SUBJECT CODE : 210253

As per Revised Syllabus of

SAVITRIBAI PHULE PUNE UNIVERSITY

Choice Based Credit System (CBCS)

S.E. (Computer) Semester - IV

SOFTWARE ENGINEERING

Anuradha A. Puntambekar

M.E. (Computer)

Formerly Assistant Professor in

P.E.S. Modern College of Engineering, Pune

Dr. Jayashree R. Prasad

Ph.D. (Computer Engineering)

M.E. (Computer Engineering)

Professor (Dept. of Computer Engineering)

Sinhgad College of Engineering, Pune

TECHNICAL PUBLICATIONS

SINCE 1993

An Up-Thrust for Knowledge

SOFTWARE ENGINEERING

Subject Code : 210253

S.E. (Computer) Semester - IV

Published by :

TECHNICAL PUBLICATIONS
SINCE 1993
An Up-Thrust for Knowledge

Amit Residency, Office No.1, 412, Shaniwar Peth,
Pune - 411030, M.S. INDIA, Ph.: +91-020-24495496/97
Email : sales@technicalpublications.org Website : www.technicalpublications.org

Printer :
Yogiraj Printers & Binders
Sr.No. 10/1A,
Ghule Industrial Estate, Nanded Village Road,
Tal. - Haveli, Dist. - Pune - 411041.

ISBN 978-93-90450-41-1

9 789390 450411

SPPU 19

PREFACE

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

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

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

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

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

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

Authors
A. A. Puntambekar
Dr. Jayashree R. Prasad

Dedicated to God

SYLLABUS

Software Engineering - 210253

Credit Scheme	Examination Scheme and Marks
03	Mid _Semester (TH) : 30 Marks
	End _ Semester (TH) : 70 Marks

Unit I Introduction to Software Engineering and Software Process Models

Software Engineering Fundamentals : Introduction to software engineering, The Nature of Software, Defining Software, Software Engineering Practice. **Software Process** : A Generic Process Model, defining a Framework Activity, Identifying a Task Set, Process Patterns, Process Assessment and Improvement, Prescriptive Process Models, The Waterfall Model, Incremental Process Models, Evolutionary Process Models, Concurrent Models, A Final Word on Evolutionary Processes. Unified Process, Agile software development: Agile methods, plan driven and agile development. **(Chapter 1)**

Unit II Software Requirements Engineering and Analysis

Modeling : Requirements Engineering, Establishing the Groundwork, Identifying Stakeholders, Recognizing Multiple Viewpoints, working toward Collaboration, Asking the First Questions, Eliciting Requirements, Collaborative Requirements Gathering, Usage Scenarios, Elicitation Work Products, Developing Use Cases, Building the Requirements Model, Elements of the Requirements Model, Negotiating Requirements, Validating Requirements. **Suggested Free Open Source tools** : StarUML, Modelio, SmartDraw. **(Chapter 2)**

Unit III Estimation and Scheduling

Estimation for Software Projects : The Project Planning Process, Defining Software Scope and Checking Feasibility, Resources management, Reusable Software Resources, Environmental Resources, Software Project Estimation, Decomposition Techniques, Software Sizing, Problem-Based Estimation, LOC-Based Estimation, FP-Based Estimation, Object Point (OP)-based estimation, Process-Based Estimation, Estimation with Use Cases, Use- Case - Based Estimation, Reconciling Estimates, Empirical Estimation Models, The

Structure of Estimation Models, The COCOMO II Mode, Preparing Requirement Traceability Matrix **Project Scheduling :** Project Scheduling, Defining a Task for the Software Project, Scheduling. **Suggested Free Open Source Tools :** Gantt Project, Agantty, Project Libre. **(Chapter 3)**

Unit IV Design Engineering

Design Concepts : Design within the Context of Software Engineering, The Design Process, Software Quality Guidelines and Attributes, Design Concepts - Abstraction, Architecture, design Patterns, Separation of Concerns, Modularity, Information Hiding, Functional Independence, Refinement, Aspects, Refactoring, Object-Oriented Design Concept, Design Classes, The Design Model , Data Design Elements, Architectural Design Elements, Interface Design Elements, Component-Level Design Elements, Component Level Design for Web Apps, Content Design at the Component Level, Functional Design at the Component Level, Deployment-Level Design Elements. **Architectural Design :** Software Architecture, What is Architecture, Why is Architecture Important, Architectural Styles, A brief Taxonomy of Architectural Styles. **Suggested Free Open Source Tool :** Smart Draw **(Chapter 4)**

Unit V Risks and Configuration Management

Risk Management : Software Risks, Risk Identification, Risk Projection, Risk Refinement, Risk Mitigation, Monitoring, and Management, The RMMM Plan. **Software Configuration Management :** Software Configuration Management, The SCM Repository The SCM Process, Configuration Management for any suitable software system. **Suggested Free Open Source Tools :** CF Engine Configuration Tool, Puppet Configuration Tool. **(Chapter 5)**

Unit VI Software Testing

A Strategic Approach to Software Testing, Verification and Validation, Organizing for Software Testing, Software Testing Strategy - The Big Picture, Criteria for Completion of Testing, Strategic Issues, Test Strategies for Conventional Software, Unit Tesing, Integration Testing, Test Strategies for Object-Oriented Software, Unit Testing in the OO Context, Integration Testing in the OO Context, Test Strategies for WebApps, Validation Testing, Validation -Test Criteria, Configuration Review. **Suggested Free Open Source Tools :** Selenium, JUnit. **(Chapter 6)**

TABLE OF CONTENTS

Unit - I

Unit - II

Unit - III

Unit - IV

PART II : ARCHITECTURAL DESIGN

Unit – V

Unit – VI

UNIT - I

1 Introduction to Software Engineering and Process Models

Syllabus

Software Engineering Fundamentals : *Introduction to software engineering, The Nature of Software, Defining Software, Software Engineering Practice.*

Software Process : *A Generic Process Model, defining a Framework Activity, Identifying a Task Set, Process Patterns, Process Assessment and Improvement, Prescriptive Process Models, The Waterfall Model, Incremental Process Models, Evolutionary Process Models, Concurrent Models, A Final Word on Evolutionary Processes. Unified Process, Agile software development: Agile methods, plan driven and agile development.*

Contents

Part I : Software Engineering Fundamentals

1.1 Introduction to Software Engineering

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

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

The **definition** of software engineering is based on two terms :

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

1.2 The Nature of Software

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

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

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

1.3 Defining Software

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

Software products may be :

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

1.4 Software Engineering Practice

Following are some principles that focus on software engineering practice. These are the principles that are suggested by David Hooker

1. **Reason for Existence :** The intention of software system is to provide value to its users.
2. **Keep It Simple Stupid(KISS) :** Software design must be simple to implement and the resultant software system must be more maintainable and less error-prone.
3. **Vision :** The goal and objective of the software must be defined.
4. **The produce is getting consumed :** While developing the software system, the documentation must be maintained so that the maintenances of the system becomes easy.
5. **Be Open to future :** If some changes need to be incorporated in the system, then those must be accommodated easily.
6. **Plan for Reuse :** The software system components can be made reusable so that the time and efforts can be saved. The programming techniques like Object oriented programming can be used for it.
7. **Think :** Before performing every software engineering activity think and understand it.

1.5 Software Characteristics

SPPU : May-11,14,19, Aug.-17, Oct.-18, Dec.-16, Marks 10

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

• Software is engineered, not manufactured

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

- **Software does not ware out**

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

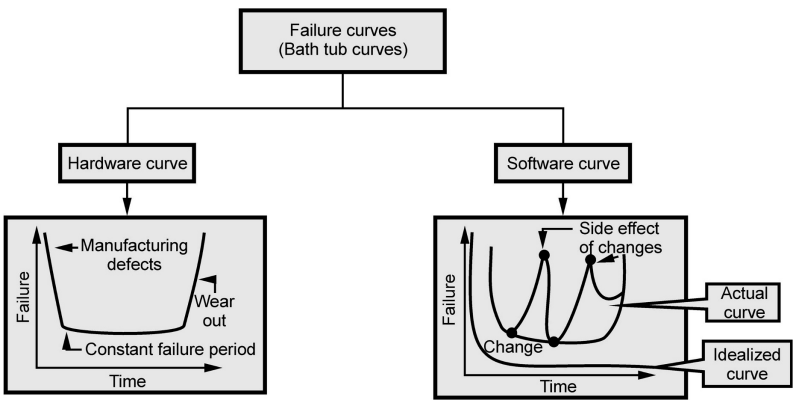


Fig. 1.5.1 Failure curves for hardware and software

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

- **Most software is custom built rather than being assembled from components**

1) While developing any hardware product firstly the circuit design with desired functioning properties is created. Then required hardware components such as ICs, capacitors and registers are assembled according to the design, but this is not done while developing software product.
2) Most of the software is custom built.

3) However, now the software development approach is getting changed and we look for reusability of software components.
4) It is practiced to reuse algorithms and data structures.
5) Today software industry is trying to make library of reusable components. **For example :** In today's software, GUI is built using the reusable components such as message windows, pull down menus and many more such components.
6) The approach is getting developed to use in-built components in the software. This stream of software is popularly known as *component engineering*.

Difference between Hardware Engineering and Software Engineering

Sr. No.	Hardware Engineering	Software Engineering
1.	Hardware products are manufactured. The quality problems that occur in hardware manufacturing phase can not be removed easily.	Software is engineered and not manufactured. During software development process, various problems can be identified and rectified.
2.	In early stage of hardware development process, the failure rate is very high because of manufacturing defects. But after correcting such defects the failure rate gets reduced. The failure rate remains constant for some period of time and again it starts increasing because of environmental maladies.	On the other hand, software does not get affected from the environmental maladies. But due to some undiscovered errors the failure rate is high and drops down as soon as the errors get corrected.
3.	There are spare parts available for the hardware components.	There are no spare parts for software components to replace.
4.	The hardware unit is assembled from components.	Most software is custom built rather than being assembled from components.

Review Questions

1. Define software engineering. What are the software characteristics ? What are the various categories of software ? **SPPU : May-14, Marks 10**
2. Define software engineering. How software engineering is different from hardware engineering ? Justify. **SPPU : May-11, Marks 8**
3. What is software engineering ? What are the characteristics of software ? **SPPU : Dec.-16, End Sem, Marks 5**
4. Define terms 'Software' and 'Software engineering'. "Software does not wear out" State whether this statement is true or false. Justify your answer. **SPPU : Aug.-17, Oct.-18, In Sem, Marks 3, May -19, End Sem, Marks 6**

1.6 Software Engineering Myths

SPPU : April-16, Aug. 17, Dec.-17, 18,19, Oct.-19, Marks 5

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

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

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

Review Questions

1. What are customer myths ? Discuss the reality of these myths.
   SPPU : Aug-17, In Sem, Marks 3
2. What are the practitioner's myths ? Discuss the reality of these myths.
   SPPU : April-16, In Sem, Marks 4, Dec.-17, Dec.-18, End Sem, Marks 5
3. List and explain practitioner's myth and its realities **SPPU : Oct.-19, In Sem, Marks 5**

Part II : Software Process

1.7 Generic Process Model

SPPU : Dec.-19, Marks 7

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

The fundamental activities are :

- Specification
- Design and implementation
- Validation
- Evolution

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

1.7.1 Layered Technology

SPPU : May-13, Marks 8

- Software engineering is a layered technology. Any software can be developed using these layered approaches. Various layers on which the technology is based are **quality management layer, process layer, methods layer, tools layer**.

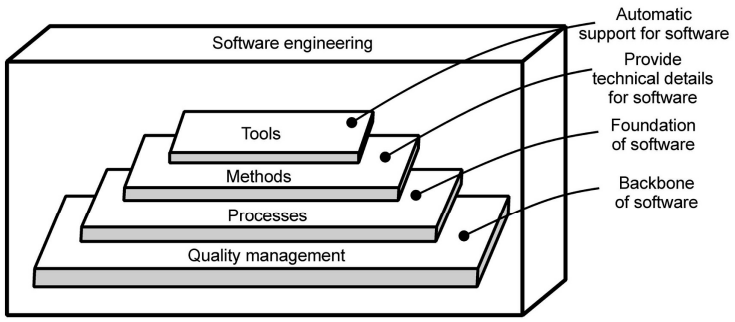


Fig. 1.7.1 Software engineering : A layered approach

- A disciplined **quality management** is a backbone of software engineering technology.
- **Process layer** is a foundation of software engineering. Basically, process defines the framework for timely delivery of software.
- In **method layer** the actual method of implementation is carried out with the help of requirement analysis, designing, coding using desired programming constructs and testing.
- Software **tools** are used to bring automation in software development process.
- Thus software engineering is a **combination** of process, methods, and tools for development of quality software.

Review Question

1. Elaborate how software engineering is a layered technology.

SPPU : Dec.-19, End Sem, Marks 7

1.8 Defining Framework Activity

SPPU : May-16, Marks 7

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

As shown in Fig. 1.8.1, the software process is characterized by process framework activities, task sets and umbrella activities.

Process framework activities

- Communication
  - By communicating customer requirement gathering is done.
- Planning - Establishes engineering work plan, describes technical risks, lists resource requirements, work products produced and defines work schedule.

- Modeling - The software model is prepared by :
  - Analysis of requirements
  - Design
- Construction - The software design is mapped into a code by :
  - Code generation
  - Testing
- Deployment - The software delivered for customer evaluation and feedback is obtained.

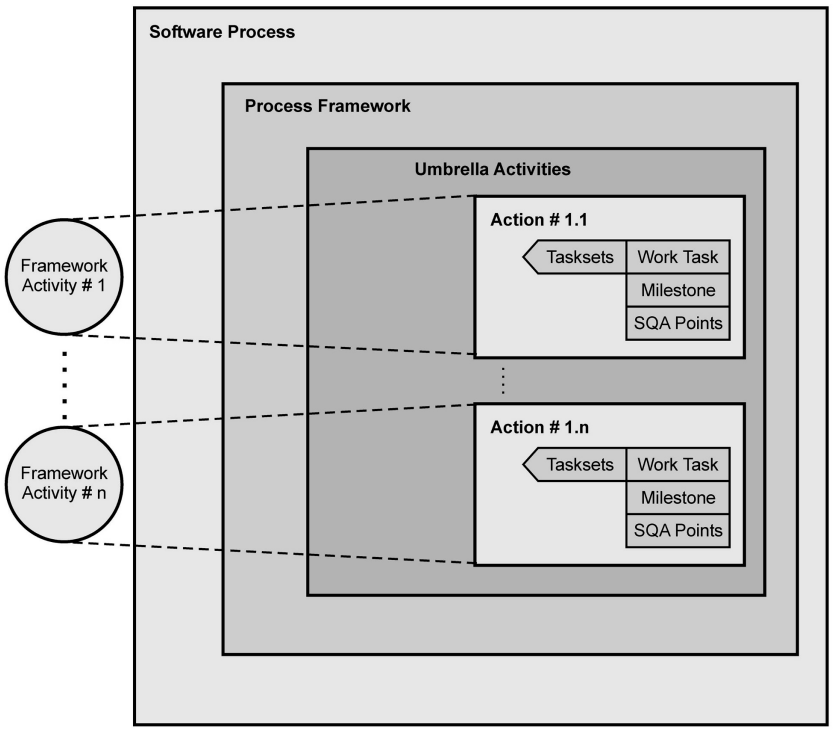


Fig. 1.8.1 Software process framework

Review Question

1. What are the various umbrella activities applied throughout a software project ?

SPPU : May-16, End Sem, Marks 7

1.9 Identifying a Task Set

SPPU : Dec.-11, April-15, 16, Aug.-17, May-18, Oct.-19, Marks 8

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

- Collection of software engineering work tasks

- Project milestones
- Software quality assurance points

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

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

Review Questions

1. Explain the generic framework activities in a software engineering process and also mention the umbrella activities. **SPPU : Dec.-11, Marks 8**
2. What are various umbrella activities applied throughout a software project ? **SPPU : April-15, In Sem, Marks 6, April-16, In Sem, Marks 7, Aug.-17, In Sem, Marks 4**
3. What is software process framework ? Explain in detail
4. What are the reasons to have a software process ? What are the issues addressed by umbrella activities in layered model of software engineering. **SPPU : May-18, End Sem, Oct.-19, In Sem, Marks 5**
5. What are the issues addressed by umbrella activities in layered model of software engineering. **SPPU : Oct.-19, In Sem, Marks 5**

1.10 Process Pattern

- Process is defined as a series of activities in which one or more inputs are used to produce one or more outputs.
- The **process pattern** is a **template** which appears as a general solution to a common problem.

- Process pattern acts as a consistent method for describing an important characteristic of software process.
- Using process pattern we can easily build the process that satisfies all the requirements.
- Process pattern describes -
  - Complete process
  - Important framework activity
  - Task within the framework activity
- Scott Ambler - an object oriented consultant has proposed a **template** for process pattern as follows :

• Pattern name	• Intent
• Type	• Initial context
• Problem	• Solution
• Resulting context	• Known uses

The description of process pattern is as follows -

Pattern name

The pattern name should be a meaningful name given to the pattern. From pattern name one can guess its functionality.

Intent

The objective or the purpose of the pattern should be described here.

Type

The type of pattern should be specified here.

Ambler has suggested **three types** of patterns

1) **Task Pattern -** It represents the software engineering action or work task which is a part of process. For example, **Formal Technical review** is a task pattern.
2) **Stage Pattern -** It defines the process framework activity. A framework activity has multiple work tasks; hence stage pattern consists of multiple task patterns. For example, **Coding phase** is a stage pattern.
3) **Phase Patterns -** It defines the sequence of framework activities. For example the phrase pattern can be **spiral model or rapid prototype model.**

Initial context

In this section the conditions under which the pattern applies are described.

Sometimes the **entry conditions** must be true before the process begins.

In this section following issues need to described :

1. The set of organisational or team related activities that have already occurred.
2. The list of entry state processed.
3. Already existing software engineering or project related information.

Problem

Under this section the problem is mentioned for which the pattern is to be described. For example : *insufficient requirements* is a problem. That means customers are not sure about what they want exactly. They could not specify the requirements in proper manner.

Solution

Every problem for which pattern has to be described should be accompanied with some solution. For example: The problem of *insufficient requirements* has solution. That is - Establish effective communication with the customer. Ask questions in order to obtain meaningful requirements. Conduct timely reviews to modify/redefine the requirements. This solution will help the software developer to get useful information before the actual work starts.

Resulting context

It describes the results after successful implementation of pattern. The resulting context should have following type of information on successful completion of pattern -

1. The team-related or organizational activities that must have occurred.
2. Exit state for the process.
3. The software engineering information or project information that has been developed.

Known uses/Examples

The specific instances or applications in which the described pattern is useful should be mentioned under this section. In other words we describe applicability of the pattern. For example : spiral model is useful for the large scale projects in which work products must be examined in several iterations.

1.11 Process Assessment and Improvement

Normally process is suffered by following problems -

1. The software has to be delivered on time.
2. The software should satisfy customer needs and requirements.
3. The software should posses the long term quality characteristics.

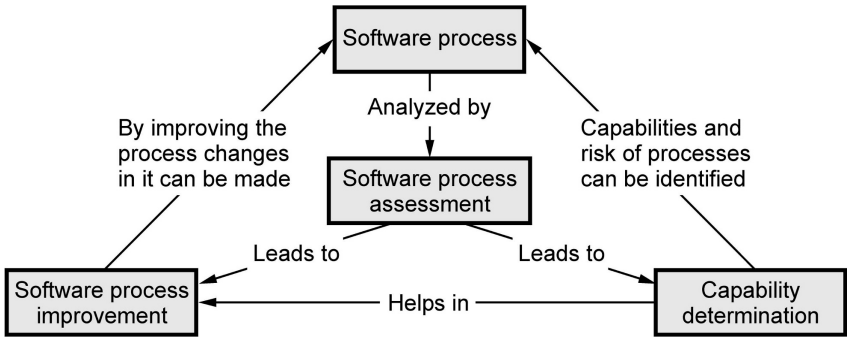


Fig. 1.11.1 Software process assessment

Process assessment is an activity in which it is ensured whether the software meets the basic criteria for successful software engineering project.

Following approaches are used for software assessment -

Standard CMMI assessment method for process improvement

It is a five step process assessment model. These five steps are initiating, diagnosing, establishing, acting and learning. This model makes use of SEI CMM as the base model.

CMM-based appraisal for internal process improvement

This method provides the diagnostic technique for internal process assessment.

SPICE

Using this standard all the requirements are analyzed for software process assessment. This standard helps in doing an objective evaluation on efficiency of any process.

ISO 9001:2000

This is a popularly used standard for improving the overall quality of the organization. The International Organization for Standardization i.e. ISO has developed this standard.

1.12 Prescriptive Process Model

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

- Various prescriptive process models are as shown in Fig. 1.12.1.

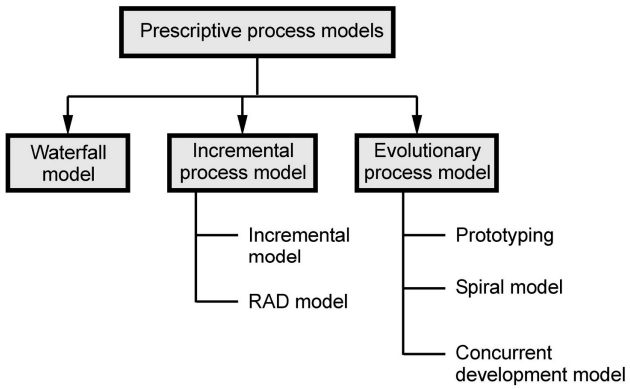


Fig. 1.12.1 Prescriptive process model

1.12.1 Need for Process Model

- The software development team must decide the process model that is to be used for software product development and then the entire team must adhere to it. This is necessary because the software product development can then be done systematically.
- Each team member will **understand** - **what is the next activity** and **how** to do it. Thus **process model** will bring the **definiteness and discipline** in overall development process.
- Every process model consists of **definite entry and exit criteria** for **each phase**. Hence the transition of the product through various **phases is definite**.
- If the process model is not followed for software development then any team member can perform any software development activity, this will ultimately cause a chaos and software project will definitely fail without using process model, it is difficult to monitor the progress of software product. Thus process model plays an important rule in software engineering.

1.13 The Waterfall Model

SPPU : April-16, Dec.-18, Oct.-19, Marks 6

- The waterfall model is also called as '**linear-sequential model**' or '**classic life cycle model**'. It is the oldest software paradigm. This model suggests a systematic, **sequential approach** to software development.
- The software development starts with **requirements gathering phase**. Then progresses through **analysis**, **design**, **coding**, **testing** and **maintenance**. Following figure illustrates waterfall model.

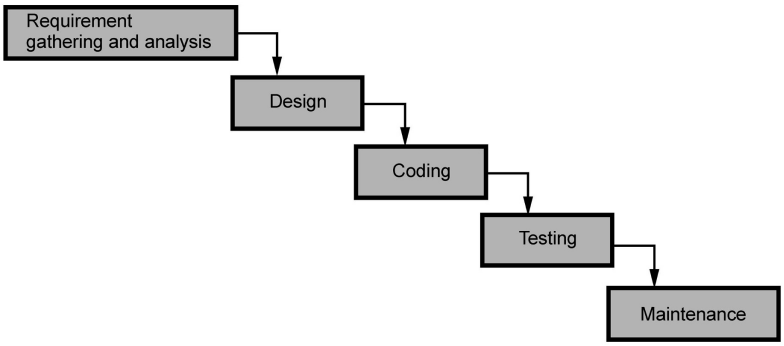


Fig. 1.13.1 Waterfall model

- In **requirement gathering and analysis** phase the **basic requirements** of the system must be understood by software engineer, who is also called **Analyst**. The **information domain**, **function**, **behavioural requirements** of the system are understood. All these requirements are then **well documented** and discussed further with the customer, for reviewing.
- The **design** is an intermediate **step between** requirements analysis and coding. Design focuses on program attributes such as -
  - Data structure
  - Software architecture
  - Interface representation
  - Algorithmic details

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

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

Benefits of waterfall model

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

Drawbacks of waterfall model

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

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

Example 1.13.1 *Explain how water-fall model is applicable for the development of the following systems :*

a) University accounting system

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

Solution :

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

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

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

Example 1.13.2 *What is meant by 'blocking states' in linear sequential model ?*

Solution :

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

Review Questions

1. What are the elements of waterfall model ? State its merits and demerits.
   SPPU : April-16, In Sem, Marks 6
2. Explain classic life cycle paradigm for software engineering and problems encountered when it is applied. **SPPU : Dec.-18, End Sem, Marks 5**
3. Why waterfall model of software engineering is not accurate reflection of software development activities? Justify. **SPPU : Dec.-18, End Sem, Oct.-19, In Sem, Marks 5**

1.14 Increment Process Model

SPPU : May-18, May-19, Dec.-19, Marks 6

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

1.14.1 Incremental Model

- The incremental model has same phases that are in waterfall model. But it is iterative in nature. The incremental model has following phases.
  1. Analysis 2. Design
  3. Code 4. Test
- The incremental model delivers series of releases to the customer. These releases are called **increments**. More and more functionality is associated with each increment.
- The first increment is called **core product**. In this release the basic requirements are implemented and then in subsequent increments new requirements are added.

- The word processing software package can be considered as an example of incremental model. In the first increment only the document processing facilities are available. In the second increment, more sophisticated document producing and processing facilities, file management functionalities are given. In the next increment spelling and grammar checking facilities can be given. Thus in incremental model progressive functionalities are obtained with each release.

When to choose it ?

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

Merits of incremental model

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

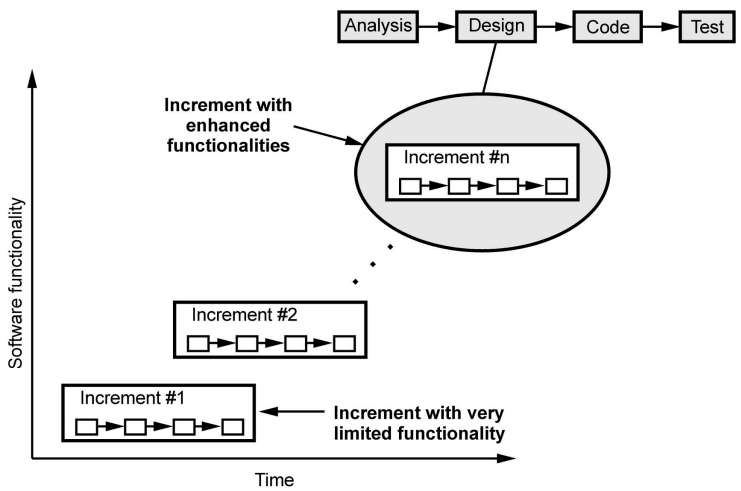


Fig. 1.14.1 The incremental model

Demerits

1) System structure tends to degrade when new increment is getting added to the existing system. This actually spends time and money on refactoring to improve the software.
2) It is not cost effective to produce document for every version of the system, hence many times the complete process is not getting documented.

1.14.2 RAD Model

- The RAD model is a type of incremental process model in which there is **extremely short development cycle.**
- When the requirements are fully understood and the **component based construction** approach is adopted then the RAD model is used.
- Using the RAD model the fully functional system can be developed within **60 to 90 days.**
- Various phases in RAD are Requirements Gathering, Analysis and Planning, Design, Build or Construction and finally Deployment.
- Multiple teams work on developing the software system using RAD model parallely.
- In the **requirements gathering** phase the developers communicate with the users of the system and understand the business process and requirements of the software system.
- During **analysis and planning** phase**,** the analysis on the gathered requirements is made and a planning for various software development activities is done.

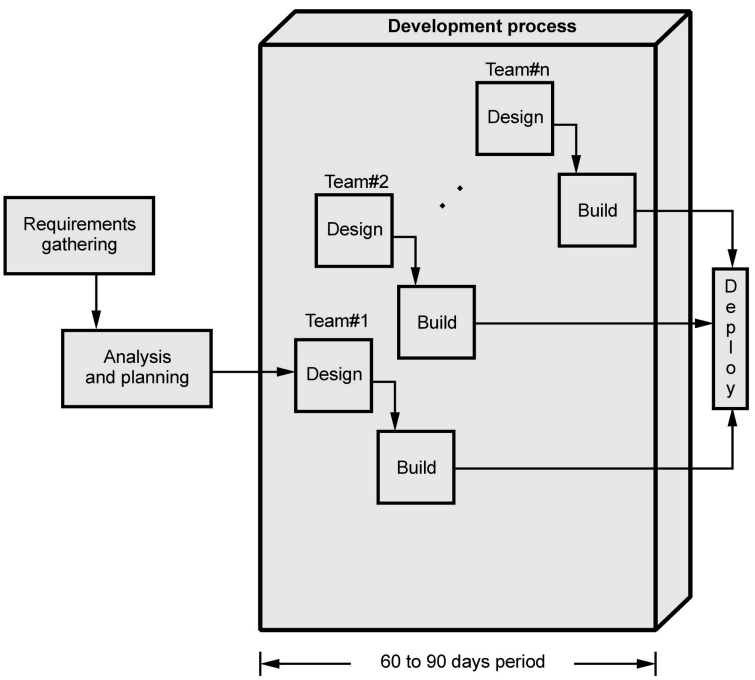


Fig. 1.14.2 Rapid application development

- During the **design** phase various models are created. Those models are Business model, data model and process model.

- The **build** is an activity in which using the existing software components and automatic code generation tool the implementation code is created for the software system. This code is well tested by its team. The functionalities developed by all the teams are integrated to form a whole.
- Finally the deployment of all the software components (created by various teams working on the project) is carried out.

Drawbacks of rapid application development

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

Example 1.14.1 *Which type of applications suit RAD model ? Justify your answer.*

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

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

Review Questions

1. Explain with neat diagram incremental model and state its advantages and disadvantages **May-18, End Sem, Marks 6**
2. What are the conditions in which rapid application development model is preferred? **May-19, End Sem, Marks 5**
3. Explain RAD model with the help of diagram **Dec.-19, End Sem, Marks 6**

1.15 Evolutionary Process Model

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

1.15.1 Prototyping

- In prototyping model initially the requirement gathering is done.
- Developer and customer define overall objectives; identify areas needing more requirement gathering.
- Then a quick design is prepared. This design represents what will be visible to user in input and output format.
- From the quick design a prototype is prepared. Customer or user evaluates the prototype in order to refine the requirements. Iteratively prototype is tuned for satisfying customer requirements. Thus prototype is important to identify the software requirements.
- When working prototype is built, developer use existing program fragments or program generators to throw away the prototype and rebuild the system to high quality.
- Certain classes of mathematical algorithms, subset of command driven systems and other applications where results can be easily examined without real time interaction can be developed using prototyping paradigm.

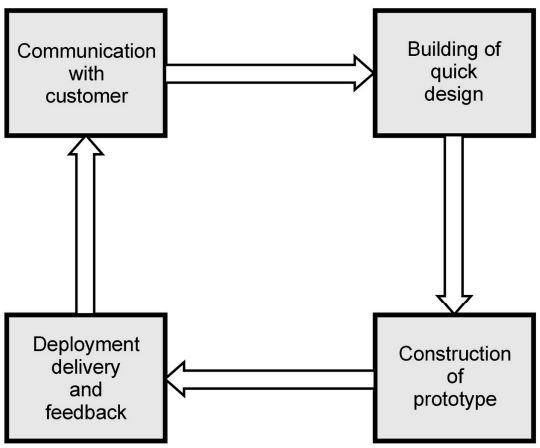


Fig. 1.15.1 Prototyping

When to choose it ?

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

Merits

1. The development team has adequate domain knowledge.
2. All the end users are involved in all phases of development.
3. There is use of reusable components.

Drawbacks of prototyping

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

Comparison between prototyping and incremental process model

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

1.15.2 Spiral Model

- This model possess the **iterative nature** of prototyping model and controlled and **systematic approaches** of the linear sequential model.
- This model gives efficient development of incremental versions of software. In this model, the software is developed in series of increments.

- The spiral model is divided into a number of **framework activities**. These framework activities are denoted by **task regions**.
- Usually there are **six tasks regions**. The spiral model is as shown in Fig. 1.15.2.
- Spiral model is realistic approach to development of **large-scale systems** and software. Because customer and developer better understand the problem statement at each evolutionary level. Also risks can be identified or rectified at each such level.
- In the initial pass, product specification is built and in subsequent passes around the spiral the prototype gets developed and then more improved versions of software gets developed.

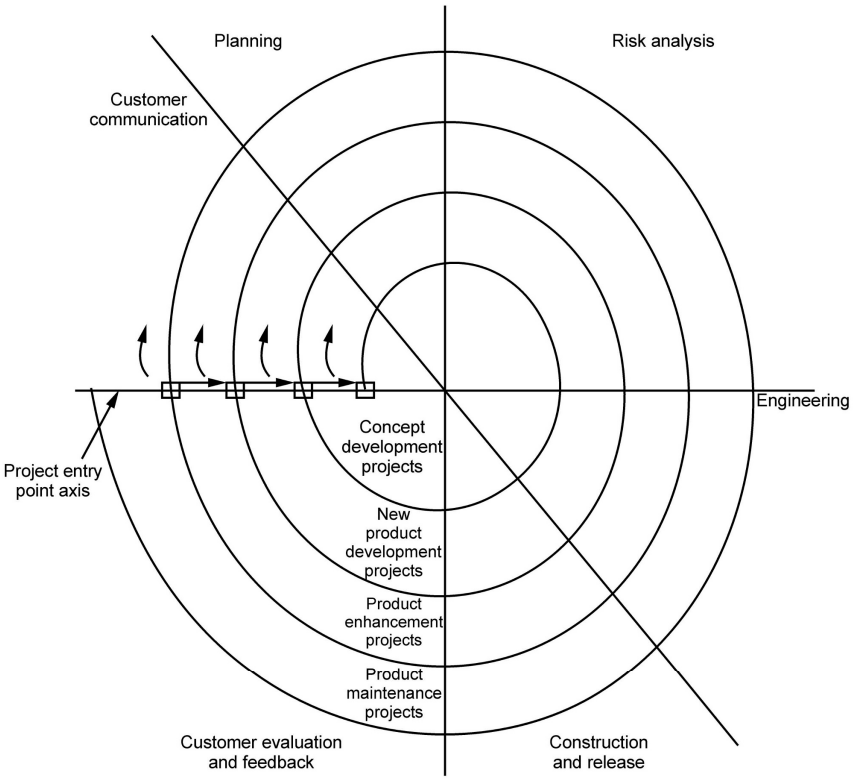


Fig. 1.15.2 Spiral model

- During planning phase, the cost and schedule of software can be planned and adjusted based on feedback obtained from customer evaluation.
- In spiral model, **project entry point axis** is defined. This axis represents starting point for **different types of projects**.

For instance, concept development project will start at core of spiral and will continue along the spiral path. If the concept has to be developed into actual project then at entry point 2 the product development process starts. Hence entry point 2 is called product development project entry point. The development of the project can be carried out in iterations.

- The task regions can be described as :

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

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

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

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

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

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

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

Advantages of spiral model

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

Drawbacks of spiral model

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

When to choose it ?

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

Comparison between Spiral model and Prototyping model

Sr. No.	Spiral model	Prototyping model
1.	The development team with less domain knowledge can be accommodated due to iterative nature of this model. The change in technology in the later phase can not be tolerated.	The development team has adequate domain knowledge. Similarly they can adopt the new technologies if product demands.
2.	All the end-users need not be involved in all the phases of development.	All the end-users are involved in all phases of development.
3.	Funding are not stable for the projects that can be developed using spiral model.	Funding are stable for these type of projects.
4.	The requirements that are gathered and analyzed are high reliability requirements.	Some requirements are gathered initially, but there may be change in requirements when the working prototype is shown to the customer.

Example 1.15.1 *As you move outward along with process flow path of the spiral model, what can we say about the software that is being developed or maintained ?*

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

Example 1.15.2 *How does "Project Risk" factor affect the spiral model of software development ?*

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

Example 1.15.3 *How does a spiral model represent a process suitable to represent a real time problem ?*

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

1. Software evolves as the project progresses. And at every evolutionary level the risks are identified and managed and risks are reduced at every stage.

2. It enables the developer to apply the prototype approach at any stage in the evolution of the product. It helps in adopting the approach systematic stepwise development of the product.
3. The iterative frameworks help in analyzing the product at every evolutionary stage.
4. The spiral model demands a direct consideration of technical risks at all stages of project. The risks are reduced before they get problematic.

1.16 Comparison between Various Process Models

SPPU : May-11,12,13,14, Dec.-12,13, Marks 8

1. Evolutionary and Incremental Process Model

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

2. Waterfall Model and Spiral Model

Sr. No.	Waterfall model	Spiral model
1.	It requires well **understanding of requirements** and familiar technology.	It is developed in **iterations**. Hence the requirements can be identified at new iterations.
2.	**Difficult** to accommodate **changes** after the process has started.	The required changes can be made at every stage of **new version**.
3.	Can accommodate iteration but indirectly.	It is iterative model.
4.	**Risks** can be identified **at the end** which may cause failure to the product.	**Risks** can be identified and **reduced** before they get problematic.

5.	The customer can see the **working model** of the project only **at the end**. After reviewing of the working model; if the customer gets dissatisfied then it causes serious problems.	The customer can see the **working product** at **certain stages** of iterations.
6.	Customers prefer this model.	Developers prefer this model.
7.	This model is good for **small systems**.	This model is good for **large systems**.
8.	It has **sequential** nature.	It has **evolutionary** nature.

3. Waterfall, Spiral, Prototype and Incremental Model

Waterfall model	Spiral model	Prototyping model	Incremental model
Requirements must be clearly understood and defined at the beginning only.	The **requirements** analysis and gathering can be done in iterations because requirements get changed quite often.	**Requirements analysis** can be made in the later stages of the development cycle, because requirements get changed quite often.	The **requirements analysis** can be made in the later stages of the development cycle.
The **development team** having the adequate experience of working on the similar project is chosen to work on this type of process model	The **development team** having less experience of working on the similar projects is allowed in this process model	The **development team** having less experience of working on the similar projects is allowed in this process model.	The **development team** having the adequate experience of working on the similar project is chosen to work on this type of process model.
There is **no user involvement** in all the phases of development process.	There is **no user involvement** in all the phases of development process.	There is **user involvement** in all the phases of development process.	There is **user involvement** in all the phases of development process.

When the requirements are reasonably well defined and the development effort suggests a purely linear effort then the **waterfall model is chosen.**	Due to iterative nature of this model, the risk identification and rectification is done before they get problematic. Hence for handling real time problems the **spiral model is chosen**.	When developer is unsure about the efficiency of an algorithm or the adaptability of an operating system then the **prototyping model is chosen**.	When the requirements are reasonably well defined, the development effort suggests a purely linear effort and when limited set of software functionality is needed quickly then the **incremental model is chosen.**

Review Questions

1. What is software process model ? Explain the incremental process model.
   SPPU : May-14, Marks 8
2. Explain evolutionary process models mentioning the types of projects for which they are suitable. **SPPU : May-12, Marks 6**
3. Explain the waterfall model with its advantages and disadvantages.
   SPPU : Dec.-12, Marks 6
4. Explain the waterfall model with work products of each activity. **SPPU : May-13, Marks 8**
5. Explain the prototyping model with its advantages and disadvantages.
   SPPU : Dec.-13, Marks 6
6. Give the need of different process models in software development.
   SPPU : Dec.-13, Marks 4
7. What do you mean by evolutionary process models ? Explain spiral model as an evolutionary process model. **SPPU : May-11, Marks 8**

1.17 Concurrent Models

- The concurrent development model is also called as **concurrent engineering.**
- In this model, the framework activities or software development tasks are represented as **states.**
- The **modeling** or **designing** phase of software development can be in one of the states like *under development*, *waiting for modification*, *under revision* or *under review* and so on.
- Fig. 1.17.1 represents these states. (Refer Fig. 1.17.1)

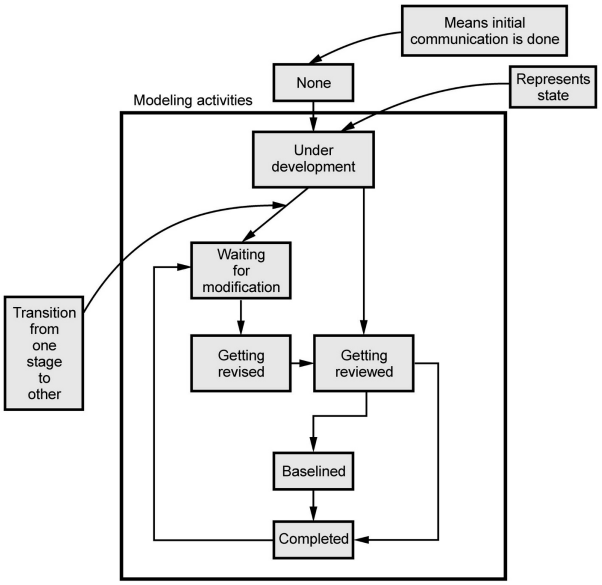


Fig. 1.17.1 Concurrent development model

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

Advantages

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

1.18 Unified Process

The unified process is a framework for object oriented models. This model is also called as **Rational Unified Process model (RUP)**. It is proposed by Ivar Jacobson, Grady Bootch and James Rumbaugh. This model is iterative and incremental by nature. Let us discuss various phases of unified process.

There are 5 phases of unified process model and those are -

- Inception
- Elaboration
- Construction
- Transition
- Production

Let us understand each of these phases in detail.

Inception

- In this phase there are two major activities that are conducted : ***Communication*** and ***planning***.
- By having customer communication business requirements can be identified.
- Then a rough architecture of the system is proposed. Using this rough architecture it then becomes easy to make a plan for the project.
- **Use cases** are created which elaborates the user scenario.
- Using use cases the sequence of actions can be identified.
- Thus use cases help to identify the scope of the project which ultimately proves to be the basis for the plan.

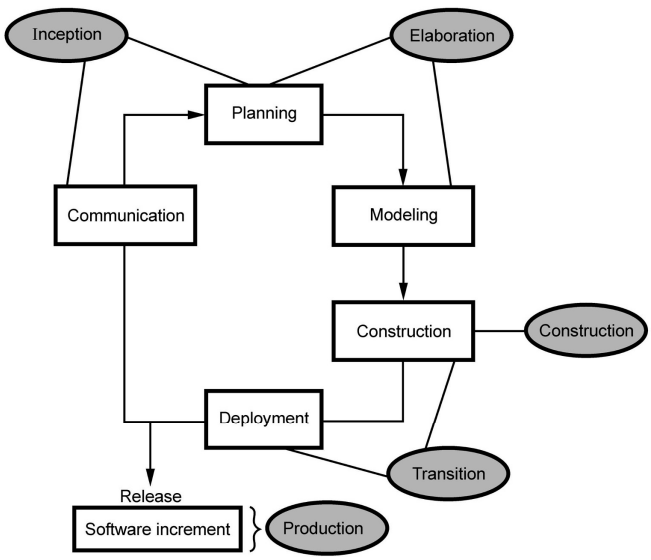


Fig. 1.18.1 Unified process model

Elaboration

- Elaboration can be done using two activities : ***Planning*** and ***Modelling***.
- In this phase the use cases are redefined. And an architectural representation is created using five models such as use-case model, analysis model, design model, implementation model and deployment model.

- Thus **executable baseline** gets created.
- Then a plan is created carefully to check whether scope, risks and delivery dates are reasonable.

Construction

- The main activity in this phase is to make the use cases **operational**.
- The analysis and design activities that are started in elaboration phase are completed in this phase and a **source code** is developed which implements all desired functionalities.
- Then unit testing is conducted and acceptance testing is carried out on the use cases.

Transition

- In the transition phase all the activities that are required at the time of deployment of the software product are carried out.
- Beta **testing** is conducted when software is delivered to the end user.
- **User feedback** report is used to remove defects from the created system.
- Finally software team prepares user manuals, installation guides and trouble shooting procedures. This makes the software more usable at the time of release.

Production

- This is the final phase of this model. In this phase mainly the maintenance activities are conducted in order to support the user in operational environment.
- Various **work products** that may get generated in every phase are as given below -

Inception phase	Elaboration phase	Construction phase	Transition phase
• Initial use case model. • Initial risk assessment. • Project plan.	• Use case model. • Requirements analysismodel • Architecture model • Preliminary design model • Risk list • Iterative project plan • Preliminary user manual	• Design model • Software components • Test plan • Test cases • User manual • Installation manual	• Delivered software increments • Beta test report • User feedback report.

- Thus the generic software process models are applied for many years in software development process in order to reduce chaos in the process of development.
- Each of these models suggest different process flow but they insist on performing the same set of generic framework activities.

1.19 Agile Software Development

SPPU : Aug.-17, Dec.-17, 18, Oct.-19, May-19, Marks 5

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

Conventional Software Development Methodology

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

Agile Methodology

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

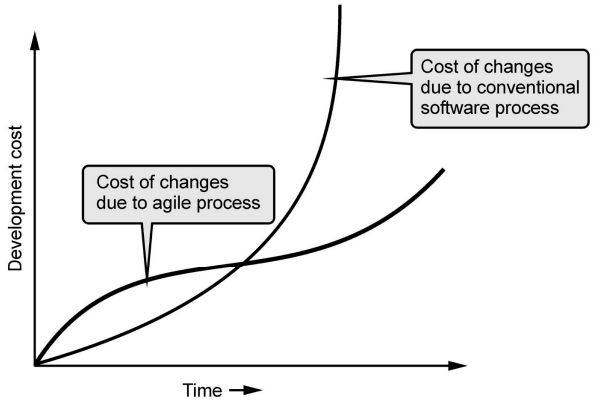


Fig. 1.19.1 Influence of software development approach on agile process

1.19.1 Agility Principles

There are famous 12 principles used as agility principles -

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

1.19.2 Concept of Agile Process

Agile process is based on following assumptions about software projects

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

Characteristics of Agile process are -

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

1.19.3 Software Evolution and Merits and Demerits of Agile Methods

Software evolution : Software evolution is a process of developing a software initially and repeatedly updating it for various reasons. Software change occurs because of **following reasons**.

1. New requirements emerge when the software is used.
2. The business environment changes.
3. Errors needs to be repaired.
4. New equipment must be accommodated.
5. The performance or reliability may have to be improved.

Agile process model

Merits :

1) Customer satisfaction can be attained by rapid and continuous delivery of useful software.
2) Customer, developer and tester interact with each other during software development process.
3) Continuous attention can be given for excellent technical design and software quality.
4) Even late changes in requirements can be accommodated.

Demerits :

1) There is lack of emphasis on necessary designing and documentation during software development process.
2) The project can easily get off the track if customer is not clear about his requirements.

Review Questions

1. Describe agile manifesto. **SPPU : Aug.-17, In Sem, Marks 04**
2. What is agility? Explain about process model. **SPPU : Dec.-17, 18, End Sem, Marks 5**
3. What is the importance of agile/XP methodology for project development? **SPPU : Oct.-19, In Sem, Dec.-18, May-19, End Sem, Marks 5**

1.20 Agile Methods

There are various agile process models -

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

Let us discuss them in detail.

1.20.1 Adaptive Software Development (ASD)

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

  1. Speculation 2. Collaboration 3. Learning.

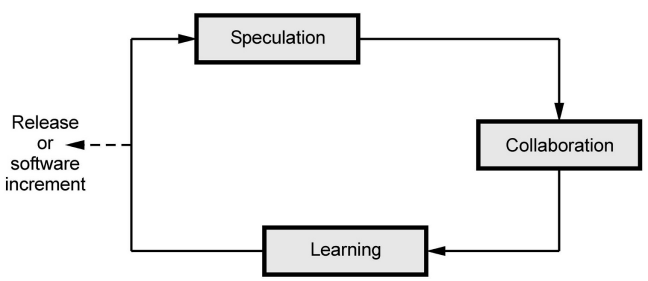


Fig. 1.20.1 Adaptive software development life cycle

1. **Speculation :** This is an initial phase of the adaptive software development process. In this phase the **adaptive cycle planning** is conducted. In this cycle planning mainly three types of information is used such as - Customer's mission statement, project constraints (delivery date, user description, budgets and so on) and basic requirements of the project.
2. **Collaboration :** The motivated people work in collaboration to develop the desired software product. In this phase **collaboration among the members** of development team is a key factor. For successful collaboration and coordination it is necessary to have following qualities in every individual -
   - Assist each other without resentment
   - Work hard.
   - Posses the required skill set.
   - Communicate problems and help each other to accomplish the given task.
   - Criticize without any hate.
3. **Learning :** As the team members go on developing the components, the emphasize is on learning new skills and techniques. There are three ways by which the team members learn-
   - **Focus groups :** The feedback from the end-users is obtained about the software component being developed. Thus direct feedback about the developed component can be obtained.
   - **Formal technical review :** This review for software components is conducted for better quality.
   - **Postmortems :** The team analyses its own performance and makes appropriate improvements.

1.20.2 Dynamic System Development Method (DSDM)

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

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

Various phases in this life cycle model are as follows -

1. **Feasibility study :** By analyzing the business requirements and constraints the viability of the application is determined in this phase.
2. **Business study :** The functional and informational requirements are identified and then the business value of the application is determined. The basic application architecture is decided in this phase.

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

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

1.20.3 Scrum

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

Principles

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

Development Activities

In SCRUM emphasize is on **software process pattern.** The software process pattern defines a set of development activities. Refer Fig. 1.20.2.

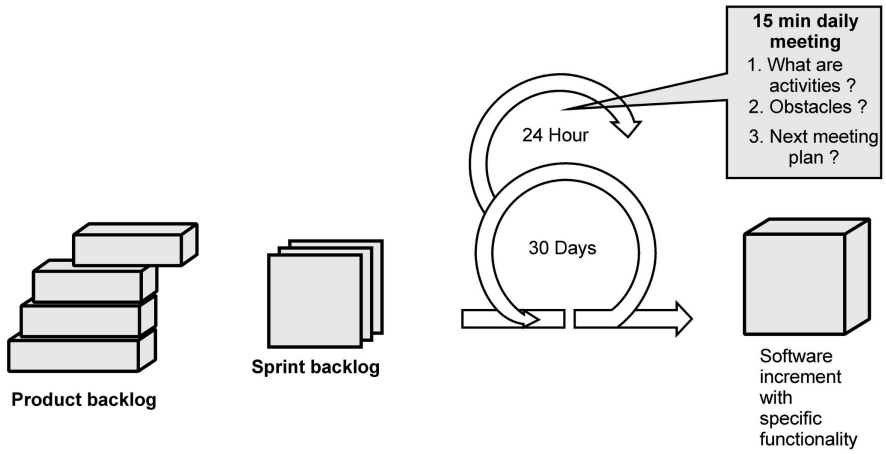


Fig. 1.20.2 SCRUM workflow activities

Various development activities in SCRUM are -

1. **Backlog :** It is basically **a list** of project requirements or features that must be provided to the customer. The items can be included in the backlog list at any time. The product manager analyses this list and updates the priorities as per the requirements.
2. **Sprint :** These are the **work units** that are needed to achieve the requirements mentioned in the backlogs. Typically the sprints have **fixed duration or time-box** (typically of 2 to 4 weeks). Thus sprints allow the team members to work in stable and short-term environment.
3. **Meetings :** These are 15 minutes daily meetings to report the completed activities, obstacles and plan for next activities. Following are three questions that are mainly discussed during the meetings
   i) What are the tasks done since last meeting ?
   ii) What are the issues (obstacles) that team is facing ?
   iii) What are the next activities that are planned ?
4. **Demo :** During this phase, the software increment is delivered to the customer. The implemented functionality which is demonstrated to the customer. Note that demo focuses on only implemented functionalities and not all the planned functionalities (and yet to get implemented) of the software product.

Roles :

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

Advantages and Disadvantages :

Advantages :

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

Disadvantages :

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

1.20.4 Feature Driven Development (FDD)

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

In FDD, the collaborative activities are carried out. These activities are called as **process** as shown in Fig. 1.20.3.

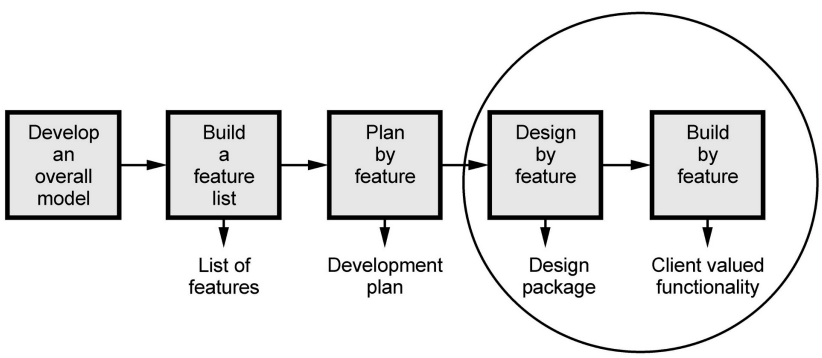


Fig. 1.20.3 Feature driven development life cycle

- Various phases in the FDD life cycle are

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

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

For example

"Display product-specifications of the product"

<Action> <result> <of> <object>

3. **Plan by feature :** After completing the building of feature list the development plan is created. The features are assigned as **classes** and are chief programmer or the class owner is assigned with appropriate classes.
4. **Design by feature :** A **design package** was produced for each feature. A chief programmer selects a small group of features and these features are to be developed within two weeks. For each feature the **sequence diagram** is created.
5. **Build by feature :** Finally a complete client valued function is developed for each feature. The class owners develop the actual code for their classes and this code is promoted to the main build.

- Following are some **benefits of the features -**
  1. Features represent small block of deliverable functionalities hence user can better describe, understand and review them.
  2. The features can be arranged into hierarchical business related grouping.
  3. The team can develop every feature within the two weeks.
  4. The features are typically smaller in size and therefore can be analyzed effectively.
  5. Project planning, scheduling and tracking can be driven by features.

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

1.20.5 Crystal

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

1.20.6 Agile Modeling (AM)

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

6. **Adapt locally :** The needs of modeling team must be satisfied by adopting appropriate modeling approach.

1.21 Plan Driven and Agile Development

- Normally there are two approaches of software development-plan driven development and agile development.
- Agile Development :
  - In agile approach, the **design and implementation** are the central activities and other activities such as requirement elicitation and analysis, testing and so on are integrated with design and implementation.
  - The agile approach is not completely code focused, some necessary documents such as design specification can be produced in this approach.
- **Plan Driven Development :**
  - On the hand, in plan driven development approach each activity is performed in separated stage and output of each stage is used for **planning** the next stage activity.
  - In plan driven approach **the iterations** occur within the activities.
  - The formal documents used to communicate between stages of software development process.

The incremental delivery is possible in plan driven development.

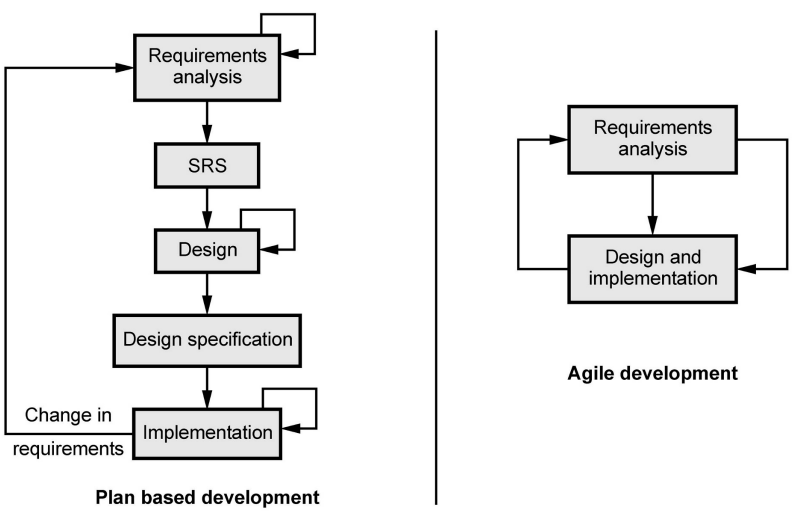


Fig. 1.21.1 Plan driven development and Agile development

Technical, Human and Organizational Issues

Software development is possible by using the elements of both the plan driven and agile approach. However following are the **issues** that are considered while deciding whether to use agile development approach or to use plan driven development

1. If the project requires, the detailed specification and implementation before going into next state then plan driven approach should be used.
2. If the rapid software development is required and incremental approach of development is adopted then go for agile approach of development.
3. For developing large system, then using small interacting team, the agile approach can be used.
4. For the complex systems like real time systems the lot of analysis is required before implementation, then the plan-driven approach is required.
5. For long-life systems in which the original documentation is used by the team members to support the software development. In this situation the plan driven approach is the most suitable one.
6. If the new tools and techniques are available for software development then agile approach is used.
7. If the development team is distributed and part of software development need to be outsourced then the plan-driven approach is adopted, because in plan driven development the detailed documents are created to communicate the development to other team members.
8. If traditional culture is preferred in organization then plan driven approach is used.
9. It is considered than the highly skilled developers are required for agile approach. If such team members are available then the agile approach is used.
10. If the system needs the approval from external regulators then it requires to produce detailed system documentation and in such case, the plan driven approach is used.

1.22 Multiple Choice Questions

Q.1 _____ software is a collection of programs written to service other.

a Application b System

c Engineering d Embedded

Q.2 Software is a process and _____.

a program b design

c product d none of the above

Q.3 Robotics, expert system and pattern recognition are the applications of _____

a system software b web programming

c artificial intelligence d game programming

Q.4 Software engineering is a layered technology. It consists of ___ layers.

a two b three

c four d five

Q.5 Quality focus layer, process layer, methods layer and____ layer is a known as software engineering as layered technology.

a presentation b business logic

c implementation d tools

Q.6 _____ process framework activity is responsible for feedback.

a Communication b Modeling

c Construction d Deployment

Q.7 ___ is a collection of software engineering work task, milestones and deliverable that must be accomplished to complete particular project.

a Program b Document

c Design d Task

Q.8 The _____ model is a realistic approach to the development of large-scale systems and software.

a spiral b RAD

c prototype d incremental

Q.9 SDLC Stands for _____.

a Software Document Listing Collection

b Software Development Life Cycle

c Software Deployment Life Cycle

d System Development Life Cycle

Q.10 Which of the following is evolutionary process model ?

a Incremental model b Spiral model

c Concurrent development model d All of the above

Q.11 Which two models doesn't allow defining requirements early in the life cycle ?

a Waterfall and RAD b Prototyping and Spiral

c Prototyping and RAD d Waterfall and Spiral

Q.12 If XYZ company wants to enhance the current software product, then which process model will you prefer to perform this job ?

a Spiral
b Iterative enhancement model
c RAD model
d both b and c

Q.13 If the project is to be completed within the tight schedule then we choose waterfall model

a True
b False

Q.14 Which of the following is not a software process model ?

a Waterfall model
b Incremental model
c Capability maturity model
d Spiral model

Q.15 Choose the type of project where the prototyping model of software development is well suited.

a For the projects with large development teams
b When the customer cannot define the requirements clearly
c When the requirements are well defined.
d none of these

Q.16 Which model is popular for students small projects ?

a Waterfall model
b RAD model
c Spiral model
d WIN WIN model

Q.17 If the requirements are easily understandable and defined then which model is best suited ?

a Spiral model
b Prototyping model
c Waterfall model
d Incremental model

Q.18 Spiral Model has high reliability requirement _____.

a True
b False

Q.19 The model in which the requirements are implemented by its category is _____.

a evolutionary development model
b waterfall model
c prototyping model
d iterative enhancement model

Q.20 RAD model has reliability requirement

a True
b False

Q.21 The most important feature of spiral model is _____.

a requirement analysis b risk management

c quality management d configuration management

Q.22 What is the main difference between the spiral model and other models ?

a Each loop is considered as a phase

b Describe the process as a spiral

c Does not include planning activities

d Explicit recognition of risk

Q.23 _____ is not an agile method.

a Extreme programming b Scrum

c Waterfall d Crystal

Q.24 _____ three framework activities are present in Adaptive Software Development(ASD)

a Analysis, Design, Coding

b Requirements gathering, Adaptive cycle planning, Iterative development

c Speculation, Collaboration, Learning

d All of the mentioned

Q.25 _____ four framework activities found in the Extreme Programming(XP).

a Analysis, Design, Coding, Testing b Planning, Analysis, Design, Coding

c Planning, Design, Coding, Testing d Planning, Analysis, Coding, Testing

Q.26 In Scrum, when is a Sprint Over ?

a After completing all the Sprint Backlog Items

b After completing all the Sprint Backlog tasks

c After completing the final testing

d When the time box expires

Answer Keys for Multiple Choice Questions :

Q.1	b	**Q.2**	c	**Q.3**	c	**Q.4**	c
Q.5	d	**Q.6**	d	**Q.7**	d	**Q.8**	a
Q.9	b	**Q.10**	d	**Q.11**	b	**Q.12**	d

Q.13	b	**Q.14**	c	**Q.15**	b	**Q.16**	a
Q.17	c	**Q.18**	a	**Q.19**	a	**Q.20**	b
Q.21	b	**Q.22**	d	**Q.23**	c	**Q.24**	c
Q.25	c	**Q.26**	d				

❑❑❑

Notes

UNIT - II

2 Software Requirement Engineering and Analysis Modeling

Syllabus

Modeling : *Requirements Engineering, Establishing the Groundwork, Identifying Stakeholders, Recognizing Multiple Viewpoints, working toward Collaboration, Asking the First Questions, Eliciting Requirements, Collaborative Requirements Gathering, Usage Scenarios, Elicitation Work Products, Developing Use Cases, Building the Requirements Model, Elements of the Requirements Model, Negotiating Requirements, Validating Requirements.*

Contents

2.1 Requirements Engineering

SPPU : May-19, Marks 5

Requirement engineering is the process of

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

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

What is a requirement ?

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

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

Types of requirements

The requirements can be classified as

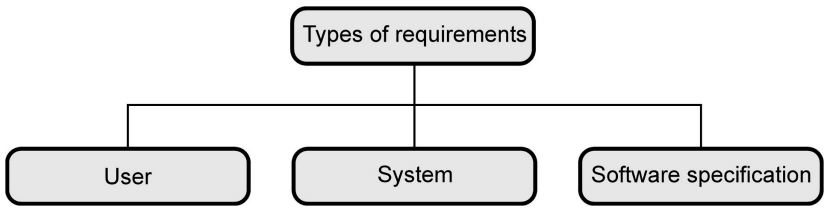


Fig. 2.1.1 Types of requirements

- **User requirements**

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

- **System requirements**

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

- **Software specification**

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

2.1.1 Need for Requirements to be Stable and Correct

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

As requirements are identified and analysis model is created the software team and other

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

Review Question

1. Explain the importance of requirement engineering. **SPPU : May-19, End Sem, Marks 5**

2.2 Requirements Engineering Tasks

SPPU : Dec.-14, 17, 19, April-16, Marks 5

- Requirement engineering is the process **characterized** for achieving following **goals** -
  - Understanding customer requirements and their needs
  - Analyzing the feasibility of the requirement
  - Negotiating the reasonable solutions
  - Specification of an unambiguous solution.
  - Managing all the requirements of the project
  - Finally transforming the requirements into the operational systems
- Requirement engineering process performs following **seven distinct functions** –

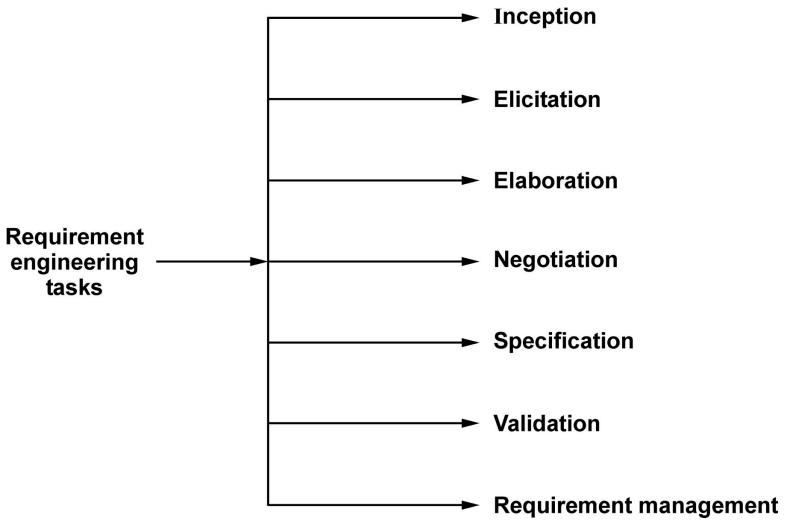


Fig. 2.2.1 Requirement engineering tasks

Let us now discuss these tasks in detail -

2.2.1 Inception

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

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

During the **inception** a set of **context free questions** is discussed. The purpose of inception is to -

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

2.2.2 Elicitation

Before the requirements can be analyzed and modelled they must undergo through the process of elicitation process. Requirements elicitation means **requirements discovery.** Requirements elicitation is very difficult task.

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

1. Customer sometimes is unable to specify the **scope of the project**. Sometimes customers specify too many technical details and this may increase the confusion.
2. There is difficulty in **understanding the problem**. Sometimes customer could not decide what are their needs and wants. Sometimes they have got poor understanding of capabilities and limitations the existing computing environment.

   Sometimes customers find it **difficult to communicate** with the system engineer about their needs. Sometimes customers may have got **some conflicting requirements.** This ultimately results in specifying **ambiguous requirements.**
3. As project progresses the needs or requirements of the customers changes. This creates a problem of **volatility**.

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

2.2.3 Elaboration

- Elaboration is an activity in which the information about the requirements is **expanded** and **refined**. This information is gained during inception and elicitation.
- The **goal** of elaboration activity is to prepare a technical model of software functions, features and constraints.
- The elaboration consists of several **modelling and refinement tasks**. In this process several **user scenarios** are created and refined. Basically these scenarios describe how end-user will interact with the system.
- During elaboration, each user scenario is parsed and various **classes** are identified. These classes are nothing but the business entities that are visible to end user. Then the **attributes** and **services** (functions) of these classes are defined. Then the relationship among these classes is identified. Thus various UML (Unified Modelling Diagrams) are developed during this task.
- Finally the **analysis model** gets developed during the elaboration phase.

2.2.4 Negotiation

Sometimes customer may **demand for more** than that is achieved or there are certain situations in which customer demands for something which cannot be achieved in **limited business resources**. To handle such situations requirement engineers must convince the customers or end users by solving various **conflicts**. For that purpose, requirement engineers must ask the customers and stakeholders to **rank** their **requirements** and then priority of these requirements is decided. Using **iterative approach** some requirements are eliminated, combined or modified. This process continues until the users' satisfaction is achieved.

2.2.5 Specification

- A specification can be a written document, mathematical or graphical model, collection of use case scenarios or may be the prototypes.
- There is a need to develop a **standard specification** in which requirements are presented in consistent and understandable manner.
- For a large system it is always better to develop the specification using natural language and in a written document form. The use of graphical models is more useful for specifying the requirements.
- Specification is the final work product of requirement engineering process. It describes the functions, constraints and performance of computer based systems.

2.2.6 Validation

- Requirement Validation is an activity in which requirement specification is analyzed in order to ensure that the requirements are specified unambiguously. If any inconsistencies, omissions and errors are identified then those are corrected or modified during the validation.
- The most commonly used requirement validation mechanism is **formal technical review (FTR)**. In FTR, the review team validates the software requirements. The review team consists of requirement engineers, customers, end users, marketing person and so on. This review team basically identifies conflicting requirements, inconstancies or unrealistic requirements.

2.2.7 Requirement Management

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

Why requirements get change ?

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

Requirement management process

Following things should be planned during requirement process.

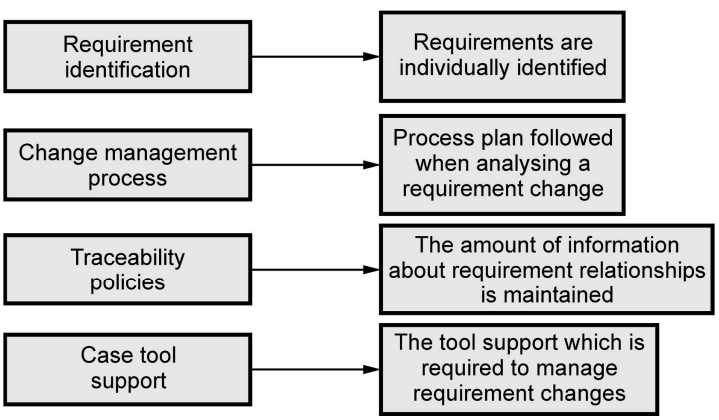


Fig. 2.2.2 Planning in requirement management process

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

Review Questions

1. Why requirement elicitation is difficult? What is meant by requirements negotiation ? Why requirements need to be stable and correct? **SPPU : Dec.-14, Marks 5**
2. What do you mean by requirements inception, elicitation and validation ?

   SPPU : April-16, In Sem, Marks 4
3. What are requirements engineering tasks? Explain in detail.

   SPPU : Dec.-17, 19, End Sem, Marks 5

2.3 Establishing the Groundwork

SPPU : Dec.-07, May-14, Marks 4

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

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

2.3.1 Identifying the Stakeholders

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

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

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

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

2.3.2 Recognizing Multiple Viewpoints

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

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

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

Example 2.3.1 *Describe two real life situations in which the customer and the end-user is the same. Describe two situations in which they are different.*

SPPU : Dec.-07, May-14, Marks 4, I.T.

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

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

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

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

2.3.3 Working Towards Collaboration

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

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

2.3.4 Asking the First Questions

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

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

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

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

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

Set of questions for performing effective communication
1. Are you an authentic person to answer the questions ?
2. Are your answers official ?

3. Am I asking too many questions ?
4. Is there any alternative source of getting answers of my questions ?
5. Are my questions relevant to the problems ?
6. Am I asking anything else ?

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

2.4 Eliciting the Requirements

SPPU : Dec.-14, 18, Marks 5

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

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

Let us discuss these aspects of requirement elicitation in detail -

2.4.1 Collaborative Requirements Gathering

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

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

Guideline for FAST approach -

1. A meeting should be conducted and attended by both software engineers and customers. The place of meeting should be a neutral site.
2. Rules for preparation and participation must be prepared.
3. An agenda should be prepared in such a way that it covers all the important point as well as it allows all the new innovative ideas.
4. A facilitator controls the meeting. He could be customer, developer or outsider.
5. A definition mechanism is used. The mechanism can be work sheets, flip charts, wall stickers, electronic bulletin board, chart room, virtual forum.

6. The goal is to identify the problem, decide the elements of solution, negotiate different approaches and specify the preliminary set of solution requirements.
   - In FAST meeting each FAST attendee is asked to prepare - a list of objects, list of services and a list of constraints.
   - The list of objects consists of all the objects used in the system, the objects that are produced by the system and the objects that surround the system.
   - The list of services contain all the required functionalities that manipulate or interact with the objects.
   - The list of constraints consists of all the constraints of the system such as cost, rules, memory requirement, speed accuracy etc.
   - As the FAST meeting begins, the very first issue of discussion is the need and justification for the new product. Once everyone agrees upon the fact that the product is justified, each participant has to present his lists.
   - These lists are then discussed, manipulated and these modified or refined lists are combined by a group.
   - The combined list eliminates redundant entries adds new ideas that come up during the discussion. The combined list is refined in such a way that it helps in building the system.
   - The combined list should be prepared in such a way that a "consensus lists" can be prepared, for object, services and constraints.
   - A team is divided into subteams. Each subteam develops a minispecification from each consensus list.
   - Finally a complete draft specification is developed.

For example -

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

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

The FAST attendee prepare following lists -

1) List of objects - Display, menu, a sound, an event (moving from one level to another) and so on.
2) List of services - Setting sounds, setting colors in GUI, HELP, instructions for players, score card etc.

3) List of constraints - Must be user friendly, must have high speed, must accommodate less size, should have less cost.

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

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

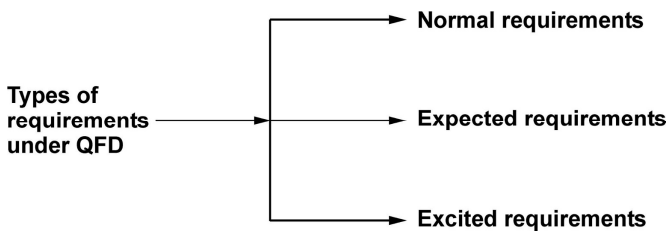
2.4.2 Quality Function Development

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

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

Normal requirements -

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



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

Expected requirements -

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

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

Exciting requirements -

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

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

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

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

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

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

2.4.3 Usage Scenario

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

2.4.4 Elicitation Work Products

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

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

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

Review Question

1. Explain the tasks done during elicitation and requirement management.

SPPU : May-18, End Sem, Marks 5

2.5 Developing Use Cases

SPPU : May-11, Dec.-14, Marks 5

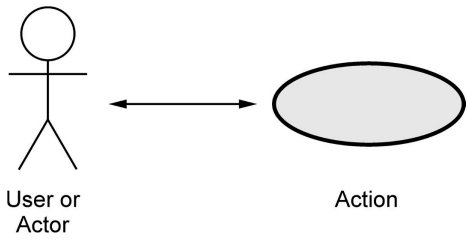


Fig. 2.5.1 Use case notation

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

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

Example 2.5.1 *Develop use case for Bank system.*

Solution : Following is the template suggested by **Cockburn**

Use Case template	
Use Case	ATM system
Primary Actor	Customer
Goal in Context	To monitor all the functionalities required to establish the session
Preconditions	ATM system has to be programmed and as to recognise and validate the PIN.
Trigger	On completion of every activity a beep has to be generated by the system.
Scenario	1. Customer observes the control panel. 2. Customer enters the ATM card 3. Customer enters the PIN. 4. Customer selects the operation(withdrawal, deposit, inquiry) 5. Customer collects the cash, statement, card etc.
Exceptions	1. The control panel is not ready. 2. PIN is incorrect. 3. Card is not recognised. 4. Insufficient balance 5. Limit for the transaction exceeds. 6. Total number of allowed transactions per day.
Priority	Essential and must be implemented in banking system.
Secondary Actor	Administrator
Open Issues	1. Is there a need to display any additional information by the control panel? 2. For how many time the PIN is allowed to enter on incorrect supplement. 3. How much time is allotted to the customer to pick the items like cash or card after completion of operation?

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

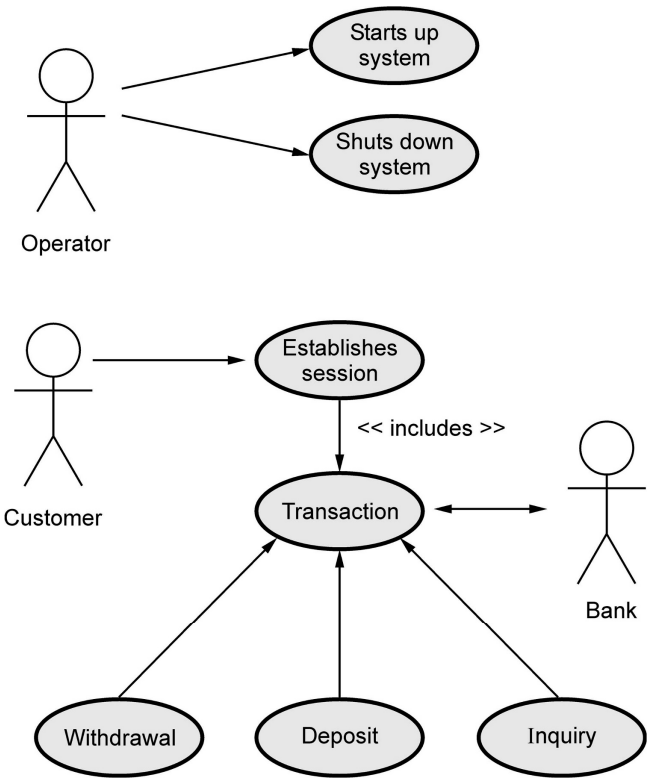


Fig. 2.5.2 Use cases for ATM system

Example 2.5.2 *Write an use case for 'login' with a template and diagram.*

SPPU : May-11, Marks 4

Solution :

Use case template for login	
Use case	Course registration system.
Primary Actor	Student.
Goal in Context	To monitor all the functionalities required to register for the course.
Preconditions	None
Scenario / Basic Flow	1. The system requests student to enter his name and password. 2. The student enters his/her name and password. 3. The system validates the entered name and password and logs the student into the system.

Exceptions	1. The control panel is not ready. 2. User name is invalid. 3. Password is incorrect.
Priority	Essential and must be implemented in course registration system.
Postcondition	If user logs in successfully then course information must be displayed to him / her.
Secondary Actor	Administrator.

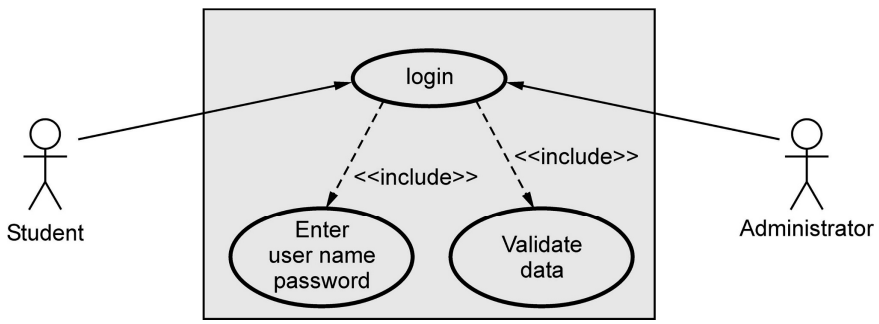


Fig. 2.5.3 Use case diagram

Review Question

1. What do you need to know in order to develop an effective use case ? Describe a standard use case documentation template. **SPPU : Dec.-14, Marks 5**

2.6 Building Requirements Models

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

Requirement analysis produces a software specification.

How is requirement analysis helpful ?

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

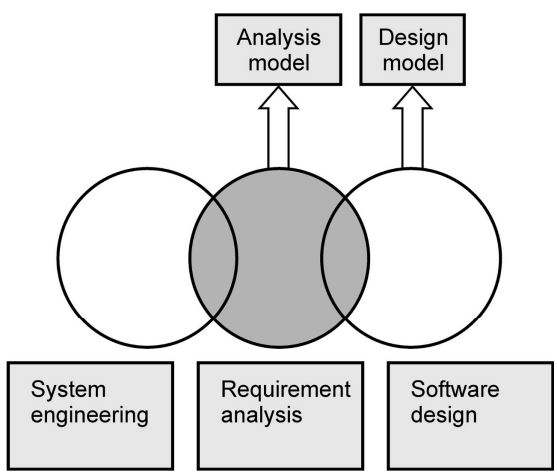


Fig. 2.6.1 Requirement analysis : An intermediate step

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

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

What are requirement analysis efforts ?

1. **Problem recognition**

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

2. **Evaluation and synthesis**

   Following are the tasks that must be done in evaluation and synthesis phase.
   i) Define all externally observable data objects evaluate data flow.
   ii) Define software functions.
   iii) Understand the behaviour of the system.
   iv) Establish system interface characteristics.
   v) Uncover the design constraints.

3. **Modelling**

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

4. **Specification**

   The requirement specification (SRS) must be built.

5. **Review**

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

2.6.1 Overall Objectives

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

2.6.2 Analysis Rules of Thumb

Arlow and Neustadt suggested some rules that should be followed during the creation of analysis model. These rules are called **analysis rules of thumb.** They are as follows -

- The analysis model should focus only on requirements that are visible within the business domain. That means there is no need to focus on detailed requirements.
- Each element of analysis model should help in building overall understanding of software. These elements should provide insight into information domain, function and behaviour of the system.
- Minimize coupling within the system. Coupling is basically used to represent the relationship between the two objects by means of functionalities.
- Delay use of infrastructure and other non functional models until detailed software design is obtained.
- Some analysis models should provide the values to the stakeholders. For example, the business stakeholder should focus on the functionalities that are basic demands of the market; software designer must focus on basic design of the product.
- The analysis model should be very simple so that it could be understood easily.

2.6.3 Domain Analysis

It is observed that there are varieties of **patterns** that occur in many applications with in the same business domain. If these patterns are categorized systematically then that allows software engineer to **reuse** them. Analysis model helps in identifying such patterns.

Definition of domain analysis - The domain analysis can be defined as identification of common requirements from a specific application domain, for reusing them on multiple projects within that specific application domain.

How to do domain analysis ?

The domain analysis can be done by finding or creating those **analysis classes** that are having common functions and features that can be **reused.**

It is the responsibility of domain analyst to discover and define reusable analysis patterns, analysis classes and related information.

What are the advantages of domain analysis ?

1. The domain analysis done for finding the reusable design patterns and executable software components helps in building the system or product very efficiently.
2. It reduces the cost of development of design patterns.

2.7 Elements of the Requirements Model

SPPU : Dec.-14, Marks 5

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

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

Following are the elements of analysis model -

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

Following Fig. 2.7.1 illustrates the elements of analysis model.

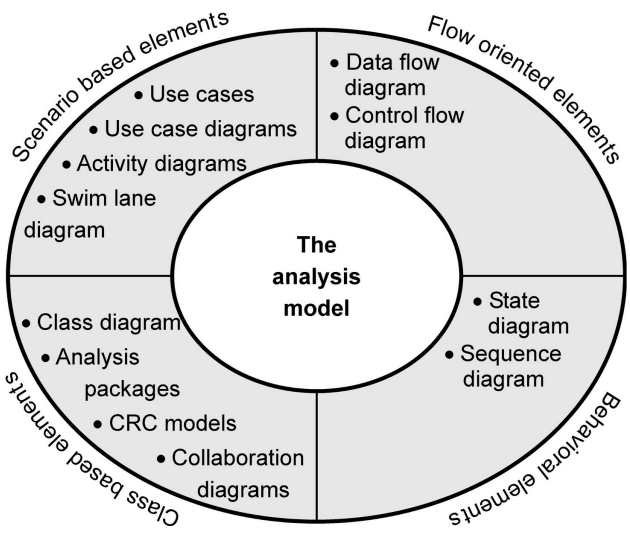


Fig. 2.7.1 Analysis model

Review Question

1. What are different approaches or elements of a requirement analysis model ?

SPPU : Dec.-14, Marks 5

2.8 Scenario Based Modeling

SPPU : May-13, Dec.-13, Marks 8

When user of the computer-based system gets satisfied completely then that system or a product is said to be successful product. The software engineers try to understand how an user will interact with the system by properly characterising the requirements and by building the

analysis models. In analysis modelling, at the beginning, creation of scenarios in the form of use cases, activity diagrams and swimlane diagram occurs.

2.8.1 Writing Use Cases

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

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

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

Actor

An actor is an entity which interacts with the system. Actors carry out use cases. **For example :** In above use case diagram student is an actor.

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

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

Use cases

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

Relationships

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

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

Extend relationship : Identifies a reusable use case that *conditionally* interrupts the

execution of another use case by using its functionality. The decision of when the extending use case should be used is taken by the extending use case itself.

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

Fig. 2.8.1 shows the association, include relationship, generalization and Extended use case for **Clinical system**.

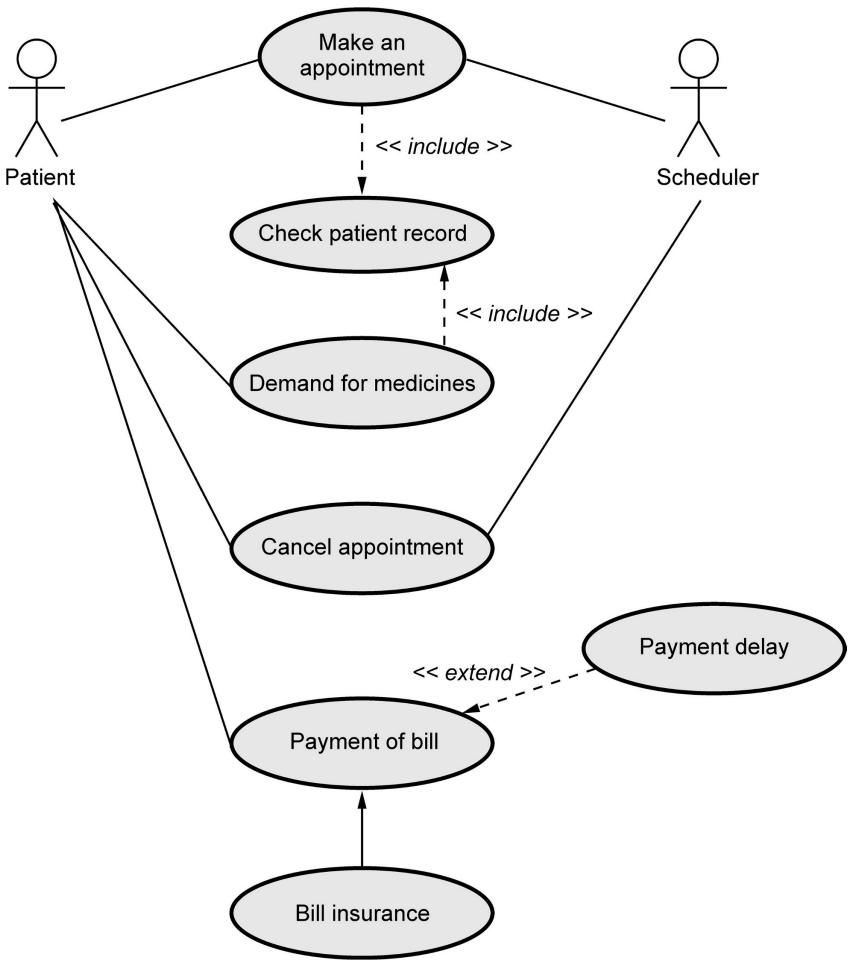


Fig. 2.8.1 Use case diagram with various relationship

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

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

Example 2.8.1 *Design use case diagram for user interaction with ATM system.*

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

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

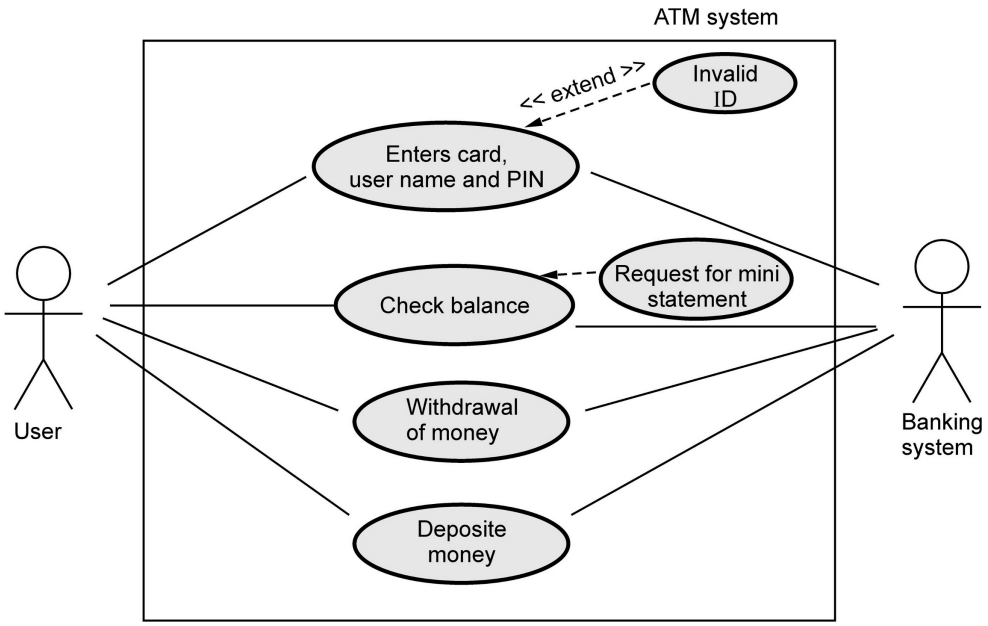


Fig. 2.8.2

Example 2.8.2 *Draw a use case diagram for library management system.*

Solution :

Use case : For library management system

1. For library system user can be a staff or a student who initially registers himself as a new user.
   a) For registering as a new user registration form is filled up.
   b) Then librarian issues a library card on which some ID is assigned to the card holder.
2. The user requests for a new book.
3. The user then reserves a desired book.
4. The user may renew a book.
5. User pays fine if the book is returned late.

6. User can also submit his feedback by returning the feedback form.
7. There is another important actor in this system and that is "Librarian". The librarian adds the record in the library database.
8. Librarian deletes a record, if the book is not existing in the library or if student leaves the college.
9. Librarian updates the database.

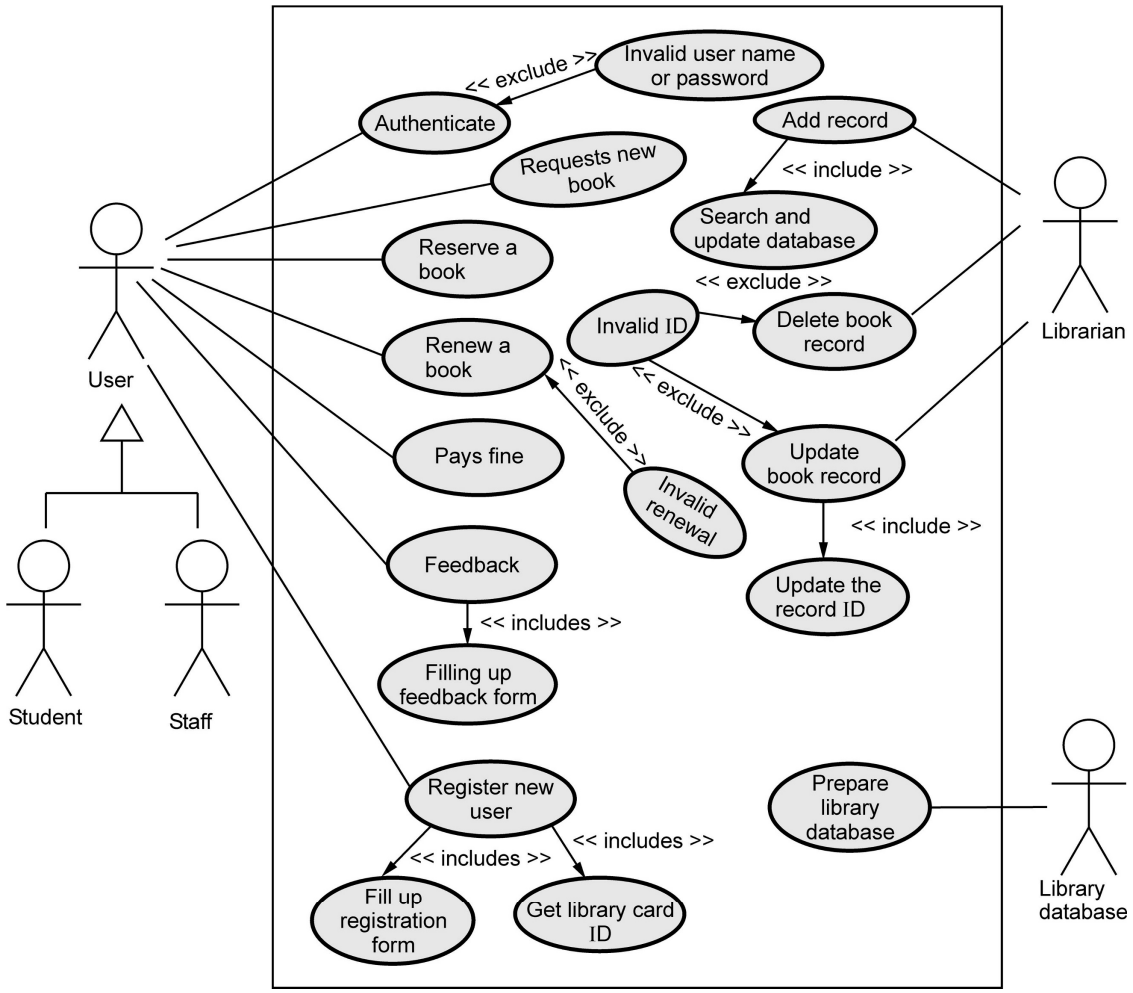


Fig. 2.8.3 Use case diagram for library management system

Example 2.8.3 *Give the actors, use cases and use case diagram for "online mobile order and purchase system". Write the scope of the system.* **SPPU : Dec.-13, Marks 8**

Solution : The basic idea is that customers can buy products using online. It consists of product details, security system, status and exits. The administrator can enter the name and password and generate the report and can perform operations like add, search, delete the

products in the database. The online mobile shopping system enables vendors to set up online shops, customers to browse through the shops and a system administrator to approve and reject requests for new shops and maintain lists of shop categories. Also on the agenda is designing an online shopping site to manage the items in the shop and also help customers purchase them online without having to visit the shop physically. Web customer actor uses some web site to make purchases online. Top level use cases are View Items, Make Purchase and Client Register.

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

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

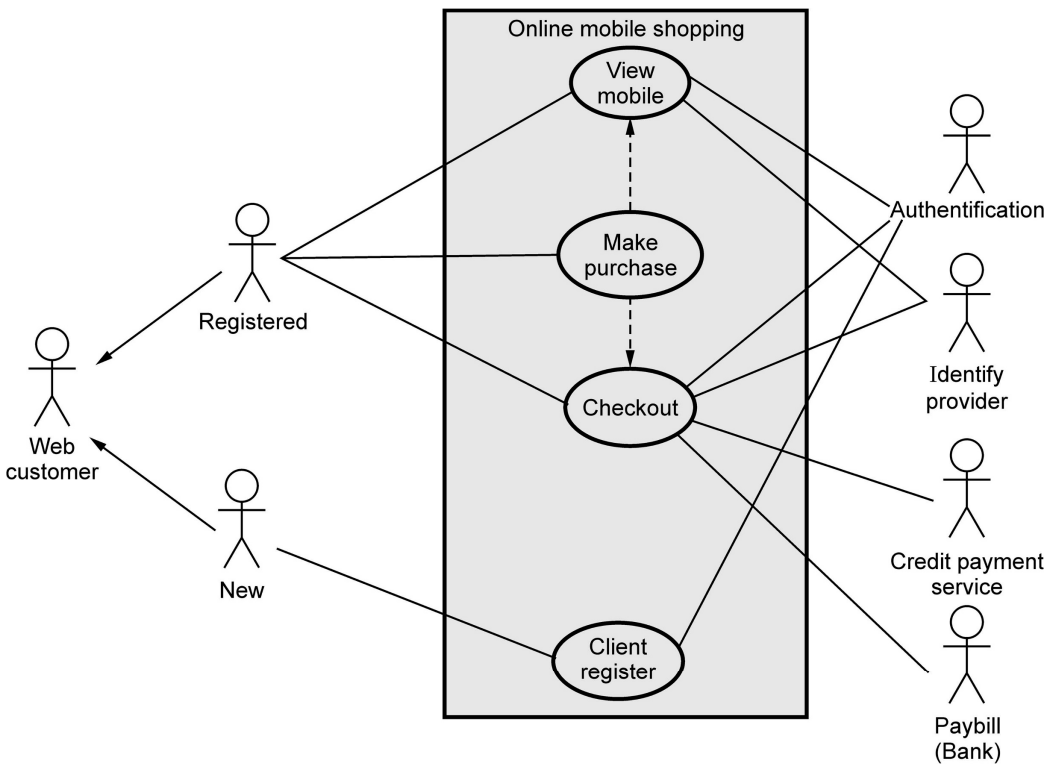


Fig. 2.8.4 Use case diagram for online mobile purchase

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

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

2.8.2 Activity Diagram

The activity diagram is a graphical representation for representing the flow of interaction within specific scenarios. It is similar to a flowchart in which various activities that can be performed in the system are represented. This diagram must be read from top to bottom. It consists of **forks** and **branches.** The fork is used to represent that many activities can be parallely carried out. This diagram also consists of **merge,** where multiple branches get combined. Before transitioning into final activity state there comes a **join.** A typical structure of activity diagram is -

Below is the railway reservation system. The activity diagram for reserving a ticket is shown below. (Also refer Fig. 2.8.6 on next page)

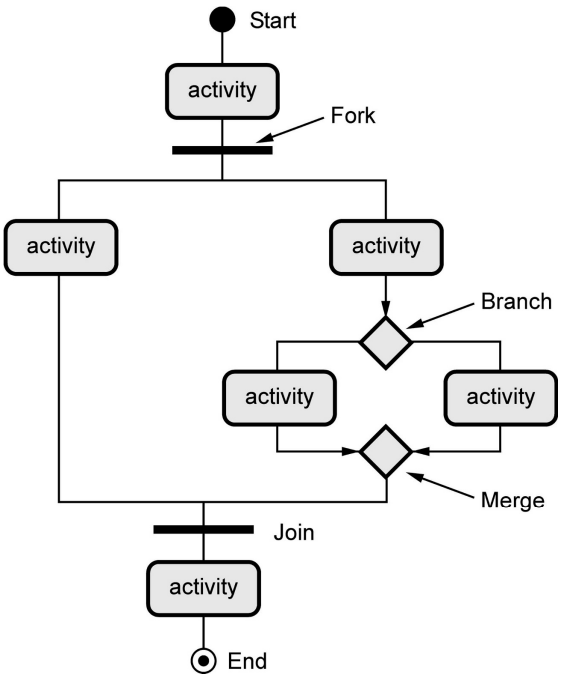


Fig. 2.8.5 Components of activity diagram

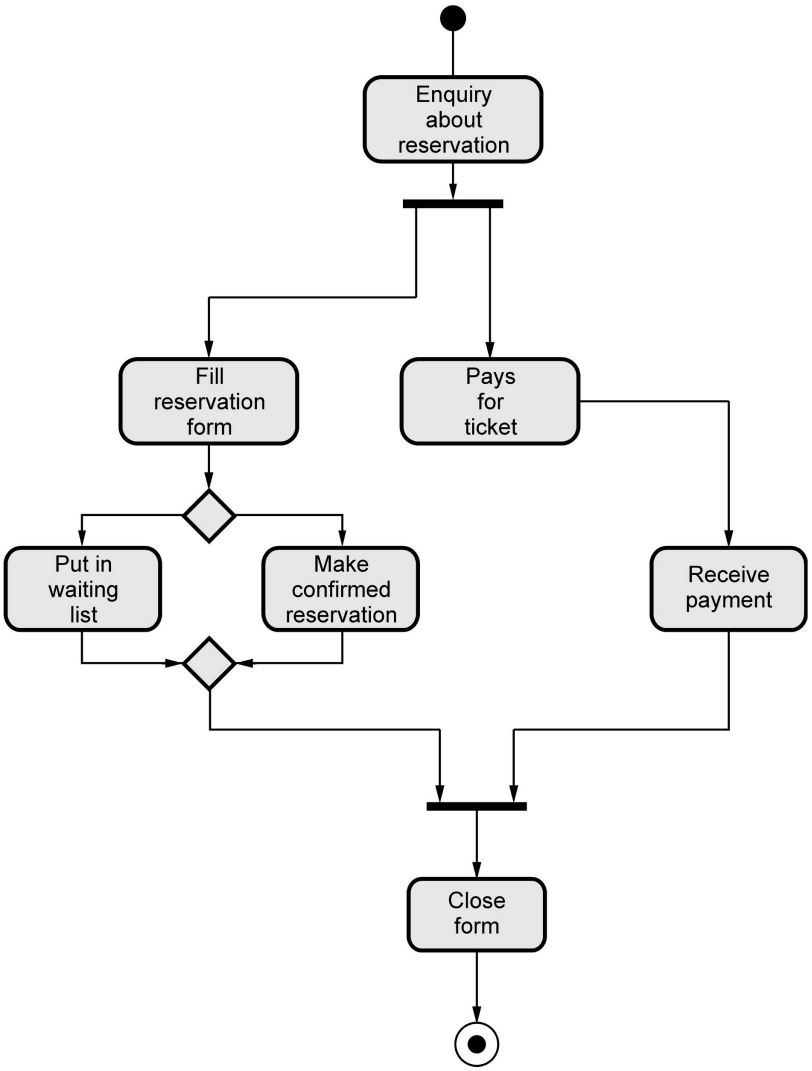


Fig. 2.8.6 Activity diagram for reservation

2.8.3 Swimlane Diagram

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

Example 2.8.4 *Draw swimlane diagram for railway reservation activity.*

Solution : Refer Fig. 2.8.7 on next page.

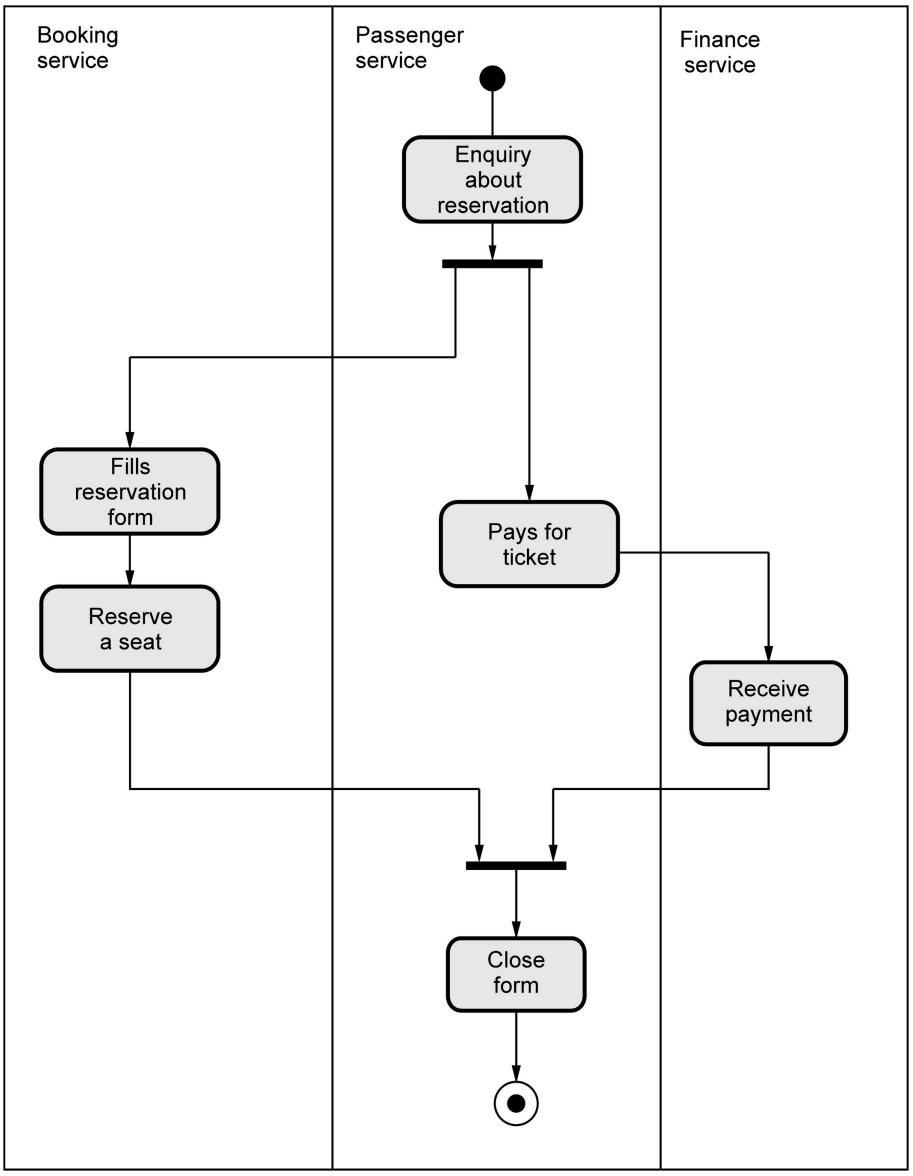


Fig. 2.8.7 Swimlane diagram for reservation

Example 2.8.5 *Create the swimlane diagram for, monitoring of sensor in a 'Safehome security system' from control panel.* **SPPU : May-13, Marks 8**

Solution : Refer Fig. 2.8.8 on next.

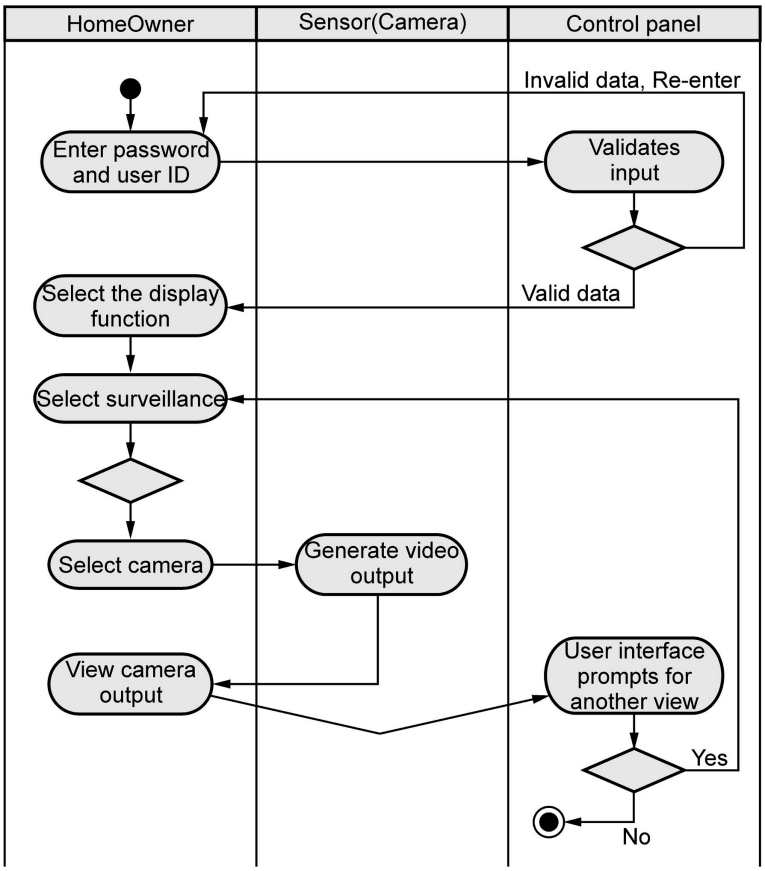


Fig. 2.8.8

2.9 Class Based Modeling

SPPU : May-13, Marks 4

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

2.9.1 Objects and Object Classes

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

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

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

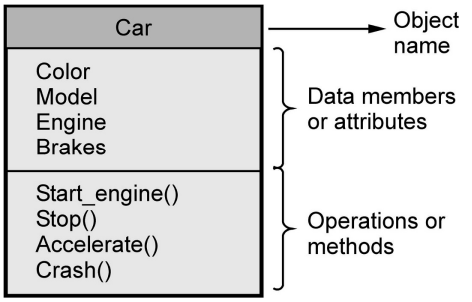


Fig. 2.9.1 Car object

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

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

Hence any operation of the object consists of -

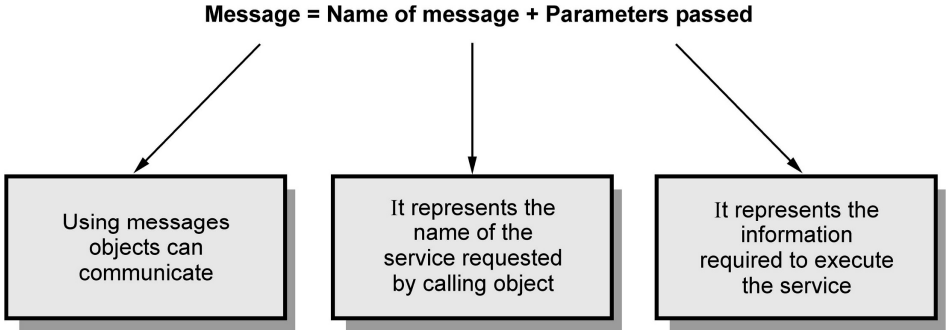
Name = Procedure name

Information = Parameter list.

The example of message is :

```
Circle.set_radius(10);
```

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



2.9.2 Generalization and Inheritance Relationship

Generalization is one of the important features of object oriented systems. A class hierarchy can be formed where one class is generalisation of one or more other classes. That means there can be one **super class** whose attributes and operations can be inherited by the

one or more **sub classes**. The sub classes may add its own attributes and methods. The example of generalization hierarchy is as shown below.

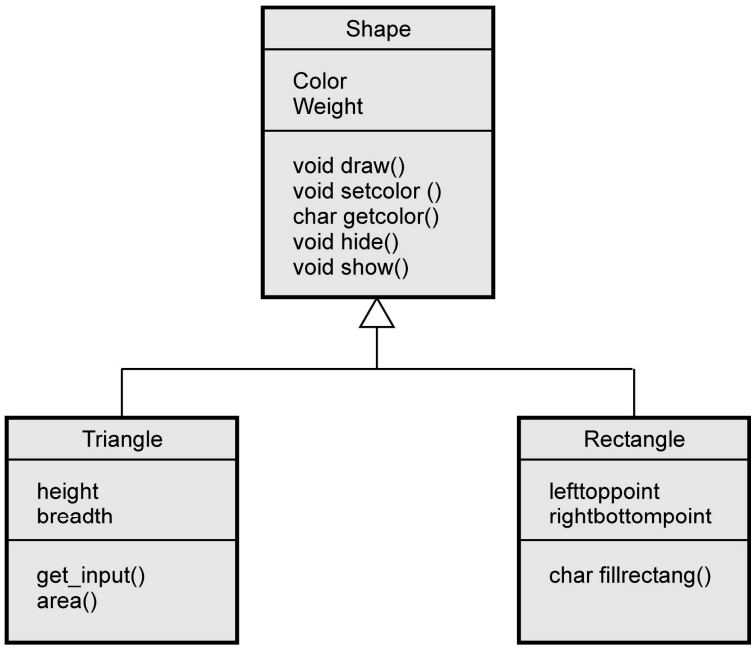


Fig. 2.9.2 Inheritance

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

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

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

The generalization indicates **type of** relationship. If the child classes are derived from more than one parent then it shows **multiple inheritance**. Fig. 2.9.3 shows the multiple inheritance.

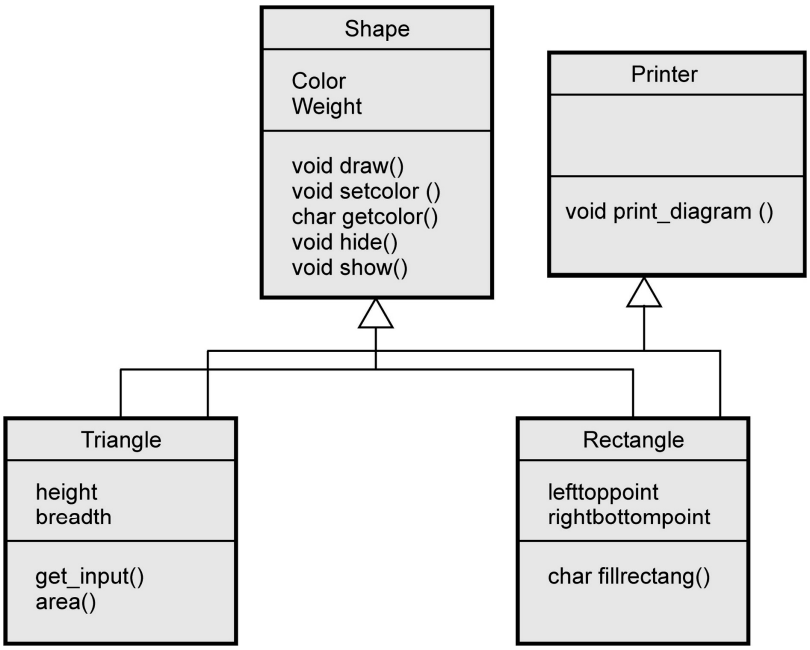


Fig. 2.9.3 Multiple inheritance

Advantages of inheritance

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

Drawbacks of inheritance

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

2.9.2.1 Association Relationship

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

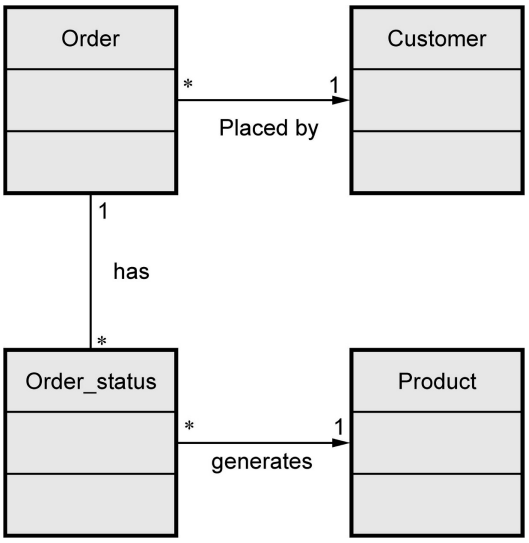


Fig. 2.9.4 Associations

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

Example 2.9.1 *Draw a class diagram for library management system.*

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

Multiplicity : One or more instances of particular class can be associated with one or more instances of another class. For denoting one instance we write 1 and to denote many instances is used. For example, in above given class diagram, many users can be associated with many books. On the other hand, every single user will have only one account. (See Fig. 2.9.5 on next page).

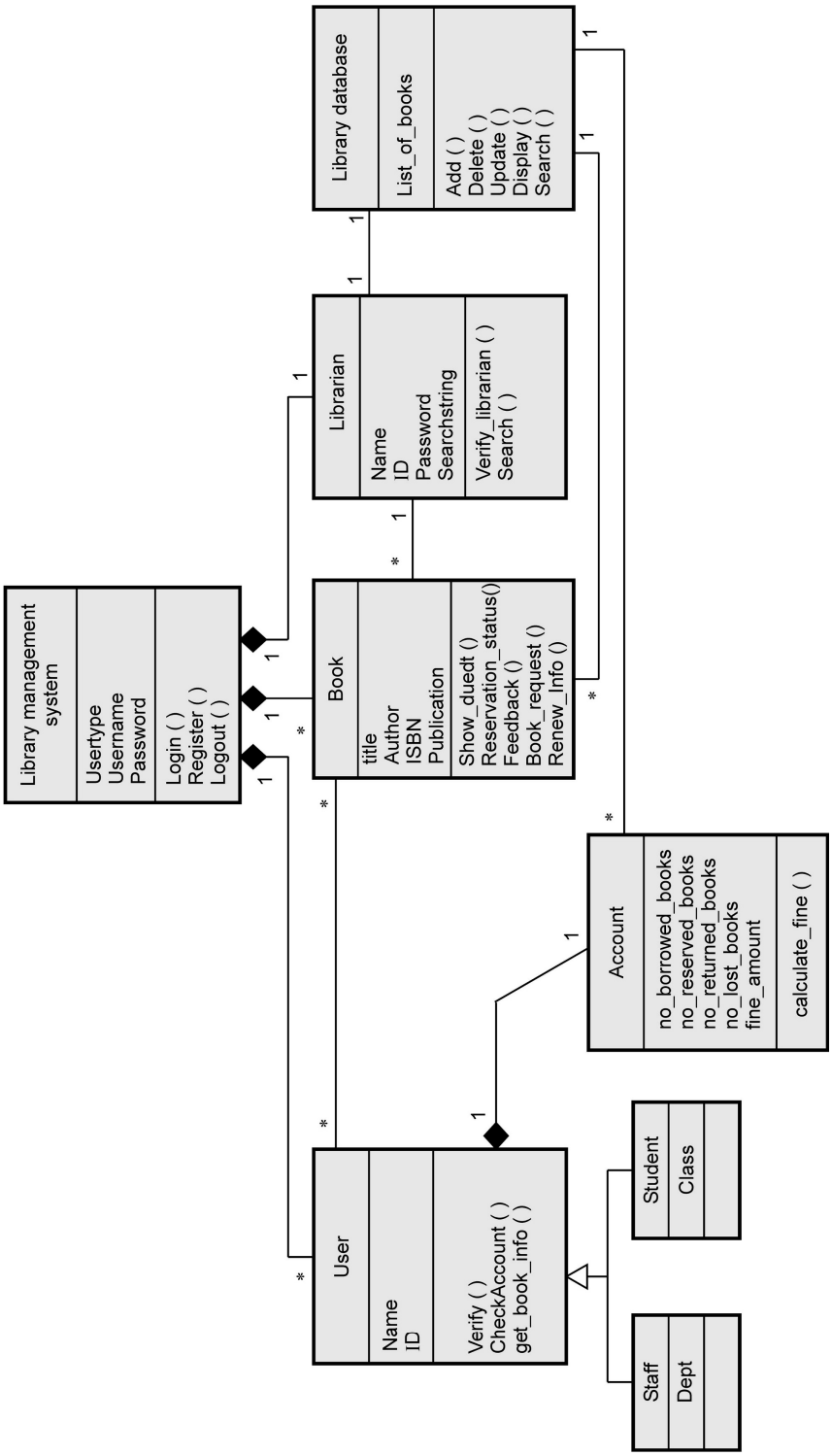


Fig. 2.9.5 Class diagram for library management system

2.9.3 Object Identification

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

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

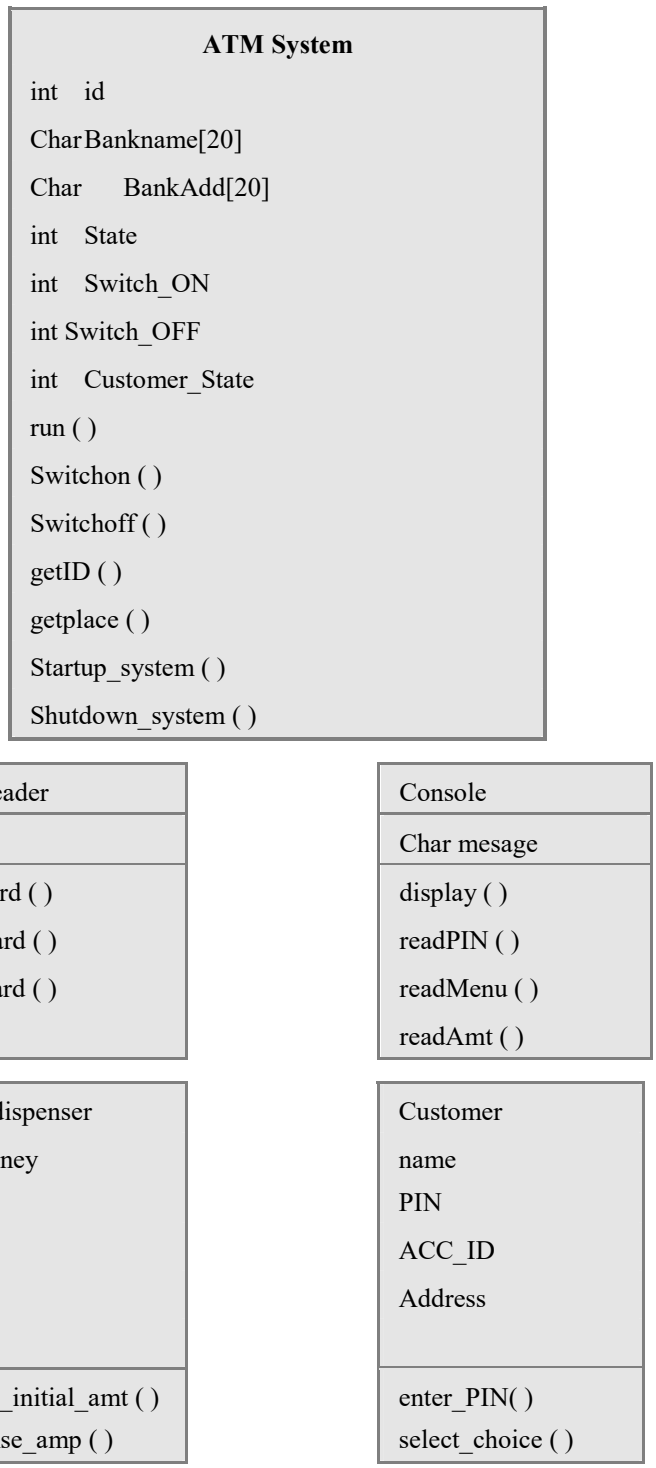
For example : "Consider an ATM System in which the customer with an ATM card can establish session with the system. First of all he inserts the card into the system and enters the PIN. The console of the system reads the card and authenticates the customer. The system has console containing display and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

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

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

1) ATM system is controller object.
2) Individual components can be
   - Card reader
   - Console
   - Cash dispenser
3) The initiating object is customer.
4) The controlling objects are
   - Session
   - Transaction

Various objects can then be as shown below.



2.9.4 Class-Responsibility-Collaborator(CRC) Modelling

- The **C**lass-**R**esponsibility-**C**ollaborator modelling is a technique which helps in identifying and organizing the classes that are relevant to the system requirements. Using this technique the classes required by the system are identified their responsibilities are defined and their collaborations are identified.
- **Ambler** has described the CRC modelling as : "The CRC model is a collection of standard **index cards**. This card is divided into three sections. The first section contains name of the class, type of the class and characteristics of the class. Then the second section is described at the left side and third section is described at the right side of CRC card. The second section describes the responsibilities and the third section describes the Collaborations.
- The typical structure of CRC card is as shown by Fig. 2.9.6.

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

Fig. 2.9.6 Structure of CRC card

- **Class -** For selecting particular entity as a class following guideline can be used -

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

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

Fishersmith has suggested following types of classes.

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

Various characteristics of objects are :

1. **Tangibility :** Does the class represent the tangible thing or represent an abstract information?
2. **Sequentiality :** Whether the class is concurrent or sequentially defined?
3. **Inclusiveness :** Does the class include other class or it is independent of other class? This represents **atomic** or **aggregate** property.
4. **Persistence :** Is the class permanent or temporary ? Sometimes the class created at the design time are removed during the implementation. Such classes are temporary classes and there are some classes that remain in the database. Such classes are called permanent classes.
5. **Integrity** do the class allow other resources to access it or does it protect itself from other resources ? If the class does not allow other resources to access itself then such classes are called **guarded** and if the classes allow other resources to access itself then such classes are called **corruptible.**

- **Responsibilities :** The responsibilities of a class can be determined by its attributes and responsibilities. **Wirfs-Brock** has suggested following guidelines for allocating the responsibilities to the classes -

1. System intelligence should be evenly distributed

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

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

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

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

2. Each responsibility should be mentioned in generalised way.

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

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

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

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

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

5. Appropriate responsibilities can be shared among multiple classes.

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

- **Collaborations**

  The class can fulfill its responsibilities by

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

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

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

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

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

Example 2.9.2 *Write a CRC for the class 'Email system user'.* **SPPU : May-13, Marks 4**

Solution :

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

2.9.5 Object Relationship Model

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

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

  Step 2 : Identify the responsibilities of each class.

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

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

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

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

1. Using CRC index cards the Collaborator objects can be identified.

Following Fig. 2.9.7 represents the various CRC index cards that can be connected together. Note that these objects are simply connected together no named relationship exists at this stage.

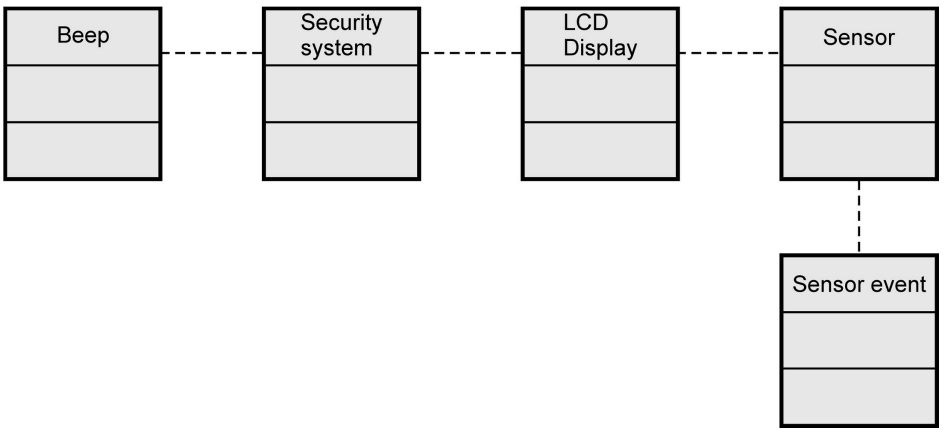


Fig. 2.9.7 CRC index cards

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

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

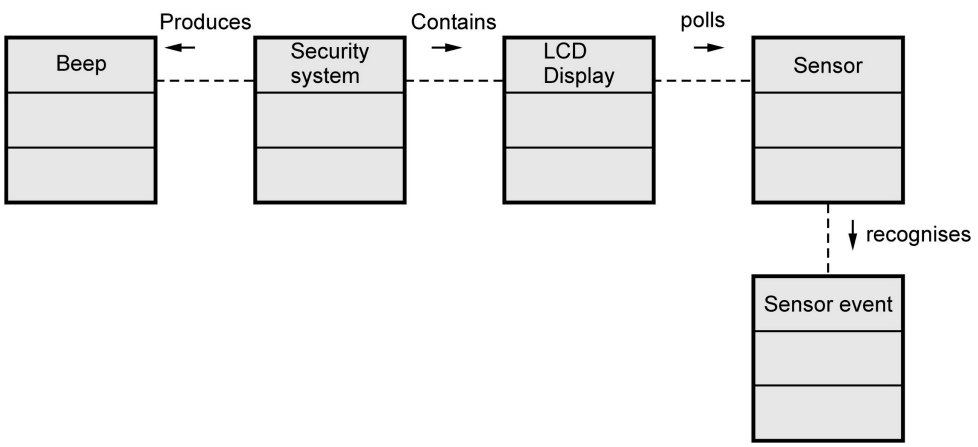


Fig. 2.9.8 Cardinality of relationship

3. Determine the cardinality.

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

For example :

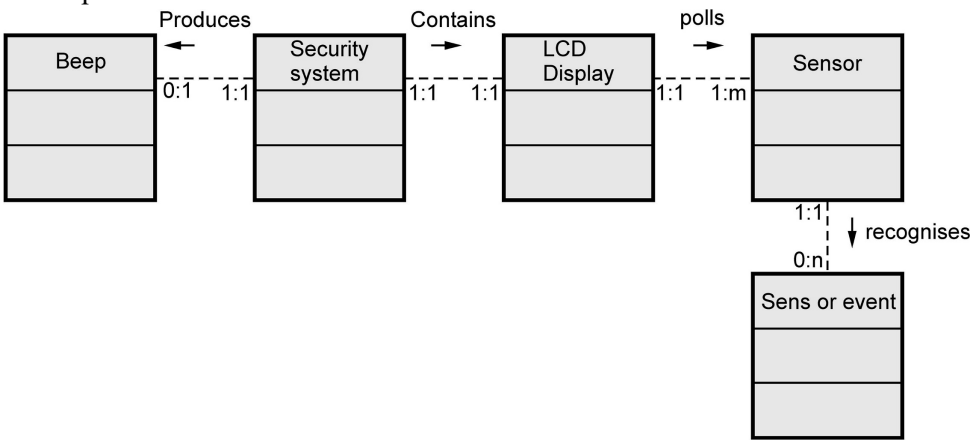


Fig. 2.9.9 Cardinality of relationship

2.10 Data Modeling

SPPU : Dec.-11, 14, May-12, Oct.-19, Marks 5

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

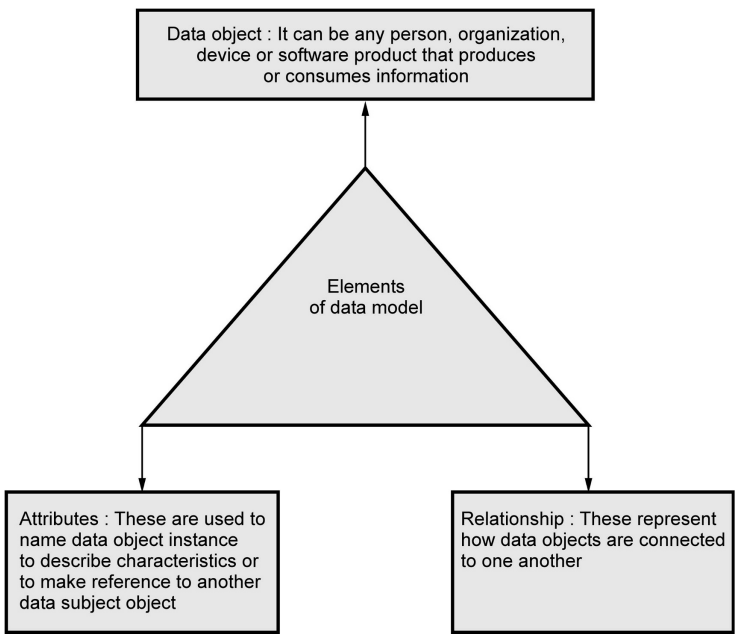


Fig. 2.10.1 Elements of data model

2.10.1 Data Object, Attributes and Relationships

What is data object ?

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

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

Object : *Vehicle*
Attributes:
Make
Model
Color
Owner
Price

Fig. 2.10.2 Object

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

Typical data objects are

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

What are attributes ?

Attributes define properties of data object

Typically there are three types of attributes -

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

What is relationship ?

Relationship represents the connection between the data objects. For example

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

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

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

2.10.2 Cardinality and Modality

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

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

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

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

Example

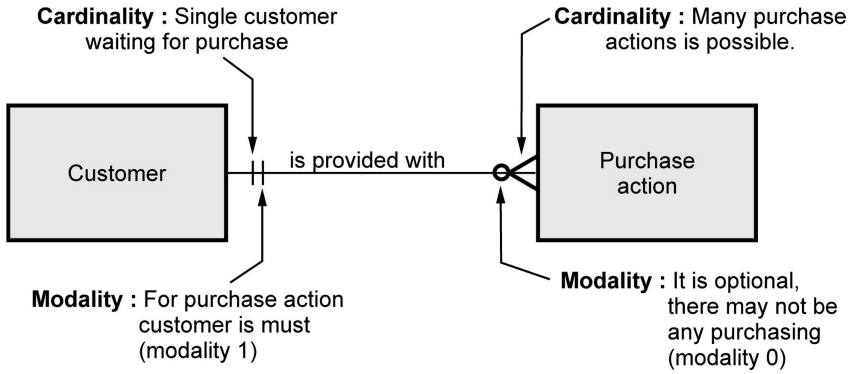


Fig. 2.10.3 Cardinality and modality

2.10.3 Entity Relationship Diagram

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

Entity

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

Relationship

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

Attribute

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

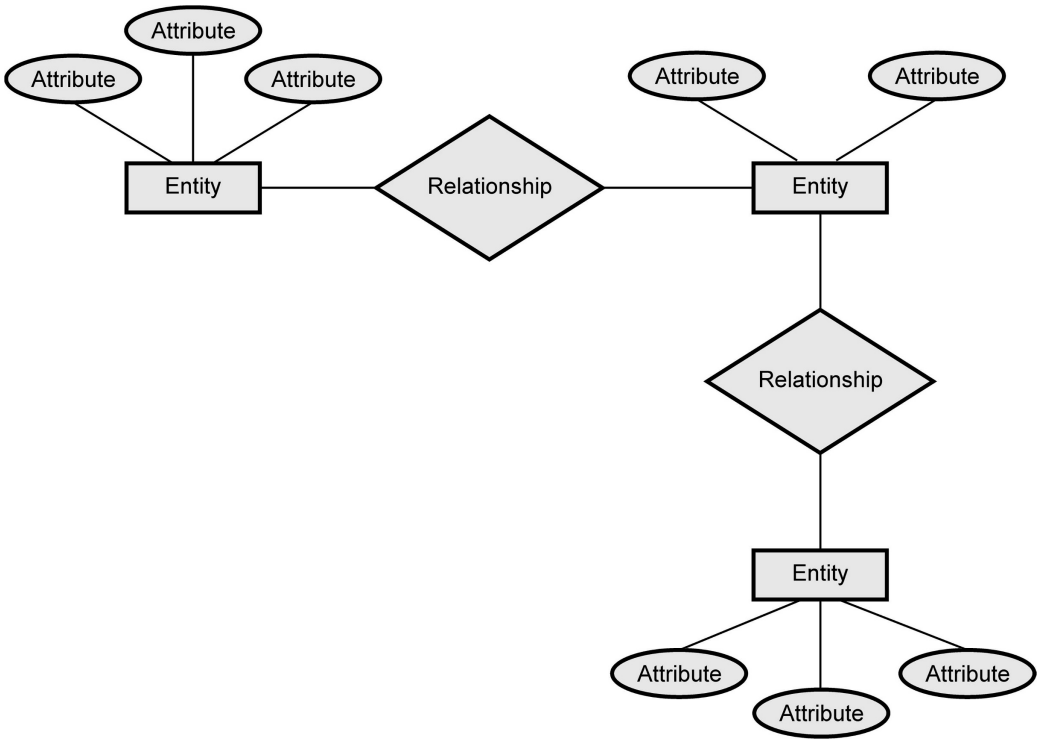


Fig. 2.10.4 ER diagram

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

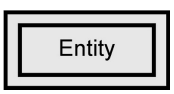
Notations used in ER diagram

Entity

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

Weak entity

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



Attribute

The attributes are properties or characteristics of an entity.

Derived attribute

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

Key attribute

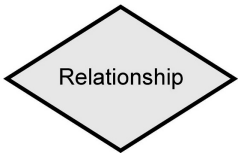
A key attribute is an unique attribute representing distinguishing characteristic of entity. Typically primary key of record is a key attribute.

Multivalued attribute

A multivalued attribute have more than one value.

Relationship

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



Notations to show cardinality

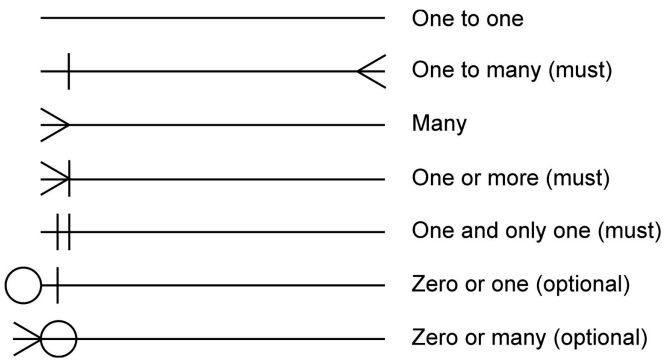


Fig. 2.10.5 Cardinality

2.10.3.1 Design Guideline and Examples

The entity relationship diagram is used to represent the data objects their attributes and the relationship among these data objects. The ERD is constructed in iterative manner. Following guideline is used while drawing the ERD.

1. During the requirement elicitation process, the requirements should be collected in such a way that we can evolve input, output data objects and external entities for system modeling.
2. The analysis and customer should be in a position to define the relationship between the data objects.
3. Whenever a connection between data objects is identified the object relationship pair must be established. Thus iteratively relationship between all the objects must be established.
4. For each object relationship pair the cardinality and modality is set.
5. The attributes of each entity must be defined.
6. The entity relationship diagram is formalized and reviewed.
7. All the above steps are repeated until data modeling is complete.

Example 2.10.1 *Draw an ER diagram for the relationship of teacher and courses. Also specify the association, cardinality and modality.*

Solution :

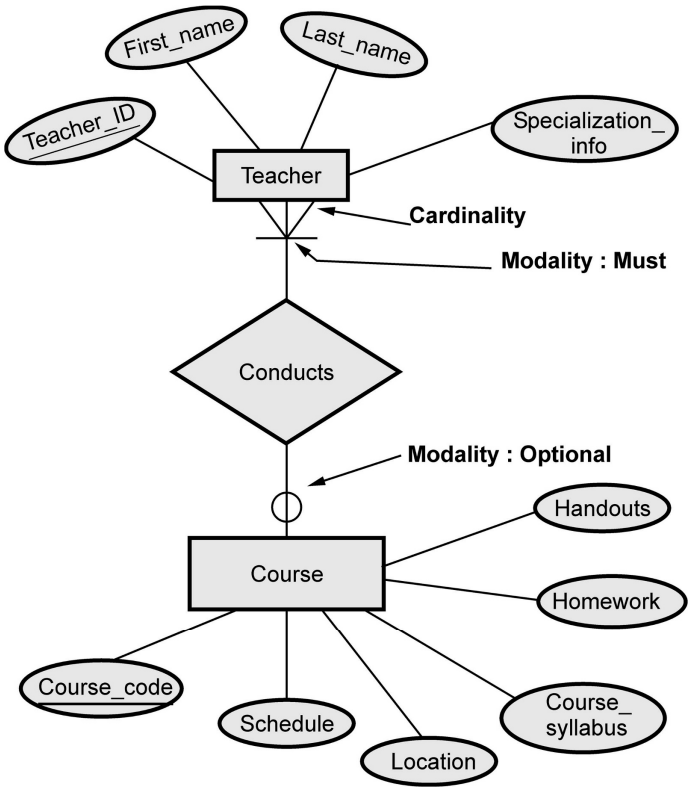


Fig. 2.10.6

Association

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

Cardinality

Many Teachers can conduct the single course.

Modality

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

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

Example 2.10.2 *Draw an ER diagram for the relationship between customer and banking system.*

Solution :

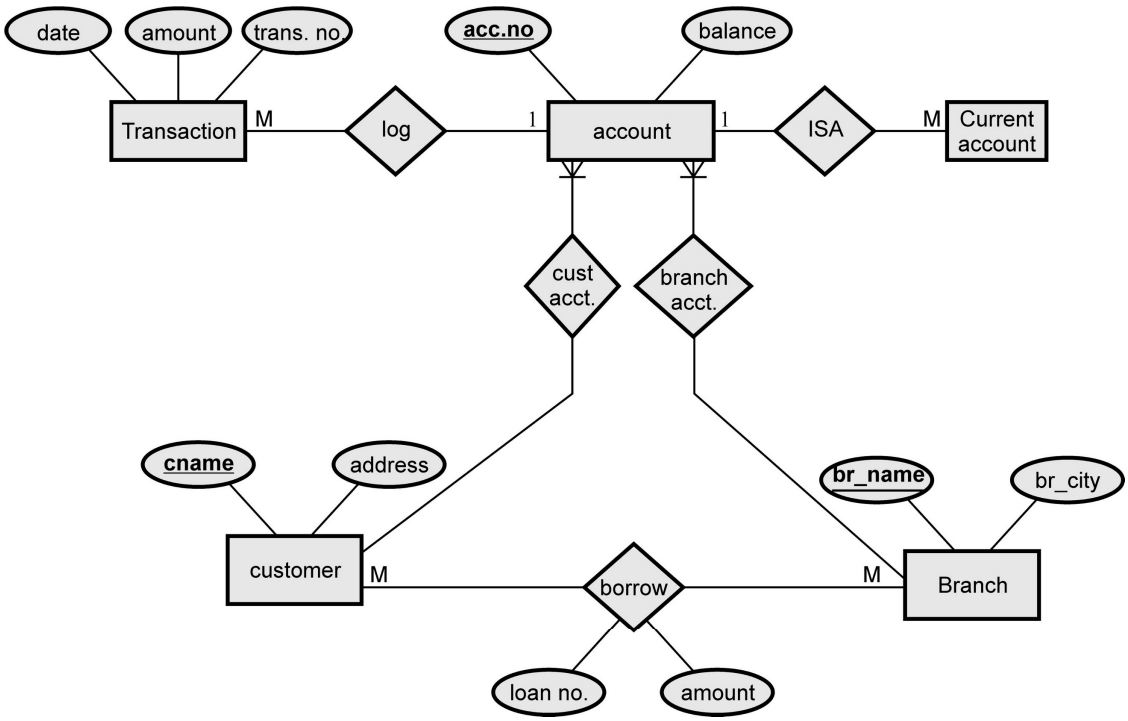


Fig. 2.10.7 ERD

Example 2.10.3 *Technical Publication (A well known publishing company) publishes Engineering books on various subjects. The books are written by authors who specialize in his/her subject. The publication employs editors who take sole responsibility of editing one or more manuscripts. While writing a particular book, each author interacts with editor. But the author is supposed to submit his work to publisher always. In order improve competitiveness, the publication tries to employ a variety of authors who are specialist in more than one subject.*

Draw the E-R diagram for above described scenario.

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

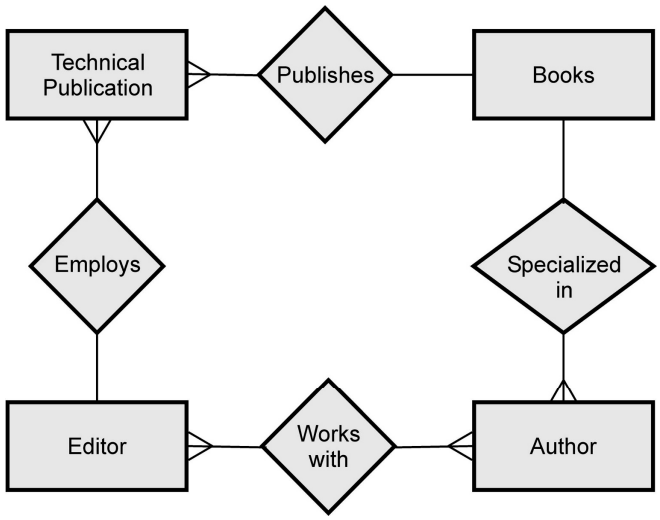


Fig. 2.10.8

Review Question

1. Explain data flow architecture style with neat diagram.

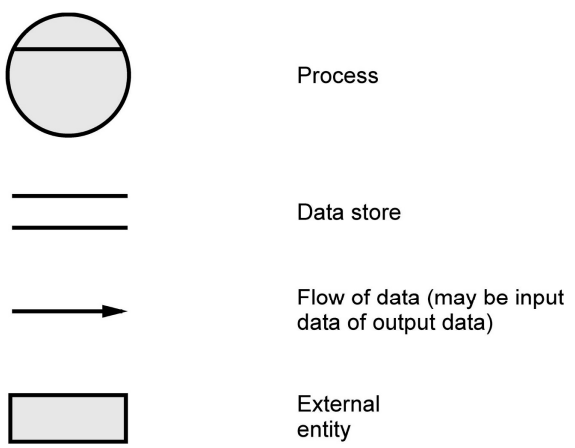
SPPU : Dec.-11, May-12, Dec.-14, Oct.-19, In Sem, Marks 5

2.11 Flow Modeling

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

2.11.1 Data Flow Diagram

- The data flow diagrams depict the information flow and the transforms that are applied on the data as it moves from input to output.
- The symbols that are used in data flow diagrams are -
- The data flow diagrams are used to represent the system at any level of abstraction.
- The DFD can be partitioned into levels that represent increase in information flow and detailed functionality.



- A level 0 DFD is called as 'fundamental system model' or 'context model'. In the context model the entire software system is represented by a bubble with input and output indicated by incoming and outgoing arrows.
- Each process shown in level 1 represents the sub functions of overall system.
- The number of levels in DFD can be increased until every process represents the basic functionality.
- As the number of levels gets increased in the DFD, each bubble gets refined. The following figure shows the leveling in DFD. Note that the information flow continuity must be maintained.

2.11.1.1 Designing Data Flow Diagrams

- The data flow diagrams are used to model the information and function domain. Refinement of DFD into greater levels helps the analyst to perform functional decomposition.
- The guideline for creating a DFD is as given below -
  1. Level 0 DFD i.e. Context level DFD should depict the system as a single bubble.
  2. Primary input and primary output should be carefully identified.
  3. While doing the refinement isolate processes, data objects and data stores to represent the next level.
  4. All the bubbles (processes) and arrows should be appropriately named.
  5. One bubble at a time should be refined.
  6. Information flow continuity must be maintained from level to level.
- A simple and effective approach to expand the level 0 DFD to level 1 is to perform "grammatical parse" on the problem description. Identify nouns and verbs from it. Typically nouns represent the external entities, data objects or data stores and verbs represent the processes. Although grammatical parsing is not a foolproof but we can gather much useful information to create the data flow diagrams.

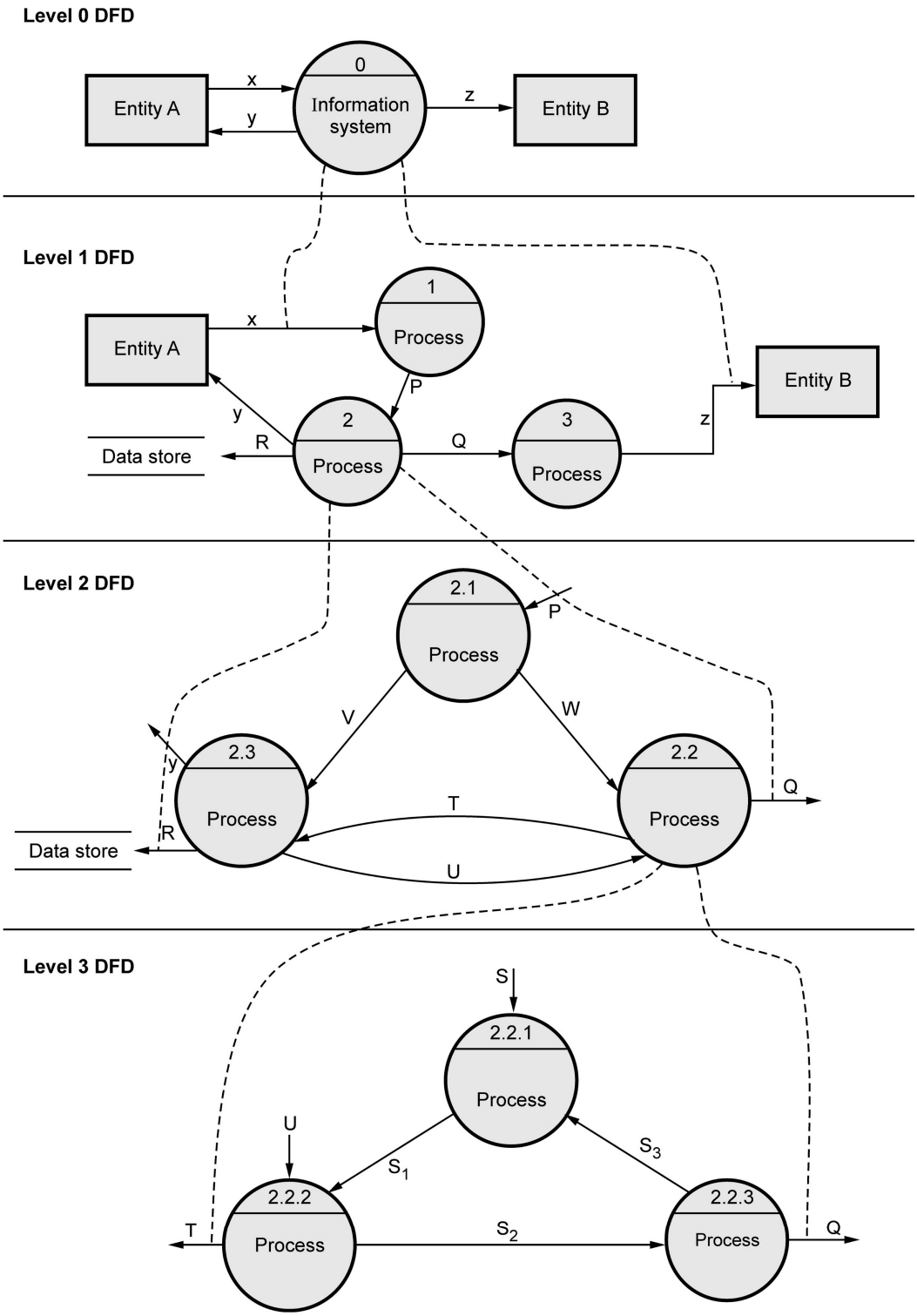


Fig. 2.11.1 Levels of DFD

Rules for designing DFD

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

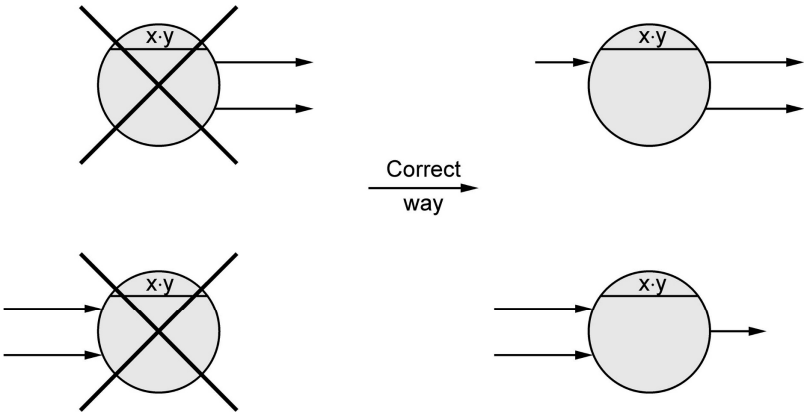


Fig. 2.11.2

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

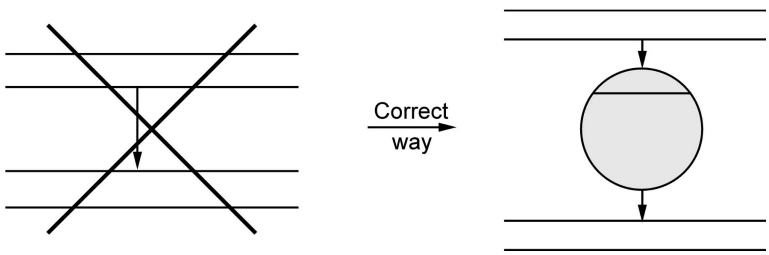


Fig. 2.11.3

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

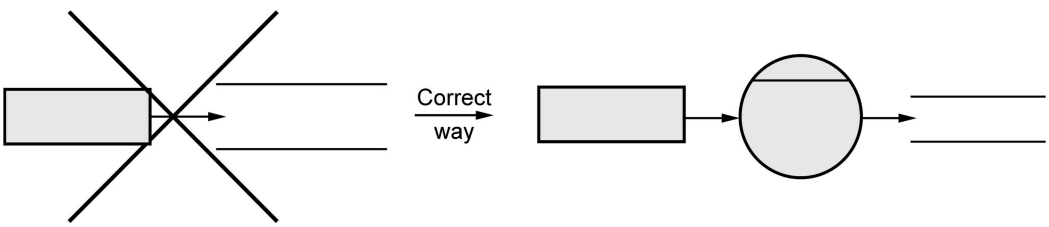


Fig. 2.11.4 Rules for DFD

4. Data store labels should be noun phrases from problem description.
5. No data should move directly between external entities. The data flow should go through a process.

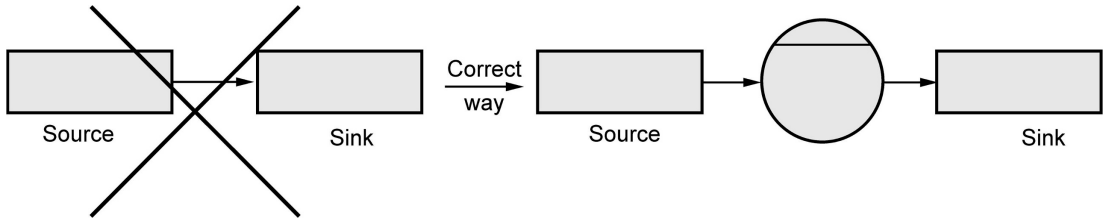


Fig. 2.11.5 Rule for DFD

6. Generally source and sink labels are noun phrases.
7. Interactions between external entities is outside scope of the system and therefore not represented in DFD.
8. Data flow from process to a data store is for updation/insertion/deletion.
9. Data flow from data store to process is for retrieving or using the information.
10. Data flow labels are noun phrases from problem description.

2.11.2 Control Flow Diagram

- The control flow diagrams show the same processes as in data flow diagrams but rather than showing data flow they show control flows.
- The control flow diagrams show how events flow among processes. It also shows how external events activate the processes.
- The dashed arrow is used to represent the control flow or event.
- A solid bar is used to represent the window. This window is used to control the processes used in the DFD based on the event that is passed through the window.
- Instead of representing control processes directly in the model the specifications are used to represent how the processes are controlled.
- There are two commonly used representations of specifications : Control Specification (CSPEC) and Process Specification (PSPEC).
- The CSPEC is used to indicate -
  1. How the software behaves when an event or signal is sensed ?
  2. Which processes are invoked as a consequence of the occurrence of event ?
- The PSPEC is used to describe the inner workings of the process represented in a flow diagram.

When a data input is given to the process a **data condition** should occur to get the control output. For Example.

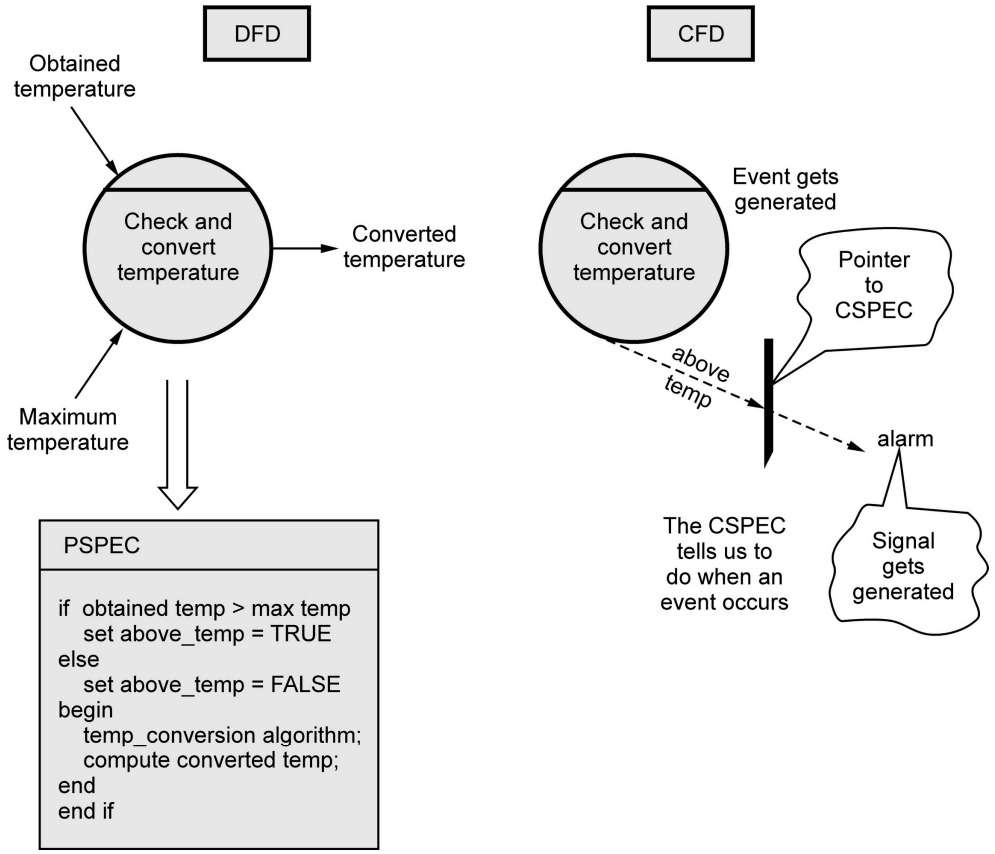


Fig. 2.11.6 Occurrence of data condition

2.11.2.1 Designing Control Flow Diagrams

- There are certain applications which are event driven rather than being data driven. They produce control information rather than producing data(may be report, displays). Such applications can be modeled with the control information along with data flow modeling.
- A graphical model used to represent the control information along with the data flow model is called control flow model.
- The following guideline is used while drawing the control flow diagrams.
  1. List all the sensors that can be read.
  2. List all the interrupt conditions.
  3. List all the data conditions.
  4. List all the switches actuated by the operator.
  5. Use noun/verb parsing technique to identify the control information.

6. Describe behavior of the system by identifying the states. Define the transition between the states.
7. Avoid common errors while specifying the control.

2.11.3 Examples

Example 2.11.1 *DFD for library information system. A student comes to a library for borrowing book. The student makes the book request by giving book title and author name. The student has to submit his library card to the library. Sometimes student may simply give topic and demand for a book (For example "just give me book on data structure"). The library information system maintains list of authors, list of titles, list of topics. This system also keeps record of topics on which books are available with the system. This system maintains information about shelf number on which books are located. Finally the list of demanded book should be displayed, on the console for ease of selection. Draw first level of DFD.*

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

i) Student requests for a book hence **Book request**.

ii) To show identity of the student he/she has to submit his/her Library card, hence **Library card**. The processing unit can be globally given as

Library information system

The system will produce following outputs -

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

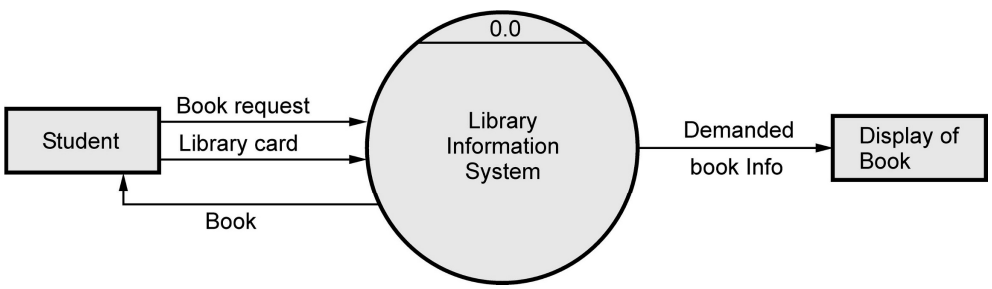


Fig. 2.11.7 Level 0 DFD (Context level DFD)

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

Level 1 DFD

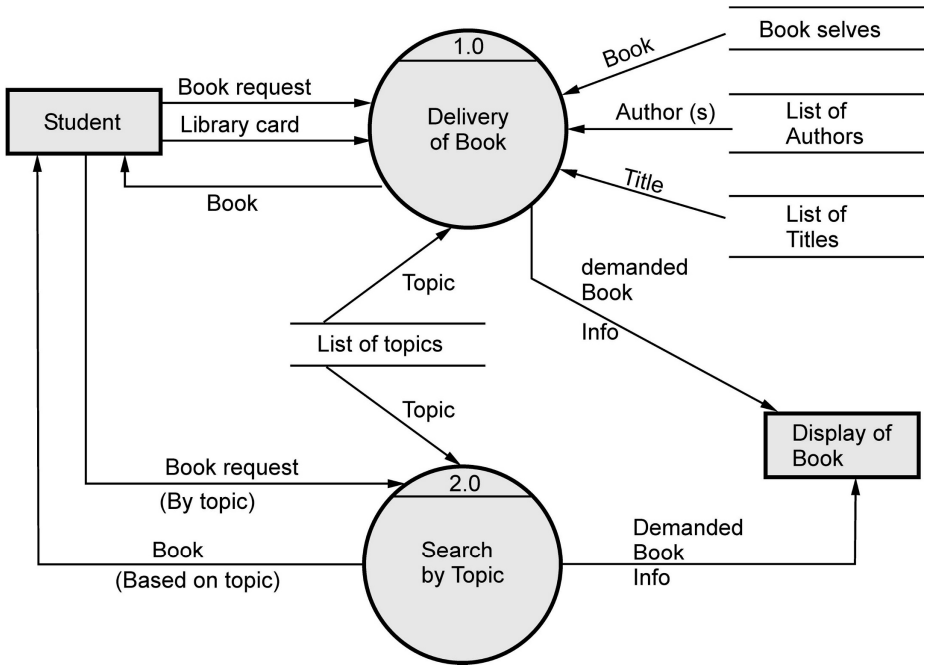


Fig. 2.11.8 Level 1 DFD

In this level, the system is exposed with more processing details. The processes that need to carried out are -

i) Delivery of Book.

ii) Search by Topic.

These processes require certain information such as List of Authors, List of Titles, List of Topics, the book selves from which books can be located. This kind of information is represented by **data store.**

Level 2 DFD

Out of scope :

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

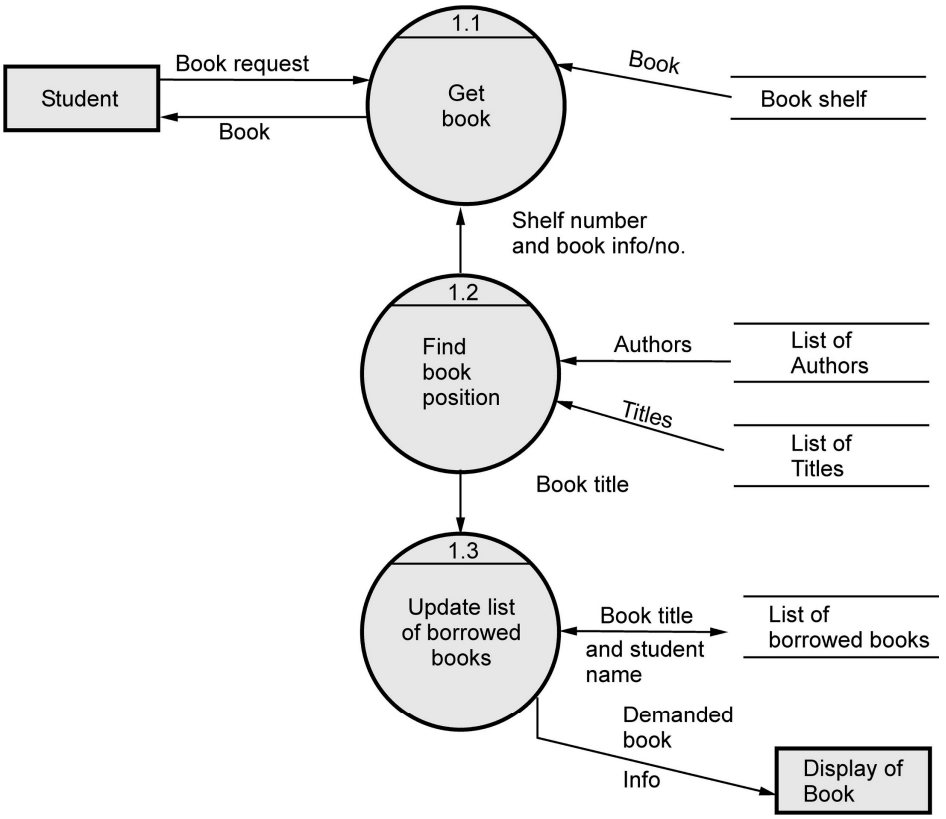


Fig. 2.11.9

Example 2.11.2 *Draw DFD for food ordering system. A* ***customer*** *goes to a restaurant and orders for the food. The food order is noted down carefully and this order is sent to kitchen for preparing the required food.*

This restaurant has to manage one housekeeping department which maintains ***sold items*** *and* ***inventory data.*** *The daily information about sold items and inventory deplition amount is used to generate a* ***management report****. Finally this management report is given to* ***restaurant manager****.*

Solution :

Level 0 DFD

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

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

   The output to food ordering system are -

   1) Receipt.

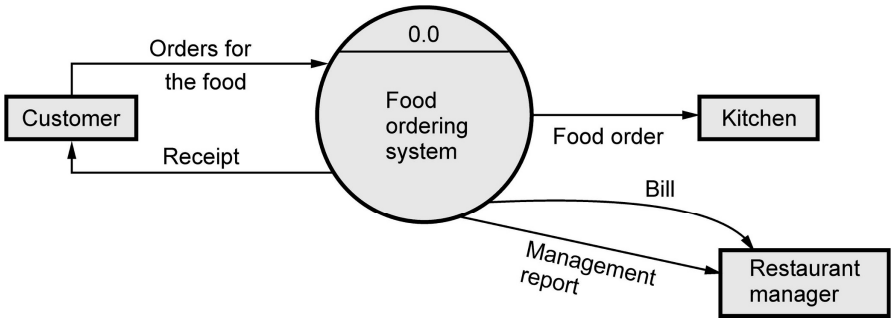


Fig. 2.11.10 Level 0 DFD (Context level DFD)

2) The food order should be further given to kitchen for processing the order.

3) Bill and management report is given to restaurant manager.

Level 1 DFD

In this level, the bubble 0.0 is shown in more detail by various processes. The process 1.0 is for processing an order. And processes 2.0, 3.0 and 4.0 are for housekeeping activities involved in food ordering system. To create a management report there should be some information of daily sold items. At the same time inventory data has to be maintained for keep track of 'instock' items. Hence we have used two **data stores** in this DFD -

1. Database of sold items

2. Inventory database.

Finally management report can be prepared using daily sold details and daily inventory deplition amount. This management report is given to restaurant manager.

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

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

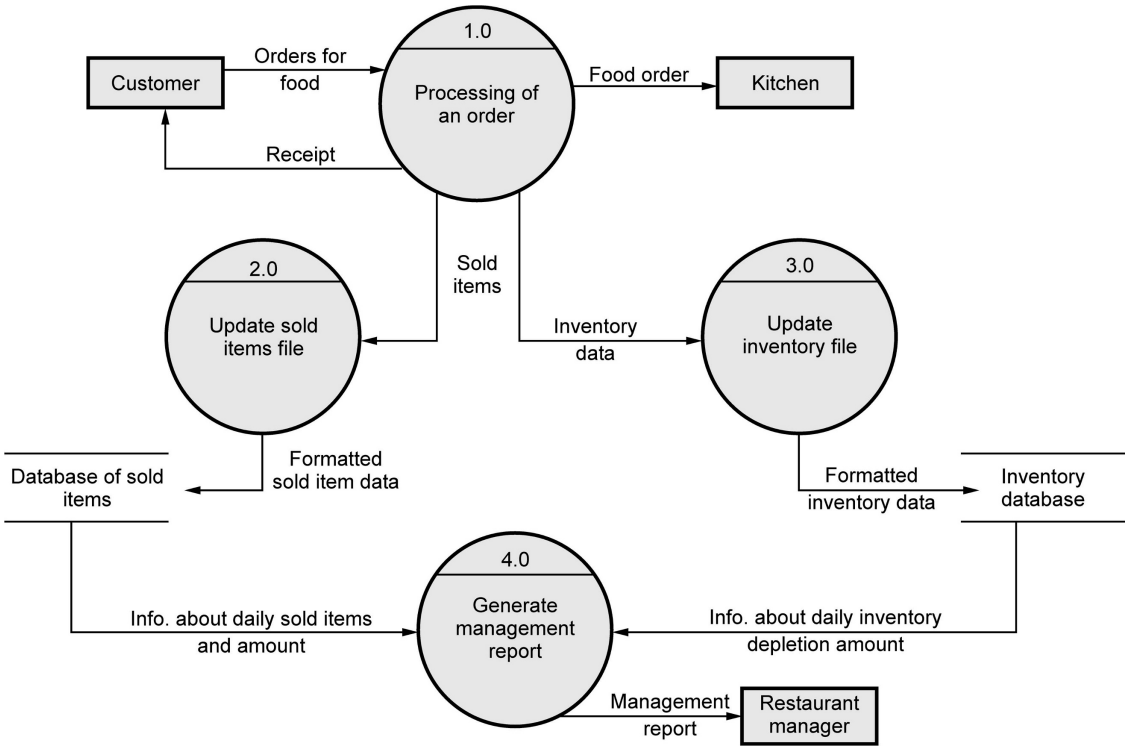


Fig. 2.11.11 Level 1 DFD

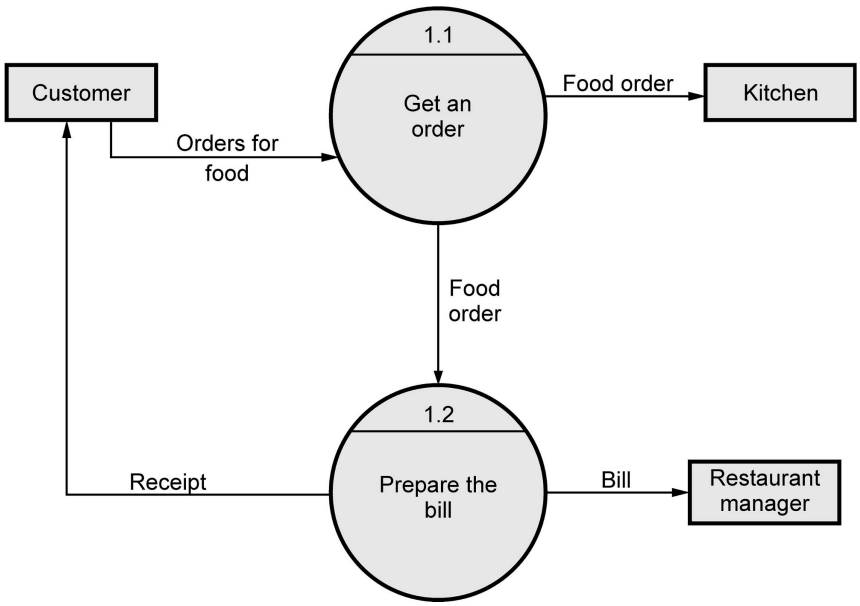


Fig. 2.11.12 Level 2 DFD

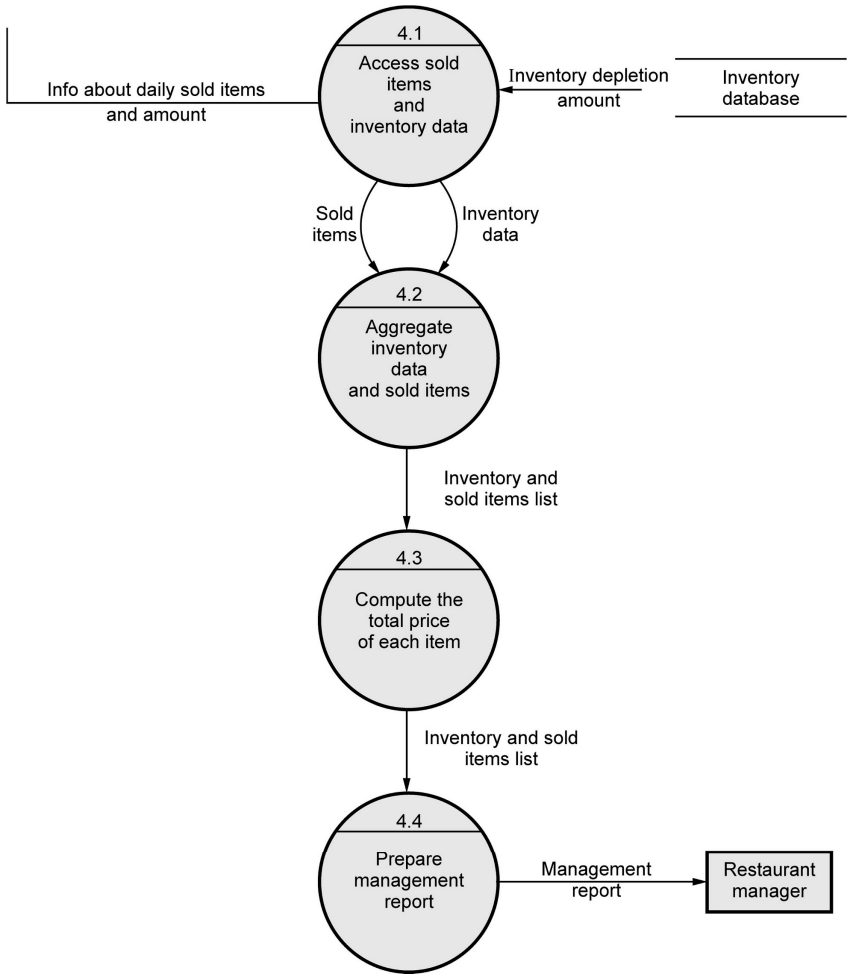


Fig. 5.11.13 Level 3 DFD

Example 2.11.3 *Prepare a CFD for a Books order processing system. The **customers** in this system are book sellers who do not make a stock of books. As the **orders** come the books are demanded from **publishers** directly.*

As per the problem description it is clear that the input to this system is 'customer' who places an order and output will be purchase order given to publisher.

Solution :

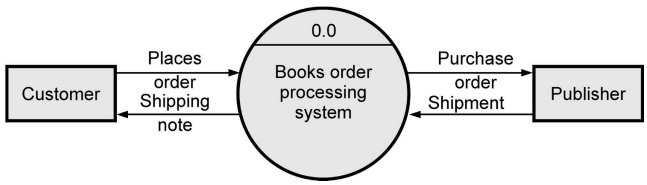


Fig. 2.11.14 Level 0 CFD

Now, we will do more detailing for order processing when an order is received by the system, it will be first verified using the books database. Credit rating will be decided by checking customer details (i.e. whether the customer is regular customer or whether he is new). For submitting a batch of orders we have to maintain pending orders list as well. There may be the order for books which are published by different publishers, in such a case database for different publishers need to be maintained by the system. This list of purchase orders is maintained by the system and then after verifying the shipment a shipping note will be given to the customer. The level 1 decomposition is as shown below –

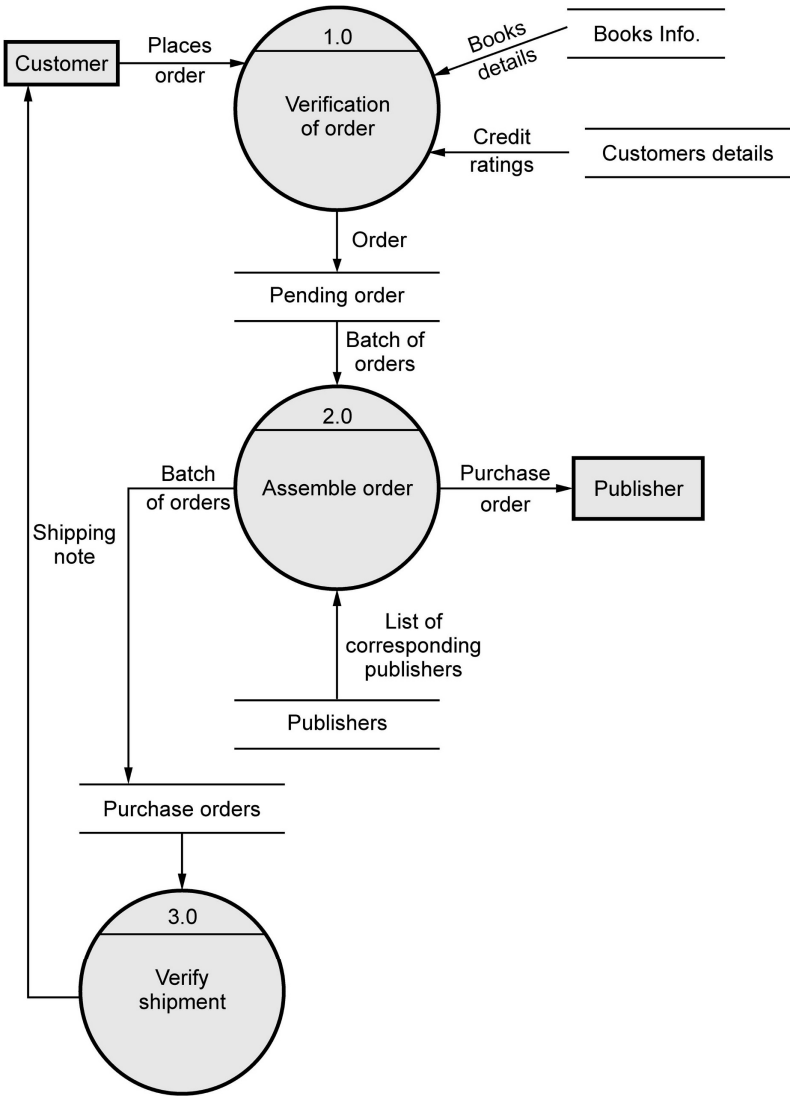


Fig. 2.11.15 Level 1 CFD

We can further decompose the system by elaborating the process **Assemble order**. In assemble order we

- First get details of total copies per title.
- Then prepare purchase order accordingly for corresponding publishers.
- Send these purchase orders to publishers.
- These purchase order details should be stored in the system. Let us draw level 2 CFD for these details.

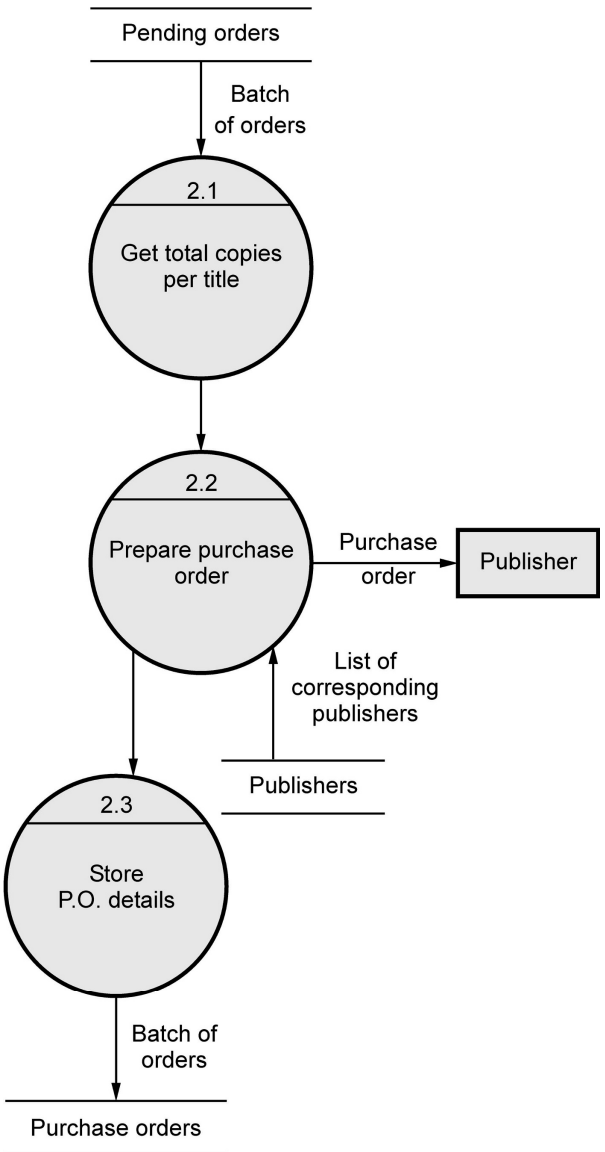


Fig. 2.11.16

Example 2.11.4 *In a hotel reservation system, a **customer** can make online booking for a hotel, by specifying the accomodation requirements such as type of room (AC/Non AC/One bed/Two bed), total number of rooms, duration of stay. The system then **selects** a suitable hotel as per customer's requirements. If such a hotel is found then the **availability** of rooms in that hotel is **checked.** The **charges are calculated** for the selected requirement and these are acknowledged to the customer. If the customer is satisfactory about the selection made by the system then he **confirms** the reservation. Draw Level 0, Level 1, Level 2 DFDs.*

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

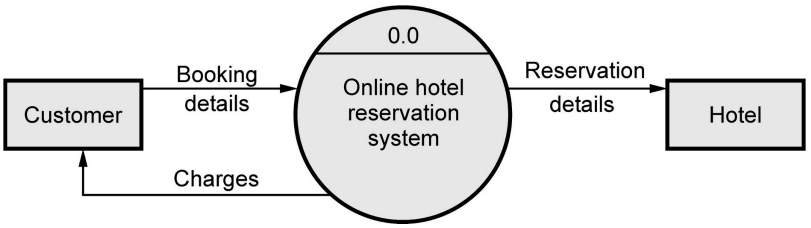


Fig. 2.11.17 Level 0 DFD

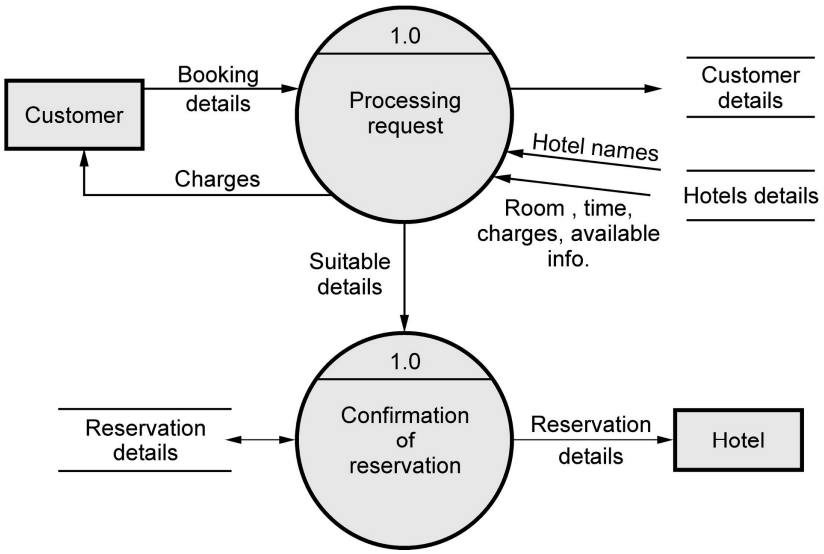


Fig. 2.11.18 Level 1 DFD

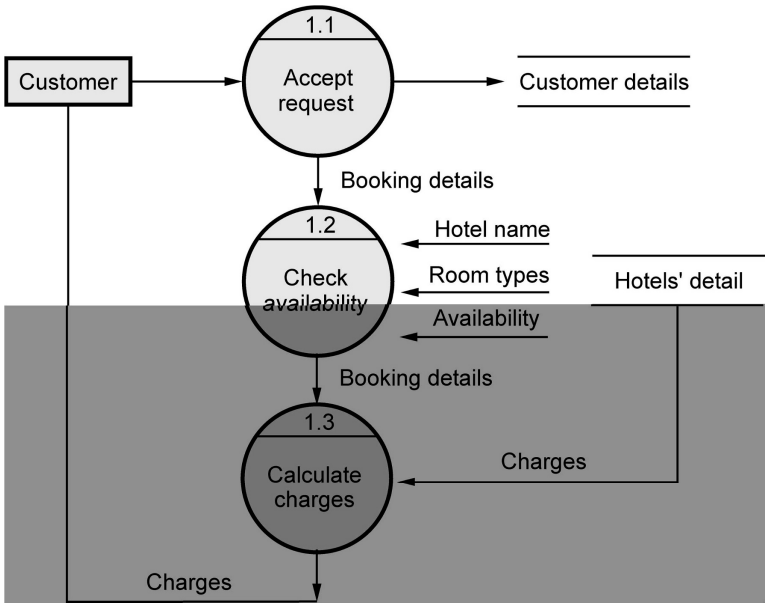


Fig. 2.11.19 Level 2 DFD

Example 2.11.5 *Based on your experience with a bank ATM draw a DFD modelling the processing involved when customer withdraws cash from the machine.*

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

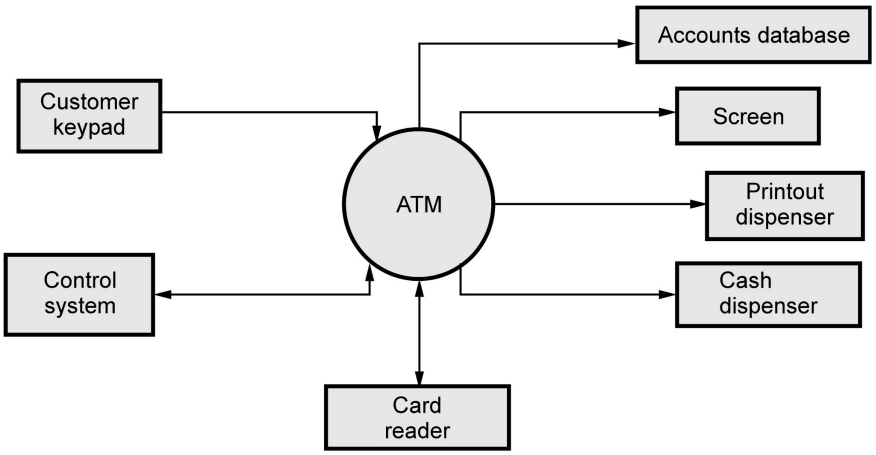


Fig. 2.11.20 Level 0 DFD

More detailing is done in level 1.

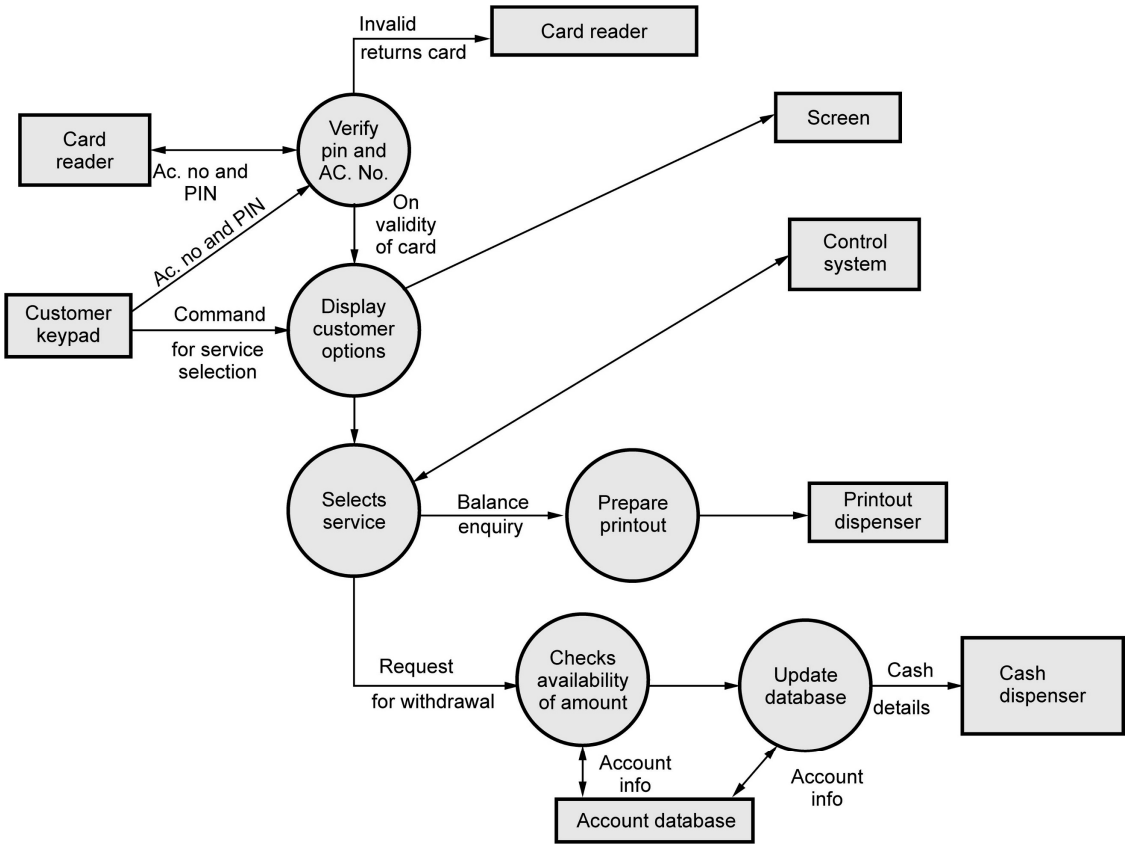


Fig. 2.11.21

Example 2.11.6 *Draw level 0 DFD and level 1 DFD for railway reservation system. Also write the data dictionary for the same.*

Solution :

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

The required DFD can be drawn in levels as follows –

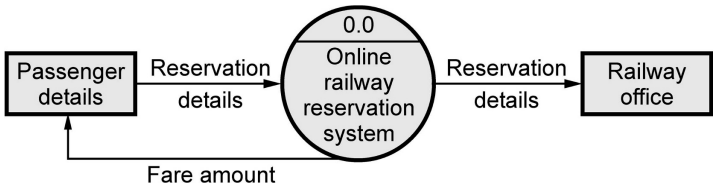


Fig. 2.11.22 Level 0 DFD

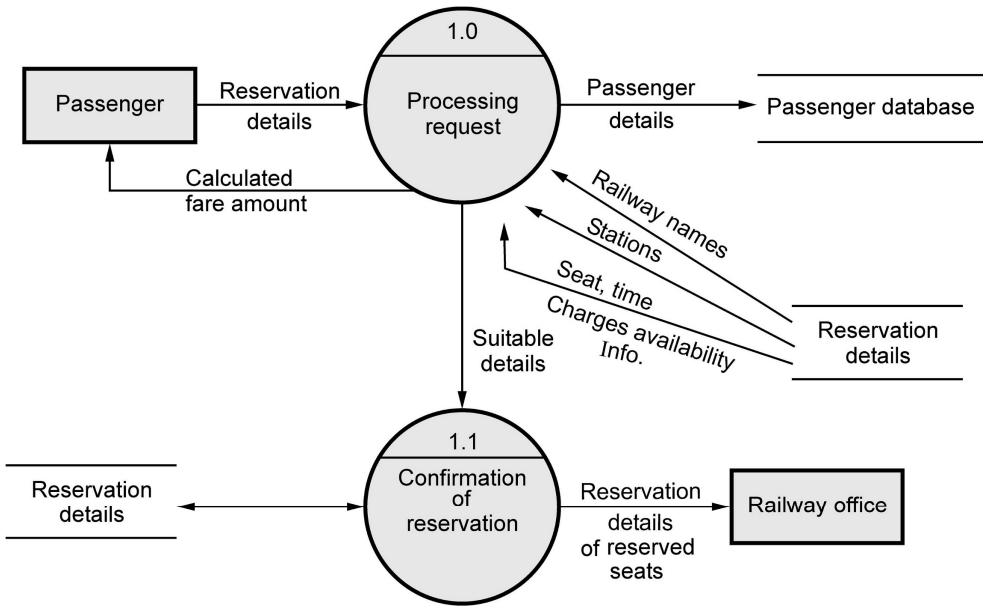


Fig. 2.11.23 Level 1 DFD

Data Dictionary

Name : Reservation Details or Ticket

Aliases : None

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

Description :

Passenger name	=	First name + Middle name + Surname.
Age	=	In number of years.
Sex	=	Male/Female.
Reservation number	=	Boggue number + Coach number
Starting location	=	Name of city
Destination location	=	Name of city
Train details	=	Train number + Name of the train + Up or down + Date and time of departure + Time of arrival
Fare	=	Total amount of fare for entire journey.

Example 2.11.7 *Consider a photocopying machine operated by software. Draw level 0, 1 and 2 DFD for this software.*

Solution : Assumptions made for software of photocopy machine : The photocopy machine will be controlled by some commands such as START, STOP or PAUSE. If user gives START command then required number of photocopies can be made by reloading papers. If paper gets jammed then diagnostic activities must be carried out and then reload the paper and then make the copies. The input to this system will be user commands and output will be photocopies.

The DFD for such a software will be –

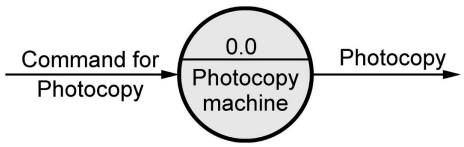


Fig. 2.11.24 Level 0 DFD

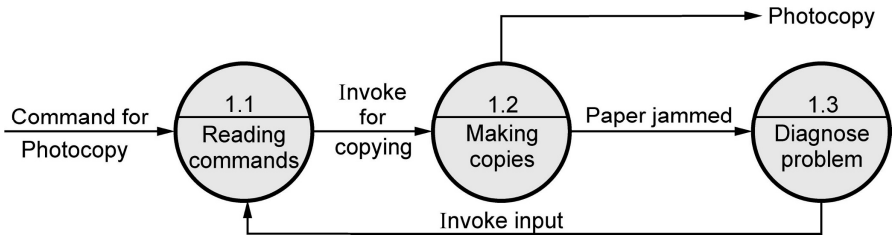


Fig. 2.11.25 Level 1 DFD

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

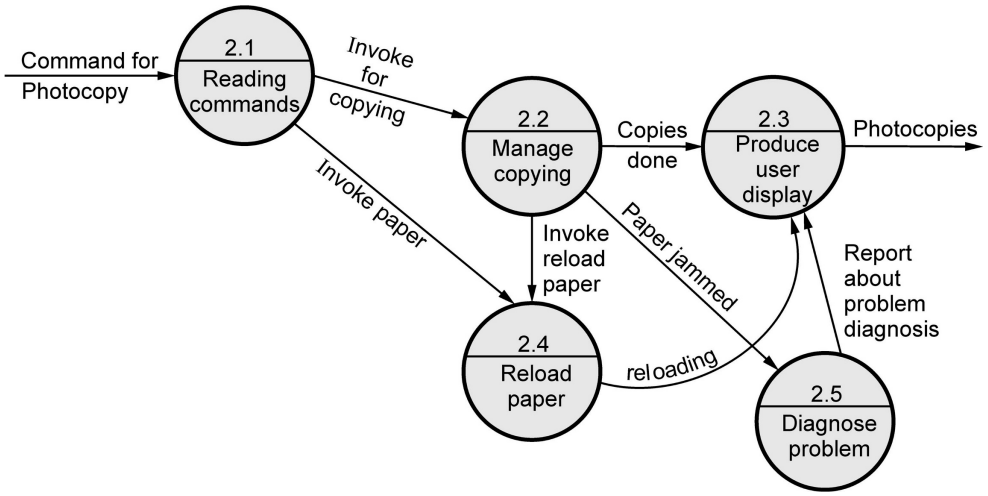


Fig. 2.11.26 Level 2 DFD

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

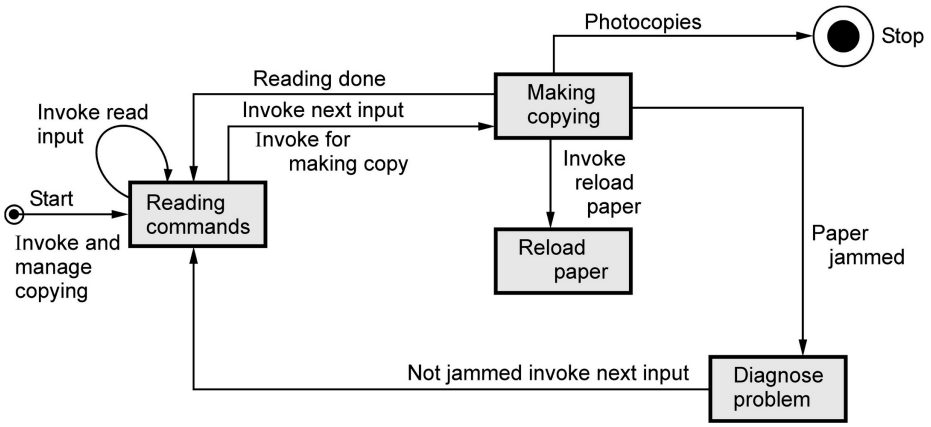


Fig. 2.11.27 STD

Example 2.11.8 *Draw the data flow diagram upto level 1, for a 'temperature monitoring system' in an intensive care unit of a hospital.* **SPPU : Dec.-11, Marks 8**

Solution : DFD Level 0

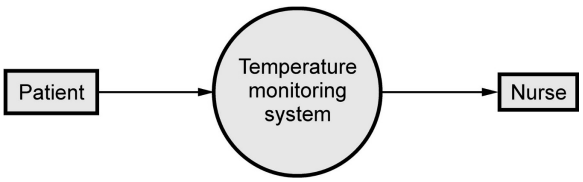


Fig. 2.11.28 DFD Level 0

DFD Level 1

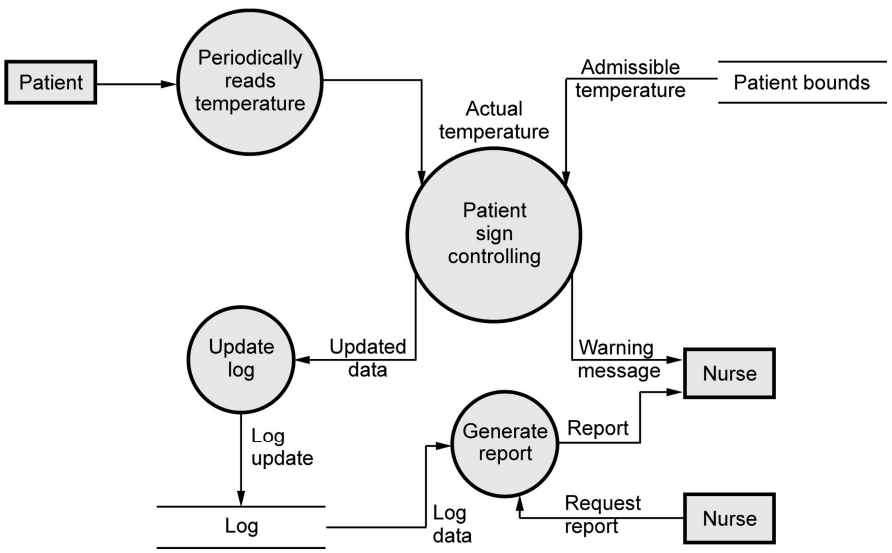


Fig. 2.11.29 DFD Level 1

Example 2.11.9 *Draw and explain context level, level 1 and level 2 DFD for college gathering.* **SPPU : May-12, Marks 8**

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

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

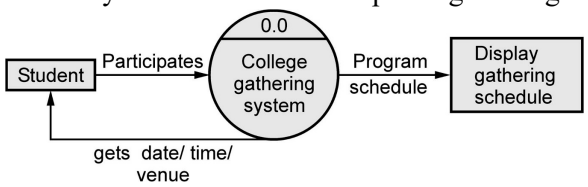


Fig. 2.11.30 Level 0 DFD

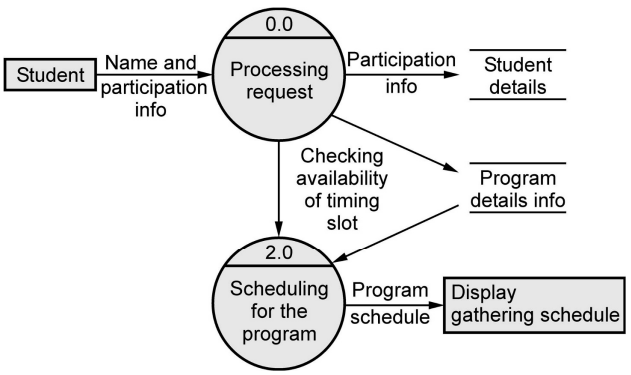


Fig. 2.11.31 Level 1 DFD

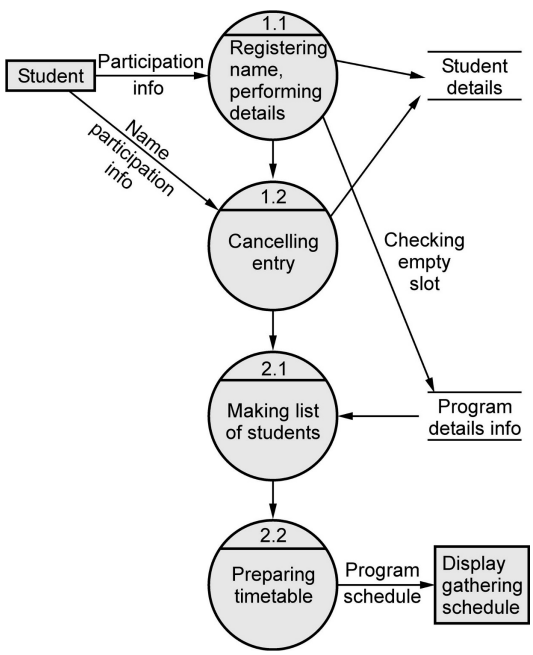


Fig. 2.11.32 Level 1 DFD

Example 2.11.10 *Draw a DFD for a vehicle renting system.* **SPPU : May-12, Marks 5**

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

DFD Level 0 :

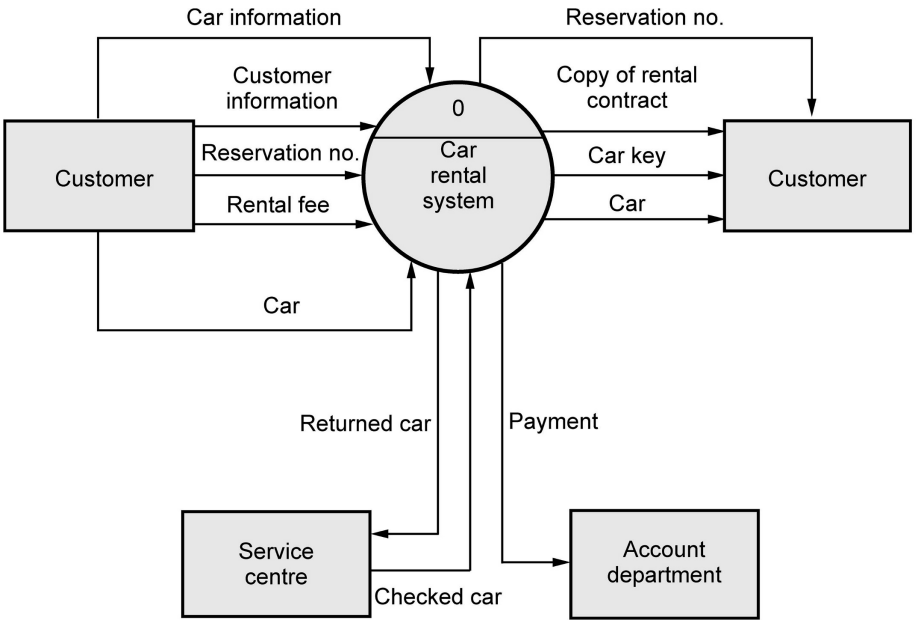


Fig. 2.11.33

DFD Level 1 : See Fig. 2.11.34 on next page.

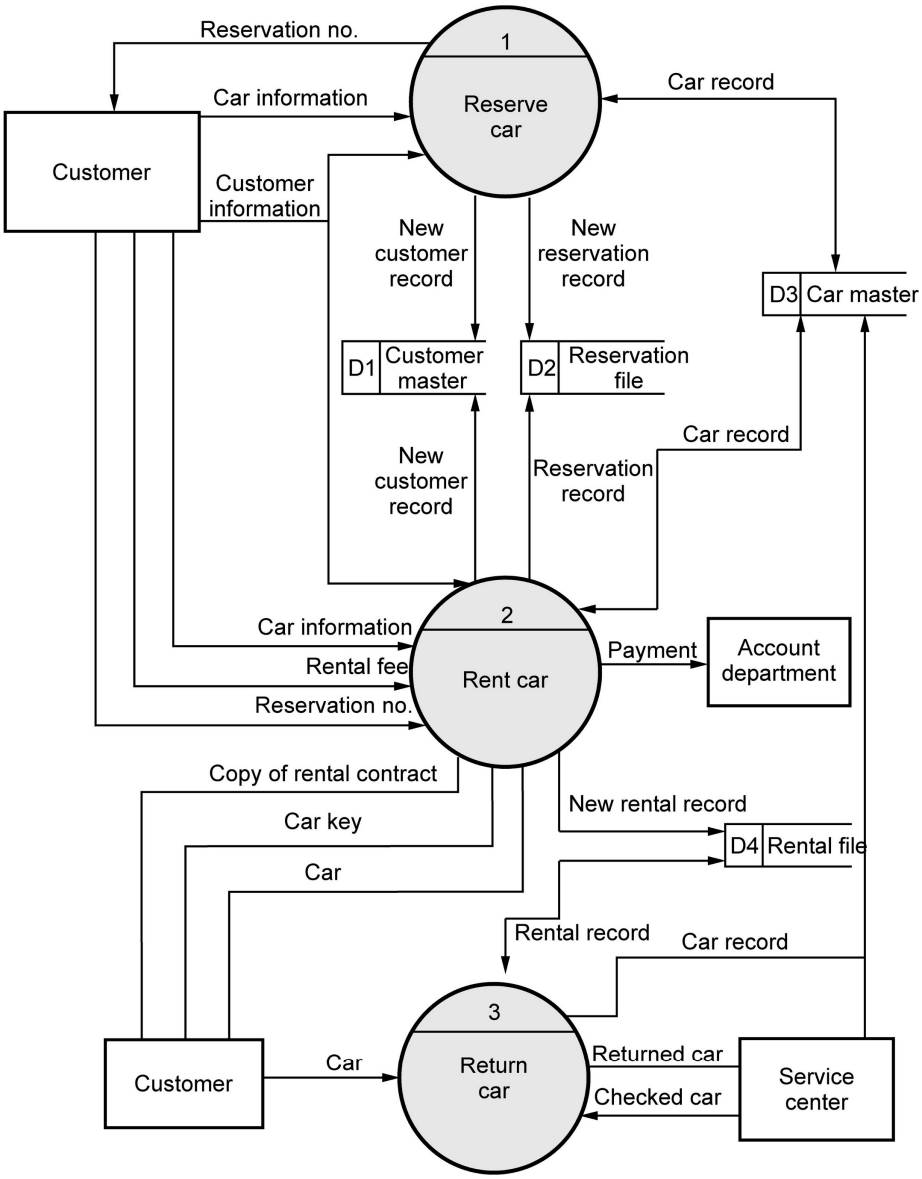


Fig. 2.11.34

Review Question

1. Explain the data flow model with example and diagram. **SPPU : Dec.-14, Marks 5**

2.12 Behavioral Modeling

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

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

2.12.1 Identifying Events with Use Cases

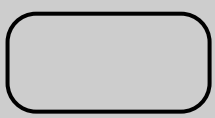
Use case represents the sequence of activities using the actors and the system. When an event occurs, then actor and system generally exchange information. For example -

For an ATM system, a customer, first of all *insert the card* and then uses a keypad to *enter the PIN*. If the *PIN is invalid* then the control panel will beep and eject the card. He then performs various operations such as *checking of balance, withdrawal of money* or deposition of money. While checking the balance the customer can *request* the banking system for issuing the *mini statement* of balance.

The underlined portions of the use case scenario indicate events. An actor for each event must be identified and the kind of information being exchanged between the user and the system must be noted. Any condition or constraints should be listed.

2.12.2 State Representation

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

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

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

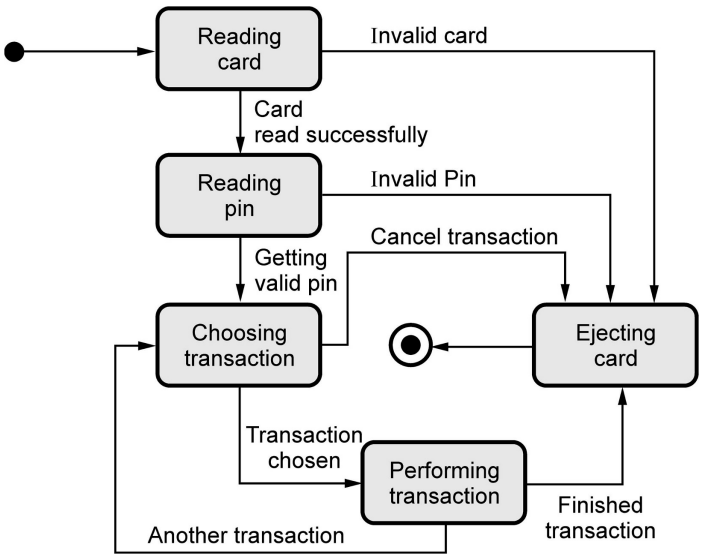


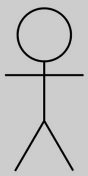
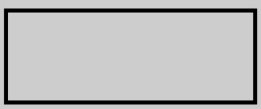
Fig. 2.12.1

2.12.3 Sequence Diagram

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

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

Various notations used in sequence diagram are

	User - who is responsible for activating various events.
	An active object who takes part in various operations.
	It represents the life line i.e. a time span of the object.

	Active procedure on which object is active.
	To interact two objects this kind of association is shown. Along with this association the message is given
	Destruction of object.

Example 2.12.1 *Draw a sequence diagram for ATM system.*

Solution : (See Fig. 2.12.2 on next page.)

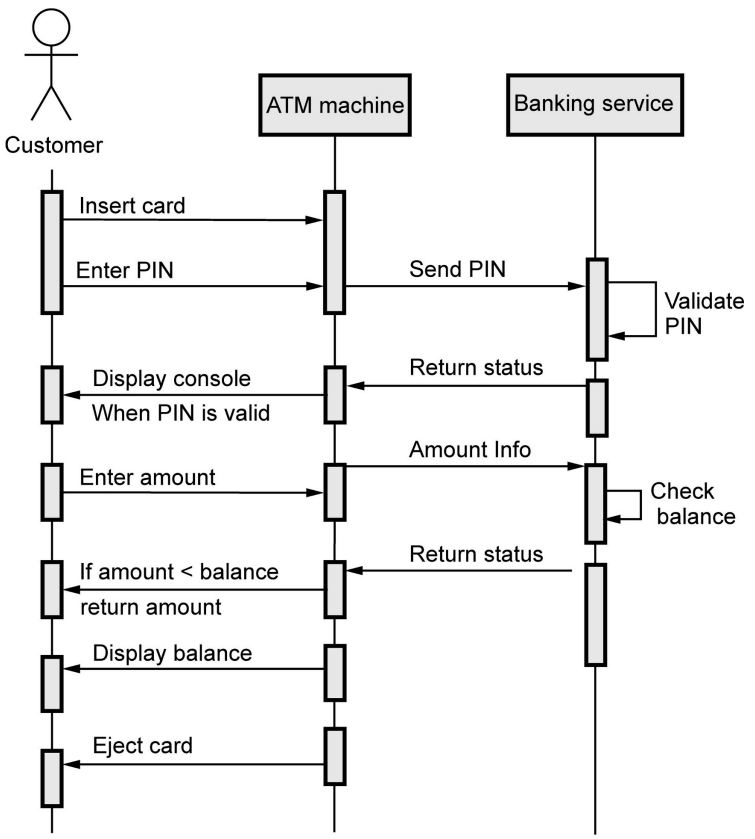


Fig. 2.12.2 Sequence diagram

2.13 Negotiating the Requirements

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

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

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

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

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

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

2.14 Validating Requirements

SPPU : Dec.-14, Marks 5

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

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

1. Does each requirement consistent ?
2. Are the requirements abstract ? That means - does the technical details of requirements are appropriate ?
3. Does particular requirement really necessary ?
4. Does each requirement bounded and unambiguous ?
5. Does the requirement have its reflection on information, function and behavior of the system ?

6. Does each requirement achievable by the software system ?
7. Does the requirement have some source ?
8. Does particular requirement conflict with some other requirement ?
9. Does each requirement tested after its implementation ?
10. Are the requirements used to simplify the requirements model ?

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

Review Question

1. What do you think happens when requirement validation uncovers an error ? Who is involved in correcting error ? **SPPU : Dec.-14, Marks 5**

2.15 Preparing Requirements Traceability Matrix

- Requirements are tracked by Requirements Traceability Matrix (RTM).
- RTM traces all the requirements from design, development, and testing.
- The traceability matrix is typically a worksheet that contains the requirements with its all possible test scenarios and cases and their current state, i.e. if they have been passed or failed. This would help the testing team to understand the level of testing activities done for the specific product.

Parameters to be included in requirement Traceability matrix

(1) Requirement ID

(2) Requirement type and description

(3) Test cases

(4) Test case status

Example

Test case ID	Test case description
TC#001	Login with invalid username but valid password
TC#002	Login with valid username but invalid password
TC#003	Login with valid user name and valid password

Requirement ID	Requirement Description	Testcase ID	Status
101	Login the website	TC#001	Pass
		TC#002	Pass
		TC#003	Pass

Advantages of requirements traceability matrix

1. It confirms 100% test coverage.
2. It highlights any requirements missing or document inconsistencies.
3. It shows the overall defects or execution status with a focus on business requirements.
4. It helps in analysing or estimating the impact on the QA team's work with respect to revisiting or re-working on the test cases.

2.16 Multiple Choice Questions

Q.1 Various tasks that are carried out during requirement engineering process are ___.

a feasibility study

b requirements gathering

c software requirements specification

d all of these

Q.2 _____ requirements describe the functionality or system services.

a System b Functional

c Non functional d User

Q.3 ____ requirements describe the system properties and constraints.

a System b Functional

c Non functional d User

Q.4 The goal of requirements analysis and specification is _____.

a to analyze the cost of the project

b to analyze the schedule of the project

c to understand the customer requirements and document them

d to determine the scope of the project

Q.5 The process to gather the software requirements from client, analyse and document them is known as _____ .

a requirement engineering
b requirement elicitation
c user interface requirements
d software system analyst

Q.6 _____ and _____ are the two issues of requirement analysis.

a Performance, design
b Stakeholder, developer
c Functional, non-functional

Q.7 The term _____ is used to refer to any person or group who will be affected by the system, directly or indirectly.

a user
b customer
c administrator
d stakeholder

Q.8 DFD stands for _____ .

a Data Flow Deployment
b Data Flow Design
c Data Flow Diagram
d Data Flow Drawing

Q.9 Which one of the following is a requirement that fits in a developer's module ?

a Availability
b Testability
c Usability
d Flexibility

Q.10 In the requirement analysis which model depicts the information domain for the problem ?

a Data models
b Class-oriented models
c Scenario-based models
d Flow-oriented models

Q.11 "Consider a system where a heat sensor detects an intrusion and alerts the security company ". What kind of requirement the system is providing ?

a Functional
b Non functional
c None of the above

Q.12 What DFD notation is represented by the rectangle ?

a Data flow
b Data store
c Process
d None of the mentioned

Q.13 In DFDs, user interactions with the system is denoted by _____.

a circle
b arrow
c rectangle
d triangle

Q.14 Behavioral model provides ____ view of the system.

a dynamic b static

c cost effective d none of these

Q.15 While dealing with the system requirements ___ and ___ factors are considered.

a coding and design b coding and maintenance

c behavioral and operational d none of these

Q.16 Inheritance is a ___ relationship.

a has a b type of

c uses d none of these

Q.17 ____ is more specifically called as generalization.

a Inheritance b Aggregation

c Composition d Association

Q.18 Sequence diagram are generally drawn for representing _____ .

a inheritance model b object aggregation

c object behavioral model d none of these

Q.19 _____ requirements constraint the system being developed and the development process that should be used.

a Functional b Nonfunctional

c User d System

Q.20 _____ model shows the principal sub-systems that make up a system.

a Architectural b Classification

c Composition d Data-flow

Q.21 Which of the following is not a check for requirements validation process ?

a Validity b Consistency

c Realism d Parallel

Q.22 What form does the requirements traceability matrix take ?

a Table or grid b Bulleted list

c Numbered list d Bar chart

Answer Keys for Multiple Choice Questions :

Q.1	d	**Q.2**	b	**Q.3**	c	**Q.4**	c
Q.5	a	**Q.6**	b	**Q.7**	d	**Q.8**	c
Q.9	b	**Q.10**	a	**Q.11**	a	**Q.12**	b
Q.13	a	**Q.14**	a	**Q.15**	c	**Q.16**	b
Q.17	a	**Q.18**	c	**Q.19**	b	**Q.20**	d
Q.21	d	**Q.22**	a				

❑❑❑

Notes

UNIT - III

3 Estimation and Scheduling

Syllabus

Estimation for Software Projects : *The Project Planning Process, Defining Software Scope and Checking Feasibility, Resources management, Reusable Software Resources, Environmental Resources, Software Project Estimation, Decomposition Techniques, Software Sizing, Problem-Based Estimation, LOC-Based Estimation, FP-Based Estimation, Object Point (OP)-based estimation, Process- Based Estimation, Estimation with Use Cases, Use-Case–Based Estimation, Reconciling Estimates, Empirical Estimation Models, The Structure of Estimation Models, The COCOMO II Mode, Preparing Requirement Traceability Matrix*

Project Scheduling : *Project Scheduling, Defining a Task for the Software Project, Scheduling.*

Contents

3.1 The Project Planning Process

- Project planning is an activity in which the project manager makes reasonable estimates of resources, cost and Schedule.
- In order to make these estimates, it is necessary to prepare a project plan and the beginning of the project and as the project proceeds this plan can be updated appropriately.
- Following figure represents various task sets that are associated with project planning process.

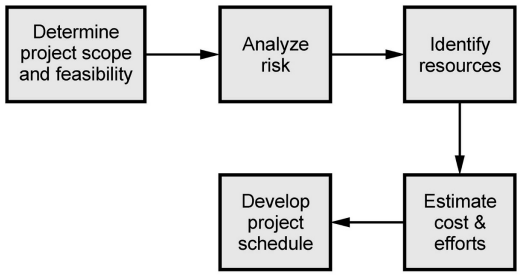


Fig. 3.1.1 Project planning process

Step 1 : This is an **initial stage** in which the scope and feasibility of the project is determined. This stage helps to identify the functions and features that can be delivered to the end user.

Step 2 : Risks are identified and analyzed.

Step 3 : The required **resources** such as human resources, environmental resources, and reusable software resources can be determined.

Step 4 : The **estimation for cost and efforts** is made. In this stage the problem is first decomposed, then using the size, function point or use cases the estimates are made.

Step 5 : Using the scheduling tools or task networks the **schedule** for the project is prepared.

3.2 Defining Software Scope and Checking Feasibility

SPPU : Dec.-17, May-18, Marks 7

Software scope describes four things -

- The **function** and **features** that are to be delivered to end-users.
  - The **data** which can be input and output.
  - The **content** of the software that is presented to user.
  - Performance, constraints, reliability and interfaces that bounds the System.
- There are two ways by which the scope can be defined -
  1. A scope can be defined using the narrative description of the software obtained after communication with all stakeholders.

2. Scope can be defined as a set of use cases developed by the end users.

- In the scope description, various **functions** are described. These functions are **evaluated** and **refined** to provide more details before the estimation of the project.
- For **performance consideration**, processing and response time requirements are analyzed.
- The constraints identify the limitations placed on the software by external hardware or any other existing system.
- After identifying the scope following questions must be asked -
  - Can we build the software to meet this scope ?
  - Is this software project feasible ?

  That means after identifying the scope of the project its feasibility must be ensured.
- Following are the four dimensions of software feasibility. To ensure the feasibility of the software project the set of questions based on these dimension has to be answered. Those are as given below -

1. Technology

- Is a project technically feasible ?
- Is it within the state of art ?
- Are the defects to be reduced to a level that satisfies the application's need ?

2. Finance

- Is it financially feasible ?
- Is the development cost completed at a cost of software organization, its client, or market affordable ?
- Are the defects to be reduced to a level that satisfies the application's need ?

3. Time

- Will the project's time to market beat the competition ?

4. Resource

- Does the organization have the resources needed to succeed ?
- **Putnam and Meyers** suggests that scoping is not enough. Once scope is understood, and and **feasibility** have been identified the next task is estimation of the **Resources** required to accomplish the software development effort.

Review Question

1. What is the need for defining a software scope ? What are the categories of software engineering resources ? **SPPU : Dec.-17,May-18, End Sem, Marks 7**

3.3 Resource Management

- The first task in project planning is to identify the scope and feasibility of the project and the second task is to estimate the resources required to accomplish the software development efforts.
- There are three major **categories** of software engineering resources -
  1. Peoples
  2. Reusable software components
  3. Development environment i.e. hardware and software tools.

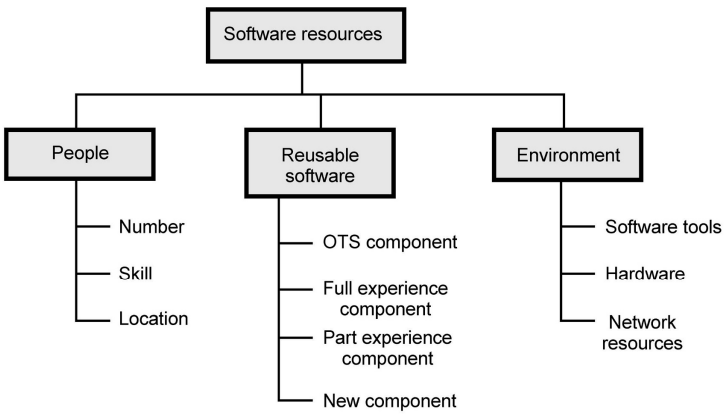


Fig. 3.3.1 Project resources

- There are four **characteristics** of above mentioned resources -
  1. **Description** of resource
  2. A statement of **availability**
  3. **Time** when resources will be required.
  4. **Duration** of the time that resource will be **applied.**

  The last two characteristics can be viewed as the **time window.**

3.3.1 Human Resources

- The Project Planner begins by **evaluating software scope** and **selecting the "skills"** required to complete development.
- Both **organizational position** (such as manager, senior software engineer and so on) and the **specialty** (database telecommunication, client/server etc) are specified for a relatively **small project**. A single individual may perform all software engineering tasks, by consulting with the specialist whenever needed.

- For **larger projects,** software team may be geographically at different locations. Hence, the location of each human resource is specified. The number of people required for a software project can be determined only after an estimate of **development effort** (e.g. person-months) is made. The techniques for estimating development effort can be LOC (Lines Of Code) based or based on FP (Function Point).

3.3.2 Reusable Software Resources

Component-Based Software Engineering (CBE) emphasizes one important characteristic and that is **Reusability**. That means by creating and reusing some building blocks the software development is done. Such building blocks are called **components**

There are **four reusable software resource categories** that should be considered at **planning-**

1. **Off-the-shelf components (OTS Components)**

   Existing software acquired from a third party or has been developed internally for a past project.

2. **Full experience components**

   The software to be built for the current project resembles the existing specification, design, code or test data developed for past projects. Members of the current software team that have had full experience in the application area represented by these components.

3. **Partial experience component**

   In this category, existing specifications as for full experience components but it requires extensive modification. The members of the software project team have only limited experience in the application area. Fair degree of risk involvement is there while using the components belonging to this category.

4. **New components**

   According to need and specification of the current project the software components can be built by the software team.

3.3.3 Environmental Resources

The **S**oftware **E**ngineering **E**nvironment (SEE) incorporates hardware and software.

Hardware provides a platform for using the tools required to produce work products. These work products are the outcome of good software Engineering practice.

3.4 Software Project Estimation

SPPU : May-19, Dec.-19, Marks 8

Software project estimation can be performed in systematic steps with acceptable risks. In order to obtain reliable cost and effort estimations various alternatives can be used –

1. Delay the estimation activity as far as possible.
2. The base estimates can be made using the similar projects.
3. Using simple decomposition techniques the project cost and effort estimates can be made.
4. Empirical models can be used for project cost and effort estimation.

For the first and second alternatives are simple. The third and fourth alternative suggests use of specific approach of project estimation techniques. Let us discuss these approaches of software project estimation in detail.

Review Question

1. What is need of project estimation? What are the steps while estimation of software?

SPPU : May-19, Dec.-19, End Sem, Marks 8

3.5 Decomposition Techniques

SPPU : May-11, 18, 19, Dec.-11, 12, 15, 19, Marks 9

Software project estimation is just similar to problem solving. Many times the problem to be solved is too complex in software engineering. Hence for solving such problems, we decompose the given problem into a set of smaller problems.

The decomposition can be done using two approached decomposition of problem or decomposition of process. Estimation uses one or both forms of decomposition (partitioning).

3.5.1 Software Sizing

- Following are certain **issues** based on which accuracy of software project estimate is predicated -
  1. The degree to which planner has properly estimated the size of the product to be built.
  2. The ability to translate the size estimate into human-effort, calendar time and money
  3. The degree to which the project plan reflects the abilities of software team.
  4. The stability of product requirements and the environment that supports the software engineering effort.
- Sizing represents the project planner's first major challenge. In the context of project planning, size refers to a quantifiable outcome of the software project.
- The sizing can be estimated using two approaches - **a direct approach** in which lines of code is considered and **an indirect approach** in which computation of function point is done.

- Putnam and Myers suggested four different approaches for sizing the problem -

1. Fuzzy logic sizing

In this approach planner must identify -

- The **type** of application
- Establish its **magnitude on a qualitative scale** and then refine the magnitude within the original range.
- Planner should also have **access to historical database** of the project so that estimates can be composed to actual experience.

2. Function point sizing

Planner develops estimates of the information domain.

3. Standard component sizing

- There are various **standard components** used in software. These components are subsystems, modules, screens, reports, interactive, programs, batch program, files, LOC (Lines Of Code) and Object-level instruction.
- The project planner estimates the number of time these standard components are used. He then uses historical project data to determine the delivered size per standard component.

4. Change sizing

- This approach is used when existing software has to be modified as per the requirement of the project.
- The size of the software is then estimated by the number and type of reuse, addition of code, change made in the code, deletion of code.
- The result of each sizing approaches must be combined statistically to create **three-point estimate** which is also known as **expected-value estimate.**

3.5.2 Problem Based Estimation

- The problem based estimation is conducted using LOC based estimation, FP (Function Point) based estimation, process based estimation and use cased based estimation.
- LOC and FP based data are used in two ways during software estimation -
  1. These are useful to estimate the **size** each element of software.
  2. The baseline metrics are collected from past project and LOC and FP data is used in conjunction with estimation variable to develop cost and effort values for the project. (LOC) and (FP) estimation are different estimation techniques. Yet, both have number of characteristics in common.

- With a bounded statement of software scope a project planning process begins and by using the statement of scope the software problem is decomposed into the functions that can be estimated individually.
- (LOC) or (FP) is then estimated for each function.
- **Baseline productively metrics** are then applied to the appropriate estimation variable and cost or effort for the function is derived.
- **Function estimates** are combined to obtain an overall estimate for the entire project.
- Using historical data the project planner **expected value** by considering following variables -

  1. Optimistic 2. Most likely 3. Pessimistic

  For example, following formula

$$S = [S_{opt} + 4 * S_m + S_{press}] / 6$$

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

3.5.3 Example of LOC Based Estimation

Consider an ABC project with some important modules such as

1. User interface and control facilities
2. 2D graphics analysis
3. 3D graphics analysis
4. Database management
5. Computer graphics display facility
6. Peripheral control function
7. Design analysis models

Estimate the project in based on LOC

For estimating the given application we consider each module as separate function and corresponding lines of code can be estimated in the following table as

Function	Estimated LOC
User Interface and Control Facilities(UICF)	2500
2D graphics analysis(2DGA)	5600
3D Geometric Analysis function(3DGA)	6450
Database Management(DBM)	3100
Computer Graphics Display Facility(CGDF)	4740
Peripheral Control Function(PCF)	2250
Design Analysis Modules (DAM)	7980
Total Estimation in LOC	**32620**

- Expected LOC for 3D Geometric analysis function based on three point estimation is -
  - Optimistic estimation 4700
  - Most likely estimation 6000
  - Pessimistic estimation 10000

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

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

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

  Then cost for lines of code can be estimated as

  cost / LOC = (6000 / 500) = $12

  By considering total estimated LOC as 32620
- Total estimated project cost = (32620 * 12) = $391440
- Total estimated project effort = (32620 / 500) = 65 Person-months

3.5.4 Example of FP Based Estimation

FP focuses on information domain values rather than software functions. Thus we create a function point calculation table for ABC project, discussed in section 3.3.3.

INFO DOMAIN VALUE	Opt.	most likely	pessimistic	esti. value	weight factor	FP
NO.OF INPUTS	25	28	32	28.1	4	112
NO. OF OUTPUTS	14	17	20	17	5	85
NO. OF INQURIES	17	23	30	23.1	5	116
NO. OF FILES	5	5	7	5.33	10	53
NO.OF EXTERNAL INTERFACES	2	2	3	2	7	15
COUNT TOTAL						**381**

- For this example we assume average complexity weighting factor.
- Each of the complexity weighting factor is estimated and the complexity adjustment factor is computed using the complexity factor table below. (Based on the 14 questions)

Sr. No.	FACTOR	VALUE (Fi).
1.	Back-up and recovery ?	4
2.	Data communication ?	2
3.	Distributed processing ?	0
4.	Performance critical ?	4

5.	Existing operational environment ?	3
6.	On-line data entry ?	4
7.	Input transactions over multiple screens ?	5
8.	Online updates ?	3
9.	Information domain values complex ?	5
10	Internal processing complex ?	5
11.	Code designed for reuse ?	4
12.	Conversion / installation in design ?	3
13.	Multiple installations ?	5
14.	Application designed for change ?	5

$$\Sigma (F_i) \rightarrow 52$$

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

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

FP ESTIMATED = COUNT TOTAL * [0.65 + (0.01 * ? Σ (Fi))]

Complexity adjustment factor = [0.65 + (0.01 * 52)] = 1.17

- **FP ESTIMATED = (381 * 1.17) = 446** (Function point count adjusted with complexity adjustment factor)
- A review of historical data indicates -
  1. Average productivity is 6.5 FP/Person month
  2. Average labor cost is $6000 per month

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

Review Questions

1. Explain the FP based estimation technique. **SPPU : May-18, End Sem, Marks 6**
2. Compare the Lines of Code(LOC) and Function Point(FP) based estimation techniques with suitable example. **SPPU : May-19, End Sem, Marks 8**
3. How to calculate FP and how it is used in estimation of software project ? **SPPU : Dec.-19, End Sem, Marks 5**

3.5.5 Process Based Estimation

- This is the most commonly used estimation technique for estimating the project based on the processes used.
- In this technique, the process is decomposed into relatively size set of tasks and the effort required to accomplish each task is estimated.

- The estimation begins with identification of software functions obtained from the project scope. Then a series of software process activities must be performed for each function. The function and software process activities are presented in a tabular form. Then planner estimates the efforts for each software function.

3.5.6 Example of Process Based Estimation

Consider the same project described in section 3.5.3. We can estimate it using process based estimation technique by creating a table as shown below. In this technique for each of the business function software engineering tasks such as analysis, design, coding and testing is carried out. The estimated efforts for each of these functions are identified and their sum is obtained. In the following table, the sum of all the rows and columns is taken and the effort for each business function is estimated in percentage.

ACTIVITY	CC	Planning	Risk analysis	Engineering		Construction release		CE	TOTAL
TASK				**Analysis**	**Design**	**Code**	**Test**		
BUSINESS FUNCTION									
User interface and control facilities(UICF)				0.50	2.75	0.50	4.50	NA	8.25
2D graphics analysis(2DGA)				0.75	4.50	0.75	2.00	NA	8.00
3D Geometric analysis function(3DGA)				0.50	4.50	1.00	3.50	NA	9.5
Database Management (DBM)				0.50	3.25	1.00	1.00	NA	5.75
Computer graphics display				0.50	3.25	0.70	1.50	NA	5.95
Facility(CGDF)									
Peripheral control function(PCF)				0.25	2.00	0.50	1.50	NA	4.25
Design Analysis Modules (DAM)				0.50	2.00	0.50	2.00	NA	5.00
Total	**0.25**	**0.25**	**0.25**	**3.50**	**22.25**	**4.95**	**16.00**		**48**
Percentage effort	**1 %**	**1 %**	**1 %**	**7 %**	**46 %**	**11 %**	**33 %**		

CC means customer communication CE means customer evaluation

The percentage effort is calculated based on the total 48.

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

22.25 ∗ 100 / 48 = 46 % approximately

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

Based on average labor cost of $6000 per month

1. Total estimated project Effort = **48 persons-month**
2. Total estimated project cost = (6000 ∗ 48) = $288000

3.5.7 Estimation with Use Case

- Use-cases also provide a software team with insight into software **scope and requirements.** But there are some difficulties in developing estimations using use cases due to the following reasons
  - There are **varieties of ways** by which use cases can be described. There is no unique format for them.
  - Use-cases represent an **external view** (User's view) of the software. These are also written at different levels of **abstraction**. That means some use cases may be written to expose only primary functions where as some other use cases may describe each function in more detail.
  - The **complexity** of the functions and features cannot be described by use cases.
  - The **complex behaviour** involved in many functions and features is also not described in use cases.
- In spite of these difficulties the use cases can be used for estimation, but only if they are considered within the context of **structured hierarchy** that use cases describe.

  Smith argues that any level of structured hierarchy -

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

- Following formula is useful to compute this estimation -

$$\textbf{LOC estimates} = N * LOC_{avg} + [(Sa / Sh - 1) + (Pa / Ph - 1)] * LOC_{adjust}$$

where
- N = Actual number of Use-cases
- LOC_{avg} = Historical average LOC/Use-case for this type subsystem
- LOC_{adjust} = Adjustment based on 'n %' of LOC_{avg}
- n = Defined locally and represents the difference between this project and 'Average' project
- Sa = Actual scenarios per use-case
- Sh = Average scenarios per use-case for this type subsystem
- Pa = Actual pages per use-case
- Ph = Average pages per use-case for this type of subsystem

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

3.5.8 Reconciling Estimation

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

LOC = 68 person-month
FP = 69 person-month
Process based = 48 person-month

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

Review Questions

1. What is software project estimation ? How use cases are used in estimation ? **SPPU : May-11, Marks 6**
2. What is LOC-based estimation ? Explain with a suitable example. **SPPU : Dec.-11, Marks 8**
3. What do you mean by software project estimation ? Explain the process based estimation with an example. **SPPU : Dec.-12, Marks 8**
4. What is project decomposition ? What are the work tasks for communication process using process decomposition. **SPPU : Dec.-15, End Sem, Marks 9**

3.6 Empirical Estimation Model

SPPU : Dec.-11, 16, 17, 18, May-16, Marks 8

Empirical estimation technique involves an educated guess of project parameters. Hence empirical estimation model is based on common sense.

Values for LOC on FP (Estimated Values) using **Three-Points on Expected Value Estimate,** are used into the Estimation model.

The empirical data are derived from a limited sample of projects. Hence this estimation model is appropriate for all classes of software product. Therefore, the results obtained from such models must be used judiciously.

3.6.1 The Structure of Estimation Model

This is a **typical Estimation Model**. It is derived using **regression analysis** on data collected from **past software projects.** The estimated efforts can be denoted by following formula -

$$E = A + B * (e_v)^C$$

where

A, B and C are the constants derived empirically.

E is the effort in persons-months

e_v is the estimation variable derived from either LOC or FP.

There are many other estimation models that have some form of project adjustment components that enables the value of E to be adjusted by other projects characteristic.

- Various LOC based estimation models are -

Name of the model	Estimation
Walston-Felix model	$E = 5.2 * (KLOC)^{0.91}$
Bailey-Basili model	$E = 5.5 + 0.73 * (KLOC)^{1.16}$
Boehm simple model	$E = 3.2 * (KLOC)^{1.05}$
Doty model	$E = 5.288 * (KLOC)^{1.047}$

- Various FP oriented estimation models are -

Name of the model	Estimation
Albrecht and Gaffney model	E = – 91.4 + 0.355 FP
Kemerer model	E = – 37 + 0.96 FP
Small project regression model	E = – 12.88 + 0.405 FP

Each of these Models shows different result for the same value of LOC an FP. Therefore estimation model must be calibrated for local models.

3.6.2 The COCOMO II Model

COCOMO II is applied for modern software development practices addressed for the projects in 1990's and 2000's.

The sub-models of COCOMO II model are -

1. Application composition model

- For estimating the efforts required for the prototyping projects and the projects in which the existing software components are used application-composition model is introduced.
- The estimation in this model is based on the number of application points. The application points are similar to the object points.
- This estimation is based on the level of difficulty of object points. Boehm has suggested the object point productivity in the following manner.

Developers experience and capability	Very low	Low	Nominal	High	Very high
CASE maturity	Very low	Low	Nominal	High	Very high
Productivity (NOP/Month)	4	7	13	25	50

- Effort computation in application-composition model can be done as follows -

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

where

PM means effort required in terms of person-months.

NAP means number of application points required.

% reuse indicates the amount of reused components in the project. These reusable components can be screens, reports or the modules used in previous projects.

PROD is the object point productivity. These values are given in the above table.

2. An early design model

- This model is used in the early stage of the project development. That is after gathering the user requirements and before the project development actually starts, this model is used. Hence approximate cost estimation can be made in this model.

- The estimation can be made based on the functional points.
- In early stage of development different ways of implementing user requirements can be estimated.
- The effort estimation (in terms of person month) in this model can be made using the following formula :

$$\textbf{Effort} = \textbf{A} \times \textbf{size}^{\textbf{B}} \times \textbf{M}$$

where

Boehm has proposed the value of coefficient **A = 2.94**.

Size should be in terms of Kilo source lines of code i.e. KSLOC.

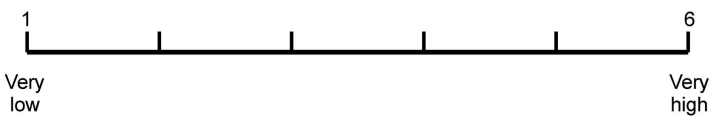
The lines of code can be computed with the help of function point.

The value of **B** is varying from 1.1 to 1.24 and depends upon the project.

M is based on the characteristics such as

1. Product reliability and complexity (RCPX)
2. Reuse required (RUSE)
3. Platform difficulty (PDIF)
4. Personnel capability (PERS)
5. Personnel experience (PREX)
6. Schedule (SCED)
7. Support facilities (FCIL)

These characteristics values can be computed on the following scale -



- Hence the effort estimation can be given as

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

$$\textbf{M} = \textbf{RUSE} \times \textbf{PDIF} \times \textbf{PERS} \times \textbf{PREX} \times \textbf{SCED} \times \textbf{FCIL}$$

3. A reuse model

- This model considers the systems that have significant amount of code which is reused from the earlier software systems. The estimation made in reuse model is nothing but the efforts required to integrated the reused models into the new systems.
- There are two types of reusable codes : black box code and white box code. The black box code is a kind of code which is simply integrated with the new system without modifying it. The white box code is a kind of code that has to be modified to some extent before integrating it with the new system, and then only it can work correctly.

- There is third category of code which is used in reuse model and it is the code which can be generated automatically. In this form of reuse the standard templates are integrated in the generator. To these generators, the system model is given as input from which some additional information about the system is taken and the code can be generated using the templates.
- The efforts required for automatically the generated code is

$$PM = (ASLOC \times AT / 100) / ATPROD$$

where

AT is percentage of automatically generated code.

ATPROD is the productivity of engineers in estimating such code

- Sometimes in the reuse model some white box code is used along with the newly developed code. The size estimate of newly developed code is equivalent to the reused code. Following formula is used to calculate the effort in such a case -

$$ESLOC = ASLOC \times (1 - AT / 100) \times AAM$$

where

ESLOC means equivalent number of lines of new source code.

ASLOC means the source lines of code in the component that has to be adapted.

AAM is adaptation Adjustment multiplier. This factor is used to take into the efforts required to reuse the code.

4. Post architecture model

- This model is a detailed model used to compute the efforts. The basic formula used in this model is

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

- In this model efforts should be estimated more accurately. In the above formula **A** is the amount of code. This code size estimate is made with the help of three components -
  1. The estimate about **new** lines of code that is added in the program.
  2. Equivalent number of Source Lines Of Code (ESLOC) used in **reuse model**.
  3. Due to changes in requirements the lines of code get modified. The estimate of amount of **code** being **modified**.

The exponent term **B** has three possible values that are related to the levels of project complexity. The values of B are continuous rather than discrete. It depends upon the five scale factors. These scale factors vary from very low to extra high (i.e. from 5 to 0).

- These factors are -

Scale factor for component B	Description
Precedentedness	This factor is for previous experience of organisation. Very low means no previous experience and high means the organisation knows the application domain.
Development flexibility	Flexibility in development process. Very low means the typical process is used. Extra high means client is responsible for defining the process goals.
Architecture/risk resolution	Amount of risk that is allowed to carry out. Very low means little risk analysis is permitted and extra high means high risk analysis is made.
Team cohesion	Represents the working environment of the team. Very low cohesion means poor communication or interaction between the team members and extra high means there is no communication problem and team can work in a good spirit.
Process maturity	This factor affects the process maturity of the organisation. This value can be computed using Capacity Maturity Model (CMM) questionnaire, for computing the estimates CMM maturity level can be subtracted from 5.

- Add up all these rating and then whatever value you get, divide it by 100. Then add the resultant value to 1.01 to get the exponent value.
- This model makes use of 17 cost attributes instead of seven. These attributes are used to adjust initial estimate.

Cost attribute	Type of attribute	Purpose
RELY	Product	System reliability that is required.
CPLX	Product	Complexity of system modules
DATA	Product	Size of the data used from database
DOCU	Product	Some amount of documentation used
RUSE	Product	Percentage of reusable components
TIME	Computer	Amount of time required for execution
PVOL	Computer	Volatility of development platform.
STOR	Computer	Memory constraint
ACAP	Personnel	Project analyst's capability to analyse the project.
PCAP	Personnel	Programmer capability

PCON	Personnel	Personnel continuity
PEXP	Personnel	Programmer's experience in project domain.
LTEX	Personnel	Experience of languages and tools that are used.
AEXP	Personal	Analyst's experience in project domain.
TOOL	Project	Use of software tools
SCED	Project	Project schedule compression.
SITE	Project	Quality of inter-site and multi-site working.

Review Questions

1. Explain COCOMO II model. **SPPU : Dec.-11, Marks 8, Dec.-16, End. Sem, Marks 5**
2. Explain COCOMO model for project estimation with suitable example

SPPU : May-16, Dec.-17, 18, End Sem, Marks 7

3.7 Object Point(OP) Based Estimation

Lorenz and Kidd suggested the estimation technique for object oriented project. The steps for this estimation technique are as follows -

1. Using requirement model, develop use cases and determine the count. As the project proceeds, the **number of use cases** may get changed.
2. Using the requirement model, determine the **number of key classes**.
3. Categorize the type of interface and develop the **multiplier**. For example

Interface Type	Multiplier
No GUI	2.0
Text based interface	2.25
Graphical User Interface	2.5
Complex GUI	3.0

Multiply the key classes by these multipliers to obtain the number of **support classes.**

4. Now multiply the key and support classes by average number of work units per class. Approximately there are 15-20 person-days per class.

3.8 Project Scheduling

SPPU : Dec.-16, 19, Marks 6

- While scheduling the project, the manager has to estimate the time and resource of the project.
- All the activities in the project must be arranged in **coherent sequence**.

- The schedules must be continually updated because some uncertain problems may occur during the project life cycle.
- For new projects initial estimates can be made optimistically.

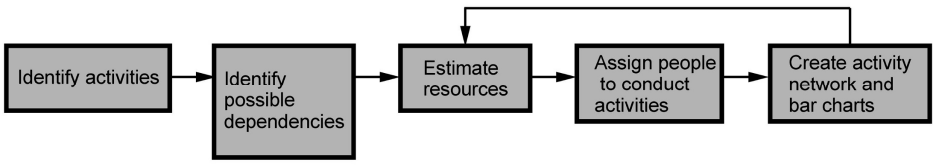


Fig. 3.8.1 Project scheduling process

- During the project scheduling the total work is separated into various **small activities**. And time required for each activity must be determined by the project manager.
- For efficient performance some activities are conducted **in parallel**.
- The project manager should be aware of the fact that : Every stage of the project may not be problem-free.
- Some of the typical problems in project development stage are :
  - People may leave or remain absent.
  - Hardware may get failed.
  - Software resource may not be available.
- To accomplish the project within given schedule the required resources must be available when needed.
- Various resources required for the project are -
  - Human effort
  - Sufficient disk space on server
  - Specialized hardware
  - Software technology
  - Travel allowance required by the project staff.
- Project schedules are represented as the set of chart in which the work-breakdown structure and dependencies within various activities is represented.

Review Question

1. What is project scheduling ? What are the basic principles of project scheduling ?

SPPU : Dec.-16, End Sem. Marks 6, Dec.-19, End Sem, Marks 5

3.9 Defining a Task Set for the Software Project

- **Definition of task set :** The task set is a collection of software engineering **work tasks**, **milestones**, and **work products** that must be accomplished to complete particular project.
- Every process model consists of various tasks sets. Using these tasks sets the software team define, develop or ultimately support computer software.
- There is no single task that is appropriate for all the projects but for developing large, complex projects the set of tasks are required. Hence every effective software process should define a collection of task sets depending upon the type of the project.
- Using tasks sets the **high quality software** can be developed and any unnecessary work can be avoided during software development.
- The number of tasks sets will vary depending upon the type of the project. Various **types of projects** are enlisted below -
  1. **Concept Development project :** These are the projects in which new business ideas or the applications based on new technologies are to be developed.
  2. **New application development project :** These projects are developed for satisfying a specific customer need.
  3. **Application upgradation project :** These are kind of projects in which existing software application needs a major change. This change can be for performance improvement, or modifications within the modules and interfaces.
  4. **Application maintenance project :** These are kind of projects that correct, adapt or extend the existing software applications.
  5. **Reengineering projects :** These are the projects in which the legacy systems are rebuilt partly or completely.
- Various **factors that influence** the tasks sets are -
  1. Size of project
  2. Project development staff
  3. Number of user of that project
  4. Application longetivity
  5. Complexity of application
  6. Performance constraints
  7. Use of technologies
- **Task set example :** Consider the concept development type of the project. Various tasks sets in this type of project are -
  - **Defining scope :** This task is for defining the scope, goal or objective of the project.
  - **Planning :** It includes the estimate for schedule, cost and people for completing the desired concept.

- **Evaluation of technology risks :** It evaluates the risk associated with the technology used in the project.
- **Concept implementation :** It includes the concept representation in the same manner as expected by the end user.

3.10 Task Network

SPPU : May-11, 18, 19, Dec.-13, 17, Marks 8

- The task is a small unit of work.
- The task network or an activity network is a graphical representation, with :
  - **(1) Nodes** corresponding to activities.
  - **(2) Tasks** or activities are linked if there is a dependency between them.
  - **(3) The task network** for the product development is as shown in Fig. 3.10.1.

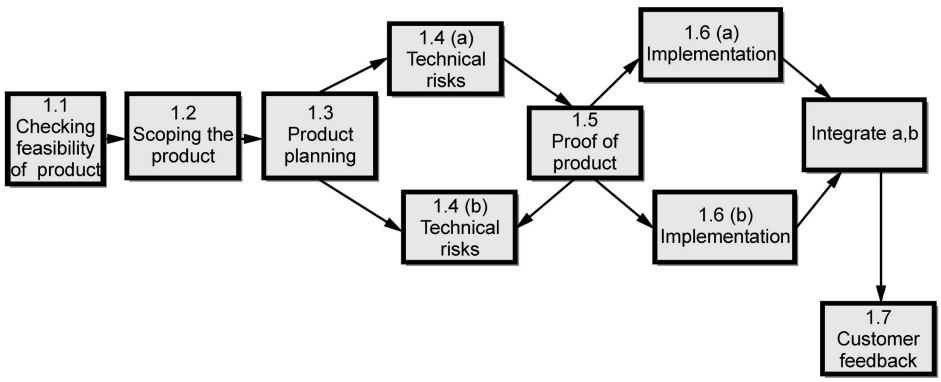


Fig. 3.10.1 Task network

- The task network definition helps project manager to understand the project work breakdown structure.
- The project manager should be aware of interdependencies among various tasks. It should be aware of all those tasks which lie on the critical path.

Review Questions

1. What is a task network in project scheduling ? Explain with an example.

   SPPU : May -11, 18, 19, End Sem, Marks 6, Dec.-17, End Sem, Marks 8

2. What is a task network ? How it is used in scheduling of software project ?

   SPPU : May -11, Dec.-13, Marks 8

3.11 Scheduling with Time Line Chart

SPPU : Dec.-12,16, May-18, Marks 8

- In software project scheduling the timeline chart is created. The **purpose** of timeline chart is to emphasize the **scope of individual task**. Hence set of tasks are given as input to the time line chart.
- The time line chart is also called as **Gantt chart**.
- The time line chart can be developed for entire project or it can be developed for individual functions.
- In time line chart
  1) All the tasks are listed at the leftmost column.
  2) The horizontal bars indicate the time required by the corresponding task.
  3) When multiple horizontal bars occur at the same time on the calendar, then that means concurrency can be applied for performing the tasks.
  4) The diamonds indicate the milestones.
- In most of the projects, after generation of time line chart the project tables are prepared. In project tables all the tasks are listed along with actual start and end dates and related information.

Example

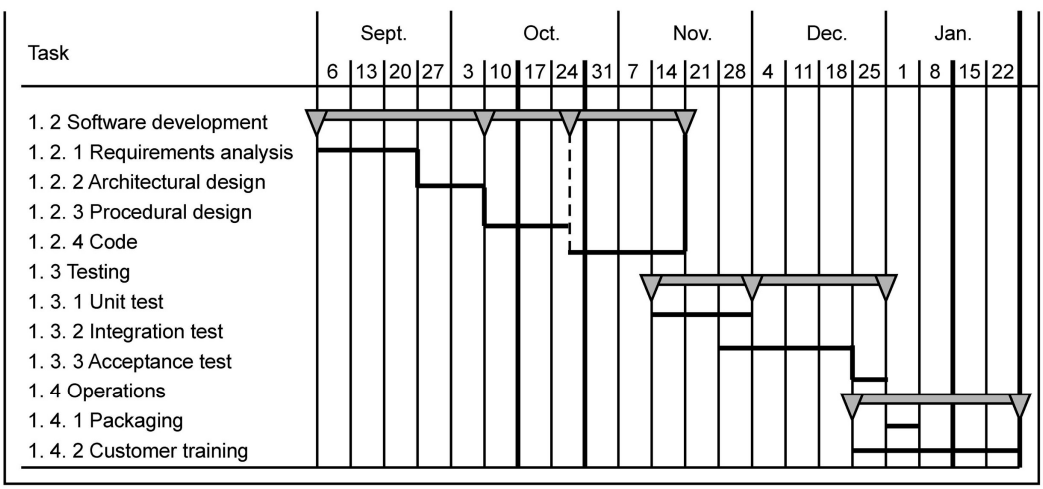


Fig. 3.11.1 Time line chart

Project table

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

Table 3.11.1 Project table

Review Questions

1. What is time line chart ? How it is used in scheduling of software project ?

SPPU : Dec.-12, Marks 8

2. What is timeline chart ? Explain with suitable examples.

SPPU : Dec.-16, End Sem, Marks 5, May -18, End Sem, Marks 6

3.12 Schedule Tracking Tools

- The schedule tracking tools help the project manager to define work tasks and their interdependencies.
- It is possible to assign human resources to these tasks and develop graphs, charts and tables that are required for tracking the schedule.
- The Microsoft project and Daily Activity Reporting and Tracking are commonly used scheduling tools. Let us discuss them briefly.

3.12.1 Microsoft Project

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

Features

Various features of MS planning tool are

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

3.12.2 Daily Activity Reporting & Tracking (DART)

- This tool enables you to track the changes to record being made by users.
- Many organizations deploy the Daily activity reporting and tracking tool that monitors the progress of the software project.
- DART collects the activity data and keep track of the work flow.
- DART can compute the set of business queries which are also called as indicators that can be used to keep track of the performance.
- The aim of DART is to empower software project stakeholders in their daily discussion regarding the performance of the project.

3.13 Multiple Choice Questions

Q.1 ____ is an activity in which project manager makes reasonable estimates of resources, cost and schedule

a Project scheduling b Risk analysis
c Project planning d None of these

Q.2 The objective of software project planning is _____.

a to convince customer that the project is feasible
b to make use of already conducted projects
c to determine probable profit margin before the actual project starts
d enable a manager to make estimation of cost and schedule

Q.3 Which of the following is activity is not the part of project planning ?

a Project estimation b Project monitoring
c Project scheduling d Risk management

Q.4 Which of the following is an important factor that affects the accuracy and efficacy of estimates ?

a Complexity of project
b Size of project
c Planning of project
d Degree of uncertainty

Q.5 Resources refers to _____.

a manpower
b machinery
c materials
d all of the above

Q.6 Following are the phases of project management life cycle. Arrange them in correct order

1. Design
2. Marketing
3. Analysis and evaluation
4. Inspection
testing and delivery

a 3-2-1-4
b 1-2-3-4
c 2-3-1-4
d 4-3-2-1

Q.7 Assembling project team and assigning their responsibilities are done during which phase of a project management ?

a Initiation
b Planning
c Execution
d Closure

Q.8 Component based software engineering emphasizes on ____ characteristic

a reliability
b availability
c security
d reusability

Q.9 Decomposition technique is used during _____.

a project planning
b project scheduling
c project estimation
d none of these

Q.10 Following is a basic parameter based on which all other estimations can be made during project planning.

a Efforts
b Duration
c Schedule
d Project size

Q.11 LOC stands for

a Log On Code
b Login On Code
c Least Object Code
d Lines Of Code

Q.12 Which of the following can be used for project size estimation ?

a Lines of code
b Function point
c Duration for the project
d Both a and b

Q.13 If an Indirect approach is taken, then the sizing approach is represented as

a LOC
b FP
c Fuzzy logic
d LOC and FP

Q.14 Which of the following is an activity that is conducted before requirements analysis and specification phase ?

a Project planning
b Project monitoring
c Project scheduling
d project control

Q.15 The Software Engineering Environment(SEE) consists of _____.

a hardware
b software
c people
d both a and b

Q.16 The size estimate for the software product build on LOC is considered as a direct measure

a True
b False

Q.17 Process based estimation requires problem decomposition based on _____.

a project schedule
b software functions
c process activities
d both b and c

Q.18 Which software project sizing approach develop estimates of the information domain characteristics ?

a Function point sizing
b Change sizing
c Standard component sizing
d Fuzzy logic sizing

Q.19 Empirical estimation models are based on _____.

a Expert judgement
b Common sense
c Trial and error determination of the parameters and coefficients
d None of these

Q.20 Which one is not a stage of COCOMO-II ?

a Early design estimation model
b Application composition estimation model
c Comprehensive cost estimation model
d Post architecture estimation model

Answer Keys for Multiple Choice Questions :

Q.1	c	**Q.2**	d	**Q.3**	b	**Q.4**	b
Q.5	d	**Q.6**	a	**Q.7**	a	**Q.8**	a
Q.9	c	**Q.10**	d	**Q.11**	d	**Q.12**	d
Q.13	b	**Q.14**	a	**Q.15**	d	**Q.16**	a
Q.17	d	**Q.18**	a	**Q.19**	b	**Q.20**	a

❑❑❑

UNIT - IV

4 Design Engineering

Syllabus

Design Concepts : *Design within the Context of Software Engineering, The Design Process, Software Quality Guidelines and Attributes, Design Concepts - Abstraction, Architecture, design Patterns, Separation of Concerns, Modularity, Information Hiding, Functional Independence, Refinement, Aspects, Refactoring, Object-Oriented Design Concept, Design Classes, The Design Model, Data Design Elements, Architectural Design Elements, Interface Design Elements, Component-Level Design Elements, Component Level Design for Web Apps, Content Design at the Component Level, Functional Design at the Component Level, Deployment-Level Design Elements.*

Architectural Design : *Software Architecture, What is Architecture, Why is Architecture Important, Architectural Styles, A brief Taxonomy of Architectural Styles.*

Contents

Part I : Introduction to Design Concepts

4.1 Concept of Design

SPPU : Dec.-12,16, May-14, Marks 8

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

- Software design is an **iterative process** using which the user requirements can be translated into the software product.
- At the initial stage of the software is represented as an **abstract view**. During the sub-sequent iterations data, functional and behavioral requirements are traced in detail. That means at each iteration the refinement is made to obtain lower level details of the software product.

Characteristics of Good Design

1. The good design should **implement all the requirements** specified by the customer. Even if there are some implicit requirements of the software product then those requirements should also be implemented by the software design.
2. The design should be **simple** enough so that the software can be understood easily by the developers, testers and customers.
3. The design should be **comprehensive**. That means it should provide complete picture of the software.

4.1.1 Design within the Context of Software Engineering

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

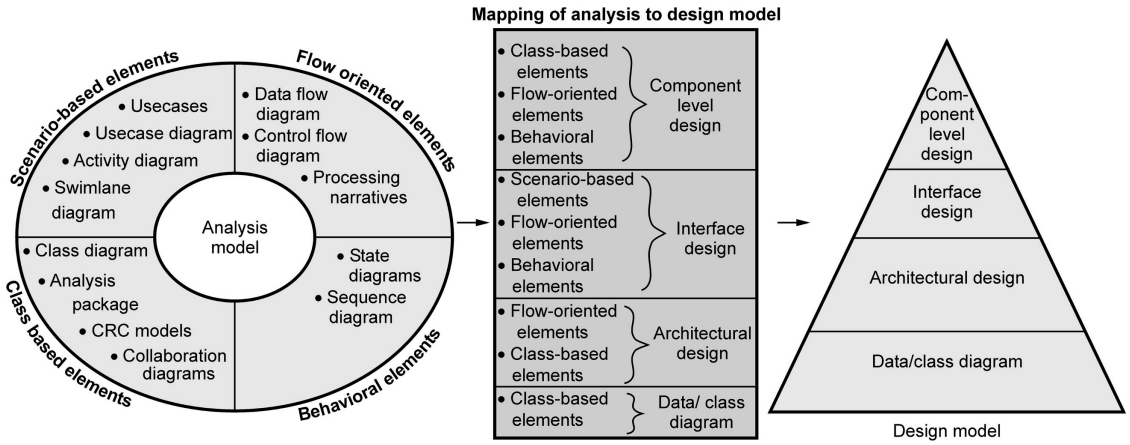


Fig. 4.1.1 Translating analysis model into the design model

- The **interface design** describes how software communicates with systems. These systems are interacting with each other as well as with the humans who operate them. Thus interface design represents the flow of information and specific type of behaviour. The usage scenarios and behavioural models of analysis modelling provide the information needed by the interface design.
- The **component-level** design transforms structural elements of software architecture into procedural description of software module. The information used by the component design is obtained from class based model, flow based model and behavioural model.
- Software design is **important** to assess the **quality** of software. Because design is the only way that we can accurately translate the user requirements into the finished software product.
- Without design unstable system may get developed. Even if some small changes are made then those changes will go fail .It will become difficult to test the product. The quality of the software product can not be assessed until late in the software process.

4.2 The Design Process

- Software design is an **iterative process** in which the requirements are translated into the blueprint of the software. Initially the software is represented at high level of abstraction, but during the subsequent iterations data, functional and behavioural requirements are traced in more detail.
- Thus refinement made during each iteration leads to design representations at much lower level of abstraction. Throughout the software design process the quality of the software is assessed by considering certain characteristics of the software design.

4.3 Software Quality Guidelines and Attributes

SPPU : Dec.-12, 16, 19, May-14, Marks 8

- The **goal** of any software design is to produce **high quality** software.
- In order to evaluate quality of software there should be some predefined rules or criteria that need to be used to assess the software product. Such criteria serve as characteristics for good design.
- The **quality guidelines** are as follows -
  1. The design architecture should be created using following issues -
     - The design should be created using **architectural styles and patterns.**
     - Each component of design should posses **good design characteristics**.
     - The implementation of design should be **evolutionary**, so that testing can be performed at each phase of implementation.
  2. In the design the data, architecture, interfaces and components should be **clearly represented.**
  3. The design should be **modular**. That means the subsystems in the design should be logically partitioned.
  4. The **data structure** should be appropriately chosen for the design of specific problem.
  5. The **components** should be used in the design so that functional independency can be achieved in the design.
  6. Using the information obtained in software **requirement analysis** the design should be created.
  7. The **interfaces** in the design should be such that the complexity between the connected components of the system gets reduced. Similarly interface of the system with external interface should be simplified one.
  8. Every design of the software system should **convey its meaning** appropriately and effectively.

Quality Attributes

- The design quality attributes popularly known as **FURPS** (**F**unctionality, **U**sability, **R**eliability, **P**erformance and **S**upportability) is a set of criteria developed by Hewlett and Packard.
- Following table represents meaning of each quality attribute

Quality Attribute	Meaning
Functionality	Functionality can be checked by assessing the set of **features and capabilities** of the functions. The functions should be **general** and should not work only for particular set of inputs. Similarly the **security** aspect should be considered while designing the function.
Usability	The usability can be assessed by knowing the **usefulness** of the system.
Reliability	Reliability is a measure of frequency and severity of **failure**. Repeatability refers to the consistency and repeatability of the measures. The Mean Time to Failure (**MTTF**) is a metric that is widely used to measure the product's performance and reliability.
Performance	It is a measure that represents the response of the system. Measuring the performance means measuring the processing speed, memory usage, response time and efficiency.
Supportability	It is also called maintainability. It is the ability to adopt the enhancement or changes made in the software. It also means the ability to withstand in a given environment.

Review Questions

1. Explain the quality attributes, considered in software design. **SPPU : Dec.-12, Marks 8**
2. What are the design quality guidelines ? **SPPU : May-14, Marks 8**
3. Explain the quality attributes, considered in software design . **SPPU : Dec.-16, End Sem, Marks 5**
4. What are the software design quality attribute and guidelines? **SPPU : Dec.-19, End Sem, Marks 7**

4.4 Design Concepts

SPPU : April-15, May-11, 13, 15, Dec.-12, 16, Marks 8

The software design concept provides a framework for implementing the right software.

Following are certain issues that are considered while designing the software -

- Abstraction
- Modularity
- Architecture
- Refinement
- Pattern
- Information hiding
- Functional independence
- Refactoring
- Design classes

4.4.1 Abstraction

- The abstraction means an ability to cope up with the complexity.
- Software design occurs at different levels of abstraction. At each stage of software design process levels of abstractions should be applied to refine the software solution.
- At the **higher level** of abstraction, the solution should be stated in **broad terms** and in the **lower level** more **detailed description** of the solution is given.
- While moving through different levels of abstraction the **procedural abstraction** and **data abstraction** are created.
- The **procedural abstraction** gives the named sequence of instructions in the specific function. That means the functionality of procedure is hidden. For example : **Search the record** is a procedural abstraction in which implementation details are hidden (i.e. Enter the name, compare each name of the record against the entered one, if a match is found then declare success !! Otherwise declare 'name not found').
- In **data abstraction** the collection of data objects is represented. For example for the procedure **search** the data abstraction will be **record**. The **record** consists of various attributes such as record ID, name, address and designation.

4.4.2 Modularity

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

  "Suppose there are two problems A and B with varying complexity. If the complexity of problem A is greater than the complexity of the problem B then obviously the efforts required for solving the problem A is greater than that of problem B. That also means the time required by the problem A to get solved is more than that of problem B."
- The overall complexity of two problems when they are combined is greater than the sum of complexity of the problems when considered individually. This leads to **divide and conquer strategy** (according to divide and conquer strategy the problem is divided into smaller subproblems and then the solution to these subproblems is obtained).
- Thus dividing the software problem into manageable number of pieces leads to the concept of modularity.

- It is possible to conclude that if we subdivide the software indefinitely then effort required to develop each software component will become very small. But this conclusion is invalid because the total number of modules get increased the efforts required for developing each module also gets increased.
- That means the cost associated with each effort gets increased.
- The effort (cost) required for integration these modules will also get increased.
- The total cost required to develop such software product is shown by Fig. 4.4.1.
- The Fig. 4.4.1 provides useful guideline for the modularity and that is - **overmodularity or the undermodularity while developing the software product must be avoided.** We should modularize the software but the modularity must remain near the region denoted by **M.**

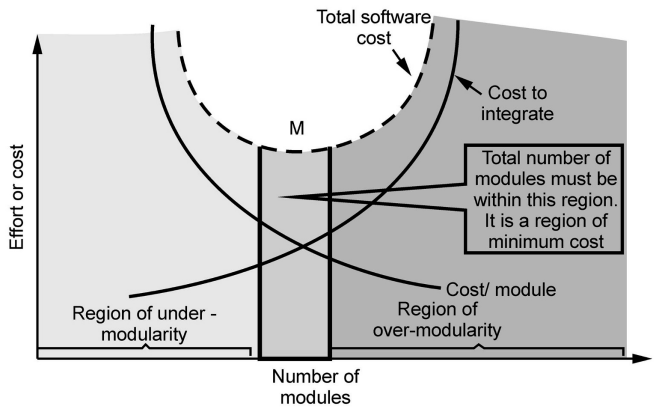


Fig. 4.4.1 Modularity and software cost

- Modularization should be such that the development can be planned easily, software increments can be defined and delivered, changes can be more easily accommodated and long term maintenance can be carried out effectively.
- **Meyer** defines five criteria that enable us to evaluate a design method with respect to its ability to define an effective modular system :
  1. **Modular decomposability :** A design method provides a systematic mechanism for decomposing the problem into sub-problems. This reduces the complexity of the problem and the modularity can be achieved.
  2. **Modular composability :** A design method enables existing design components to be assembled into a new system.
  3. **Modular understandability :** A module can be understood as a standalone unit. It will be easier to build and easier to change.

4. **Modular continuity :** Small changes to the system requirements result in changes to individual modules, rather than system-wide changes.
5. **Modular protection :** An aberrant condition occurs within a module and its effects are constrained within the module.

4.4.3 Architecture

Architecture means representation of **overall structure** of an integrated system. In architecture various components interact and the data of the structure is used by various components. These components are called system elements. Architecture provides the basic framework for the software system so that important framework activities can be conducted in systematic manner.

In architectural design **various system models** can be used and these are

Model	Functioning
Structural model	Overall architecture of the system can be represented using this model.
Framework model	This model shows the architectural framework and corresponding applicability.
Dynamic model	This model shows the reflection of changes on the system due to external events.
Process model	The sequence of processes and their functioning is represented in this model.
Functional model	The functional hierarchy occurring in the system is represented by this model.

4.4.4 Refinement

- Refinement is actually a process of elaboration.
- Stepwise refinement is a top-down design strategy proposed by Niklaus WIRTH.
- The architecture of a program is developed by successively refining levels of procedural detail.
- The process of program refinement is analogous to the process of refinement and partitioning that is used during requirements analysis.
- Abstraction and refinement are complementary concepts. The major difference is that - In the abstraction low-level details are suppressed. Refinement helps the designer to elaborate low-level details.

4.4.5 Pattern

According to **Brad Appleton** the design pattern can be defined as - It is a named nugget (something valuable) of insight which conveys the essence of proven solution to a recurring problem within a certain context.

In other words, design pattern acts as a design solution for a particular problem occurring in specific domain. Using design pattern designer can determine whether-

- Pattern can be **reusable.**
- Pattern can be used for **current work**.
- Pattern can be used to **solve similar kind of problem** with different functionality.

4.4.6 Information Hiding

- Information hiding is one of the important property of effective modular design.
- The term information hiding means the modules are designed in such a way that information contained in one module cannot be accessible to the other module (the module which does not require this information).
- Due to information hiding only limited amount of information can be passed to other module or to any local data structure used by other module.
- The **advantage** of information hiding is basically in testing and maintenance.
- Due to information hiding some data and procedures of one module can be hidden from another module. This ultimately **avoids** introduction of **errors** module from one module to another. Similarly one can make **changes** in the desired module without affecting the other module.

4.4.7 Functional Independence

- The functional independence can be achieved by developing the functional **modules** with single-minded approach.
- By using functional independence functions may be compartmentalized and **interfaces** are **simplified**.
- Independent modules are easier to maintain with **reduced error propagation**.
- Functional independence is a key to good design and design is the key to software quality.
- The **major benefit** of functional independence is in achieving effective modularity.
- The functional independence is assessed using two qualitative criteria - **Cohesion** and **coupling.**

4.4.7.1 Cohesion

- With the help of cohesion the information hiding can be done.
- A cohesive module performs only "one task" in software procedure with little interaction with other modules. In other words cohesive module performs only one thing.
- Different types of cohesion are :
  1. **Coincidentally cohesive :** The modules in which the set of tasks are related with each other loosely then such modules are called coincidentally cohesive.
  2. **Logically cohesive :** A module that performs the tasks that are logically related with each other is called logically cohesive.
  3. **Temporal cohesion :** The module in which the tasks need to be executed in some specific time span is called temporal cohesive.
  4. **Procedural cohesion :** When processing elements of a module are related with one another and must be executed in some specific order then such module is called procedural cohesive.
  5. **Communicational cohesion :** When the processing elements of a module share the data then such module is communicational cohesive.
- The goal is to achieve high cohesion for modules in the system.

4.4.7.2 Coupling

- Coupling effectively represents how the modules can be "connected" with other module or with the outside world.
- Coupling is a **measure of interconnection** among modules in a program structure.
- Coupling depends on the interface complexity between modules.
- The goal is to strive for lowest possible coupling among modules in software design.
- The property of good coupling is that it should reduce or avoid change impact and ripple effects. It should also reduce the cost in program changes, testing and maintenance.
- Various types of coupling are :
  - **i) Data coupling :** The data coupling is possible by parameter passing or data interaction.
  - **ii) Control coupling :** The modules share related control data in control coupling.
  - **iii) Common coupling :** In common coupling common data or a global data is shared among the modules.
  - **iv) Content coupling :** Content coupling occurs when one module makes use of data or control information maintained in another module.

Sr. No.	Content coupling	Common coupling
1.	Content coupling occurs between two modules when one module refers to the **internals** of the other module.	Common coupling occurs when modules communicate using **global data areas**.
2.	Content coupling is also known as **pathological coupling**.	Common coupling is also known as **global coupling**.
3.	This type of coupling does not require any global data or global variable for communication.	This type of coupling is based on the communication using **global data areas**.
4.	The **major drawback** of content coupling is that - if we want to reuse one component we need to import all the components that are coupled with the component being reused.	The **major drawback** of common coupling is that - modules in the common coupling are tightly coupled. A fault in one module using global data may show up in another module because global data may be updated by any module at any time. This greatly affects the reusability of modules.
5.	The access to internal structure is of main concern in this kind of communication.	The access to global data is of main concern in this kind of communication.
6.	In this coupling, component directly modifies another's data.	In common coupling, it is difficult to determine all the components that affect a data element.
7.	**Example -** Part of Program that searches for the entry of particular employee. when the module does not find the entry for the employee, it directly adds employee modifying the contents of data structure containing employee data.	**Example -** A process control component that maintains the status of operations in a global data store. This control block gets data from multiple sources or supplies data to multiple sinks. Each source process writes the status of its operation directly to the global data store.

Sr. No.	Coupling	Cohesion
1.	Coupling represents how the **modules are connected** with other modules or with the outside world.	In cohesion, the cohesive module performs **only one thing.**
2.	With coupling **interface complexity** is decided.	With cohesion, **data hiding** can be done.
3.	The goal of coupling is to achieve **lowest coupling.**	The goal of cohesion is to achieve **high cohesion.**
4.	Various types of couplings are - Data coupling, Control coupling, Common coupling and Content coupling.	Various types of cohesion are - Coincidental cohesion, Logical cohesion, Temporal cohesion, Procedural cohesion and Communicational cohesion.

4.4.8 Refactoring

Refactoring is necessary for simplifying the design without changing the function or behaviour. **Fowler** has defined refactoring as "The process of changing a software system in such a way that the external behavior of the design do not get changed, however the internal structure gets improved".

Benefits of refactoring are -

- The **redundancy** can be achieved.
- **Inefficient algorithms** can be eliminated or can be replaced by efficient one.
- Poorly constructed or **inaccurate data structures** can be removed or replaced.
- Other **design failures** can be rectified.

The decision of refactoring particular component is taken by the designer of the software system.

Exercise 4.4.1 *Justify the term : "Design is not coding and coding is not design".*

Solution : This is one of the **design principles** that are to be applied while designing the software systems. This means that level of abstraction of design model should be higher than the source code. At the time of coding, whatever design decisions are made can be implemented. Hence design should be such that it can be transformed into source code. Similarly, during coding it is not desired to do the design to fulfill the requirements.

Exercise 4.4.2 *Discuss the statement, "Abstraction and refinement are complementary concepts.*

Solution : Abstraction and refinement are complementary concepts. The major difference is that - in the abstraction low-level details are suppressed.

Refinement helps the designer to elaborate low-level details.

Exercise 4.4.3 *"Modularity is the single attribute of the software that allows a program to be intellectually manageable" - How this is true ?*

Solution : This is quoted by Glenford Mayers. Consider that there is a large program composed of single module. Such a program cannot be grasped by reader. The control paths, span of reference, number of variables and overall complexity of such a program is beyond understanding. On the other hand if the program is divided into certain number of modules then overall complexity of the program gets reduced. The error detection and handling can be done effectively. Also changes made during testing and maintenance become manageable. Hence it is true that "Modularity is the single attribute of the software that allows a program to be intellectually manageable".

Exercise 4.4.4 *Why modularity is important in software projects ?*

Solution : Modularity is important in software projects because of following reasons (1) Modularity reduces complexity and helps in easier implementation. (2) The parallel development of different parts of the system is possible due to modular design. (3) Changes made during testing and maintenance become manageable and they do not affect the other modules.

Exercise 4.4.5 *Differentiate between : abstraction and modularization.*

Solution : Abstraction is a software design concept which is applied to refine software solution. In the process of abstraction only necessary information is abstracted form requirement analysis to create the software solution in broad terms. While moving through different levels of abstraction the procedural and data abstraction are prepared. During abstraction the procedural and data objects are prepared.

Modularity is a design concept in which requirements are divided into separately named and addresable components called as modules. Modularity in design reduces complexity and helps in easier implementation.

Exercise 4.4.6 *Discuss the relationship between the concept of information hiding as an attribute of effective modularity and the concept of module independence ?*

Solution :

- Information hiding can be related to both coupling and cohesion concepts.
- For reducing the coupling only those modules are linked together that are absolutely needed. This reduces the coupling between modules.
- Similarly information is isolated so that the functionalities can be isolated. This improves the cohesion of individual modules.
- **Information hiding implies that effective modularity** can be achieved by defining a set of independent modules that communicate with one another only that information necessary to achieve software function.
- Using abstractions the procedural entities are defined. This is helpful for testing and maintenance.
- The concept of module independence helps in achieving modularity, abstraction and information hiding. Independent modules are easier to maintain and test. Reusability of such modules is also possible.

Example 4.4.7 *Justify " A cohesive design should have high cohesiveness and low coupling".*

SPPU : April-15, In Sem, Marks 4

Solution :

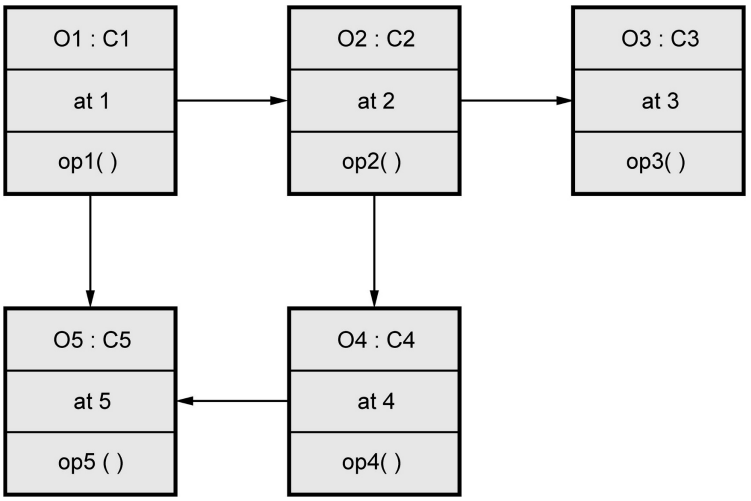
- Low coupling is an approach to interconnecting the components in a system or network so that those components, are less dependant upon each other.
- High cohesiveness indicate that module perform only one task.
- Ideally the components in the system must be least dependent on each other and every component must perform only one task at particular instance.
- Hence it is said that the system must have high cohesiveness and low coupling.

Review Questions

1. Explain the following design concepts. i) Modularity ii) Architecture
   SPPU : May-11, Marks 6
2. What are the characteristics of a well formed design class ? **SPPU : Dec.-12, Marks 6**
3. Differentiate between abstraction and refinement. **SPPU : Dec.-12, Marks 4**
4. Explain the following design concepts. i) Refinement ii) Modularity iii) Architecture
   SPPU : May-13, Marks 8
5. Explain about various design concepts considered during design.
   SPPU : April-15, In Sem, Marks 6
6. What do you understand by refactoring ? Give the importance of refactoring in improving quality of software. **SPPU : May-15, End Sem, Marks 6**
7. What do you mean by the term cohesion and coupling in context of software design ? How are these concepts useful in arriving at a good design of a system ?
   SPPU : Dec.-16, End Sem, Marks 5
8. Define following design concepts -
   i) Pattern ii) Information Hiding
   iii) Architecture iv) Refinement **SPPU : Aug.-17, In Sem, Marks 4**
9. Explain different Design concepts. **SPPU : Aug.-17, In Sem, Marks 3**
10. What do you mean by the term cohesion and coupling in the context of software design ? How are these concepts useful in arriving at a good design of a system ?
    SPPU : Aug.-17, In Sem, Marks 3
11. What is the relationship between modularity and functional dependence ?
    SPPU : Dec.-18, End Sem, Marks 5
12. A Design should have high cohesive and low coupling. Justify.
    SPPU : Dec.-18, Oct.-19, In Sem, Marks 5
13. What do you understand by refactoring ? Give importance of refactoring in improving the quality of software. **SPPU : Dec.-18, End Sem, Oct.-19, In Sem, Marks 5**

4.5 Object Oriented Design Concepts

Object is an important element of an Object Oriented Design (OOD). Various objects and their interaction with each other objects is graphically represented in object oriented design. An object is an entity which can be obtained from its belonging class and it posses attributes and operations. The typical presentation of object oriented system is depicted by following Fig. 4.5.1.



In object oriented system three strategies are mainly applied -

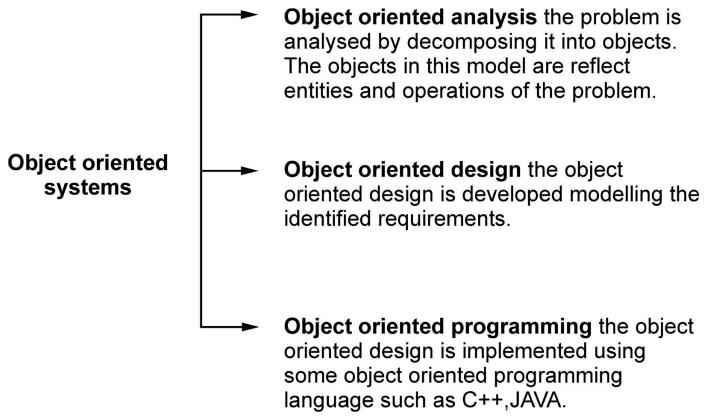


Fig. 4.5.1

Advantages of object oriented system

1. Reusability of objects reduces programming efforts.
2. The system design is more manageable if it is done using object oriented approach.
3. Error detection and handling is easy.
4. Changes can be incorporated very easily in the software system.

Disadvantages of object oriented system

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

The Unified Modelling Language (UML) was developed by Grady Booch, Jim Rumbaugh and Ivar Jacobson for modelling the object oriented systems.

Under this package various diagrams can be created to represent the system from different perspectives.

1) **Activity -** The actions taken by an operation.
2) **Class diagram -** The static relationships, methods, and fields of an object.
3) **Collaboration -** The relationship between components.
4) **Sequence -** Flow of events over time (oddly enough contains the same info as Collaboration).
5) **Component -** The physical components.
6) **Deployment -** How to push the software to hardware.
7) **Object -** Objects and their relationships.
8) **State chart -** Operation of system shown through states.
9) **Use Case -** The functions of a system from the user's perspective.

4.6 Design Classes

SPPU : Dec.-13, Marks 6

Design classes are defined as the classes that describe some elements of problem domain, focus on various aspects of problem from user's point of view.

The goals of design classes is :

1. To refine the analysis classes by providing the detail design, so that further implementation can be done easily.
2. To create new set of classes for implementing the infrastructure of the software.

There are five different types of design classes

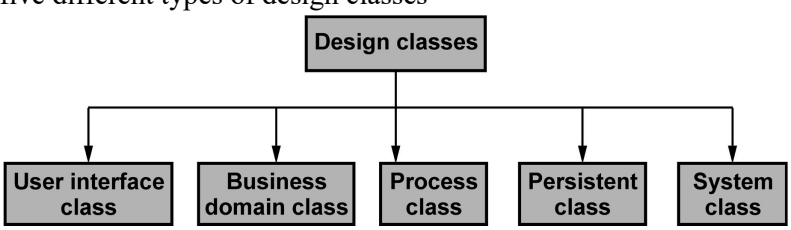


Fig. 4.6.1 Types of design classes

1. **User Interface Class :** The user interface class defines all the abstractions that are necessary for Human Computer Interface(HCI). The user interface classes is basically a **visual representation** of the HCI.
2. **Business Domain Class :** Business domain classes are the refinement of analysis classes. These classes identify the **attributes and services** that are needed to implement some elements of **business domain.**

3. **Process Class :** Process class implement lower level business abstractions used by the business domain.
4. **Persistent Class :** These classes represent the data store such as databases which will be retained as it is even after the execution of the software.
5. **System Class :** These classes are responsible for software management and control functions that are used for system operation.

Each class must be **well formed design class**. Following are some characteristics of well formed design classes -

- **Complete and Efficient**

A design class must be properly **encapsulated** with corresponding attributes and methods. Design class must contain all those methods that are sufficient to achieve the intent of the class.

- **Primitiveness**

Methods associated with one particular class must perform unique service and the class should not provide another way to implement the same service.

- **High Cohesion**

A cohesive designed class must posses small, focused set of responsibilities and these responsibilities must be associated with all the attributes of that class.

- **Low Coupling**

Design classes must be collaborated with manageable number of classes. If one design class is collaborated with all other design classes then the implementation, testing and maintenance of the system becomes complicated. The **law of Demeter** suggests that the one particular design class should send messages to the methods of neighbouring design classes only.

Review Question

1. Explain the types of design classes. **SPPU : Dec.-13, Marks 6**

4.7 The Design Model and Elements

SPPU : Dec.-11, May-13, 14, Marks 8

- The **process dimension** denotes that the design model evolutes due to various software tasks that get executed as the part of software process.
- The **abstract dimension** represents level of details as each element of analysis model is transformed into design equivalent.
- In following Fig. 4.7.1 the **dashed line** shows the boundary between analysis and design model.

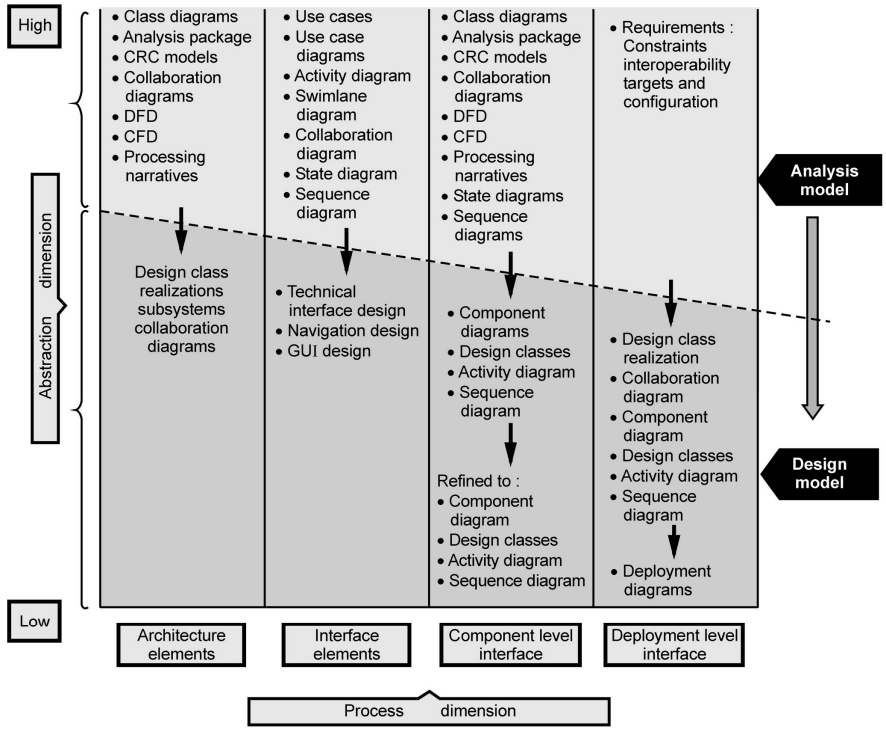


Fig. 4.7.1 Dimension of design model

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

4.7.1 Data Design Elements

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

4.7.2 Architectural Design Elements

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

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

4.7.3 Interface Design Elements

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

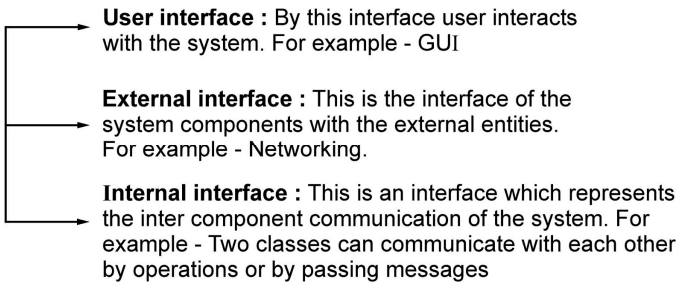


Fig. 4.7.2

4.7.4 Component Level Design Elements

- The component level design is more detailed design of the software system along with the specifications. The component level design elements describe the internal details of the component. In component level design all the local data objects, required data structures and algorithmic details and procedural details are exposed.
- Fig. 4.7.3 represents that component **order** makes use of another component **form**.
- The **order** is dependent upon the component form. These two objects can be interfaced with each other.

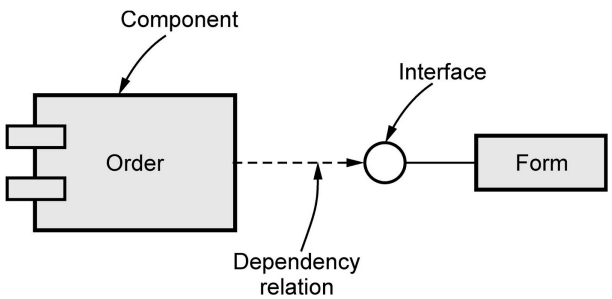


Fig. 4.7.3 Components

4.7.5 Deployment Level Design Elements

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

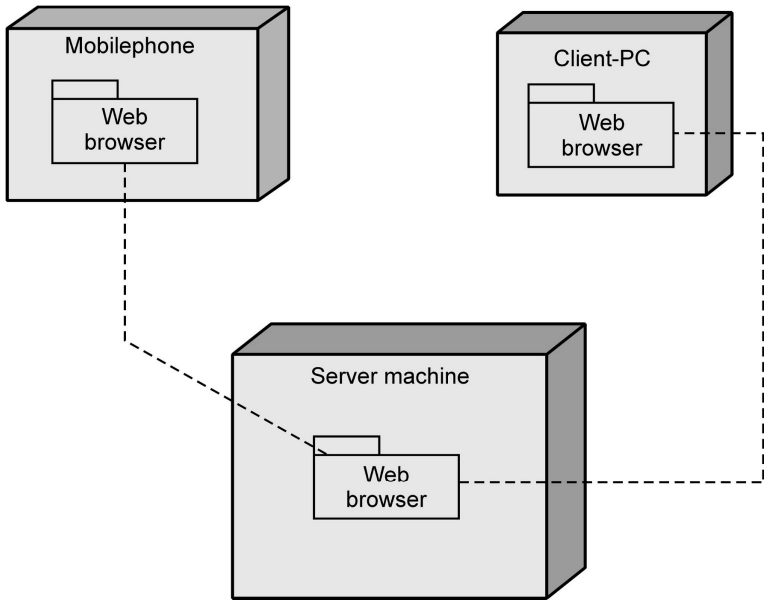


Fig. 4.7.4 Deployment diagram

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

Review Questions

1. Describe the design model with a neat diagram and give the traceability of each layer of design model to analysis model. **SPPU : Dec.-11, Marks 8**
2. What are the deployment level design elements ? **SPPU : Dec.-11, Marks 4**
3. Explain the elements of design model. **SPPU : May-13, Marks 8**
4. What are the elements in data design ? What are the guidelines for the data design ? **SPPU : May-14, Marks 8**

4.8 Component Level Design

What is Component ?

- Component is nothing but a model for computer software.
- The OMG Unified Modeling Language specification defines the concept of component more formally and it is -
- "Component is a modular, deployable and replaceable part of system that encapsulates the implementation and exposes the set of interfaces".

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

4.9 Class Based Design

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

4.9.1 Basic Design Principle

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

1. **The Open-closed Principle**

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

2. **The Liskov Substitution Principle**

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

3. **Dependency Inversion Principle**

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

4. **The interface Segregation Principle**

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

4.9.2 Component Level Design Guideline

- **Ambler** suggested following guideline for conducting component level design -
- **Components :** Components are the part of architectural model. The naming conventions should be established for components. The component names should be specified from the problem domain. These names should be meaningful.
- **Interfaces :** Interfaces serve important role in communication and collaboration. The interface should be drawn as a circle with a solid line connecting it to another element. This indicates that the element to which it is connected offers the interface. The interface should flow from left side of component. Only important interfaces must be drawn.
- **Dependencies and interfaces :** The dependencies must be shown from left to right. The inheritance should be shown from bottom to top i.e. from derived to base class. The component interdependencies should be represented via interfaces.

4.9.3 Cohesion

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

- **Functional :** In this type of cohesion the each module performs only one component.
- **Layer :** This type of cohesion is used in packages, components and classes. In this type of cohesion the higher level layer access the functionalities of the lower levels but the lower level layers do not access the functionalities of the higher level layers.
- **Communicational :** When all the operations of the class access the same set of data then this type of cohesion occurs.
- **Sequential :** In this type of cohesion, the components are grouped together in such a manner that the output of one operation is provided as input to another operation. So that all the operations execute in some specific sequence.
- **Procedural :** In this cohesion the procedures are invoked immediately one after the another.
- **Temporal :** The cohesion in which the operations specify specific behaviour is called temporal cohesion.
- **Utility :** The components, classes or operations are grouped together if they perform some specific operation otherwise they are not related with each other.

4.9.4 Coupling

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

Various types of coupling are :

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

Review Question

1. Explain class based modeling with suitable example

SPPU : April-16, In Sem, Marks 4, May-16, End Sem, Marks 7

4.10 Conducting Component Level Design

Following is a set of steps that are applied for component level design -

1. Identify all the design classes from the problem domain.
2. Identify all the design classes from the infrastructure domain.
3. Detail out all the design classes which are not the reusable components.
   a. When components communicate or collaborate then represent their communication using messages.
   b. Identify interfaces for each component.
   c. Define the attributes of the classes by specifying the data types and deciding the data structures.
   d. Specify all the operations by describing the processing flow within each operation.
4. Specify the databases and files and identify the classes that are involved in handling them
5. Describe the behavioural representation for the class.
6. Design the deployment diagram fro providing additional implementation.
7. Factor the component level design.
8. Find out alternative design solutions as well.

Review Question

1. Explain the guidelines of component level design.

SPPU : Dec.-17, End Sem, Dec.-18, End Sem, Marks 5

4.11 Component Level Design for Web Apps

Following are some tasks of modern web application -

1. Carry out various processing activities for generating the web contents.
2. Improve navigation capability in dynamic order.
3. Establish data interfaces with external corporate system.
4. Provide computation or data processing capability.
5. Provide the mechanism for database query processing and handling.

These tasks can be accomplished by designing and constructing the program components that are identical in the form to software component for conventional software.

4.11.1 Content Design at the Component Level

- During the content design at the component level, the focus is on two things - i) **Content objects** and ii) The way the content objects are packaged for presentation to the end user.
- At the component level, the content design need to focus on the characteristics of the WebApps.
- It is necessary to organize the contents in such a way that the design manipulation and reference could be easier.
- If the contents are highly dynamic, it becomes important to establish a clear structural model that incorporates content components.

4.11.2 Functional Design at the Component Level

- Modern web applications provide various **useful functionalities** such as :
  i) Processing of contents and navigation in **dynamic fashion**.
  ii) Performing **computations or data processing operations** dynamically.
  iii) **Efficient query processing** and access to database.
  iv) Establishing **data interfaces** with external corporate systems.
- To achieve these functionalities, it is necessary to design and construct the **functional components** of web applications.
- The Web contents and above mentioned functionalities are combined to create functional architecture.
- A **functional architecture** is a representation of the functional domain of the WebApplications and describes the key functional components in the WebApplications and how these components interact with each other.

Part II : Architectural Design

4.12 Software Architecture

4.12.1 What is Architecture ?

SPPU : May-12, Marks 4

- The architectural design is the design process for identifying the subsystems making up the system and framework for subsystem control and communication.
- The **goal** of architectural design is to establish the overall structure of software system. Architectural design represents the link between design specification and actual design process.

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

4.12.2 Why is Architecture Important ?

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

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

4.12.3 Structural Partitioning

The program structure can be partitioned horizontally or vertically.

Horizontal partitioning

- Horizontal partitioning defines separate branches of the modular hierarchy for each major program function.
- Horizontal partitioning can be done by partitioning system into : Input, data transformation (processing) and output.
- In horizontal partitioning the design making modules are at the top of the architecture.

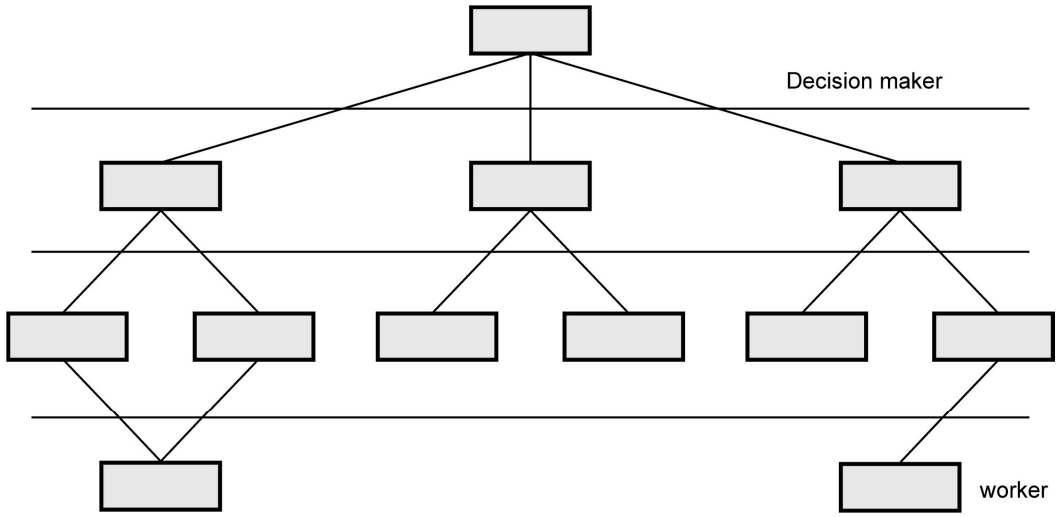


Fig. 4.12.1 Horizontal partitioning

Advantages of horizontal partition

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

Disadvantage of horizontal partition

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

Vertical partitioning

Vertical partitioning suggests the control and work should be distributed top-down in program structure. (Refer Fig. 4.12.2).

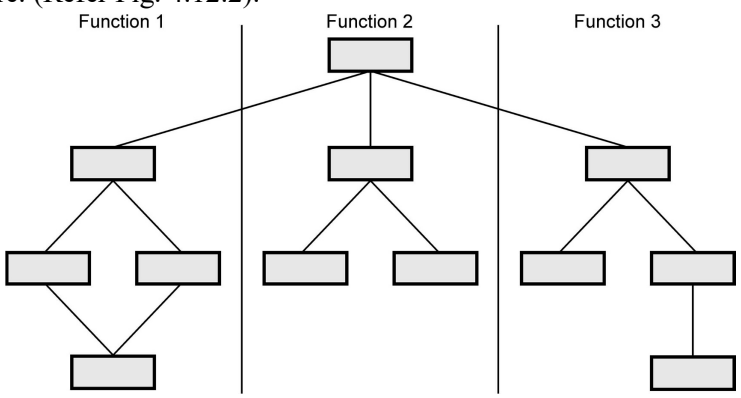


Fig. 4.12.2 Vertical partitioning

In vertical partitioning

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

Advantages of vertical partition

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

4.12.4 Difference between Horizontal and Vertical Partition

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

Review Questions

1. Explain software architecture as a concept of software design. **SPPU : May-12, Marks 4**
2. What do you mean by software architecture? Explain with suitable example. **SPPU : April-16, In Sem, Marks 4**

4.13 Architectural Styles

SPPU : Dec.-11,12, May-12, Marks 8

4.13.1 Architectural Styles

- The **architectural model or style** is a pattern for creating the system architecture for given problem. However, most of the large systems are heterogeneous and do not follow single architectural style.
- System categories define the architectural style
  1. **Components :** They perform a function.
     For example : Database, simple computational modules, clients, servers and filters.
  2. **Connectors :** Enable communications. They define how the components communicate, co-ordinate and co-operate.
     For example : Call, event broadcasting, pipes
  3. **Constraints :** Define how the system can be integrated.
  4. **Semantic models :** Specify how to determine a system's overall properties from the properties of its parts.
- The commonly used architectural styles are
  1. Data centered architectures.
  2. Data flow architectures.
  3. Call and return architectures.
  4. Object oriented architectures.
  5. Layered architectures.

4.13.1.1 Data Centered Architectures

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

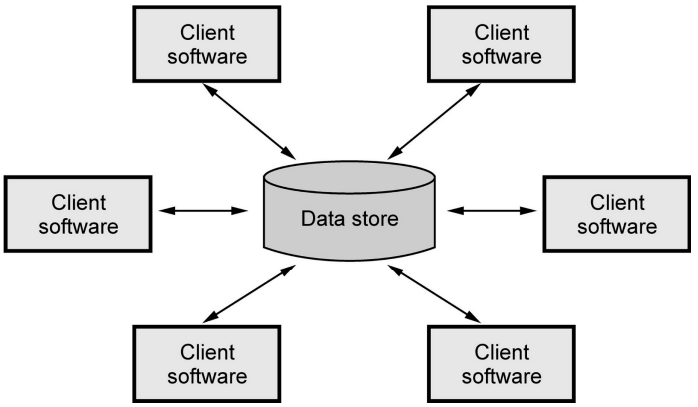


Fig. 4.13.1 Data centered architecture

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

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

In data centered architecture,

Components are : Database elements such as tables, queries.

Communication are : By relationships

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

4.13.1.2 Data Flow Architectures

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

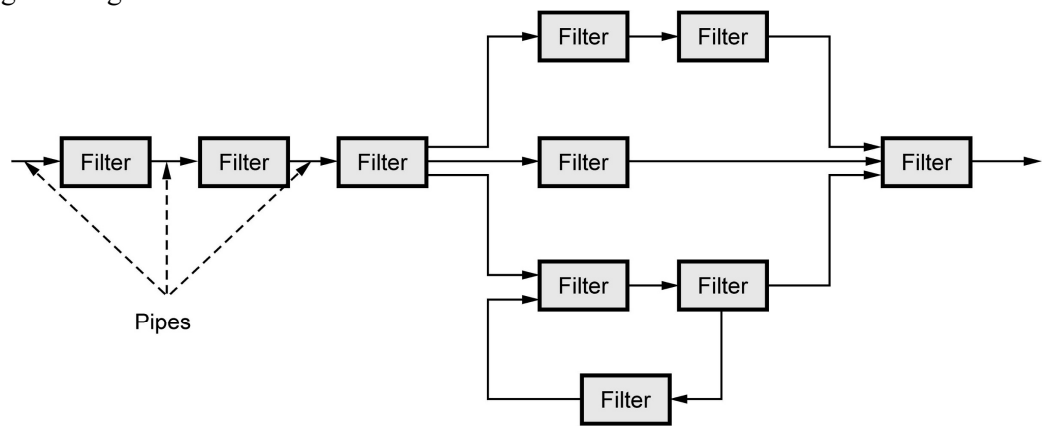


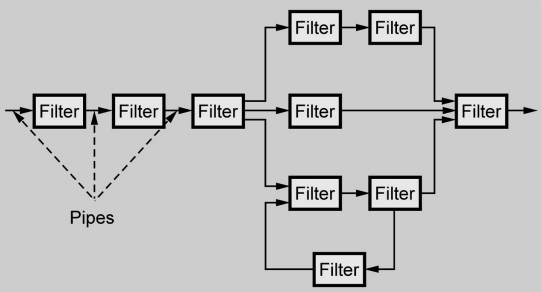
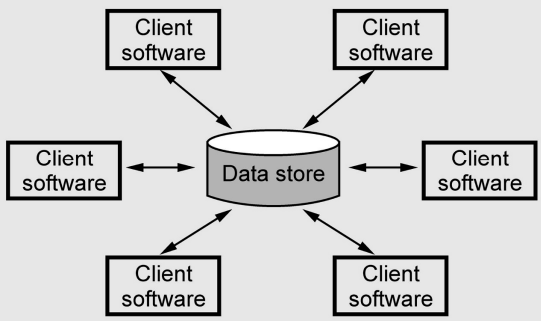
Fig. 4.13.2 Pipes and filters

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

Fig. 4.13.3 Batch sequential

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

Comparison between Data Flow and Data Center Architecture

Data flow architecture	Data centered architecture
In this architecture series of transformations are applied to produce output data.	In this architecture the data store lies at the center of the architecture and other components frequently access it by performing add, delete, and modify operations.
In this architecture the set of components are called filters and are connected by pipes to transform data from one component to another.	In data centered architecture the components are the database elements such as tables and queries. And the communication among these components takes place with the help of relationship.
The filters normally work independently without bothering about working of neighboring filter.	The components of this architecture perform with the central repository without affecting the working of other components.
	

4.13.1.3 Call and Return Architecture

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

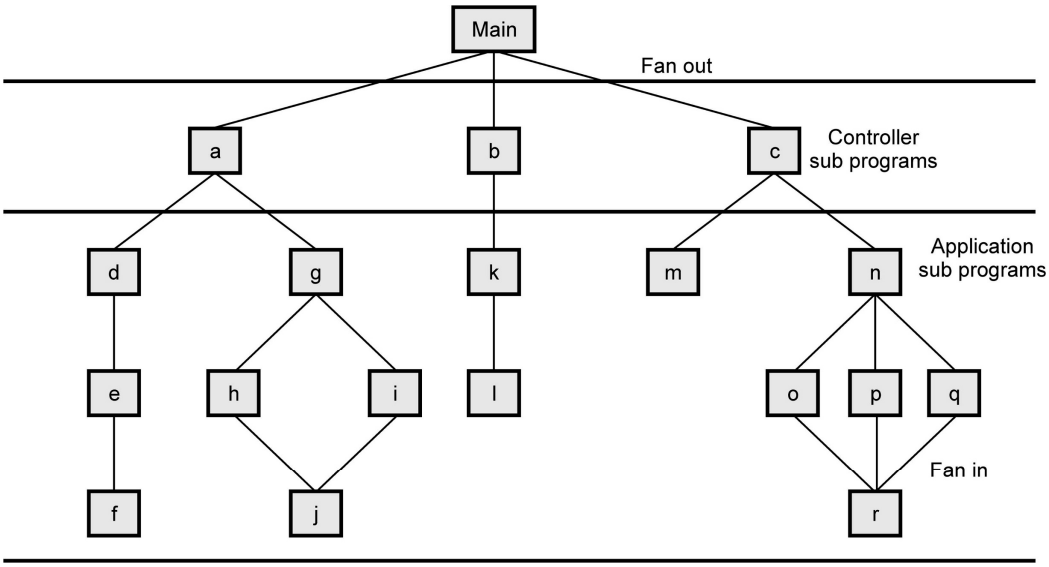


Fig. 4.13.4 Call and return architecture

4.13.1.4 Object Oriented Architecture

- In this architecture the system is decomposed into number of interacting objects.
- These objects encapsulate data and the corresponding operations that must be applied to manipulate the data.
- The object oriented decomposition is concerned with identifying objects classes, their attributes and the corresponding operations. There is some control models used to co-ordinate the object operations.

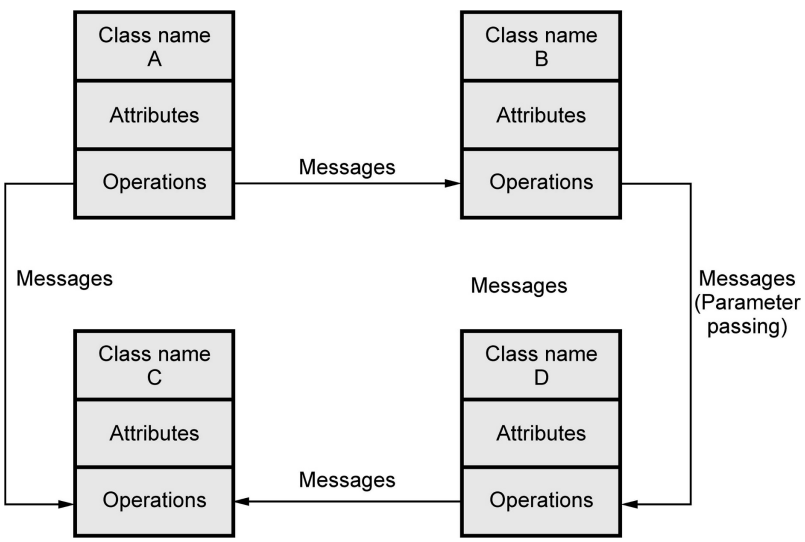


Fig. 4.13.5 Object oriented architecture

4.13.1.5 Layered Architecture

- The layered architecture is composed of different layers. **Each layer** is intended to perform **specific operations** so machine instruction set can be generated. Various components in each layer perform specific operations.
- The **outer layer** is responsible for performing the **user interface** operations while the components in the inner layer perform operating system interfaces.
- The components in **intermediate layer** perform **utility services** and **application software** functions.

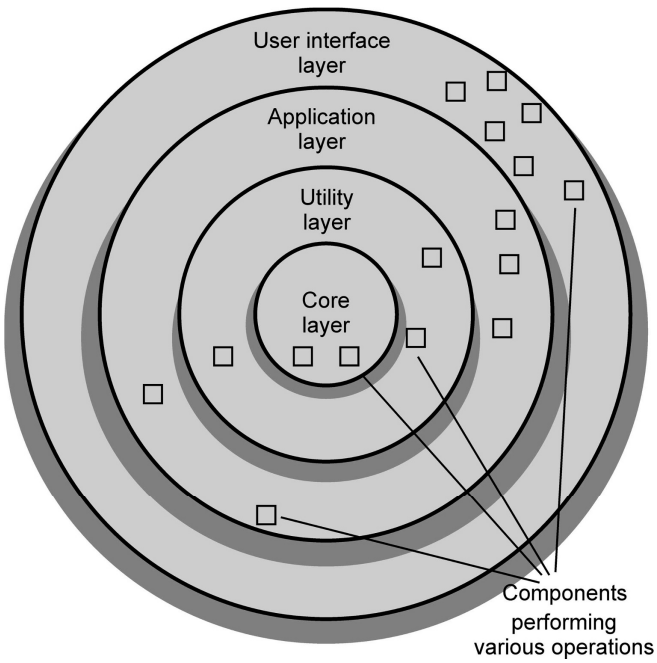


Fig. 4.13.6 Layer architecture of component

4.13.2 Architectural Patterns

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

1. Concurrency

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

2. Persistence

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

3. Distribution

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

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

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

Review Questions

1. Explain data centered and layered architectures with neat diagrams. **SPPU : Dec.-11, Marks 8**
2. Explain any 3 architecture styles. **SPPU : May-12, Marks 6**
3. Explain in brief call and return architecture. **SPPU : Dec.-12, Marks 4**
4. What are different architectural styles exist for software ? Explain any one software architecture in detail. **SPPU : April-16, In Sem, Marks 4**
5. Explain in detail data-centered architectural style. **SPPU : May-18, End Sem, Marks 5**
6. Explain layered architecture style with neat diagrams. **SPPU : Dec.-18, End Sem, Marks 5**
7. Explain in detail call and return architectural style. **SPPU : May-19, End Sem, Marks 5**

4.14 Multiple Choice Questions

Q.1 ____ is a process of translating analysis model into design model.

a Software designing b Coding

c Requirements gathering d All of the above

Q.2 A _____ is a named collection of data that describes a data object.

a Data encapsulation b Data abstraction

c Data hiding d None of these

Q.3 In ____ coupling the global variables are used.

a stamp b data

c common d content

Q.4 ____ and ___ are two qualitative criteria used for functional independence.

a Cohesion, coupling b Abstraction, refinement

c Data hiding and encapsulation d None of these

Q.5 The goal of good design is ____ cohesion and ___ coupling

a low high b high, low

c low, low d high, high

Q.6 Which of the following is a tool in design phase ?

a Abstraction b Refinement

c Information hiding d All of the above

Q.7 Information hiding is to hide information from user ____.

a that is relevant to him

b that is irrelevant to him

c that can be maliciously handled by him

d none of these

Q.8 Design phase includes _____.

a data, architectural and procedural designs only

b architectural, procedural and interface design only

c data, architectural and interface design only

d data, architectural interface and procedural design only

Q.9 _____ incorporates data, architectural, interface and procedural representations of the software.

a Design model b User's model

c System image d All of these

Q.10 What is the meaning of functional cohesion ?

a Operations are part of single functional task and are placed in same procedures.

b All operations that access the same data are defined within one class.

c All operations that access the data from outside the module.

d None of the above.

Q.11 Which is the worst type of coupling ?

a Control coupling

b Data coupling

c Content coupling

d Stamp coupling

Q.12 Stepwise refinement is _____ strategy suggested by Niklaus WIRTH.

a top down

b bottom up

c either top down or bottom up

Q.13 Commonly used architectural styles are _____.

a data centered architecture

b data flow architecture

c call and return architecture

d all of these

Q.14 During architectural design at the initial stage ___ is prepared.

a use case diagram

b context model

c component diagram

d none of these

Q.15 Control systems may make use of the environmental control pattern, which is a general control pattern that includes _____processes.

a sensor

b actuator

c pipeline

d both sensor and actuator

Q.16 A _____ view shows the system hardware and how software components are distributed across the processors in the system.

a physical

b logical

c process

d all of the mentioned

Q.17 Which view in architectural design shows the key abstractions in the system as objects or object classes ?

a Physical

b Development

c Logical

d Process

Q.18 _____ is an indication of the relative functional strength of a module.

a Coupling b Cohesion

c Coordination d Quality

Q.19 Coupling is a measure of _____.

a relative functional strength b interdependence among module

c both of the above d none of the above

Q.20 _____ classes represent data stores.

a Process b User interface

c System d Persistent

Q.21 UML is mainly used for ___ design.

a functional design b web design

c structural design d object oriented design

Answer Keys for Multiple Choice Questions :

Q.1	a	**Q.2**	b	**Q.3**	c	**Q.4**	a
Q.5	b	**Q.6**	d	**Q.7**	c	**Q.8**	d
Q.9	a	**Q.10**	a	**Q.11**	c	**Q.12**	a
Q.13	d	**Q.14**	b	**Q.15**	d	**Q.16**	a
Q.17	c	**Q.18**	b	**Q.19**	b	**Q.20**	d
Q.21	d						

❑❑❑

Notes

UNIT - V

5 Risk and Configuration Management

Syllabus

Risk Management : *Software Risks, Risk Identification, Risk Projection, Risk Refinement, Risk Mitigation, Monitoring, and Management, The RMMM Plan.*

Software Configuration Management : *Software Configuration Management, The SCM Repository The SCM Process, Configuration Management for any suitable software system.*

Contents

Part I : Risk Management

5.1 Introduction to Concept of Risk management

- **Definition of risk :** The risk denotes the uncertainty that may occur in the choices due to past actions and risk is something which causes heavy losses.
- **Definition of risk management :** Risk management refers to the process of making decisions based on an evaluation of the factors that threats to the business.
- Various **activities** that are carried out for **risk management** are -
  1. Risk identification
  2. Risk projection
  3. Risk refinement
  4. Risk mitigation, monitoring and management.

5.2 Reactive Vs. Proactive Risk Strategies

SPPU : May-14, Marks 6

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

Reactive risk strategy

- Reactive risk management is a risk management strategy in which when project gets into trouble then only **corrective action** is taken. But when such risks can not be managed and new risks come up one after the other, the software team flies into action in an attempt to correct problems rapidly. These activities are called **"firefighting" activities**.
- Resources are utilized to manage such risks. And if still the risks do not get managed then project is in danger.
- In this strategy no preventive care is taken about the risks. They are handled only on their occurrences.
- This is an older approach of risk management.

Proactive risk strategy

- Proactive risk management strategy begins before the technical activity by **considering the probable risk**.
- In this strategy potential risks are identified first then their probability and impact is analyzed. Such risks are then specified according to their priorities (i.e. high priority risks should be managed first!). Finally the software team prepares a plan for managing these risks.
- The objective of this strategy is to **avoid the risks**(*prevention is better than cure!!!)*. But it

is not possible to avoid all the risks, hence team prepares the risk management plan in such a manner that risk controlling can done efficiently.

- This is an **intelligent strategy** for risk management and now a day it is used by most of the IT industries.

Review Question

1. What are the types of risks ? Explain in brief. **SPPU : May-14, Marks 6**

5.3 Software Risks

SPPU : Dec.-17, Marks 6

There are **two characteristics** of the risks

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

Different types of risk

1. Project risk

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

2. Technical risk

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

3. Business risk

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

i) Market risk - When a quality software product is built but if there is no customer for this product then it is called market risk (i.e. *no market for the product)*.

ii) Strategic risk - When a product is built and if it is not following the company's business policies then such a product brings strategic risks.

iii) Sales risk - When a product is built but how to sell is not clear then such a situation brings sales risk.

iv) Management risk - When senior management or the responsible staff leaves the organization then management risk occurs.

v) Budget risk - Losing the overall budget of the project is called budget risk.

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

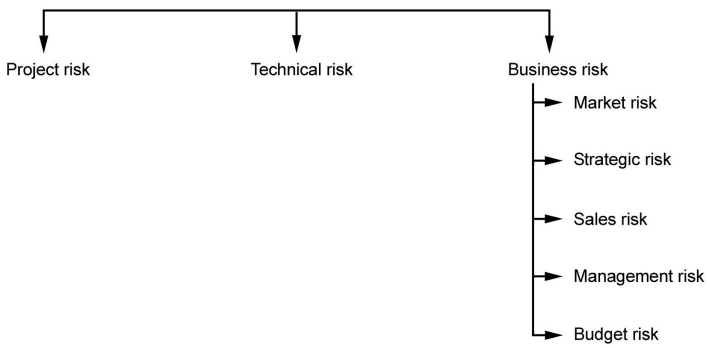


Fig. 5.3.1 Categorization of risk

- **Known risks** are those risk that are identified after evaluating the project plan. These risks can also be identified from other sources such as environment in which the product gets developed, unrealistic dead lines, poor requirement specification and software scope. There are two types of known risks - *predictable* and *unpredictable* risks.
  - ***Predictable risks*** are those risks that can be identified in advance based on past project experience. For example : Experienced and skilled staff leaving in between or improper communication with customer resulting in poor requirement specification.
  - ***Unpredictable risks*** *are* those risks that can not be guessed earlier.

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

Review Question

1. Explain various risk associated with software project. How they are managed ?

SPPU : Dec.-17, End Sem, Marks 6

5.4 Risk Identification

SPPU : May-13, 18, Dec.-13, 16, 18, 19, Marks 8

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

The risk identification is based on **two approaches**

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

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

Step 1 : Preparation of risk item check list

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

i) **Product size -** The risk items based on overall size of the software product is identified.

ii) **Business impact -** Risk items related to the marketplace or management can be predicted.

iii) **Customer characteristics -** Risks associated with customer-developer communication can be identified.

iv) **Process definition -** Risks that get raised with the definition of software process. This category exposes important risks items because whichever is the process definition made, is then followed by the whole team.

v) **Development environment -** The risks associated with the technology and tool being used for developing the product.

vi) **Staff size and experience -** Once the technology and tool related risks items are identified it is essential to identify the risk associated with sufficient highly experienced and skilled staff who will do the development.

vii) **Technology to be built -** complexity of the system should be understood and related risk items needs to be identified.

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

Step 2 : Creating risk components and drivers list.

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

Let us understand which are the risk components and drivers.

5.4.1 Risk Components and Drivers

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

1. ***Performance risk*** **-** It is the degree of uncertainty that the product will satisfy the requirements
2. ***Cost risk*** **-** It is the degree of uncertainty that the project will maintain the budget.

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

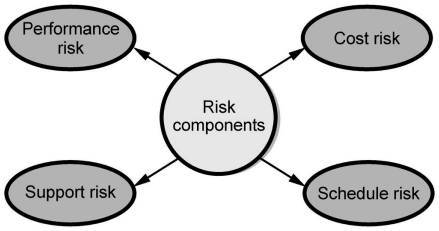


Fig. 5.4.1 Components of risk

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

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

Fig. 5.4.2

5.4.2 How to Asses Overall Project Risk ?

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

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

10. Is the size of team sufficient to develop the required product ?
11. Is that all the customers know the importance of the product/ requirements of the system to be built ?

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

Review Questions

1. Explain the risk identification and assessment process for a software project
   SPPU : May-13, Dec.-13, Marks 8
2. What is risk identification? What are different categories of risk ?
   SPPU : Dec.-16, End Sem, Marks 7
3. What is risk identification? What are various categories of risks ?
   SPPU : May-18, Dec.-18, 19, End Sem, Marks 6
4. Explain risk identification and assessment process for software project.
   SPPU : Dec.-19, End Sem, Marks 6

5.5 Risk Projection

SPPU : May-11, 19, Marks 8

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

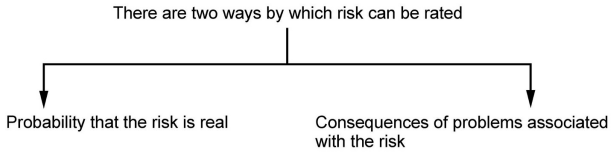


Fig. 5.5.1

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

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

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

5.5.1 Building Risk Table

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

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

While building the risk table

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

2. After building this table it is then sorted by probability and impact. The high probability and high impact risks will be at the top of the table. And low probability and low impact risk will be at the bottom of the table. This arrangement of the table is called **first-order prioritization**.

3. Then the project manager goes through this first-order prioritized risk table and draws a horizontal line at some point in the table. This line is called **cut off line**. The risks table above the cut off line is now considered for further risk analysis.

4. The risk table below the cut off line is again sorted and a **second-order prioritization** is applied on this table.

5. The risk table above the cut-off line is having the risks with high probability and high impact and such risks should occupy the significant amount of management time.
6. All the risks that lie above the cut off line should be managed. Using Risk mitigation, monitoring and management plan the last column of the risk table is filled up.

5.5.2 Assessing Risk Impact

- While assessing the risks impact three factors are considered
  1) Nature of risk
  2) Scope of the risk
  3) Timing at which risk occurs.
- **Nature** of risk denotes the type or kind of risk. For example if software requirement is poorly understood, the software processes gets poorly designed and ultimately it will create a problem in unit testing.
- **Scope** of the risk means severity of the risk.
- And **timing** of risk means determining at which phase of software development life cycle the risk will occur and how long it will persist.

  U.S. Air Force has suggested following steps in order to determine the impact of risk -
  1. The probability of all the components of risk (*performance, cost, support and schedule*) is calculated and averaged.
  2. Using risk drivers (*catastrophic, critical, marginal, negligible)* the impact of risk on each components is determined.
  3. Build the risk table and analyse the high impact, high probability risks.

Risk exposure

The risk exposure can be calculated by following formula

Risk Exposure = Probability of occurrence of risk × Cost

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

First of all we will compute

cost = Number of components*LOC*cost of each LOC

= 15*500*10=$75000

Then Risk Exposure = Probability of occurrence of risk × Cost

= 77 / 100 *75000

= $57750

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

Review Questions

1. How risk projection is carried out risk table ? **SPPU : May-11, Marks 8**
2. How risk projection is carried out using risk table? **SPPU : May-19, End Sem, Marks 5**

5.6 Risk Refinement

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

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

5.7 Risk Mitigation, Monitoring and Management

SPPU : May-12, 14, Dec.-19, Marks 10

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

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

Risk mitigation

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

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

7. For conducting every critical activity during software development, provide the additional staff if required.

Risk monitoring

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

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

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

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

The **objective** of risk monitoring is

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

Risk management

- Project manager performs this task when risk becomes a reality.
- If project manager is successful in applying the project mitigation effectively then it becomes very much easy to manage the risks.
- **For example**, consider a scenario that many people are leaving the organization then if sufficient additional staff is available, if current development activity is known to everybody in the team, if latest and systematic documentation is available then any 'new comer' can easily understand current development activity. This will ultimately help in continuing the work without any interval.

Review Questions

1. What is risk mitigation, monitoring and management (RMMM) ? Write a note on it. **SPPU : May-12, Marks 10**
2. Write short note on : RMMM. **SPPU : May-14, Marks 5**
3. What is risk mitigation, monitoring, and management (RMMM) **SPPU : Dec.-19, End Sem, Marks 5**

5.8 The RMMM Plan

SPPU : May-18, Dec.-17,18, Marks 6

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

<table>
<tr><td colspan="4">Risk information sheet</td></tr>
<tr><td colspan="4">Project name <enter name of the project for which risks can be identified></td></tr>
<tr><td>Risk id <#></td><td>Date
<date at which risk is identified ></td><td>Probability
<risk probability></td><td>Impact
<low/medium/high></td></tr>
<tr><td colspan="2">Origin
<the person who has identified the risk></td><td colspan="2">Assigned to
<who is responsible for mitigating the risk></td></tr>
<tr><td colspan="4">Description
<Description of risk identified></td></tr>
<tr><td colspan="4">Refinement/Context
<associated information for risk refinement></td></tr>
<tr><td colspan="4">Mitigation/Monitoring
<enter the mitigation/monitoring steps taken></td></tr>
<tr><td colspan="4">Trigger/Contingency plan
<if risk mitigation fails then the plan for handling the risk></td></tr>
<tr><td colspan="4">Status
<Running status that provides a history of what is being done for the risk and changes in the risk. Include the date the status entry was made></td></tr>
<tr><td colspan="2">Approval
<name and signature of person approving closure>.</td><td colspan="2">Closing date
<date></td></tr>
</table>

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

Example 5.8.1 *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.* **SPPU : Dec.-17, Marks 6**

Solution : Refer section 5.8

Risk Involved in Computerized examination system

Risks	Probability	Impact
Computer crash	70 %	Catastrophic
Late delivery	30 %	Critical
Technology will not meet expectations	20 %	Catastrophic
Change in requirement	25 %	Critical
Deviation from software engineering standards	10 %	Marginal
Poor comments in code	20 %	Negligible

Following is RMMM information for sample two risks mentioned above -

Risk : Computer Crash

Mitigation : The computer crash result in loss of data. Due to loss of data the online examination system can not be delivered properly. Such a system will not be acceptable by the customer.

Monitoring : While working on building of online examination system, the staff member should always be aware of the stability of computing environment. Any changes in stability of environment should be recognized and taken seriously.

Management : The lack of stable system is extremely hazardous to software development team. If such unstable system is found then development team should cease work on the system until the environment is made stable again.

Risk : Late Delivery

Mitigation : The cost associated with late delivery of online examination system is critical. This result in rescheduling of online examination system.

Monitoring : Realistic schedule must be established for monitoring the project status.

Management : If the project cannot be delivered on time then the university can not conduct the examination system properly. If it becomes apparent that the project will not be completed on time, the only course of action available would be to request an extension to the deadline.

Review Question

1. Explain RMMM plan with suitable example. **SPPU : May-18, Dec.-18, End Sem, Marks 6**

Part II : Software Configuration Management

5.9 Concept of Software Configuration Management

SPPU : Dec.-17, May-19, Marks 8

Definition : Software configuration management is a set of activities carried out for identifying, organizing and controlling changes throughout the lifecycle of computer software.

- During the development of software change must be managed and controlled in order to improve quality and reduce error.
- Hence Software Configuration Management is a **quality assurance activity** that is applied throughout the software process.
- The **origins of changes** that are requested for software are -
  1. New business or market positions cause the changes in the requirements. Due to which the changes need to occur.
  2. New stakeholders may require some changes in the existing requirements.
  3. Due to business growth or project extension, it is essential to makes changes in the project.
  4. Sometimes due to schedule or budge constraints, changes are must in project.
- The software configuration management is concerned with managing evolving software systems.
- Software configuration management is a set of tracking and control activities that begin when a software development project begins and terminates when the software is taken out of operation.

5.9.1 Goals in Change Management

- During software development process, various roles and tasks are involved.
- The goal of project manager is to ensure that the project is getting completed within given schedule and budget. Hence project managers continuously track the progress of the project.
- The configuration managers ensures that the changes in the projects is appropriately taking place and the testing is conducted thoroughly after the modification in the project.
- The goal of software engineer is to work on the assigned module to produce efficient code and error free code.
- Finally the customer uses the product, analyze it and may request for the change or find out the bugs.

5.9.2 Elements of Configuration Management System

Susan Dart identified four important elements for software configuration management system. These elements are -

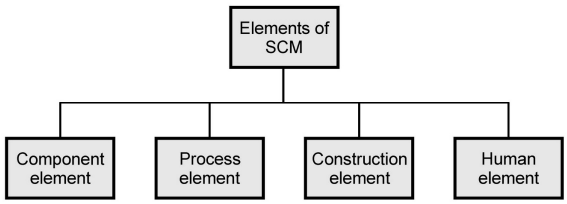


Fig. 5.9.1 Elements of SCM

1. **Component Elements :** It consists of collection of tools that are used for file management system. For example - databases.
2. **Process Elements :** It consists of actions and tasks used during change management and use of software.
3. **Construction Elements :** It is a collection of tools that automate the construction of software.
4. **Human Elements :** It consists of set of tools that are used by software team to implement software configuration management.

5.9.3 Baselines

- The IEEE (IEEE Std. No. 610.12-1990) defines a baseline as :
  - A **specification or product** that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.
- A baseline is a milestone in the development of software that is marked by the delivery of one or more software configuration items and the approval of them is obtained through formal technical review.
- The baseline is the shared project database. It is an SCM task to maintain the integrity of the set of artifacts.
- The elements of a design model have been first documented and reviewed. From this design model errors are identified and corrected. Once all parts of the model have been reviewed, corrected and then approved, the design model becomes a baseline.

- Following figure represents the common software baseline.

Step 1 : The Software engineering task produce one or more SCIs.

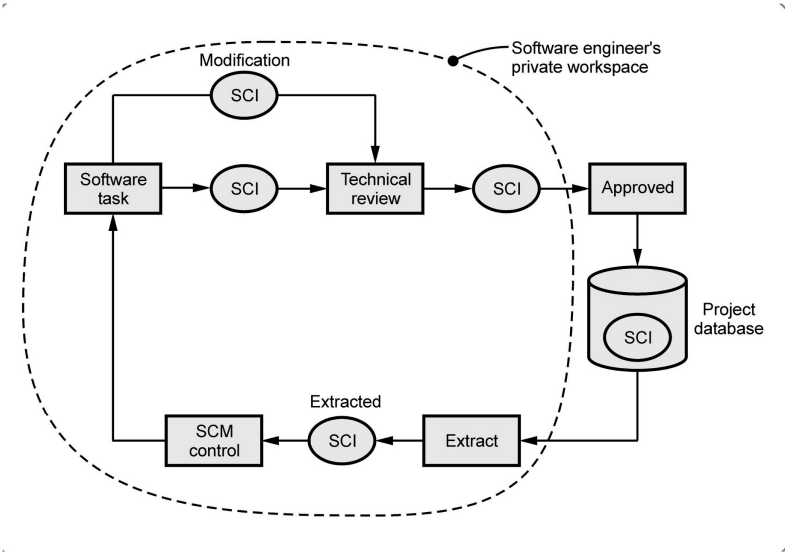


Fig. 5.9.2 Software baseline

Step 2 : These SCIs are then reviewed and approved.

Step 3 : These are then placed in project database or project library.

Step 4 : When software engineer wants to make changes to baselined SCI, it is copied to software engineers private workspace.

Step 5 : The extracted SCI is modified only if SCM controls are followed.

5.9.4 Software Configuration Items

- Software configuration item is basically the **information** that is created as a part of software engineering process.
- **Examples** of Software Configuration Items are
  - Computer programs
    - Source programs
    - Executable programs
  - Documents describing the programs
    - Technical manual
    - Users manual

- Data
  - Program components or functions
  - External data
  - File structure

- For each type of item, there may be a large number of different individual items produced. For instance there may be many documents for a software specification such as project plan, quality plan, test plan, design documents, programs, test reports, review reports.
- These SCI or items will be produced during the project, stored, retrieved, changed, stored again, and so on.
- Each configuration item must have a unique name, and a description or specification which distinguishes it from other items of the same type.

Review Questions

1. What are the elements that exist when an effective SCM system is implemented? Discuss each briefly. **SPPU : Dec.-17, End Sem, Marks 6**
2. What do you understand by Software Configuration management(SCM)? Discuss the importance of SCM. **SPPU : May-19, End Sem, Marks 8**

5.10 The SCM Repository

SPPU : Dec.-17, 18, May-18, Marks 6

- Software configuration Items (SCI) are maintained in project repository or project library.
- The software repository is basically a database that acts as a center for both accumulation and storage for software engineering information.
- The software engineer interacts with repository using tools that are integrated within it.

5.10.1 Role of Project Repository

- Software repository is a **collection of information** accessed by software engineers to make appropriate changes in it.
- This repository is handled using all the modern database management functions.
- It must maintain important properties such as data integrity, sharing and integration.
- The repository must maintain uniform structure and format for software engineering work products.
- To achieve these capabilities, the repository is defined in terms of **meta-model**. The meta model determines -
  i) How information is stored in repository ?

ii) How data can be accessed by using the tool ?

iii) How the security and integrity of the data is maintained ?

iv) How the existing model can be extended to accommodate new requirements ?

5.10.2 Features

The Software Configuration Management can be performed by interacting with repository. The repository must have toolset that provides support for following features –

1. **Versioning :** As project progresses various versions of individual work products may get created. The repository must be able to maintain all these versions and permit the developer to go back to previous versions during testing and debugging.
2. **Dependency Tracking and Change Management** : The data elements stored in the repository are related to each other. The repository must have an ability to keep track of these relationship and to maintain data integrity.
3. **Requirements Tracing :** If the constructed components, its designed is tracked, then particular requirement can be traced out. The repository must have this ability to trace the requirement from constructed components.
4. **Configuration Management :** The repository must be able to keep track of series of configurations representing project milestones and production releases.
5. **Audit Trails :** The audit trail

Review Questions

1. What is software SCM repository? Explain the features of tool set supporting SCM repository. **SPPU : Dec.-17, 18, End Sem, Marks 6**
2. Explain the SCM repository in detail. What are the advantages of SCM repository ? **SPPU : May-18, End Sem, Marks 6**

5.11 The SCM Process

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

The **primary objectives** of Software Configuration Management process (SCM) are -

1. **Configuration Identification :** Identify the items that define the software configuration.
2. **Change Control :** Manage changes to one or more items.
3. **Version Control :** Facilitate to create different versions of the application.
4. **Configuration Authentication :** To ensure that the quality of the software is maintained as the configuration evolves over the time.

The SCM process must be developed in such a way that the software team must answer the following set of questions -

1. How does the software team identify the software configuration items ?
2. How does the software team control the changes in the software before and after delivering it to the customer ?
3. How does the software team manage the versions of the programs in the software package ?
4. How does team get ensured that the changes are made properly ?
5. Who is responsible for approving the changes in the software ?

The answers to these questions lead the definition of five tasks of SCM and those are - Identification, change control, version control and configuration audit and status reporting.

5.11.1 Identification of Objects in Software Configuration

The software configuration items must be separately named and identified as object.

- These objects must be arranged using **object oriented approach**.
- There are two categories of objects - **basic objects** and **aggregate objects**.
- The **basic object** is unit of information created during requirements analysis, design, coding or testing. For example basic object can be part of **source code**.
- Aggregate object is a collection of basic objects and other aggregate objects. For example SRS or data model can be aggregate object.
- Each object can be uniquely identified because it has got -
  1. **Name :** The name of the object is nothing but the collection of characters (string) or some text. It is unique.
  2. **Description :** For describing the object, the object description can be given. This description contains document, program or some other description such as project identifier or version information.
  3. **List of resources :** The resources are the entities that are used for accessing, referencing and processing of objects. The data types and functionalities can serve as a resource.
  4. **Realization or identification :** It is pointer to object.
- The configuration object identification can also consider relationships that exist between the named objects.
- If a change is made to one configuration object it is possible to determine which other configuration objects in the repository are affected by the change.

- Basically object evolve throughout the software process. During the process of object identification the evolution of objects along with its process must be identified.
- Major modifications in the object must be noted.

5.11.2 Change Control

- Changes in any software projects are vital. Sometimes, introducing small changes in the system may lead to big problems in product.
- Similarly, introducing some changes may enhance the capabilities of the system.
- According to **James Bach** too little changes may create some problems and too big changes may create another problems.
- For a large software engineering project, uncontrolled change creates lot of chaos. For managing such changes, human procedures or automated tools can be used.
- The change control process is shown by following Fig. 5.11.1 (See Fig. 5.11.1 on Next page)

Step 1 : First of all there arises a need for the change.

Step 2 : The change request is then submitted by the user.

Step 3 : Developers evaluate this request to assess technical merits, potential side effects, and overall impact on system functions and cost of the project.

Step 4 : A change report is then generated and presented to the Change Control Authority (CCA).

Step 5 : The change control authority is a person or a group of people who makes a final decision on status or priority of the change.

Step 6 : An Engineering Change Order **(ECO)** is generated when the change gets approved. In ECO the change is described, the restrictions and criteria for review and audit are mentioned.

Step 7 : The object that needs to be changed is checked out of the project database.

Step 8 : The changes are then made on the corresponding object and appropriate SQA activities are then applied.

Step 9 : The changed object is then checked in to the database and appropriate version control is made to create new version.

The checked in and checked out mechanisms require **two important elements** -

- Access control
- Synchronization control

The **access control** mechanism gives the authority to the software engineer to access and modify the specific configuring object. The **synchronization control** mechanism allows to

make parallel changes or the changes made by two different people without overwriting each other's work.

5.11.3 Version Control

- Version is an instance of a system which is functionally distinct in some way from other system instances.
- Version control works to help manage different versions of configuration items during the development process.

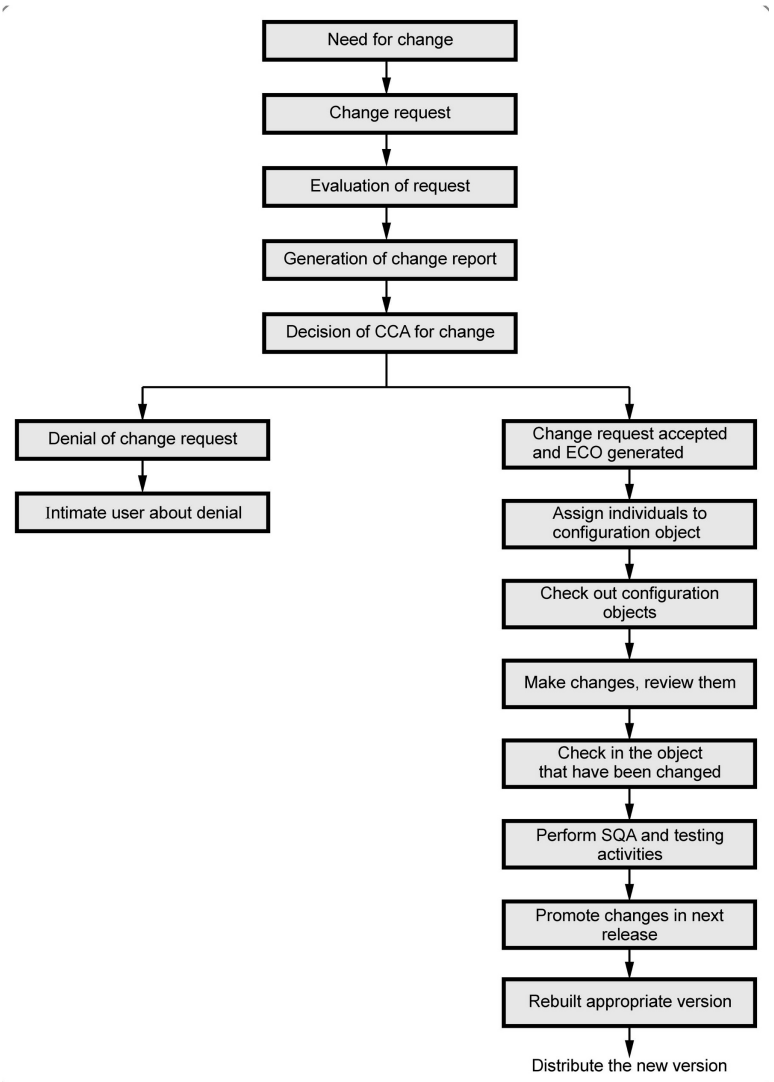


Fig. 5.11.1 Change control process

- The configuration management allows a user to specify the alternative configurations of the software system by selecting appropriate version.
- Certain attributes are associated with each software version. These attributes are useful in identifying the version. For example : The attribute can be 'date', 'creator', 'customer', 'status'.
- In practice the version needs an associated name for easy reference.
- Different versions of a system can be shown by an evolution graph as shown in Fig. 5.11.2.
- Each version of software system is a collection of software configuration items.

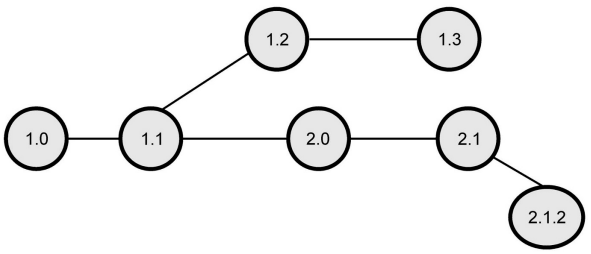


Fig. 5.11.2 Version numbering in evolution graph

5.11.4 Configuration Audit

- In order to ensure that the change has been properly implemented or not two activities are carried out.
  1. Formal Technical Review (FTR)
  2. Software Configuration Audit
- In Formal Technical Review, the correctness of configuration object is identified and corrected. It is conducted by technical reviewer.
- The software configuration audit assess the configuration object for the characteristics that are not reviewed in formal technical review. It is conducted by software quality assurance group.
- Following are some primary questions that are asked during configuration audit -
  1. Whether FTR is conducted to assess the technical correctness ?
  2. Whether or not the change specified by ECO has been made ?
  3. If additional changes need to be made or not ?
  4. Whether the software engineering standards are properly followed ?
  5. Do the attributes of configuration objects reflect the change ?
  6. Whether all the SCI are updated properly ?

7. Whether the SCM process (object identification, change and version control, configuration audit and status reporting) are properly followed ?

- The above questions can be asked as a part of formal technical review.

5.11.5 Status Reporting

The status reporting focuses on **communication of changes to all people** in an organization that involve with changes.

During status reporting following type of questions were asked.

1. What happened ? : What are the changes that are required ?

2. Who did it ? : Who will be handling these changes ?

3. When did it happen ? : The time at which these changes are arised.

4. What else will be affected ? : The objects or part of the software that might be reflected due to these changes.

Review Questions

1. What is Software Configuration Management (SCM)? Explain the change control mechanism in SCM. **SPPU : May-16, End Sem, Marks 8**
2. What are the layers of SCM process? Explain each in detail. **SPPU : May-18, 19, Dec.19, End Sem, Marks 6**
3. What is software configuration management? Explain the change control mechanism in software configuration management **SPPU : May-16, Dec.-18, End Sem, Marks 6**
4. Explain change control mechanism in SCM. **SPPU : Dec.-19, End Sem, Marks 5**

5.12 Configuration Management for Any Suitable Software System

We will discuss the configuration management for web based applications.

5.12.1 Dominant Issues

The webapps get developed in a short span of time using customer driven approach. Hence there are four major issues that need to be handled during configuration management for web applications. These issues are -

1. Content :

- The typical web application contains text, graphics, scripts, audio/video files, forms, active web pages and so on.
- There is a **challenge** to **organize** these contents into **configurable objects** and establish **control mechanism** for these objects.

2. People :

- Many times the contents for web apps are developed by the people having no software engineering background. Such people are completely unaware of need for configuration management.
- As a result changes are accommodated in the web app in an uncontrolled manner.

3. Scalability :

- The web app grow significantly with interconnections with information subsystems.
- As size and complexity grows the small changes are difficult to incorporate or these changes may lead to problematic situations.
- Hence as the scalability of web apps grows, the configuration control mechanism need to be rigorously applied.

4. Politics :

- There can be issues related to -
  1. Responsibility for quality control of the web site before publishing the web site.
  2. Responsibility about making changes.
  3. Responsibility for accuracy of information on web page.
  4. Responsibility for cost changes.

People who handle the above mentioned responsibilities adopt configuration management process for web apps.

5.12.2 Content Management

- Content management is a process in which contents are organized, presented to the end user and then displayed in client server environment.
- The content management system is more useful for handling dynamic web application.
- The content management is carried out with the help of three subsystems -

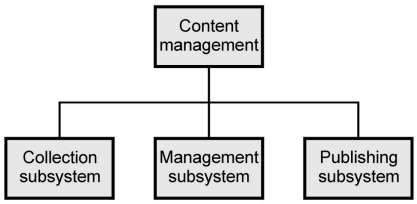


Fig. 5.12.1

1. **Collection subsystem :** Content management system is responsible for
   i) Converting the contents into the form represented by markup languages such as HTML, XML and so on.
   ii) Organized contents for client side display.

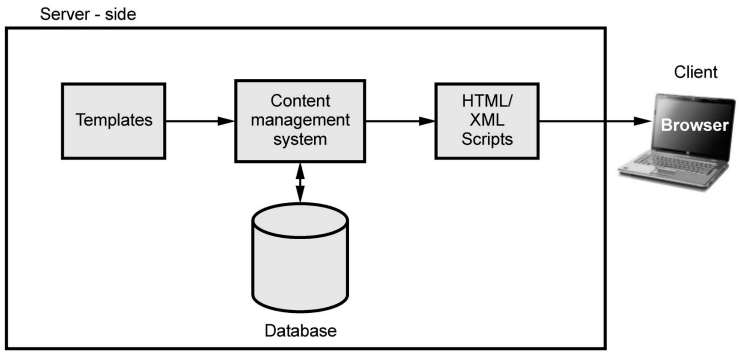


Fig. 5.12.2 Content management system

2. Management Subsystem :

- The **contents** prepared in content management are **stored in repository**.
- The management subsystem implements the repository that contains following elements -
  1. **Content database :** It contains the information structure used for storing the data.
  2. **Database capabilities :** It contains functionalities used for storing and retrieving the data objects.
  3. **Configuration management functions :** It includes functions related to version control, change management, change auditing and reporting.

3. Publishing subsystem :

- The publishing subsystem comes in a picture when contents are ready to present on client browsers.
- Various elements of publishing subsystem are -
  i) static elements such as text, graphics, scripts.
  ii) publication services that include the functionalities related to retrieval and formatting of the contents.
  iii) external services for accessing external infrastructure.

5.12.3 Change Management

- Each change in the change management is categorized into **four classes** -
  - **Class 1 :** This type of content change while correcting the error enhances local functions.
  - **Class 2 :** This type of content change creates influence on other functional component.

- **Class 3 :** This type of content change causes major influence across the web applications.
- **Class 4 :** This type of content change demand for major design change.

- The class 1 and class 2 changes are informally handled using agile method. For the class 2 changes the impact study is done.
- The class 3 changes are reviewed by developers.
- The class 4 changes are reviewed by developers and stakeholders.

5.12.4 Version Control

As the web app gets developed different versions exist at the same time. Version control is a mechanism to manage these versions in appropriate manner.

The version control process is as follows -

Step 1 : There should be some central repository for the web app project. This repository contains current versions of all web app configurable objects.

Step 2 : Each developer creates a working folder. This folder contains current object being created or changed.

Step 3 : The clocks on all the workstations should be synchronized.

Step 4 : The new objects are developed or existing objects are changed.

Step 5 : When objects are imported or exported from the repository, an automatic log messages can be generated.

5.12.5 Auditing and Reporting

- All objects that are checked into or out of the repository are recorded into log. This log can be viewed at any time.
- In addition to that, automatic e-mail notification can be sent to developers when the objects that are checked into or out of the repository.

5.13 Multiple Choice Questions

Q.1 What is risk ?

a Negative consequence that could occur

b Negative consequence that will occur

c Negative consequence that must occur

d Negative consequence that shall occur

Q.2 Risk management is a responsibility of the _____.

a customer b investor

c developer d project team

Q.3 The goal of quality assurance is to provide management with the data needed to determine which software engineers are producing the most defects.

a True b False

Q.4 The problem that threatens the success of a project but which has not yet happened is a ____.

a bug b error

c risk d failure

Q.5 _____risks threaten the quality and timeliness of the software to be built.

a Business risk b Project risk

c Technical risks d Management risk

Q.6 Firefighting activities are associated with _____.

a reactive risk strategy b proactive risk strategy

c both a and b d none of these

Q.7 If P is risk probability, L is loss, then Risk Exposure (RE) is computed as _____.

a RE = P/L b RE = P + L

c RE = P*L d RE = 2* P *L

Q.8 Software risk always involves two characteristics. What are those characteristics ?

a Uncertainty and Loss b Certainty and Profit

c Staff size and Budget d Project deadline and Budget

Q.9 When senior management or responsible staff leaves the organization then following type of risk occurs _____.

a management risk b technical risk

c project risk d all of these

Q.10 When risk is associated with hardware or software technology then following type of risk occurs _____.

a management risk b technical risk

c project risk d all of these

Q.11 RE represents _____ .

a Risk Expense b Related Expense

c Risk Exposure d Risk Evaluation

Q.12 Non-conformance to software requirements is known as _____.

a software availability b software reliability

c software failure d all of the above

Q.13 _____ is the process, which controls the changes made to a system and manages the different versions of the software project.

a Software maintenance b Configuration management

c Software re-engineering d Software refactoring

Q.14 _____ process involve technical change analysis, cost benefit analysis and change tracking.

a Release management b Version management

c Change management d Configuration management

Q.15 _____ is a software configuration management concept that helps us to control change.

a Baseline b Procedure

c Audit d None of the above

Q.16 Which of the following task is not part of software configuration management ?

a Change control b Reporting

c Statistical quality control d Version control

Q.17 What is the main aim of Software Configuration Management (SCM) ?

a Identify change b Control change

c Ensure that change is being properly implemented

d All of the above

Answer Keys for Multiple Choice Questions :

Q.1	a	**Q.2**	d	**Q.3**	b	**Q.4**	c
Q.5	c	**Q.6**	a	**Q.7**	c	**Q.8**	a
Q.9	a	**Q.10**	b	**Q.11**	c	**Q.12**	c
Q.13	b	**Q.14**	c	**Q.15**	a	**Q.16**	c
Q.17	d						

❑❑❑

UNIT - VI

6 Software Testing

Syllabus

A Strategic Approach to Software Testing, Verification and Validation, Organizing for Software Testing, Software Testing Strategy - The Big Picture, Criteria for Completion of Testing, Strategic Issues, Test Strategies for Conventional Software, Unit Testing, Integration Testing, Test Strategies for Object-Oriented Software, Unit Testing in the OO Context, Integration Testing in the OO Context, Test Strategies for WebApps, Validation Testing, Validation-Test Criteria, Configuration Review.

Contents

6.1 Fundamental Concepts of Testing

SPPU : May-14, 16, Dec.-19 Marks 9

Definition : Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding.

The purpose of software testing is to ensure whether the software functions appear to be working according to specifications and performance requirements.

6.1.1 Testing Objectives

According to **Glen Myers** the testing objectives are

1. Testing is a process of executing a program with the intend of finding an error.
2. A good test case is one that has high probability of finding an undiscovered error.
3. A successful test is one that uncovers an as-yet undiscovered error.

The major testing objective is to design tests that systematically uncover types of errors with minimum time and effort.

6.1.2 Testing Principles

Every software engineer must apply following testing principles while performing the software testing.

1. All tests should be traceable to customer requirements.
2. Tests should be planned long before testing begins.
3. The Pareto principle can be applied to software testing - 80 % of all errors uncovered during testing will likely be traceable to 20 % of all program modules.
4. Testing should begin “in the small” and progress toward testing “in the large”.
5. Exhaustive testing is not possible.
6. To be most effective, testing should be conducted by an independent third party.

6.1.3 Why Testing is Important ?

- Generally, testing is a process that requires more efforts than any other software engineering activity.
- Testing is a set of activities that can be planned in advance and conducted systematically.
- If it is conducted haphazardly, then only time will be wasted and more even worse errors may get introduced.

- This may lead to have many undetected errors in the system being developed. Hence performing testing by adopting systematic strategies is very much essential in during development of software.

Review Questions

1. What is importance of testing practices ? What are the principles of testing practices ? **SPPU : May-14, Marks 8**
2. What are the main objectives of software testing and what are the principles of software testing ? **SPPU : May-16, End Sem, Marks 9,Dec.-19, End Sem, Marks 5**

6.2 A Strategic Approach to Software Testing

SPPU : Dec.-12, 13, 18, May-16, 18, 19 Marks 8

- A testing strategy provides a process that describes for the developer, quality analysts and the customer the steps conducted as part of testing. The testing strategy includes
  - Test planning
  - Test case design
  - Test execution
  - Data collection
  - Effectiveness evaluation.
- The strategic approach for software testing can be -
  1. Just before starting the testing process the **formal technical reviews** must be conducted. This will eliminate many errors before the actual testing process.
  2. At the beginning, various **components** of the system are tested, then gradually each **interface** is tested and thus the **entire computer based system** is tested.
  3. Different testing techniques can be applied at different point of time.
  4. The **developer** of the software conducts testing. For the large projects the **Independent Test Groups** (ITG) also assist the developers.
  5. **Testing and debugging** are different activities that must be carried out in software testing.
  6. Debugging also lies within any testing strategy.
- There are two levels specified in the testing strategy and those are **low level** and **high level.** The low level tests that are necessary to verify that small source code segment has been correctly implemented. Similarly the high level tests should be conducted that validate major system functions against customer requirements.

- Who are involving software testing ?
  - Software developers
  - Testers (test engineers) in Independent Test Group (ITG)
  - SQA group i.e. Software Quality Assurance group.

6.2.1 Verification and Validation

- **Verification** refers to the set of activities that ensure that software correctly implements a specific function.
- **Validation** refers to a different set of activities that ensure that the software that has been built is traceable to customer requirements.
- According to Boehm

  Verification : "Are we building the product right ?"

  Validation : "Are we building the right product ?"
- Software testing is only one element of Software Quality Assurance (SQA).
- Quality must be built into the development process, you can't use testing to add quality after the fact.
- Verification and validation involve large number of software quality assurance activities such as -
  - Formal technical reviews
  - Quality and configuration audits
  - Performance monitoring
  - Feasibility study
  - Documentation review
  - Database review
  - Algorithmic analysis
  - Development testing
  - Installation testing

Difference between Verification and Validation

Sr. No.	Verification	Validation
1.	Verification refers to the set of activities that ensure software correctly implements the specific function.	Validation refers to the set of activities that ensure that the software that has been built is traceable to customer requirements.
2.	After a valid and complete specification the verification starts.	Validation begins as soon as project starts.
3.	Verification is for prevention of errors.	Validation is for detection of errors.
4.	Verification is conducted using reviews, walkthroughs, inspections and audits.	Validation is conducted using system testing, user interface testing and stress testing.
5.	Verification is also termed as white box testing or static testing as work product goes through reviews.	Validation can be termed as black box testing or dynamic testing as work product is executed.
6.	Verification finds about 50 to 60 % of the defects.	Validation finds about 20 to 30 % of the defects.
7.	Verification is based on the opinion of reviewer and may change from person to person.	Validation is based on the fact and is often stable.
8.	The verification verifies the problem statements, decisions taken during the development and execution paths.	The validation validates the requirements, functionalities and features of the product.
9.	Verification is about process, standard and guideline.	Validation is about the product.

Review Questions

1. Explain the strategic approach to software testing. **SPPU : Dec.-12, Marks 6**
2. What is software testing ? Explain the software testing strategy for software development. **SPPU : Dec.-13,18, May-19, Marks 8**
3. Differentiate between verification and validation. **SPPU : Dec.-13, May-16, 18, End Sem, Marks 4**

6.3 Organizing for Software Testing

- Testing needs to be conducted from the beginning of the software development process.
- It is desired that the software developer should test the individual unit of software system for its proper functioning. This is called unit testing. That means, there must be an involvement of software developers for unit testing activities.
- In many cases, the software developers also perform the integration testing. Thus the complete software architecture is been tested by the software developers during development phases.
- Then the independent test group(ITG) gets involved in the software testing activity.
- The role of ITG is to remove other inherent problems and conflicts associated with the software system.
- The software developers and the ITG work together to remove all possible errors from the software system and a thorough testing must be performed.
- While testing is conducted, the developer must be available to correct errors that are uncovered.
- The ITG is part of the software development project team in the sense that it becomes involvedduring analysis and design and stays involved throughout a large project.
- In many cases, the ITG reports to **software quality assurance organization** so that the independent testing can be performed.

6.4 Criteria for Completion of Testing

Entry criteria

- Software testing should start early in the Software Development Life Cycle. This helps to capture and eliminate defects in the early stages of SDLC i.e requirement gathering and design phases.
- An early start to testing helps to reduce the number of defects and ultimately the rework cost in the end.
- **Definition :** Entry criteria for testing can be defined as "Specific conditions or on-going activities that must be present before a process can begin." The Software Testing Life Cycle (STLC) specifies the entry criteria required during each testing phase.

- It also defines the time interval or the expected amount of lead-time to make the entry criteria item available to the process.
- Following are the inputs necessary for the entry criteria –
  - The requirements documents
  - Complete understanding of the application flow
  - Test plan

Exit criteria

- **Definition :** It can be defined as "The specific conditions or on-going activities that should be fulfilled before completing the software testing life cycle."
- The exit criteria can identify the intermediate deliverables.
- The following exit criteria should be considered for completion of a testing phase:
  - Ensuring all critical Test Cases are passed
  - Achieving complete Functional Coverage
  - Identifying and fixing all the high-priority defects
- The output achieved through exit criteria are –
  - Test summary report
  - Test logs

6.5 Strategic Issues

- ***Before the testing starts, Specify product requirements appropriately*** **-** Certain quality characteristics of the software such as maintainability, portability and usability should be specified in order to obtain the unambiguous test results.
- ***Specify testing objectives clearly*** **-** Testing objectives such as effectiveness, mean time to failure and cost of defects should be stated clearly in the test plan.
- ***Identify categories of users for the software and their role with reference to software system*** - Use cases describe the interactions among different class of users and thereby testing can focus on the actual use of the product.
- ***Develop a test plan that emphasizes rapid cycle testing*** **-** Test plan is an important document which helps the tester to perform rapid cycle testing (2 percent of project effort).

- ***Build robust software which can be useful for testing -*** The software should be capable of detecting certain classes of errors. Moreover, the software design should allow automated testing and regression testing.
- ***Use effective formal reviews before actual testing process begins -*** Formal technical reviews need to be conducted to uncover errors. The effective technical reviews conducted before testing, reduce significant amount of testing efforts.
- ***Conduct formal technical reviews to assess the test strategy and test cases -*** The formal technical review helps to detect any the lacuna in testing approach. Hence it is necessary to assess the test strategy and test cases by technical reviewers to improve the quality of software.
- ***Throughout the testing process adopt the continuous development strategy -*** The measured test strategy should be used as part of statistical process control approach for software testing.

6.6 Testing Strategies for Conventional Software

SPPU : May-11, 13, 14, 16, 18 Dec.-11, 12, 13, 17, 19 Marks 8

We begin by 'testing-in-the-small' and move toward 'testing-in-the-large'.

Various testing strategies for conventional software are

1. Unit testing
2. Integration testing
3. Validation testing
4. System testing

1. **Unit testing -** In this type of testing techniques are applied to detect the errors from each software component individually.
2. **Integration testing -** It focuses on issues associated with verification and program construction as components begin interacting with one another.
3. **Validation testing -** It provides assurance that the software validation criteria (established during requirements analysis) meets all functional, behavioural and performance requirements.
4. **System testing -** In system testing all system elements forming the system is tested as a whole.

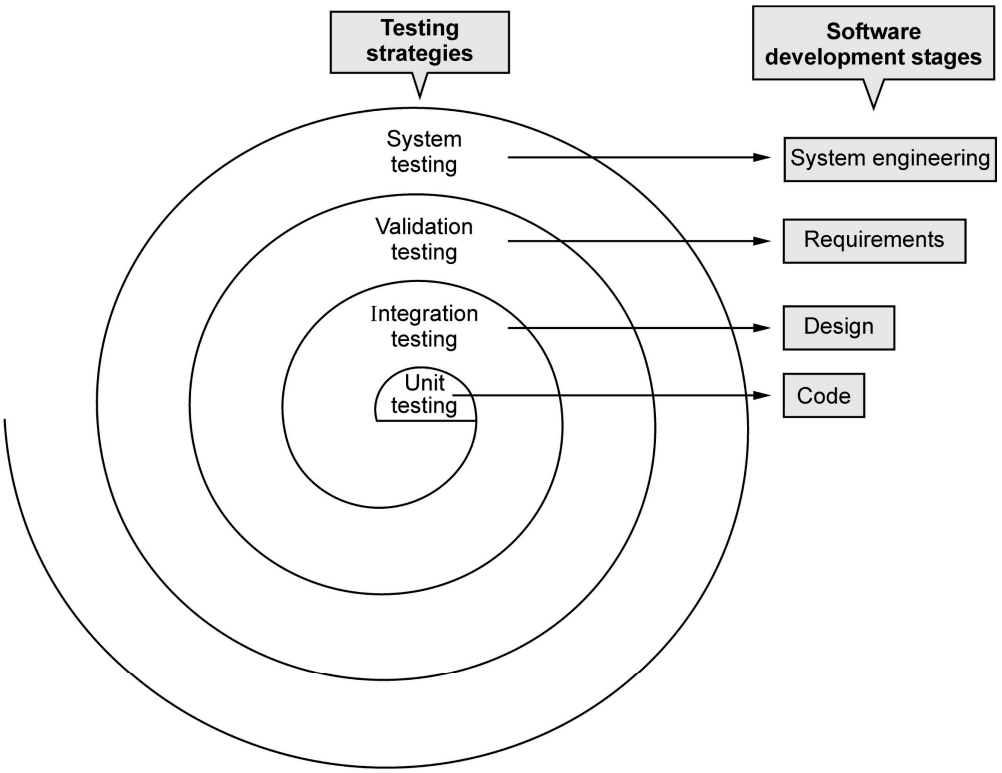


Fig. 6.6.1 Testing strategy

6.6.1 Unit Testing

- In unit testing the individual components are tested independently to ensure their quality.
- The focus is to uncover the errors in design and implementation.
- The various tests that are conducted during the unit test are described as below.
  1. Module interfaces are tested for proper information flow in and out of the program.
  2. Local data are examined to ensure that integrity is maintained.
  3. Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.
  4. All the basis (independent) paths are tested for ensuring that all statements in the module have been executed only once.
  5. All error handling paths should be tested.

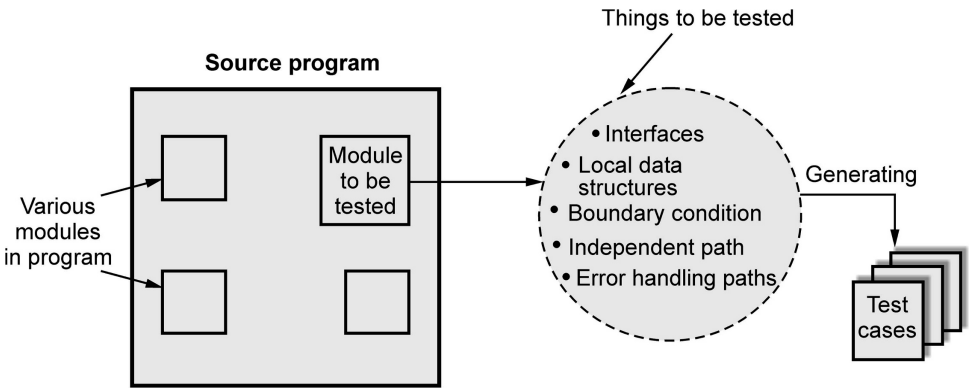


Fig. 6.6.2 Unit testing

6. **Drivers and stub** software need to be developed to test incomplete software. The "driver" is a program that accepts the test data and prints the relevant results. And the "stub" is a subprogram that uses the module interfaces and performs the minimal data manipulation if required. This is illustrated by following Fig. 6.6.3.

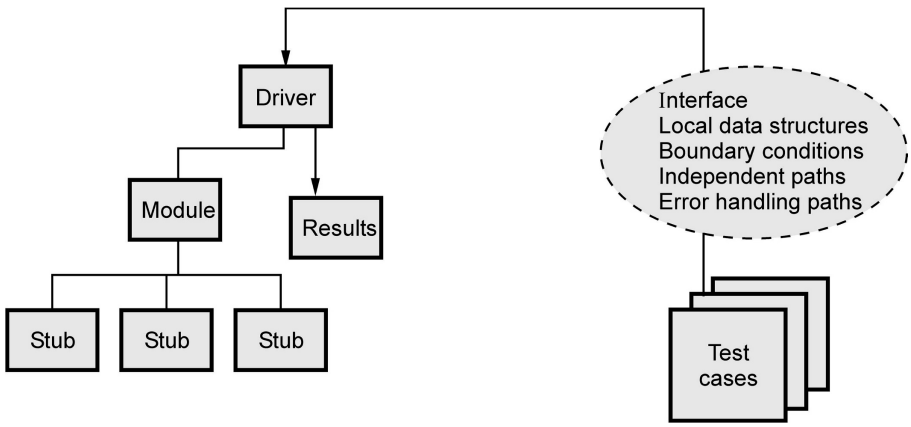


Fig. 6.6.3 Unit testing environment

7. The unit testing is simplified when a component with high cohesion (with one function) is designed. In such a design the number of test cases are less and one can easily predict or uncover errors.

6.6.1.1 Errors Identified during Unit Testing

Various data structure errors that can be identified during the unit testing are -

1. Incorrect arithmetic precedence.
2. Mixed mode operations.
3. Precision inaccuracy.
4. Comparison of different data types.

6.6.1.2 Driver and Stub

- Stubs and drivers are two such elements used in software testing process, which act as a temporary replacement for a module.
- The driver and stubs are generally dummy codes or fake codes that help to simulate the behavior of existing code.
- The stubs and drives are specifically developed to meet the necessary requirements of the unavailable modules and are useful in getting expected results.
- Generally driver and stubs are used in unit testing or during integration testing.

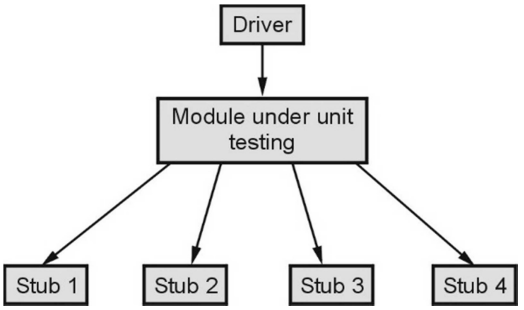


Fig. 6.6.4 Driver and stub

- **Stub**
  - These are dummy code.
  - These are used in testing when lower level of code are not developed.
  - To test the upper level of code the stubs are used.
  - It just accepts the value from calling module and returns null value.
- **Driver**
  - These are dummy code.
  - These are used in testing when upper level of code is not developed.
  - To test the lower level code the drivers are used.
  - Basically we call the driver as main function which calls other modules to form complete applications.

6.6.2 Integration Testing

- A group of dependent components are tested together to ensure their quality of their integration unit.
- The **objective** is to take unit tested components and build a program structure that has been dictated by software design.
- The **focus** of integration testing is to uncover errors in :
  - Design and construction of software architecture.
  - Integrated functions or operations at subsystem level.
  - Interfaces and interactions between them.
  - Resource integration and/or environment integration.

- The integration testing can be carried out using two approaches.
  1. The non-incremental integration
  2. Incremental integration

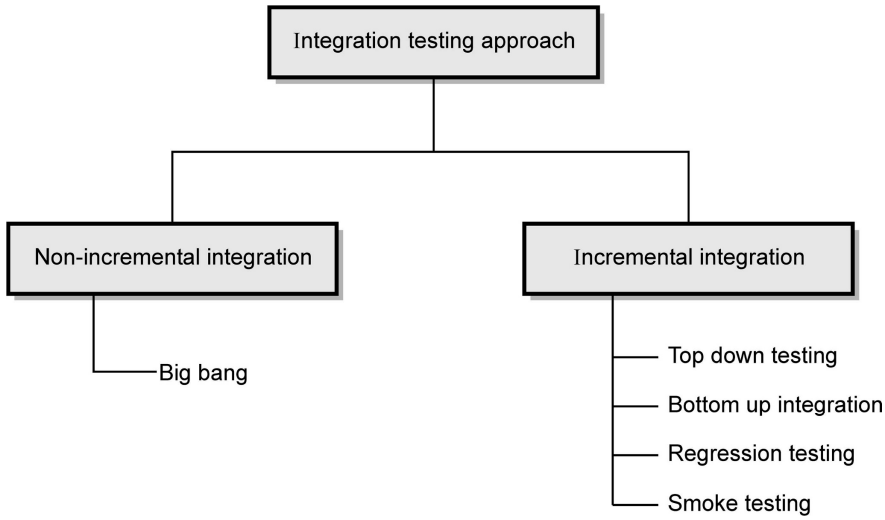


Fig. 6.6.5 Integration testing approach

- The non-incremental integration is given by the "**big bang**" approach. All components are combined in advance. The entire program is tested as a whole. And chaos usually results. A set of errors is tested as a whole. Correction is difficult because isolation of causes is complicated by the size of the entire program. Once these errors are corrected new ones appear. This process continues infinitely.

  Advantage of big-bang : This approach is simple.

  Disadvantages :
  1. It is hard to debug.
  2. It is not easy to isolate errors while testing.
  3. In this approach it is not easy to validate test results.
  4. After performing testing, it is impossible to form an integrated system.

- An incremental construction strategy includes :
  1. Top down integration
  2. Bottom up integration
  3. Regression testing
  4. Smoke testing

6.6.2.1 Top Down Integration Testing

- Top down testing is an incremental approach in which modules are integrated by moving down through the control structure.
- Modules subordinate to the main control module are incorporated into the system in either a depth first or breadth first manner.
- Integration process can be performed using following steps.
  1. The main control module is used as a test driver and the stubs are substituted for all modules directly subordinate to the main control module.
  2. Subordinate stubs are replaced one at a time with actual modules using either depth first or breadth first method.
  3. Tests are conducted as each module is integrated.
  4. On completion of each set of tests, another stub is replaced with the real module.
  5. Regression testing is conducted to prevent the introduction of new errors.

For example :

In top down integration if the depth first approach is adopted then we will start integration from module M1 then we will integrate M2 then M3, M4, M5, M6 and then M7.

If breadth first approach is adopted then we will integrate module M1 first then M2, M6. Then we will integrate module M3, M4, M5 and finally M7.

6.6.2.2 Bottom Up Integration Testing

In bottom up integration the modules at the lowest levels are integrated at first, then integration is done by moving upward through the control structure.

The bottom up integration process can be carried out using following steps.

1. Low-level modules are combined into clusters that perform a specific software subfunction.
2. A driver program is written to co-ordinate test case input and output.
3. The whole cluster is tested.
4. Drivers are removed and clusters are combined moving upward in the program structure.

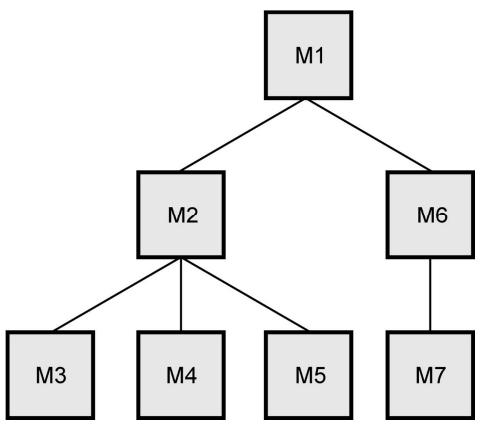


Fig. 6.6.6 Program structure

For example :

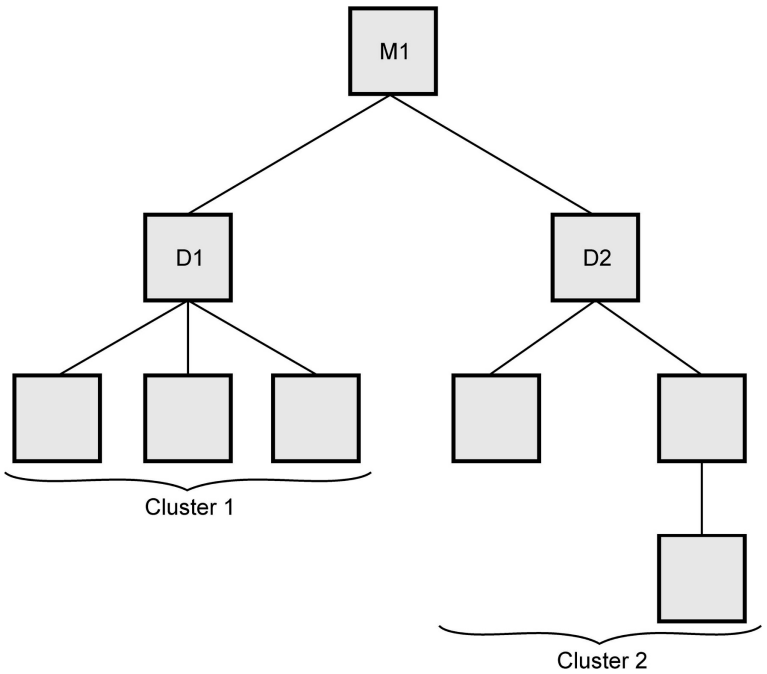


Fig. 6.6.7 Bottom up integration testing

First components are collected together to form cluster 1 and cluster 2. Then each cluster is tested using a driver program. The clusters subordinate the driver module. After testing the driver is removed and clusters are directly interfaced to the modules.

6.6.2.3 Regression Testing

- Regression testing is used to check for defects propagated to other modules by changes made to existing program. Thus regression testing is used to reduce the side effects of the changes.
- There are three different classes of test cases involved in regression testing -
  - Representative sample of existing test cases is used to exercise all software functions.
  - Additional test cases focusing software functions likely to be affected by the change.
  - Tests cases that focus on the changed software components.
- After product had been deployed, regression testing would be necessary because after a change has been made to the product an error that can be discovered and it should be corrected. Similarly for deployed product addition of new feature may be requested and implemented. For that reason regression testing is essential.

6.6.2.4 Smoke Testing

- The smoke testing is a kind of integration testing technique used for time critical projects wherein the project needs to be assessed on frequent basis.
- Following activities need to be carried out in smoke testing -
  1. Software components already translated into code are integrated into a “build”. The “build” can be data files, libraries, reusable modules or program components.
  2. A series of tests are designed to expose errors from build so that the “build” performs its functioning correctly.
  3. The “build” is integrated with the other builds and the entire product is smoke tested daily.

Smoke testing benefits

1. Integration risk is minimized.
2. The quality of the end product is improved.
3. Error diagnosis and correction are simplified.
4. Assessment of progress is easy.

6.6.3 System Testing

The system test is a series of tests conducted to fully the computer based system.

Various types of system tests are :

1. **Recovery testing**
2. **Security testing**
3. **Stress testing**
4. **Performance testing**

The main focus of such testing is to test

- System functions and performance.
- System reliability and recoverability (recovery test).
- System installation (installation test).
- System behaviour in the special conditions (stress test).
- System user operations (acceptance test/alpha test).
- Hardware and software integration and collaboration.
- Integration of external software and the system.

6.6.3.1 Recovery Testing

- Recovery testing is intended to check the system's ability to recover from failures.
- In this type of testing the software is forced to fail and then it is verified whether the system recovers properly or not.
- For automated recovery then reinitialization, checkpoint mechanisms, data recovery and restart are verified.

6.6.3.2 Security Testing

- Security testing verifies that system protection mechanism prevent improper penetration or data alteration.
- It also verifies that protection mechanisms built into the system prevent intrusion such as unauthorized internal or external access or willful damage.
- System design goal is to make the penetration attempt more costly than the value of the information that will be obtained.

6.6.3.3 Stress Testing

- Determines breakpoint of a system to establish maximum service level.
- In stress testing the system is executed in a manner that demands resources in abnormal quantity, frequency or volume.
- A variation of stress testing is a technique called sensitivity testing.
- The sensitive testing is a testing in which it is tried to uncover data from a large class of valid data that may cause instability or improper processing.

6.6.3.4 Performance Testing

- Performance testing evaluates the run time performance of the software, especially real time software.
- In performance testing resource utilization such as CPU load, throughput, response time, memory usage can be measured.
- For big systems (e.g. banking systems) involving many users connecting to servers (e.g. using internet) performance testing is very difficult.
- Beta testing is useful for performance testing.

Review Questions

1. What is system testing ? Explain any two system testing strategies. **SPPU : May-11, Marks 8**
2. Explain the approaches in integration testing. **SPPU : Dec.-11, Marks 8**
3. Differentiate between alpha and beta testing. **SPPU : Dec.-11, Marks 4**
4. What is unit testing ? Explain the unit testing process. **SPPU : Dec.-12, Marks 8**
5. Explain any two systems testing methods. **SPPU : Dec.-12, Marks 6**
6. What do you mean by acceptance testing ? **SPPU : May-13, Marks 4**
7. What is the need of stubs and drivers in software testing ? **SPPU : Dec.-13, Marks 4**
8. Explain the integration testing approaches. **SPPU : Dec.-13, Marks 8**
9. Explain in detail :

   1) Top-down integration testing 2) Bottom-up testing. **SPPU : May-14, Marks 8**
10. Explain following testing types **SPPU : May-14, Marks 8**

    1) Validation testing 2) Acceptance testing 3) Smoke testing process.
11. What do you understand by system testing ? What are the different kinds of system testing that are usually performed on large software testing. **SPPU : Dec.-17, End Sem, Marks 8**
12. What is software testing ? Explain the software testing strategies for software development. **SPPU : May-18, End Sem, Marks 7**
13. What is the need of stubs and drivers in software testing ? **SPPU : Dec.-19, End Sem, Marks 5**
14. What do you understand by integration testing ? Explain the objectives of integration testing **SPPU : May-16, Dec.-19, End Sem, Marks 6**

6.7 Testing Strategies for Object Oriented Software

SPPU : May-12, 13, Marks 6

- The primary objective of testing for object oriented software is to **uncover as much errors as** possible with manageable amount of efforts in realistic time span.
- The object oriented testing strategy is identical to conventional testing strategy. The strategy is start **testing in small** and work outward to **testing in large**. The basic unit of testing is **class** that contains attributes and operations. These classes **integrate** to form object oriented architecture. These are called **collaborating classes.**

6.7.1 Unit Testing in OO Context

- **Class** is an encapsulation of data attributes and corresponding set of operations.
- The **object** is an instance of a class. Hence objects also specify some data attributes and operations.
- In object oriented software the focus of unit testing is considered as classes or objects.
- However, operations within the classes are also the testable units. In fact in OO context the operation can not be tested as single isolated unit because one operation can be defined in one particular class and can be used in multiple classes at the same time. For example consider the operation **display()**. This operation is defined as super class an at the same time it can be used by the can be multiple derived classes. Hence it is not possible to test the operation as single module.
- Thus class testing for OO software is equivalent to unit testing for conventional software. In conventional software system, the algorithmic details and data that flow are units of testing but in OO software the encapsulated classes and operations (that are encapsulated within the class and state behavior of class) are the unit of testing.

6.7.2 Integration Testing in OO Context

- There are two strategies used for integration testing and those are -
  1. **Thread based testing** 2. **Use-based testing.**
- The **thread based testing** integrates all the classes that respond to one input or event for the system. Each thread is then tested individually.
- In **use-based testing** the independent classes and dependent classes are tested. The **independent classes** are those classes that uses the server classes and **dependant classes** are those classes that use the independent classes. In use based testing the independent classes are tested at the beginning and the testing proceeds towards testing of dependent classes.

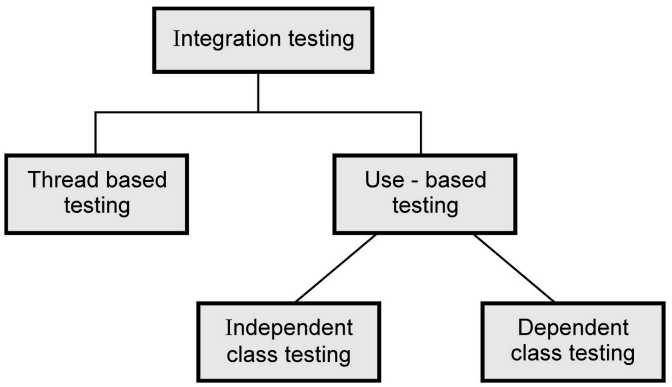


Fig. 6.7.1 Integration testing in OO context

- **Drivers and stub :** In OO context the driver can be used to test operations at lowest level

and for testing whole group of classes. The stub can be used in the situations in which collaborations of the classes is required and one collaborating class is not fully implemented.

- The **cluster testing** is one step in integration testing in OO context. In this step all the collaborating classes are tested in order to uncover the errors.

6.7.3 Difference between OO Testing Strategy and Conventional Testing Strategy

- The testing strategies applied to conventional software are based on **unit testing**, **integration testing**, **validation testing** and **system testing**. But in object oriented testing, the unit testing loses some of its meaning and integration testing changes significantly.
- The object oriented testing strategy is also based on the principle - "test in small" and then work "in the large".
  1. **Testing in the small** involves **class attributes and operations;** the **main focus** is on communication and collaboration within the **class**.
  2. **Testing in the large** involves a series of regression tests to uncover errors due to **communication and collaboration among classes**

Finally, the system as a whole is tested to detect errors in fulfilling requirements

Review Questions

1. Explain object oriented testing. **SPPU : May-12, Marks 3**
2. How object oriented testing differs from conventional testing strategies ? **SPPU : May-13, Marks 6**

6.8 Test Strategies for Web Apps

Testing strategy suggests to use following basic **principles of software testing** -

1. The **content model** must be reviewed in order to **uncover the errors**.
2. The **interfaces model** is reviewed to ensure all the **use cases**.
3. The **design model** is reviewed to uncover **navigation errors**.
4. **User interface** must be tested to remove the **navigation errors**.
5. For selected function components unit testing is done.
6. Navigation must be tested for the complete web architecture.
7. The web application is tested in different environmental configuration.
8. **Security tests** must be conducted to exploit **vulnerabilities** of the web application.

9. **Performance** of the web application must be tested using **performance testing**.
10. Finally controlled and monitored group of users must test the web application.

The web application **evolves continuously** and hence it is an on-going activity.

6.9 Validation Testing

SPPU : May-19, Dec.-19, Marks 4

- The integrated software is tested based on requirements to ensure that the desired product is obtained.
- In validation testing the main focus is to uncover errors in
  - System input/output
  - System functions and information data
  - System interfaces with external parts
  - User interfaces
  - System behaviour and performance
- Software validation can be performed through a series of **black box tests**.
- After performing the validation tests there exists two conditions.
  1. The function or performance characteristics are **according** to the **specifications** and are accepted.
  2. The requirement specifications are derived and the deficiency list is created. The **deficiencies** then can be **resolved** by establishing the proper communication with the customer.
- Finally in validation testing a review is taken to ensure that all the elements of software configuration are developed as per requirements. This review is called configuration review or audit.

6.9.1 Acceptance Testing

The acceptance testing is a kind of testing conducted to ensure that the software works correctly in the **user work environment**.

The acceptance testing can be conducted over a period of weeks or months.

The types of acceptance testing are :

1. **Alpha test** - The alpha testing is a testing in which the version of complete software is tested by the customer under the supervision of developer. This testing is performed at developer's site. The software is used in natural setting in presence of developer. This test is conducted in controlled environment.

2. **Beta test** - The beta testing is a testing in which the version of software is tested by the customer without the developer being present. This testing is performed at customer's site. As there is no presence of developer during testing, it is not controlled by developer. The end user records the problems and report them to developer. The developer then makes appropriate modification.

Difference between Alpha Testing and Beta Testing

Sr. No.	Alpha testing	Beta testing
1	Performed at developer's site	Performed at end user's site
2	Performed in controlled environment as developer is present	Performed in uncontrolled environment as developer is not present
3	Less probability of finding of error as it is driven by developers.	High probability of finding of error as end user can use it the way he wants.
4	It is not considered as live application	It is considered as live application
5	It is done during implementation phase of software	It is done as pre-release of software
6	Less time consuming as developer can make necessary changes in given time.	More time consuming. As user has to report bugs if any via appropriate channel.

6.9.2 Validation Test Criteria

- Validation is a technique, to evaluate whether the final built software product fulfils the customer requirements.
- A test plan is created that contains the **classes of tests** to be conducted and **test procedure** defines the specific tests to be conducted.
- After each validation testing one of the two possible conditions exists:
  1) The function or performance characteristics conforms to specification and is accepted. or
  2) A deviation from specification is represented. In other words the list of requirements that does not get fulfilled by the system is represented.

6.9.3 Configuration Review

- Configuration review is an important element of the validation process.
- The main purpose of this review is to ensure that all the elements of the software configuration have been properly developed and catalogued.
- The configuration review is also called as **audit.**

Review Question

1. Differentiate between alpha testing and beta testing.

SPPU : May-19, Dec.-19, End Sem,Marks 4

6.10 Types of Testing

There are two general approaches for the software testing.

1. Black box testing

The black box testing is used to demonstrate that the software functions are operational. As the name suggests in black box testing it is tested whether the input is accepted properly and output is correctly produced.

The major focus of black box testing is on functions, operations, external interfaces, external data and information.

2. White box testing

In white box testing the procedural details are closely examined. In this testing the internals of software are tested to make sure that they operate according to specifications and designs. Thus major focus of white box testing is on internal structures, logic paths, control flows, data flows, internal data structures, conditions, loops, etc.

6.11 White Box Testing

SPPU : Dec.-11,12, 16, May-11,12,13,14,16, Marks 8

6.11.1 Basis Path Testing

In this method the procedural design using basis set of execution path is tested. This basis set ensures that every execution path will be tested at least once.

6.11.1.1 Flow Graph Notation

Path testing is a structural testing strategy. This method is intended to exercise every independent execution path of a program atleast once.

Following are the steps that are carried out while performing path testing.

Step 1 : Design the flow graph for the program or a component.

Step 2 : Calculate the cyclomatic complexity.

Step 3 : Select a basis set of path.

Step 4 : Generate test cases for these paths.

Let us discuss each in detail.

Step 1 : Design the flow graph for the program or a component.

Flow graph is a graphical representation of logical control flow of the program. Such a graph consists of circle called a ***flow graph node*** which basically represents one or more procedural statements and arrow called as ***edges*** or ***links*** which basically represent control flow. In this flow graph the areas bounded by nodes and edges are called ***regions***. Various notations used in flow graph are (See Fig. 6.11.1 on next page) –

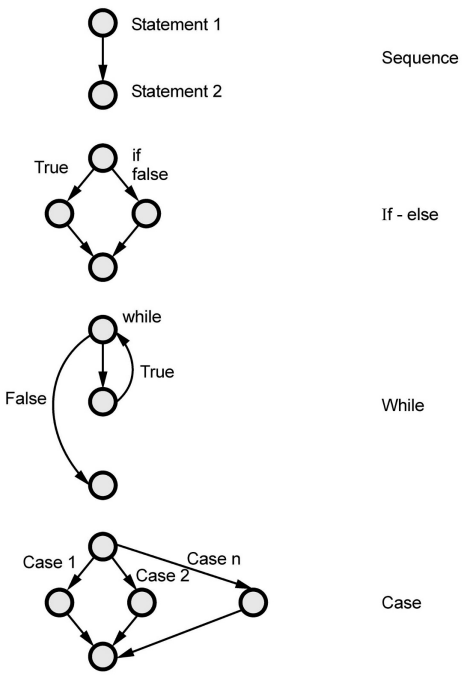


Fig. 6.11.1

For example : Following program is for searching a number using binary search method. Draw a flow graph for the same.

```
void search (int key, int n, int a [ ])
    {
        int mid;
1)      int bottom = 0;
2)      int top = n – 1;
3)      while (bottom <=top)
4)      { mid = (top + bottom) / 2;
5)      if (a [mid] = = key)
        {
6)      printf ("Element is present");
7)      return;
        } // end of if
        else
        {
8)      if (a [mid] < key)
9)      bottom = mid + 1;
        else
10)     top = mid – 1;
        } // end of else
        } // end of while
11)     } // end of search
```

The flow graph will be

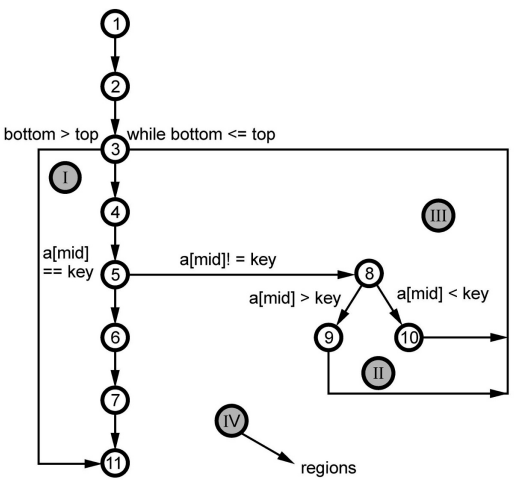


Fig. 6.11.2

Step 2 : Calculate the cyclomatic complexity.

The cyclomatic complexity can be computed by three ways.

1) Cyclomatic complexity = Total number of regions in the flow graph = **4** (note that in above flow graph regions are given by shaded roman letters).
2) Cyclomatic complexity = E – N + 2 = 13 edges – 11 nodes + 2
   = 2 + 2 = **4**
3) Cyclomatic complexity = P + 1 = 3 + 1 = **4**. There are 3 predicate (decision making) nodes : Nodes 3, 5 and 8.

Step 3 : Select a basis set of path

The basis paths are

Path 1 : 1, 2, 3, 4, 5, 6, 7, 11

Path 2 : 1, 2, 3, 11

Path 3 : 1, 2, 3, 4, 5, 8, 9, 3 …

Path 4 : 1, 2, 3, 4, 5, 8, 10, 3 …

Step 4 : Generate test cases for these paths.

After computing cyclomatic complexity and finding independent basis paths, the test cases has to be executed for these paths. The format for test case is -

Preconditions :

Test case id	Test case name	Test case description	Test steps			Test case status (Pass /Fail)	Test priority	Defect severity
			Step	Expected	Actual			
1								
2								
n								

The test case for binary search can be written as -

Precondition : There should be list of elements arranged in ascending order. The element to be searched from the list, its value should be entered and will be stored in variable *'key'*.

Test case id	Test case name	Test case description	Test steps			Test case status	Test status (P / F)	Test priority	Defect severity
			Step	**Expected**	**Actual**				
1.	Validating the list boundary	Checking the bottom and top values for the list of elements	Set bottom = 0 top = n – 1 check if bottom < = top by while loop. This condition defines the length of the list from which the key is searched.	Initially bottom < = top will be true. But during iterations list's length will be reduced and if entire list gets scanned at one point (bottom > = top) will be reached then return to main.		Design			
2.	Checking list element with key.	Checking if middle element of array is equal to key value	Set mid = (top + bottom) / 2 Then compare if a[mid] is equal to key.	If a[mid] = key value then print message "Element is present" and return to main.		Design			
		if a[mid] is < key value if a[mid] is > key value	Set bottom = mid + 1 and then goto "while" set top = mid +1 and go back to "while" loop	The right sublist will be searched. The left sublist will be searched for key element		Design			

Example 6.11.1 *Draw the flow graph for finding maximum of three numbers and derive the testcases using cyclomatic complexity.* **SPPU : Dec.-11, Marks 8**

Solution :

The flow graph for given program is - (From Fig. 6.11.3)

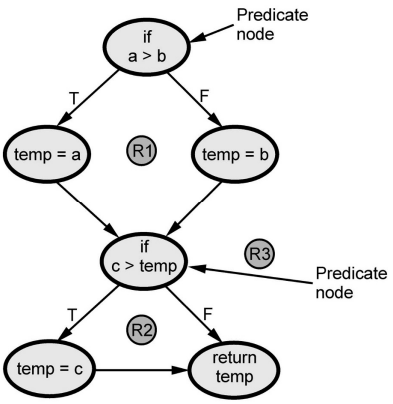


Fig. 6.11.3 Flow graph for finding maximum of three numbers

Cyclomatic complexity = E – N + 2 = 7 – 6 + 2 = 3

Cyclomatic complexity = P + 1 = 2 + 1 = 3

Cyclomatic complexity = Regions encountered = 3

Hence cyclomatic complexity of given program is **3.**

6.11.1.2 Graph Matrices

Definition : Graph matrix is a square matrix whose size is equal to number of nodes of the flow graph.

For example consider a flow graph -

The graph matrix will be

For computing the cyclomatic complexity. Following steps are adopted -

Step 1 : Create a graph matrix. Mark the corresponding entry as 1 if node A is connected to node B.

Step 2 : Count total number of 1's from each row and subtract 1 from each corresponding row.

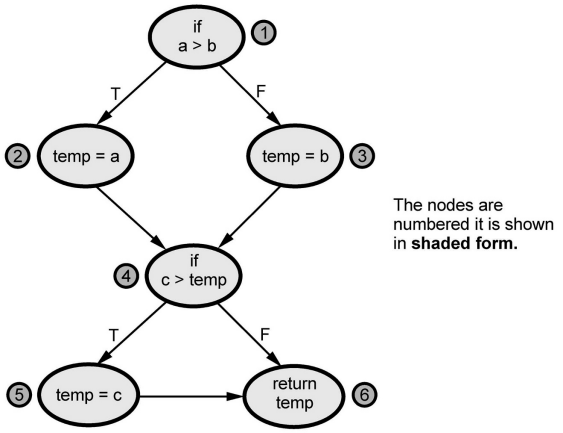


Fig. 6.11.4 Flow graph

Step 3 : Add cyclometic complexity.

	1	2	3	4	5	6	
1		1	1				2 – 1 = 1
2				1			1 – 1 = 0
3				1			1 – 1 = 0
4					1	1	2 – 1 = 1
5						1	1 – 1 = 0
6							2 + 1 = **3**

the results of each row and add 1 to it. The resultant value is the cyclomatic complexity.

How to extend graph matrix for use in testing ?

Following properties indicate how to extend graph matrix for use in testing.

i) The number of times of link between the nodes get executed.

ii) The processing time spent in traversal of a link.

iii) The number of resources required.

iv) The amount of memory required to traverse the link.

6.11.2 Control Structure Testing

- The structural testing is sometime called as white box testing.
- In structural testing derivation of test cases is according to program structure. Hence knowledge of the program is used to identify additional test cases.
- Objective of structural testing is to exercise all program statements.

6.11.2.1 Condition Testing

- To test the logical conditions in the program module the condition testing is used. This condition can be a Boolean condition or a relational expression.
- The condition is incorrect in following situations.
  i) Boolean operator is incorrect, missing or extra.
  ii) Boolean variable is incorrect.
  iii) Boolean parenthesis may be missing, incorrect or extra.
  iv) Error in relational operator.
  v) Error in arithmetic expression.
- The *condition testing* focuses on each testing condition in the program.
- The *branch testing* is a condition testing strategy in which for a compound condition each and every true or false branches are tested.
- The *domain testing* is a testing strategy in which relational expression can be tested using three or four tests.

6.11.2.2 Loop Testing

Loop testing is a white box testing technique which is used to test the loop constructs. Basically there are four types of loops.

1. Simple loops :

The tests can be performed for n number of classes.

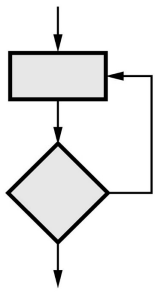


Fig. 6.11.5 Simple loop

where

i) n = 0 that means skip the loop completely.
ii) n = 1 that means one pass through the loop is tested.
iii) n = 2 that means two passes through the loop is tested.

iv) n = m that means testing is done when there are m passes where m < n.

v) Perform the testing when number of passes are n – 1, n, n + 1.

2. Nested loops :

The nested loop can be tested as follows.

i) Testing begins from the innermost loop first. At the same time set all the other loops to minimum values.

ii) The simple loop test for innermost loop is done.

iii) Conduct the loop testing for the next loop by keeping the outer loops at minimum values and other nested loops at some specified value.

iv) This testing process is continued until all the loops have been tested.

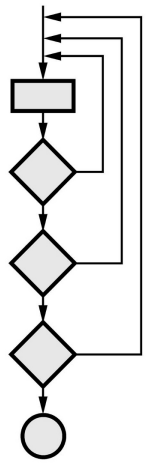


Fig. 6.11.6 Nested loops

3. Concatenated loops :

The concatenated loops can be tested in the same manner as simple loop tests. (Refer Fig. 6.11.7)

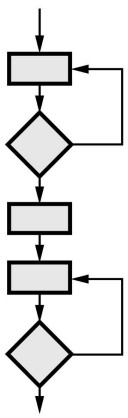


Fig. 6.11.7 Concatenated loops

4. Unstructured loops :

The testing cannot be effectively conducted for unstructured loops. Hence these types of loops needs to be redesigned. (Refer Fig. 6.11.8)

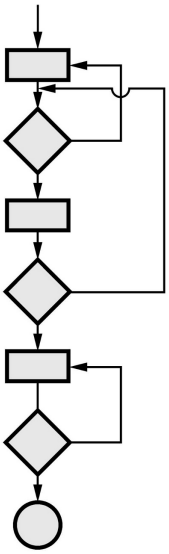


Fig. 6.11.8 Unstructured loops

Review Questions

1. Explain the following : i) Condition testing ii) Loop testing. **SPPU : May-11, Marks 8**
2. What is basis path testing ? What is cyclomatic complexity ? How is it determined for a flow graph ? Illustrate with an example. **SPPU : May-12, Marks 8**
3. Basis path testing covers all statements in a program module. Justify with example. **SPPU : Dec.-12, Marks 8**
4. Explain the graph matrix and loop testing methods. **SPPU : May-13, Marks 8**
5. What is cyclomatic complexity ? How is it determined for a flow graph ? Illustrate with example. **SPPU : May-14, Marks 8**
6. What do you mean by white box testing ? **SPPU : Dec.-16, End Sem, Marks 7**

6.12 Black Box Testing

SPPU : Dec.-11, Marks 6

- The black box testing is also called as **behavioural testing**.
- Black box testing methods focus on the functional requirements of the software. Test sets are derived that fully exercise all functional requirements.
- The black box testing is not an alternative to white box testing and it uncovers different class of errors than white box testing.

Why to perform black box testing ?

Black box testing uncovers following types of errors.

1. Incorrect or missing functions
2. Interface errors
3. Errors in data structures
4. Performance errors
5. Initialization or termination errors

6.12.1 Equivalence Partitioning

- It is a black box technique that divides the input domain into **classes of data**. From this data test cases can be derived.
- An ideal test case uncovers a class of errors that might require many arbitrary test cases to be executed before a general error is observed.
- In equivalence partitioning the equivalence classes are evaluated for given input condition. Equivalence class represents a set of valid or invalid states for input conditions.
- Equivalence class guidelines can be as given below :
  - If input condition specifies a range, one valid and two invalid equivalence classes are defined.

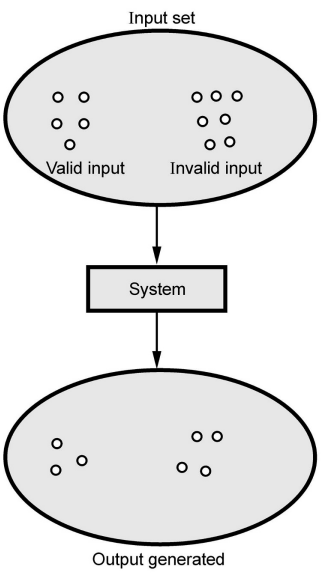


Fig. 6.12.1

- If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
- If an input condition specifies a member of a set, one valid and one invalid equivalence class is defined.
- If an input condition is Boolean, one valid and one invalid equivalence class is defined.

For example :

Area code : Input condition, Boolean - The area code may or may not be present.

Input condition, range - Value defined between 200 and 700.

Password : Input condition, Boolean - A password may or may not be present.

Input condition, value - Seven character string.

Command : Input condition, set - Containing commands noted before.

6.12.2 Boundary Value Analysis (BVA)

- Boundary value analysis is done to check **boundary conditions**.
- A boundary value analysis is a testing technique in which the elements at the edge of the domain are selected and tested.
- Using boundary value analysis, instead of focusing on input conditions only, the test cases from output domain are also derived.
- Boundary value analysis is a test case design technique that complements equivalence partitioning technique.
- Guidelines for boundary value analysis technique are
  1. If the input condition specified the range bounded by values x and y, then test cases should be designed with values x and y. Also test cases should be with the values above and below x and y.
  2. If input condition specifies the number of values then the test cases should be designed with minimum and maximum values as well as with the values that are just above and below the maximum and minimum should be tested.
  3. If the output condition specified the range bounded by values x and y, then test cases should be designed with values x and y. Also test cases should be with the values above and below x and y.
  4. If output condition specifies the number of values then the test cases should be designed with minimum and maximum values as well as with the values that are just above and below the maximum and minimum should be tested.

5. If the internal program data structures specify such boundaries then the test cases must be designed such that the values at the boundaries of data structure can be tested.

For example :

Integer D with input condition [– 2, 10],

Test values : – 2, 10, 11, – 1, 0

If input condition specifies a number values, test cases should developed to exercise the minimum and maximum numbers. Values just above and below this min and max should be tested.

Enumerate data E with input condition : {2, 7, 100, 102}

Test values : 2, 102, – 1, 200, 7

6.12.3 Graph based Testing

- In the graph based testing, a graph of objects present in the system is created.
- The graph is basically a collection of nodes and links. Each node represents the object that is participating in the software system and links represent the relationship among these objects.
- The node weight represents the properties of object and link weight represents the properties or characteristics of the relationship of the objects.
- After creating the graph, important objects and their relationships are tested.]

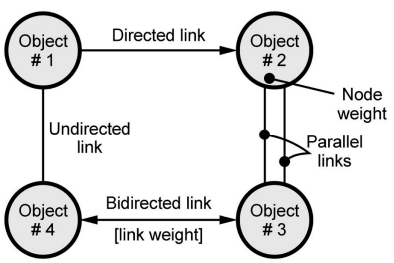


Fig. 6.12.2 Graph notations

6.12.4 Orthogonal Array Testing

- There are many applications for which very **small number of input** is needed and values that each input requires might be **bounded**.
- In such a situation the number of **test cases** is relatively **small** and can be **manageable**. But if the number of input gets increased then it will increase there will be large number of test cases. And testing may become impractical or impossible.

- **Orthogonal array testing** is a kind of testing method which can be applied to the applications in which **input domain is relatively small** but there could be large number of test cases.
- Using orthogonal array testing method, only **faulty regions** can be tested and thus the number of test cases can be reduced.
- Orthogonal arrays are **two dimensional arrays** of numbers which possess the interesting quality that by choosing any two columns in the array you receive an even distribution of all the pair-wise combinations of values in the array.
- Following are some important terminologies used in orthogonal testing methods -

Runs

It denotes the number of rows in the array. These can be directly translated to the test cases.

Factors

It denotes the number of columns in the array. These can be directly translated to maximum number of variables that can be handled by the array.

Level

This number denotes the maximum number of values that a single factor(column) can take.

L9 orthogonal array

This array is used to generate the test cases. This array has a **balancing** property. That means the testing can be done uniformly by executing the test cases generated by the L9 orthogonal array.

Example

Consider that we want to develop an application which has three sections - top, bottom and middle.

- Associated with each section we will consider one variable. That means now we have to analyse only three variables.
- These variables can be assigned with Boolean values and hence the values can be true or false.
- If we decide to test it completely then there would be $2^3 = 8$ test cases
- By orthogonal array testing method, mapping the values to L9 orthogonal array would be -

	factor1	factor2	factor3
Run1	false	false	false
Run2	true	false	false
Run3	false	true	false
Run4	false	false	true

Variable from 1st section is true

Variable from second section is true

Variable from third section is true

- Thus the values are left to all the levels.
- Now the test cases would be
  1. Display the application without any section.
  2. Display the application with top section only.
  3. Display the application with the middle section only.
  4. Display the application with the bottom section only.

Thus there are only four test cases in all and no need to test all the combinations. In fact we have considered this test cases by considering the single parameter. This kind of testing is called **single mode fault detection**. By considering these test case the faults can be fixed.

Double mode fault detection - In this method, two parameters can be considered at a time for determining the number of test cases.

Advantage

The orthogonal array testing approach enable the developer to provide good test coverage with very few test cases.

Review Question

1. Explain equivalent partioning testing. **SPPU : Dec.-11, Marks 6**

6.13 Comparison between Black Box and White Box Testing

SPPU : Dec.-12, Marks 4

Sr. No.	Black box testing	White box testing
1.	Black box testing is called behavioural testing.	White box testing is called glass box testing.
2.	Black box testing examines some fundamental aspect of the system with little regard for internal logical structure of the software.	In white box testing the procedural details, all the logical paths, all the internal data structures are closely examined.

3.	During black box testing the program cannot be tested 100 percent.	White box testing lead to test the program thoroughly.
4.	This type of testing is suitable for large projects.	This type of testing is suitable for small projects.

Advantages and Disadvantages of Black box Testing

Advantages :

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

Disadvantages :

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

Advantages and Disadvantages of White box Testing

Advantages :

1. Each procedure can be tested thoroughly. The internal structures, data flows, logical paths, conditions and loops can be tested in detail.
2. It helps in optimizing the code.
3. White box testing can be easily automated.
4. Due to knowledge of internal coding structure it is easy to find out which type of input data can help in testing the application efficiently.

Disadvantages :

1. The knowledge of internal structure and coding is desired for the tested. Thus the skilled tester is required for whitebox testing. Due to this the testing cost is increased.

2. Sometimes it is difficult to test each and every path of the software and hence many paths may go untested.
3. Maintaining the white box testing is very difficult because it may use specialized tools like code analyzer and debugging tools are required.
4. The missing functionality can not be identified.

Review Question

1. Differentiate between white box and black box testing. **SPPU : Dec.-12, Marks 4**

6.14 Multiple Choice Questions

Q.1 Which one of the following term describes testing ?

a Evaluating deliverable to find errors

b A stage of all projects

c Finding broken code

d None of the these

Q.2 Testing of individual components by the developers are comes under _____ .

a integration testing b validation testing

c unit testing d none of mentioned

Q.3 Unit testing is done by _____.

a users b developers

c customers d none of the mentioned

Q.4 The testing have been stopped when _____ .

a the faults have been fixed b all the tests run

c the time completed d the risk are resolved

Q.5 Software mistakes during coding are known as _____ .

a errors b bugs

c failures d defects

Q.6 Test cases are designed during which of the following stages ?

a Test recording b Test configuration

c Test planning d Test specification

Q.7 Which is not the other name for structural testing ?

a Behavioral testing b Glass box testing

c White box testing d None of the above

Q.8 Which is not a type of incremental testing approach ?

a Bottom up b Top down

c Big-bang d Functional incrimination

Q.9 Which one is the main focus of acceptance testing ?

a Testing The System With Other Systems

b Testing For A Business Perspective

c Finding Faults In The System

d Testing the system with other systems

Q.10 Testing of software with actual data and in actual environment is known as _____.

a regression testing b beta testing

c alpha testing d none of the above

Q.11 Beta Testing is done at _____.

a developer's end b user's end

c user's & developer's end d none of the mentioned

Q.12 Black box testing is only functional testing.

a True b False

Answer Keys for Multiple Choice Questions :

Q.1	a	**Q.2**	c	**Q.3**	b	**Q.4**	d
Q.5	b	**Q.6**	d	**Q.7**	a	**Q.8**	c
Q.9	b	**Q.10**	b	**Q.11**	b	**Q.12**	b

❑❑❑

Notes

SOLVED MODEL QUESTION PAPER (In Sem)
Software Engineering

S.E.(Computer) Semester - IV (As per 2019 Pattern)

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

Instructions :

1. **Attempt Q.1 or Q.2, Q.3 or Q.4.**
2. **Make suitable assumptions wherever necessary.**
3. **Figures to the right indicate full marks.**
4. **Neat diagrams must be drawn wherever necessary.**

Q.1 **(a)** *What is software engineering ? What are the characteristics of software ?* **(Refer section 1.5)** **[4]**

(b) *What is meant by 'blocking states' in linear sequential model ?* **(Refer example 1.13.2)** **[3]**

(c) *Explain with neat diagram incremental model and state its advantages and disadvantages.* **(Refer section 1.14.2)** **[8]**

OR

Q.2 **(a)** *Elaborate how software engineering is a layered technology.* **(Refer section 1.7)** **[6]**

(b) *Explain Feature Driven Development (FDD).* **(Refer section 1.20.4)** **[4]**

(c) *What is agility ? Explain about process model.* **(Refer section 1.19)** **[5]**

Q.3 **(a)** *What is requirement ? What are the types of requirements ?* **(Refer section 2.1)** **[3]**

(b) *Write an use case for 'login' with a template and diagram.* **(Refer example 2.5.2)** **[4]**

(c) *What are requirements engineering tasks ? Explain in detail.* **(Refer section 2.2)** **[8]**

OR

Q.4 **(a)** *Explain the tasks done during elicitation and requirement management.* **(Refer section 2.4)** **[5]**

(b) *Explain the concept of traceability matrix.* **(Refer section 2.15)** **[3]**

(c) *Create the swimlane diagram for, monitoring of sensor in a 'Safehome security system' from control panel.* **(Refer example 2.8.5)** **[7]**

SOLVED MODEL QUESTION PAPER (End Sem)
Software Engineering

S.E.(Computer) Semester - IV (As per 2019 Pattern)

Time : 2 $\frac{1}{2}$ Hours] **[Maximum Marks : 70**

Instructions :

1. **Attempt Q.1 or Q.2, Q.3 or Q.4, Q.5 or Q.6, Q.7 or Q.8.**
2. **Make suitable assumptions wherever necessary.**
3. **Figures to the right indicate full marks.**
4. **Neat diagrams must be drawn wherever necessary.**

Q.1 **(a)** *What is the need for defining a software scope ? What are the categories of software engineering resources.* **(Refer section 3.2)** **[8]**

(b) *Explain the FP based estimation technique.* **(Refer section 3.5)** **[10]**

OR

Q.2 **(a)** *Explain COCOMO model for project estimation with suitable example.* **(Refer section 3.6)** **[10]**

(b) *What is project scheduling ? What are the basic principles of project scheduling.* **(Refer section 3.8)** **[8]**

Q.3 **(a)** *Explain design within the context of software engineering.* **(Refer section 4.1.1)** **[5]**

(b) *Explain different design concepts.* **(Refer section 4.4)** **[12]**

OR

Q.4 **(a)** *What are the deployment level design elements ?* **(Refer section 4.7)** **[3]**

(b) *Explain data centered and layered architectures with neat diagrams.* **(Refer section 4.13)** **[8]**

(c) *Explain the types of design classes.* **(Refer section 4.6)** **[6]**

Q.5 **(a)** *Write short note on - RMMM.* **(Refer section 5.7)** **[4]**

(b) *Explain the risk identification and assessment process for a software project.* **(Refer section 5.4)** **[8]**

(c) *What are the types of risks ? Explain in brief.* **(Refer section 5.2)** **[6]**

OR

Q.6 **(a)** *What do you understand by Software Configuration Management (SCM) ? Discuss the importance of SCM.* **(Refer section 5.9)** **[8]**

(b) *Explain change control mechanism in SCM.* **(Refer section 5.11)** **[6]**

(c) *How risk projection is carried out using risk table ?* **(Refer section 5.5)** **[4]**

Q.7 **(a)** *What is importance of testing practices ? What are the principles of testing practices.* **(Refer section 6.1)** **[6]**

(b) *Explain in detail : 1) Top-down integration testing 2) Bottom-up testing.* **(Refer section 6.6)** **[6]**

(c) *Differentiate between verification and validation.* **(Refer section 6.2)** **[5]**

OR

Q.8 **(a)** *Differentiate between alpha testing and beta testing.* **(Refer section 6.9)** **[6]**

(b) *Explain testing strategies for object oriented software.* **(Refer section 6.7)** **[8]**

(c) *What is entry and exit criteria for completion of testing.* **(Refer section 6.4)** **[3]**

❑❑❑

Notes