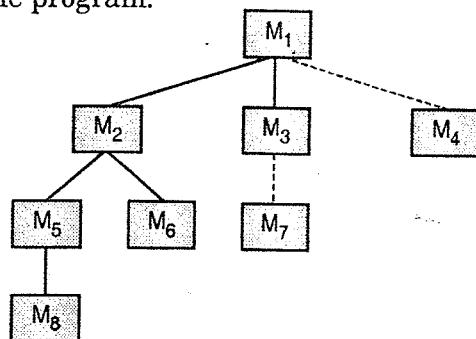


Fig. 1-Q. 10(b) : Top-down integration

Problems associated with Top down approach of testing

- In incremental integration testing approach, a set of errors are encountered and correction seems to be very difficult due to isolation of causes and the complications by program since program is expanded over several modules. After correction of these errors, some new may appear and the process continues. It looks like an infinite loop.
- Top-down integration testing is one of the incremental testing strategies for construction of the software architecture.
- The main problem with top down approach is that the test conditions are nearly impossible or they are extremely difficult to create.

- Stub modules are complicated.

(2) Bottom-up integration

- In bottom-up integration testing, the testing begins with the construction and components at the lowest level. These components are called as atomic modules.
- Since the components are integrated from the bottom to top, the required processing is always available for components subordinate in all the levels. Here in this approach the need of stub is completely eliminated. The use of stub was actually a disadvantage due to overhead.
- Following steps are required for implementation of bottom-up integration strategy:
 - (1) The low-level components are integrated to form clusters that can perform sub function of a specific software.
 - (2) A driver is needed to coordinate all the test case inputs and outputs.
 - (3) Later all the clusters are tested.
 - (4) Then, drivers are removed and all the clusters are integrated once again from bottom to top direction in the program.
- Integration follows the pattern illustrated in Fig. 2-Q.10(b).

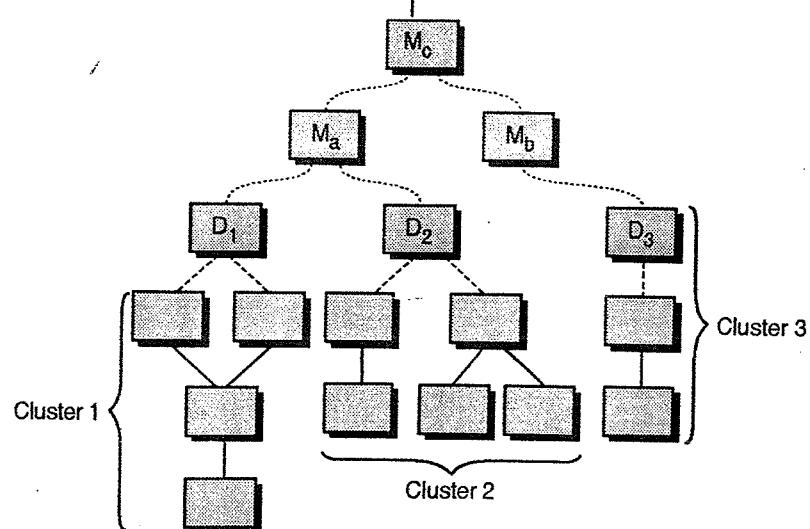


Fig. 2-Q. 10(b) : Bottom-up integration



- The components are integrated in such a fashion that they can form clusters 1, 2 and 3. These clusters are tested by using a driver (marked by a dashed block)
- The components in clusters 1 and 2 are subordinate to module M_a . The drivers D_1 and D_2 are then removed and the clusters are connected directly to M_a .
- In the same way, the driver D_3 for the cluster 3 is removed and then cluster 3 is directly connected to module M_b . Here both the modules, M_a and M_b are finally integrated with the component M_c , and so on.

Q. 10(c) What is difference between Testing and Debugging ?

(Ans. : Refer section 13.1, 13.15.1)(4 Marks)

Ans. :

Testing

- Testing is an important activity in software development process. Before the start of the development, testing is planned and is conducted very systematically. In order to implement the testing, there are various testing strategies defined in the software engineering literature.
- All these strategies provide a testing template to the developers. The testing templates should possess following characteristics that is applicable to any software development process :
 - o For effective testing, formal technical reviews must be conducted by the development team.
 - o Frequent communication between developer and customer resolves most of the complexities and confusion in the beginning itself. Thus before start of the testing process, number of errors may be uncovered and then fixed.
 - o Testing starts from the component level and then finally complete computer based system is integrated at the end.

- o There are various testing techniques are available. So at different point of time, suitable testing technique may be applied.
- o Testing may be conducted by the software developer itself for usually smaller projects. For the larger projects, an independent group (especially those people that are not the part of development team) may be hired for conducting an effective testing.
- o The members of independent testing group may be the member of some other team or may be outsourced.
- o Even though testing and debugging are two separate activities, debugging should be accommodated in testing strategies.

- Any testing strategy should be able to conduct low level tests, since it verifies small source code modules and can be easily implemented.
- It gives better precision. In addition to these low level tests, a testing strategy should also be able to conduct high level tests and it should be able to validate all the major functionalities as per customer's requirements.
- Software testing process is a systematic and planned activity and it is an integral part of the development process.
- The test cases are conducted, and a strategy can be defined, and the results can be evaluated as per the requirements.

Debugging

- Debugging occurs when a test case is successfully conducted. It means that the test case uncovers various errors, and the debugging process can be started. The debugging process attempts to eliminate all the errors that were uncovered in the testing process.

Even if debugging is an orderly process, it is considered to be an art.

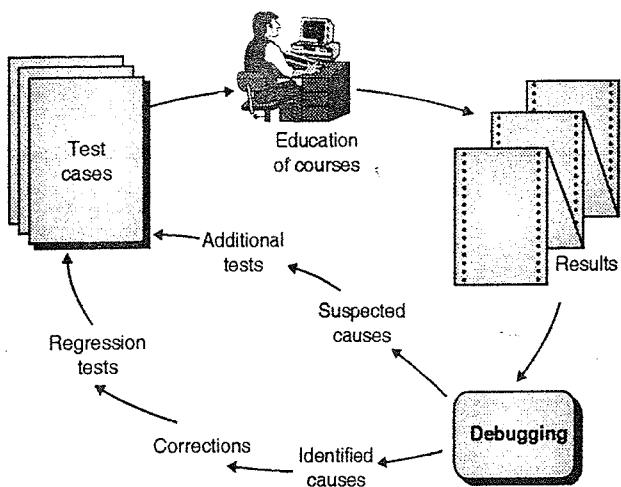


Fig. 1-Q. 10(c) : The debugging process

The Debugging Process

- Debugging is not a testing process but it is the consequence of testing. In the Fig. 1-Q. 10(c), the debugging process starts with conduction of a test case.
- The results are evaluated and a difference between expected and actual performance is encountered.
- The debugging process has one of two outcomes as follows :
 - (1) The reason for causing error are found and corrected, or
 - (2) The reason for causing the error will not be found.
- In the case 2, the person performing debugging will be suspicious about a cause. Conducting the test cases will help to validate that suspicion, and the person can work to correct the errors in iterative fashion.
- It is observed that the debugging is a difficult process. Sometimes human psychology is required in addition to software technology. Following are some characteristics of bugs that will provide some clues to clear the doubts about

debugging process :

- o The cause or the symptom can be geographically remote.
- o The symptom may be unavailable temporarily, while some other errors are corrected.
- o The symptom may also be caused by non-errors (e.g., round-off inaccuracies).
- o The symptom may be caused by some human error and that can not be easily traced.
- o The symptom may be due to timing problems rather than processing problems.
- o It is very difficult to reproduce input conditions accurately. For example real-time applications where order of the input values is indeterminate.
- o The symptom can also be intermittent i.e. not continuous. This scenario is very common in embedded systems.
- o The symptom can also be caused by number of tasks running on different processors in a distributed system scenario.
- In the debugging process, numerous error are encountered that may be mild annoying or catastrophic.
- The examples of mild annoying errors are :
 - o Incorrect output format
- The examples of catastrophic errors are :
 - o System fails
 - o Physical damage
 - o Economical losses

In debugging process, finding the cause of the errors is more important than finding the errors itself. The software developer sometimes eliminates an error and at the same time, two more errors are introduced.

