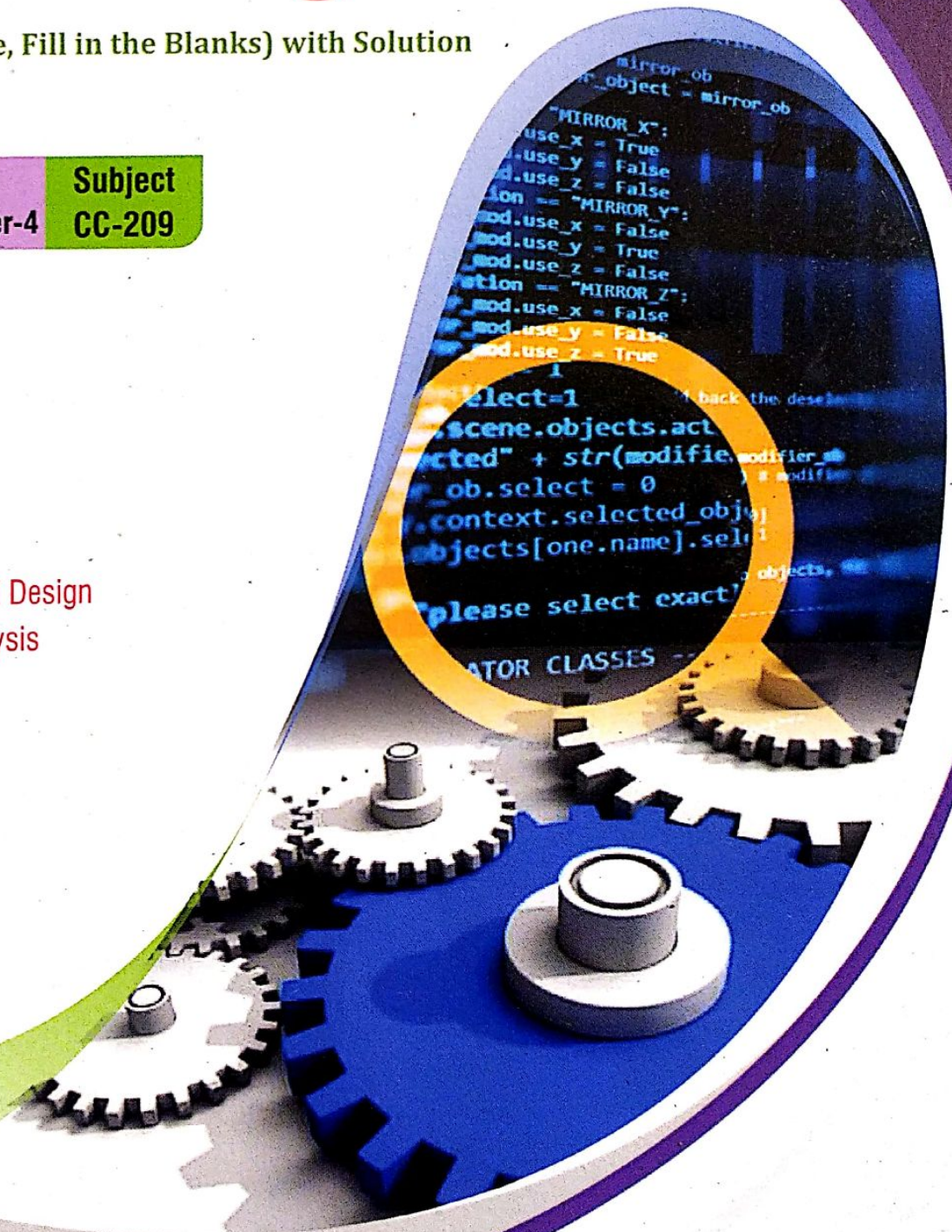
System Analysis, QA and Testing

Exercise (MCQ, True-False, Fill in the Blanks) with Solution

BCA Semester-4	Subject CC-209



- System Analysis And Design
- Object Oriented Analysis
- UML Diagrams
- QA and Testing

COMPUTER WORLD®
PUBLICATION
A Division of Live Education System Pvt. Ltd.

BCA Semester-4

System Analysis, QA and Testing

Subject Code : CC-209

Prof. Parth Parekh
M.E., B.Tech & Diploma Computer Engineering
Institute of Computer Technology, Ganpat University.

Prof. Sejal Vaghela
MCA (Master in Computer Applications)
Lokmanya College of Computer Applications,
Satellite, Ahmedabad

Prof. Aafrin Julaya
MSc. [Computer Application &
Information Technology]
Asia Pacific Institute of Management,
Usmanpura, Ahmedabad.

COMPUTER WORLD®
PUBLICATION
A Division of Live Education System Pvt. Ltd.

COMPUTER WORLD
43, 5th Floor, SANIDHYA Complex, Nr. M. J. Library,
Opp. Sanyas Ashram, Ashram Road, Ahmedabad-09.
Ph. : +91- 79 26580723, 26580823 | Mobile : 9724011150, 9725022917
URL : www.computerworld.ind.in | e-Mail :info@computerworld.ind.in

Publish By :

COMPUTER WORLD

A Division of Live Education System Pvt. Ltd.

An ISO 9001 : 2008 Certified Company

Kalpesh Patel - kalpesh@computerworld.ind.in

Prepare by : Computer World Research Department

Prof. Parth Parekh, Prof. Sejal Vaghela

Prof. Aafrin Julaya, Ravindra Parmar

Price : ₹ 80/-

ISBN : 978-93-88092-21-0

Book Code : CEBCA115

Edition : 1st

Notice of Rights

No part of this publication may be reproduced, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Computer World except under the terms of a Computer World license agreement.

Notice of Liability

The information in this courseware title is distributed on an 'as is' basis, without warranty. While very precaution has been taken in the preparation of this course, neither the authors nor Computer World shall have any liability to my person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the instruction contained in this book or by the computer software and hardware products described in it.

Disclaimer

We make a sincere effort to ensure the accuracy of the material described here in however, Computer World makes no warranty, expressed or implied, with respect of the quality, correctness, reliability, accuracy, of freedom from error of this document or the products it describes. Data used as examples and sample data files are intended to be fictional. Any resemblance to real persons or companies is entirely coincidental.

Content of Copyright : COMPUTER WORLD PUBLICATION - http://www.computerworld.ind.in

Contact : 43, 5th Floor, Sanidhya Complex, Opp. Capital Comm. Center, Ashram Road, Ahmedabad-380009

Phone : +91-79-26580823 / +91-79-26580723, | Mobile : 97240 11150, 97250 22917

URL : www.computerworld.ind.in | e-Mail : info@computerworld.ind.in, info.computerworld@gmail.com

INDEX

Unit-1 System Analysis and Design 05 to 24

1.1 Software Development Models
1.1.1 Waterfall Model
1.1.2 Incremental Model
1.1.3 Spiral Model

1.2 Requirement Modelling / Fact – Finding technique
1.2.1 Interview
1.2.2 Document Review
1.2.3 Observation
1.2.4 Questionnaire and Surveys
1.2.5 Overview Feasibility Study
1.2.6 Data Flow Diagram – DFD

Unit – 2 Object Oriented Analysis 25 to 46

2.1 Fundamentals of Object Oriented Analysis and Design Concept
2.1.1 Objects and classes
2.1.2 Class Relationships
2.1.3 Key concepts of OOAD
2.1.4 Advantages of Object Oriented Design and Analysis

2.2 UML (Unified Modelling Language) Diagrams
2.2.1 UML Diagrams
2.2.2 Use Case Diagram
2.2.3 Representation of Use Cases
2.2.4 Significance of Use Case Diagram
2.2.5 How can one identify Use Cases of a system?
2.2.6 Simplification of use cases through factoring of commonality
2.2.7 Organization of Use Cases
2.2.8 Use Case Packaging
2.2.9 Interaction Diagram
2.2.10 Sequence Diagram

Unit – 3 UML Diagrams 47 to 61

3.1 Activity Diagram
3.1.1 Elements of Activity Diagram
3.1.2 The following shows example of an activity diagram

3.2 Class Diagram
3.2.1 Elements of Class diagram
3.2.2 The following shows example of a class diagram

Unit – 4 QA and Testing 62 to 89

4.1 Software Quality, Software Quality Assurance and Control
4.1.1 Software Quality Assurance vs. Quality Control
4.1.2 Software Errors, Faults and Failures
4.1.3 Software Quality Factors

4.2 Software Testing
4.2.1 Software Testing Objectives
4.2.2 Software Testing Classification
4.2.3 Integration Testing Strategies

4.3 System Testing
4.3.1 Performance Testing

4.4 Regression Testing

4.5 Test Metrics and Measurements
4.5.1 Objectives of Software Quality Metrics
4.5.2 Software Quality Metrics categories
4.5.3 Limitations of Software Metrics

Paper 2019

Unit -1

System Analysis and Design

- **Software Development Models**
  - Waterfall Model
  - Incremental Model
  - Spiral Model
- **Requirement Modelling / Fact – Finding technique**
  - Interview
  - Document Review
  - Observation
  - Questionnaire and Surveys
  - Overview Feasibility Study
  - Data Flow Diagram – DFD

Unit-1 System Analysis and Design

❖ Introduction:

System means is organized relationship between any set of components or degree to achieve some common cause or objective.

System Analysis it is a process of collecting and interpreting the facts, identifying the problems and decomposition of a system into its components.
System analysis is conducted for the purpose of studying the system in order to identify its objective. It is problem solving mechanism that improves the system and ensures that all the components of the system work efficiently to accomplish their purpose.

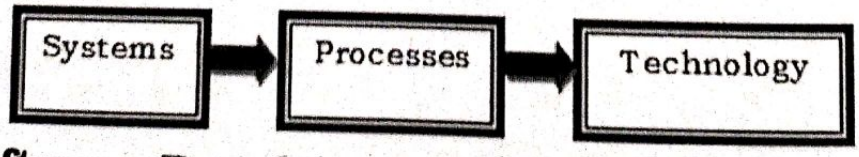
The **design of the system** is a layout or plan for a solution for the system. The design process of a system is has two levels. At the first level the focus is on deciding which modules are needed for the system, the specifications of these modules, and how the modules should be inter – connected. This type of design is called as **High level design**.

In second level, the things like internal design of the modules or how the specifications of the module can be satisfied is decided. This type of design is called as **Detailed design or Logic design.**

Detailed design essentially expands the system design to contain a more detailed description of the processing logic and data structures so that the design sufficiently complete for further coding portion.

The **Design of the system** is correct if a system built exactly according to the design which satisfies the requirements of that system. The goal of design process is not simply to produce a design for the system. Instead goal is find to the **best** possible design within the limitations imposed by the requirements and the physical and social environment in which the system will operate.

System analysis and design (SAD) is mainly focus on:



1.1 Software Development Models:

Basically, a model is a representation of a real or planned system. A model is used to visualize relationship.

1.1.1 Waterfall Model:

The waterfall model was the first process model to be introduced. It is also referred as a **Linear Life Cycle** model. It is very simple to understand and use.

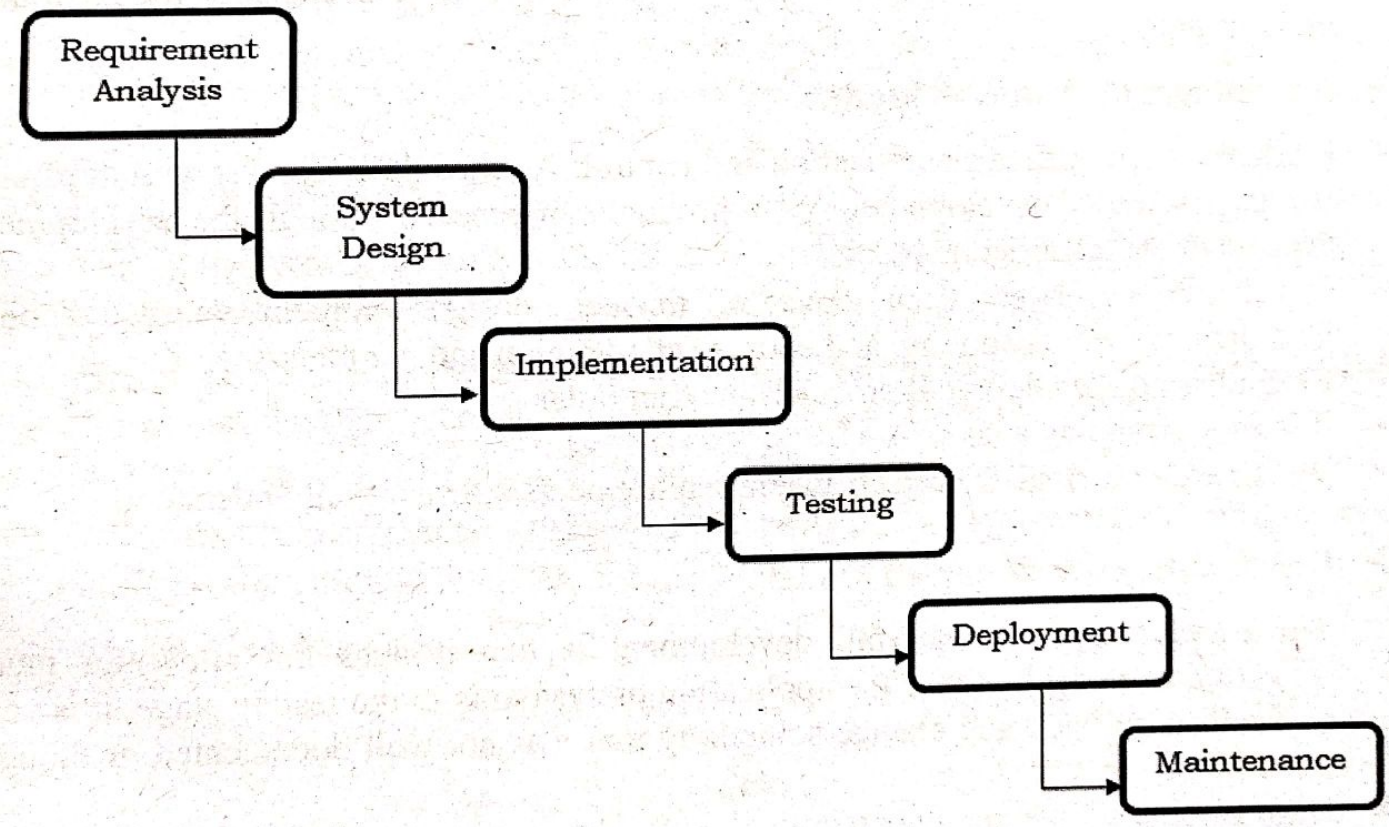
In waterfall model, each phase must be completed before next phase can begin and there is no overlapping in the phases.

Waterfall model is earliest SDLC approach that was used for the software development.

- **Waterfall model design:**

In waterfall approach, the whole process of software development is divided into separate phase. The outcome of the one phase acts as the input for the next phase sequentially.

Following is the diagrammatic representation of different phases of waterfall model:



The Sequential phases in waterfall model are:

Requirement Gathering and Analysis:
All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.

System Design:

The requirement specifications from first phase are studied in this phase and system design is prepared. System design helps in specifying hardware and system requirements and also helps in defining overall system architecture.

Implementation:

With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is called as Unit Testing.

Integration and Testing:

All the units developed in the implementation phase are integrated into a system after testing of each unit.

Deployment of System:

Once the functional and non functional testing is done, the product is deployed in the customer environment or released into the market.

Maintenance:

There are some issue which come up in the client side environment. To fix the issues patches are released. Maintenance is done to deliver these changes in the customer environment.

- **Advantages of Waterfall Model:**

✓ It allows for the departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.

✓ Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance.

✓ Each phase of development proceeds in strict order.

✓ It is easy arranging a task.

✓ Works well for smaller projects where requirements are very well understood.

- **Disadvantages of Waterfall Model:**

× The disadvantage of waterfall development is that it does not allow for much reflection or revision. Once the application or system is in the testing stage, it is very difficult to go back and change something that was not well documented or thought upon in the concept stage.

× High amount of risk and uncertainty.

× Not a good model for complex and object oriented projects.

× It is difficult to measure progress within stages.

1.1.2 Incremental Model:

The Incremental model is a method of software development where the model is designed, implemented and tested incrementally until the product is finished.

It involves both **development** and **maintenance**. The product is defined as finished when it satisfies all the requirements of the client.

This model is combines the elements of waterfall model.

- **When to use Incremental Model:**

This model is used where requirements are clear and can implement by phase wise. Mostly this model is used in web applications and product based companies.

- **Advantages of Incremental Model:**

✓ This model is more flexible, less costly to change scope and requirements.
✓ Easier to test and debug during the smaller iteration.
✓ Easier to manage risk because risky pieces are identified and handled during its iteration.

- **Disadvantages of Incremental Model:**

× Each phase of an iteration is rigid and do not overlap to each other.
× Problems may arise pertaining to system architecture because not all requirements are gathered up front for the entire software life cycle.

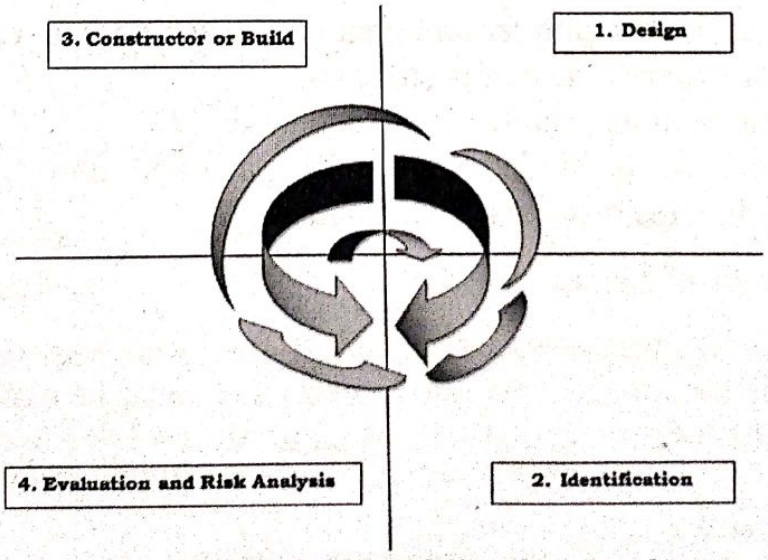
1.1.3 Spiral Model:

The spiral model combines the idea of iterative development with the systematic, controlled aspects of the waterfall model.

Spiral model is a combination of iterative development process model and sequential linear development model i.e. waterfall model with very high emphasis on risk analysis.

It allows for incremental releases of the product, or incremental refinement through each iteration around the spiral

- **Spiral Model Design:**



The spiral model has four phases. A software project repeatedly passes through these phases in iterations called **Spirals**.

Identification:
This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

This also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral the product is deployed in the identified market.

Design:
Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and final design in the subsequent spirals.

Construct or Build:
Construct phase refers to production of the actual software product at every spiral. In the baseline spiral when the product is just thought of and the design is being developed a POC *ProofofConcept* is developed in this phase to get customer feedback.

Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to customer for feedback.

Evaluation and Risk Analysis:
Risk Analysis includes identifying, estimating, and monitoring technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

➢ **Advantages of Spiral Model:**

✓ In spiral model, changing requirement can be accommodated.
✓ It allows for extensive use of prototypes.
✓ Requirements can be captured more accurately.
✓ Development can be divided into smaller parts and more risky parts can be developed earlier which helps better risk management.

➢ **Disadvantage of Spiral Model:**

× Management is more complex.
× Not suitable for small or low risk projects and could be expensive for small projects
× Large number of intermediate stages requires excessive documentation.

1.2 Requirement Modelling / Fact – Finding technique:

There are various fact-finding techniques used for gathering information about the system under study, some of these are:

- **Interviews**
- **Document review**
- **Observation**
- **Questionnaires and Surveys**
- **Sampling and research**

Let's understand about all fact finding technique one by one.

1.2.1 Interview:

An interview can be defined as a gentleman conversation between two people or more where questions are asked to a person to get the required responses or answers.

The interviews are an important fact-finding technique. It is a planned meeting during which you obtain information from people. The analyst must have skills needed to plan, conduct, and document interviews successfully.

➢ **Objective of an Interview:**

It helps to verify the information provided by the candidate. It helps to ascertain the accuracy of the provided facts and information about the candidate.

Its helps in establishing the mutual relation between the employee and the company. It is also used to candidate so that he comes to know about his profession, the type of work that is expected from him and he gets to know about the company or organisation.

➢ **Interview Process:**

Interviewing process consists of the following steps:

1. Prepare for the interview.
2. Establish objectives for the interview.
3. Determine the people to interview.
4. Write down all the questions that must ask.
5. Conduct the interview.
6. Document the interview.
7. Evaluate the interview.

To conduct an Interview following things need to consider:

- Use simple and clear words.
- Don't mix between facts and opinions.
- Listen to the answer.

- Don't interrupt.
- Don't suggest.
- Don't make the interview personally.
- Let him think. - Be flexible.

Advantages of Interview:

✓ Give analyst opportunity to motivate interviewee to respond freely and openly.
✓ Allow analyst to probe for more feedback Permit analyst to adapt or reword questions for each individual.
✓ It can observe nonverbal communication.

Disadvantage of Interview:

× It is Time – Consuming Process.
× Success highly dependent on analyst's human relations skills.
× May be impractical due to location of interviewees.

1.2.2 Document Review:

Document review can help you to understand how the current system is supposed to work. You should review all the system documentation, which include technical documentation and operations manual such as user guide, also include the software package documentation if any.

1.2.3 Observation:

Observation is broad approaches available for the collection of the data in primary data collection method.

In **Observation method**, only present/current behaviour can be studied. Therefore many researchers feel that this is great disadvantage.

A casual observation can familiarize the researcher to identify the problem. Such a length of the queue in front of a food chain, price and advertising activity of the competitor etc.

Observation is the least expensive of the data collection.

➢ **Types of Observation Methods:**

There are several methods of observation of which, any one or combination of some them, can be used by the observer. They are:

- Structured or unstructured observation methods
- Disguised or undisguised observation methods
- Direct – Indirect observation methods
- Human – mechanical observation methods

- **Structured or unstructured Observation Methods:**

Observation should be structured or unstructured depends on the data needed to be observed.

Example 1: A manager of a hotel wants to know "How many o his customer visits the hotel with family and how many visits as single customer".

Here observation is structured because since it is cleared "What is to be observed?" He may tell waiters to record this. This information is required to decide the tables and chairs requirements and also the layouts.

Suppose, the manager wants to be know how single customer and customer with family behave and what is their mood. This study is unclear or vague, it needs non-structured observation.

It is easier to record structured observation than non structured observation.

- **Disguised - Undisguised Observation Methods:**

In Disguised observation, the respondents do not know that they are being observed. In Undisguised observation, the respondents are well aware that they are being observed.

The main strength of disguised observation is that, it allows for maintaining the true reactions of the individuals.

In undisguised method, observation may be contained due to induced error by the objects of observation. The ethical aspect of disguised observation is still questionable.

- **Direct – Indirect Observation Methods:**

In direct observation method, the actual behaviour or phenomenon of interest is observed.

In Indirect observation, results of the consequences of the phenomenon are observed. Suppose, researcher is wants to know about the soft drink consumption of a student in hostel room. He may like to observe empty soft drink bottles dropped in to the dustbin. Similarly, the observer may seek the permission of the hotel owner to visit the kitchen or store. He may carry out the kitchen/store audit, to find out the consumption of various brands of spice items being used by the hotel.

It may be noted that the success of an Indirect observation largely depends on '"How best the observer able to identify the physical evidence of the problem under study".

- **Human - Mechanical Observation Methods:**

This type of observation is mostly done by the human observation, wherein trained observers are required to observe and record their observations.

In some cases mechanical devices like eye cameras are used for observation.

One of the major advantages of electrical or mechanical devices is that, their recordings are free from subjective bias.

➢ **Advantages of Observation Methods:**

✓ The original or actual data can be collected at the time of occurrence of the event.
✓ Observation is done in natural surroundings. Therefore, facts are known.
✓ Observation can be done on those who cannot voluble.
✓ Bias of the researcher is greatly reduced in observation method.

➢ **Limitations of Observation Methods:**

× This is an expensive method.
× For observation, extensive training of observer is required.
× External observation gives only surface indications. So only concrete behaviour can be observed.
× Two observers may observe the same event but may draw inference differently.
× It is very difficult to gather information on opinions and intentions etc.
× Observation is required over a long period of time and hence delay may occur.

1.2.4 Questionnaire and Surveys:

Questionnaire is a tool used to collect the data. Questionnaires are used to obtain information about workloads, reports received, volumes of transactions handled, typed of job duties, difficulties, and opinions, of how the job could be performed better or more efficiently.

A typical questionnaire starts with a heading , which includes a title , a brief statements, of purpose, the name and telephone number of the contact persons, the deadline date for completion , and how and where to return the form.

The heading usually is followed by general instruction that provide clear guidance on how to answer the questions. Also the questionnaires contain conclusion that thanks the participants and reminds them how to return the form.

➢ **Characteristics of Questionnaire:**

- It must be simple. Respondent should be able to understand the questions.
- It must be generate replies, which can be easily recorded by the unreview.
- It should be specific, so as to allow the interviewer to keep the interview to the point.
- It should be well arranged, to facilitate analysis and interpretation.
- It must keep the respondent interested throughout.

➢ **Different Types of Questionnaire:**

(i) Structured and Non Structured Questionnaire
(ii) Structured disguised Questionnaire

(iii)Non Structured disguised Questionnaire
(iv)Non Structured and Non disguised Questionnaire

Now, let understand about different types of Questionnaire in detail.

(i) Structured and Non Structured Questionnaire:

In this type of questionnaire, all the questions are structured so as to get the facts. The interviewer will ask the questions strictly as per the pre arranged order. E.g. "What is the strength of soap A in comparison with soap B?"

- Cost is less
- Lasts longer
- Better fragrance
- Produces more lather
- Comes in more convenient sizes

Structure, none disguised is widely used in market research. Questions are presented with exactly the same wording and same order to the entire respondent. The reason for standardizing questions is, to ensure that all respondents reply the same questions. The purpose of question is clear. The researcher wants the respondent to choose one of the five options given above. In non disguised type, the purpose of the questionnaire is known to the respondent.

(ii) Structured disguised Questionnaire:

- This type of questionnaire is used to find people's attitude, when a direct undisguised questions produces a bias.
- In this type of questionnaire what comes out is "What does the respondent know rather than what he feels". In a disguised type, the respondent is not revealed the purpose of the questionnaire.

(iii) Non structured and disguised Questionnaire:

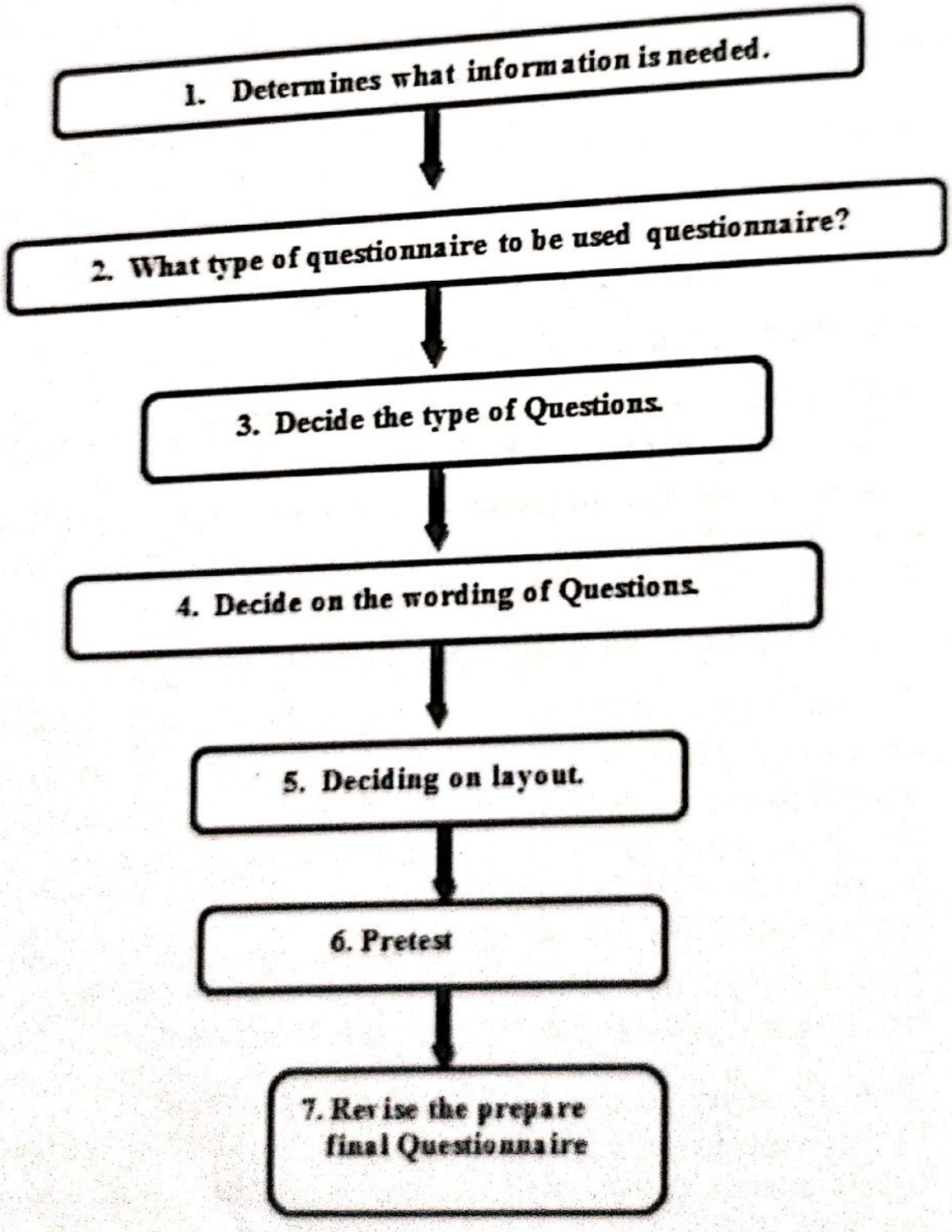
- The main objective is to conceal the topic of enquiry by using a disguised stimulus. Though the stimulus is standardized by researcher, respondent is allowed to answer in an unstructured manner.
- The assumption made here is that individual's reaction is an indication of respondent's basic perception.
- The techniques involve the vogue stimulus, that an individual is asked to expand or describe or build a story, three common types under this category are (1) Word association (2) Sentence Completion (3) Story telling.

- **Non structured and non disguised Questionnaire:**
- In this type method, purpose of the study is clear, but the responses to the questions are open ended. In this type, the initial part of the question is constant.

- After presenting the initial questions, the interviewer becomes much unstructured as the interviewer probes more deeply. Respondent subsequent answer determines the direction the interviewer takes next.
- The question asked by the interviewer varies from person to person. This method is called "**Depth interview**". The main advantage of this method is freedom permitted to the interviewer.
- The main disadvantage of this method of interviewing is that, it takes time and respondents may not co – operate. Another disadvantage is that coding of open ended questions may pose a challenge.

Questionnaire Designing:

Basically, Questionnaire is designed in 7 steps. Let's understand that 7 steps via figure.



1.2.5 Overview Feasibility Study:

Feasibility is a process that identifies, describes and evaluates proposed systems and selects the best system for the Job.

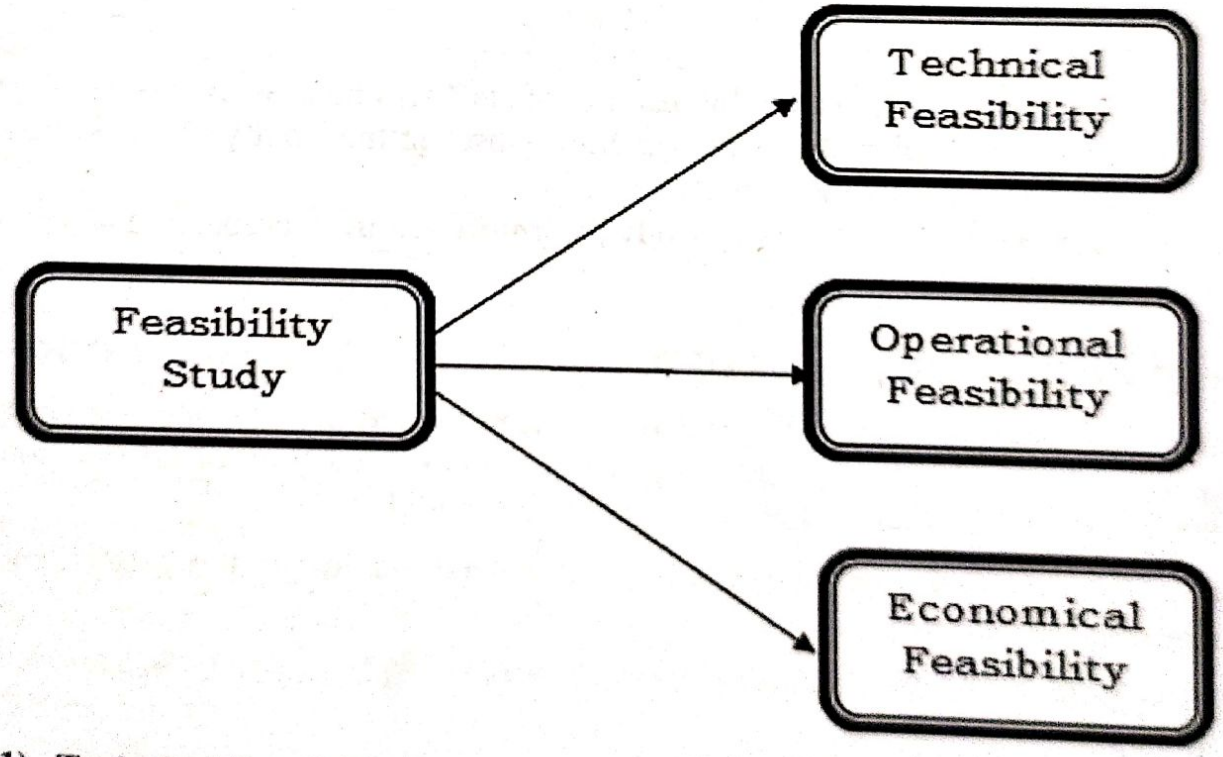
The objective of feasibility study is to establish the reasons for developing the software that is acceptable to end users, adaptable to change and conformable to established standards.

During the study, the problem definition is solved and all aspects of problems to be included in the system are determined.

To analyze whether the software will meet organizational requirements. To determine whether the software can be implemented using the current technology and within a specified budget and schedule.

➢ **Types of Feasibility:**

Various types of feasibility that are commonly considered include technical feasibility, Operational feasibility and Economic feasibility.



1) Technical Feasibility:

- It determines whether the technology needed for the proposed system is available and how it can be integrated within the organisation.
- Technical evaluation must also assess whether the staff have technical expert to understand and use the new technology.

2) **Operational Feasibility:**

It is concerned with human, organisational and political aspects to study. It includes two ways:

(i) **Technical Performance:** It includes issues such as whether the system can provide the right information for organisation's personnel at right time and right place.

(ii) **Acceptance within the organisation:**

It determines the general attitudes and job skills of existing personnel and whether any changes in job work will be acceptable by the current users.

3) **Economical Feasibility:**

- The purpose of assessing economical feasibility is to identify the financial benefits and cost associated with the development of the system.
- Economic feasibility is often known as the cost benefit analysis. To carry out an economical feasibility study actual money value against activities needed for implementing the system.

4) **Behavioural Feasibility:**

- It includes how strong the reaction of staff will be towards the development of new system that involves computer's user in their daily work. So resistant to change is identified.
- Note that after feasibility study ***feasibility report*** is prepared and submitted to the management.

1.2.6 Data Flow Diagram – DFD:

Data flow diagram (DFD) illustrated how data is processed by a system in terms of inputs and outputs. As its name indicates its focus is on flow of information, where data comes from, where it goes and how it gets stored.

DFD diagram describes how information enters and leaves the system and where information is stored.

It shows the scope boundaries of system as a communication tools between a systems analysis.

Creating Data Flow Diagrams Steps:

1. Create a list of activities
2. Construct Context Level DFD (Indentifies external entities and processes)
3. Construct level 0 DFD (Indentifies manageable sub process)
4. Construct level 1 – n DFD (Indentifies actual data flows and data stores)
5. Check against rules of DFD

DFD Naming Guidelines:

External Entity -> Noun
Data Flow -> Names of data
Process -> Verb Phrase

- a system name
- a sub – system name

Data Store -> Noun

Data Flow Notations:

Data flows are pipelines through which packets of information flow. Label the arrows with the name of the data that moves through it.

External Entity Notations:

External entities are objects outside the system, with which the system communicates. External entities are destinations of the system's inputs and outputs.

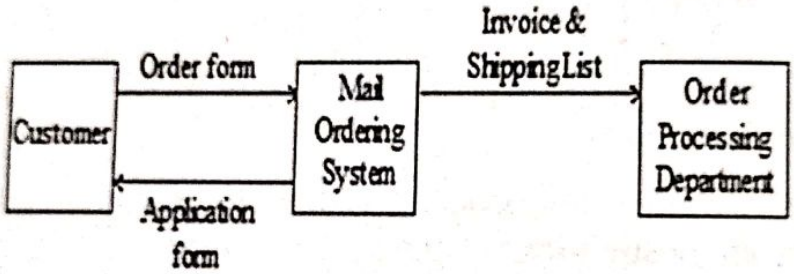
❖ DFD Examples :

Example 1 : Mail Ordering System :

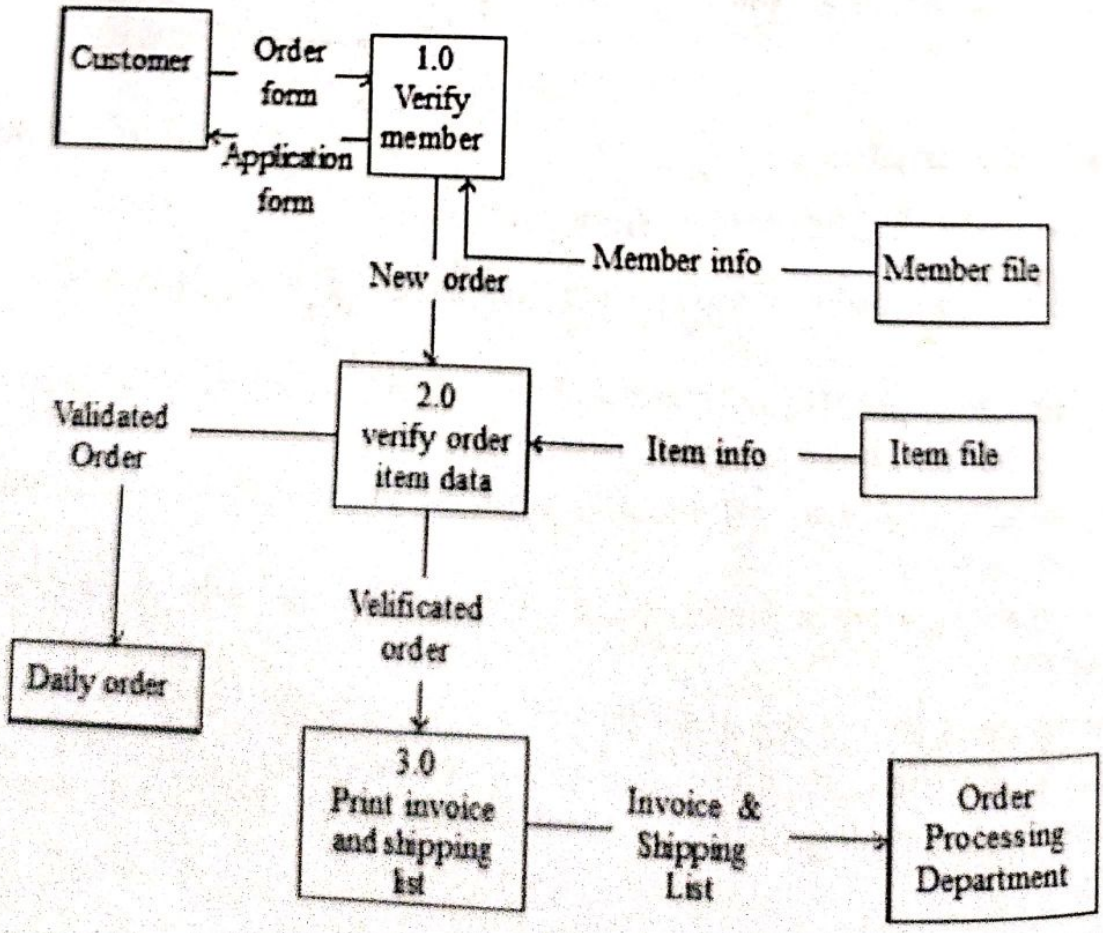
- Consider a mail order company that distributes CDs and tapes at discount prices to its members.
- When an order processing clerk receives an order form she/he verifies that the sender is a member by cheking the MEMBER FILE.
- If the sender is not a member the clerk returns the order along with a membership application form.
- If the customers are a member the clerk verifies the order item data by checking the ITEM FILE.
- Then the clerk enters the order data and saves it to the DAILY ORDERS FILE.
- At the same time the clerk also prints an invoice and shipping list for each order, which are forwarded to the ORDER FULFILMENT DEPARTMENT for processing there.

- A mail order company that distributes CDs and tapes at discount prices to record club members (tells the purpose of the system).
- When an order processing clerk receives an order form (data flow) she verifies (process # 1) that the sender is a club member by checking the MEMBER FILE (data store). If the sender is not a member the clerk returns the order along with a membership application form (data flow). If the customer is a member the clerk verifies the order item data (process #2) by checking the ITEM FILE (data store).
- Then the clerk enters the order data (data flow) and saves it to the DAILY ORDER FILE (data store).
- At the same time the clerk also prints an invoice and shipping list (data flow) for each order (process #3) which are forwarded to the ORDER FULEMENT DEPARTMENT (external entity) for processing there.

➢ **Context Diagram :** Mail Ordering System

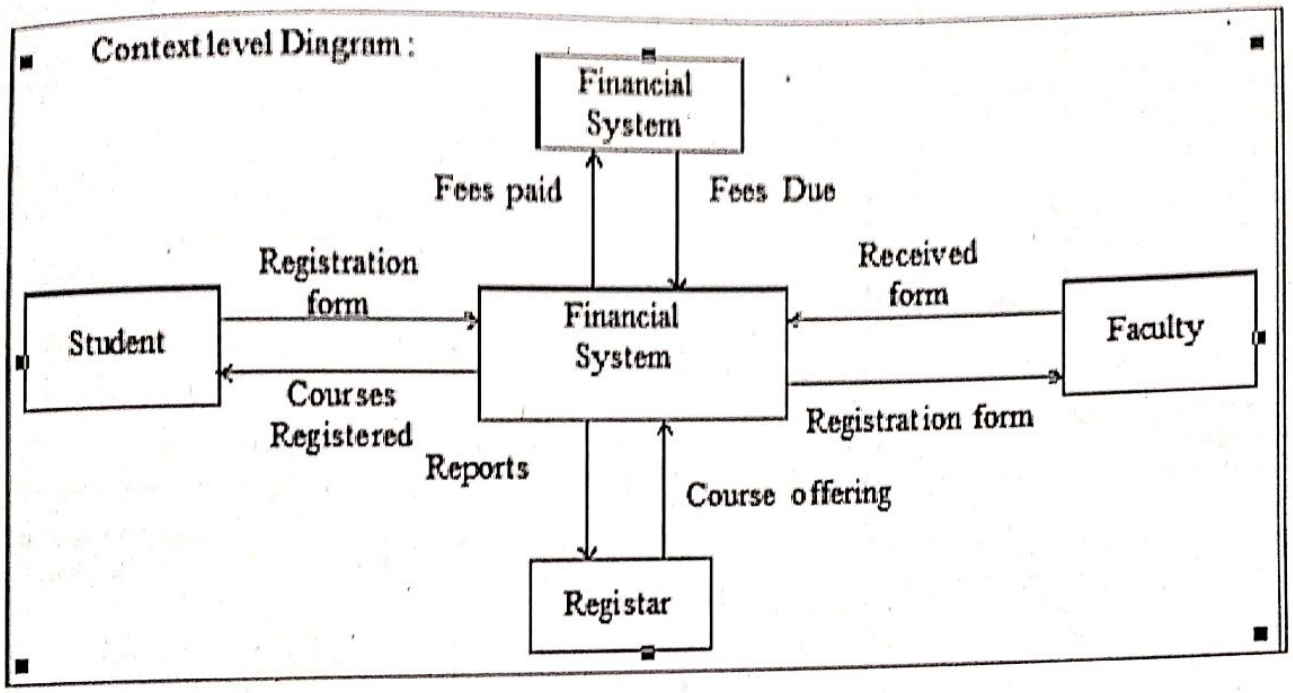


Level 0 diagram: Mail Ordering System

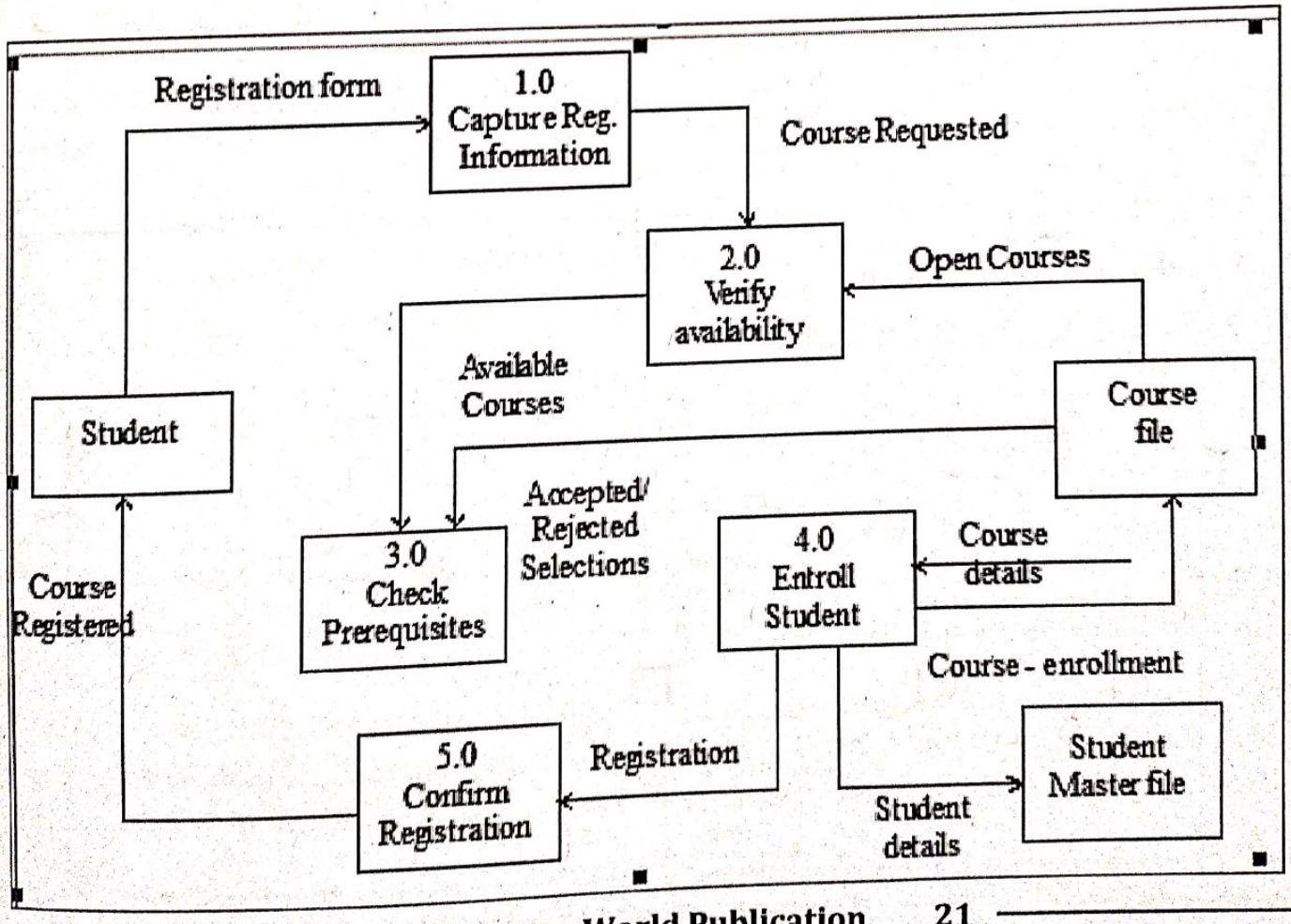


Example 2: Registration System

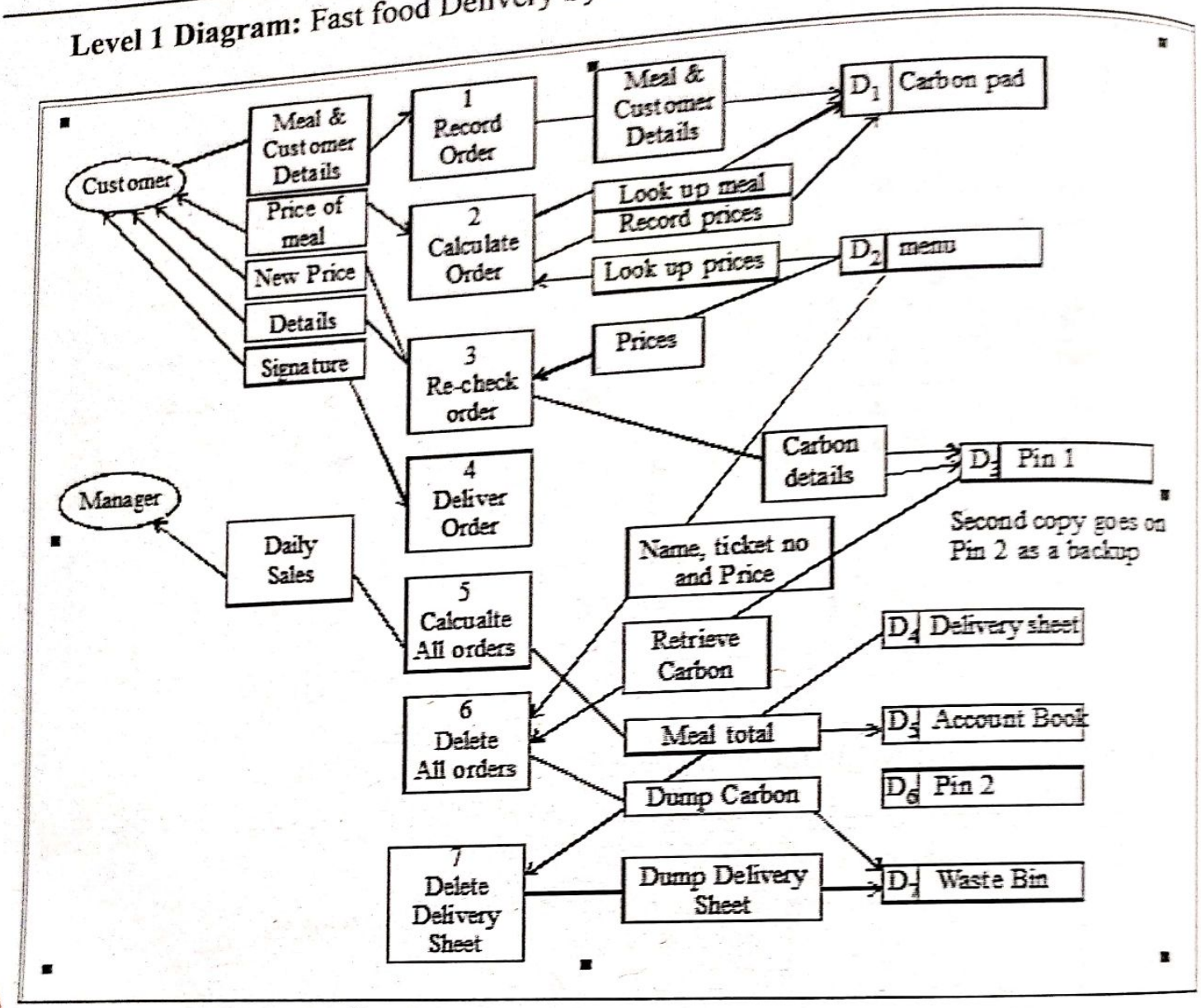
Context level Diagram:



Level 0 Diagram: Registration System

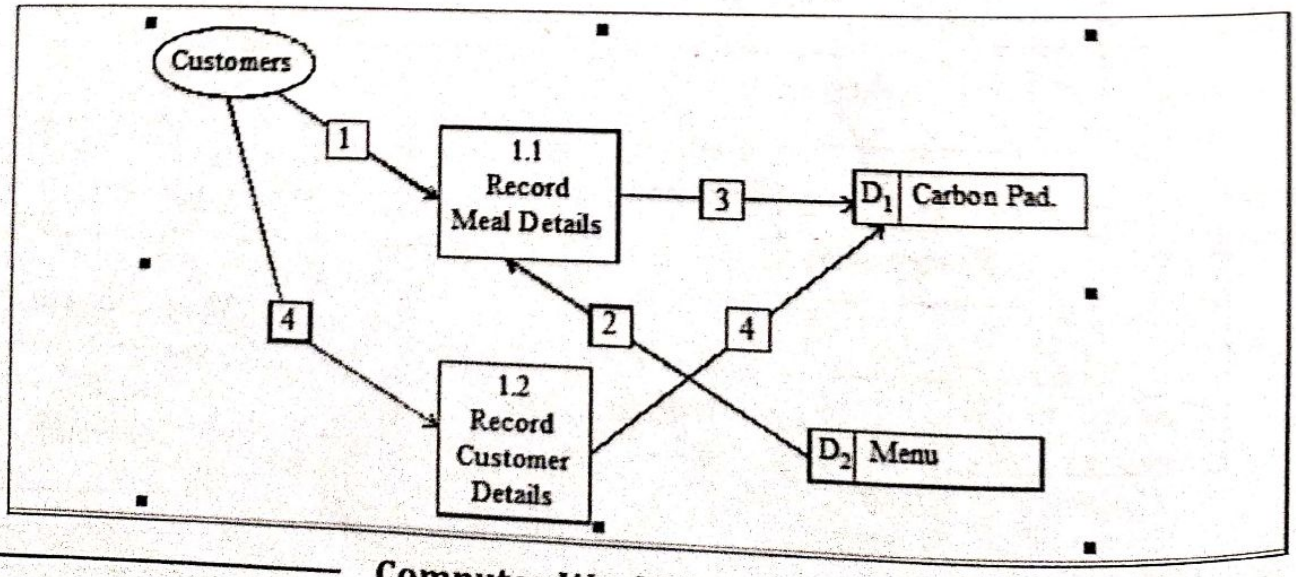


Level 1 Diagram: Fast food Delivery System



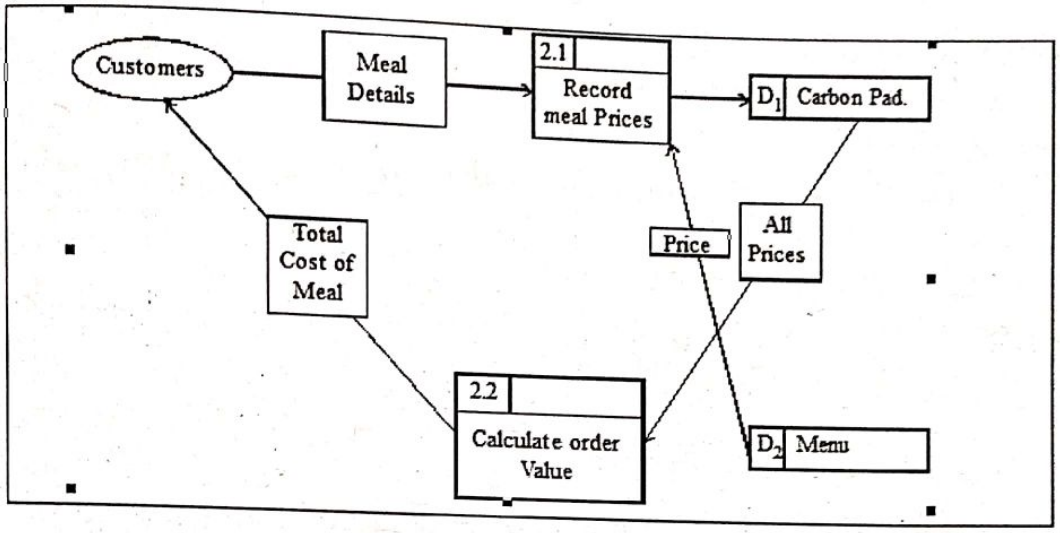
Level 2: Current System :

Process 1: Record Order Details



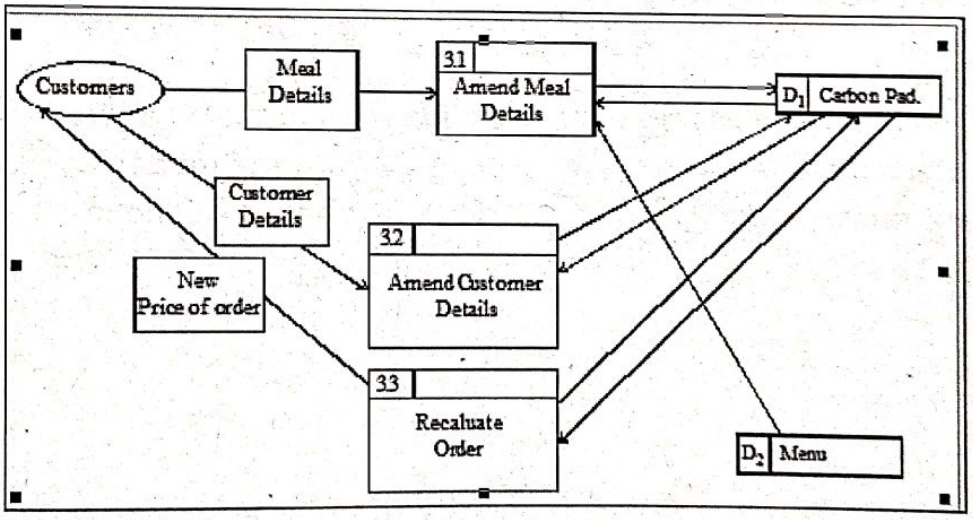
Notes: Customer gives meal details (1) and prices are looked up (2) in the menu before being added (3) to the carbon pad.
Customer details such as address etc. also then added (4) to the carbon pad.

Process 2: Calculate Order Value



Notes: During a call individual meal prices are looked up in the mean and added to the carbon pad. On completion of an order the prices of all ordered meals, on the pad, are added together to give the total cost of the meal. This is the relayed to the customer.

Level 2: Current System
Process 3: Re-Check Order
Level 2: Current System
Process 3: Re-check order



Notes: After an order is complete the customer may ring up to check they have given the correct details or to amend an order. In this case details on the carbon pad are checked and amended as necessary before the customer is given the near price of the order.

☺☺☺☺☺

Exercises

1. Explain waterfall model.
2. When to use incremental model?
3. List out all fact finding techniques.
4. What is interview technologies and write down the advantage and disadvantages of it?
5. Describe different types of questionnaire.
6. Write down the advantages of spiral model over the waterfall model.
7. Write a short note an feasibility study.

Select proper options:

(i) Which fact finding techniques can help to understand that how the current system is supposed to work?

(a) Questionnaire (b) interview
(c) Document review (d) A & B both

(ii) ..... Is a process that identifies describes and evaluates processed systems and select the best system for the job.

(a) Feasibility (b) Questinnaire Designing
(c) Interview (d) none of above

(iii) Which software model is refered as "Linear like cycle" model?
(a) Waterfall model (b) spiral model
(c) Incremental model (d) none of above

Ans: **1 – c,** **2 – c,** **3 – a**

☺☺☺☺☺

Unit -2

Object Oriented Analysis

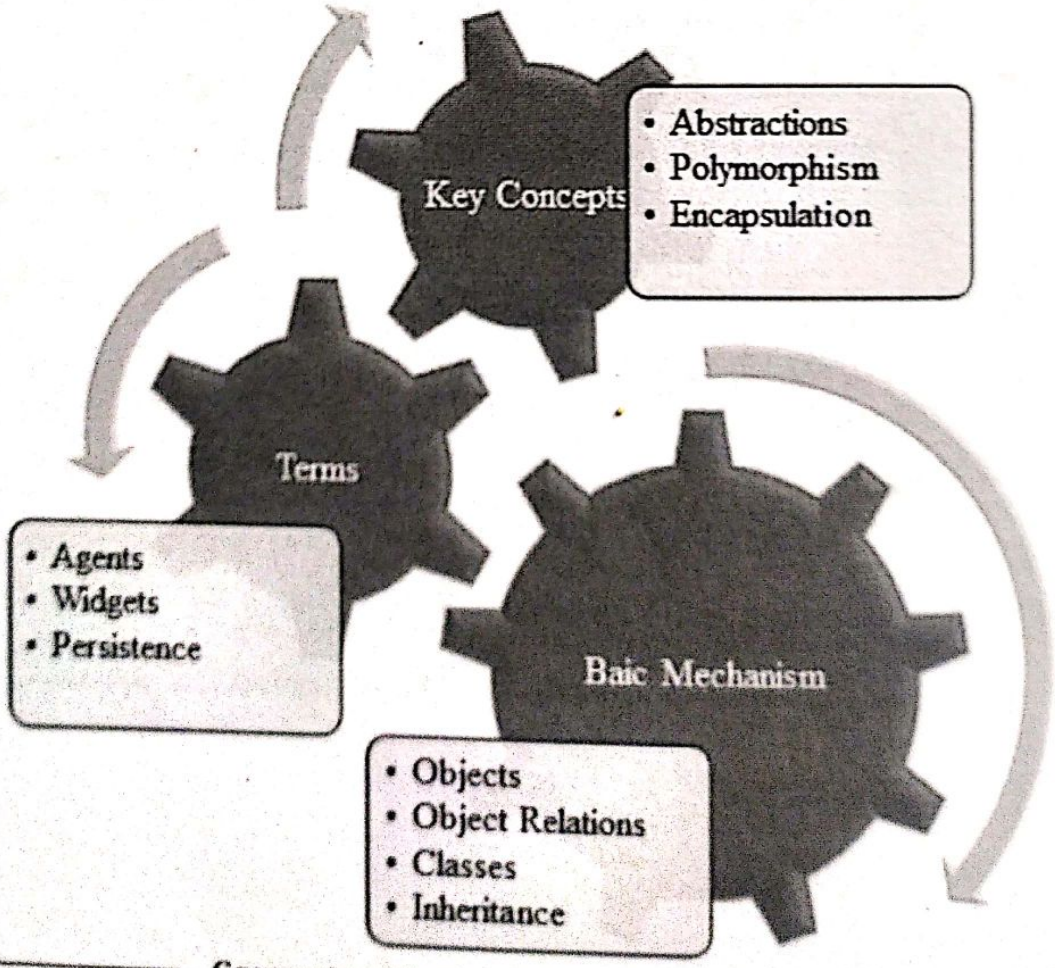
- **Fundamentals of Object Oriented Analysis and Design Concept**
  - Objects and classes
  - Class Relationships
  - Key concepts of OOAD
  - Advantages of Object Oriented Design and Analysis
- **UML (Unified Modelling Language) Diagrams**
  - UML Diagrams
  - Use Case Diagram
  - Representation of Use Cases
  - Significance of Use Case Diagram
  - How can one identify Use Cases of a system?
  - Simplification of use cases through factoring of commonality
  - Organization of Use Cases
  - Use Case Packaging
  - Interaction Diagram
  - Sequence Diagram

Unit-2 Object Oriented Analysis

Modelling is the way of generating a pictorial representation of solutions of a problem. A simple problem may have multiple models out of which the most suitable can be used or a single model may be sufficient enough to match requirements. Designing process is quite different from modelling. ***Modelling uses set of notations to prepare a document of results of analysis and design***. While, ***a design process focuses on how a problem can be solved using certain steps***. Design process is sometimes referred as a ***design methodology*** as well. Object Oriented Analysis and Designing has been an important component of Software Development Life Cycle. In this chapter we are going to know about the elements of Object Oriented Analysis and Design and will cover the most basic modelling diagram called UML diagram.

2.1 Fundamentals of Object Oriented Analysis and Design Concept:

Object Oriented Analysis and Design is based on Object Oriented approach only. Approaches adopted in programming such as abstraction, class, object, etc.



2.1.1 Objects and classes:

In object oriented paradigm, a system is developed using interacting objects. Object can be imagined as any living or non-living thing. While developing any system (software), we need to consider the functionalities of the system and create intangible objects as well.

For example, Learning Management System is there to maintain the reading material for students. All the reading materials can be considered to be an object of the system. The benefit of creating natural objects for any system is ease of understanding of the system. The reason behind the simplicity is the decomposition of the system using objects. As described by any object oriented programing language, Object is a collection of data used privately within the object and functions to access those objects. This clearly shows that any object cannot access private data of any other object. This can be considered as a feature of data hiding or data abstraction. For an example, object of result and object of attendance have some private data such as %result of object of result and %attendance of attendance object. Now %result cannot be accessible by attendance object.

Let's assume Professor is an object of the learning management system. Following data are the private data of this object:

- Name of the professor
- Employee Code of professor
- Department of the professor
- Contact number of the professor
- Email address of the professor
- Subject allotted to professor
- Assignment number
- Assignment date
- Assignment file
- Material file
- Material number
- Material date

Functionalities that will be performed by the object are as follows:

- Upload Assignment
- Collect Assignment
- Upload material

The data mentioned above are the attributes of the object and the functionalities are the functions / methods. Though it is mentioned very clearly here that an application is made up from objects only, there will be the use of primitive datatypes such as int,

float, etc. These datatypes are not really objects they are just primitive data. Hence, programs having these are called ***Hybrid***.

All such objects constitute a class. This means all the objects of same classes possesses same kind of data and methods. Class is an abstract data type. For example, set of all learning management system members would be covered under a class in the system. In our case, each member will have above mentioned data and methods. Once, we define a class and encapsulate (wrap up) data and methods under a class name, it will provide a kind of template which we can reuse anytime. Only, object creation is needed to replicate information. Here, class is well thought-out to be an abstract datatype which leads to some inferences as mentioned below:

- Data can be accessed only through methods available for object.
- We can instantiate an object from class. (because class is a type)

Methods are implemented to provide operations that a user can carry out. Above mentioned operations (functionalities) such as upload material, upload assignment, etc. will be used by the member and they will be provided as functions which are generally called methods here. Methods and operations are no more different in understanding. But, at the same time they have very minute difference according to the technical terminologies. ***An operation is a specific functionality or responsibility of a class while method is the actual implementation of the responsibility.***

NOTE: It may be possible that user defined multiple methods with same name but different behaviour. This is called method overloading.
These methods are available to objects so that objects can access and manipulate data of another object.

2.1.2 Class Relationships:

Classes can be related to each other via following four ways:

- Inheritance
- Association
- Aggregation and composition
- Dependency

➢ **Inheritance:**

Inheritance, as the name suggests, it is the feature to have new class from the existing class by extending and/or modifying the existing class. The original class is called ***Base Class (Parent Class)*** and newly acquired class is called ***Derived Class (Child Class)***. For example, let us assume that learning management system is accessible by various members such as Professors, Students, etc.

Personal details such as name, address, date of birth, contact number, email address etc. will be data for all kinds of members. While, uploading the material will be the feature provided to the professor only. Student will have functionalities which they

can accomplish such as upload their solution for particular assignment and get the marks. General data such as personal details can be defined in a general class such as members and data regarding the user functionalities can be defined for particular user class such as professor and student class. These both class definitions can extend general member class so that they do not require defining personal information related data again. Each derived class is the specialization of its base class as we are adding special functionalities using methods in the definition. ***Inheritance is very much important as it enhance the reusability of the code.***

A class may inherit data and methods from multiple classes. Such as Students and Professor both have profile page and blog. This both features are implemented using a class named as "Blog". Now Blog will require information of both class's professor and student. So, it has to inherit both classes or declare new data which can accomplish tasks. This is called multiple inheritances.

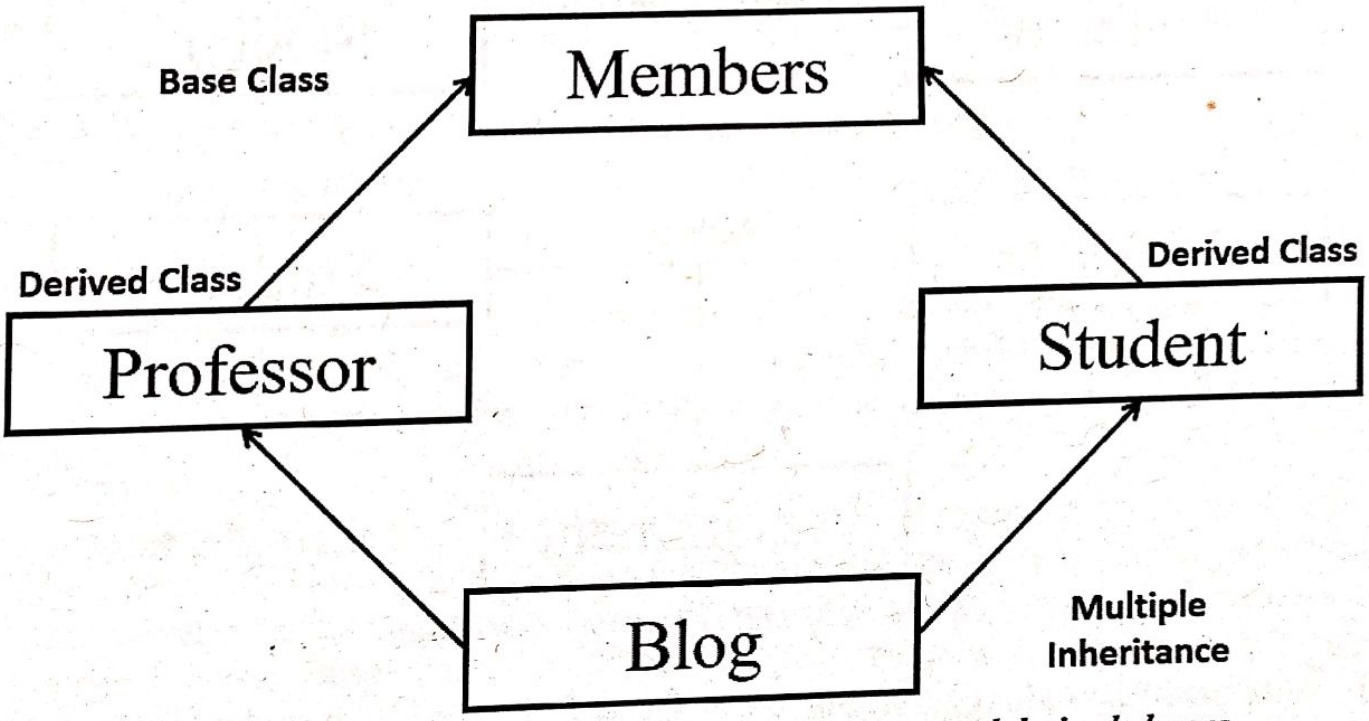


Figure 1 Example of multiple inheritance with base and derived classes

- **Association and Link:**

Association here refers to the accessibility of object of different classes. If two classes are associated, then their object can access methods of each other. Both the object will have sufficient information about other classes so that it becomes easier for an object of one class to utilize method of other class. Let us assume that Professor is associated with student. So, the object of professor class has access to methods of student class. This means that an object of professor class can invoke some methods of student class such as ***CalculateAttendancePerformance*** to calculate the overall attendance of the student.

Link is the relationship between two objects of the associated classes. For example, professor assigns assignment to students. Here professor and students are classes and assigning assignment is the link between these two classes. So all such links are part of an association. Here, we mentioned only one method but there will be more than one always. All such links constitute association. Hence, association is nothing but a collection of similar links between two classes.

Mathematically, link is an ordered list. For example, Prof. Parth Assigned assignment to Student named James. So, Parth and James are object instances of professor and student classes. This can be represented as an ordered list {Parth, James}. This is a binary relation. In certain process, more than two classes can be associated as well. For example, assignment itself is a separate class, then it will become ternary Association. Recursive association can be there if two (or more) different objects of same classes are linked by association relationship.

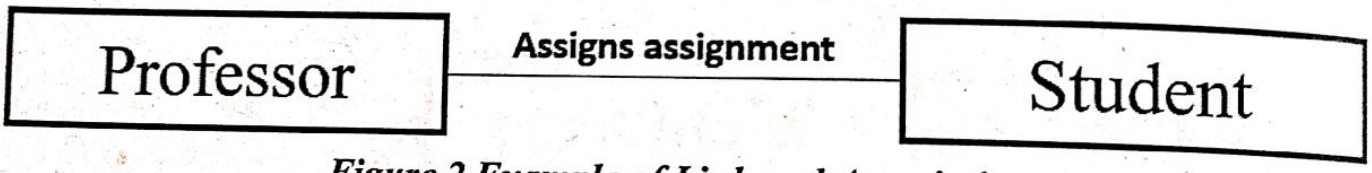


Figure 2 Example of Link and Association

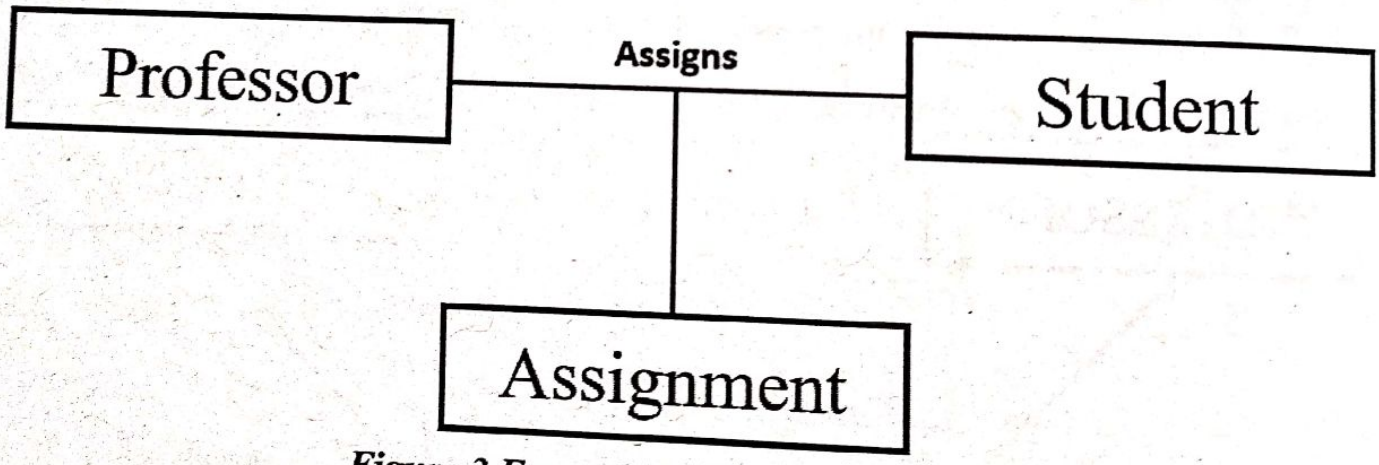


Figure 3 Example of ternary association

- **Composition and Aggregation:**

It can be easily understood from association that two objects of same or different classes can be in relationship or linked to each other. But, it is not clear whether two objects have complete relationship or partial. This can be represented using Composition and Aggregation. Objects which encompass other objects are called composite objects. Suppose a student can have 3 assignments for a subject. Here, we can see a student object is composed with 3 assignment objects. This is why composition and aggregation is known as ***has*** relationship. These relationships cannot be circular or we can say reflexive. Unlike recursion association, these relationships cannot contain object of same type within itself.

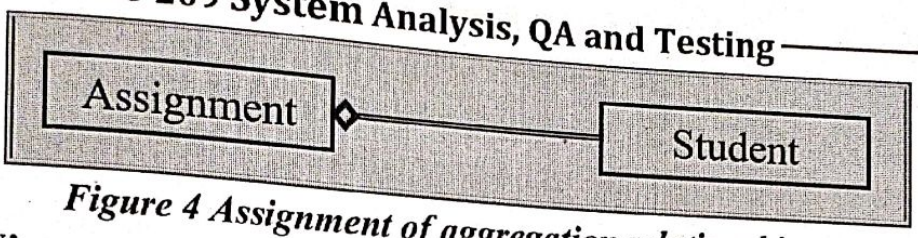


Figure 4 Assignment of aggregation relationship

- **Dependency:**

Dependency is also a kind of relationship between two classes. Let us understand a requirement of learning management system. If professor removes an assignment, then marks related to that assignment must be deleted too. As it is not going to be considered for final result. In such case, a dependency relation between two classes shows that any change made to the independent class should reflect to dependent class too. Dependency relation exists to fulfil following requirements:

- ✓ A class can call methods of another class
- ✓ A class has an interface class implementation too. If any changes are made to interface class, then the class implementing interface class will be having the changes too.

2.1.3 Key concepts of OOAD:

Following section will have detail description about few key concepts used in OOAD.

- **Abstraction:**

Abstraction means focusing on few contexts level aspects of any problem only. As it is very much abstract, it is easy to present and prepare. Preparation of abstraction depends on the purpose of system. One can achieve abstraction for Features as well as Data.

Feature Abstraction: A class hierarchy is an example of feature abstraction. The base class will have abstraction to features developed in the child/derived class. If multilevel inheritance is used, then every base class will be abstract representation of derived class. Base classes will contain only necessary features which will be inherited and used in further levels. If a class is extended by more than one classes, it can be easily understood that base class will have all the abstract level features which will be reserved by derived classes and which are common in all.

Data Abstraction: Objet as defined earlier, contains data and methods to manipulate those data. Object hides private data. Methods which have access to private data can only access private data. This means, data abstraction is about hiding data within the object. Actual storage and usage of the data is completely hidden as only methods will be called using objects. Data abstraction using objects is always beneficial to reduce the coupling between various objects in OOAD. Object of a class cannot access data of other classes.

Abstraction is the best way to reduce perseverance of complexity of any software designing. It will always increase productivity of software developers.

- **Encapsulation:**

Data is encapsulated within methods as object cannot directly access private data of class. Object will be able to access data by invoking methods only.

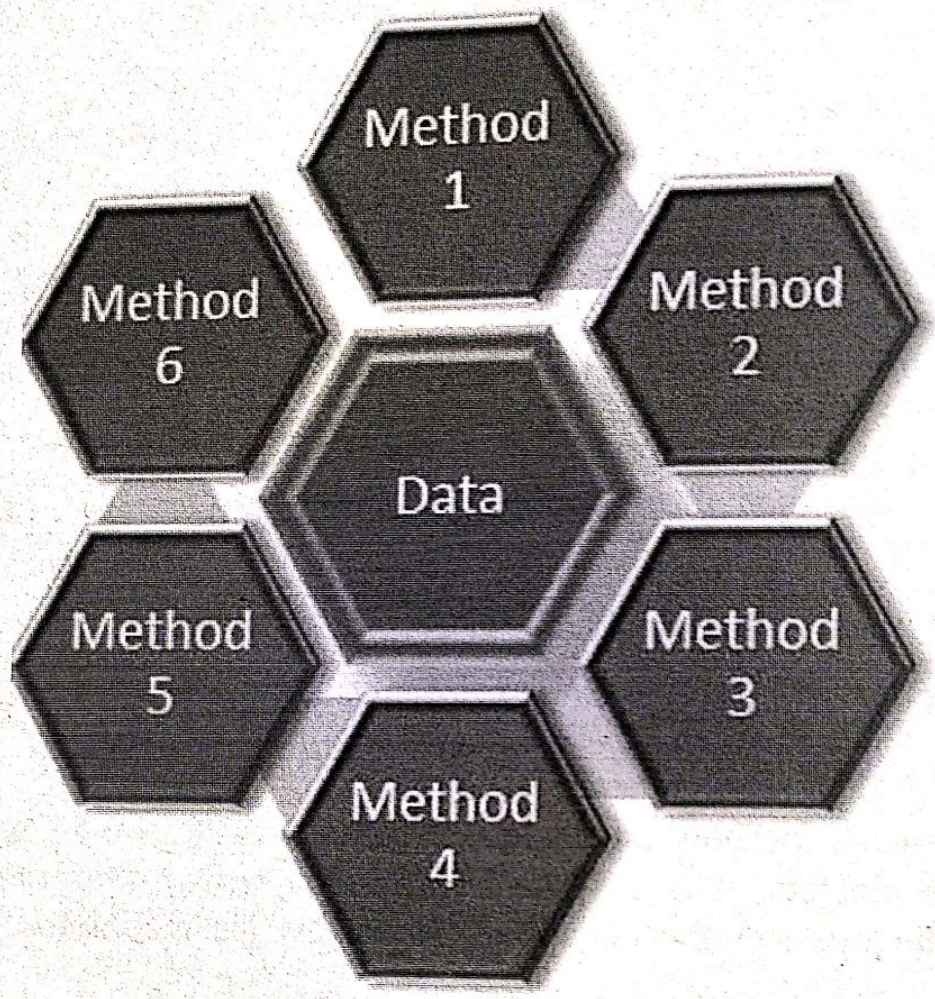


Figure 5 Schematic representation of encapsulation

Above shown diagram shows that there is no way to access data directly without using methods which can be invoked using objects only. Below mentioned are the advantages of encapsulations.

Data Protection: As object cannot directly access data, there are very less chances that a variable or data gets corrupted by other objects. Both kinds of protections, protection against unauthorised access and protection against issues due to concurrent access, are provided by Data Protection. Followings are the key benefits of this feature.

Data Hiding: Data hiding is about hiding the internal process of data manipulation from external view. This provides reusability features as well. It is also to notice that objects can be used across various projects as well. If internal process or implementation of any object is redefined or updated, it does not affect any other objects. This is why error identification, resolution and maintenance become easy.

Weak Coupling: Objects are going to communicate with each other through messages only. There won't be any way other than that. This shows every object can be studied and understood easily with design perspective and implement perspectives as well. One can isolate whole object from other objects as no strong coupling is there. Objects are not integrated at all. They are just used to accomplish particular tasks using sequence of tasks. Objects will use message passing to receive certain output or returns of other methods of other objects.

Polymorphism: Polymorphism means same name but different behaviour. Same method may respond different depending upon the way it is invoked. It can be implemented using parameters passed to method or return values of a method. Polymorphism can be static and dynamic. Static polymorphism refers to the methods which are predefined to accept particular parameters. For example, area is the name of a method which accepts 1 or 2 arguments (parameters). If it is called using singe parameter it will find area of a circle ($A = \pi r^2$). And, if two parameters are passed it will find out area of square using $A = l * b$. This is known as **static polymorphism.** Dynamic Polymorphism is related to the dynamic address binding to the invocation of methods. Method overridden is one of the ways of implementing dynamic polymorphism. Inheritance provides a way to redefine methods implemented by Base Class. When a method is invoked by object of derived class, appropriate method from base or derived class is called. This behaviour is called dynamic polymorphism. Because, exact method will be decided at the time of execution i.e. runtime. ***The foremost advantage of dynamic binding is that it provides complete support for code reuse and easy maintenance. Even programming becomes easier.***

```
Class shape{
      public area(float r){
            return 3.14 * r *r;
      }
      public area(float r){
            return l*b;
      }
```

Figure 6 Overloaded method example.

Genericity: Genericity offers a way to defined classes with parameters. For example, in the most used language like python and R Programming, a common class print is used. If you use print for float value, it will print float and if you use for high level data structure, it will print them too.

```
myVariable = 10

myVariable2 = [1,2,3]

#Above declared are 2 variables. myVariable is of integer type
and #myVariable2 is a list (kind of single dimensional array).
#Generally, we require loop to print but in python print class
is #so generic that it works with all data structures.


print(myVariable)

Output: 10 #prints value


print(myVariable2)
```

Figure 7 Example of Genericity of print class in python programming

2.1.4 Advantages of Object Oriented Design and Analysis:

- ✓ Using OOAD, we can have code reusability.
- ✓ As code can be easily extended, productivity is increased.
- ✓ Objects are having their own methods. So, testing and maintenance become easier too.
- ✓ It is a kind of modularity approach so one can have better code and understandability.

2.2 UML (Unified Modelling Language) Diagrams:

Object oriented designing was growing at the rapid rate in late eighties and early nineties. Many UML were proposed and accepted at that time. All the techniques and approaches were having completely or partially different way of generating models. Even their notations were not similar as well. Because of so many alternatives and diversity in the notations used by various approaches, a lot of confusion was raised for document development especially OOAD documents. UML is considered to be a language to design models for application. Unlike other languages, UML has symbols to represent and rules to prepare sentences for modelling. UML is not used to design methodology. It can only be used to design models. Once, a methodology is adopted, it is followed for all projects. Due to this practise in older times, it was difficult for users to move to some other methodology. Every methodology was having their own

notations to design model. This problem was rightly addressed by the developers of UML. They kept all the notations independent of other methodologies. UML is different than Booch's methodology, OMT, etc. These were few well known modelling methodologies. They provided the methodology called unified process. Solution might be designed using any methodology. But, we can apply UML to every solution.

What is model?

A model is an abstraction of a real problem (or situation), and is constructed by leaving out unnecessary details. This is used to reduce the complexity of the system. Apparently, simplicity makes it easy to understand for all. Model mainly focuses on important features of the system. It shows overview of system. It is very much important to construct model to develop a good understanding of any problem.

For example, management wants to start a new course in the college. At the initial level, management will only look forward for syllabus and outline of the course. Management will never fall into the depth of the execution such as how to run the course, how many seats should be there and all. Initial thought creation process will have an outline. This is what modelling is. As we model solution of our day-to-day problems, we need to design model for application.

Why construct a model?

An important reason behind constructing a model is that it helps to manage the complexity. Another reason is the reduction in cost and providing good solution. Once models of a system have been constructed, these can be used for following purposes:

- Analysis
- Specification
- Design
- Coding
- Understanding of an implementation
- Testing

A model can be used for demonstrating any of the above mentioned process. So, model designed in the Analysis phase is different from the model of Design and coding.

2.2.1 UML Diagrams:

UML diagrams can be created with various perspectives. Each perspective focus on specific aspect and ignores all other aspects. Various models are created because single model make it very much complex to understand and use.

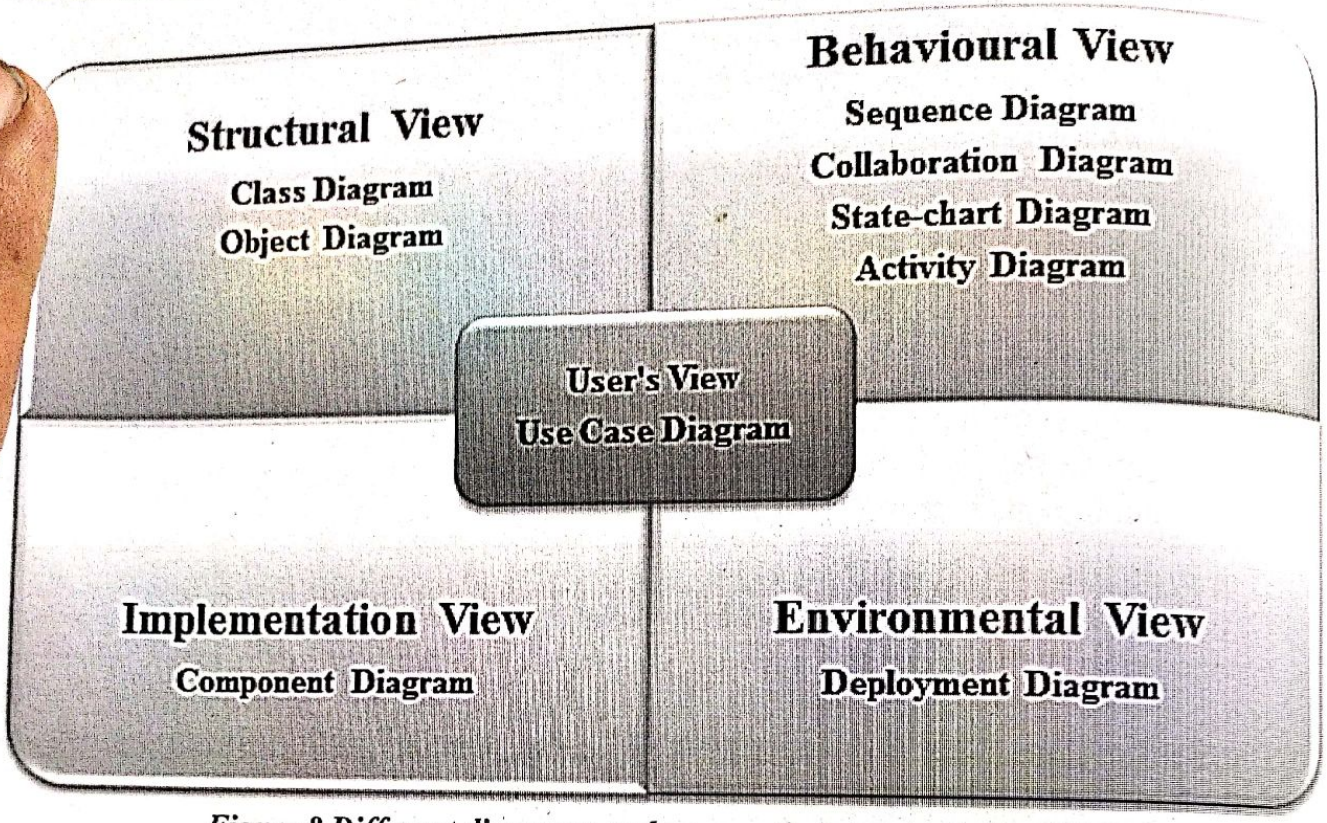


Figure 8 Different diagrams and perspectives supported un UML

1. **User's View:**
   User's view only emphases on functionalities of the system. All the functionalities are captured and demonstrated in user's view. It is a kind of overview of the system from where internal structure is hidden. As shown in the above figure, user view is the central view which is confirmed by all other views.

2. **Structural View:**
   Structural View clears the structure of the problem. This is why objects and classes are mainly handle and described in this view. It is not as dynamic as other models. The structure of the system is generally, not changed and this is why it is refereed to be a static model.

3. **Behavioural View:**
   As the name show, it is the interaction of the objects to realize the behaviour of the system. Hence, this is a dynamic structure in general.

4. **Implementation View:**
   Implementation view will focus on the actual communication between components. Components like GUI, middleware, database, etc. are interdependent on each other. This dependency can be shown using implementation view.

5. **Environmental View:**

Environment is always the infrastructure of the system such as hardware. A system must have a view to demonstrate how system components are implemented on various environments.

2.2.2 Use Case Diagram:

It is consisting of various use cases of a system. It is the representation of how a system can be used in actual situation. For example, if we think of developing learning management system for students and professor, ***"the question is what all users can do?"*** Use cases for such system can be as follows:

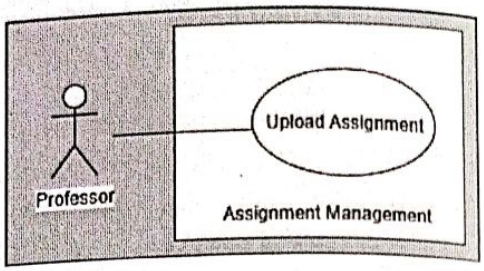
- Mark attendance
- Give Assignment
- Upload Reading Material
- Accept solution of assignment
- Evaluate / Assess
- Score calculation

The use case will not consider the internal structure to develop above mentioned functionalities. Use case will divide the functionalities in to the transactions or micro tasks. It won't include any algorithm for any functionality. It will only use a sequence of interactions between user and the system. System may have variation in the sequence of such transactions. This is called alternate sequences. Alternate sequence may occur due to multiple ways of generating a situation. Such as Assignment file size is an issue in uploading, file type is another issue, etc. All such will lead to same situation which is failure of assignment submission. The mainline sequence and all such alternate sequences corresponding to the call of use-cases are called a scenario. A use case is nothing but a set of all such related scenarios bound together by a single goal. Use case model represents functional or process model only.

2.2.3 Representation of Use Cases:

A use case model can be created by drawing use case diagram and by writing textual description of the same. Following symbols are used to draw Use case diagram.

Symbol	Usage	Example
Ellipse	It is used to represent any use case. Name / title of the use case will be written within the ellipse.	For example, uploading an assignment is a use case related to professor.

Rectangle	It is used to represent system boundary. All the use cases are encompassed within a rectangle. Name of the system is mentioned inside the rectangle. It shows scope of the system.	For the same example mentioned above, we can draw like: 
Actor **Stick Person**	It is used to represent the user of the use case. It is advised to consider the roles as an actor of the system. An actor may participate in more than one use case. A single user may play role of multiple actors as well. Actor can be a human or external system as well. An external system is generally represented as ***<<external system>>***.	For example as shown in the above rectangle, professor is the actor doing the act of uploading assignment.

The basic idea behind having minimal number of icon is to set a stereotype. The more icons a diagram has, the more confusion it creates. The best example to understand this is the usage of stick person symbol. Even external systems are represented with stick person only. IF specific symbols are generated for external systems, new symbols were required to be used and mentioned. Usage of common and less number of symbols can certainly give easiness in leering as well as building UML.

Each ellipse shows a use case. It has some text included within the symbol itself but it does not have complete information. Text Description is used to provide complete description of functionality, how user interacts with the system, etc. It should address minute level details of mainline sequence as well as alternate sequences. It must include exceptional conditions, system responses, etc. One can use following structure to prepare a text description.

1. ***Contact Person:***
   This segment contains information about the stake holders with whom use cases are discussed and defined. It will have minutes of the meeting as well.
2. ***Actor***
   In this information about various actors are included. This is significant because it will help in implementation of use cases of various actors.
3. ***Preconditions***
   This includes information about the conditions which are there before the actual execution of the use case starts.

4. ***Post-conditions***
   This collects information about the state of the system after successful execution of use case.
5. ***Non-functional requirements***
   Non-functional requirements such as environment, infrastructure for the implementation of the code, implementation for the execution of the use case, etc.
6. ***Exceptions and error situations***
   These audits the error related to the domain. For example, professor doesn't have rights to upload assignment. This is domain related error not a software related error.
7. ***Sample dialogs***
   This provides examples to illustrate use cases.
8. ***Precise user interface requirements***
   It is as per the name. It requires and stores information about the interfaces such as forms, dialogs, UX UI.
9. ***Document references***
   This is the document to understand the system functionalities or operations.

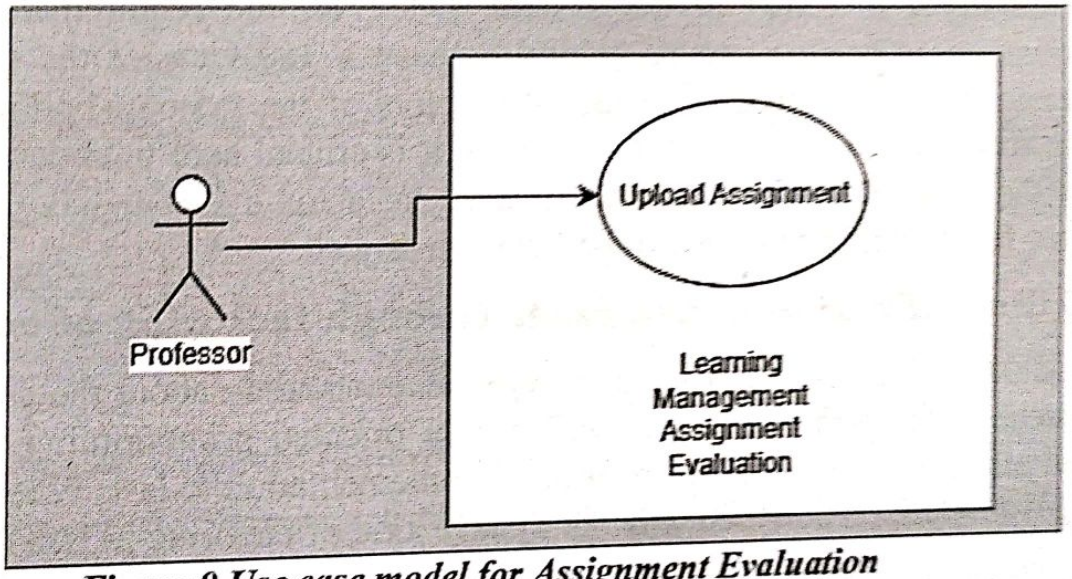


Figure 9 Use case model for Assignment Evaluation

Text description:

U1: Use case for Professor upload assignment:

1. **Scenario 1: Mainline Sequence**
   1. Professor: Selects upload assignment option
   2. System: Upload Document file and enter title of the assignment
   3. Professor: Provide file and title
   4. System: Display message that assignment **is uploaded successfully**
2. **Scenario 2: At step 4 of mainline sequence**
   4. System: Display message that valid file is not uploaded
3. **Scenario 3: At step 4 of mainline sequence**
   4. System: Display message that title is not entered.

2.2.4 Significance of Use Case Diagram:

- Text description with ellipse in Use Case Diagram provides complete requirement specification for the system.
- As mentioned earlier, Use Case Diagram provides a core model which must be conformed and followed by all other models.
- If specific authentication mechanism is developed, actor based use case will provide more clear understanding about the functionalities.
- Every use case will provide efficient and specific information about the User Interface design as well.
- Use case with text description includes all the detail information about the functionalities. So, it becomes very much useful in developing Help manual.
- All the identified actors also help in understanding the functionality requirements and clear-cut functioning of the system.

2.2.5 How can one identify Use Cases of a system?

Rigorous study of the Software Requirement Specification document is the main task to identify Use Cases of the system. Once, all the requirements are cleared, a brainstorming session can be helpful to have a 360 view of the system. The most commonly used method is to identify Actors of the system. Following this, all the functionalities of each actor is identified. It is crucial here to understand that actor is not the initiator of functionality always. An actor may be indirectly associated to some functionality as actor can be a system as well.

2.2.6 Simplification of use cases through factoring of commonality:

Modularization always provide in depth understanding about the components of any system. This works very well with object oriented programming. As every logical design is converted to product through implementation at last. When we are going to apply modularization then why not to have kind of factoring in Use Case Diagrams.

- Complex Use Cases are needed to be factorized into simple Use Cases very first. Tracing a Use Case would become very easy if it is simple and factored.
- In addition to this, we can reuse common functionalities of a Use Case. We can define functionality through a Use Case once and reuse it as and when required.
- This will also help in developing class diagram easily. As Actors are having complete and discrete description of their actions; including methods and data in class diagram will be swift and faster.

We need to keep in mind that breaking of Use Cases is not sensible always just have advantages mentioned above. We need to understand the pros and cons first and then make a wise decision. Following are three approaches for factoring Use Case.

- **Generalization:**

Generalization is used in Use Case when two Use Cases have similar functionalities with minor difference. The working would be the same but the approaches would be different. It can also be understood as a kind of inheritance we see in classes. A parent class can be extended by two or more classes which will have few general features form parent class. Child classes will have their own features as well. It can be represented as follows:

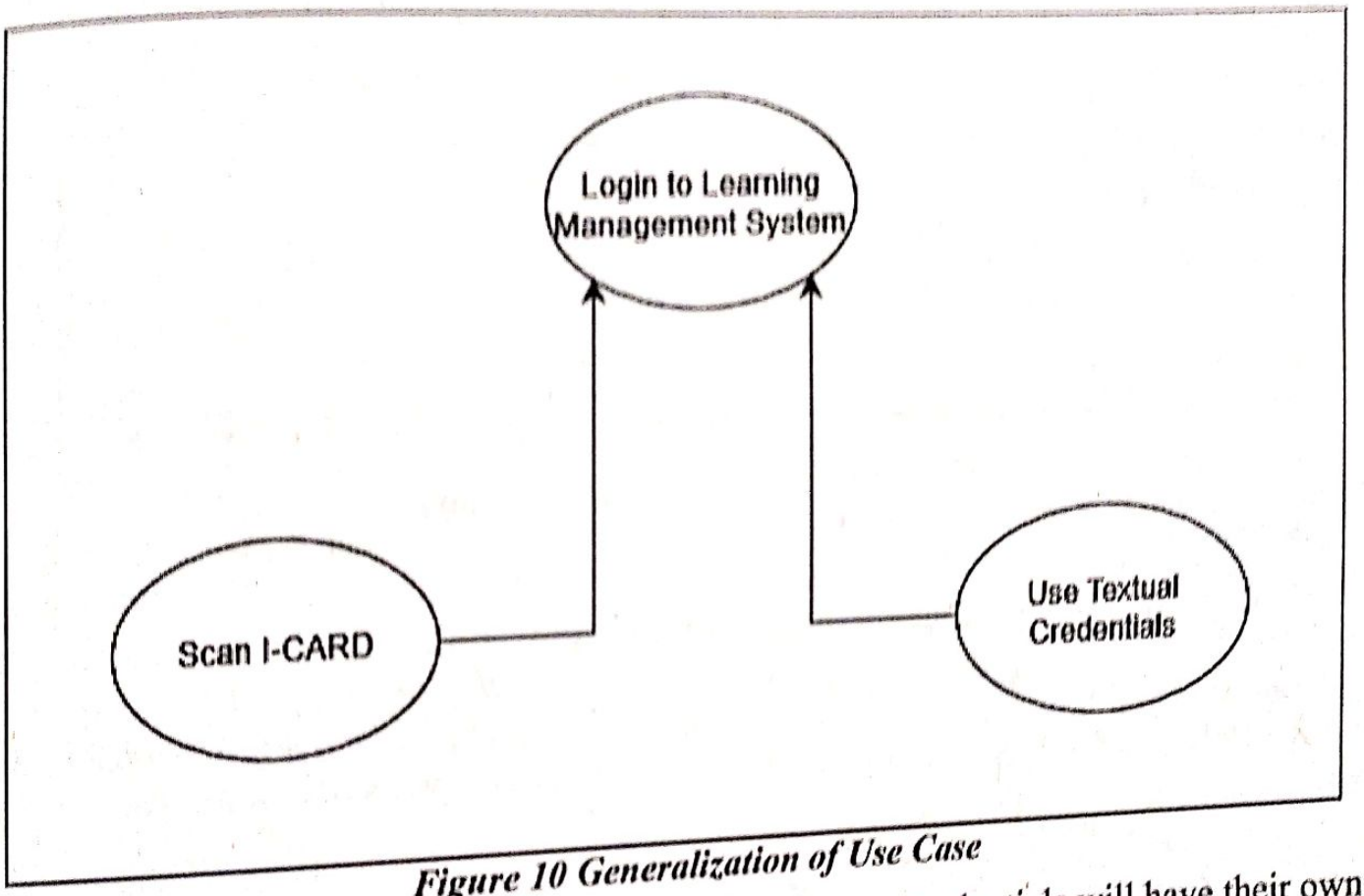


Figure 10 Generalization of Use Case

Here, both uses cases Scan I-Card and Use Textual Credentials will have their own text descriptions.

A. **Includes:**

Include relationship is showing the inclusive features of a Use Case. A Use Case may be containing sequence of steps involved as we have seen in text description. This can be represented using ***<<includes>>***. It will automatically or manually show inclusion of certain use case in base use case. For example, uploading a document can be done after logging to the system only. This means login process is by default included in the uploading a document use case.

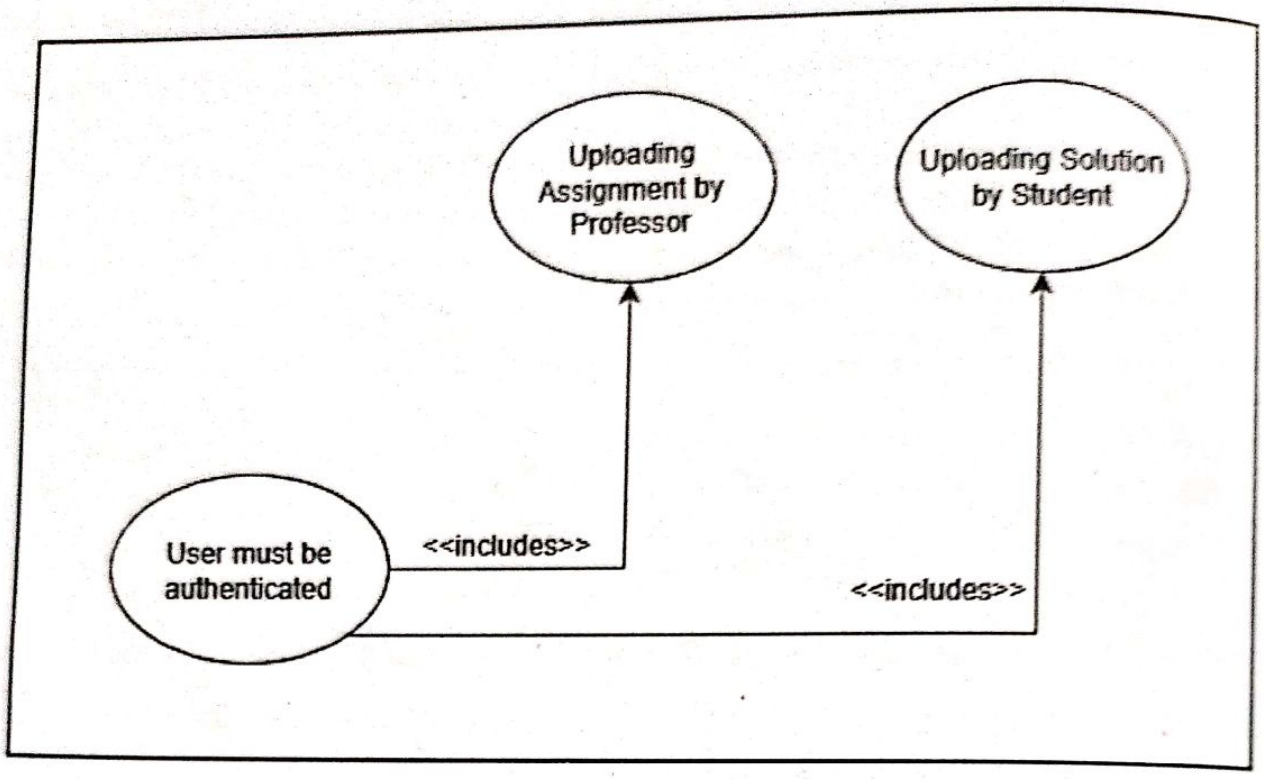


Figure 11 Use Case Inclusion Example

B. **Extends:**

Extend relationship is nearly similar to Generalization. But, this focuses in adding new features to the use case in certain condition. Whenever a variation is occurring into any mainline sequence, there are chances of generating extend relationship. It can be clearly understood that it is used to collect alternate paths or conditions.

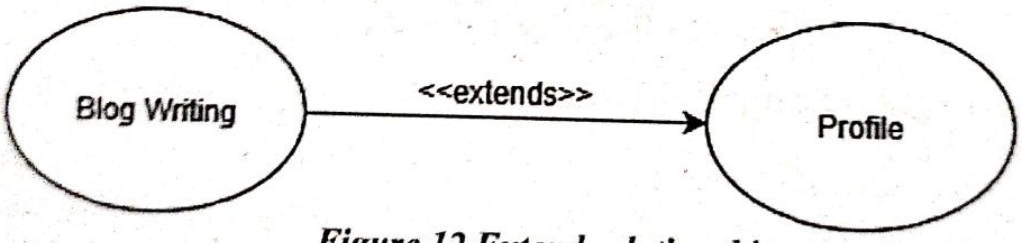


Figure 12 Extend relationship

2.2.7 Organization of Use Cases:

Whenever Use Cases are factored, they are managed hierarchically. Complex Use Cases are broken down to simple Use Cases. This is why we have level based hierarchy in the Use Case Model. A Use Case can have one or more subsystems / Use Cases. In the top level Use Case Model, only fundamental information is used. This is why it is called Context Level Use Case. In this, only external users will interact. Subsystem of this context level Use Cases provide functionalities of context level functions. As in context level, functionality limits are emphasized. In the lowest level Use Case Hierarchy, methods are defined.

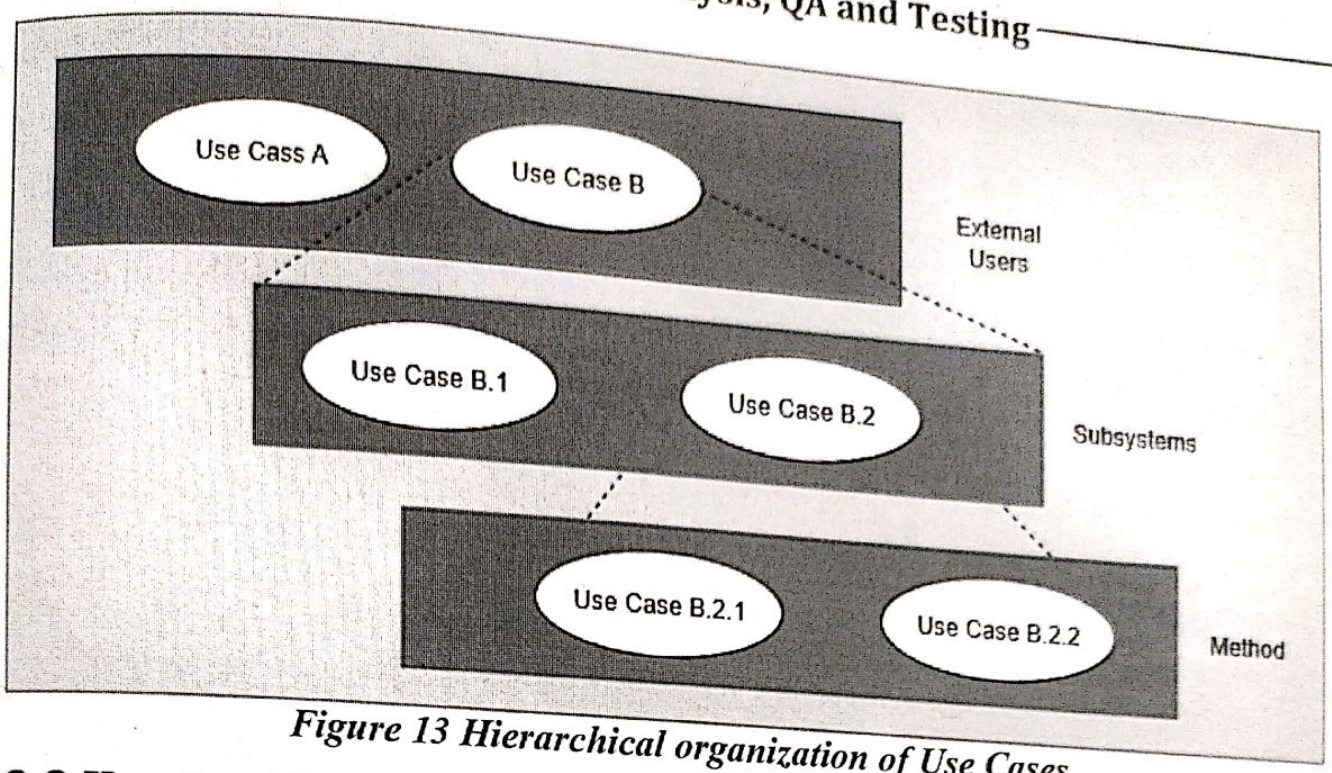


Figure 13 Hierarchical organization of Use Cases

2.2.8 Use Case Packaging:

Packaging is used to managing complexity of the system. We can create packages by breaking large systems and prepare a combined folder kind Use Case with all packages.

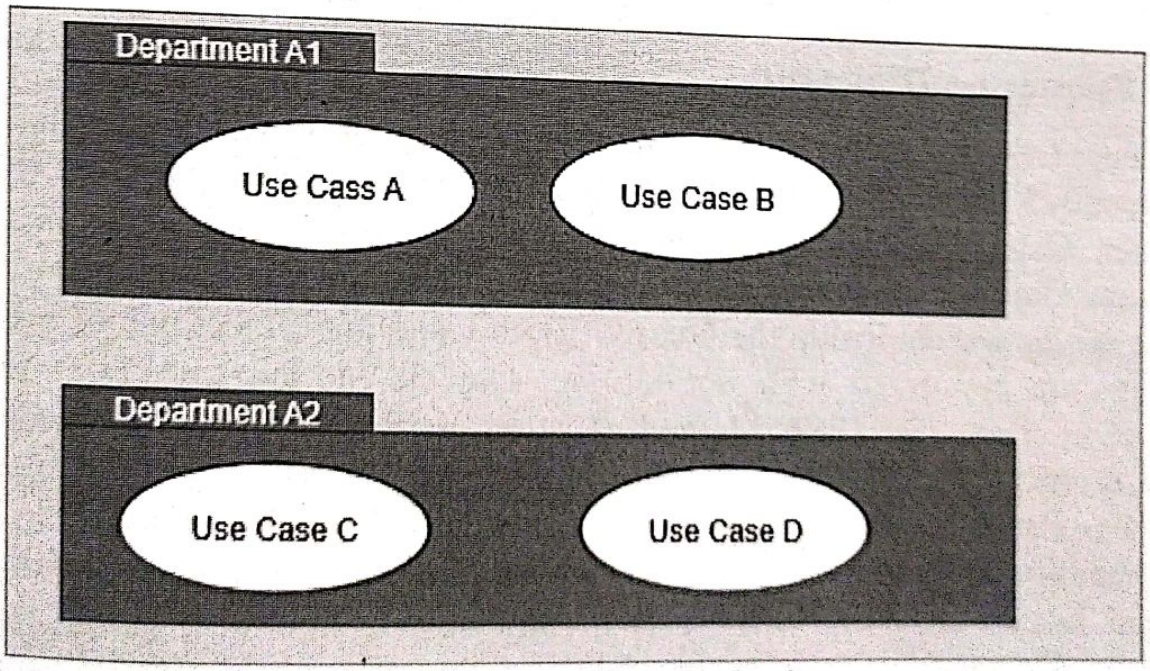


Figure 14 Use Case Packaging

2.2.9 Interaction Diagram:

Sequence Diagram is considered under an interaction diagram. Every functionality in OOAD approach is supposed to be implemented using objects. Whenever any Use Case is executed or any functionality is used, Objects are required to communicate with within objects. Interactions are done using message passing mechanism.

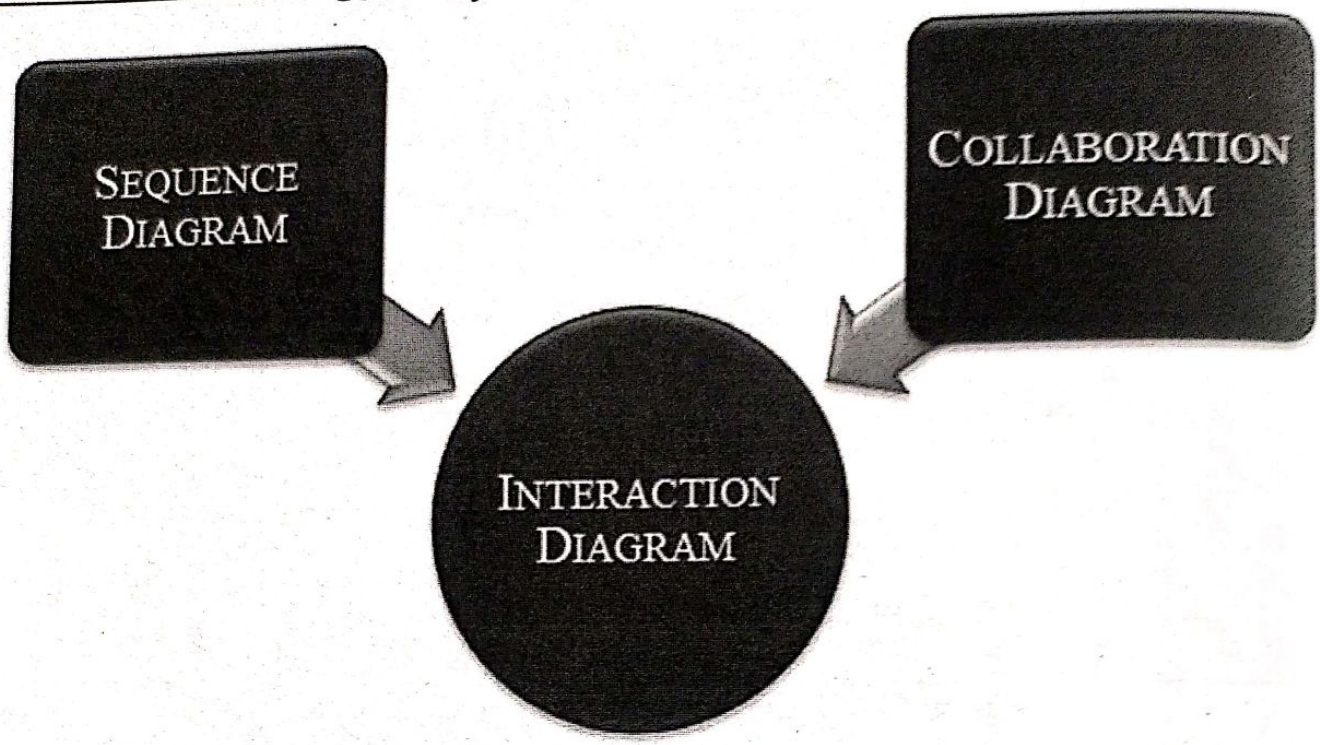


Figure 15 Interaction Diagram

Above mentioned diagram shows that interaction diagrams are of two kinds: Sequence Diagram and Collaboration Diagram. One diagram can automatically deduce another one. But, they both are still useful. Interaction diagrams play a vital role in defining effective OOAD.

2.2.10 Sequence Diagram:

Sequence Diagram shows interaction among objects as a two-dimensional chart. Flow is described from top to bottom. Sequence Diagram is prepared in the following manner. Objects are shown in the rectangle box in the top most position of the diagram. Object name is written using ":" Colon sign before name of the object. Class name can be written inside the box too. Both Class and object names are underlined. Objects are connected with dotted line. Objects, at the top, exist regardless of execution of any Use Case. Method calls are generated during the execution of Use Case. Vertical Dashed line as shown in below diagram is called the Object's lifeline. This line shows the existence of an object. If dashed line is not available after certain functionality, it means object doesn't exist anymore. Rectangle is drawn on the dashed line to show the activeness of any object. It is also called the activation symbol. Message passing is shown using an arrow between two objects. Each message is written over arrow. Even few extra information is covered as well. This is called controlling information. They are of two types: A condition which is shown using rectangle braces "[ ]". Indication of messages between objects is represented in chronological order.

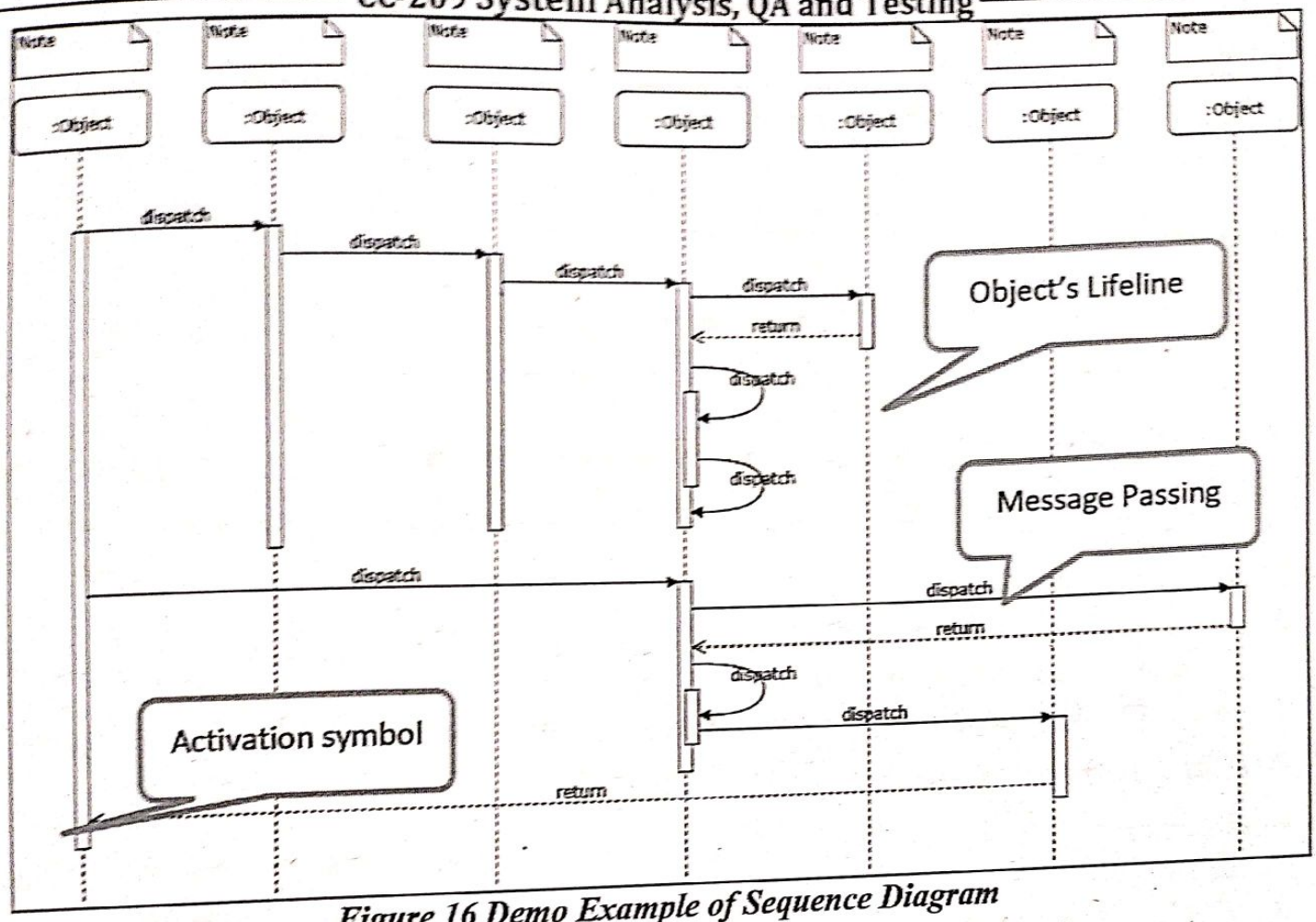


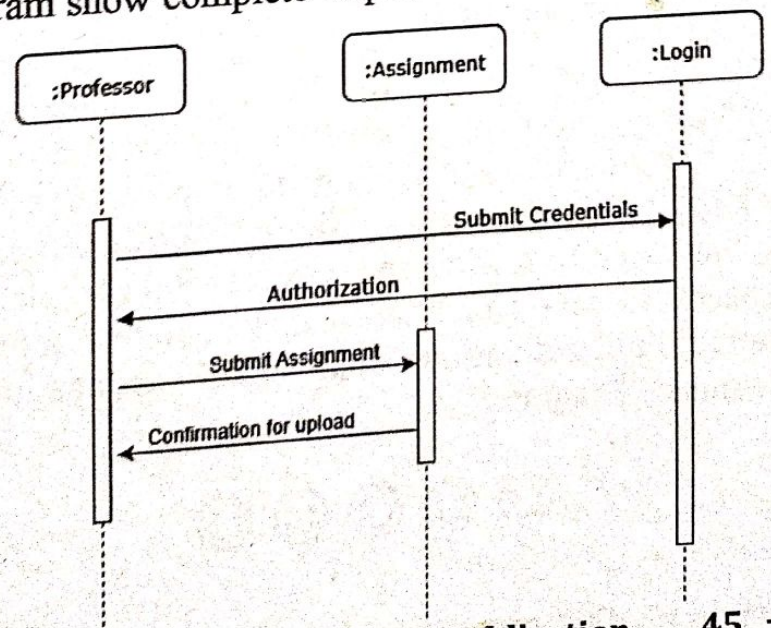
Figure 16 Demo Example of Sequence Diagram

Sequence Diagram Case Study:

As mentioned earlier in Use Case Diagram, a professor can upload assignment. But, this functionality will require multiple steps. Here, assume that Assignment, Professor, Login are the objects. Professor will first login into system using login object. Once, authentication is successful, he will upload assignment.

Relationship between UML and Sequence Diagram:

✓ Sequence diagram show complete sequence of micro tasks while UML show abstract



view.

- ✓ Sequence Diagram does not include aggregation, generalization, etc. It is mainly related to behavioural view of the application.

Exercises

State whether following statements are true or false:

1. Object Oriented Programming paradigms are accepted for Object Oriented Analysis and Design. – ***TRUE***
2. Class is an Abstract Data Type. – ***TRUE***
3. Derived Class will have features from Base Class only. – ***FALSE***
4. A class cannot inherit more than one class. – ***FALSE***
5. UML model is supposed to be followed by all other models and diagrams. – ***TRUE***

Answer following questions:

1. What are various relationships that may exist among classes? Provide suitable example.
2. Explain Association and Link in detail.
3. What is the meaning of Abstract Class?
4. Why Object Oriented Analysis and Design is more useful than traditional approach?
5. What is stereotype in UML?
6. Which are various Interaction Diagrams?
7. Which are the User Views of Use Case?

Prepare Complete Use Case Diagram and Sequence Diagram for following system requirements:

There are around 1000 speeches recorded by various lecturers. Whenever, a lecturer delivers lecture, he may speak things which are not covered in books. This is because lecture deliver is instantaneous and depends on experience as well. Students want to refer every work of the lecturer but they are not able as recording of speeches are in audio form and they need to rewind and play, rewind and play. This problem can be solved using speech to text converted and a complete reference material can be prepared and stored. Find out various Actors and Functionalities to prepare Use Case Diagram and Sequence Diagram/

Unit -3

UML Diagrams

- **Activity Diagram**
  - **Elements of Activity Diagram**
  - **The following shows example of an activity diagram**
- **Class Diagram**
  - **Elements of Class diagram**
  - **The following shows example of a class diagram**

Unit-3 UML Diagrams

3.1 Activity Diagram:

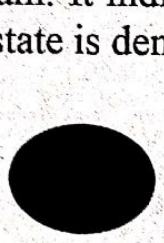
An activity diagram is used for displaying the logic used by a single use-case. It is used to define a specific actor's workflow within the entire system. An activity diagram shows all the sequence flows in the activity. It is a type of dynamic diagram that shows activity in the particular state.

3.1.1 Elements of Activity Diagram:

Activity diagram has following elements that are used to construct an activity diagram.

1) Initial State:

An initial state defines the starting state of an activity diagram. It shows the beginning of the workflow on an activity diagram. It indicates starting point of the sequence of activity using initial state. An initial state is denoted by "Solid circle" with an optional name or label.



Initial State

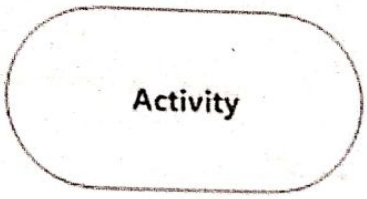
2) Final State:

A final state shows end of the workflow within the activity diagram. It is drawn as a filled circle within the larger unfilled circle called 'bull's eye' with an optional name or a label.

Final State

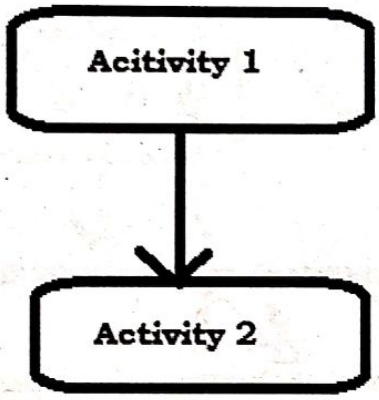
3) Action/ Activity:

An action state represents various non-interruptible actions which can't be further decomposed. The activity state may be interrupted and can be further decomposed also. An action/activity is denoted by a rounded rectangle with a name of activity within it.



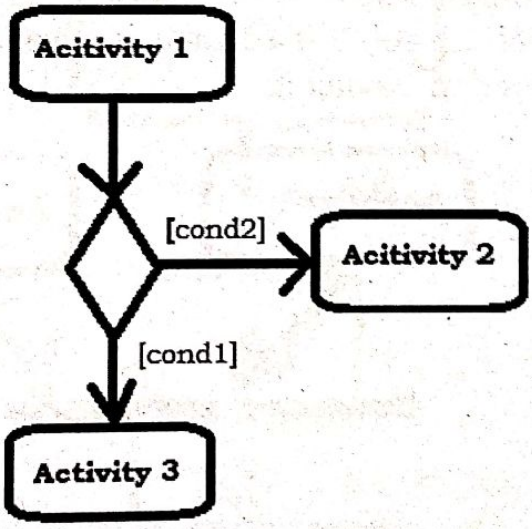
4) Transitions:

The transition connects various elements of activity diagram. It represents a workflow between two or more activities. It is denoted by a "solid line with an open arrowhead". It can have an optional label enclosed in square brackets called "guard condition".



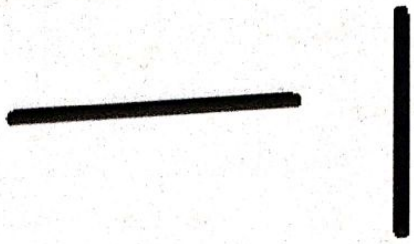
5) Decisions:

A decision has one incoming transition and two or more outgoing transitions based on the result of a guard condition. It is denoted by a diamond shape without any name or label called a decision box.



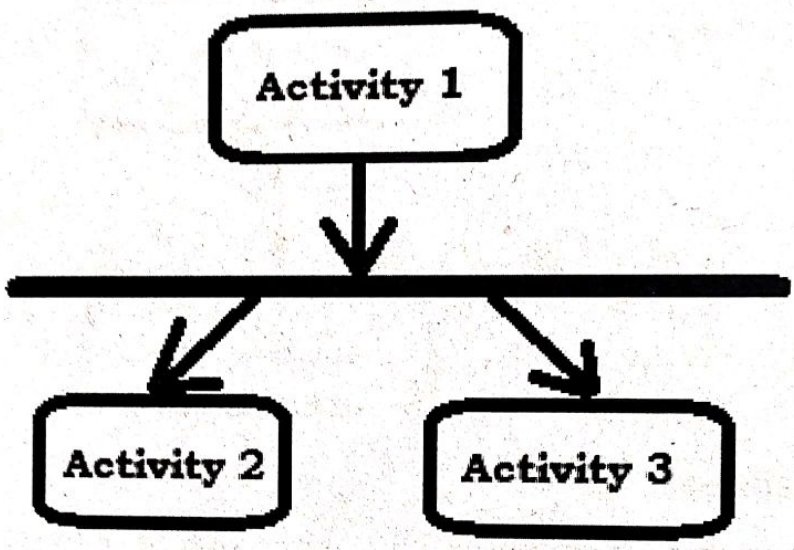
6) Synchronization, Fork and Join:

Synchronization allows simultaneous workflow (Parallel workflow) in an activity diagram. It defines fork and join that represents parallel execution of the workflow. It is drawn as a horizontal or vertical bar with no name or label.



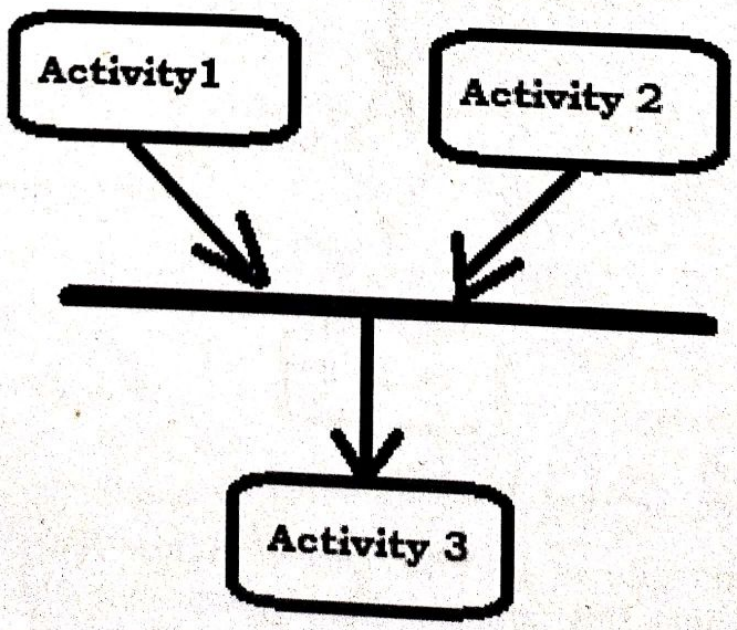
Fork:

A fork is type of synchronization that defines simultaneous workflow in activity. A fork identifies where a single flow of control divides into two or more simultaneous flows. It is denoted as a bar with one transition going in and two or more transitions leaving it.



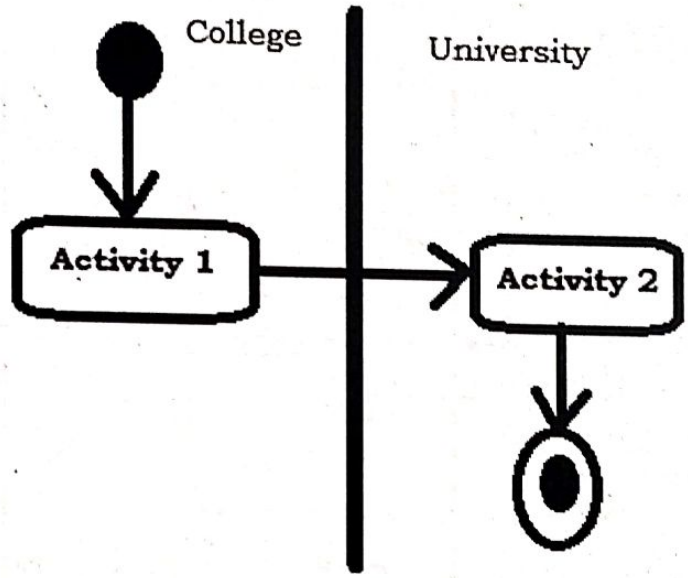
Join:

A join is another type of synchronization that defines simultaneous workflow in activity. A join defines where two or more simultaneous flows of control combine into a single flow of control. It is denoted as a bar with two or more transitions going into it and one transition leaving it.



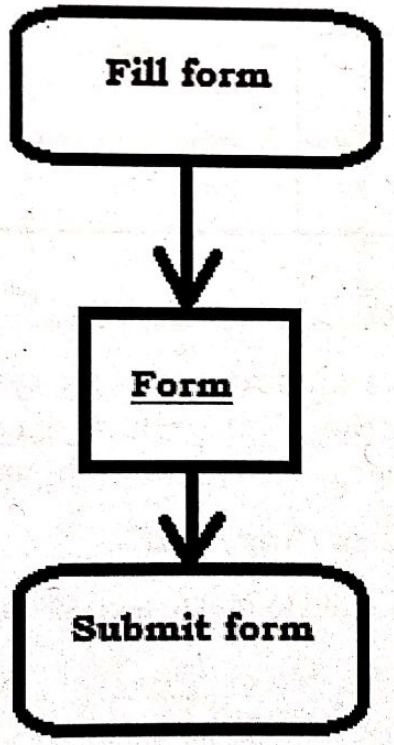
7) Swimlanes:

Swimlanes define who is responsible for carrying out specific activities. It can represent a user, a role or an organizational unit in activity diagram. An action transition takes place between two Swimlanes. They are drawn as vertical columns or horizontal rows with the name associated with each swimlanes. It should be avoided to have more than five Swimlanes in a single activity diagram.



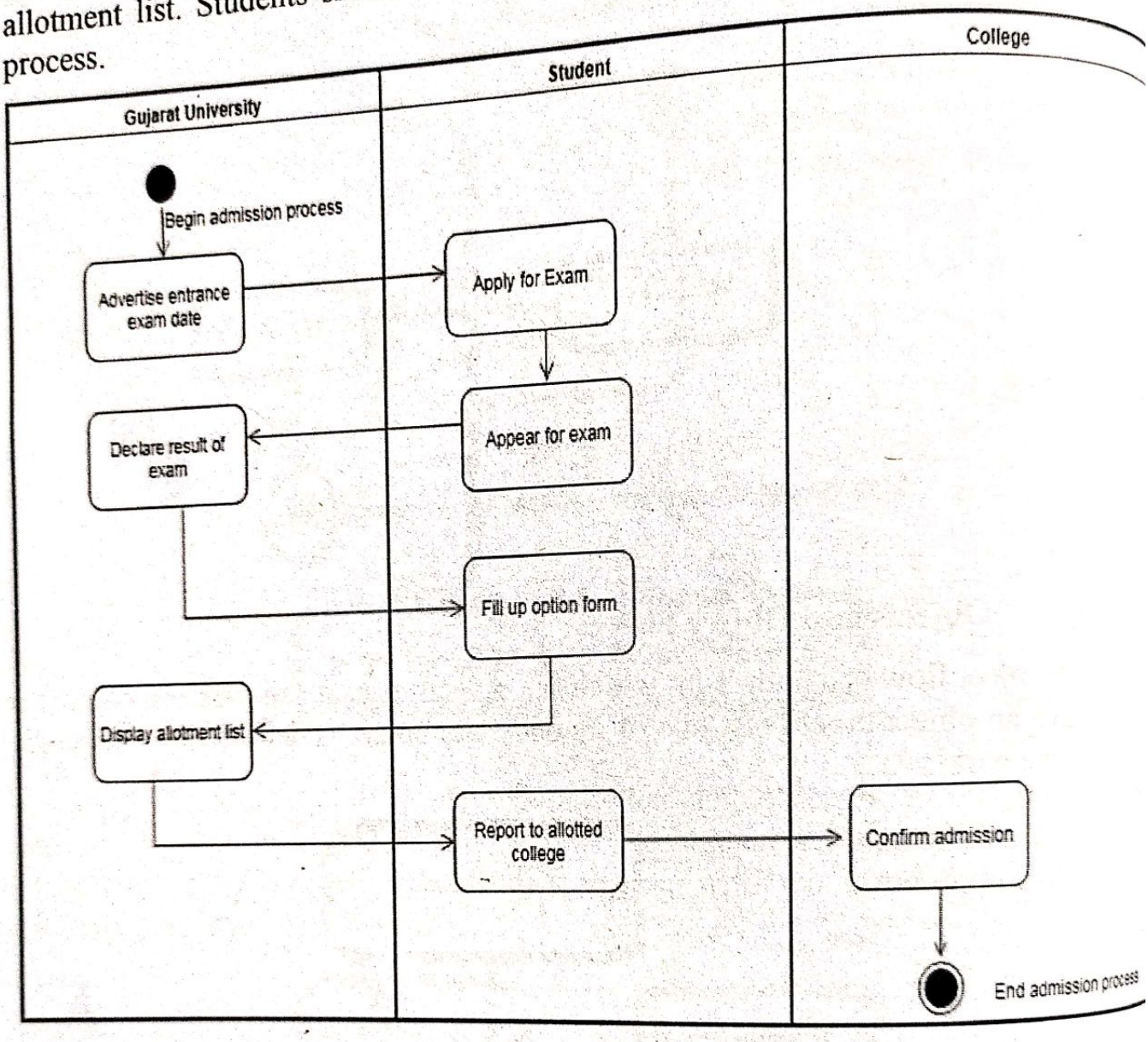
8) Objects and Object Flows:

An object flow is a path from which the objects or data can pass. An object flow must have an object on at least one of its ends. An object is denoted by a rectangle with a name within it.



3.1.2 The following shows example of an activity diagram:

Gujarat University advertises the date of MCA entrance exam. The students have to apply for the exam. The results are declared by Gujarat University. Students have to fill up the option form to select college of his/her choice. Gujarat University displays allotment list. Students should report the allotted colleges and complete admission process.



3.2 Class Diagram:

A class diagram shows static structure of the system. Class diagram represents classes and relationship between them. Class diagrams are used to understand the requirements and detailed design of the proposed system.

3.2.1 Elements of Class diagram:

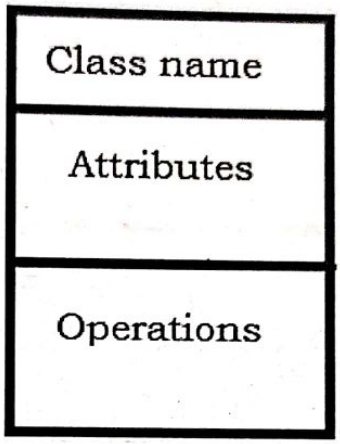
Following elements are used to create a class diagram.

- Class
- Relationship
- Association class

- Interface
- Package

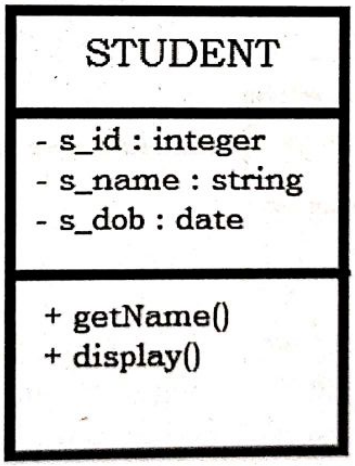
1) Class:

A class is collection of attributes and operations and relationship with other classes. A class is divided into 3 parts; the topmost part contains a class name. The middle part contains list of attributes and the last part contains list of operations. A class is denoted as,



Class is a noun in problem statement. Attributes are used as adverbs or adjectives. Operations correspond to methods for managing value of attributes.

Example:



2) Relationship:

A relationship is connection between two or more classes. There are various types of relationships used in class diagram.

- Association
- Generalization
- Aggregation
- Composition
- Dependency
- Multiplicity

a) **Association:**

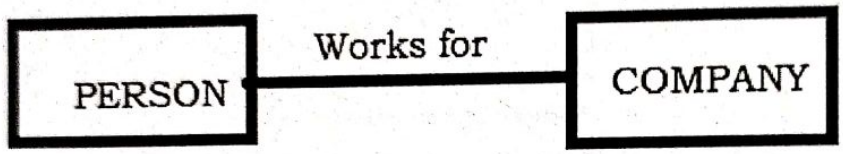
An association represents a conceptual connection between two or more classes. Association is the "glue" that ties a system and classes together. Association is denoted by a solid line.

There are three types of association.

- **Bidirectional Association:**

In bidirectional association, both the classes are aware of each other and their relationship. It is indicated by a solid line between two classes.

Example:



- **Unidirectional Association:**

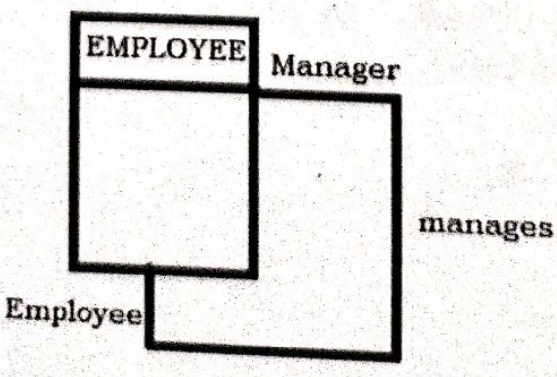
In unidirectional association, two classes are connected but only one class knows that the relationship exists. Unidirectional association is denoted by the solid line with an open arrowhead.

Example:

- **Reflexive Association:**

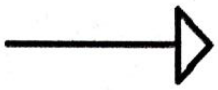
When a class is associated with itself then it is called reflexive association. It means that an instance of the class is related to another instance of the same class. Each connection of an association is called association end. Association end names are optional.

Example:

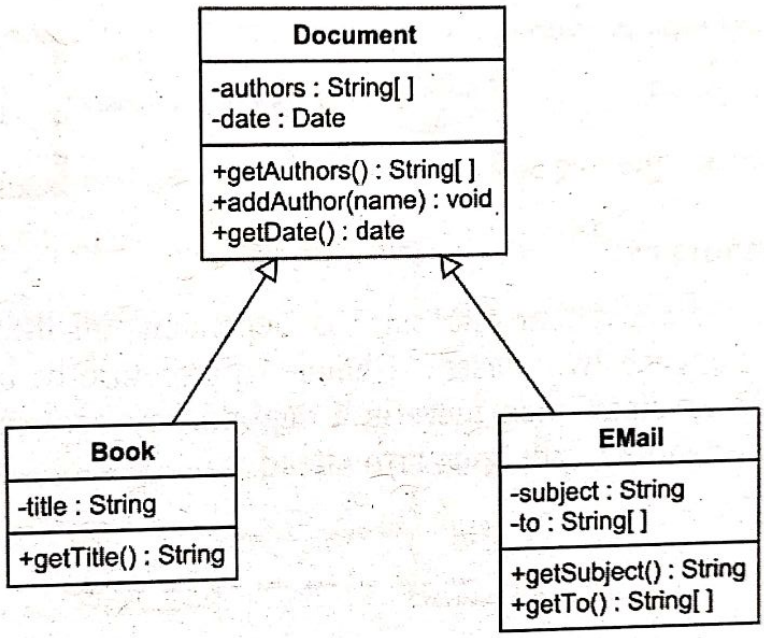


b) Generalization:

Generalization is a relationship between a super class and one or more sub classes. It is used to indicate inheritance. Generalization is also known as "is-a" relationship. It is a bottom-up approach in which two or more entities can be generalized to a higher level entity if they have some attributes in common. It is used to extract common properties from a set of classes and create a generalized class from it. It is denoted by a solid line with hollow arrowhead.

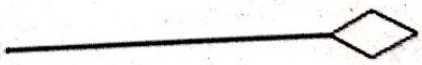


Example:

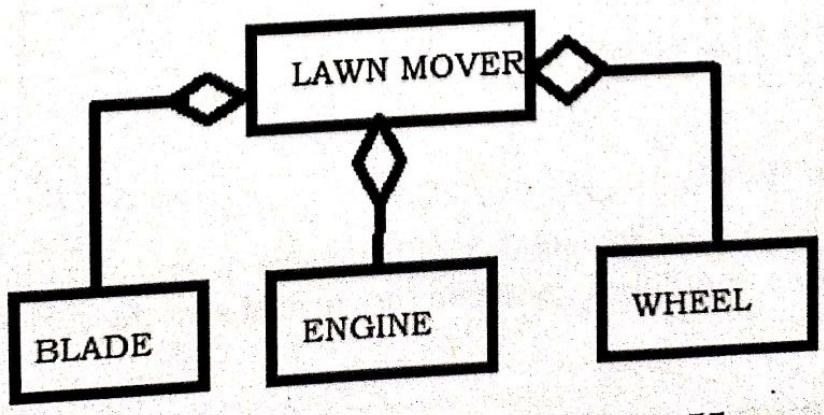


c) Aggregation:

Aggregation is a kind of association that represents a part-whole or part-of relationship. When a class is formed as a collection of other classes then it is called an aggregation. It is also known as "has-a" relationship. It is denoted by a solid line with hollow diamond which indicates the assembly end.



Example:



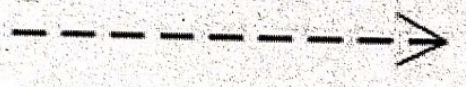
d) Composition:

Composition is a stronger form of aggregation. It is used to allocate and release memory. It has two constraints. A constituent part can belong to at most one assembly. When an object contains the other object, if the contained object cannot exist without the existence of container object, then it is called composition. It is denoted by black diamond. The arrowhead on the other end of the relationship denotes the relationship is possible in only one direction.

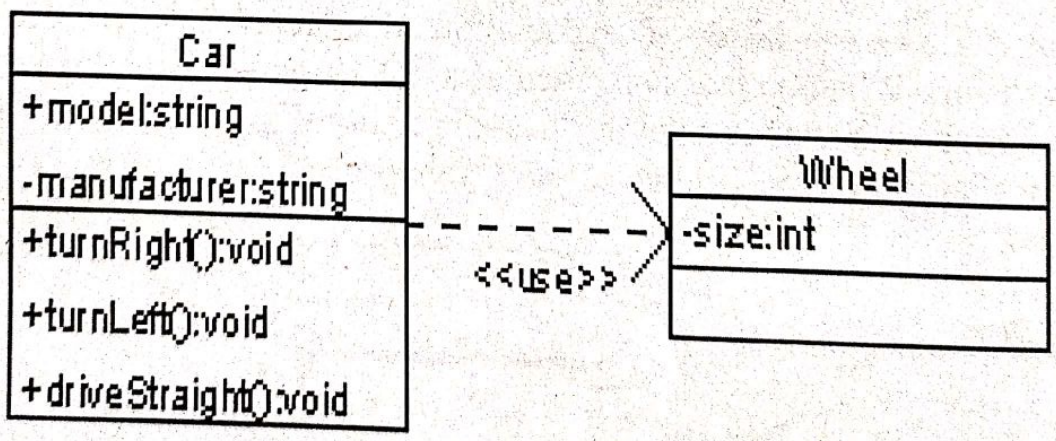
Example:

e) Dependency:

Dependency indicated that one class is dependent on the other class. It is a weak relationship between two classes. Change in structure or behavior of a class affects the other related class, then there is a dependency between those two classes. It is denoted by dotted line with open arrowhead.



Example:

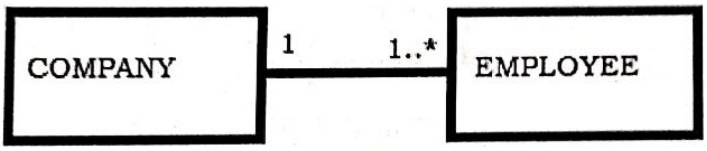


f) Multiplicity:

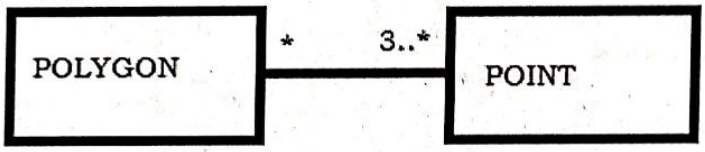
Multiplicity indicates the number of instances of one class linked to one instance of the other class. Multiplicity notations are placed near the ends of an association.

Indicator	Description
1	Only one
0..1	Zero or one
0..*	Zero or more
*	Many
1..*	One or more
4	Four only
0..5	Zero to five
5..16	Five to sixteen
3,5	Three or five

Example:
One company has one or more employees.



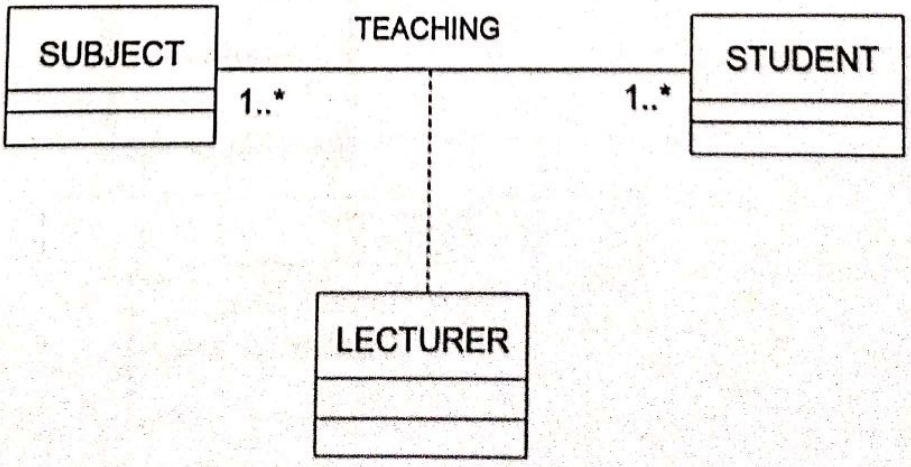
The number of polygons has 3 or more vertices.



3) Association Class:

An association class is a class that is part of an association relationship between two other classes. You can attach an association class to an association relationship to provide additional information about the relationship. An association class is identical to other classes and can contain operations, attributes, as well as other associations. Association classes are denoted as a class with a dashed line to the association line.

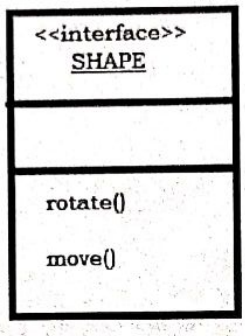
Example:



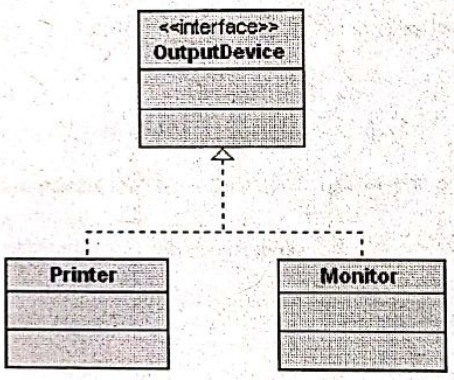
4) Interface:

An interface provides only a definition of business functionality of a system. An interface shares the same features as a class like attributes and operations. The methods are only declared in the interface and will be implemented by the class that implements interface.

An interface is denoted same as a class except that it has a special denotation called a stereotype (<< >>). The stereotype indicates that the class is an interface. An interface does not have member variables.

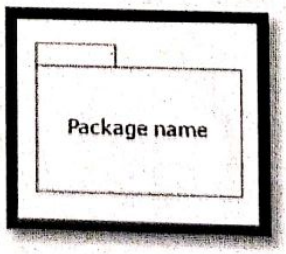


Example:

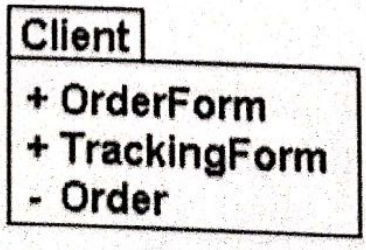


5) Package:

A package is a UML construct that enables you to organize model elements, such as use cases or classes, into groups. A package provides ability to group together classes or interfaces or both that is related. A package can have association with other packages also.

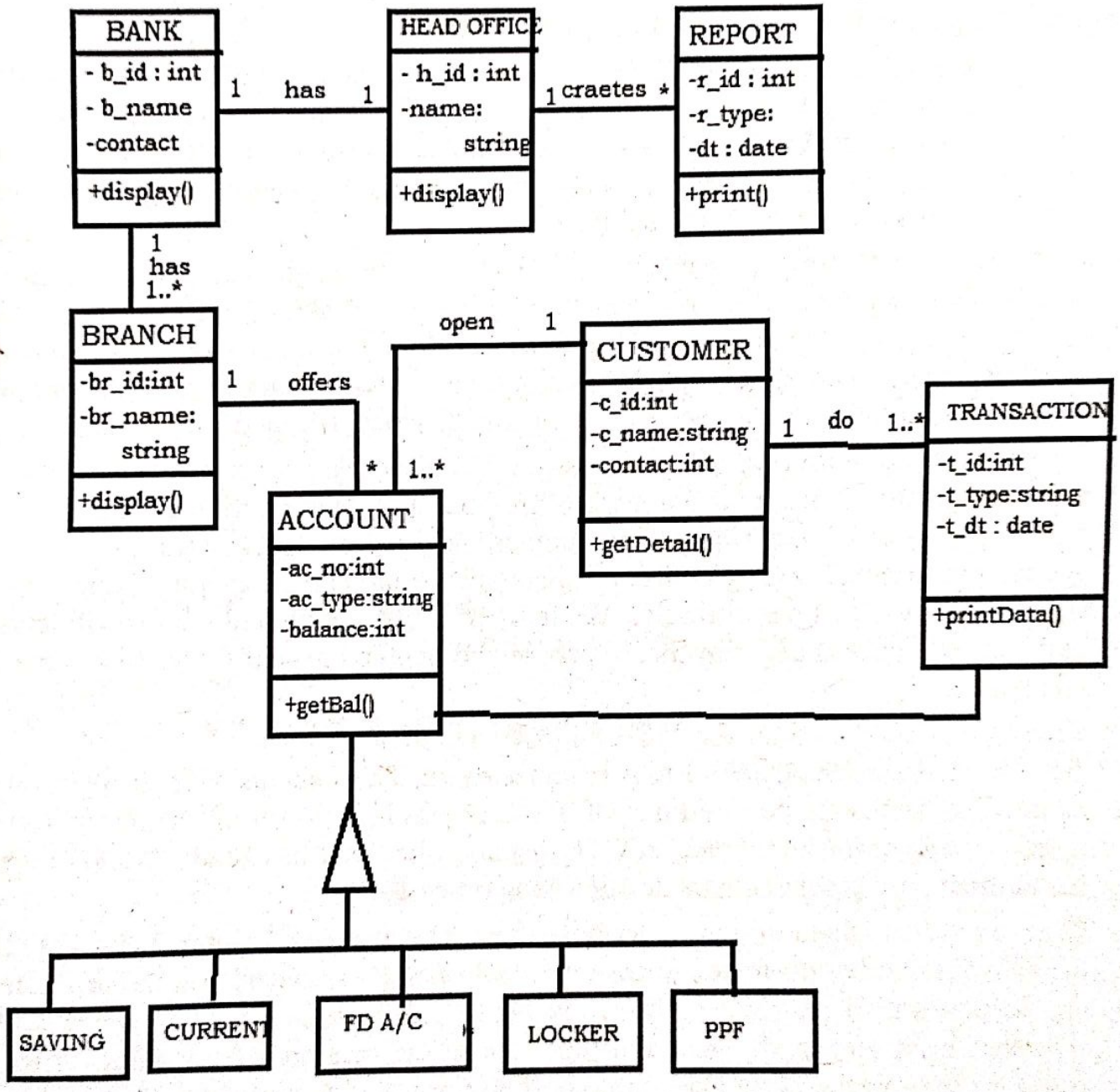


Example:



3.2.2 The following shows example of a class diagram:

A corporate bank has its head office in Mumbai and its branches in several parts of India. The bank has several types of accounts like saving, current, fixed deposits, locker, PPF and many others. The customers can open any number of accounts in any branch of the bank. They will be allowed to access account. The customers do various transactions on their account like debiting, crediting, depositing and withdrawing. Bank calculates interest and credits it to the counts of the customers. The head office manages reports from the branches.



Exercises

Answer the following:

1. What is an activity diagram? List and explain elements of it.
2. Explain synchronization, fork, join and swimlanes with example.
3. What is a class diagram? List and explain elements of class diagram.
4. Explain types of relationships in a class diagram.
5. Explain generalization, aggregation and composition with example.

Create diagram for the following case studies:

1. Create a class diagram for an Information Modeling System for a college. In a college there are one or more courses. Each course offers one or more subjects to the students. A subject will be offered by only one course. Each course has instructors and instructors can work for one or more course. Student can enroll in up to 5 subjects and instructors can teach up to 3 subjects.
2. Draw a class diagram for a car rental application. The car rental agency has multiple offices. The customer visits the agency for inquiry and takes a test ride, then selects the car by signing the form. The customer can also book a car through telephone, email and SMS. The agency checks avaibility and gives status to the customer. The billing is done based on the type of vehicle and distance travelled.
3. Create a class diagram for a hockey league. A hockey league is made up of hockey teams. Each hockey team is composed of players. A team has attributes like name and a record. Players have attributes like number and position. Hockey team plays games against each other. Each game has attributes like score and a location. Teams are led by a coach. A coach has attributes like level of accreditation and number of years of experience. Coaches and players are people and people have attributes like name and address.
4. Create an activity diagram for MCA admission system.
   AICTE advertises the date of MCA entrance exam. The students have to apply for the exam. The results are declared by AICTE. Students have to fill up the option form to select college of his/her choice. AICTE displays allotment list. Students should report the allotted colleges and complete admission process.
5. Draw an activity diagram for the purpose of the Open Access Insurance System. They provide automotive insurance to car owners. Initially, customers fill out an insurance application which provides information about customer and his/her vehicles. This information is sent to an agent and then sent to the insurance companies. When the response returns, the agent determines best policy and gives customer a copy of it.
6. Draw a class diagram for a publisher who publishes different books. An author can write different books for the same publisher. A contract is signed between the

publisher and the author. Reports such as the number of books sold, number of complementary copies given, and royalty amount paid to the author are generated from the system.

Do as directed:

1. _____ Diagram shows workflow of the entire system. **(Activity)**
2. _____ divides activity diagram into sections. **(Swimlanes)**
3. An aggregation relationship is a parent child relationship between use cases. [True/ False] **(False)**
4. The purpose of class diagram is to show the static structure of the system being modeled. [True/ False] **(True)**
5. Unidirectional association means both classes are aware of each other and their relationship. [True/ False] **(False)**
6. When a class is associated with itself then it is called _____ association. **(Reflexive)**
7. Activity diagram is used to model the lifeline of an object. [True/ False] **(False)**
8. A _____ relationship is a parent child relationship between classes. **(Generalization)**
9. It should be avoided to have more than five swimlanes in a single activity diagram. [True/False] **(True)**
10. A _____ is collection of attributes and operations. **(Class)**
11. A _____ is used to represent relationship between two classes. **(Association)**
12. Generalization is also called _____ relationship. **("is-a")**
13. A _____ identifies that a single flow of control divides into two or more separate flows. **(Fork)**
14. Aggregation is used to represent partial relationship. [True/ False] **(False)**
15. When using _____ link, the child classes have the option to override the operations of the parent class. **(Generalization)**
16. Aggregation is also called _____ relationship. **("has-a")**
17. A _____ identifies where two or more simultaneous flows of control unite into a single flow of control. **(Join)**
18. There should be more than one initial state in activity diagram. [True/ False] **(False)**
19. A _____ is a weak relationship between two classes. **(Dependency)**
20. _____ represents physical or conceptual connection between two or more classes. **(Association)**

☺☺☺☺☺

Unit -4

QA and Testing

- **Software Quality, Software Quality Assurance and Control**
  - Software Quality Assurance vs. Quality Control
  - Software Errors, Faults and Failures
  - Software Quality Factors
- **Software Testing**
  - Software Testing Objectives
  - Software Testing Classification
  - Integration Testing Strategies
- **System Testing**
  - Performance Testing
- **Regression Testing**
- **Test Metrics and Measurements**
  - Objectives of Software Quality Metrics
  - Software Quality Metrics categories
  - Limitations of Software Metrics

Unit-4 QA and Testing

4.1 Software Quality, Software Quality Assurance and Control:

Software Quality as per IEEE (IEEE, 1991) edition, is the degree to which a system, component or process meet specified documents and customer or user needs or expectations.

The founder of modern quality assurance, Philip B. Crosby and Joseph M. Juran defined Quality with slight different conception of Software Quality.

In 1979, Crosby defined Quality as: ***"Quality means conformance to requirements"***

While Juran in 1988 defined it as: Quality consists of those features which meet the needs of customers and there by provide product satisfaction. He added that Quality consists of freedom from deficiencies.

4.1.1 Software Quality Assurance vs. Quality Control:

Whenever Software Quality term is discussed, Quality Assurance and Quality Control are constantly repeated and used. This creates a myth or confusion that they are synonym. But, these two concepts have discrete and distinct concept.

Software Control is defined by IEEE (IEEE 1991) as ***"a set of activities designed to evaluate the quality of a product developed or manufactured".*** From this definition, we can clearly understand that Software Control mainly focus on cover-up of any product that doesn't qualify the requirements. Hence, Quality Control is conducted before delivering the product to the client. In contrast to this, ***Quality Assurance focuses on minimizing the cost of guaranteeing quality by various activities.*** These activities are performed throughout the development phase. It prevents causes of errors and detection as well as correction of errors.

We can conclude that both the activities serve different activities and Quality control activities are the part of Quality Assurance only.

4.1.2 Software Errors, Faults and Failures:

Test Case 1: A Company Manufactures Software X conducts feedback about their product. The first client gave feedback that they have been using the product since last 3 years and did not find any failure or errors. Another client gave feedback that they purchased the product before 2 months and facing lot many issues and difficulties. It is clearly seen here that the product served an organization successfully for 3 years while struggled in delivering good results for new customers. Is it possible in context of any product? This may happen with updated packages of a software or in the development phase.

The cause of such error can be the programming error or logical error. While developing update for the product, new package may have certain intrinsic error in the semantics or programming error in the development. In designing new requirements or correcting existing errors, new package may result in faulty update. This raise a question whether Software Error is same as Software Fault? The answer is *No*. Software errors may affect the overall performance or may not. All the errors are not supposed to distort the performance. But, there are some errors which will cause improper functioning of the software in specific manner. All the software faults don't lead to software failure. ***Software Failure occurs when the software fault is activated and draws the software towards disrupts the use of software.*** Software errors, fault and failure hierarchy can be understood in the following manner.

➢ Software Errors:

Software Errors effect software quality and lead to poor software quality. We need to understand the root cause of any error with a view to correct them accurately. The error can be because of code, procedure, document or software data. Failure of software depends on the errors. The errors generated due to compiler, operating system, interpreter, automatic software generators, etc. are also considered to be the human errors. Errors depend on the stages of software development life cycle.

Faulty Definition of Requirements:

Requirements specified by the clients may be erroneous themselves. Faults are generated before even building SRS. The reasons may be as followings.

- Requirements defined are the erroneous.
- Client, being a Lehman, does not understand which information is important and which is not. This becomes the prime reason for the absence of significant information.
- Additional information about the requirements which are not required at all in the initial stage. In implementation of this additional requirements, software suffers from quality and may have these errors.
- Inadequate information about certain requirements. For example, Students are allowed to have credit for the extra certificate. But, this information was not mentioned in the requirement specification.

Error I. Client Developer Communication Error:

This error is raised in the early stage of Product Development. Client- Developer communication is one of the vital process. The more misunderstandings are there in the communication; the more communication errors will be there. Followings can be the reasons for this error.

- Inappropriate understanding of the requirements specified in the System Requirements Specifications.

- It may be possible that certain instructions are given to the developer orally. Any oral communication is not fruitful and produces Communication Error.
- After delivering an increment, client may give some updates. If these updates are not informed to the developer properly, errors may be generated. This error generally occurs in the development phase.

Error II. Deviations from Software Requirement:

In certain circumstances, developer may deviate from the requirement specifications mentioned in the SRS. This error is generated because of course of actioned mentioned below:

- For the rapid development, developers generally use the codes or modules implemented earlier. While updating the existing modules for running projects, developer may have errors. This is the error generated due to deliberate deviations.
- Due to lack of enough time allowed for developments, developer overlooks certain requirements or disciplines of development. Irregular practices are the root causes of these errors.
- Developer may sometimes be interested in providing extra features. Developers release unapproved increments with additional features. And, this may make software work faulty.

Error III. Logical Design Errors:

Software Analyst, Software Architecture, and Software Engineers prepare the logical design of the requirements specified in the SRS. While framing the logical designs, errors may be generated as well. Following scenarios represent the logical errors.

- For transforming specific requirements into logical design, system analyst design algorithms. This algorithm may be erroneous or not optimized. When such algorithm is implemented, software becomes erroneous as well.
- While designing process flow, system analyst may have inappropriate system flow. IF particular sequences are not following in the logical design then actual product will have the incorrect flow. For example, if a student does not submit assignment, professor may forward his name to the head of department. Head department will decide whether he should be allowed for exam or not. But, the system analyst designs it logically as follows: If a student doesn't respond within three reminders, his name would be forwarded to the head of department. Here, the difference is that, there is no intervention of the professor in the logical design. In actual requirement, professor was allowed to decide whether name is supposed to be forwarded or not. But, in the logical design, the name is directly forwarded without having any concern from the professor. This is an error.
- Errors may be generated in the definition of the boundaries. For example, in the actual design, a student is declared as passed one if he scores more than or equal to 40. But,

in the logical design, system analyst generated the boundaries as less than or equal to 40 marks will disqualify the student from getting passing certificate.

- Exclusion of illegal operation specified in the SRS. For example, student is not allowed to submit assignment for more than twice and he/she can submit PDF only. When user tries to accomplish submission task, he is notified using alert boxes for PDF requirements and two attempts. But user does not receive any kind of system reactions on illegal operation. System allows even if user submit other than PDF.

Error IV. Coding Error:

Coding errors are instigated by the programmers in the development phase. Programmers may do mistakes due to misunderstanding of the requirements, verbal requirements, usage of rapid development tools, etc.

Error V. Non-compliance with documentations and coding instructions:

Every unit needs to have particular compliances such as testing, development, coding guidelines, etc. Understanding and joining a team becomes easy if Compliances are followed. Coding standards must be followed while developing any software. Coding standards are discussed in next section. Test teams can't work upon efficiently if compliances are not monitored. Same happens with development as well as maintenance teams.

Error VI. Limitations of the Testing Process:

Limitations of the testing process have a huge impact on the error rate as errors are not corrected or detected. These errors are engendered because of following reasons:

- If test plans are not prepared for software functions, states or may be whole portion.
- Testing team fails to document and report the errors and bugs.
- Unawareness about the causes of the failure or bug. If appropriate reason is not known for any bug, then it is almost impossible to correct.
- Shortage of time for implementation or negligence towards correction of faults/bugs.

Error VII. Errors due to procedures:

Incorrect procedures lead to errors. For example, concurrent access to databases leads to inappropriate retrieval of data. Same kind of errors are generated is process is not understood properly. These errors are generated when functionalities are not used in a specific way or as per the steps mentioned in the documentation. Every software has a help manual for end user. Users are expected to utilize software according to the manual only. When the steps are not understood as per the guidelines or manual, errors are raised.

Error VIII. Errors in Documentation:

As mentioned in the errors due to procedure that every software has a manual for help to end user. These errors are generated when product is delivered and required to be updated. Next increment or update is surely affected by the documentation generated at the time of delivery. Errors are sometimes generated in the help available in the software itself. For example, an error is mentioned in the manual leads to ***"dead-end"***. Version management related bugs will be also classified under errors in documentation.

4.1.3 Software Quality Factors:

Software Quality Factors and their categorizations have been recommended over the years. McCall's software factors model is the most preferred model as of now. In addition to this there are other models such as Evans and Marciniak's model, Duetsch and Willis model, etc. We are going to study the software quality factors recommended by the McCall.

McCall defined a model with 11 software quality factors which are further grouped in product operation factors, product revision factors and product transition factors. All three Quality Factor Categories are discussed in next 3 sections.

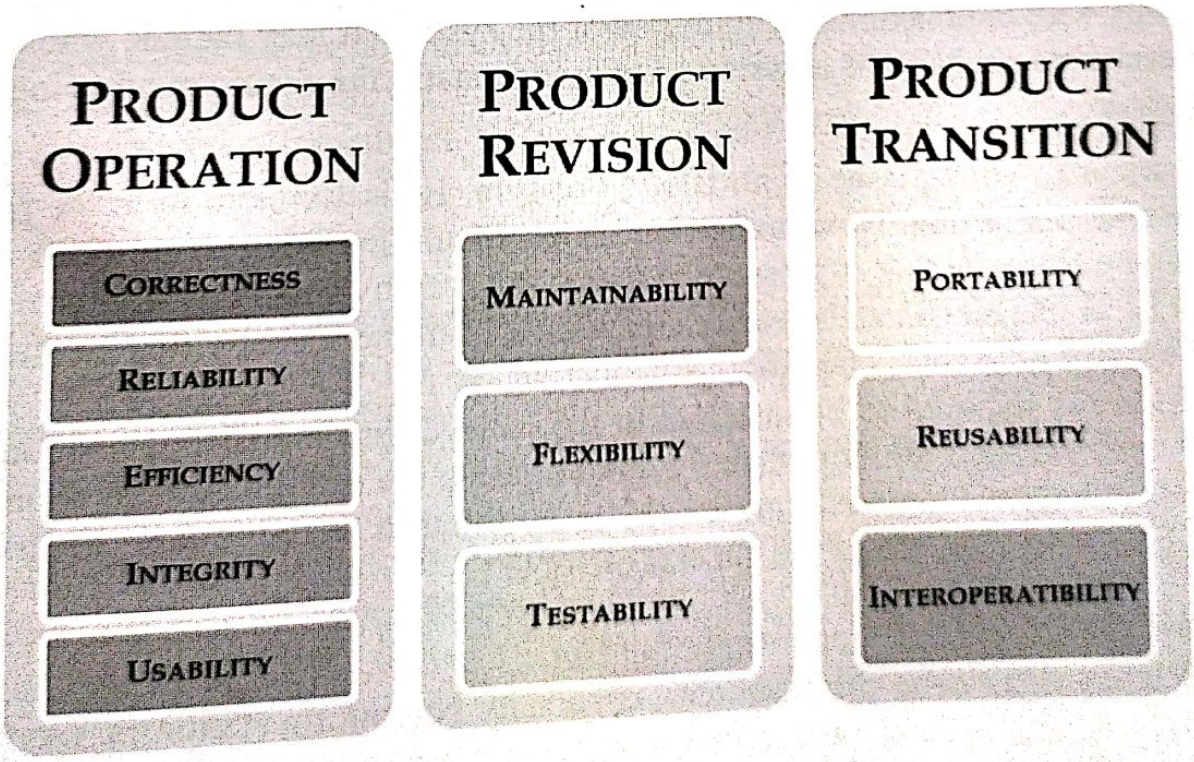


Figure 17 McCall's factor model

- **Product Operation Software Quality Factors:**

In this section, we will discuss about five software quality factors classified under Product Operation Software Quality Factors. These quality factors are having a direct impact on the day-to-day functionalities of the software.

a) *Correctness:*

Correctness requirements are listed in the software requirements' outcomes or outputs such as display students' result in percentage, overall performance of a student on basis of cumulative results, etc. Such outputs are depended on multiple parameters or factors. Followings are the example of such requirements:

- Cumulative Progress Index (CPI) will be calculated on the basis of results of all semesters.
- The required accuracy of the output depends on the data. If data is inaccurate or calculations are inaccurate, then outputs will be affected negatively.
- The completeness of the output information is adversely affected by the imperfect data.
- Accuracy of the data based on the time is also considered. For example, a student has a backlog in a subject. Student applies for rechecking. CPI is calculated using results of other subjects. It is not tested whether updated result of the subject with backlog is available or not. We need to consider the updated data always in the calculations.
- We need to consider minimum time required to retrieve the data.

b) *Reliability:*

Reliability requirement focuses on how much reliable a software and its services are. It deals with the maximum failure rate of the system. It may include whole system or a portion or package of a system. Let's understand the requirement using following example:

- A patient is having issues with Blood Pressure. He is admitted in to the Hospital for constant monitoring. Now, understand what should the failure rate of the monitoring machine. Assume that if pressure is increased even by small number, patient's health will be in danger. It can be easily understood from the scenario that no one can allow failure rate of even 1% per 1000 patients.
- Software is required to be installed in the college for Online Examination. Software must not be failed for more than 5 minutes during the examination duration. Off-time[1] for any failure is generally around 15 to 30 minutes. If any failure occurs, it should not take more than 5% of the general off-time.

c) *Efficiency:*

Efficiency deals with the resources, especially computing resources, required by the system. Computing resources can be the computing power of the CPU such as 2.3 GHz[2]. We need to take care of the storage resources[3] as well. How much storage is

[1] Off-time is the time required to recover or repair any system from failure.

[2] Giga Hertz, should 10^6 cycles per second. Another parameter is instruction per second such as Million Instruction Per Second (MIPS).

required for the software should be address in this. In case of network related efficiency, we need to know the data communication speed[4] of the network. If portable systems are used such as handheld scanners, efficiency requirements must include time required to charge or recharge the batter or system. We can summarize efficiency as the dealing with efficiency of the hardware requirements.

- Your college invites application for attendance management system. Two companies named "A" and "B" send their proposals. Company "A" has requirement of 10GB storage and 10 MBPS network speed for smooth functioning of the system. Company "B" offers a software which requires minimum 5 GB of storage and 8 MBPS network speed. It can be easily understood that proposal submitted by Company "B" is efficient as it requires less amount of computing resources.
- Latest Samsung M30S is better than other phones as it provides 6000 MAh battery backup.

d) *Integrity:*

Integrity requirements focus on the integrity aspect of the security. From the CIA triad, Integrity concerns are chosen in the requirement. No unauthorized person should get any access to the resources. In other words, we can say that only intended user or receiver should have rights to access or receive.

- In an attendance management system, student must be allowed to have view rights only. They must not be able to change the attendance.
- Only, concerned faculty is supposed to fill the attendance. If professor **"A"** is conducting lecture for subject ***"X"***, then professor ***"B"*** must not have the rights to fill attendance for that lecture.

e) *Usability:*

Usability requirements are listed to help the staff members to learn about the system and how to use it. It is mainly considered while training staff. Staff members are informed to fulfil specific requirements set by the developer. For example, a student using Online Learning Management System developed for the college. Students are informed for following requirements.

- Student can submit assignment within 7 days from the published date.
- Student can raise query for the marks within 2 days after the declaration.
- Faculty must not use more than 2 hours to train students about using the software.

[3] Storage Resources are nothing but Hard drive, or external storage device, etc. It is measure in Gigabytes (GBs) or Terabytes (TBs) nowadays.

[4] Data Communication speed is generally measured in KBPS (Kilo Byte Per Second) or MBPS (Mega Byte Per Second)

➢ Product Revision Software Quality Factors:

Product Revision category take account of three factors as shown in the figure. i.e. Maintainability, Flexibility, and Testability. These are affecting the revision of the software hence it deals with correction, updates, etc. These can be corrective to correct the existing error or adaptive to include new functionalities or perfective actions to maintain a perfect execution of the system.

f) *Maintainability:*

Maintainability Requirements is the preparation of the needs of actions required to identify and correct the reasons of failure. It also includes the requirements to verify the corrections. It has important information for the users and maintenance personnel. It refers to the program documentation, structure of the software which is in general modular, and programmer's manual. Followings are the examples of the same.

- Programmer needs to follow the coding standards formulated by the company.
- No function should contain more than 30 statements.
- Optimize algorithm to lower down the time complexity by 10%.

g) *Flexibility:*

Flexibility requirements are all about adaptive maintenance. These requirements are focused on the adaptability of the software for multiple clients of similar requirements. This factor's requirements are in favor of perfective maintenance activities as well. Changes or addition to the existing software to improve service are the examples of such requirements.

- Learning Management System is needed to be developed for online assignment submission and grading. If Assignment is not submitted within deadline, system should send auto generated report to the student mail. This system is having flexibility requirements as follows:
- The Software must be capable enough to be used for all subjects and all semesters.
- Professors conducting non-technical subjects should be able to use system and create assignment and alerts.
- In general, we can say that system should be flexible enough for all kind of users and generic one.

h) *Testability:*

Testability requirements focus on development of instructions or operations which will be executed for testing all the criteria. Proper testing of the system is needed always. Testability requirements help tester to work upon special features. It may be considered for automatic diagnosis of the system as well. This can be easily understood from following example.

- Let's assume student dashboard shows attendance, progress report, etc. every day on the dashboard as soon as student logs into the system.
- System should run automatic diagnosis at the start of every day. It should have sample data for generating and testing demo dashboard. Such requirements are mentioned in the testability requirements.

➢ Product Transition Software Quality Factors:

An application may need to interact with other systems for specific tasks. Such an adaptable nature is always expected when requirements are already mentioned in the SRS. This is related as product transition and three important factors play a vital role. IT also focuses on reusability of the software.

i) Portability:

Portability requirements allows application to function on various platforms. An application needed to be used on Windows should be usable on Linux as well. Such requirements are listed and targeted in the portability requirements. Kindly, note that operating system is just an example. Hardware, other resources, etc. can be the environment for execution of software.

j) Reusability:

As the name suggests, reusability requirements are prepared to make use of existing modules. An application package or module designed for a system may be useful for another system as well. Reusability requirements guide the developer and tester to complete tasks faster than regular speed. For example, a system needs to have club management feature for students. Students from the club are allowed to read books. There are no reusability requirements as of now. But the developer understood that if system is extended for some other use then few common features will be reusable such as,

- Registration for Writing club such as reading one
- A member from the club can write articles

Above mentioned requirements can be easily fulfilled using reusability of the module developed for reading club members.

k) Interoperability:

Whenever a system is required to have communication with other devices or software, it needs interoperability. This requirement will specify for which kind of firmware or software; application is allowed to be integrated or used with. For example, a system is required to be developed for Medical Diagnosis. Now the hardware used for diagnosis is required to be interoperable with system developed for the same.

- **Other models for Software Quality Factors:**

The Evans and Marciniak factor model (Evans and Marciniak, 1987) and The Deutsch and Willis factor model (Deutsch and Willis, 1988) are the alternatives to the McCalls model.

Evans and Marciniak defined a model with 3 categories consisting 12 factors. While Deutsch and Willis developed a model consisting 15 factors which are categories in 4 various categories. Five new factors are included by these two models which are, Verifiability (by both models), Expandability (by both models), Safety (by Deutsch and Willis), Manageability (by Deutsch and Willis), Survivability (by Deutsch and Willis).

4.2 Software Testing:

As discussed in the previous section Software Quality Control can be managed on the basis of multiple factors. There are many tools available for this. The oldest one is the "***Software Testing***". When software testing was started to be practiced, it was used for testing in the final stage only. Gradually, people started understanding the importance of having testing I each and every stage of the product development. Software defects are needed to be identified as early as possible so that it can be corrected in early stage itself. This led the testing to Module testing and then testing at the integration level. Testing is not same as Software Quality Assurance activities. We are going emphasize on testing stratifies and test classification.

"Testing is the process of operating system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or components." This was the formal definition provided by IEEE.

IEEE provided another definition as, ***"The process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software item."*** In this definition, IEEE excluded the actual execution i.e. running the code.

Special test cases are designed for software testing. In the formal testing, all the tests are executed for approved procedures[5] on approved cases[6].

[5] They are the testing procedures approved by the testing team as per the Software Quality Assurance.

[6] Test cases prepared according to the test plan. No requirements are omitted in preparation of the test cases.

4.2.1 Software Testing Objectives:

As the formal definition clears the meaning of Software Testing, we will understand the objectives that can be satisfied through the testing process.

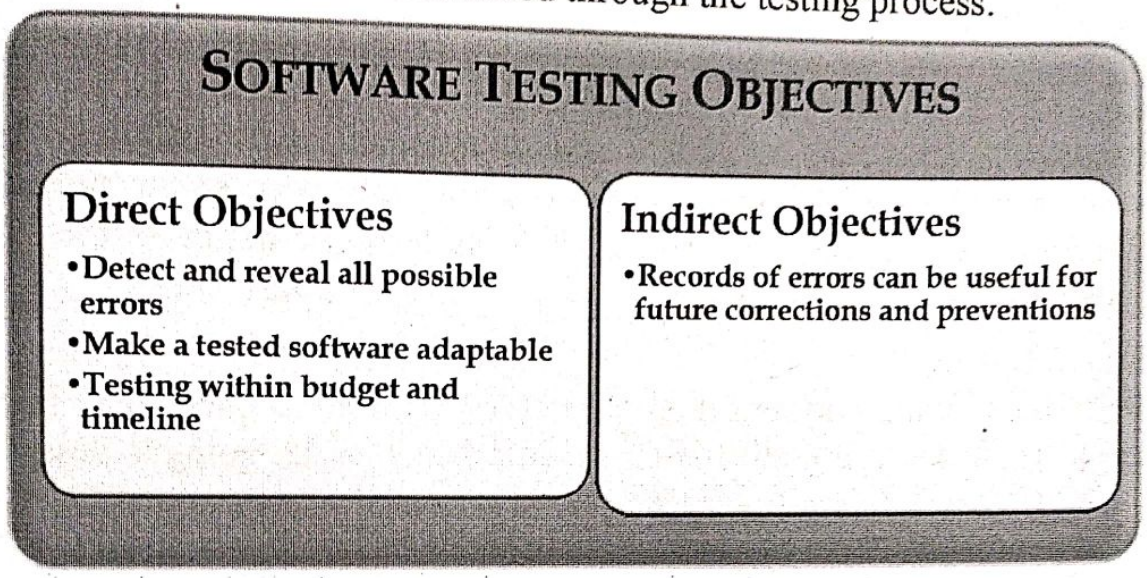


Figure 18 Software Testing Objectives

4.2.2 Software Testing Classification:

It is the dilemma for all tested whether testing functionalities only is enough or not. As per the community discussion, we should include mechanism of the internal architecture or code is also needed. On this contrast whether system or complete functionalities, two approaches or classes are defined: White Box Testing and Black Box Testing.

1. **White Box Testing:** This focuses on structural testing. It also tests internal mechanism, calculations, to find out possible errors or bugs in the system.
2. **Black Box Testing:** It focuses on functionalities only. If output of the functionality meets the expected output, then testing will not consider the internal working of the code.

IEE definitions for both of them are as follows:

Black box testing:

- Testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions.
- Testing conducted to evaluate the compliance of a system or component with specified functional requirements.

4.2.2.1 White Box Testing:

White Box Testing is significant for Unit Testing. There is a numerous way to work upon white box testing. In any approach, you need to define test cases based on the

study of the internal code and based on some experiential. White Box Testing strategy can be coverage-based or fault-based.

- ***Coverage Based Testing:***

  As the name suggests, it covers specific portion of the program only. For example, someone needs to test the working of the particular control flow or statement only. For coverage based testing, you need to select particular element of the program. ***This set of specific elements that are supposed to be executed under a testing strategy, is called testing criterion of the strategy.*** For example, you need to execute all the "IF Statements" of the program once, then we can say testing criterion of the strategy is branch coverage.

- ***Fault Based Testing:***

  In Fault Based Testing, only faults are targeted. You need to understand that we are not going to work upon all faults but for specific faults we are going to design the test cases.

  You already know that there can be number of white box testing strategies. It is key to identify a strong or weak testing strategy. How can we compare two testing has been a point of discussion always? A strategy, which covers testing of the elements tested by another strategy and also provides something additional, is called the Strong Testing Strategy than other one. In figure 5, we can see that Strategy A covers all the elements covered by strategy B. So Strategy A is accepted as strong testing strategy. In Figure 4,

Figure 4 Strong and Weak Testing Strategy

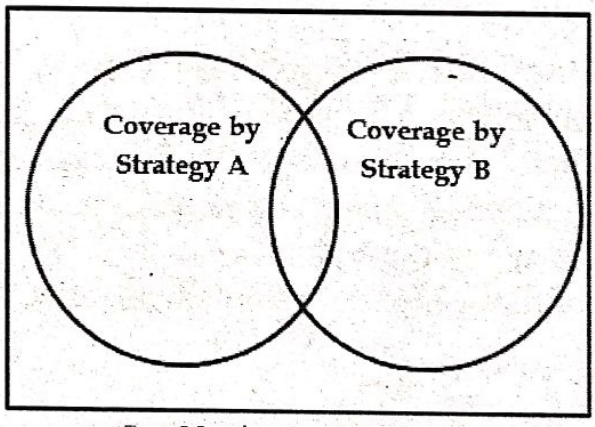


Figure 3 Complementary Testing Strategies

we can see that both the strategies use different elements of the program, such strategies are called complementary strategies. Some of the elements covered by A are not covered by B and vice versa. You need to understand that no strategy will have complete coverage.

- ***Statement Coverage:***

  This coverage is used to prepare test cases to test every statement of program at least once. No one can identify an error without executing particular statement. Hence, executing once at least is the only way to know if any statement has any error or bug.

This is the principal idea behind executing every statement once. Let's assume that we need to design test case for following code which finds a larger number.

```
1. def getLargerNumber(x,y):
2.     if(x != y):
3.         if(x > y):
4.             print("Larger Number is :", x)
5.         else:
6.             print("Larger Number is :", y)
7.     else:
8.         print("Both Numbers are equal")
```

To design the test cases for the statement coverage, "if statement" is needed to be true as well as false to test both the cases. So we need to choose test set such as {(x=2, y=2), (x=2, y=3), (x=3, y=2)} and execute all the statement at least once for test set.

- ***Branch Coverage:***

Branch Coverage based testing requires test cases defined for all the possible branches. For example, in the above shown code, 2 IF statements are there. We need to define set for both of them. Hence, test set will be, {(x=2, y=2), (x=2, y=1), (x=3, y=2), (x=2, y=3)}. Branch Coverage based testing is considered to be the stronger one than statement level coverage. Following reasons are adequate enough to prove this.

- Branch coverage will cover all the statement coverage based tests as well. You need to assume there is no unreachable code in the program.
- Statement Coverage based test set will not cover all branch coverage based test set. For example, in the following code, test set for statement coverage will be {2}. Test value 2 will test the working of ***"if statement"***. But to have branch coverage, both branches are needed to be identified i.e. branch for true and branch for false. For test set {2}, condition will never be false and hence the branch under false will never be tested.

```
o       if (x > 1): x = x+1
```

- ***Condition Coverage:***

In the condition coverage based testing, test cases are designed to make every condition true and false. When conditions are composite one, all the true and false values are tested. For example, in the following condition-1 is $x != y$ and condition-2 is $x > y$. So here total test case required are 4. Generally, number of test case required for n conditions in a composite statement will be 2^n. This shows that test set increases exponentially in case of Condition coverage. Branch testing is nothing but simplified Condition Coverage-based testing. As in branching the final value of composite condition is used while for condition coverage needs every condition within composite one to be true and false.

Condition	Set for making True	Set for making False
x != y	{x=2, y=3}	{x=2, y=2}
x > y	{x =2, y=1}	{x=1, y=2 }

Table 1 Test set creation for Condition Coverage

- if (x != y && x > y): print("X is larger")

➢ ***Path Coverage***

Path Coverage based testing strategy requires designing test cases which can test all the linearly independent paths in the program. All the basic paths are executed once at least. To prepare test case for this strategy, one needs to understand the Control Flow Graph. Following is an example of CFG (Control Flow Graph).

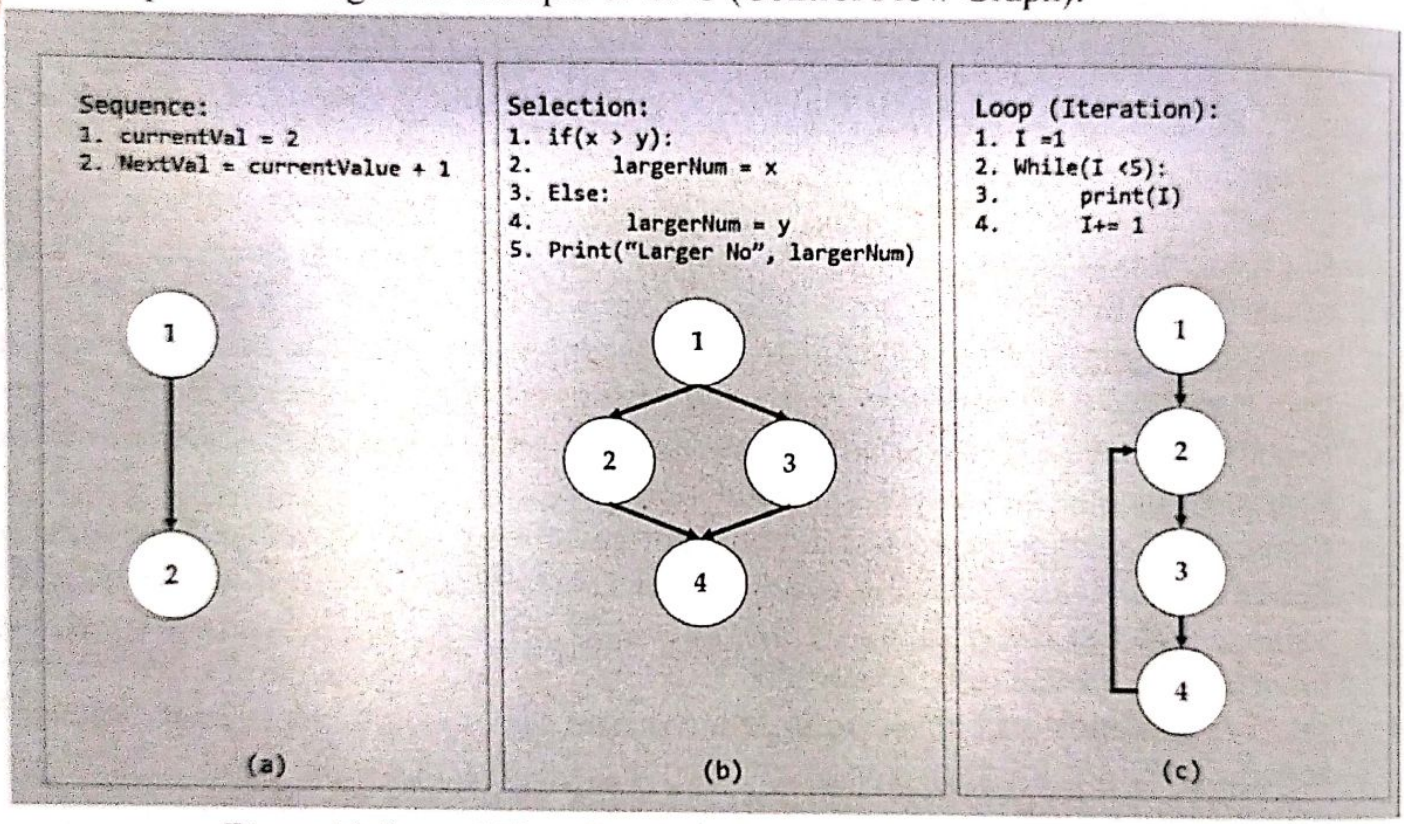


Figure 19 Control Flow Graph for Sequence, Selection and iteration

To draw control flow graph, you need to number the statements of the program at very first stage. Line number will be used as node of the graph. If one statement can lead the flow to another statement, then there will be an edge between two nodes. Formally, CFG is defined as a directed graph containing a set of nodes and edges (N,E), such that each node n Є N corresponds to the unique program statement and edge exists between two nodes if control can transfer from one statement to another statement.

A path is ay node and edge sequence from the start node to end node of the control flow graph of a program in the presence of return or exit statements. For example, path for figure 6(c) is [12342342342342]. Length of the path depends on the ending condition which may lead to infinity value and infinite number of paths too. This is

the reason why we consider subset of path called linearly independent path and exclude consideration for all possible paths. To calculate the linearly independent paths, you can use McCabe's Cyclomatic Complexity Metric. The naivest method to calculate Cyclomatic complexity V(G) as,

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

Where, E is the number of edges and N is the number of nodes. For example, V(G) for figure 6(c),

$$V(G) = 4 - 4 + 2 = 2$$

Another method to calculate the Cyclomatic complexity is to use number of loops in the program. If number of loops in the program are N, then

$$V(G) = N + 1$$

A simple algorithm for carrying path coverage based testing is as following:

Step 1) Draw CFG (Control Flow Graph)
Step 2) Determine vertices of the Graph (V(G))
Step 3) Determine minimum number of test cases required for path coverage
Step 4) Repeat
   a. test using a randomly designed set of test cases
   b. perform dynamic analysis to check path coverage achieved

until 90% path coverage is achieved.

4.2.2.2. Black Box Testing:

As mentioned above, black box testing is not about internal structure and it depends on input and output only, it can be done using two kind of test cases.

➢ ***Equivalence Class Partitioning:***

In this, all the domain values are classified into various equivalence classes. It is taken care of that all the data within an equivalence class have same behavior after executing the code. The advantage of equivalence class partitioning is that we don't need to test for all data. As data receives similar behavior from the program, we can test using single data. General guidelines to create equivalence classes are as follows:

a) If your input data values are used from a range, then you are required to submit one valid and two invalid equivalence classes. For example, if equivalence class is set of integers in the range of 1 to 20 (i.e. [1,20]), then invalid classes will be [-∞, 0] and [21, ∞].

b) If input data values are from set of various members of some domain, you need to define two classes. One class for valid and another for invalid data inputs. Let's assume, valid input values are characters "S" and "P" in our system. Here, we have discrete values so we need to define a set. Set which is acceptable is {S,P} and the set which is not valid is U – {S,P} (i.e. a set containing all values other than S and P).

➢ ***Boundary Value Analysis:***

In general, programmers make errors while defining boundaries for special cases. This is only psychological factor which make them make such errors. Data inputs lying on the boundaries require special treatment so a class can be built for these values. A simple example is the use of < in place of <= or > instead of >=.

This is why Boundary Value Analysis is very much useful for creating boundaries of various equivalence classes to handle them wisely. As we know, equivalence classes have two kinds of input data either range or discrete set, we need to check classes carefully while defining Boundaries.

You cannot define any boundaries if your input data is not in a range or just discrete values. This is very much obvious and intrinsic behavior. If data input values are using a range such as shown in above example that accepted values are from the range 1 to 20, then boundary values test suite is {0,1,20,21}. Everything less than and equal to zero are invalid as well as everything more than equal to 21 are invalid.
Black Box Testing is quite simple approach. The vital step of this procedure is to identify the equivalence classes which may not be straight forward sometimes. As testers are working on this thoroughly, after few practices it becomes easier for anyone to define such classes. It is clearly seen from the example that boundary values can be automatically defined using some calculations. This is why identification of Equivalence classes plays an imperative role in Black Box Testing.

4.2.3 Integration Testing Strategies:

Software Testing may have numerous way to do it. Here, we are going to discuss two most preferred framework or strategies for integration testing. Test the system development step by step. Every stage will have certain outcome. So the output will be tested. Whenever a unit or module is completed, it is tested. This is called ***Module testing*** in general. Next to this, all modules are integrated. It may be possible that integrated system may not work flawlessly. This is why we test integrated system which is called ***Integration Testing***.

Once this testing is done, whole package is generated. Finally, this package is tested which called system is ***testing***. As we are going step by step, it is called "***incremental testing***". This can be done through bottom-up or top-down approach.

➢ **Big-Bang Integration Testing**:

Big-bang testing is the most basic testing strategy for integration testing. In this approach, all the modules are integrated in single stage. This must be used when system is small. As all the modules are integrated in the single stage, error identification is the most difficult task. It is not possible to identify which module is raising the error. This is why it is never used for large programs.

- **Bottom-Up Integration Testing**: Bottom up testing is shown in figure 2. There are 4 stages shown in the image. M1, M2, M3, M4 and M5 are developed in the stage 1. In this stage, all 5 are tested. In stage 2, M1 and M2 are integrated with M6 and released as Integration 1. So here in stage 2, Integration test will be executed as well. In stage 3, M6 integrated unit will be integrated with M7 developed in the current stage. Also, M4 and M5 are integrated with M8 developed in the current stage. Hence, 2 integrations are released named Integration 2 and Integration 3. These both integrations are tested in stage 3. Finally, all integrations are integrated with M9 and final package is released. Here, we will have complete system. So in stage 4, system testing will be done.

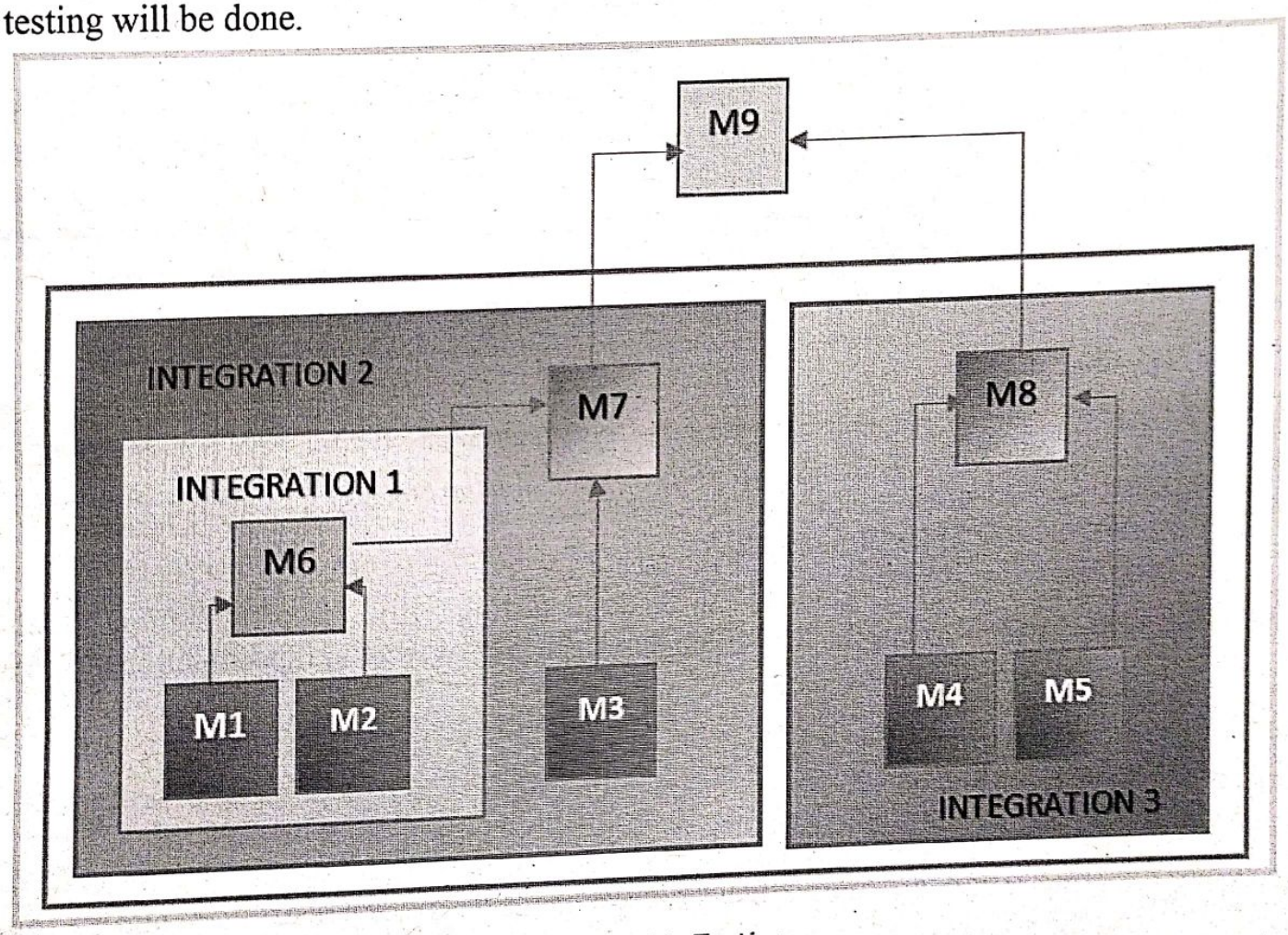


Figure 20 Bottom Up Testing

Top-Down Integration Testing:
Approach shown in figure 3 is top down testing approach. Here, Module 9 is first tested in stage 1. In stage 2, Module 7 and 8 are tested and integrated with test module 9. This integration is also tested in the stage 2. The same process is followed until we test all the modules, integrations and finally, the product. It starts with the root module and one or two subordinate modules in the system. Once, top-level skeleton is tested, the underlying layers are tested.

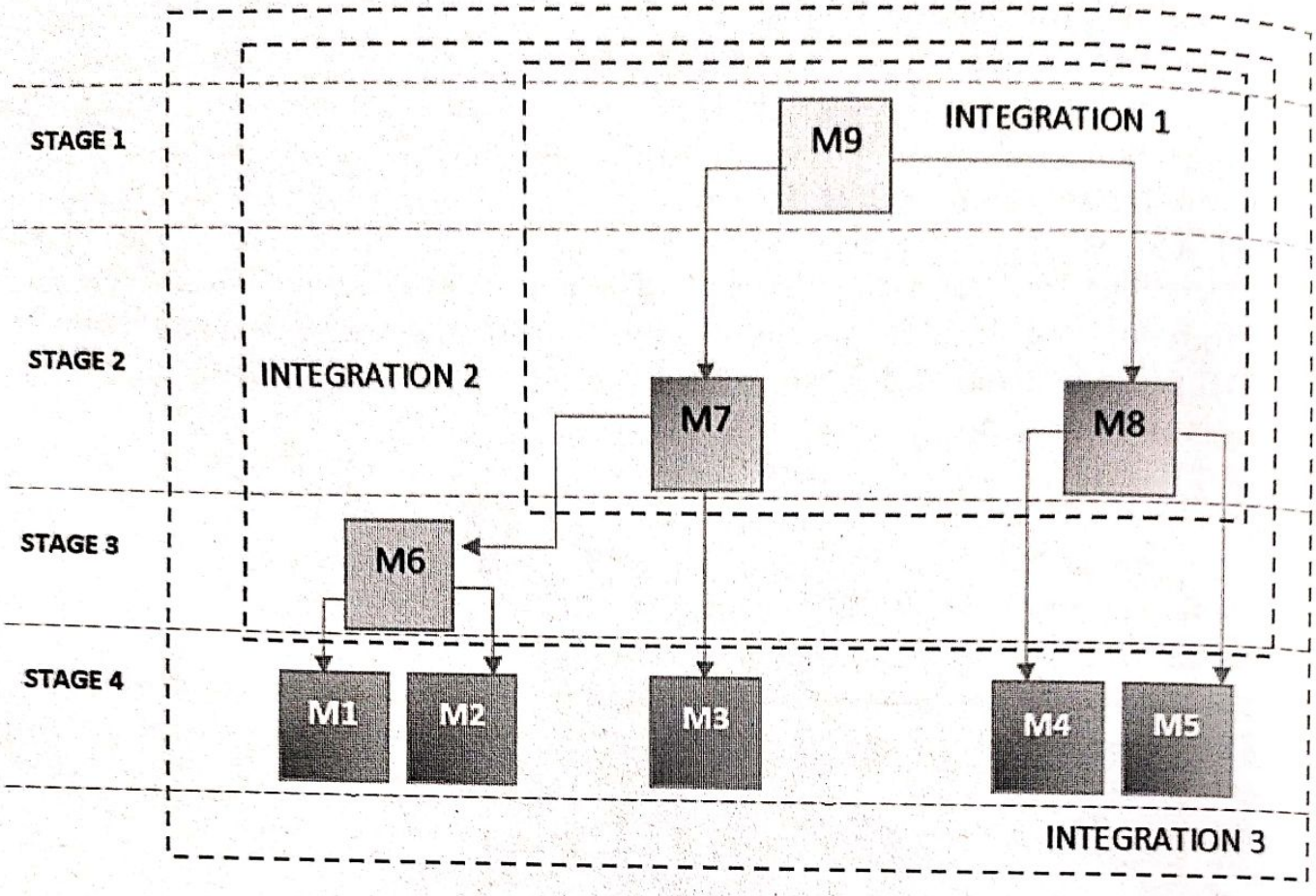


Figure 21 Top Down Testing Strategy

- **Mixed Integration Testing**:

It is also called sandwiched integration testing. It is the combination of top-bottom and bottom-up approach. Mixed approach can overcome the limitations of top-bottom and bottom-up approach.

4.3 System Testing:

System Testing is designed to validate a completely developed system. All the requirements mentioned in the SRS are supposed to be met at the end of system testing. There are main 3 ways to test whether a system is efficient and meeting all the requirements or not.

- Alpha Testing
  Alpha testing is carried out by the tester team within the organization only.
- Beta Testing
  Beta testing is carried out by the stake holders. It may the customers themselves.
- System Acceptance Testing
  Acceptance Testing is the system testing accomplished by the customer to determine whether he should accept the system delivery or not.
  In all three kinds of testing, test cases will remain same but the context will be different as all the testers are from different domain.

4.3.1 Performance Testing:

Performance testing is an important type of system testing. IT focuses on non-functional requirements. Purpose of the performance testing is to test whether systems meets the non-functional requirements mentioned in the SRS. All the performance tests are considered as a Block Box Testing

- **Stress Testing:**

Stress testing is also known as Endurance Testing. It is used to evaluate the performance is system is stressed for a short span of time. You need to test all the non-function illegal input conditions. Stress for a software can be understood as the extra load given to the software. For example, if a software is supposed to be used by 30 students at a time, then you will test system initially with 30. Once this test is accepted, you need to try with increased value. Here, system will be called stressed. It can be useful when systems are performing below their capabilities. It is tested for specific conditions. For example, at the time of result, almost 100s of students are accessing the result website. In other condition, result website will not have enough amount if visitors. But at the time of result, all the students are supposed to visit the website. For such non-functional requirements, performance is tested. This condition will occur very less frequently but the system must not suffer in such conditions too. This purpose can be served using Performance Testing.

- **Volume Testing:**

Volume testing works upon the data structure requirements. It is necessary to handle resource requirements of the software efficiently. Memory leak, stack overflow, etc. can lead to poor performance. For example, a web application is tested to check how much stack size is used and whether it raise stack overflow or stack usage error or not.

- **Configuration Testing:**

As the name suggests, configuration testing is carried out to test the performance of the system for various platform. These platforms can be software as well as hardware. Sometimes, client request to operate system in different environment and want his employee use in different environment. For example, as a client I need to have smooth execution of my software in windows. While, I need same software to be used by my employees on Ubuntu. For these requirements, system must be tested for both configurations.

- **Compatibility Testing:**

Compatibility testing is required to test whether system provides acceptable performance with various interface. It is different from Configuration testing. For example, I need to test whether my application can retrieve data with required speed from large database server or not. Such testing is called compatibility testing.

- **Regression Testing:**

  Regression Testing is required when we update the existing system. If a delivered system or existing system is tested and found to be faulty, we need to update the system to resolve bugs. Testing of such system is done using Regression Testing. It will be discussed in further section.

- **Recovery Testing:**

  Recovery Testing is carried out to test how system will perform when it is under faulty conditions. Sometimes, client has the requirement to store as much data as possible. This kind of requirement needs special attention to store data in extremely failure conditions such as loss of electricity or power.

- **Maintenance Testing:**

  This addresses testing the diagnostic program, and other procedures that are required to help maintenance of the system.

- **Documentation Testing:**

  It is used to check whether manual is properly prepared for users or not. If there are multiple types of users of a system, then it should be designed and tested according to the compliances.

- **Usability Testing:**

  Usability Testing is executed to test the User Experience with System. A system may require specific UI, or report generation, etc. It can be called a GUI testing as well because you are going to test all screen, display, reports, formatting, etc.

- **Security Testing:**

  Security Testing is required when you have very much confidential data. Certain compliances such as HIPAA are required to be followed. If any organization fails for such compliance, organization is responsible for the same. A system must be secure enough so that no intruder can attack and get access to the data.

4.4 Regression Testing:

Regression Testing cannot be covered as Unit, Integration or System Testing. It is a distinct dimension other than these three types of testing. As it is affecting after updating a system, you need to test a system after each and every change in the system. If any bug is fixed, you need to do regression testing after update. If number of bugs are very less and very less number of changes are there, then entire test suite is not needed to be executed. One needs to just test the required functioned which are updated after changing or solving the bugs.

4.5 Test Metrics and Measurements:

"You can't control what you can't measure" – Tom DeMarco (1982). The statement said by Tom DeMarco has been an axiom for software testing industry people. Quality Metrics can be defined as follows by IEEE: *"A quantity measure of degree to which an item possesses a given quality attribute"* Or *"A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given quality attribute".*

4.5.1 Objectives of Software Quality Metrics:

Software Quality Metrics are not applied as they are applied in other industries as of now. Software Engineering Industry has a new objectives of the Software Quality Metrics. The prime objectives are as follows:

- Software Quality Metrics can facilitate to know the nonconformities of actual quality performance from planned performance.
- It can be used as a software quality assurance tool to measure the difference between the actual time and budget performance from the planned one.
- Metrics collected as an information can be very much useful to identify the situations that need or permit maintenance process involvement in form of some actions.

4.5.2 Software Quality Metrics categories:

Software Quality Metrics can be decided and defined as per the testing team's requirements and feasibility. It may depend on the system and the way a system will be defined. Here, categories are classified in two levels. First level is used for matrices related to life cycle and phases of the software system. These metrics are process metrics and product metrics. While, second level refers to the subject of measurements such as Quality, Productivity, Effectiveness, and Timetable.

4.5.2.1 Software Quality Process Metrics:

Software Development Process Metrics can be considered as one of the following four categories.

1. **Software Process Quality Metrics:**
   Error Density Metrics and Error Severity Metrics are the major quality metrics under the software process category.

   Error density metrics can be calculated using software volume and number of errors counted. Software volume metric uses number of lines of code while other metric applies function points. Error counted works upon number of errors or weighted number of errors. Following example illustrates the calculations of Error Density Metrics.

Example: A software development team wants to use two measures Number of code errors and Weighted number of code error. Weightages as per the error severity is defined as following table:

Error Severity Class	Relative Weight
Low	1
Medium	3
High	9

Table 2 Relative Weightage of Error according to Error Severity Class

The code error summary for a project of learning management system is as follows: 10 low severity errors, 5 medium severity errors and 4 high severity errors. Calculations for Number of code errors (NCE) and weighted number of code errors (WCE) are as follows:

Error severity class a	Calculations of NCE b	Calculation of WCE	
		Relative Weight c	Weighted Errors D = b*c
Low severity	10	1	10
Medium severity	5	3	15
High severity	4	9	36
Total	19	-	61
NCE	19	-	
WCE	-	-	61

Table 3 Calculations of Error Density Metrics

Size of any software is measures in thousands of lines. This metric is known as KLOC. This metric is very much useful in calculations of density metrics. Followings are six important error density metrics.

$$\textbf{Code Error Density (CED)} = \frac{NCE}{KLOC}$$

$$\textbf{Development Error Density (DED)} = \frac{NDE}{KLOC}$$

$\ldots\ldots\ldots\ldots\ldots$ *Where NDE is number of development errors*

$$\textbf{Weidghted Code Error Density (WCED)} = \frac{WCE}{KLOC}$$

$$\textbf{Weidghted Development Error Density (WDED)} = \frac{WDE}{KLOC}$$

$$\textbf{Weidghted Code Error per Function Point } (WCEF) = \frac{WCE}{NFP}$$

..... Where NFP is number of function points collected from the surveys.

$$\textbf{Weidghted Development Error per Function Point } (WDEF) = \frac{WDE}{NFP}$$

Let's consider the same project of learning management system. Assume that number of lines are 38 KLOC which is apparently size of software. We can calculate CED and WCED as follows:

Measures and Metrics	Code Error Density	Weighted Code Error Density
NCE	19	-
WCE	-	61
KLOC	38	38
CED (NCE / KLOC)	0.5	-
WCED (WCE / KLOC)	-	1.6

Table 4 Calculation of CED and WCED

It can be concluded that CED doesn't provide appropriate information about acceptance or rejection but WCE metric can be useful to decide whether quality is acceptable or not. If Error density is too high, the unit's quality should not be accepted.

Error Severity Metrics are used to detect adverse impact of the errors. It is used when number of code errors and weighted code errors are decreasing but severe errors are increasing.

Error severity metric	Formula
Average Severity of Code Errors (ASCE)	WCE / NCE
Average Severity of Development Errors (ASDE)	WDE / NDE

Table 5 Error Severity Metrics

2. Software Process Timetable Metrics:

Software Process Timetable Metrics are useful to determine the progress of the project. It ca be used to calculate the average delay in achieving a milestone. Followings are the metrics' calculation used as timetable metrics;

Timetable metric	Formula
Time Table Observance (TTO)	$\frac{Milestones\ Completed\ On\ Time\ (MSOT)}{Total\ No.of\ Milestons\ (MS)}$
Average Delay of Milestone Completion (ADMC)	$\frac{Total\ Completion\ Delays\ in\ Milestones\ (TCDAM)}{Total\ No.of\ Milestons\ (MS)}$

Table 6 Software Process Timetable Metrics

3. **Error Removal Effectiveness Metrics:**

Error removal can be measured after period of operation such as after a span of a year. It gathers error records at the time of development and failure records raised at the time of compilation for a regular operation. Followings are the Error Removal Effectiveness Metrics:

Error Removal Effectiveness Metric	Formula
Development Error Removal Effectiveness (DERE)	$\frac{NDE}{NDE + NYF}$
Development Weighted Error Removal Effectiveness (DWERE)	$\frac{WDE}{WDE + WYF}$

Table 7 Error Removal Effectiveness Metrics

Here, NYF is number of failures detected during a year of maintenance service and WYF is weighted number of failures detected during a year of maintenance service.

4. **Software Process Productivity Metrics:**

This metrics can be considered as direct metric as well as indirect metric. When we consider direct metric, it focuses on human resource productivity. While indirect metric focus on extent of software reusability.

Software Process Productivity Metrics	Formula
Development Productivity (DevP)	$\frac{DevH}{KLOC}$
Function point Development Productivity (FDevP)	$\frac{DevH}{NFP}$

Code Reuse (CRe)	$\frac{ReKLOC}{KLOC}$
Document Reuse (DocRe)	$\frac{ReDoc}{NDoc}$

Table 8 Software Process Productivity Metrics

Here,

DevH = total working hours invested in the development of the software system.

- ReKLOC = number of thousands of reused lines of code.
- ReDoc = number of reused pages of documentation.
- NDoc = number of pages of documentation.

4.5.2.2 Software Quality Product Metrics:

Software Quality Product Metrics are useful in the operational phase of any software. For providing quality service either for a purchased software or software under development., a strong software support is required. This can be done two ways:

Help desk services (HD) – HD services are the services to help client in using the software features or functionalities. Demand of this service depends on the quality of help manual or documentation provided to the client.

Corrective maintenance services – This includes correction of the errors or failures before getting them identified by the clients. It is always better to test and provide completely tested product. But, whenever a customer service team notices a failure, it should be recorded for corrective actions.

4.5.3 Limitations of Software Metrics:

- Though Software Metrics are so much useful. Allocation of budget has been a major problem. This requires a dedicated human resource, funds etc.
- Disagreement of employees for evaluation of their productivity
- Ambiguity in data's validity

Exercises

Answer the following Questions:

1. Explain the objectives of software quality metrics.
2. Define Software Quality Assurance and Control.
3. Which are the types of software errors?
4. Discuss about software quality factors.
5. What are the product transition software quality factors?
6. Design Software Testing.
7. What are the objectives behind software testing?
8. When would you use Black Box Testing?
9. What strategies are called complementary testing strategies?
10. What are the methods to calculate Cyclomatic Complexity of a CFG? Provide suitable example.
11. Which approach is more suitable for Integration Testing? Discuss about available approaches for the same.
12. What is the difference between Configuration Testing and Compatibility Testing?
13. When should one use Regression Testing?
14. Which are the major software quality metrics?
15. Which are the metrics available for error removal effectiveness?
16. How can one measure productivity of Human Resource?
17. What are the limitations of Software Quality Metrics?

State whether following statements are true or false:

1. An activity carried out to evaluate quality of a product is Quality Assurance. – False
2. Software errors are quite different from Software Fault. – **True**
3. If Errors are generated due to wrong definition of requirement is covered under client developer communication error. – **False**
4. Rapid Development is one of the reasons for Deviation from Software Requirement. – **True**
5. Flexibility is covered under product operation Factors. – **False.**
6. Integrity factor resemble with integrity mentioned for information security. – True
7. Recording of all errors is the direct and major objective of software Testing. – False
8. Branch coverage is better than Condition Coverage. – **False**

9. You should cover at least 90% independent paths for path coverage testing. – **True**
10. Mix testing resolves the limitations of bottom-up and top-bottom integration testing. – **True**
11. Breach in securing data is covered under security testing. – **True**
12. Code Error Density provides enough measures to decide acceptability of a unit pf a product. – **False**

Match the following:

Metric	Formula
1. Average Severity of Code Errors (ASCE)	a) $\frac{NDE}{NDE+NYF}$
2. Average Severity of Development Errors (ASDE)	b) WCE/NCE
3. Development Error Removal Effectiveness (DERE)	c) $\frac{WDE}{WDE+WYF}$
4. Development Weighted Error Removal Effectiveness (DWERE)	d) WDE/NDE

Ans: 1 – b, 2 – d, 3 – a, 4 – c

Solve the example:

1. Write down program to find whether a number is odd or even. Prepare CFG for the same and calculate the Cyclomatic Complexity of the program.
2. Calculate NCE, WCE, CED and WCED for following specifications. A software is developed using 45K lines of code. When system was tested, 23 low severity errors, 18 medium severity errors and 15 high severity errors are collected. Calculate above mentioned parameters and discuss about your conclusion.
3. Company "*A*" was asked to develop a software "*X*". The development team divided the work in to 18 milestones. Due to heavy work load and few inefficient developers, only 13 milestones were achieved on time and all other milestones were delayed by 6 days. Project was delivered after a month. Calculate Time Table Observance Metric and Average Delay in Milestone Completion.

☺☺☺☺☺

MU-140

March-2019

B.C.A., Sem-IV

CC-209: SAQT

(New)

Time: 2:30 Hours] [Max. Marks: 70

1. (A) Answer the following:

(i) Discuss incremental model 7

(ii) Draw a context level and first level DFD for railway reservation system. 7

OR

(i) List and explain the points to be kept in mind while preparing survey.

(ii) Draw acontext level and first level DFD for online examination system.

(B) Fill in the blanks: (any **four** out of **six)** 4

(i) A questionaire is also called a_____.

(ii) DFD entities are also called_____.

(iii) _____is a planned meeting in order to obtain information for SDLC.

(iv) In DFD_____is a top level views of an information system.

(v) The process which produces only output is called_____.

(vi) _____question limit or restrict the response.

2. (A) Answer the following :

(i) List and explain the guidelines for designing of use-case. 7

(ii) Draw the sequence and collaboration for shopping cart application. 7

OR

(i) Explain the context of different types of messages in sequence diagram.

(ii) Draw a use-case diagram for adnission process in Gujarat University.

(B) True or False: (Any **four** out of **six)** 4

(i) Actors are always external to the system.

(ii) Activation represents communication between instances.

(iii) For each use-case many sequence diagrams can be drawn.

(iv) Rectangle around the use case id called border of the syatem.

(v) Combined fragment are used to show conditional flow in sequence diagram.

(vi) Primary actors must be specified on the left most side of the diagram.

3. (A) Answer the following:

(i) Explain different types of relationship in a class diagram. 7

(ii) Discuss any five guidelines for drawing activity diagram. 7

OR

(i) Discuss the difference between analysis class diagram and design class diagram.

(ii) Explain the use of interface in class diagram.

(B) Fill in the blanks: (Any **three** out of **five**) 3

(i) A class associated with it self is called_____association.

(ii) A_____is a physical connection among the objects.

(iii) The transition element in square brackets is called_____condition.

(iv) Initial state in activity diagram is always placed in_____corner.

(v) Association_____names are optional.

4. (A) Answer the following :

(i) List and explain the factors governing performance testing. 7

(ii) Explaint the importance of black box testing. 7

OR

(i) Discuss integration testing.

(ii) Explain the types of regression testing.

(B) Define the terms: (Any **three** out of **five**) 3

(i) Response time

(ii) SLA

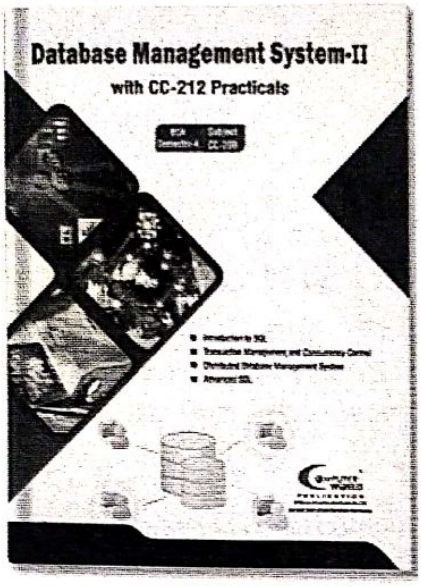
(iii) Stress testing

(iv) Test scenarios

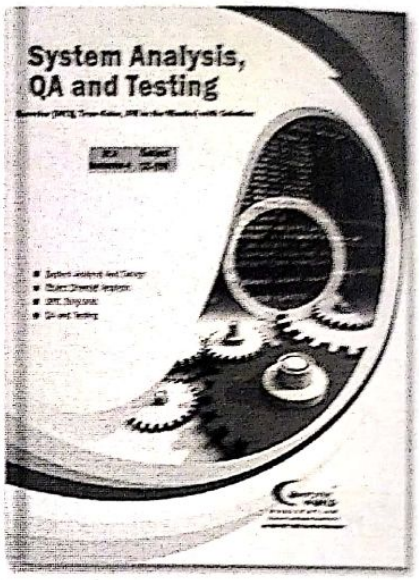
(v) IDE

☺☺☺☺☺

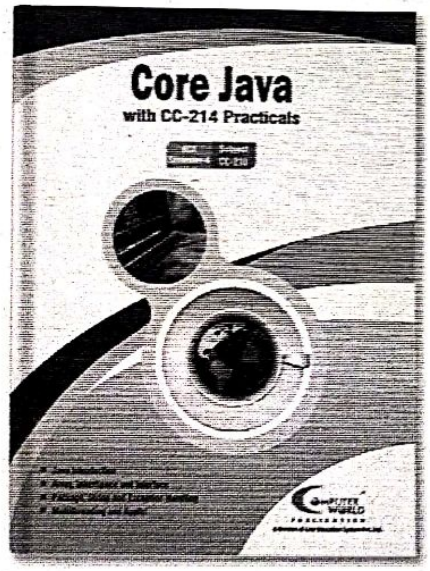
Study Material for
Bachelor of Computer Application (BCA) Sem-4



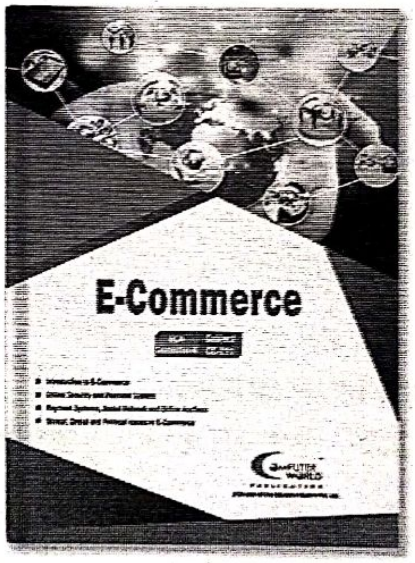
CC-208
Database Management System-2
with CC-212 Practical



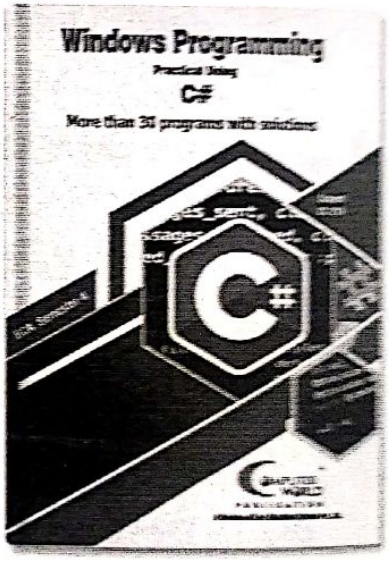
CC-209
System Analysis, QA and Testing



CC-210
Core Java with CC-214 Practical



CC-211
E-Commerce



CC-213
Windows Programming using C#

Also available Books for BCA

emester-1, Semester-2, Semester-3, Semester-4, Semester-5 and Semester-6

According to the New Syllabus of
Gujarat University

Bachelor Computer Application (BCA)
Study Material

More Books for BCA

Semester-1, Semester-2, Semester-3, Semester-4, Semester-5 and Semester-6

Shop Online
@
www.computerworld.ind.in

COMPUTER WORLD PUBLICATION
A Division of Live Education System Pvt. Ltd.
An ISO 9001:2008 Certified Company

COMPUTER WORLD PUBLICATION
43, 5th Floor, Sanidhya Complex, Opp. Sanyas Ashram,
Nr. M. J. Library, Ashram Road, Ahmedabad-380009.
Ph. : (079) 26580723 / 823, 97240 11150, 97250 22917
URL: www.computerworld.ind.in, www.cworld.co.in
e-Mail : info@computerworld.ind.in

978-93-88092-21-0
9 789388 092210
Computer World

Book Code : CEBCA115 | Price ₹ 8