Jawaharlal Nehru Technological University
College of Engineering (Autonomous) , Anantapur

.

SOFTWARE ENGINEERING
LAB RECORD

Name :
Admission No :
Branch : Computer Science And Engineering
CLASS : B.Tech - II - SEM – II

1.Draw the Work Breakdown Structure for the system to be automated.

AIM: To draw the work breakdown structure for the system to be automated.

WORK BREAKDOWN STRUCTURE:

A work breakdown structure starts with a large project or objective and breaks it down into smaller, more manageable pieces that you can reasonably evaluate and assign to teams. Rather than focusing on individual actions that must be taken to accomplish a project, a WBS generally focuses on deliverables or concrete, measurable milestones. These deliverables may also be called work packages, tasks, sub-tasks, or terminal elements.

USES OF WBS:

There are a number of reasons why breaking down a large project is beneficial. It helps you to:

- Estimate the cost of a project.
- Establish dependencies.
- Determine a project timeline and develop a schedule.
- To write a statement of work.
- Assign responsibilities and clarify roles.
- Track the progress of a project.
- Identify risk.

CREATION OF WORK BREAKDOWN STRUCTURE:

1.Record the objective you are trying to accomplish. This objective could be anything from developing a new software feature to building a missile.

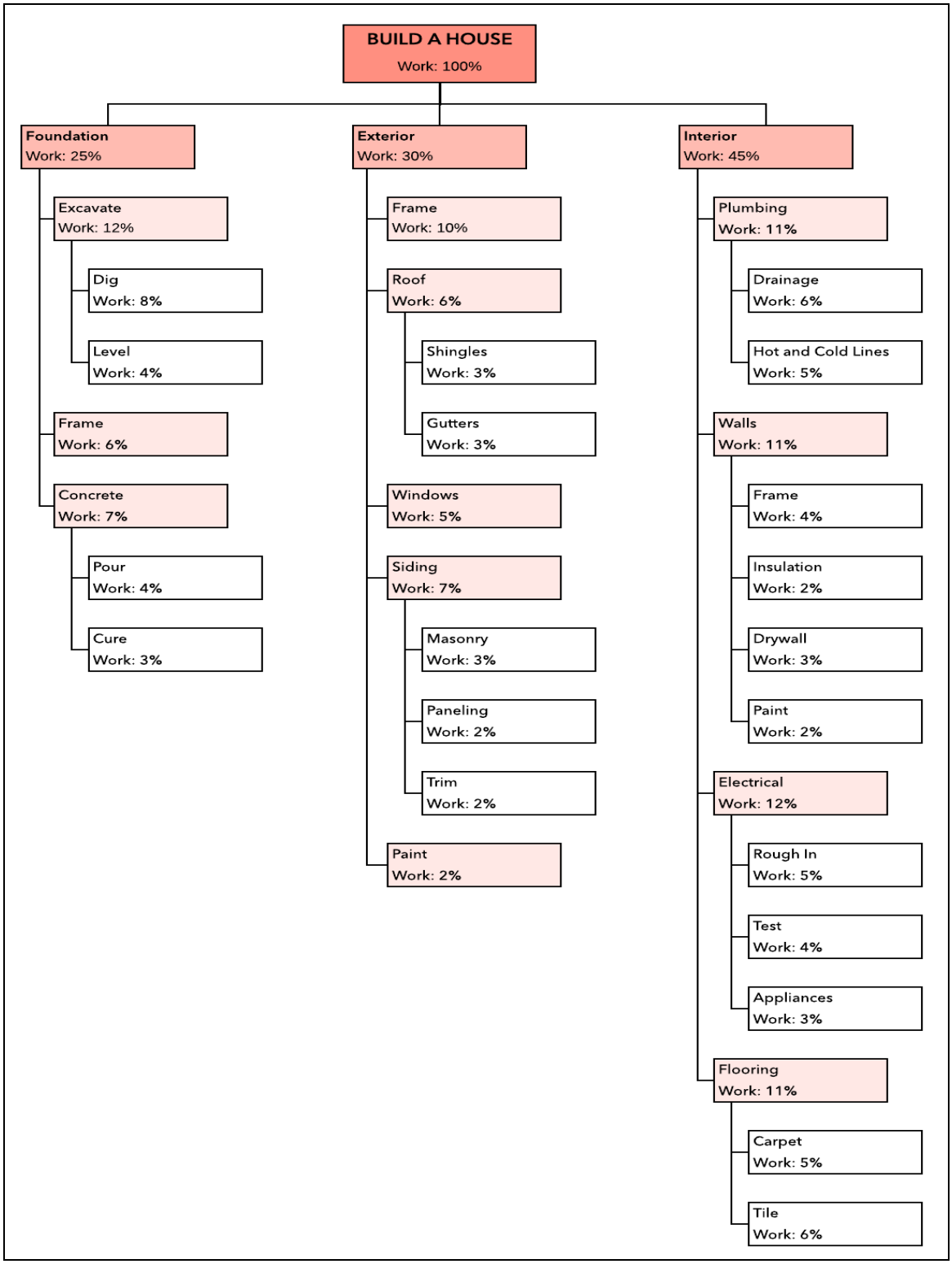
2.Divide the project into smaller and smaller pieces, but stop before you get to the point of listing out every action that must be taken. Remember to focus on concrete deliverables rather than actions.

3.Depending on the nature of your project, start dividing by project phases, specific large deliverables, or sub-tasks.

RULES FOR MAKING WBS:

- The 100% rule: The work represented by your WBS must include 100% of the work necessary to complete the goal.
- Mutually exclusive**:** Do not include a sub-task twice.
- Outcomes, not actions**:** Remember to focus on deliverables and outcomes rather than actions.
- The 8/80 rule**:** A work package should take no less than eight hours of effort, but no more than 80. Other rules suggest no more than ten days (which is the same as 80 hours if you work full time).

SAMPLE OUTPUT:
Work breakdown structure to build a house:



WORK BREAKDOWN STRUCTURE FORMATS:

1.OUTLINE STRUCTURE:

A text outline is the simplest WBS format.

Build a House

1 Foundation

1.1 Excavate

1.1.1 Dig

1.1.2 Level

1.2 Frame

1.3 Concrete

1.3.1 Pour

1.3.2 Cure

2 Exterior

3 Interior

Hierarchical structure

This format is less visually intuitive but shows the hierarchy of tasks. Because it is a table, this format fits easily onto a page.

Level	WBS Code	Element Name
1	1	Foundations
2	1.1	Excavate
3	1.1.1	Dig
3	1.1.2	Level
2	1.2	Frame
2	1.3	Concrete
3	1.3.1	Pour
3	1.3.2	Cure
1	2	Exterior
1	3	Interior

Tabular view

A tabular view is a more visually intuitive way to show hierarchy using a table.

Level 1	Level 2	Level 3
1 Foundation	1.1 Excavate	1.1.1 Dig 1.1.2 Level
	1.2 Frame	
	1.3 Concrete	1.3.1 Pour 1.3.2 Cure
2 Exterior		
3 Interior		

WBS DICTIONARY:

A WBS dictionary is formatted like the hierarchical structure, but it includes a brief description of each work package.

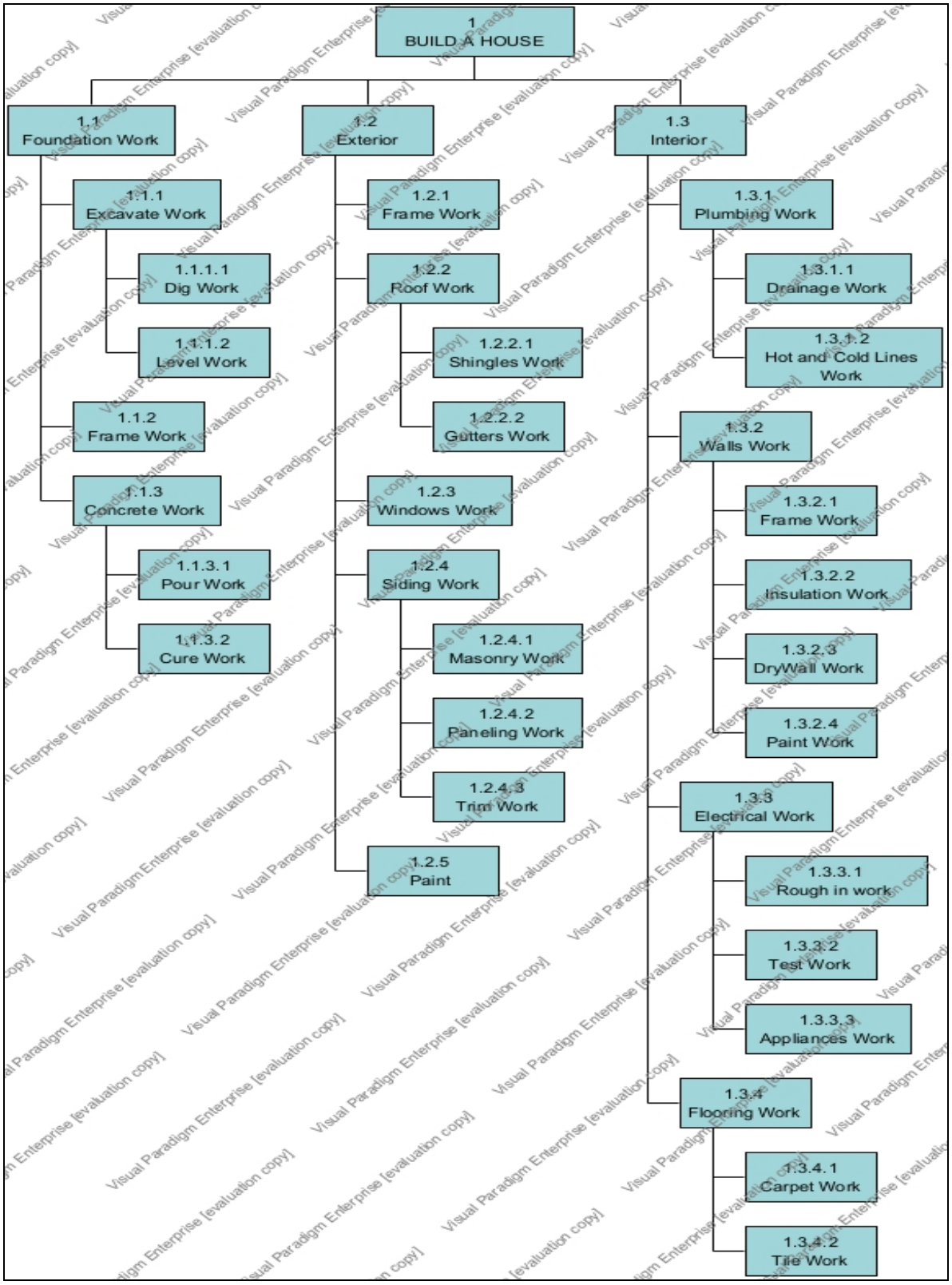
Level	WBS Code	Element Name	Element Name
1	1	Foundations	All of the work necessary to build a foundation
2	1.1	Excavate	Create a hole ready for the foundation to be framed and poured
3	1.1.1	Dig	Dig a hole of the right shape and size in the correct location
3	1.1.2	Level	Level the hole so that is packed, even, and ready to receive the foundation
2	1.2	Frame	Frame the foundation including steel supports
2	1.3	Concrete	Acquire, transport, pour, and cure the concrete foundation
3	1.3.1	Pour	Pour, pack and level the foundation
3	1.3.2	Cure	All procedures necessary for the foundation to cure successfully
1	2	Exterior	All of the work necessary to complete the exterior of the house
1	3	Interior	All of the work necessary to complete the interior of the house

PROCEDURE: (USING VISUAL PARADIGM)

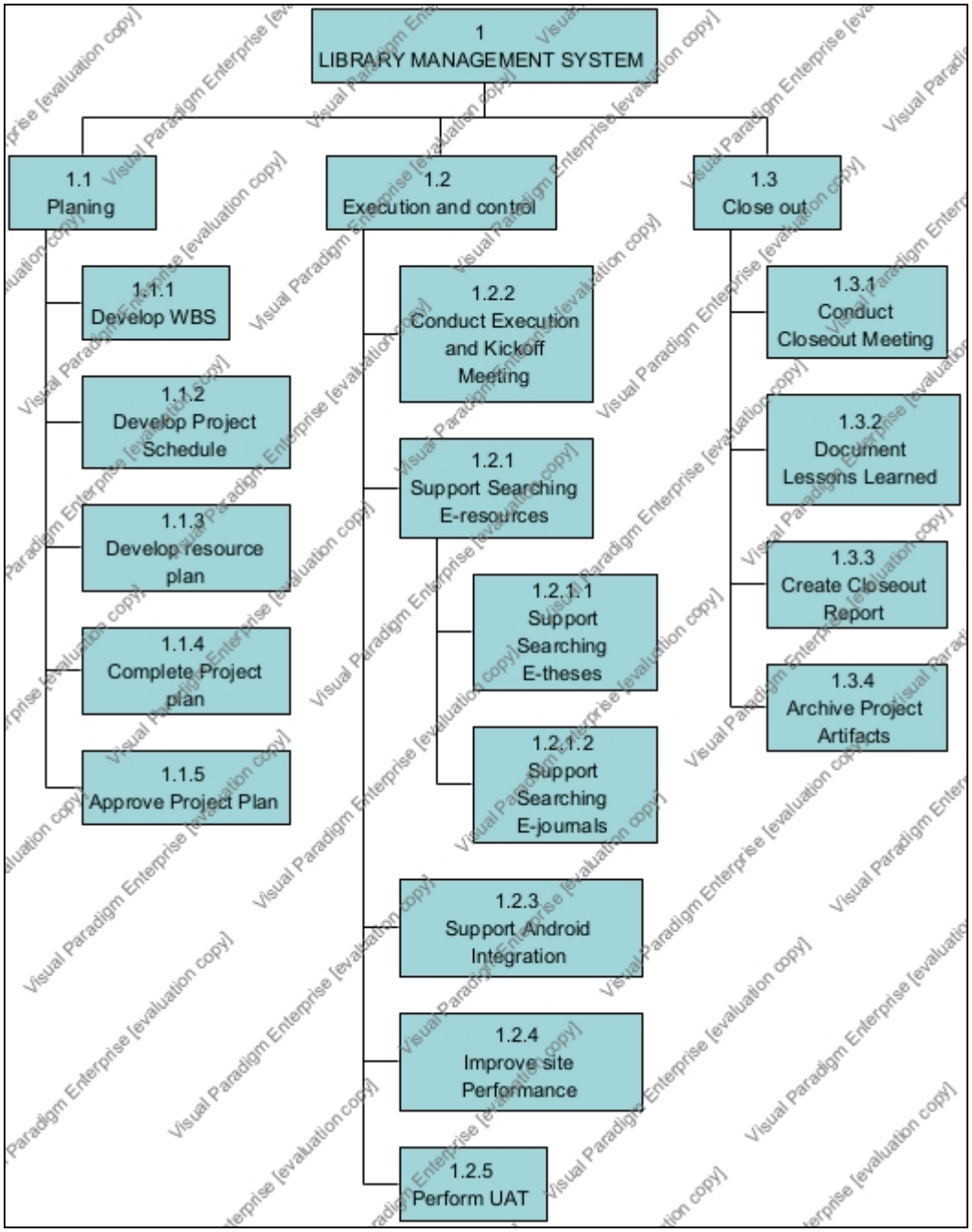
1. Select Diagram>New from the main menu.
2. Select Breakdown structure and click next.
3. Enter the diagram name and click OK. Note that diagram name will become the root element.
4. Rename the root element as needed.
5. Progressively create finer level breakdown structure elements.

OUTPUT:

Work Breakdown Structure for building a house



Work Breakdown Structure for Library Management System


1
LIBRARY MANAGEMENT SYSTEM
1.1
Planing
1.2
Execution and control
1.3
Close out
1.1.1
Develop WBS
1.1.2
Develop Project Schedule
1.1.3
Develop resource plan
1.1.4
Complete Project plan
1.1.5
Approve Project Plan
1.2.2
Conduct Execution and Kickoff Meeting
1.2.1
Support Searching E-resources
1.2.1.1
Support Searching E-theses
1.2.1.2
Support Searching E-journals
1.2.3
Support Android Integration
1.2.4
Improve site Performance
1.2.5
Perform UAT
1.3.1
Conduct Closeout Meeting
1.3.2
Document Lessons Learned
1.3.3
Create Closeout Report
1.3.4
Archive Project Artifacts

Q2. Schedule all the activities and sub-activities using the PERT/CPM charts.

Aim:To Schedule all the activities and sub-activities using the PERT/CPM charts.

Project Evaluation and Review Technique (PERT) is a procedure through which activities of a project are represented in its appropriate sequence and timing. It is a scheduling technique used to schedule, organize and integrate tasks within a project. PERT is basically a mechanism for management planning and control which provides blueprint for a particular project. All of the primary elements or events of a project have been fnally identifed by the PERT.
In this technique, a PERT Chart is made which represent a schedule for all the specifed tasks in the project. The reporting levels of the tasks or events in the PERT Charts is somewhat same as defned in the work breakdown structure (WBS).

Characteristics of PERT:

The main characteristics of PERT are as following :

1. It serves as a base for obtaining the important facts for implementing the decision-making.
2. It forms the basis for all the planning activities.
3. PERT helps management in deciding the best possible resource utilization method.
4. PERT take advantage by using time network analysis technique.
5. PERT presents the structure for reporting information.
6. It helps the management in identifying the essential elements for the completion of the project within time.

Advantages of PERT:

It has the following advantages :

1. Estimation of completion time of project is given by the PERT.
2. It supports the identifcation of the activities with slack time.
3. The start and dates of the activities of a specifc project is determined.

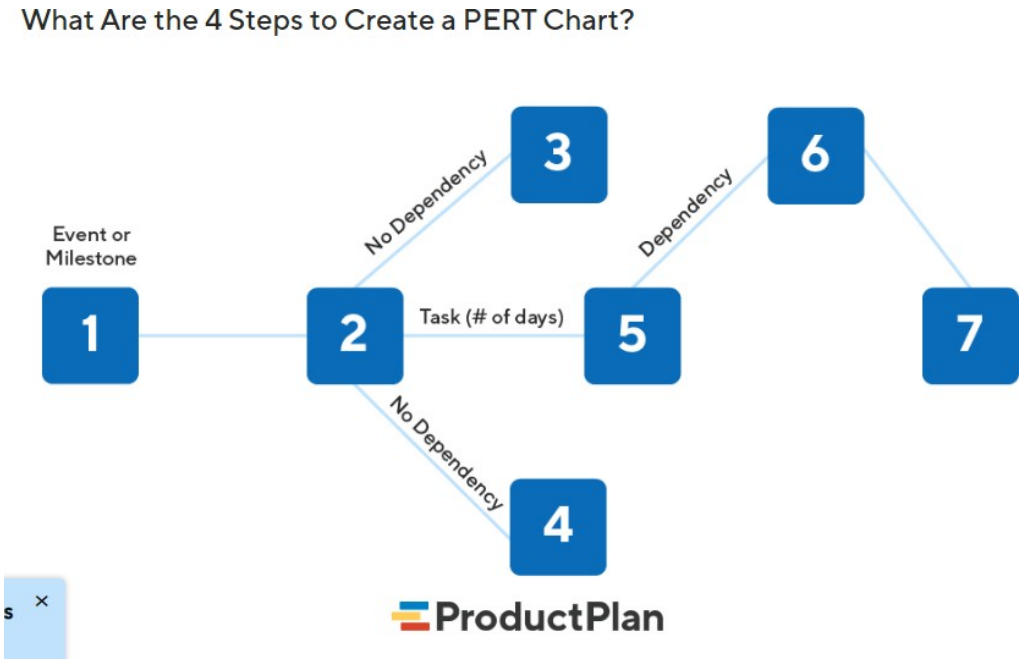
4. It helps project manager in identifying the critical path activities.
5. PERT makes well organized diagram for the representation of large amount of data.

Disadvantages of PERT:

It has the following disadvantages :

1. The complexity of PERT is more which leads to the problem in implementation.
2. The estimation of activity time are subjective in PERT which is a major disadvantage.
3. Maintenance of PERT is also expensive and complex.
4. The actual distribution of may be diferent from the PERT beta distribution which causes wrong assumptions.
5. It under estimates the expected project completion time as there is chances that other paths can become the critical path if their related activities are deferred.

What Are the 4 Steps to Create a PERT Chart?



To create a PERT chart, a project management team should follow these steps.

Step 1: Identify all of the project's activities.

First, define all of the major phases, milestones, and tasks needed to complete the project.

Step 2: Identify dependencies

If you determine some tasks or activities have dependencies, you will want to depict those tasks with directional arrows. This will ensure your team knows the sequence they need to tackle each task.

Step 3: Draw your chart.

The next step is to take the events and milestones (numbered nodes) you've identified and draw them out. Then write out the tasks and activities that the team must complete between each node, using directional arrows or divergent arrows accordingly.

Step 4: Establish timelines for all activities.

You should now set a timeframe when the team will need to complete those tasks along with all arrows. For example, in our mockup above, you can see the "Train sales" activity has a timeframe of 1 day. This can represent the estimated timeframe and/or deadline you set for the activity.

Critical Path Method (CPM)

Critical Path Method (CPM) is a method used in project planning, generally for project scheduling for the on-time completion of the project. It actually helps in the determination of the earliest time by which the whole project can be completed. There are two main concepts in this method namely critical task and critical path. **Critical task** is the task/activity which can't be delayed otherwise the completion of the whole project will be delayed. It must be completed on-time before starting the other dependent tasks.

Critical path is a sequence of critical tasks/activities and is the largest path in the project network. It gives us the minimum time which is required to complete the whole project. The activities in the critical path are known as critical activities and if these activities are delayed then the completion of the whole project is also delayed.

Major steps of the Critical Path Method:

1. Identifying the activities
2. Construct the project network
3. Perform time estimation using forward and backward pass
4. Identify the critical path

The table given below contains the activity label, its respective duration (in weeks) and its precedents. We will use critical path method to find the critical path and activities of this project.

Activity	Duration (in weeks)	Precedents

A	6	–
B	4	–

Activity	Duration (in weeks)	Precedents
C	3	A
D	4	B
E	3	B
F	10	–
G	3	E, F
H	2	C, D

Rules for designing the Activity-on-Node network diagram:

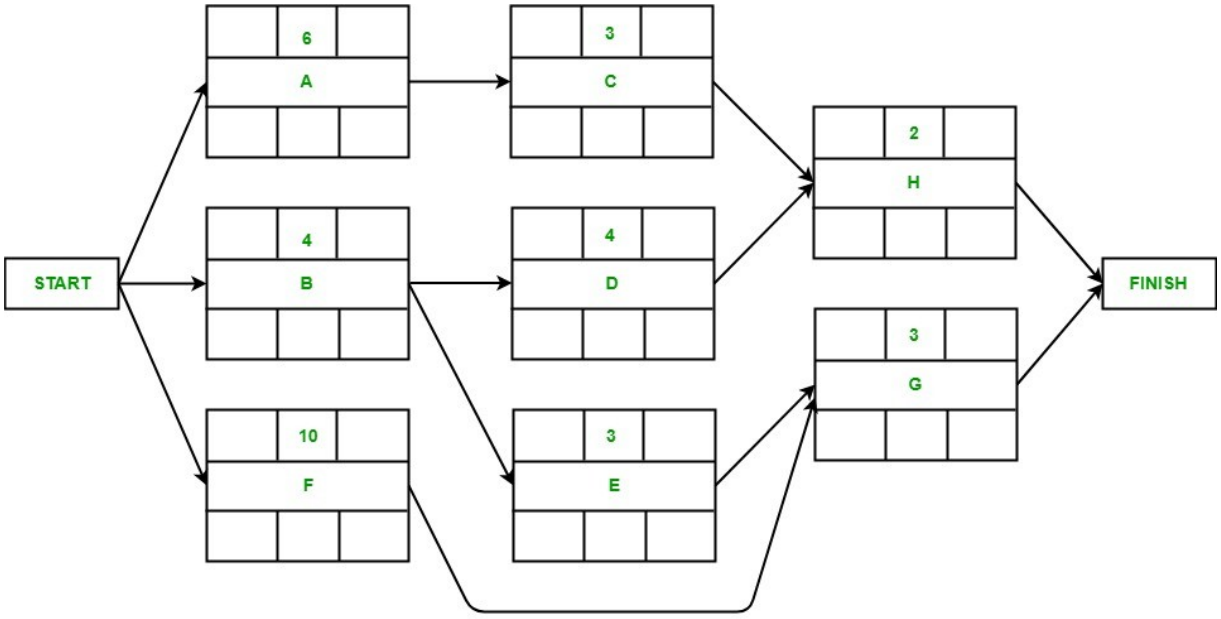
- ☐ A project network should have only one start node
- ☐ A project network should have only one end node
- ☐ A node has a duration
- ☐ Links normally have no duration
- ☐ "Precedents" are the immediate preceding activities
- ☐ Time moves from left to right in the project network
- ☐ A network should not contain loops
- ☐ A network should not contain dangles

Node Representation:

- ☐ **Activity label** is the name of the activity represented by that node.
- ☐ **Earliest Start** is the date or time at which the activity can be started at the earliest.
- ☐ **Earliest Finish** is the date or time at which the activity can completed at the earliest.
- ☐ **Latest Start** is the date or time at which the activity can be started at the latest.

- **Latest Finish** is the date or time at which the activity can be finished at the latest.
- **Float** is equal to the difference between earliest start and latest start or earliest finish and latest finish.

Activity-On-Node diagram:



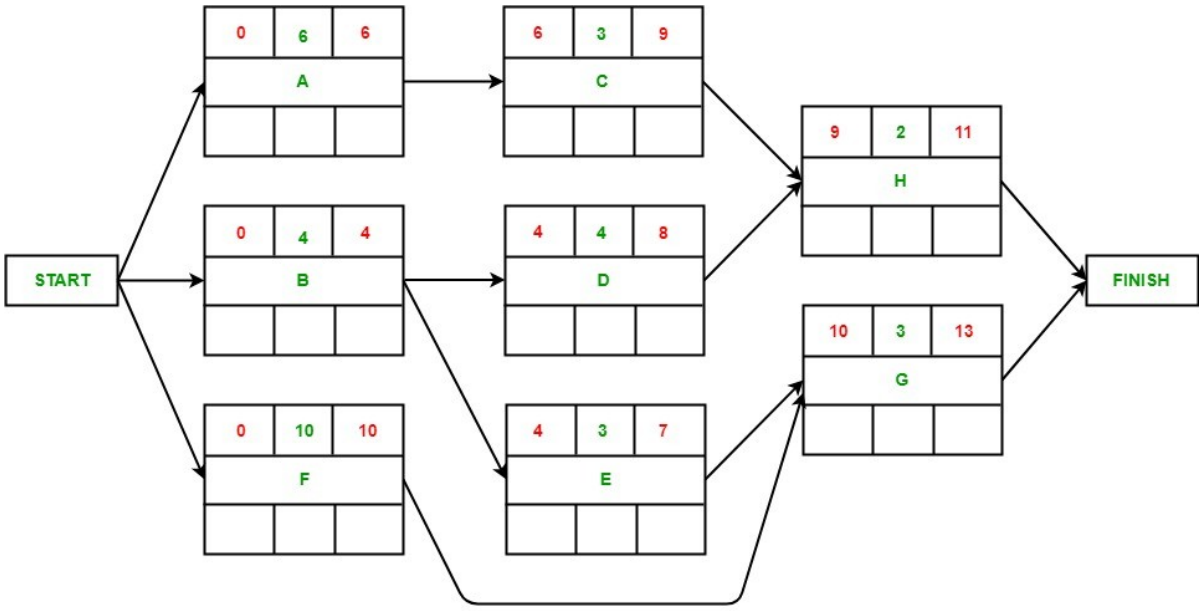
Forward Pass:

The forward pass is carried out to calculate the earliest dates on which each activity may be started and completed.

1. Activity A may start immediately. Hence, earliest date for its start is zero i.e. ES(A) = 0. It takes 6 weeks to complete its execution. Hence, earliest it can finish is week 6 i.e. EF(A) = 6.
2. Activity B may start immediately. Hence, earliest date for its start is zero i.e. ES(B) = 0. It takes 4 weeks to complete its execution. Hence, earliest it can finish is week 4 i.e. EF(B) = 4.
3. Activity F may start immediately. Hence, earliest date for its start is zero i.e. ES(F) = 0. It takes 10 weeks to complete its execution. Hence, earliest it can finish is week 10 i.e. EF(F) = 10.
4. Activity C starts as soon as activity A completes its execution. Hence, earliest week it can start its execution is week 6 i.e. ES(C) = 6. It takes 3 weeks to complete its execution. Hence, earliest it can finish is week 9 i.e. EF(C) = 9.
5. Activity D starts as soon as activity B completes its execution. Hence, earliest week it can start its execution is week 4 i.e. ES(D) = 4. It takes 4 weeks to complete its execution. Hence, earliest it can finish is week 8 i.e. EF(D) = 8.
6. Activity E starts as soon as activity B completes its execution. Hence, earliest week it can start its execution is week 4 i.e. ES(E) = 4. It takes 3

weeks to complete its execution. Hence, earliest it can finish is week 7 i.e. EF(E) = 7.

7. Activity G starts as soon as activity E and activity F completes their execution. Since, activity requires the completion of both for starting its execution, we would consider the MAX(ES(E), ES(F)). Hence, earliest week it can start its execution is week 10 i.e. ES(G) = 10. It takes 3 weeks to complete its execution. Hence, earliest it can finish is week 13 i.e. EF(G) = 13.
8. Activity H starts as soon as activity C and activity D completes their execution. Since, activity requires the completion of both for starting its execution, we would consider the MAX(ES(C), ES(D)). Hence, earliest week it can start its execution is week 9 i.e. ES(H) = 9. It takes 2 weeks to complete its execution. Hence, earliest it can finish is week 11 i.e. EF(H)= 11.



Backward Pass:

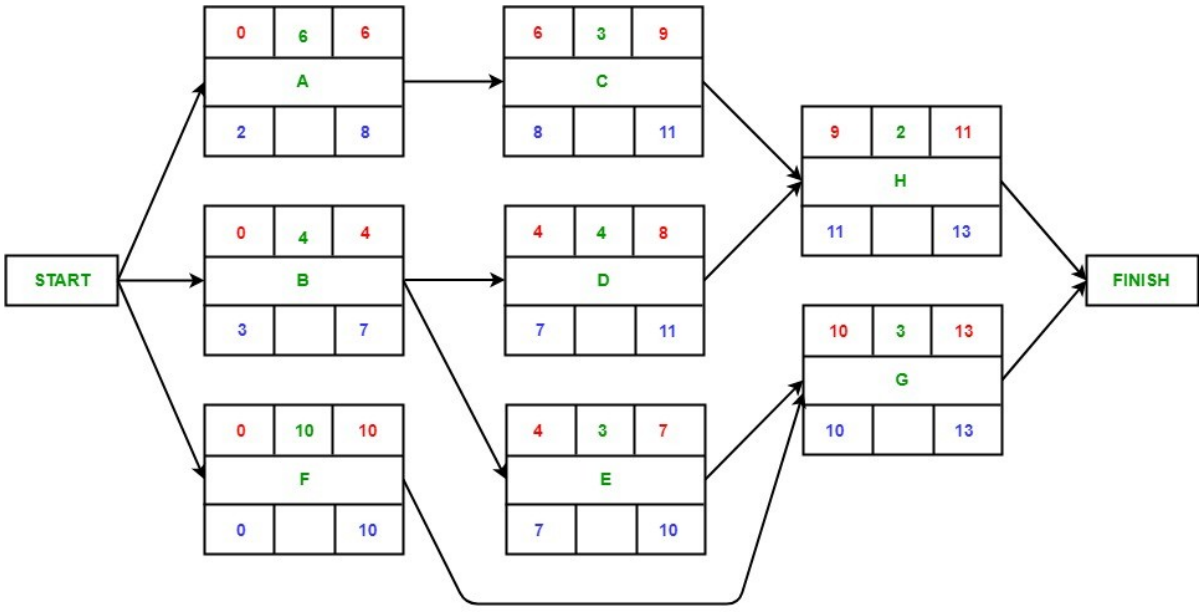
The backward pass is carried out to calculate the latest dates on which each activity may be started and finished without delaying the end date of the project.

Assumption: Latest finish date = Earliest Finish date (of project).

1. Activity G's latest finish date is equal to the earliest finish date of the precedent activity of finish according to the assumption i.e. LF(G) = 13. It takes 3 weeks to complete its execution. Hence, latest it can start is week 10 i.e. LS(G) = 10.
2. Activity H's latest finish date is equal to the earliest finish date of the precedent activity of finish according to the assumption i.e. LF(H) = 13. It

takes 2 weeks to complete its execution. Hence, latest it can start is week 11 i.e. LS(H) = 11.

3. The latest end date for activity C would be the latest start date of H i.e. LF(C) = 11. It takes 3 weeks to complete its execution. Hence, latest it can start is week 8 i.e. LS(C) = 8.
4. The latest end date for activity D would be the latest start date of H i.e. LF(D) = 11. It takes 4 weeks to complete its execution. Hence, latest it can start is week 7 i.e. LS(D) = 7.
5. The latest end date for activity E would be the latest start date of G i.e. LF(G) = 10. It takes 3 weeks to complete its execution. Hence, latest it can start is week 7 i.e. LS(E) = 7.
6. The latest end date for activity F would be the latest start date of G i.e. LF(G) = 10. It takes 10 weeks to complete its execution. Hence, latest it can start is week 0 i.e. LS(F) = 0.
7. The latest end date for activity A would be the latest start date of C i.e. LF(A) = 8. It takes 6 weeks to complete its execution. Hence, latest it can start is week 2 i.e. LS(A) = 2.
8. The latest end date for activity B would be the earliest of the latest start date of D and E i.e. LF(B) = 7. It takes 4 weeks to complete its execution. Hence, latest it can start is week 3 i.e. LS(B) = 3.

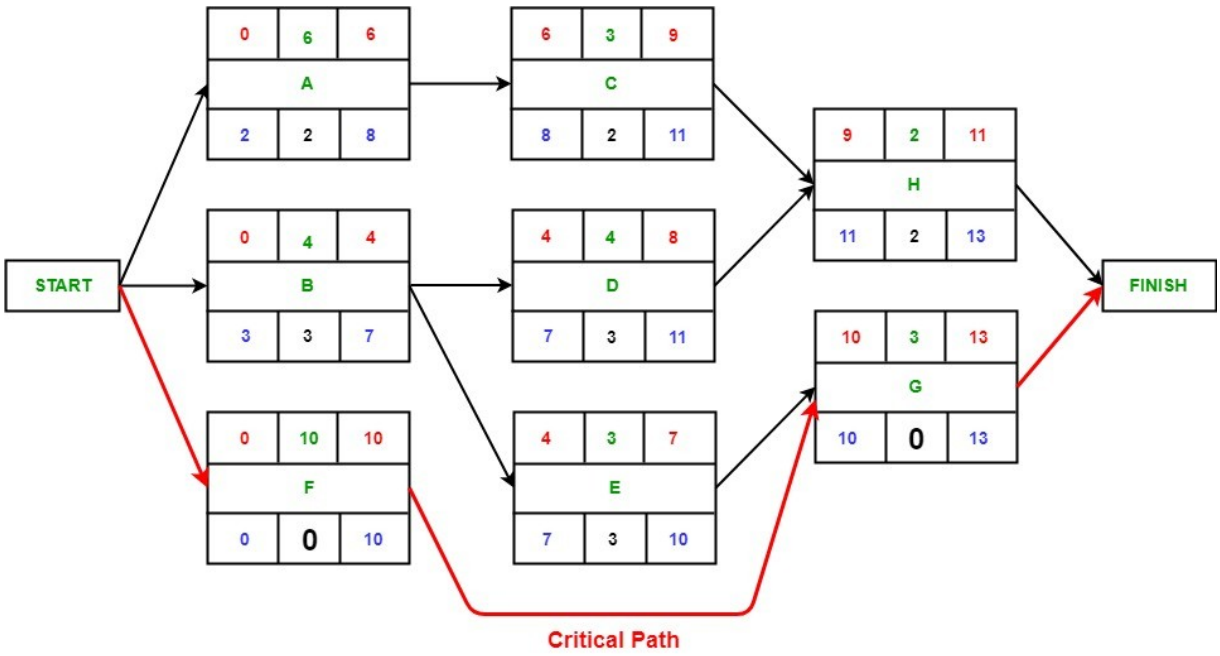


Identifying Critical Path:

Critical path is the path which gives us or helps us to estimate the earliest time in which the whole project can be completed. Any delay to an activity on this critical path will lead to a delay in the completion of the whole project. In order to identify the critical path, we need to calculate the activity float for each activity.
Activity float is actually the difference between an activity's Earliest start and its latest start date or the difference between the activity's Earliest finish and its latest finish date and it

indicates that how much the activity can be delayed without delaying the completion of the whole project. If the float of an activity is zero, then the activity is an critical activity and must be added to the critical path of the project network. In this example, activity F and G have zero float and hence, are critical activities.

OUTPUT :



Applications of Network Analysis (PERT and CPM)

1. Research and Development Project (R and D Projects)
2. Equipment Maintenance and hauling
3. Construction Projects (Buildings, bridges, and Dams)
4. Setting up new industries
5. Planning and launching of new products
6. Design of plants, machines, and systems
7. Shifting the manufacturing location to another location
8. Control of production in large shops
9. Market penetration programs
10. The organization of big programs, conferences etc.

3. Define use cases and represent them in use-case document for all the stakeholders of the System to be automated.

Aim: To define use cases and represent them in use-case document for all the stakeholders of the System to be automated.

Introduction:

USE CASES

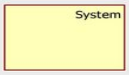
- The requirements really just give us an outline of what we are trying to build. Use Cases are the next step in the design process
- Use cases integrate the requirements into a comprehensive package that describes the interaction of the user with the system
- The Use Case should describe the interaction between the actor and the system - what the actor does and how the system reacts.
- Use Cases are also extremely valuable since the stakeholders of the application can easily understand them. Therefore, they help to eliminate misunderstandings about the scope and functionality of the system.

USE CASE DIAGRAM:

A use case diagram is agraphical depiction of a user's possible interactions with the system . A use case diagram shows various use cases and different types of users the

system.

Use Case Symbol

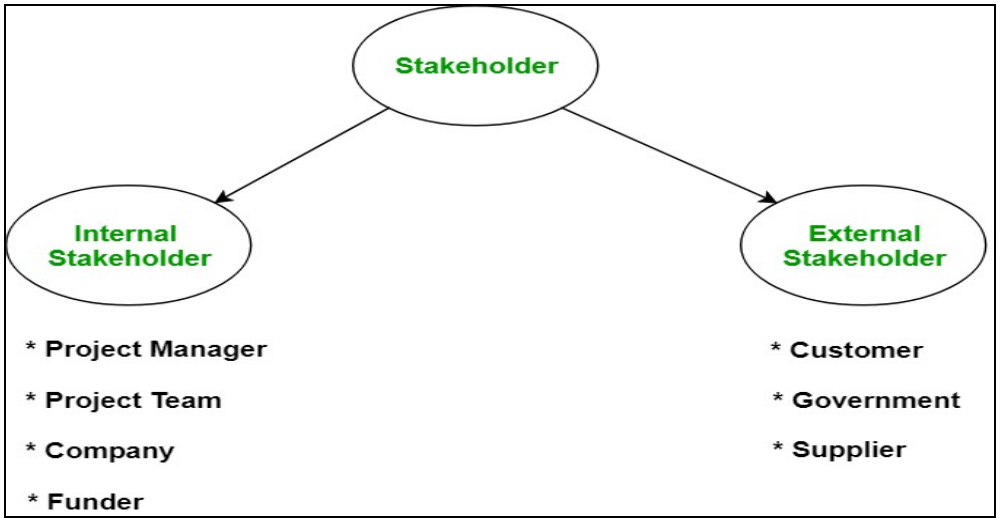
SYMBOL	NAME OF SYMBOL	EXPLANATION
	Actor	Accessing use case
	Use Case	Show what the system do
	Association	Relate the actor with use case
	System Boundary	Show boundary between system and its environment

IMPORTANCE OF USE CASES:

- The list of goal names provides the shortest summary of what the system will offer
- It gives an overview of the roles of each and every component in the system. It will help us in defining the role of users, administrators etc.
- It helps us in extensively defining the user's need and exploring it as to how it will work.
- It provides solutions and answers to many questions that might pop up if we start a project unplanned.

STAKE HOLDERS

The term Software Project Stakeholder refers to, "a person, group or company that is directly or indirectly involved in the project and who may affect or get affected by the outcome of the project". The main difference between the stakeholder and Actor is that stakeholders does not appear on the uses.



USE CASE DOCUMENT

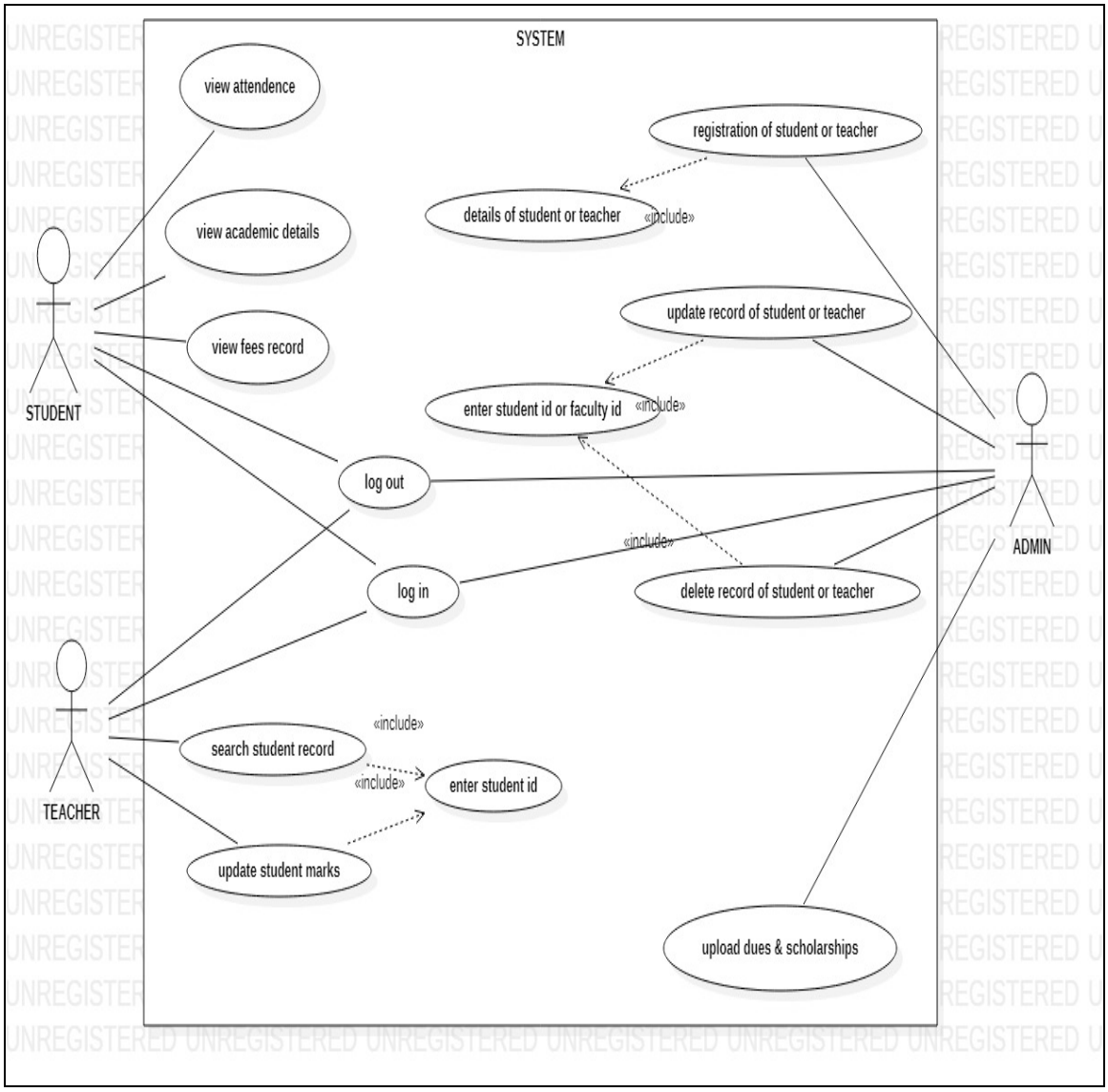
Here we created a use case document based on the example of the student management system

- This documentation gives a complete overview of the distinct ways in which the user interacts with a system to achieve the goal. Better documentation can help to identify the requirement for a software system in a much easier way.
- This documentation can be used by Software developers, software testers as well as Stakeholders.

USES OF THE DOCUMENTS:

- Developers use the documents for implementing the code and designing it.
- Testers use them for creating the test cases.
- Business stakeholders use the document for understanding the software requirements.
- The stake holders of student management system are students,teachers.

STUDENT MANAGEMENT SYSTEM


SYSTEM
view attendence
registration of student or teacher
details of student or teacher
«include»
view academic details
update record of student or teacher
view fees record
STUDENT
enter student id or faculty id
«include»
log out
ADMIN
«include»
log in
delete record of student or teacher
«include»
search student record
«include»
enter student id
TEACHER
update student marks
upload dues & scholarships

For Example, Consider the 'view Student Marks' case, in a Student Management System.

Use case Name: view Student Marks Actors: students, teacher.

Pre-Condition:

1) The system must be connected to the network.

2) Actors must have a 'Student ID'.

Main Scenario	Serial Number	Steps
A: Actor/ S: System	1	Enter Student Name
	2	System Validates Student Name
	3	Enter Student ID
	4	System Validates Student ID
	5	System shows Student Marks
Extensions	3a	Invalid Student ID S: Shows an error message
	3b	Invalid Student ID entered 4 times. S: Application Closes

USE CASE TESTING

- A Use Case in Testing is a brief description

of a particular use of the software application by an actor or user.

- Use Case Testing is a software testing technique that helps to identify test cases that cover entire system on a transaction by transaction basis from start to end. Test cases are the interactions between users and software application.

For Example, Consider the 'view Student Marks' case, in a Student Management System.

Use case Name: view Student Marks Actors: students, teacher.

Pre-Condition:

1) The system must be connected to the network.

2) Actors must have a 'Student ID'.

Main Scenario	Serial Number	Steps
A: Actor/ S: System	1	Enter Student Name
	2	System Validates Student Name
	3	Enter Student ID
	4	System Validates Student ID
	5	System shows Student Marks
Extensions	3a	Invalid Student ID S: Shows an error message
	3b	Invalid Student ID entered 4 times. S: Application Closes

USE CASE TESTING

- A Use Case in Testing is a brief description of a particular use of the software application by an actor or user.
- Use Case Testing is a software testing technique that helps to identify test cases that cover entire system on a transaction by transaction basis from start to end. Test cases are the interactions between users and software application.

TEST CASE FOR VIEW STUDENTS MARKS CASE:

Test Cases	Steps	Expected Result
A	View Student Mark List 1 -Normal Flow	

Test Cases	Steps	Expected Result
1	Enter Student Name	User can enter Student name
2	Enter Student ID	User can enter Student ID
3	Click on View Mark	System displays Student Marks
B	**View Student Mark List 2-Invalid ID**	
1	Repeat steps 1 and 2 of View Student Mark List 1	
2	Enter Student ID	System displays Error message

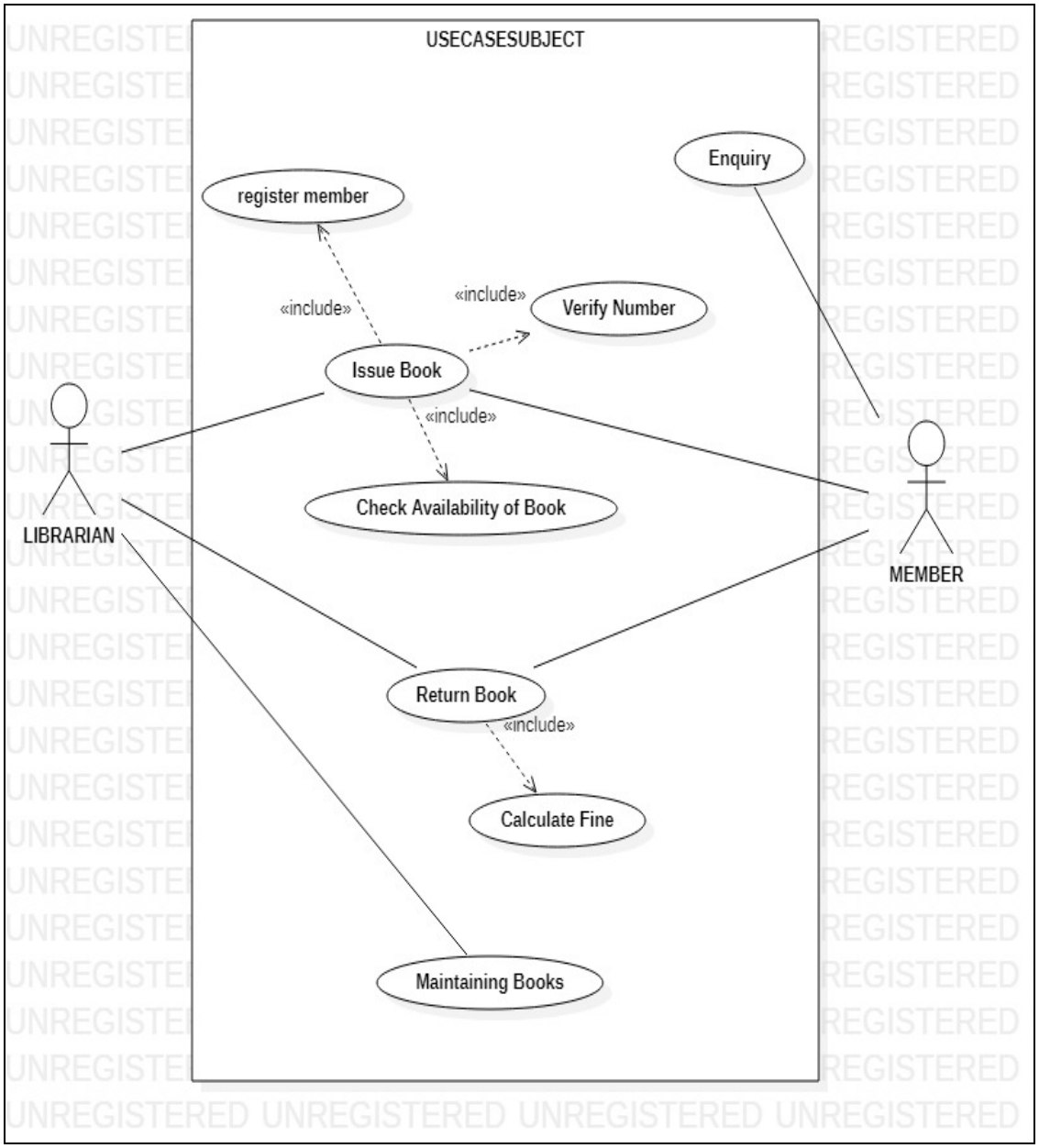
- The table displays the 'Test Case' corresponding to the 'View Student Mark' case as shown above.
- The best way to write test cases is to write the test cases for 'the Main scenario' first, and then write them for 'Alternate Steps'. The stepsin Test Cases are got from Use Case documents. The very first Step of 'View Student Mark' case, 'Enter Student Name' will become the first Step in the 'Test Case'.
- The User/Actor must be able to enter it. This becomes the Expected result.

We can seek the help of test design technique like 'boundary analysis', 'equivalence partitioning' ,while we preparing the test cases. The test design technique will help to reduces the no of test cases and there by reducing time for testing.

PROCEDURE:

We use Star UML for the creation of the use case diagrams
Steps:
1.First open StarUML
2. Select MODEL
3.In that select add diagram option and choose use case diagram in menu bar.
4.To create an use case subject select use case subject in ToolBox drag the diagram as the size of use case subject.
5.To create an Actor select Actor in ToolBox and name the actor according to the use case.
6.Create as many Actors as per the required use case diagram.
7.To create an use case select an use case in ToolBox and enter the content of the use case.
8.Create as many as use cases required and draw the Association from Actor to Use case by selecting and locking out the Association.
9.To create an include select include in ToolBox and drag from a Use case and drop on the another use case to be included.
10.The Use case diagram of student management system using the StarUML looks like:

OUTPUT:: LIBRARY MANAGEMENT SYSTEM


USECASESUBJECT
Enquiry
register member
«include»
«include»
Verify Number
Issue Book
«include»
Check Availability of Book
LIBRARIAN
MEMBER
Return Book
«include»
Calculate Fine
Maintaining Books

4.Identify and analyze all the possible risks and its risk mitigation plan for the system to be automated.

AIM: To identify all possible risks and its risk mitigation plans to mitigate the risks.

INTRODUCTION:

Risk:

A RISK is an uncertain event or condition that, if it occurs, has a positive or negative effect on the project objective.
It's an activity or event that may compromise the success of a software development project.
These include terminations, discontinuities, schedule delays, cost underestimation, and overrun of project resources.
Not all risks are negative. Some events (like finding an easier way to do an activity) or conditions (like lower prices for certain materials) can help your project. When this happens, we call it an opportunity; but it's still handled just like a risk.

Risk management:

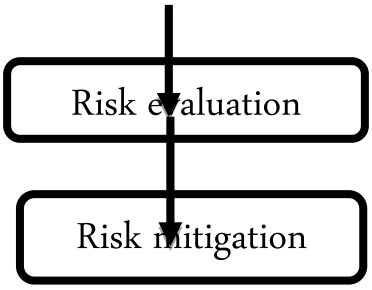
Managing risks on projects is a process that includes risk assessment and a mitigation strategy for those risks. Risk assessment includes both the identification of potential risk and the evaluation of the potential impact of the risk. A risk mitigation plan is designed to eliminate or minimize the impact of the risk events occurrences that have a negative impact on the project. Identifying risk is both a creative and a disciplined process. The creative process includes brainstorming sessions where the team is asked to create a list of everything that could go wrong. All ideas are welcome at this stage with the evaluation of the ideas coming later.

Risk management steps:

There are 3 risk management steps

1. Risk identification
2. Risk evaluation
3. Risk mitigation

Risk identification



Risk management steps

The first step is to identify the risk, then evaluating the risk and then risk mitigation plans are implemented.

Risk identification:

Some companies and industries develop risk checklists based on experience from past projects. These checklists can be helpful to the project manager and project team in identifying both specific risks on the checklist and expanding the thinking of the team.

Identifying the sources of risk by category is another method for exploring potential risk on a project.

Risk evaluation:

After the potential risks have been identified, the project team then evaluates each risk based on the probability that a risk event will occur and the potential loss associated with it. Not all risks are equal. Some risk events are more likely to happen than others, and the cost of a risk can vary greatly. Evaluating the risk for probability of occurrence and the severity or the potential loss to the project is the next step in the risk management process.

Risk mitigation:

Risk mitigation simply means to reduce adverse effect and impact of risks that are harmful to project and Business continuity.

PROCEDURE:

Mitigation:

Mitigation means reducing risk of loss from the occurrence of any undesirable event.

`Risk mitigation:

Risk mitigation simply means to reduce adverse effect and impact of risks that are harmful to project and Business continuity.

For example, a jewellery store might mitigate the risk of theft, by having a security system or even a security guard at entrance.

- After the risk has been identified and evaluated, the project team develops a risk mitigation plan, which is a plan to reduce the impact of an unexpected event.

RISK MITIGATION PLANS:
(risk mitigation strategies)

The project team mitigates risks in various ways:

1. Avoidance of risk
2. Transference of risk.
3. Controlling risk.
4. Assume and accept risk.

1.Avoid Risk:

It stands for not undertaking an activity which causes risk.

The best thing you can do with a risk is avoid it. If you can prevent it from happening, it definitely won't hurt your project.

- Risk avoidance usually involves developing an alternative strategy that has a higher probability of success but usually at a higher cost associated with accomplishing a project task.

2.Transference of Risk:

Reassign organizational accountability, responsibility and authority to another stake holder to accept the risk.

Risk sharing involves partnering with others to share responsibility for the risky activities.

- If a risk event does occur, then the partnering company absorbs some or all of the negative impact of the event. The company will also derive some of the profit or benefit gained by a successful project.

3.Controlling risk:

It stands for reducing the severity of the impact.

If you can't avoid the risk, you can control it. This means taking some sort of action that will cause it to do as little damage to your project as possible.

4.Accept Risk:

Acknowledge the existence of a particular risk, and make a deliberate decision to accept it without engaging in special efforts to control it.

- When you can't avoid, control, or transfer a risk, then you have to accept it. But even when you accept a risk, at least you've looked at the alternatives and you know what will happen if it occurs. If you can't avoid the risk, and there's nothing you can do to reduce its impact, then accepting it is your only choice.

Let's look into a EXAMPLE which clearly explains the above 4 startegies.

Let's consider,If your project requires that you stand on the edge of a cliff, then there's a risk that you could fall. If it's very windy out or if the ground is slippery and uneven, then falling is more likely.

When you're planning your project, risks are still uncertain: they haven't happened yet. But eventually, some of the risks that you plan for do happen, and that's when you have to deal with them. There are four basic ways to handle a risk.

Avoid- he easiest way to avoid this risk is to walk away from the cliff, but that may not be an option on this project.

Transfer- The most common way to do this is to buy *insurance*.

Control- This means taking some sort of action that will cause it to do as little damage to your project as possible.

Accept- When you can't avoid, reduce, or transfer a risk, then you have to accept it. Which means jumping of the cliff.

OUTPUT:

These are few common project risks

1. Project purpose and need is not well defined.
2. Project design and deliverable definition is incomplete.
3. Project schedule is not clearly defined or understood.
4. No control over staff priorities.
5. Consultant or contractor delays.
6. Lake of communication, causing lake of clarity and confusion.
7. Customer refuses to approve deliverables / Milestones or delays approval putting pressure on project manager to 'work at risk '.
8. Legal action delays or pauses project.

Through risk mitigation plans i.e, avoid , transfer, control, accept.

All possible risks are mitigated.

5. Diagnose any risk using Ishikawa Diagram (Can be called as Fish Bone Diagram or Cause & Effect Diagram).

AIM : IDENTIFY AND ANALYZING THE RISK USING ISHIKAWA DIAGRAM .

ISHIKAWA DIAGRAM

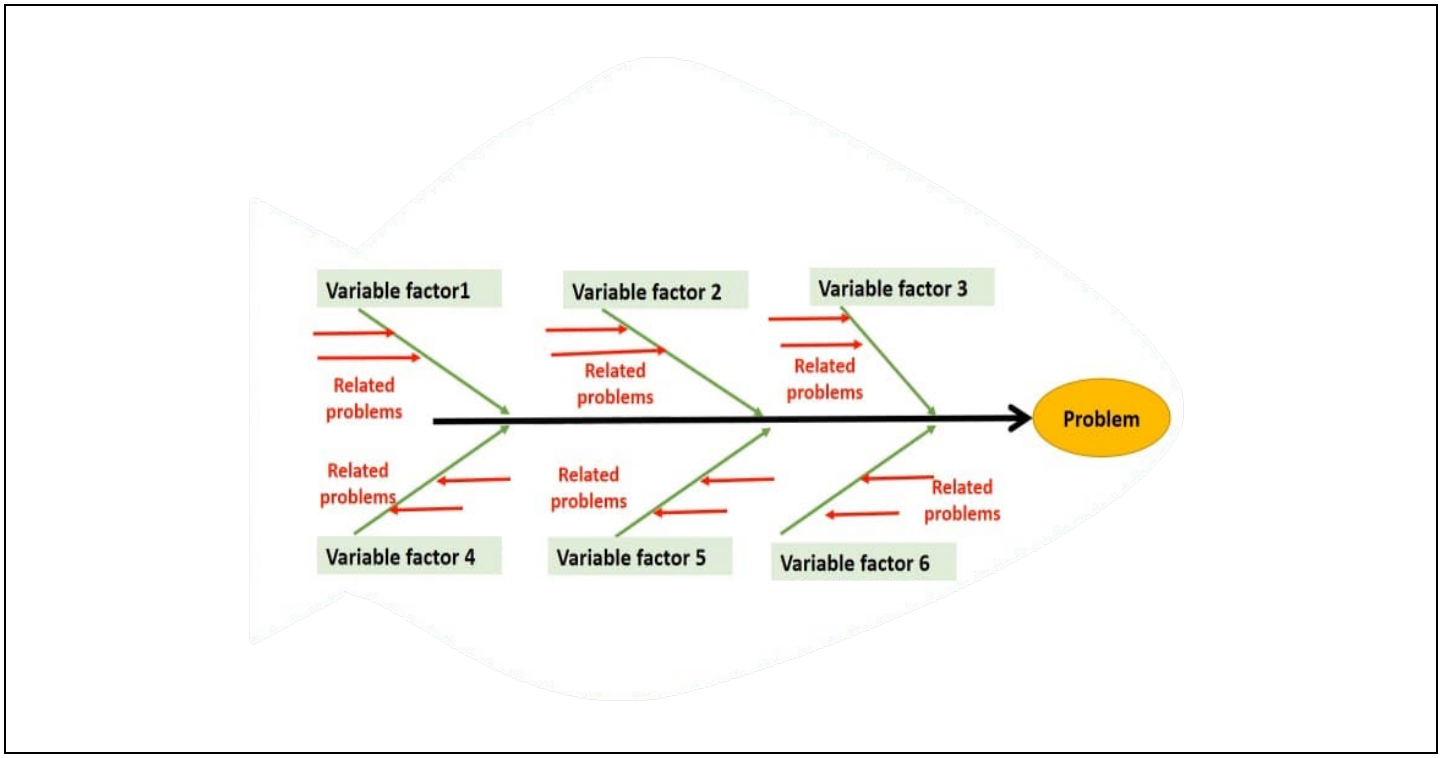
- *Fishbone diagram also called as cause & effect diagram was first used by Dr. Kaoru Ishikawa of the University of Tokyo in 1943-hence it is also known as Ishikawa Diagram.*

➢ PURPOSE OF ISHIKAWA DIAGRAM:

- *Identifies various causes affecting a process.*
- *Helps groups in reaching a common understanding of a problem.*
- *It helps managers to track down the reasons for imperfections, variations , defects or failures in a process*

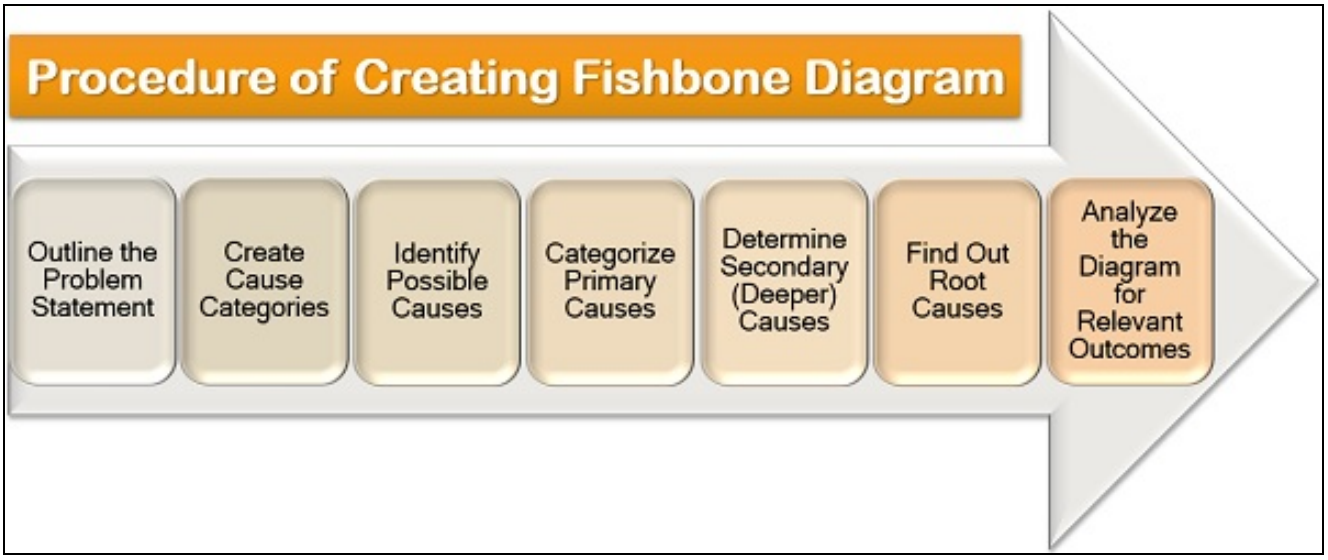
STRUCTURE OF ISHIKAWA DIAGRAM

- *The structure of the Ishikawa diagram looks just like a fish's skeleton with the problem at its head and the causes for the problem feeding into the spine.*



PROCEDURE FOR CREATING A FISHBONE DIAGRAM

- *Organize an appropriate team.*
- *Identify the problem.*
- *Brainstorm the major categories of causes(6 M's).*
- *Identify possible causes(5 Why's) for the respective categories.*
- *Analyze the diagram.*

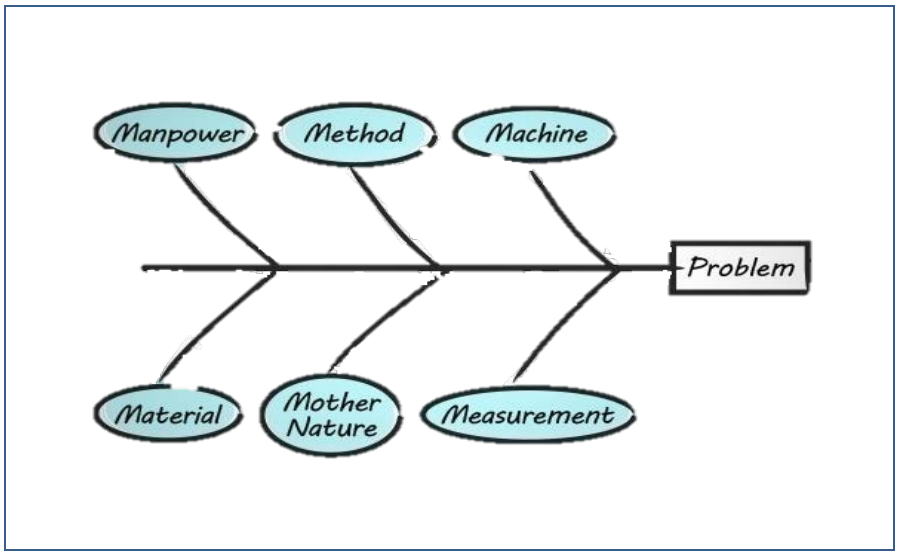


BASIC TYPES OF FISHBONE DIAGRAM

One of the first steps in creating a fishbone diagram is determining the factors that contribute to variations within a process.

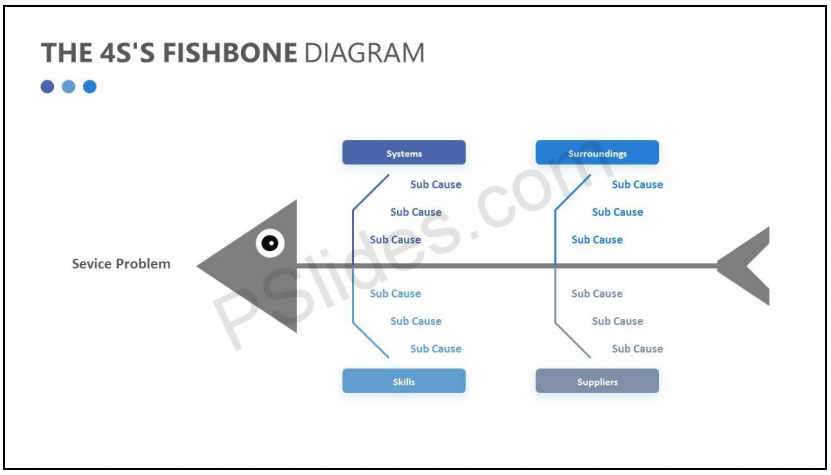
▪ THE 6M'S FISHBONE DIAGRAM:

- *The 6M's here are Man,Materials , Machine , Methods, Measurement and Mother nature .These categories are mostly used to draw a cause and effect diagram in the manufacturing industry.*



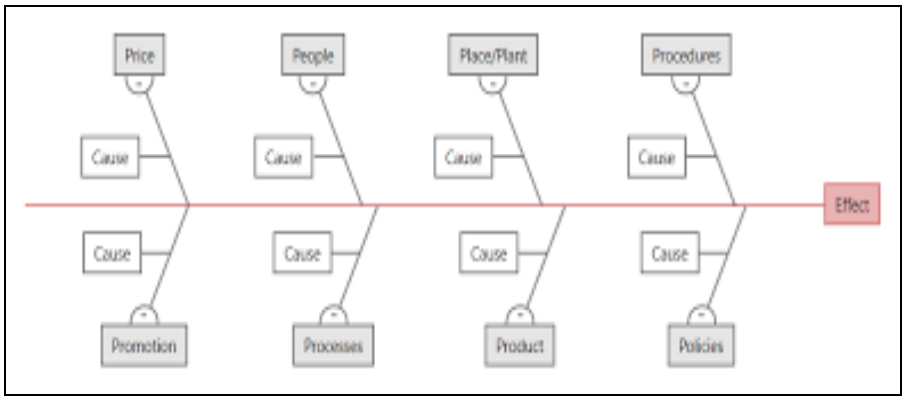
❑ 4S FISHBONE DIAGRAM:

If the organization operates in the service industry, then this fishbone diagram is mostly used. All potential causes, along with their information, are grouped into four sections: systems, surroundings, skills and suppliers.



❑ 8P FISHBONE DIAGRAM:

This type of fishbone is named after the 8 categories that make it up: Procedures, Policies, Place, Product, People, Processes, Price and Promotion. This variation is also commonly used in the service industry, but can be applied in nearly any type of business or industry

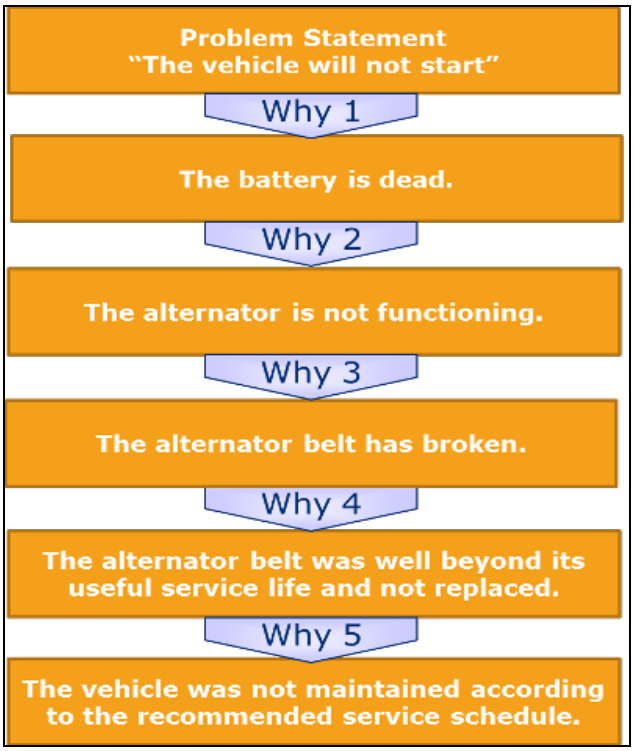


FISHBONE DIAGRAM AND 5WHYS

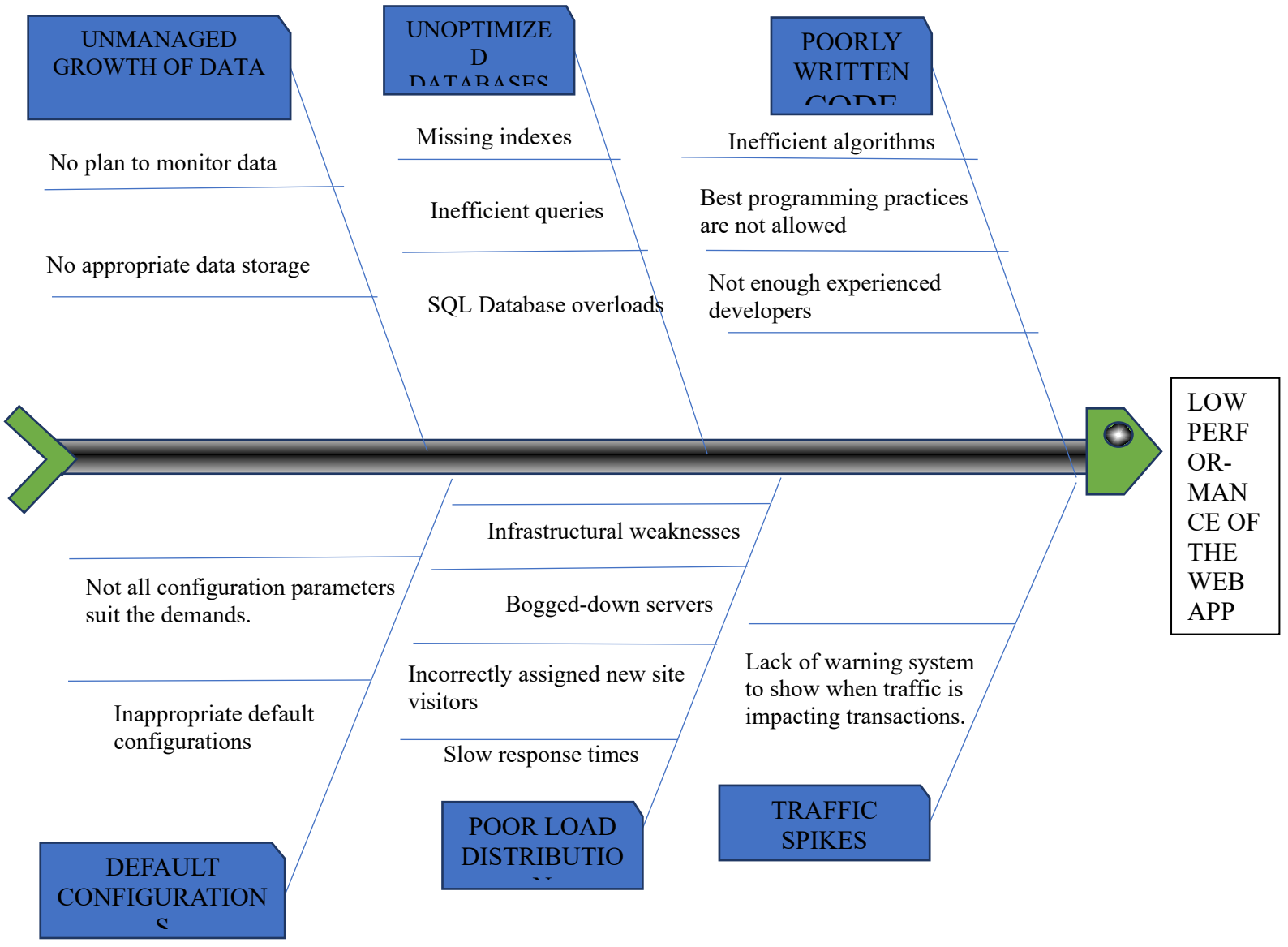
- ***Five whys*** *(or* ***5 whys****) is an iterative interrogative technique used to explore the cause-and-effect relationship underlying a particular problem .*

- *The primary goal of the technique is to determine the root cause of a defect or problem by repeating the question "Why?*

- *Each answer forms the basis of the next question.*



EXAMPLE OF FISHBONE DIAGRAM



ADVANTAGES AND DISADVANTAGES OF FISHBONE DIAGRAM

❖ ***ADVANTAGES:***

✓ *Easy to understand.*

✓ *Ensures the teams to cover all possible causes.*

✓ *Works well with the 5Whys to drill down to root cause quickly.*

✓ *Can be applied to a range of problems.*

❖ **DISADVANTAGES:**

- *Irrelevant potential causes can cause confusion.*
- *Complex issues may lead to a messy diagram.*
- *The output from brainstorming is only as good as your brainstorming session.*
- *Based on opinion rather than evidence, it needs to testing to prove results.*

-------***-------

6 Define Complete Project plan for the system to be automated using Microsoft Project Tool.

AIM

To define complete Project plan for the system to be automated using MS Project tool.

DESCRIPTION

MS PROJECT

Microsoft Project is a project management software program developed and sold by Microsoft, designed to assist a project manager in developing a schedule, assigning resources to tasks, tracking progress, managing the budget, and analyzing workloads

Project Management

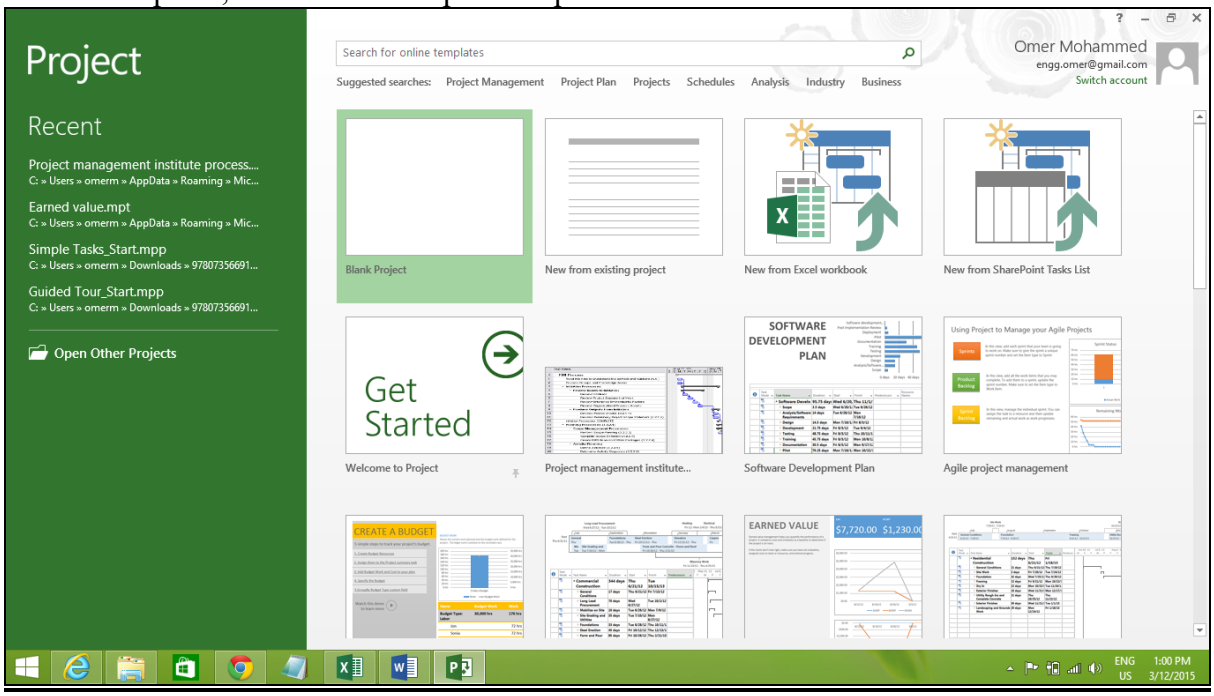
MS Project is feature rich, but project management techniques are required to drive a project effectively. A lot of project managers get confused between a schedule and a plan. MS Project can help you in creating a Schedule for the project even with the provided constraints.

MS Project can help you:

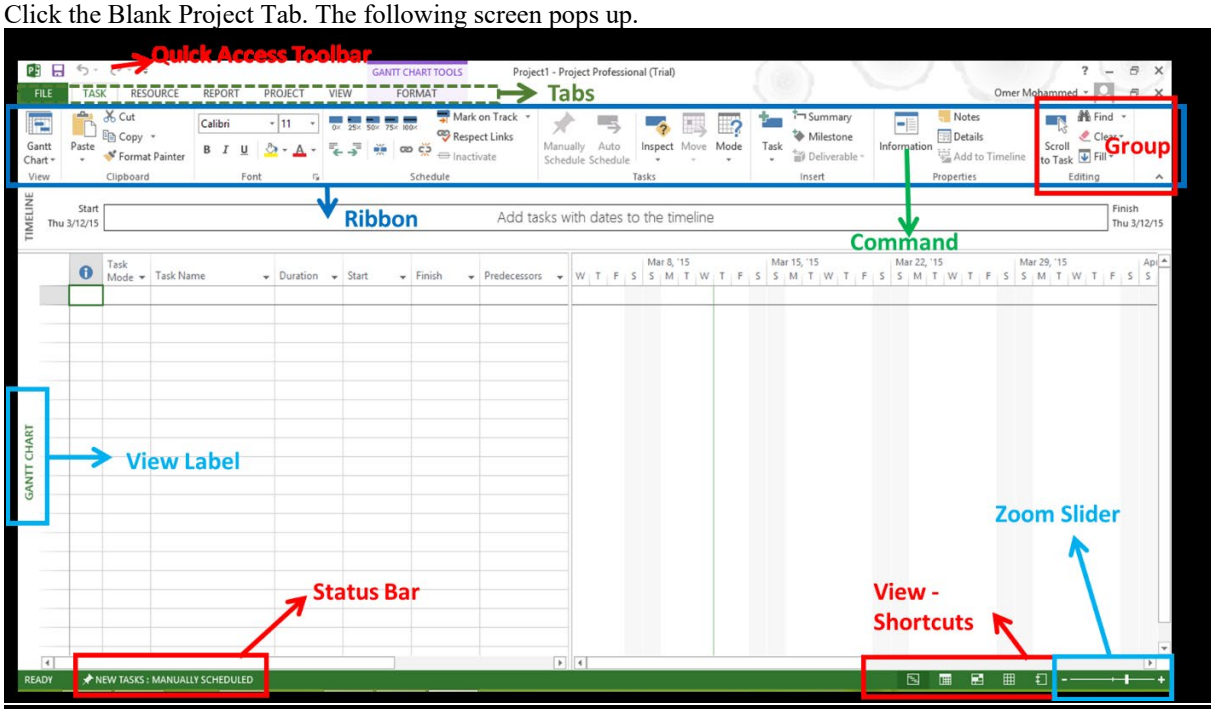
- Visualize your project plan in standard defined formats.
- Schedule tasks and resources consistently and effectively.
- Track information about the work, duration, and resource requirements for your project.
- Generate reports to share in progress meetings.

MS Project UI

The following screen is the Project's start screen. Here you have options to open a new plan, some other plans, and even a new plan template.



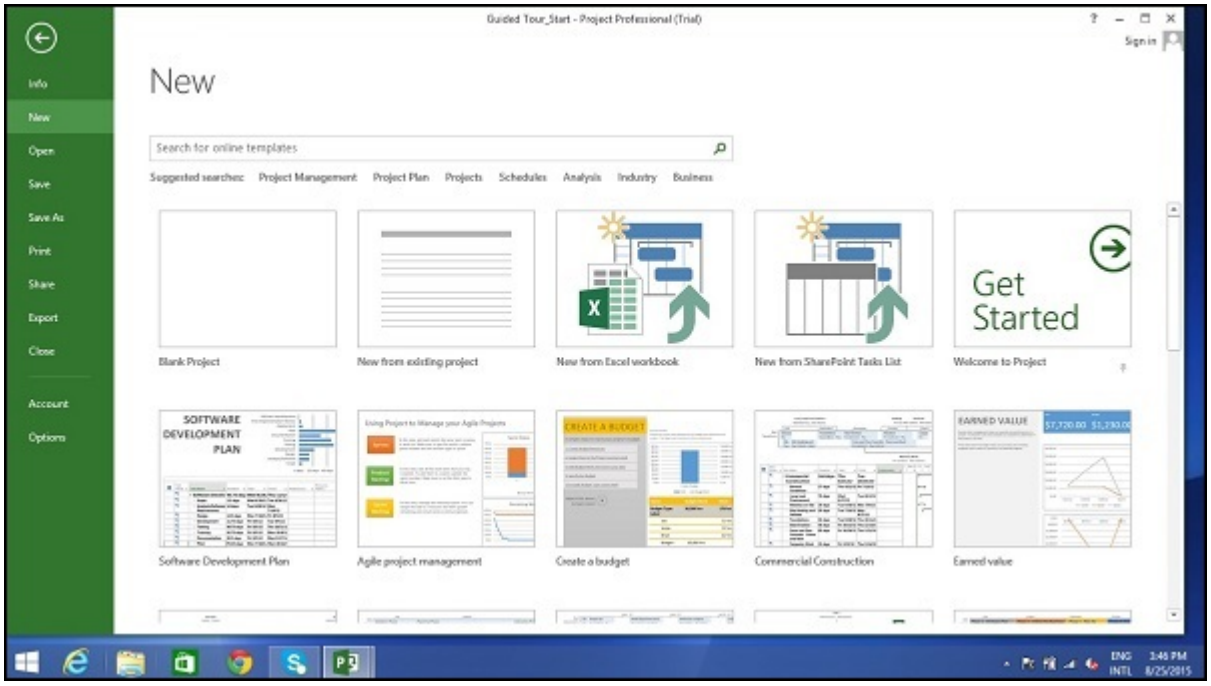
Click the Blank Project Tab. The following screen pops up.



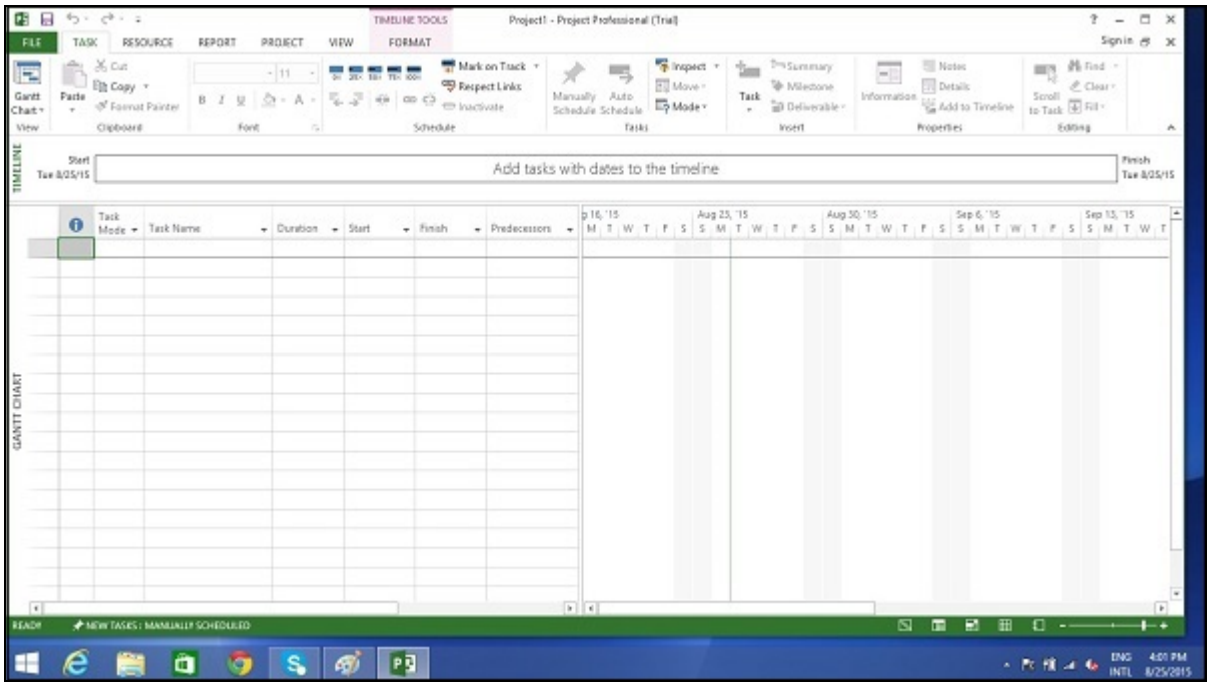
PROCEDURE

Create Blank Project

MS Project 2013 will display a list of options. In the list of available templates, click **Blank Project**.



Project sets the plan's start date to current date, a thin green vertical line in the chart portion of the Gantt Chart View indicates this current date.



Project Information

Let us change the project start date and add some more information.

Start Date

Click Project tab → Properties Group → Project Information.

A dialog box appears. In the start date box, type 11/5/15, or click the down arrow to display the calendar, select November 5, 2015 (or any date of your choice).

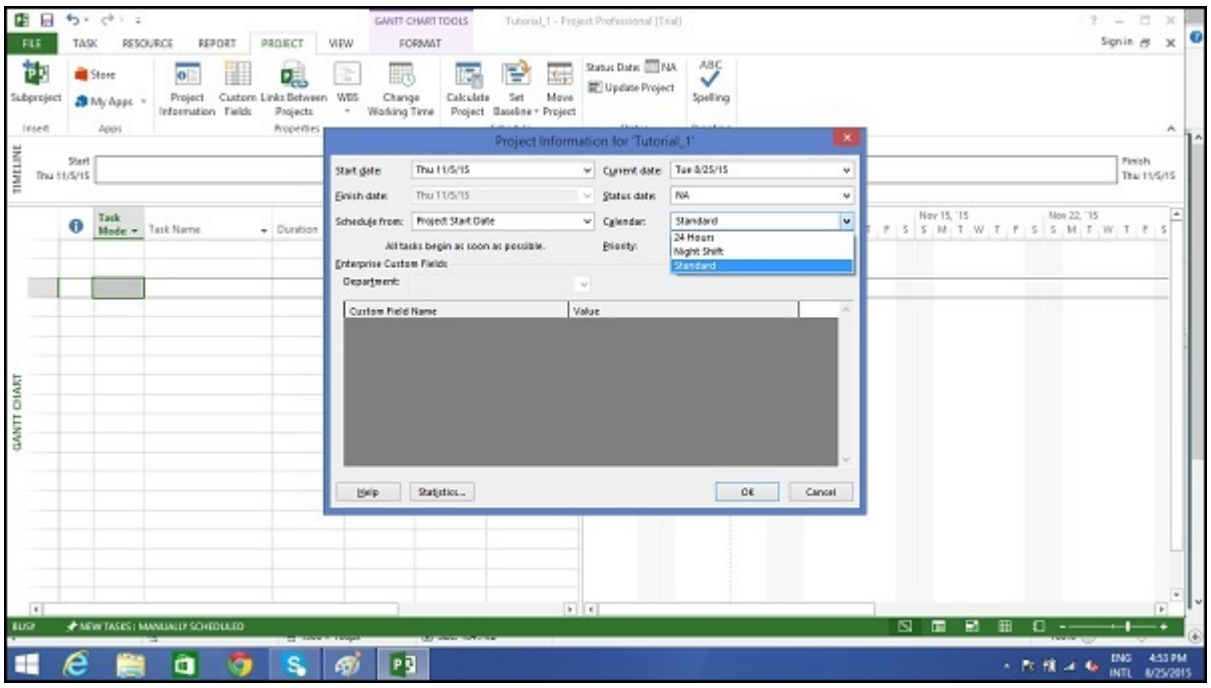
Click OK to accept the start date.

Set Up Calendar

Click Project tab → Properties Group → Project Information.

Click the arrow on the Current Date dropdown box. A list appears containing three base calendars.

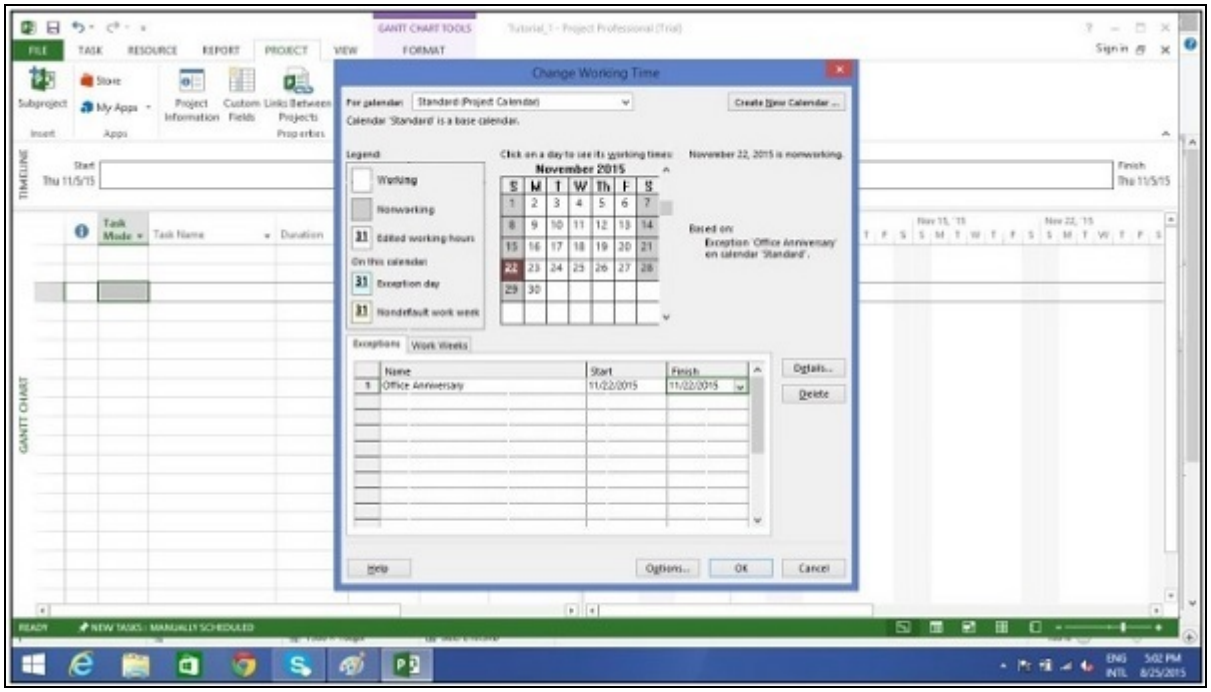
- **24 Hour** − A calendar with no non-working time.
- **Night Shift** − Covers 11 PM to 8 AM, night shifts covering all nights from Monday to Friday, with one hour breaks.
- **Standard** − Regular working hours, Monday to Friday between 8 AM to 5 PM, with one hour breaks.



Select a Standard Calendar as your project Calendar.

Adding Exceptions to Calendar

Exceptions are used to modify a Project calendar to have a non-standard workday or a non-working day. You can also allot unique working hours for a particular resource as well.



Build Task List

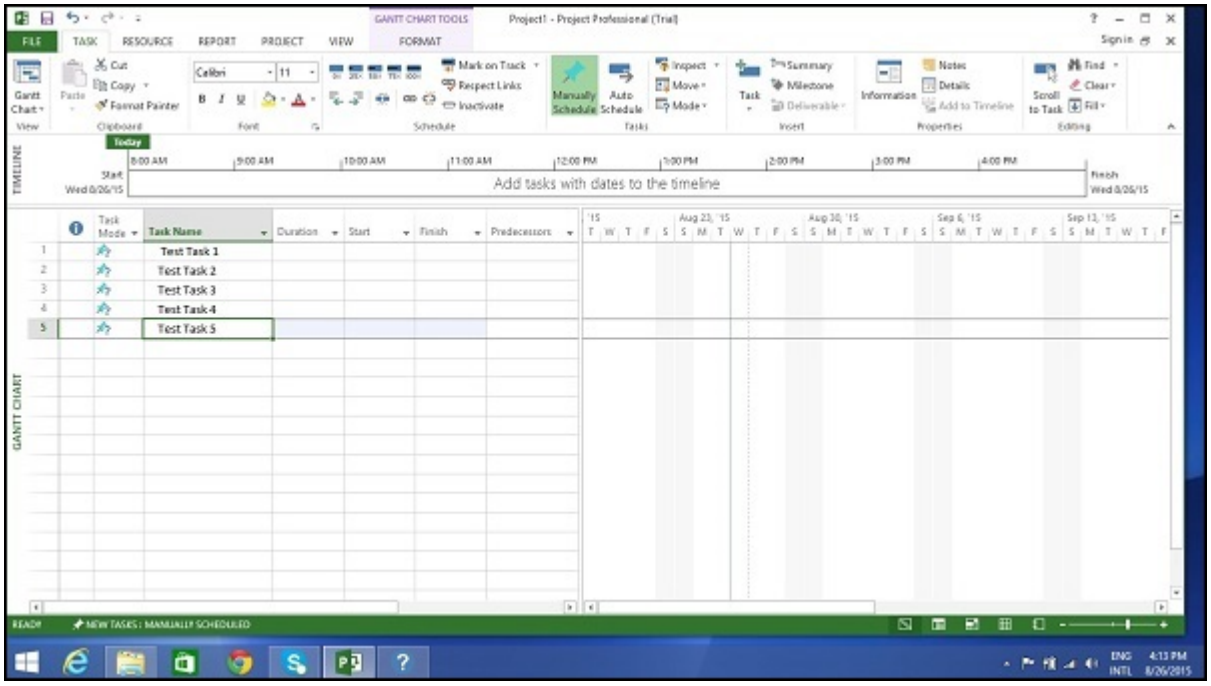
Before we start, let us assume you already have a Work Breakdown Structure (WBS). In context of WBS, “Work” refers to “Deliverables” and not effort.

WBS identifies the deliverable at the lowest level as work package. This work package is decomposed into smaller tasks/activities, which is the effort necessary to complete the work package. So a task is action-oriented, and the work package is the deliverable or a result of one or more tasks being performed.

There is a significant amount of confusion between what constitutes an activity and what constitutes a task within the project management community. But for MS Project, a task is the effort and action required to produce a particular project deliverable. MS Project does not use the term “activity”.

Enter Task

This is simple. In **Gantt Chart** View, just click a cell directly below the Task Name column. Enter the task name. In the following screen, we have entered 5 different tasks.



Enter Duration

A duration of the task is the estimated amount of time it will take to complete a task. As a project manager you can estimate a task duration using expert judgment, historical information, analogous estimates or parametric estimates.

You can enter task duration in terms of different dimensional units of time, namely minutes, hours, days, weeks, and months. You can use abbreviations for simplicity and ease as shown in the following table.

Value you want to enter	Abbreviation	Appearance
45 minutes	45 m	45 mins
2 hours	2h	2 hrs
3 days	3d	3 days
6 weeks	6w	6 weeks
2 months	2mo	2 mons

Remember, Project default values depend on your work hours. So 1 day is not equivalent to 24 hours but has 8 hours of work for the day.

Create Milestones

In Project Management, Milestones are specific points in a project timeline. They are used as major progress points to manage project success and stakeholder expectations. They are primarily used for review, inputs and budgets.

Mathematically, a milestone is a task of zero duration. And they can be put where there is a logical conclusion of a phase of work, or at deadlines imposed by the project plan.

There are two ways you can insert a milestone.

Method 1: Inserting a Milestone

Click name of the Task which you want to insert a Milestone

`Click Task tab → Insert group → Click Milestone.`

MS Project names the new task as <New Milestone> with zero-day duration.

`Click on <New Milestone> to change its name.`

You can see the milestone appear with a rhombus symbol in the Gantt Chart View on the right.

Create Summary Task

There can be a huge number of tasks in a project schedule, it is therefore a good idea to have a bunch of related tasks rolled up into a **Summary Task** to help you organize the plan in a better way. It helps you organize your plan into phases.

In MS Project 2013, you can have several number of sub-tasks under any higher level task. These higher level tasks are called Summary Task. At an even higher level, they are called **Phases**. The highest level of a plan's outline structure is called the **Project Summary Task**, which encompasses the entire project schedule.

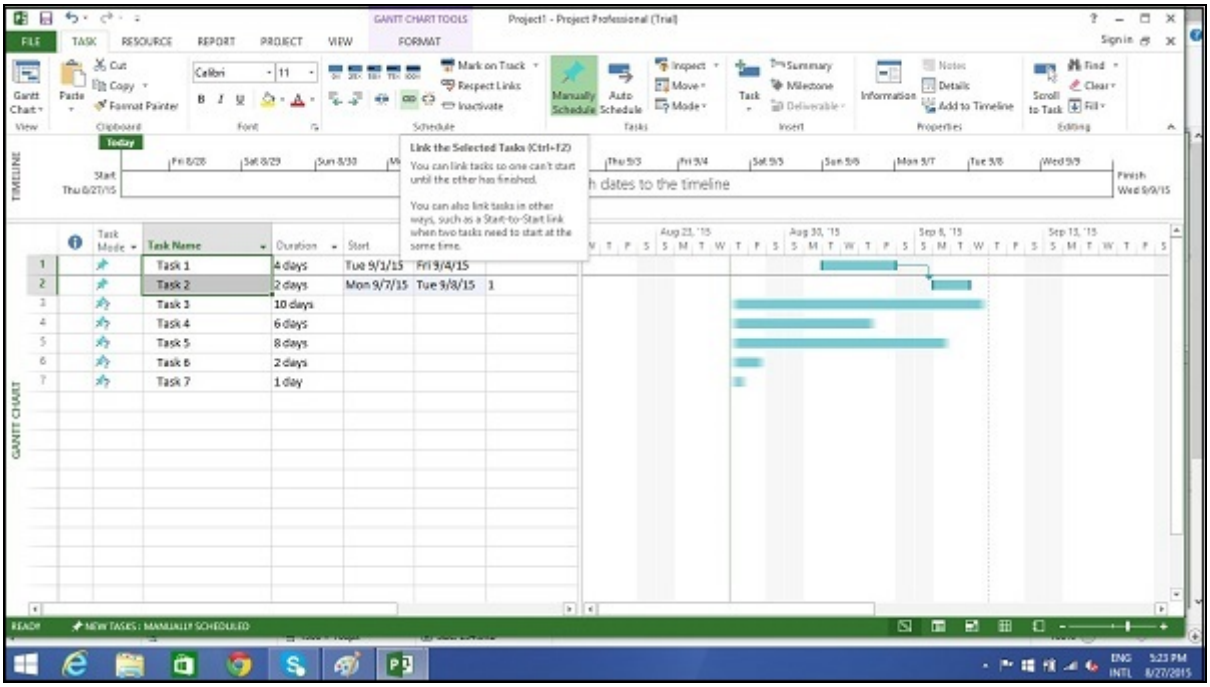
`Click **Task** Tab → group **Insert** → Click **Summary**`

Link Tasks

Once you have a list of tasks ready to accomplish your project objectives, you need to link them with their task relationships called dependencies

In MS Project you can identify the Task Links −

- **Gantt Chart** − In Gantt Chart and **Network Diagram** views, task relationships appear as the links connecting tasks.
- **Tables** − In Tables, task ID numbers of predecessor task appear in the predecessor fields of successor tasks.
- `Click Task tab → Schedule group → Link the Selected Tasks.`



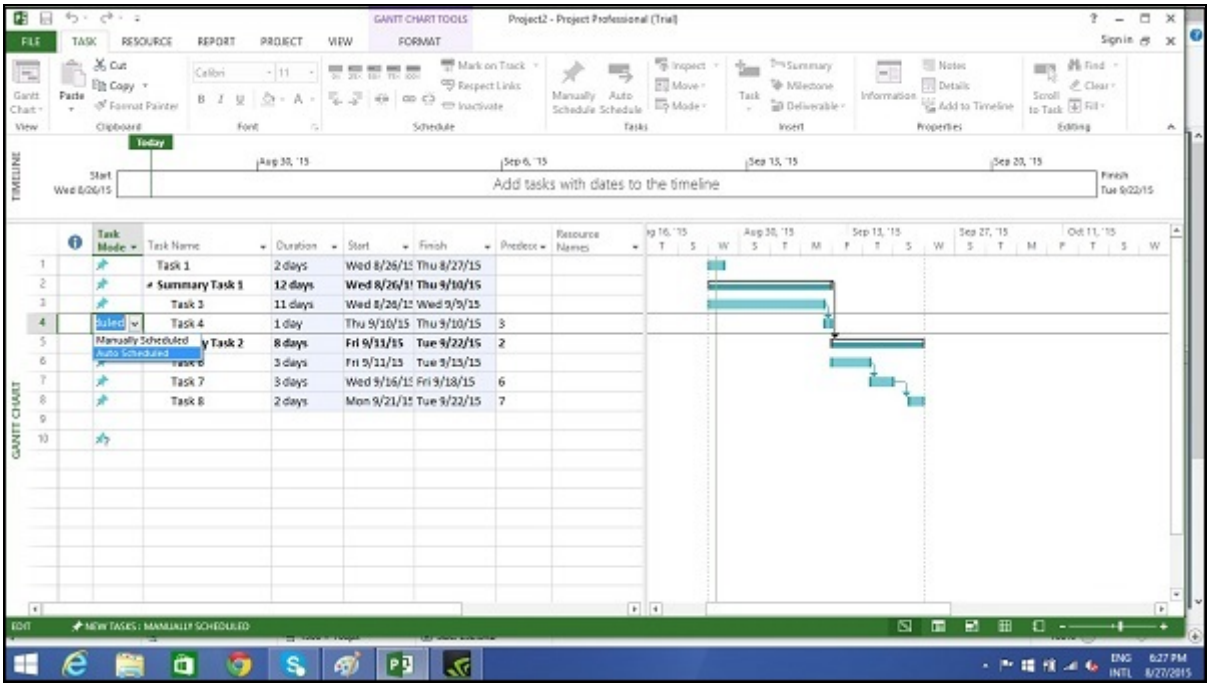
Switching Task – Manual to Automatic

MS Project by default sets new tasks to be manually scheduled. Scheduling is controlled in two ways.

Manual Scheduling − This is done to quickly capture some details without actually scheduling the tasks. You can leave out details for some of the tasks with respect to duration, start and finish dates, if you don't know them yet.

Automatic Scheduling − This uses the Scheduling engine in MS Project. It calculates values such as task durations, start dates, and finish dates automatically. It takes into accounts all constraints, links and calendars.

If you want to change the mode for a particular task, say Task 5 in the following example. Click on **Task Mode** cell in the same row. Then, click the down arrow to open a dropdown box, you can select Auto Scheduled.



Choose *Blank Project.*
You will see a blank window.

First, let's create a **Project Summary Task**. This is like an overall "wrapper" task that contains our entire project.

Give your project a suitable name i.e we write a plan for Chocolate Store Project.

Enter Project start date

To set the project start date, open the Project tab and click *Project Information.*

Enter the project start date as 14th September 2021.

Enter the list of the tasks

- **Create business plan**
- **Get business license**
- **Set up bank account**
- **Get funding**
- **Pick a business location**
- **Set up office equipment and furniture**
- **Hire team**
- **Run promotion**

Enter the tasks into the table next to the *Gantt view*

link all tasks in the right order and enter the estimated durations.

Duration is the total timespan until a task is finished.

Enter the estimated duration in the duration column.

tasks should be performed in a specific order.

getting a business license should follow after *creating business plan*.

To tell Project that *getting business license* comes after *create business plan*, you can simply enter create business plan as a **predecessor** for task *Get business license* (in the *Predecessors* column).

Then Link all the remaining tasks by selecting all of them and pressing link button in the task tab.

tasks are now arranged in the right order, and the Gantt chart is updated.

To mark the point where we've fully set up our business, create a new milestone named '*Business fully set up*'.

Click on

schedule the "milestone task" right after the last regular task (Run promotion).

The milestone we entered also marks the estimated finish date of the project.

OUTPUT

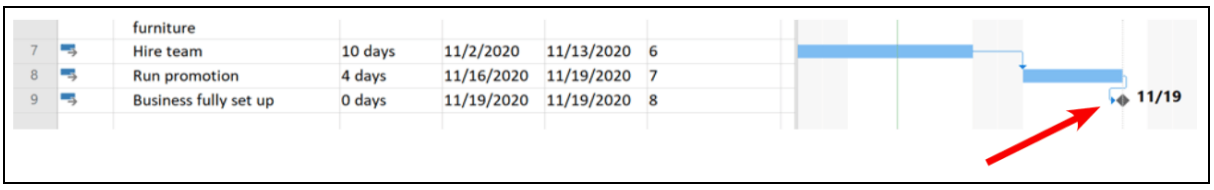
CHOCOLATE STORE PROJECT

TASKS

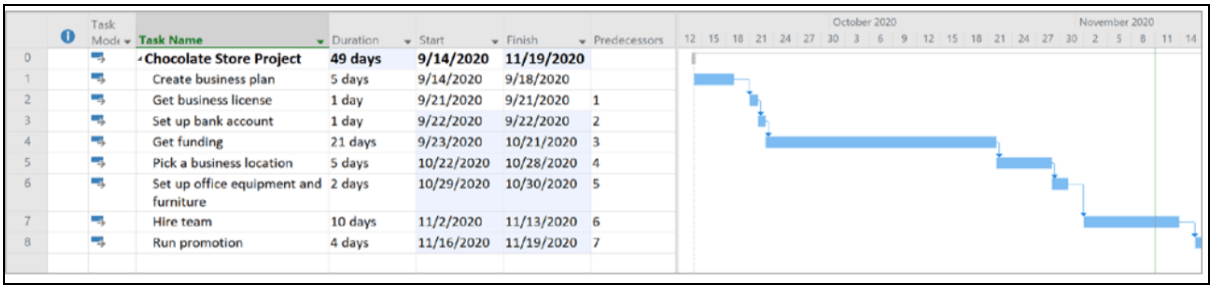
	Task Mode	Task Name	Duration	Start	Finish	Predeces
0		**Chocolate Store Project**	**21 days**	**9/14/2020**	**10/12/2020**	
1		Create business plan	5 days	9/14/2020	9/18/2020	
2		Get business license	1 day	9/14/2020	9/14/2020	
3		Set up bank account	1 day	9/14/2020	9/14/2020	
4		Get funding	21 days	9/14/2020	10/12/2020	
5		Pick a business location	5 days	9/14/2020	9/18/2020	
6		Set up office equipment and furniture	2 days	9/14/2020	9/15/2020	
7		Hire team	10 days	9/14/2020	9/25/2020	
8		Run promotion	4 days	9/14/2020	9/17/2020	

CHART

MILESTONES



TASKS WITH GANTT CHART



CONCLUSION:

Microsoft is a PM tool which is used to

- Stay organized.
- Easily plan and manage your projects using the latest Project templates.
- Deliver projects successfully.
- Improve everyday collaboration.

Quickly deliver project information to your team and easily receive their changes from virtually anywhere with improved task list synchronization

7. Define the Features, Vision, Business objectives, Business rules and stakeholders in the vision document.

AIM:

To define the features, Vision, Business Objectives, Business Rules and Stakeholders in the Vision Document.

INTRODUCTION:

A **Vision Document** is a document that describes a compelling idea, project or other future state for a particular organization, product or service. The Vision defines the product/service to be developed specified in terms of the stakeholder's key needs and desired features. Containing an outline of the envisioned core requirements, it provides the contractual basis for the more detailed technical requirements.

PURPOSE:

The Vision document provides a high-level executive summary for a project or other undertaking. For software development, for example, a Vision captures the essence of the envisioned solution in the form of high-level requirements and design constraints that provide stakeholders an overview of

the system to be developed from a behavioral requirements perspective.
A Vision document provides input to the project approval process and is, therefore, closely related to the Business case. It communicates the fundamental "why and what" for the project and is a gauge against which all future decisions should be validated.

A Vision document generally contains some or all of the following:

- Revision History
- Signature Page
- Introduction
- Purpose Statement
- Scope Statement
- Problem Statement (What is being solved by the Vision?)
- Business Needs/Requirements
- Product/Solution Overview
- Major Features (Optional)
- Stakeholder Register
- Budgetary Details (Rough Order of Magnitude, ROM)
- Assumptions
- Definitions

- Process Details

PROCEDURE:

VISION STATEMENT:

A vision statement describes what a company desires to achieve in long-run, generally in a time frame of five to ten years, or sometimes even longer. It depicts a vision of what the company will look like in the future.

MISSION STATEMENT:

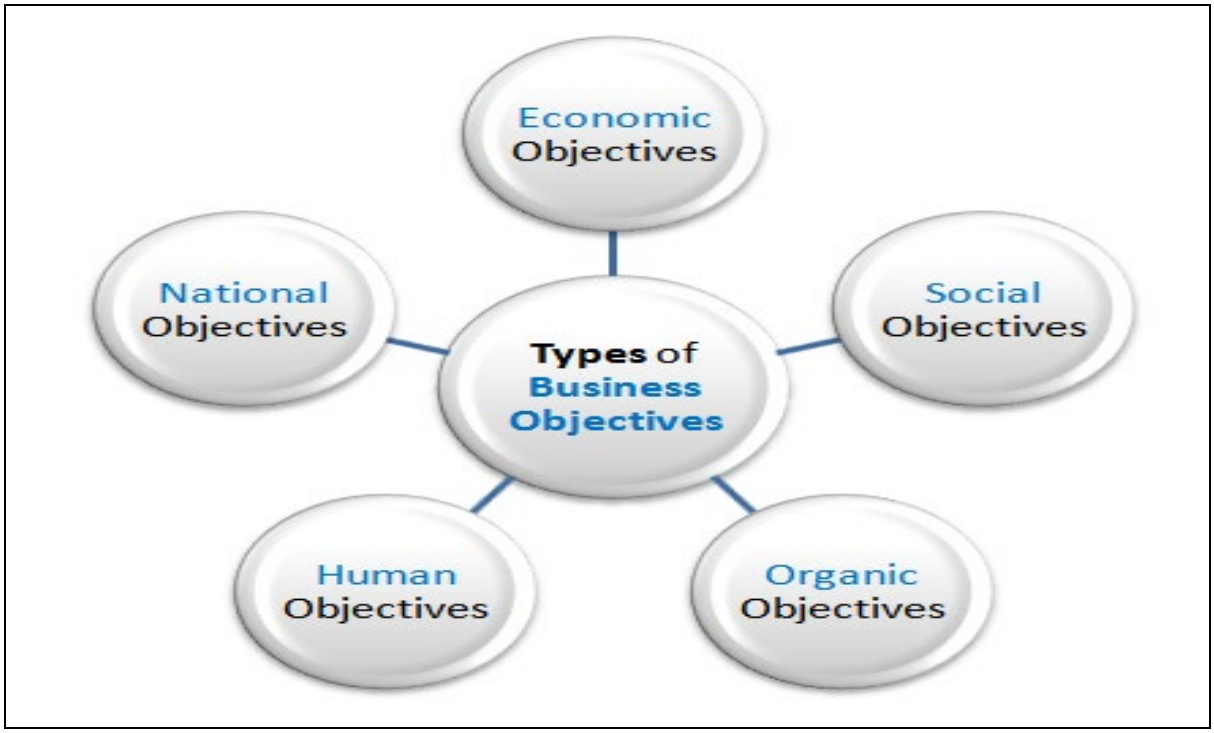
A mission statement is a concise explanation of the organization's reason for existance. It describes the organisations purpose and its overall intention. The mission statement supports the vision and serves to communicate purpose and direction to employees, customers, vendors, and other stakeholders.

Simply, mission is nothing but in what way you are going to approach that particular target.

BUSINESS OBJECTIVES:

A business objective, is what a company wants to achieve throughout the year. Instead of focusing on what you're currently doing in your business, an objective is something you want to achieve going forward.


Development of Human Resources
Profit Earning
Business Objectives
Creation of Customers
Supply of Quality Goods
Innovations



ECONOMIC OBJECTIVES:

1.Profit earning
2.Market share
3. Utilization of resources
4.Increasing Productivity

SOCIAL OBJECTIVES

1.Production and supply of quality goods and services.
2.Adoption of fair trade practices.
3.Contribution to general welfare of the society.

ORGANIC OBJECTIVES:

SURVIVAL

GROWTH
GOOD WILL and IMAGE

HUMAN OBJECTIVES

1.Economic well being of the employees.
2.Development of human resources.
3.Economic well being of socially and economically backward people.

NATIONAL OBJECTIVES

1.Creation of employment.
2.Production according to national priority.
3.Contribution to the revenue of the Country.

BUSINESS RULES:

Business rules define specific instructions or constraints on how certain day-to-day actions should be performed.

SOME BASIC ESSENTIAL RULES FOR RUNNING A BUSINESS

1. Meet the needs of consumers

The primary reason why people spend on products and services is because they need those things in their life. The only way to make consumers buy your product or avail your service is by meeting their needs.

2. Be true to what you sell

Value your customers over everything else. If your customers are placing their trust in your product, you need to deliver nothing but the best to them. When your customers see that they are getting value for their money, they'll spread the word about the goodness of your commodity. Word of mouth has always been the primary source for the expansion and growth of any business. Be true to what you sell and your business will take off like never before.

3. Don't stick to the number 'one'

One client, one product, one service, all your business records on one hard drive – these are a recipe for failure in business. If you've built a loyal customer base over time and yet you sell only one product or service, you miss out on selling more things to people who already know and trust you. If you have only one major client, you'll hit the wall running if they decide to change vendors or if they suddenly develop cash-flow problems. It is therefore advised to expand your business gradually and not play it safe all the time.

STAKEHOLDERS

Stakeholders are those individuals, groups and institutions which have a stake in the functioning and performance of a commercial organization or a business enterprise.

. Affect a business
. Be affected by a business

INTERNAL STAKEHOLDERS

Internal stakeholders are groups or people who work directly within the business, such as employees, managers, owners.

EXTERNAL STAKEHOLDERS

External stakeholders are groups ouside a business for people who don't work inside the business organizations but are effected in some way by the decisions and actions of the business.

ROLE OF STAKEHOLDERS IN BUSINESS:

LOCAL COMMUNITY:

Local community has a stake in the business because it provides jobs which generate economic activity within the community.

CUSTOMERS

. Customers are the people who buy business products.
. Customers expect the business to provide efficient and high quality products and services. In general, meeting the customers needs is an extremely important area of

concern for ensuring the success of any business.
. Customers are directly impacted by the product quality a business gives.

GOVERNMENT

Governments can also be considered a major stakeholder in a business, as they collect taxes from the company,as well as all the people it employs.

EMPLOYEES

. Employees have a direct stake in the company. They interact directly with customers, earn money to support themselves, and give support to the business operations as well.

. They expect benefits like job security, career growth and job satisfaction.

SHAREHOLDERS

. Shareholders are individuals who invest something of value into an organization in exchange for a fractional ownership.

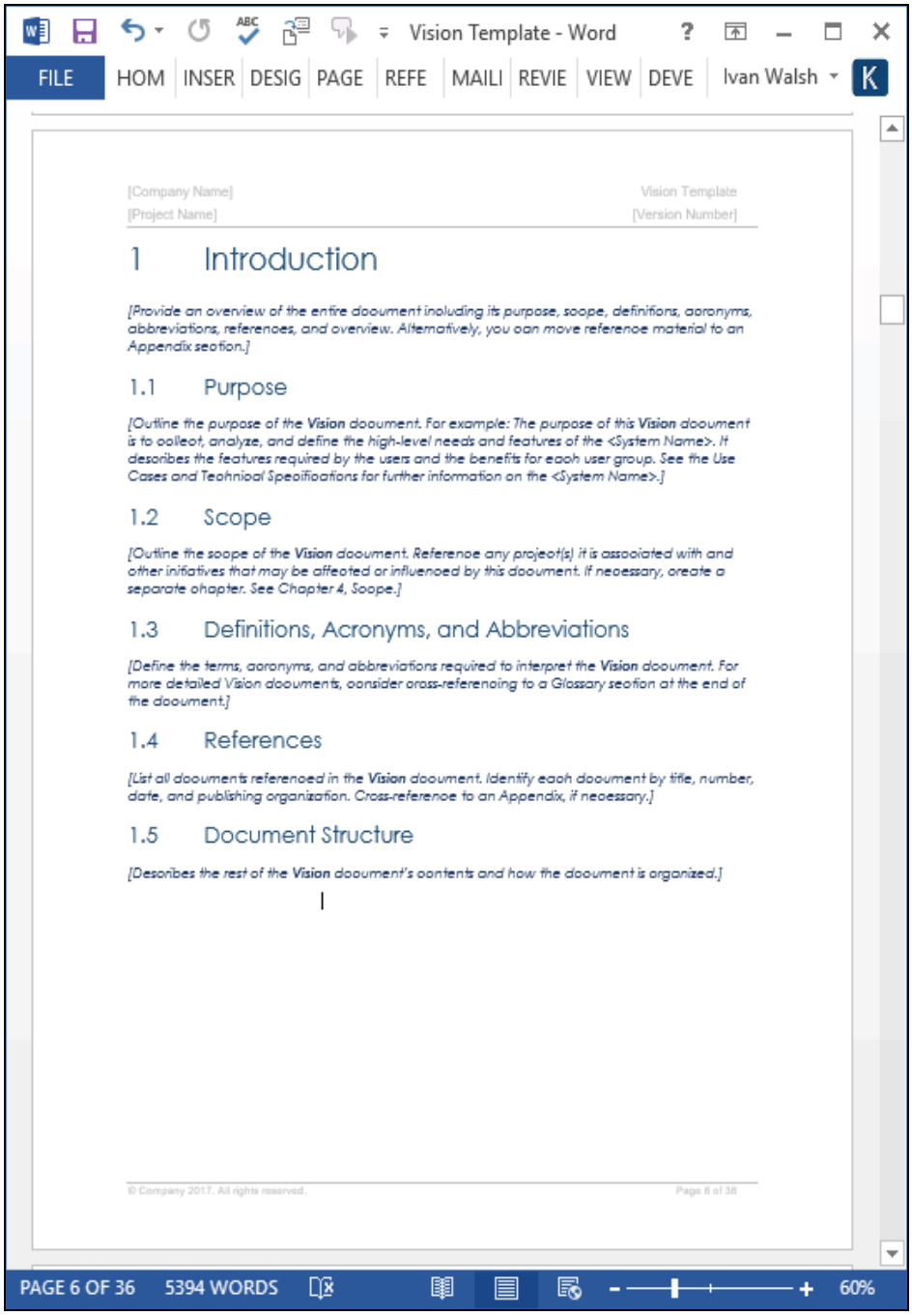
. Their interest is focused on financial gain and their role in the company's operations is often minimal.

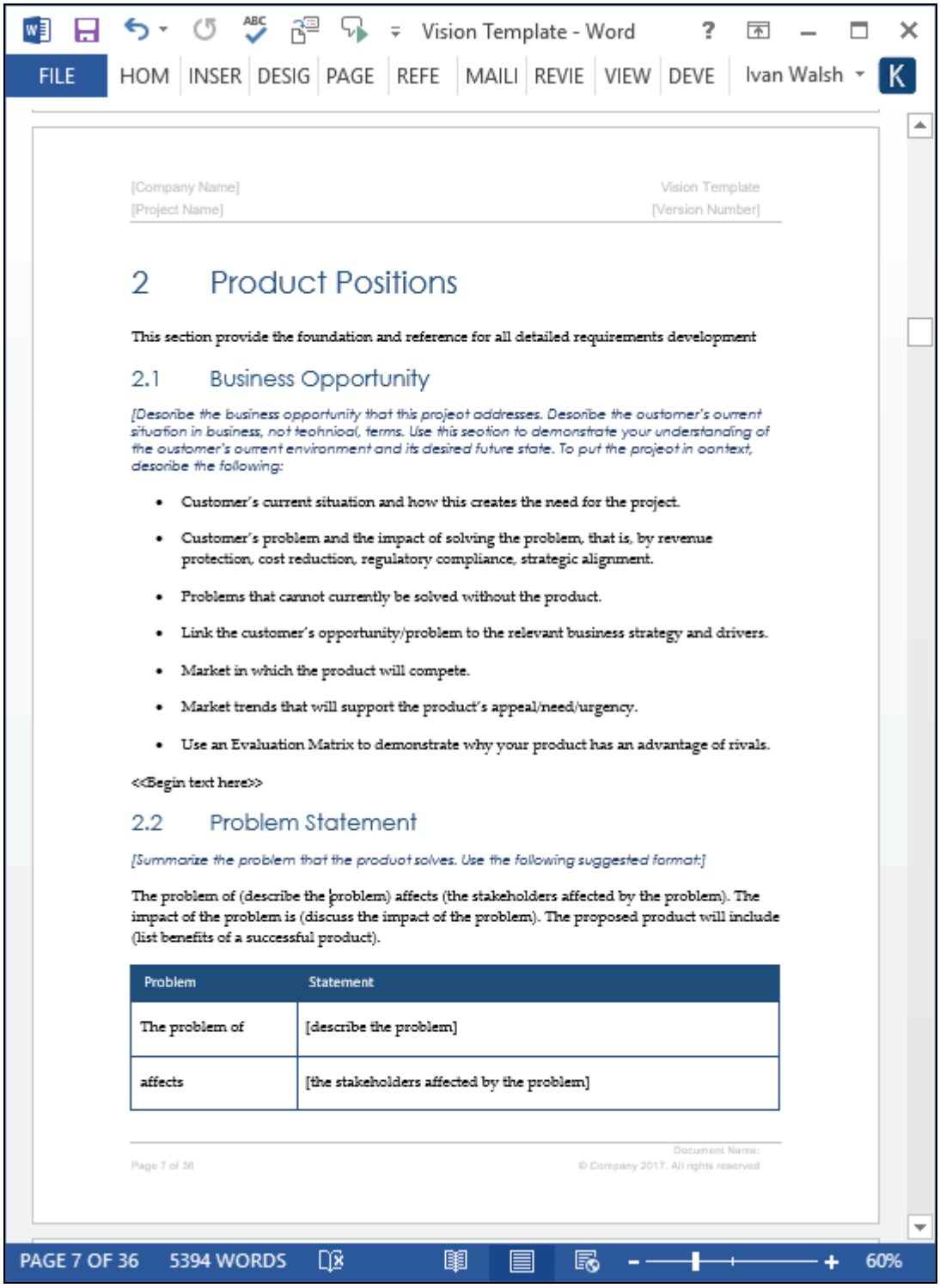
SUPPLIERS

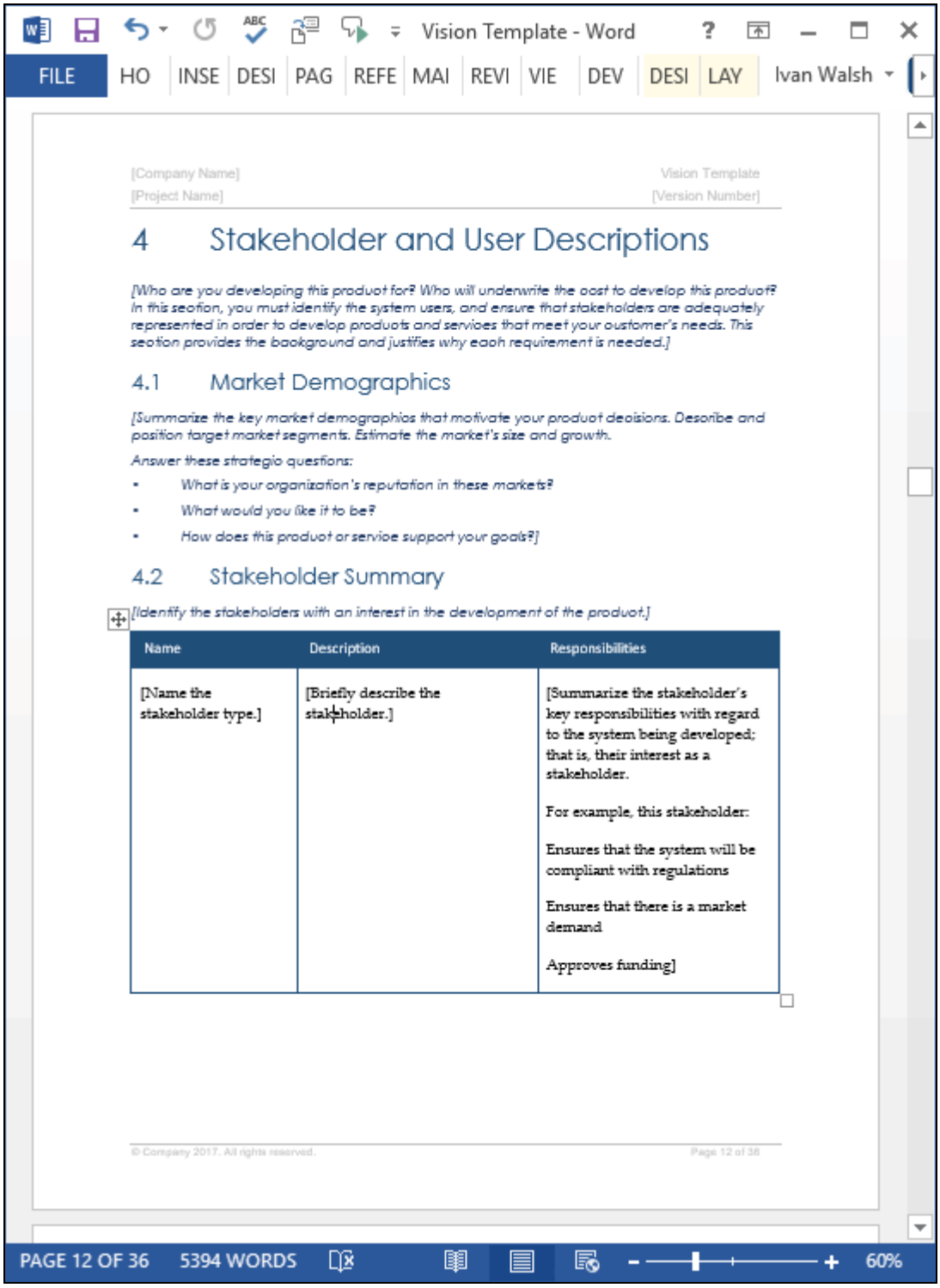
. Suppliers are people or business who sell goods to your business and rely on you for revenue from the sale of those goods.

. Suppliers provide the raw materials that a company uses to create its products.

TEMPLATE OF A VISION DOCUMENT:


Vision Template - Word
FILE
HOM
INSER
DESIG
PAGE
REFE
MAILI
REVIE
VIEW
DEVE
Ivan Walsh
[Company Name]
[Project Name]
Vision Template
[Version Number]
1 Introduction
[Provide an overview of the entire document including its purpose, scope, definitions, acronyms, abbreviations, references, and overview. Alternatively, you can move reference material to an Appendix section.]
1.1 Purpose
[Outline the purpose of the Vision document. For example: The purpose of this Vision document is to collect, analyze, and define the high-level needs and features of the <System Name>. It describes the features required by the users and the benefits for each user group. See the Use Cases and Technical Specifications for further information on the <System Name>.]
1.2 Scope
[Outline the scope of the Vision document. Reference any project(s) it is associated with and other initiatives that may be affected or influenced by this document. If necessary, create a separate chapter. See Chapter 4, Scope.]
1.3 Definitions, Acronyms, and Abbreviations
[Define the terms, acronyms, and abbreviations required to interpret the Vision document. For more detailed Vision documents, consider cross-referencing to a Glossary section at the end of the document.]
1.4 References
[List all documents referenced in the Vision document. Identify each document by title, number, date, and publishing organization. Cross-reference to an Appendix, if necessary.]
1.5 Document Structure
[Describes the rest of the Vision document's contents and how the document is organized.]
© Company 2017. All rights reserved.
Page 6 of 36
PAGE 6 OF 36
5394 WORDS
60%


Vision Template - Word
FILE
HOM
INSER
DESIG
PAGE
REFE
MAILI
REVIE
VIEW
DEVE
Ivan Walsh
[Company Name]
[Project Name]
Vision Template
[Version Number]
2 Product Positions
This section provide the foundation and reference for all detailed requirements development
2.1 Business Opportunity
[Describe the business opportunity that this project addresses. Describe the customer's current situation in business, not technical, terms. Use this section to demonstrate your understanding of the customer's current environment and its desired future state. To put the project in context, describe the following:
Customer's current situation and how this creates the need for the project.
Customer's problem and the impact of solving the problem, that is, by revenue protection, cost reduction, regulatory compliance, strategic alignment.
Problems that cannot currently be solved without the product.
Link the customer's opportunity/problem to the relevant business strategy and drivers.
Market in which the product will compete.
Market trends that will support the product's appeal/need/urgency.
Use an Evaluation Matrix to demonstrate why your product has an advantage of rivals.
<<Begin text here>>
2.2 Problem Statement
[Summarize the problem that the product solves. Use the following suggested format:]
The problem of (describe the problem) affects (the stakeholders affected by the problem). The impact of the problem is (discuss the impact of the problem). The proposed product will include (list benefits of a successful product).
Problem
Statement
The problem of
[describe the problem]
affects
[the stakeholders affected by the problem]
Document Name:
Page 7 of 36
© Company 2017. All rights reserved
PAGE 7 OF 36
5394 WORDS
60%


Vision Template - Word
FILE
HO
INSE
DESI
PAG
REFE
MAI
REVI
VIE
DEV
DESI
LAY
Ivan Walsh
[Company Name]
[Project Name]
Vision Template
[Version Number]
4 Stakeholder and User Descriptions
[Who are you developing this product for? Who will underwrite the cost to develop this product? In this section, you must identify the system users, and ensure that stakeholders are adequately represented in order to develop products and services that meet your customer's needs. This section provides the background and justifies why each requirement is needed.]
4.1 Market Demographics
[Summarize the key market demographics that motivate your product decisions. Describe and position target market segments. Estimate the market's size and growth.
Answer these strategic questions:
What is your organization's reputation in these markets?
What would you like it to be?
How does this product or service support your goals?]
4.2 Stakeholder Summary
[Identify the stakeholders with an interest in the development of the product.]
Name
Description
Responsibilities
[Name the stakeholder type.]
[Briefly describe the stakeholder.]
[Summarize the stakeholder's key responsibilities with regard to the system being developed; that is, their interest as a stakeholder.
For example, this stakeholder:
Ensures that the system will be compliant with regulations
Ensures that there is a market demand
Approves funding]
© Company 2017. All rights reserved.
Page 12 of 38
PAGE 12 OF 36
5394 WORDS
60%

8 .Define the functional and non-functional requirements of the system to be automated by using Use cases and document in SRS document.

AIM: To define functional and nonfunctional requirements of the system to be automated by using the USECASES and SRS document.

PRODUCT REQUIREMENTS:

Product requirements is nothing but it is a documented version of project needs, deadlines, risks etc.

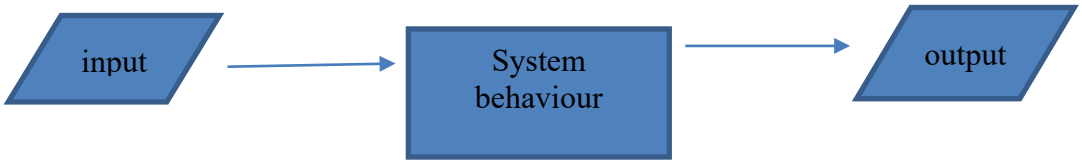
Product requirements mainly describes multiple aspects like purpose of software we are building, individual features, functionalities etc.

Types of product requirements:

1. Business requirements
2. User requirements
3. Functional requirements
4. Nonfunctional requirements
5. Implementation requirements.

Functional requirements:

- Functional requirements mainly define the basic system behaviour. Essentially, they are what the system does or must not do, and can be thought of in terms of how the system responds to inputs. Functional requirements usually define if/then behaviours and include calculations, data input, and business processes.



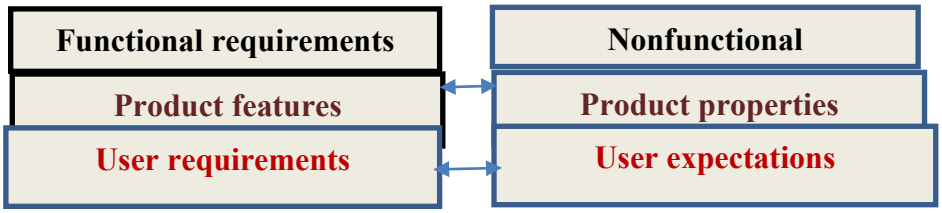
- Functional requirements are features that allow the system to function as it was intended by the developer. In case if the functional requirements are not met, the system will not work.
- Functional requirements are nothing but product features and mainly focus on user requirements.

Nonfunctional requirements:

- While functional requirements define what the system does or must not do, whereas non-functional requirements specify how the system should do it. Non-functional requirements do not affect the basic functionality of how the system hence named non-functional requirements. Even if the non-functional requirements are not met, the system will still work and perform its basic purpose.

- If a system will still perform without meeting the non-functional requirements, why are they important?

- The answer is usability. Non-functional requirements define system behaviour, features, and general characteristics that affect the user experience. How well non-functional requirements are defined and executed determines how easy the system is to use, and is used to judge system performance.
- Non-functional requirements are product properties and focus on user expectations.



EXAMPLE: password (functional)- 8 characters (nonfunctional)

SOFTWARE REQUIREMENT SPECIFICATION (SRS):

- ✓ A software requirements specification (SRS) is a document that describes what the software we are building will do and how it will be expected to perform.
- ✓ It also describes the functionalities that the product/software must fulfill all the stakeholders (business, users) needs.
- ✓ This document lists sufficient and necessary requirements for the project development.
- ✓ It is a developed based agreement between customer, contractors or suppliers, how the software product should function. It may include the use cases of how user is going to interact with software system.
- ✓ The software requirement specification document consistent of all necessary requirements required for project development.
- ✓ A good SRS defines the how Software System will interact with all internal modules, hardware, communication with other programs and human user interactions with wide range of real-life scenarios.
- ✓ Using the Software requirements specification (SRS) document on QA lead, managers creates test plan. It is very important that testers must be cleared with every detail specified in this document in order to avoid faults in test cases and its expected results.
- ✓ It is modeled after business requirements specification (CONOPS), also known as a stakeholder requirements specification (StRS).
- ✓ The software requirements specification lays out functional and non-functional requirements, and it may include a set of use cases that describe user interactions that the software must provide to the user for perfect interaction.
- ✓ To derive the requirements, the developer needs to have clear and thorough understanding of the products under development. This is achieved through detailed and continuous communications with the project team and customer throughout the software development process.
- ✓ Basically, the SRS document is written by a technical writer, a systems architect, or a software programmer.

The software requirements specification (SRS) is a communication tool between users and software designers.

It mainly describes:

1. Introduction
   - 1.1.Purpose
   - 1.2.Scope
   - 1.3.Definitions(overview)
   - 1.4.Additional information
2. General Description
3. Functional Requirement
   - 3.1. Description
   - 3.2 Technical Issues
4. Interface Requirement
   - 4.1 GUI
   - 4.2 Hardware Interface
   - 4.3 Software Interface
5. Performance Requirement
6. Design Constraints
7. Other non-functional requirement
   - 7.1 Security
   - 7.2 Reliability
   - 7.3 Availability
   - 7.4 Maintainability7.5 Portability
8. Operational Scenario
9. Preliminary Schedule

OUTPUT:

SRS DOCUMENT AND USECASE DIAGRAM FOR ONLINE SHOPPING SYSTEM:

1. Introduction

1.1 Purpose: This document is meant to delineate the features of OSS, so as to serve as a guide to the developers on one hand and a software validation document for the prospective client on the other. The Online Shopping System (OSS)for furniture shop web application is intended to provide complete solutions for vendors as well as customers through a single get way using the internet. It will enable vendors to setup online shops, customer to browse through the shop and purchase them online without having to visit the shop physically. The administration module will enable a system administrator to approve and reject requests for new shops and maintain various lists of shop category.

1.2 Scope:

Dismiss user rating prompt

Improve Your Experience

Rating will help us to suggest even better related documents to all of our readers!
79% found this document useful, Mark this document as useful
21% found this document not useful, Mark this document as not useful
This system allows the customers to maintain their cart for add or remove the product over the internet.

1.3 Definitions

OSS- Online shopping System (for furniture shop)
SRS- Software Requirement Specification
GUI- Graphical User Interface Stockholder- The person who will participate in system Ex. Customer, Administrator, Visitor etc.

1.3.1 Overview

This system provides an easy to solution customers to buy the product without go to the shop and also shop owner to sale the product.

1.4 Additional information:

The system work on internet server, so it will be operated by any end user for the buying purpose.

2. General Description

The Online Shopping system (OSS) application enables vendors to set up online shops, customers to browse through the shops, and a system administrator to approve and reject requests for new shops and maintain lists of shop categories. Also, the developer is designing an online shopping site to manage the items in the shop and also help customers purchase them online without having to visit the shop physically. The online shopping system will use the internet as the sole method for selling goods to its consumers.

3. Functional Requirement

This section provides requirement overview of the system.
Various functional modules that can be implemented by the system will be -

3.1 Description3.1.1 Registration

If customer wants to buy the product then he/she must be registered, unregistered user can't go to the shopping cart.

3.1.2 Login

Customer logins to the system by entering valid user id and password for the shopping

3.1.3 cart:

Changes to Cart Changes to cart means the customer after login or registration can make order or cancel order of the product from the shopping cart.

3.1.4 Payment

For customer there are many type of secure billing will be prepaid as debit or credit card, post-paid as after shipping, check or bank draft. The security will provide by the third party like Pay-Pal etc.

3.1.5 Logout

After the payment or surf the product the customer will logged out.

3.1.6 Report Generation:

After all transaction the system can generate the portable document file (.pdf) and then sent one copy to the customer's Email
-address and another one for the system data base to calculate the monthly transaction .

3.2 Technical Issues:

This system will work on client-server architecture. It will require an internet server and which will be able to run PHP application. The system should support some commonly used browser such as IE etc.

4. Interface Requirement

Various interfaces for the product could be-

1. Login Page
2. Registration Form
3. There will be a screen displaying information about product that the shophaving.
4. If the customers select the buy button then another screen of shopping cart will be opened.
5. After all transaction the system makes the selling report as portable documentfile (.pdf) and sent to the customer E-mail address.

4.1 GUI

1. Login Page
2. Registration Form
3. Product Page
4. Shopping Cart
5. Portable Document file (.pdf) buying report

4.2 Hardware Interface:

The System must run over the internet, all the hardware shall require to connect internet will be hardware interface for the system. As for e.g. Modem, WAN
–
LAN, Ethernet Cross-Cable.

4.3 Software Interface

The system is on server so it requires the any scripting language like PHP, VBScript etc. The system requires Data Base also for the store the any transaction of the system like MYSQL etc. system also require DNS (domain name space) for the naming on the internet. At the last user need web browser for interact with the system.

5. Performance Requirement

There is no performance requirement in this system because the server request and response is depended on the end user internet connection.

6. Design Constrain

The system shall be built using a standard web page development tool that conforms to Microsoft's GUI standards like HTML, XML etc.

7. Other non-functional requirement

7.1 Security

- The system use SSL (secured socket layer) in all transactions that include any confidential customer information.
- The system must automatically log out all customers after a period of inactivity.
- The system should not leave any cookies on the customer's computer containing the user's password.
- The system's back
- -end servers shall only be accessible to authenticated administrators.
- Sensitive data will be encrypted before being sent over insecure connections like the internet.

7.2 Reliability:

- The system provides storage of all databases on redundant computers with automatic switchover.
- The reliability of the overall program depends on the reliability of the separate components. The main pillar of reliability of the system is the backup of the database which is continuously maintained and updated to reflect the most recent changes.
- Thus ,the overall stability of the system depends on the stability of container and its underlying operating system.

7.3 Availability

The system should be available at all times, meaning the user can access it using a web browser, only restricted by the down time of the server on which the system runs. In case of a of a hardware failure or database corruption, a replacement page will be shown. Also, in case of a hardware failure or database corruption, backups of the database should be retrieved from the server and saved by the administrator. Then the service will be restarted. It means 24 X 7 availability.

7.4 Maintainability

A commercial database is used for maintaining the database and the application server takes care of the site. In case of a failure, a re-initialization of the program will be done.
Also, the software design is being done with modularity in mind so that maintainability can be done efficiently.

7.5 Portability:

- The application is HTML and scripting language based. So The end-user part is fully portable and any system using any web browser should be able to use the features of the system, including any hardware platform that is available or will be available in the future.
- An end-user is use this system on any OS; either it is Windows or Linux.

- The system shall run on PC, Laptops, and PDA etc.

8. Operational Scenario

- The customer wants to buy item. The system shows all product categories to customer. If customer select item then they listed in shopping cart for buying.

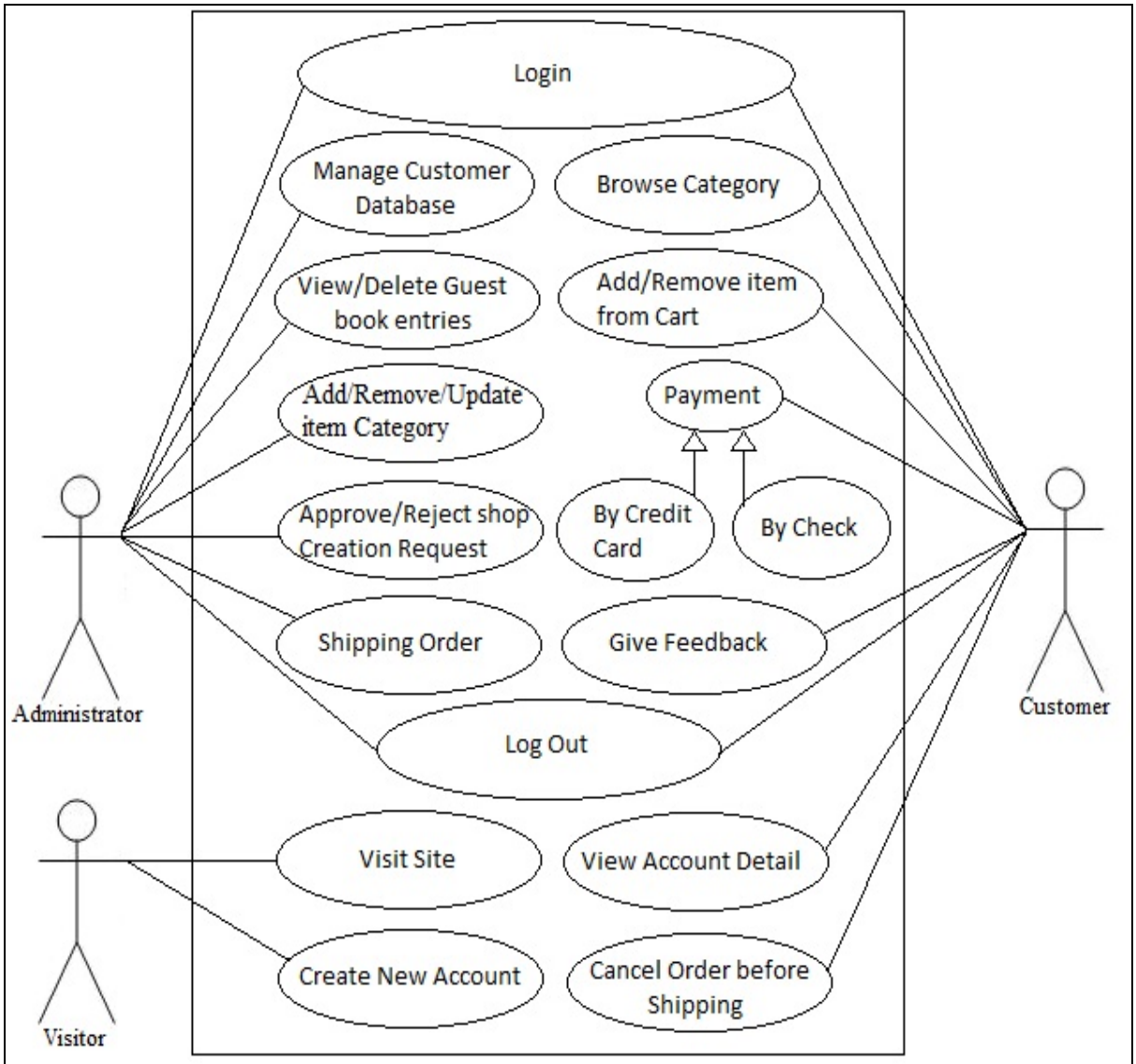
- The payment will made with credit card or bank check. If customer wants to cancel the order before shipping then he or she can cancel it.

- Customer can see the buying report on account detail.

9. Preliminary Schedule:

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

USE CASES: (Refer 3rd program)



Use case diagram for Customer, Visitor and Administrator.

QUESTION NO:9

Defining TRACEABILITY MATRIX for the following

A. To define traceability matrix for Usecases vs features

B. To define traceability matrix Functional requirements vs Usecases

TRACEABILITY MATRIX

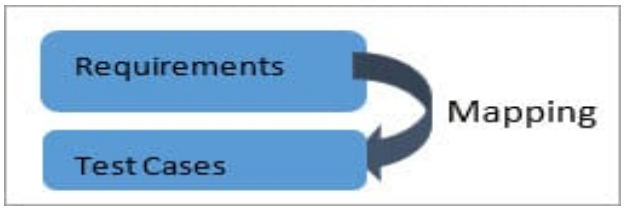
It is a document that traces and maps the relationship between the requirements and the test cases. This includes one with requirement specifications and other with the test cases. Traceability matrix is an essential tool during the development of the new product. The traceability matrix helps to ensure transparency and completeness of the software testing products.

It usually includes columns consists of requirements and rows consists of test cases.

The product is prepared according to the requirements of the stakeholder

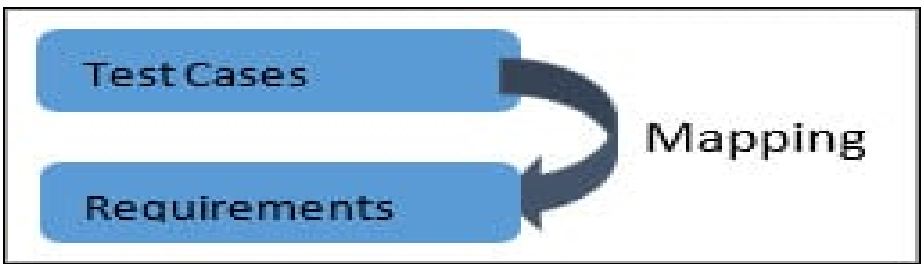
Forward Traceability

In ‘Forward Traceability’ Requirements to the Test cases. It ensures that the project progresses as per the desired direction and that every requirement is tested thoroughly.



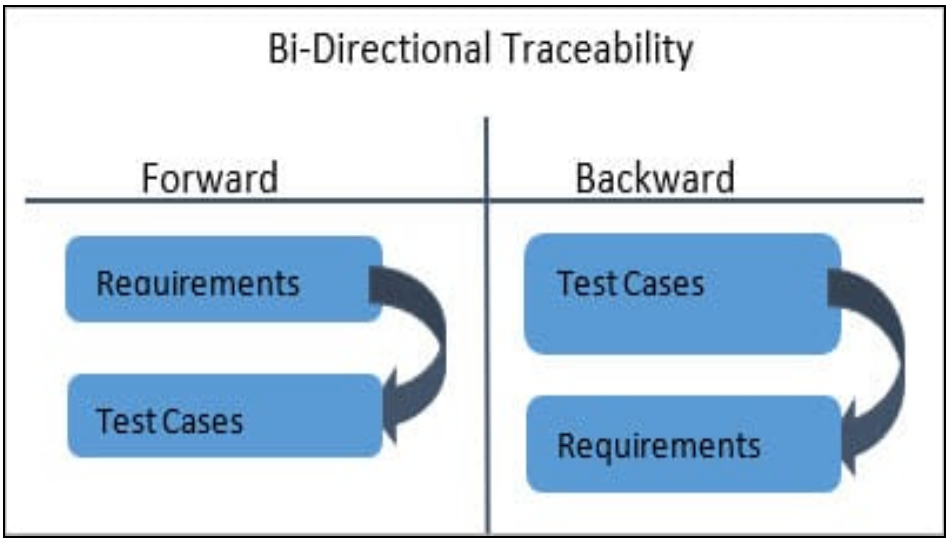
Backward Traceability

The Test Cases are mapped with the Requirements in ‘Backward Traceability’. Its main purpose is to ensure that the current product being developed is on the right track. It also helps to determine that no extra unspecified functionalities are added and thus the scope of the project is affected.



Bi-Directional Traceability

(Forward + Backward): A Good Traceability matrix has references from test cases to requirements and vice versa (requirements to test cases). This is referred to as 'Bi-Directional' Traceability. It ensures that all the Test cases can be traced to requirements and each and every requirement specified has accurate and valid Test cases for them.



CREATING TRACEABILITY MATRIX

In the **process of creating a Traceability Matrix**, you need to take the following steps:

1. Set goals
2. Collect artifacts
3. Prepare a traceability matrix template
4. Adding the artifacts
5. Update the traceability matrix

1.SET GOALS

The step is to define your goals before you actually create the traceability matrix
Here is an example of a goal
I want to create a traceability matrix to keep the track of test cases and functional requirements

2. COLLECT ARTIFACTS

As you have defined your goal, now you need to know which artifacts you will need in order to accomplish your goal. For creating a Requirements Traceability Matrix, you will need:

- Requirements
- Test cases

- Test results
- Bugs

The next step will be to collect these artifacts. For this, you will have to access the latest version of requirements and make sure each requirement has a unique identification ID. You can then gather all the test cases from the testing team. If the testing is going on or it has been completed, you will have access to the test results as well as the bugs found.

3- Prepare a traceability matrix template

For a requirements traceability matrix template, you can create a spreadsheet in excel and add a column for each artifact that you have collected. The columns in the excel will be as follows:

Requirements	Test cases	Test results	Bugs

4- Adding the artifacts

You can start adding the artifacts you have to the columns. You can now copy and paste requirements, test cases, test results & bugs in the respective columns. You need to ensure that the requirements, test cases, and bugs have unique ids.

5- Update the traceability matrix

Updating the traceability matrix is an ongoing job which continues until the project completes. If there is any change in the requirements, you need to update the traceability matrix. There might be a case that a requirement is dropped; you need to update this in the matrix. If a new test case is added or a new bug is found, you need to update this in the requirements traceability matrix.

9.a) Aim:- to define traceability matrix for usacases versus features

USE CASES VS FEATURES

Now, To draw a Traceability Matrix for features of an USECASE we have to list the features of use case

* We have to list the Test Case ID's and Test Scenarios
* Then we will test the features of an use case whether all the requirements have been met.
* Then we will draw a table to check the working of features.
* If there is an error or exception, we will leave the field empty.
* If it's working perfectly then we will map the test case and features.

EXAPMLE

Online Book Selling													
		Features											
ID	Test Scenarios	Create Account	Login	Search Books	Create Order	Orders List	Add to Cart	Shipping Address	Payment	Confirm Order	View History	Track Order	Close Account
TC01	Add Users	X	X										
TC02	Secure Acess		X										
TC03	Store User Data		X										
TC04	Validate User		X										
TC05	Search Books			X									
TC06	Add to Cart						X						
TC07	Order Books				X								
TC08	Recieve Books												X
TC09	Ordered Products					X							
TC10	Payment								X				
TC11	Remove Account												X

9.b) Aim:- To define traceability matrix for functional requirements versus usecases

FUNCTIONAL REQUIREMENTS VS USE CASES

A functional requirement defines a function of a system or its component, where a function is described as **a specification of behavior between inputs and outputs**.

EXAMPLE

Booking of movie tickets

Once the tickets has been compiled, the next step is to create a requirements traceability matrix. This matrix simply takes the list of requirements, grouped by category, and puts them into a matrix. One effective technique is to store information on a system that can be easily be shared and modified by each team member.

Entry no.	Specification requirements	Build	Use case
1	In order to purchase tickets customer should provide the system with the info – movie, theatre, time and number of tickets	1.0	UC001
2	Customer shall either be able to pay by credit card or at the ticket counter	1.0	UC001
3	If the customer is paying by the credit card, they shall provide their name, card number, CVV and expiry date	1.0	UC001
4	Customers should collect their ticket at the counter	1.0	UC001
5	Customer need to give the confirmation number after they have made their purchase	1.0	UC001
6	Customers must arrive 15 mins before the movie starts to collect their tickets	1.0	UC001

10.Estimate the effort using the following methods for the system to be automated
1.use case point metric
2.Function point metric

Aim: To estimate the effort using Use-Case Point

Procedure:

Use-Case Points – Definition

A **Use-Case** is a series of related interactions between a user and a system that enables the user to achieve a goal.

Use-Case Points UCP is a software estimation technique used to measure the software size with use cases. The concept of UCP is similar to FPs.

The number of UCPs in a project is based on the following −

- The number and complexity of the use cases in the system.
- The number and complexity of the actors on the system.
    - Various non-functional requirements such as portability, performance, maintainability that are not written as use cases.
    - The environment in which the project will be developed such as the language, the team's motivation etc...

Use-Case Points Counting Process

The Use−Case Points counting process has the following steps −

- Calculate unadjusted UCPs
- Adjust for technical complexity
- Adjust for environmental complexity
- Calculate adjusted UCPs

Step 1: Calculate Unadjusted Use-Case Points.

You calculate Unadjusted Use−Case Points first, by the following steps −

- Determine Unadjusted Use-Case Weight
- Determine Unadjusted Actor Weight
- Calculate Unadjusted Use-Case Points

Step 1.1 − Determine Unadjusted Use−Case Weight.

Step 1.1.1 − Find the number of transactions in each Use−Case.

If the Use-Cases are written with User Goal Levels, a transaction is equivalent to a step in the Use-Case. Find the number of transactions by counting the steps in the Use-Case.

Step 1.1.2 – Classify each Use-Case as Simple, Average or Complex based on the number of transactions in the Use-Case. Also, assign Use-Case Weight as shown in the following table –

Use-Case Complexity	Number of Transactions	Use-Case Weight
Simple	≤3	5
Average	4 to 7	10
Complex	>7	15

Step 1.1.3 – Repeat for each Use-Case and get all the Use-Case Weights. Unadjusted Use-Case UUC Weight is the sum of all the Use-Case Weights.

Step 1.1.4 – Find Unadjusted Use-Case Weight UUC using the following table –

Use-Case Complexity	Use-Case Weight	Number of Use-Cases	Product
Simple	5	NSUC	5 × NSUC
Average	10	NAUC	10 × NAUC
Complex	15	NCUC	15 × NCUC
Unadjusted Use-Case Weight UUC			5 × NSUC + 10 × NAUC + 15 × NCUC

Where,

NSUC is the no. of Simple Use-Cases.

NAUC is the no. of Average Use-Cases.

NCUC is the no. of Complex Use-Cases.

Step 1.2 – Determine Unadjusted Actor Weight.

An Actor in a Use-Case might be a person, another program, etc. Some actors, such as a system with defined API, have very simple needs and increase the complexity of a Use-Case only slightly.

Some actors, such as a system interacting through a protocol have more needs and increase the complexity of a Use-Case to a certain extent.

Other Actors, such as a user interacting through GUI have a significant impact on the complexity of a Use-Case. Based on these differences, you can classify actors as Simple, Average and Complex.

Step 1.2.1 – Classify Actors as Simple, Average and Complex and assign Actor Weights as shown in the following table –

Actor Complexity	**Example**	**Actor Weight**
Simple	A System with defined API	1
Average	A System interacting through a Protocol	2
Complex	A User interacting through GUI	3

Step 1.2.2 – Repeat for each Actor and get all the Actor Weights. Unadjusted Actor UA Weight is the sum of all the Actor Weights.

Step 1.2.3 – Find Unadjusted Actor UA Weight using the following table –

Actor Complexity	**Actor Weight**	**Number of Actors**	**Product**
Simple	1	NSA	1 × NSA
Average	2	NAA	2 × NAA
Complex	3	NCA	3 × NCA
Unadjusted Actor Weight UA			1 × NSA + 2 × NAA + 3 × NCA

Where,

NSA is the no. of Simple Actors.

NAA is the no. of Average Actors.

NCA is the no. of Complex Actors.

Step 1.3 – Calculate Unadjusted Use–Case Points.

The Unadjusted Use-Case Weight UUC and the Unadjusted Actor UA Weight together give the unadjusted size of the system, referred to as Unadjusted Use-Case Points.

Unadjusted Use-Case Points UUCP = UUCW + UAW

The next steps are to adjust the Unadjusted Use-Case Points UUCP for Technical Complexity and Environmental Complexity.

Step 2: Adjust For Technical Complexity

Step 2.1 – Consider the 13 Factors that contribute to the impact of the Technical Complexity of a project on Use–Case Points and their corresponding Weights as given in the following table –

Factor	Description	Weight
T1	Distributed System	2.0
T2	Response time or throughput performance objectives	1.0
T3	End user efficiency	1.0
T4	Complex internal processing	1.0
T5	Code must be reusable	1.0
T6	Easy to install	.5
T7	Easy to use	.5
T8	Portable	2.0
T9	Easy to change	1.0
T10	Concurrent	1.0
T11	Includes special security objectives	1.0
T12	Provides direct access for third parties	1.0
T13	Special user training facilities are required	1.0

Many of these factors represent the project's non functional requirements.

Step 2.2 – For each of the 13 Factors, assess the project and rate from 0 irrelevant to 5 very important.

Step 2.3 – Calculate the Impact of the Factor from Impact Weight of the Factor and the Rated Value for the project as

Impact of the Factor = Impact Weight × Rated Value

Step 2.4 – Calculate the sum of Impact of all the Factors. This gives the Total Technical Factor TF as given in table below –

Factor	**Description**	**Weight**	**Rated Value** 0to5 RV	**Impact** (I -W)× RV
T1	Distributed System	2.0		
T2	Response time or throughput performance objectives	1.0		
T3	End user efficiency	1.0		
T4	Complex internal processing	1.0		
T5	Code must be reusable	1.0		
T6	Easy to install	.5		
T7	Easy to use	.5		
T8	Portable	2.0		
T9	Easy to change	1.0		

T10	Concurrent	1.0		
T11	Includes special security objectives	1.0		
T12	Provides direct access for third parties	1.0		
T13	Special user training facilities are required	1.0		
Total Technical Factor TFactor				

Step 2.5 – Calculate the Technical Complexity Factor TCF as –

$$\mathbf{TCF = 0.6 +} \ 0.01 \times \text{TFactor}$$

Step 3: Adjust For Environmental Complexity

Step 3.1 – Consider the 8 Environmental Factors that could affect the project execution and their corresponding Weights as given in the following table –

Factor	Description	Weight
F1	Familiar with the project model that is used	1.5
F2	Application experience	.5
F3	Object oriented experience	1.0
F4	Lead analyst capability	.5
F5	Motivation	1.0
F6	Stable requirements	2.0

F7	Part-time staff	-1.0
F8	Difficult programming language	-1.0

Step 3.2 – For each of the 8 Factors, assess the project and rate from 0 irrelevant to 5 very important

Step 3.3 – Calculate the Impact of the Factor from Impact Weight of the Factor and the Rated Value for the project as

Impact of the Factor = Impact Weight × Rated Value

Step 3.4 – Calculate the sum of Impact of all the Factors. This gives the Total Environment Factor EFactor as given in the following table –

Factor	**Description**	**Weight**	**Rated Value** 0to5 RV	**Impact** (I -W)× RV
F1	Familiar with the project model that is used	1.5		
F2	Application experience	.5		
F3	Object-oriented experience	1.0		
F4	Lead analyst capability	.5		
F5	Motivation	1.0		
F6	Stable requirements	2.0		
F7	Part-time staff	-1.0		
F8	Difficult programming language	-1.0		
Total Environment Factor EFactor				

Step 3.5 – Calculate the Environmental Factor EF as –

1.4 + -0.03 * EFactor

Step 4: Calculate Adjusted Use-Case Points UCP

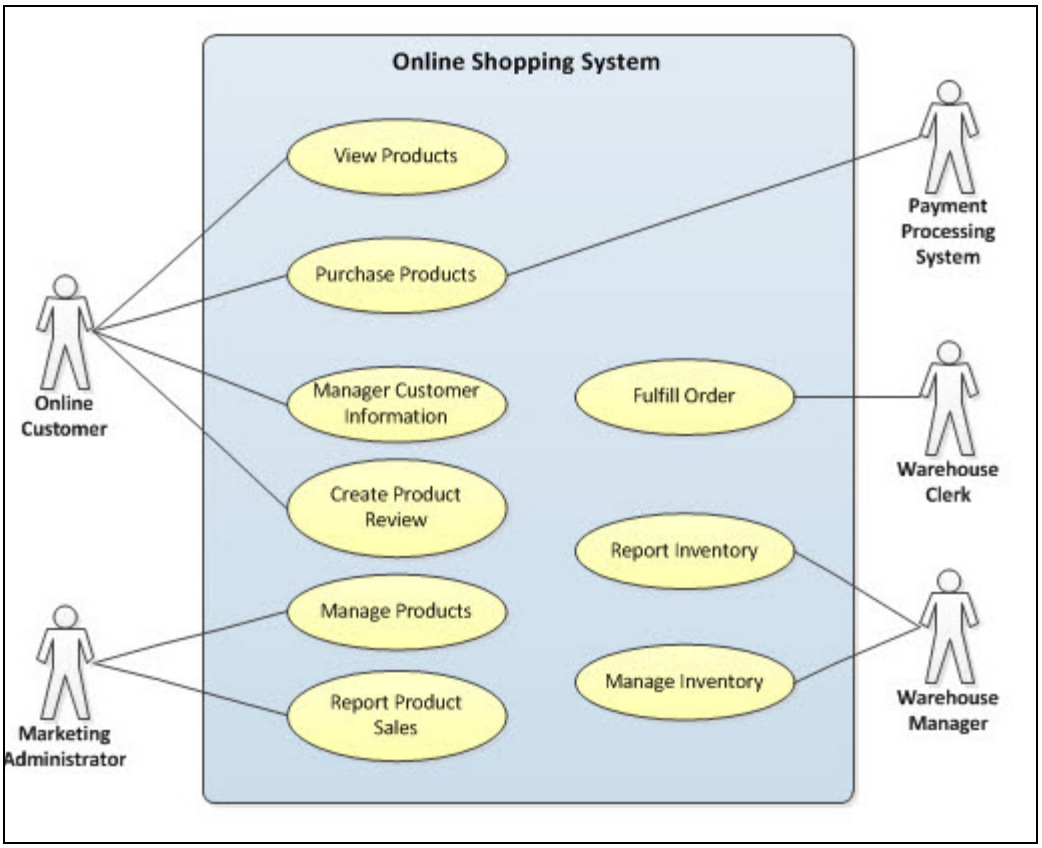
Calculate Adjusted Use–Case Points as –

UCP = UUCP × TCF × EF

OUTPUT:

EXAMPLE:

To illustrate the process of calculating the UCP, an Online Shopping System will be used. The diagram below depicts the Use Case Diagram for the system to be developed.



Unadjusted Use Case Weight (UUCW)

To calculate the UUCW, the use cases must be defined and the number of transactions for each use case identified. The Online Shopping System use case diagram is depicting that nine use cases exist for the system. Assuming 2 of these use cases are simple, 3 are average and 4 are complex, the calculation for UUCW is as follows:

UUCW = (Total No. of Simple Use Cases x 5) + (Total No. Average Use Cases x 10) + (Total No. Complex Use Cases x 15)

For the Online Shopping System, the UUCW = (2 x 5) + (3 x 10) + (4 x 15) = 100

UUCW = 100

Unadjusted Actor Weight (UAW)

To calculate the UAW, the actors must be identified. The Online Shopping System use case diagram is depicting five actors; One simple for the Payment Processing System and four complex for each of the human users actors (i.e. Online Customer, Marketing Administrator, Warehouse Clerk, Warehouse Manager.) The calculation for UAW is as follows:

UAW = (Total No. of Simple Actors x 1) + (Total No. Average Actors x 2) + (Total No. Complex Actors x 3)

Factor	Description	Weight	Assigned Value	Weight x Assigned Value
T1	Distributed system	2.0	5	10
T2	Response time/performance objectives	1.0	5	5
T3	End-user efficiency	1.0	3	3
T4	Internal processing complexity	1.0	2	2
T5	Code reusability	1.0	3	3
T6	Easy to install	0.5	1	0.5
T7	Easy to use	0.5	5	2.5
T8	Portability to other platforms	2.0	2	4
T9	System maintenance	1.0	2	2
T10	Concurrent/parallel processing	1.0	3	3
T11	Security features	1.0	5	5
T12	Access for third parties	1.0	1	1
T13	End user training	1.0	1	1

1) + (0 x 2) + (4 x 3) = 13
UAW = 13

	Total (TF):	**42**

Technical Complexity Factor (TCF)

To calculate the TCF, each of the technical factors is assigned a value based on how essential the technical aspect is to the system being developed. The diagram below shows the assigned values for the Online Shopping System. The values are multiplied by the weighted values and the total TF is determined.

Next, the TCF is calculated:

TCF = 0.6 + (TF/100)
For the Online Shopping System, TCF = 0.6 + (42/100) = 1.02
TCF = 1.02

Environmental Complexity Factor (ECF)

Factor	Description	Weight	Assigned Value	Weight x Assigned Value
E1	Familiarity with development process used	1.5	3	4.5
E2	Application experience	0.5	3	1.5
E3	Object-oriented experience of team	1.0	2	2
E4	Lead analyst capability	0.5	5	2.5
E5	Motivation of the team	1.0	2	2
E6	Stability of requirements	2.0	1	2
E7	Part-time staff	-1.0	0	0

E8	Difficult programming language	-1.0	4	-4
			Total (EF):	**10.5**

To calculate the ECF, each of the environmental factors is assigned a value based on the team experience level. The diagram below shows the assigned values for the Online Shopping System. The values are multiplied by the weighted values and the total EF is determined.

Next, the ECF is calculated:

ECF = 1.4 + (-0.03 x EF)
For the Online Shopping System, ECF = 1.4 + (-0.03 * 10.5) = 1.085
ECF = 1.085

Use Case Points (UCP)

Once the Unadjusted Use Case Weight (UUCW), Unadjusted Actor Weight (UAW), Technical Complexity Factor (TCF) and Environmental Complexity Factor (ECF) has been determined, the Use Case Points (UCP) can be calculated with the following formula:

UCP = (UUCW + UAW) x TCF x ECF
For the Online Shopping System, UCP = (100 + 13) x 1.02 x 1.085 = 125.06
UCP = 125.06

For the Online Shopping System, the total estimated size to develop the software is 125.06 Use Case Points.

FUNCTION POINT ;

Aim: To estimate the effort using Function point metric .

Procedure:

FP measures functionality from the User's point of view like what the user receives from the software and what the user requests from the software. It focuses on what functionality is being delivered.

"A Functional Point" is a unit of measurement to express the amount of business functionality an information system provides to a user."

Using historical data, the FP metric can then be used to

1. Estimate the cost or effort required to design, code, and test the software.
2. Predict the number of errors that will be encountered during testing
3. Forecast the number of components and/or the number of projected source lines in the implemented system.

FPs are derived using an empirical relationship based on countable measures of software's information domain and qualitative assessments of software complexity. The five functional units which is consider as information domain as input to calculate the FP are.

1. Internal Logic Files (ILF) – The control info or logically related data that is present within the system.
2. External Interface Files (EIF) – The control data or other logical data i.e referenced by the system but present in another system.
3. External Inputs (EI) – Data / control info that comes from outside our system
4. External Outputs (EO) – data that goes out of the system after generation
5. External Enquired (EQ) – Combination of i/o – o/p resulting data retrieval

To compute FP, the following relationship is used:

FP = UFP X CAF

Where UFP is Unadjusted Functional Point
CAF is Complexity Adjustment Factor

Step 1: Calculating UFP

Each Function point is ranked according to complexity. There exist pre-defined weights for each F.P in each category. Organizations that use function point methods develop criteria for determining whether a particular entry is simple, average, or complex. Nonetheless, the determination of complexity is somewhat subjective.

UFP = Sum of all the Complexities of all the EI's, EO's EQ's, ILF's and EIF's

Functional unit	WEIGHTING FACTORS		
	LOW	AVERAGE	HIGH
EXTERNAL INPUTS(EI)	3	4	6
EXTERNAL OUTPUTS(EO)	4	5	7
EXTERNAL ENQUIRED(EQ)	3	4	6
INTERNAL LOGIC FILES(ILF)	7	10	15

EXTERNAL INTERFACE FILES(EIF)	5	7	10

Step 2: Calculating CAF

CAF = 0.65 + (0.01 x ∑ Fi)

Where Fi is value adjustment factors based on responses to the 14 questions.

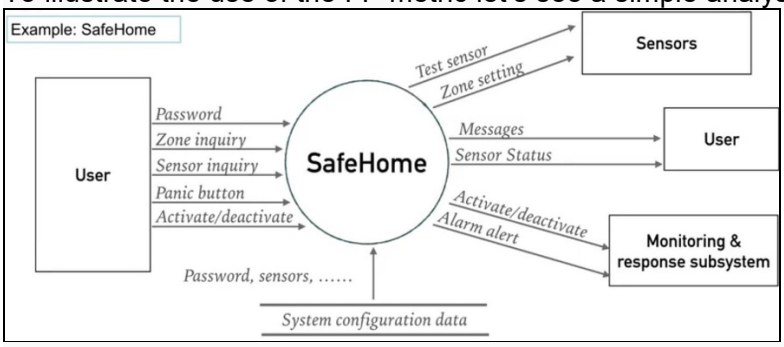
1.Data Communication
2.Distributed Data Processing
3.Performance
4.Heavily used configuration.
5.Transaction role
6.Online data entry
7.End-user efficiency
8.Online update
9.Complex Processing
10.Reusability
11.installation ease
12.Operational ease
13.Multiple sides
14.Facilitate change

Complexity Adjustment Factor is calculated using 14 aspects of processing complexity and these 14 questions answered on a scale of 0 – 5

0 – No Influences or no important or no applicable

1 – Incidental 2 – Moderate 3 – Average 4 – Significant 5 – Essential

To illustrate the use of the FP metric let's see a simple analysis model Safe Home :



The function manages user interaction, accepting a user password to activate or deactivate the system, and allows inquiries on the status of security zones and various security sensors. The function displays a series of prompting messages and sends appropriate control signals to various components of the security system.

The data flow diagram is evaluated to determine a set of the key information domain measures required for computation of the function point metric.

Three external inputs—password, panic button, and activate/deactivate

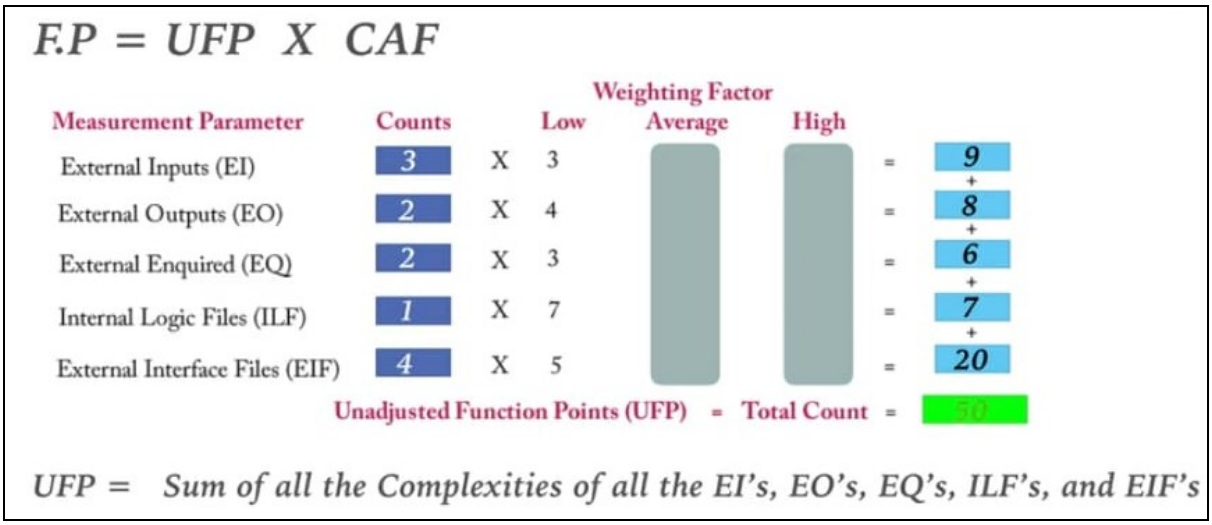
Two external inquiries—zone inquiry and sensor inquiry. One ILF (system configuration file)

Two external outputs - (messages and sensor status) and four EIFs (test sensor, zone setting, activate/deactivate, and alarm alert)

Now calculating Unadjusted FP with the simple weighting factor.

OUTPUT:

Consider EI, EO, EQ, ILF, ELF are given as 3,2,2,1,4 and are of low weighting factor. Then



Therefore, UFP is equal to 50
Now moving to calculate CAF
As we know
CAF = 0.65 + (0.01 x ∑ Fi)
Let's calculate in a moderately complex product (2-Moderate)
Then Submission Fi = 14X2 = 28
CAF =0.65+(0.01*28)= 0.93
FP= 50 x 0.93 =46.5
Therefore, FP is equal to 46.5
In this way we can calculate the unadjusted functional point, Complexity factor, and finally Functional Point and from which we can estimate the lines of code required to develop the software, the number of probability errors, the effort required, development time, etc concerning previous data.

11.Q. Develop a tool which can be used for quantification of all the non-functional requirements.

Ans:

Non-Functional Requirement: Non-functional requirements is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. The system's overall properties commonly mark the difference between whether the development project has succeeded or failed

Non-functional requirements are often called **quality attributes** of a system however, there is a distinction between the two. Non-functional requirements are the criteria for evaluating how a software system should perform and a software system must have certain quality attributes in order to meet non-functional requirements. So, when we say a system should be "secure", "highly-available", "portable", "scalable" and so on, we are talking about its quality attributes. Other terms for non-functional requirements are "qualities", "quality goals", "quality of service requirements", "constraints", "non-behavioral requirements", or "technical requirements". Informally these are sometimes called the "ilities", from attributes like stability and portability. Qualities—that is non-functional requirements—can be divided into two main categories

1. Execution qualities, such as safety, security and usability, which are observable during operation (at run time).
2. Evolution qualities, such as testability, maintainability, extensibility and scalability, which are embodied in the static structure of the system.

A system may be required to present the user with a display of the number of records in a database. This is a functional requirement. How current this number needs to be, is a non-functional requirement. If the

number needs to be updated in real time, the system architects must ensure that the system is capable of displaying the record count within an acceptably short interval of the number of records changing. Sufficient network bandwidth may be a non-functional requirement of a system. Other examples include:

- Accessibility
- Adaptability
- Portability
- Quality (e.g., faults discovered, faults delivered, fault removal efficacy)
- Readability
- Reliability
- Response time
- Reusability
- Robustness
- Documentation
- Durability
- Efficiency (resource consumption for given load)
- Effectiveness (resulting performance in relation to effort)
- Scalability (Horizontal, vertical)
- Security (Cyber and physical)
- Software, tools, standards etc. Compatibility
- Stability
- Supportability
- Testability

Importance of quantification of NFR'S

Requirement Analysis is the phase, which determines the success or failure of a software project. Poor requirement analysis increases the cost of development by 70-85%, which requires reworking on all phases of software project. It is observed that non-functional properties are often ignored while focusing on the functionality of the software. Many software systems have failed because of negligence of non-functional requirements. Therefore, it is necessary to measure the satisfaction level of non-functional requirement during software development process

Quantification of desirable properties of a system is an integral part of the software engineering. Most of requirement analysis methods which do not consider nonfunctional requirements lead to serious software development problems. The problems faced because of non-functional requirements omissions are more critical than the problems of functional requirements omissions. Therefore, it is necessary to measure non-functional requirement during software development process. Since software reliability is the major concern for quality system and considered as one of the most desirable quality attributes of the system, hence its quantitative measurement is an essential part of the development of a quality system. Estimation of software reliability becomes more important for those applications where risk is major consideration.

Software reliability is the probability to perform failure free operation and produce correct output for a specified time under specified conditions. Once the system is installed, it is not possible to test system reliability in an operational environment because failure data collected in an uncontrolled tested environment may be limited. In this case, faults may not be necessarily removed as failures are identified. Therefore, failures are considered as one of the factors affecting the software. Failure data need to be properly collected and measured during software development process for the assessment of reliability

of software systems. Software reliability is measured by collecting the historical data and various distribution curves.

How to Quantitification of NFR'S

Computer programming languages such as C, Java, and COBOL are best suited to develop such software systems whose behavior can be represented by mathematical model or logical reasoning. Since mathematical models do not work to develop software systems which require human judgment and decision-making capabilities, Zadeh [20] introduced the framework of Fuzzy sets to deal with a poorly defined concept in a coherent and structured way. Examples of poorly defined concepts suitable for the application of Fuzzy logic are semantic variables, such as high reliability, good performance, low maintenance, etc. The basis for proposing fuzzy logic was that human being relies on imprecise expression such as high, good or excellent. But software-based systems work on Boolean logic

Reliability is a key non-functional requirement, which can be divided into three Sub-NFRs (Availability and Recoverability).

1.Availability: Every system must be alive for service when it is requested by end-users. Availability of system indicates how reliable system is during operational hours. Therefore, Availability is considered as one of the important metrics used to assess the performance of a system, accounting for reliability of a system. It is defined as the ability to perform a required function under specified conditions at a given point of time or over a given time interval

2.Recoverability is another important metric used to assess the reliability of a system. It is very important to know how often system fail to deliver expected level of service. This helps to measure the amount of data loss and downtime that a business can endure and decide how to recover from these failures. Good architecture provides recoverability in the time specified in a service-level agreement. To make system recoverable, backup of data should be taken at regular

intervals. If mirroring of data is properly maintained, system will be more recoverable in case of any failure or disaster.

Bayesian belief network represents compact networks of probabilities that capture the probabilistic relationships between variables, as well as historical information on their relationships. From another perspective, Bayesian belief network is a combination of Bayesian probability theory and the concept of conditional independence

Bayesian network is a way of representing joint probability distribution using Direct Acyclic Graph network structure given a set of variables . Nodes of graph represent random uncertain variables and the arcs represent the Bayesian probabilistic relationships between these variables. It is a collection of conditional probability distributions where each variable in the directed acyclic graph is denoted by a conditional distribution given its parent nodes, Par (V_i). By using this we can quantify NFR 's

Conclusion: Therefore, it is necessary to measure the satisfaction level of non-functional requirement during software development process

QUESTION-12

Write C/C++/Java/Python program for classifying the various types of coupling.

- ## WHAT IS COUPLING?

In software engineering, coupling is the degree of interdependence between software modules; a measure of how closely connected two routines or modules are; the strength of the relationships between modules.

- Coupling can simply be understood as how the different modules in a software interact with each other.
- It is imperative to have low coupling which means, every module must have very minimal interaction with other modules.

- ## TYPES OF COUPLING

Coupling in software engineering is classified into categories :

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

1.Content coupling :

Content Coupling is observed when one module /class accesses and modifies the data of another module.

Two modules are content coupled if:

- one module references or alters data contained inside another module.
- one module branches into another module.

```
class content(object):
    def __init__(self):
        self.name = 'Before calling'
        self.contentcoupling = self.contentcoupling()
    def display(self):
        self.contentcoupling.display()
    class contentcoupling():
        def display(self):
            self.name = 'After calling'
            print(self.name)


obj1 = content()
obj1.display()


"""In this example the content class has a variable name which
initially stores some data but after the inner class contentcoupling
access it, that data is replaced by some other data.."""
```

2.Common coupling :

Here the modules have shared data such as global data structures. The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses.

```python
class Base(object):
    def __init__(self,number):
        self.number= number

    def fact(self):
        if (self.number == 0):
            print("The factorial of the number is 1")
        else:
            for i in range(1, self.number + 1):
                i = i * (i + 1)
            print(f"The factorial of the number is {format(i)}")

x  = Base(0)
x.fact()

"""Here the variable number in class Base decides which lines of code is to be executed
in the function fact()..To avoid this type of coupling we need to factory design methods
i.e using different functions for solving each case.."""

#In our example we can use one function for number == 0  and another for number > 0
```

3.Control coupling :

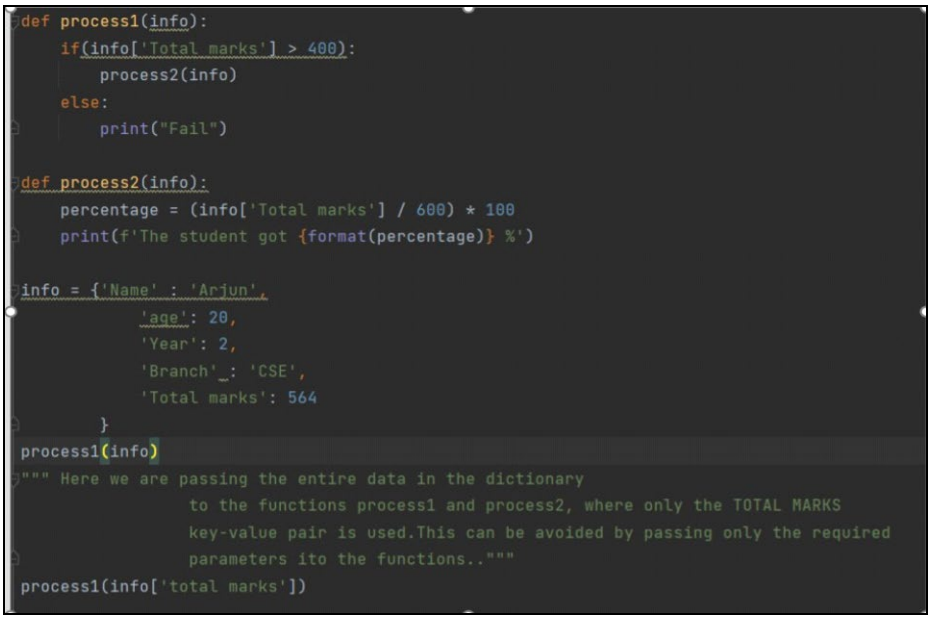
A module is said to be control coupled when the flow of execution is decided by a variable of another class. This means which lines of code should be executed is decided by some parameter or a variable. This is more commonly found where there are control structures like if-else conditions.

```python
def process1(info):
    if(info['Total marks'] > 400):
        process2(info)
    else:
        print("Fail")

def process2(info):
    percentage = (info['Total marks'] / 600) * 100
    print(f'The student got {format(percentage)} %')

info = {'Name' : 'Arjun',
            'age': 20,
            'Year': 2,
            'Branch': 'CSE',
            'Total marks': 564
        }
process1(info)
""" Here we are passing the entire data in the dictionary
                to the functions process1 and process2, where only the TOTAL MARKS
                key-value pair is used.This can be avoided by passing only the required
                parameters ito the functions.."""
process1(info['total marks'])
```

4.Stamp coupling :

Two classes are said to be stamp coupled if one class sends a collection or object as parameter and only a few data members of it is used in the

second class. This type of coupling is desirable when the number of parameters that must be passed exceeds three.



```
def process1(info):
    if(info['Total marks'] > 400):
        process2(info)
    else:
        print("Fail")

def process2(info):
    percentage = (info['Total marks'] / 600) * 100
    print(f'The student got {format(percentage)} %')

info = {'Name' : 'Arjun',
            'age': 20,
            'Year': 2,
            'Branch': 'CSE',
            'Total marks': 564
        }
process1(info)
""" Here we are passing the entire data in the dictionary
                to the functions process1 and process2, where only the TOTAL MARKS
                key-value pair is used.This can be avoided by passing only the required
                parameters ito the functions.."""
process1(info['total marks'])
```

5.Data coupling :

Data coupling is the type of coupling that occurs when necessary data is sent as parameters between methods and classes. Data coupling is the most desirable among the 5 main types of coupling.

```
def Datacoupling():
    number1 = 10
    number2 = 20
    number3 = 30
    number4 = 40
    number5 = 50
    printer(number1,number2,number3,number4,number5)

def printer(number1,number2,number3,number4,number5):
    print('Number 1:',number1)
    print('Number 2:',number2)
    print('Number 3:',number3)
    print('Number 4:',number4)
    print('Number 5:',number5)

Datacoupling()

""" To avoid this we can either create objects of the parameters and
if possible pass the object which contains the data itself but this
    results in stamp coupling.So to avoid this from our example we can
    create multiple methods for printing the numbers.."""

def printer1(number1,number2,number3):
    print('Number 1:',number1)
    print('Number 2:',number2)
    print('Number 3:',number3)

def printer2(number4,number5):
    print('Number 4:',number4)
    print('Number 5:',number5)
```

Apart from these five major types of coupling there are other types of coupling for which various design patterns have been proposed to eliminate them. We generally concentrate on these five types mainly since these are frequently found in our software applications..

13.write a c/c++/python program for classifying the various types of cohesion

AIM: To implement a program for classifying the various types of cohesio

The main points in the context of cohesion are ::

- What is cohesion
- Advantages of cohesion
- Types and levels of cohesion
- Cohesion in java/c/c++/python
- Example program
-

Cohesion :

- The measure of how strongly the elements are related functionally inside a module is called **cohesion in software engineering.**
- Elements inside a module can be instructions, groups of instructions, definition of data, call from another module etc.
- Basically, cohesion is the internal glue that keeps the module together. **A good software design will have high cohesion**.

Example :

- Suppose we have a class that multiply two numbers, but the same class creates a pop up window displaying the result. This is the example of low cohesive class because the window and the multiplication operation don't have much in common.
- To make it high cohesive, we would have to create a class Display and a class Multiply. The Display will call Multiply method to get the result and display it. This way to develop a high cohesive solution.

ADVANTAGES :

- Less complexity
- Easy to understand
- Easy maintainence
- Increases reusability

Types of cohesion :

1)FUNCTIONAL COHESION
2) SEQUENTIAL COHISION

3) COMMUNICATIONAL COHISION
4) PROCEDURAL COHISION
5) TEMPORAL COHISION
6) LOGICAL COHISION
7) COINCIDENTAL COHESION

1)FUNCTIONAL COHESION

- Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.

Characteristics and Example :

- The purpose of functional cohesion is single minded, strong and focused.
- Some of the examples of functional cohesion are read transaction record, cosine angle computation, seat assignment to an airline passenger etc.

2) SEQUENTIAL COHISION:

An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.

Characteristics and Example :

- Sequential cohesion is easy maintenance and provides a good coupling

It cannot be reused.

- Some of the examples of sequential cohesion are cross validate record and formatting of module, raw records usage, formatting of raw records, cross validation of fields in raw records, returning of formatted cross validated records.

3) COMMUNICATIONAL COHISION:

- Two elements operate on the same input data or contribute towards the same output datais known as communicational cohision.

EXAMPLE :

- this cohesion is not flexible like we can only focus on some of the activities and not others at once.
- Some of the examples of communicational cohesion are customer details determining modules, usage of customer account number, finding the name of the customer, finding the loan balance of the customer, returning loan balance and the name of the customer etc

4)PROCEDURAL COHISION :

- Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable.

Example :

- Some of the examples of procedural cohesion are read, write, edit of the module, record use out, writing out the

record, reading the record, zero padding to the numeric fields, returning records etc.

5)TEMPORAL COHISION :

- The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time-span.

Example :

- Some of the examples of temporal cohesion are initialization of module, setting the counter to zero, opening the student file, initializing the array etc.

6) LOGICAL COHISION:

- The elements of a module are related logically but not functionally.

Some of the examples of logical cohesion are module for displaying record as below:

- Read the type of record.
- If type of the record is student. then,
- Display the record from student.
- Else if the type of the record is staff, then
- Display the record from staff.

7) COINCIDENTAL COHISION :

- The elements are not related(unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion.

Example :

- Some of the examples of coincidental cohesion are module for customer record usage, displaying of customer record, calculation of total sales, reading the transaction record etc.
  COHESION IN JAVA :

- In object oriented design, cohesion refers all about how a single class is designed. Cohesion is the Object Oriented principle most closely associated with making sure that a class is designed with a single, well-focused purpose.
- The more focused a class is, the cohesiveness of that class is more. The advantages of high cohesion is that such classes are much easier to maintain (and less frequently changed) than class
- Another benefit of high cohesion is that classes with a well-focused purpose tend to be more reusable than other classes.

Example 1:-

```
// Java program to illustrate
// high cohesive behavior
class Multiply {
        int a = 5;
        int b = 5;
        public int mul(int a, int b)
        {
                this.a = a;
                this.b = b;
                return a * b;
        }
}
```

```
class Display {
	public static void main(String[] args)
	{
		Multiply m = new Multiply();
		System.out.println(m.mul(5, 5));
	}
}
```

OUTPUT:

25

Example 2:-

```
// Java program to illustrate
// high cohesive behavior
class Name {
	String name;
	public String getName(String name)
	{
		this.name = name;
		return name;
	}
}

class Age {
	int age;
	public int getAge(int age)
	{
		this.age = age;
		return age;
	}
}

class Number {
```

```
        int mobileno;
        public int getNumber(int mobileno)
        {
                this.mobileno = mobileno;
                return mobileno;
        }
}

class Display {
        public static void main(String[] args)
        {
                Name n = new Name();
                System.out.println(n.getName("Geeksforgeeks"));
                Age a = new Age();
                System.out.println(a.getAge(10));
                Number no = new Number();
                System.out.println(no.getNumber(1234567891));
        }
}
```

OUTPUT:

```
10
1234567891
```

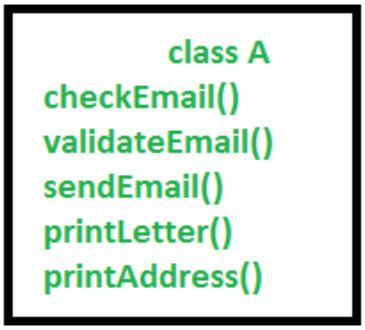


Fig: Low cohesion

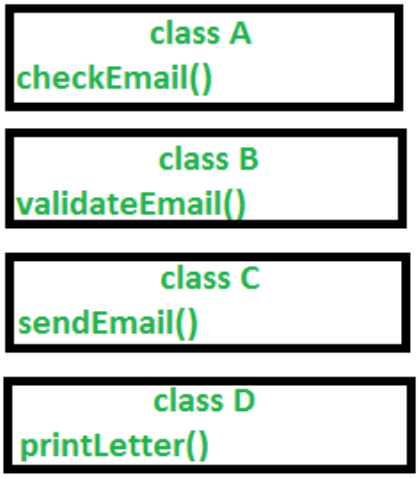


Fig: High cohesion

14.Write a C/C++/Java/Python program for object oriented metrics for design proposed Chidamber and kremer . (Popularly called as CK metrics) Explain the concept of a tree (L2).

What is CK metrics

1. THE METRICS PROPOSED BY CHIDAMBER AND KEMERER [4], NOW REFERRED TO AS THE CK METRICS, HAVE BECOME WELL KNOWN AND WIDELY ACCEPTED BY THE SOFTWARE ENGINEERING COMMUNITY. THE CK METRICS CAN BE USED TO MEASURE SOME CHARACTERISTICS OF OO SYSTEMS SUCH AS CLASSES, MESSAGE PASSING, INHERITANCE, AND ENCAPSULATION.

Types Of CK metrics

These are consists of six metrics calculated for each class.

1. Weighted Methods per Class (WMC)

2. Depth of Inheritance Tree (DIT)

3. Number of Children (NOC)

4. Response For a Class (RFC)

5. Coupling Between Objects (CBO)

6. And Lack of Cohesion in Methods (LCOM)

Weighted methods per class

Definition: Weighted methods for Class (WMC) was originally proposed by C&K as the sum of all the complexities of the methods in the class. Rather than use Cyclomatic Complexity they assigned each method a complexity of one making WMC equal to the number of methods in the class. Most 'classic' implementations follow this rule.
The number of methods and the complexity of methods involved is a predictor of how much time and effort is required to develop and maintain the class.
The larger the number of methods in a class the greater the potential impact on children, since children will inherit all the methods defined in the class.

Classes with large numbers of methods are likely to be more application specific, limiting the possibility of reuse.
I have quite a few issues with this metric, starting with the name – don't call a metric Weighted Methods per class if you are immediately going to assign each method an equal and arbitrary value of 1. Their view that classes with more methods are less likely to be reused seems strange in that, potentially, there is more there to reuse! I think that number of methods is a better name for this method and that Total Cyclomatic Complexity for a class is a more useful pointer to one or more potential 'bad smells'.

Depth of Inheritance Tree

Definition: Depth of Inheritance Tree (DIT) is a count of the classes that a particular class inherits from. C&K suggested the following consequences based on the depth of inheritance.
The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit, making it more complex to predict its behaviour.
Deeper trees constitute greater design complexity, since more methods and classes are involved.
The deeper a particular class is in the hierarchy, the greater the potential reuse of inherited methods.
The first two points seem to be interpreted as 'bad' indicators while the third seems to be suggested as 'good' – making it difficult to decide whether DIT should be large or small. I would suggest that a deep hierarchy is a good thing – you don't really want to punish OO programmers for using inheritance.

Number Of children

Definition: Number of Children (NOC) is defined by C&K the number of immediate subclasses of a class. C&K's view was that
1.The greater the number of children, the greater the level of reuse, since inheritance is a form of reuse.
2.The greater the number of children, the greater the likelihood of improper abstraction of the parent class. If a class has a large number of children, it may be a case of misuse of subclassing.
3.The number of children gives an idea of the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in that class.

Response For Class

Definition: This is the size of the Response set of a class. The Response set for a class is defined by C&K as 'a set of methods that can potentially be executed in response to a message received by an object of that class'. That means all the methods in the

class and all the methods that are called by methods in that class. As it is a set each called method is only counted once no matter how many times it is called. C&K's view was that
If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes more complicated since it requires a greater level of understanding required on the part of the tester.
The larger the number of methods that can be invoked from a class, the greater the complexity of the class.
A worst case value for possible responses will assist in appropriate allocation of testing time.

Coupling between Objects

Definition: Coupling between objects (CBO) is a count of the number of classes that are coupled to a particular class i.e. Where the methods of one class call the methods or access the variables of the other. These calls need to be counted in both directions so the CBO of class A is the size of the set of classes that class A references and those classes that reference class A. Since this is a set – each class is counted only once even if the reference operates in both directions i.e. If A references B and B references A, B is only counted once
C&K viewed that CBO should be as low as possible for three reasons,
Increased coupling increases interclass dependencies, making the code less modular and less suitable for reuse. In other words if you wanted to package up the code for reuse you might end up having to include code that was not really fundamental to the core functionality.
More coupling means that the code becomes more difficult to maintain since an alteration to code in one area runs a higher risk of affecting code in another (linked) area.
The more links between classes the more complex the code and the more difficult it will be to test.

Lack of Cohesion of Methods

Definition: LCOM is probably the most controversial and argued over of the C&K metrics. In their original incarnation C&K defined LCOM based on the numbers of pairs of methods that shared references to instance variables. Every method pair combination in the class was assessed. If the pair do not share references to any instance variable then the count is increased by 1 and if they do share any instance variables then the count is decreased by 1. LCOM is viewed as a measure of how well the methods of the class co-operate to achieve the aims of the class. A low LCOM value suggests the class is more cohesive and is viewed as better.

C&K's Rationale for the LCOM method was as follows

Cohesiveness of methods within a class is desirable, since it promotes encapsulation.

Lack of cohesion implies classes should probably be split into two or more subclasses.
Any measure of disparateness of methods helps identify flaws in the design of classes.
Low cohesion increases complexity, thereby increasing the likelihood of errors during the development process.

C# source code

```
using System;
public class employee                    //employee class
{
private:
String name;                             //employee name
int number;                              //employee number
public:
void getdata()
{
Console.WriteLine("enter name:");
name= Console.ReadLine();
Console.WriteLine("enter number :") ;
number=int.parse(Console.ReadLine()) ;
}
void putdata()
{
Console.WriteLine("The name is:" +name);
Console.WriteLine("Number=" +number);
}
}
public class manager : employee        //management class
{
private :
String title ;                             // "vice-president " etc.
double dues ;                            // golf club dues
public:
void getdata()
{
base.getdata();
Console.WriteLine("enter title :");
title=Console.ReadLine();
Console.WriteLine("enter golf club dues:");
dues=double.parse(Console.ReadLine());
}
void putdata()
{
base.putdata();
```

```
Console.WriteLine(“title:” +title);
Console.ReadLine(“dues:”+dues);
}
}
public class scientist : employee                    // scientist class
{
private:
int pubs ;
public:
void getdata()
{
base.getdata();
Console.WriteLine(”enter number of pubs:”) ;
pubs=int.parse(Console.ReadLine());
}
void putdata()
{
base.putdata();
Console.WriteLine(“number of pubs:” +pubs);
}
}
public class laborer : employee                      // laborer class
{
private:
int a;
public:
int hours;
void getdata()
{
base.getdata();
Console.WriteLine(“Enter number of hours:”) ;
hours=int.parse(Console.ReadLine());
}
void cal( )
{
int total=0;
total = LEN*40;
}
void putdata()
{
base.putdata();
Console.WritLine(“number of hours :” +hours) ;
Console.WriteLine(“Total:” +total);
}
}
public class hourlyemployee: laborer           //hourlyemployee class
{
private:
double sal;
public:
```

```
void getdata()
{
base.getdata();
Console.WriteLine(”enter number of hours:”) ;
hours=int.parse(Console.ReadLine());
}
void salary()
{
sal=hours*250;
}
void putdata()
{
base.putdata();
Console.WriteLine(“The salary is: “ +sal);
}
void main()                                          //main method
{
manager m1 = new manager();
manager m2 = new manager();
scientist s1= new scientist();
laborer L1 = new laborer();
hourlyemployee h1 = new hourlyemployee();
Console.WriteLine(“Enter data for manager 1”);      //get data for several employees
m1.getdata();
Console.WriteLine(“Enter data for manager 2”);
m2.getdata();
Console.WriteLine(“Enter data for scientist 1”);
s1.getdata();
Console.WriteLine(“Enter data for laborer 1”);
L1.getdata();
Console.WriteLine(“Enter data for hourlyemployee 1”);
h1.getdata();
Console.WriteLine(“Data on manager 1”);
m1.putdata();
Console.WriteLine(“Data on manager 2 “);
m2.putdata();
Console.WriteLine(“Data on scientist 1”);
s1.putdata();
Console.WriteLine(”Data on Laborer 1”);
L1.putdata();
Console.WriteLine(“Data on hourly employee”);
h1.putdata();
}
```

CASE TOOL FOR CK METRICS

1. **Chidamber & Kemerer Java Metrics** is an open source command-line tool. It calculates the C&K object-oriented metrics by processing the byte-code of compiled Java.

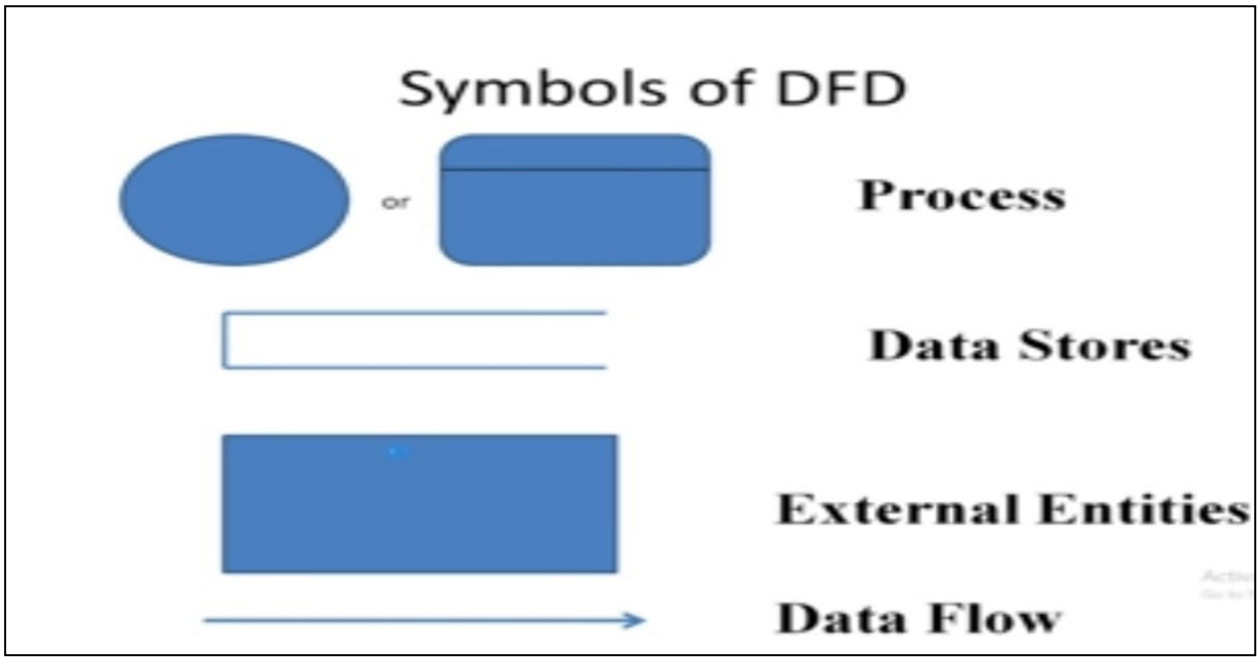
15.Convert the DFD into appropriate architecture styles.

Aim:CONVERT DFD INTO APPROPRIATE ARCHITECTURE STYLES.

Topic breif details:

DFD:

- DATA FLOW DIAGRAM IS A GRAPHICAL REPRESENTATION OF FLOW OF DATA FRIM ONE COMPONENT TO OTHER COMPONENT IN ANY INFORMATION SYSTEM.
- IT GIVES OVERVIEW OF THE SYSTEM WITHOUT GOING INTO DEEP DETAILS OF THE SYSTEM.



DFD LEVELS:

- Level 0(context diagram)
- Level 1
- Level 2
- ….
- ….

ANALYSING ARCHITECTURAL DESIGN:

- Collect scenarios
- Elicit requirements
- Describe the architectural styles/pattern
  - Process view
  - Module view
  - Data flow view
  - Evaluate quality attributes
  - Identify the sensitivity of quality attributes.

There are many types of architectural styles:

- Data centered
- Data flow

- Call and return
- Object oriented
- Layered

Each architectural style describes the system category that encompasses :

- Set of components: perform function required by the system.
- Set of connectors: Enable communication, cooperation and coordination among components.
- Constraints: define how components are integrated to form the system

- Sematic models: enables designers to understand the overall properties of the system by analyzing the known properties of its constituents.

Mapping requirements to software architecture in structure design:

Structured design

Structured design is often characterized as a data flow-oriented design method because it provides a convenient transition from a data flow diagram to software architecture.

Transform Mapping:

Transform mapping is a set of design steps that allows a DFD with transform flow characteristics to be mapped into a specific architectural style. In this section transform mapping is described by applying design steps to an example system—a portion of the SafeHome security software.

Sample output:

An Example

The Safe Home security system.It interacts with a user

through a series of typed inputs and displays. The level 0 data flow diagram for *Safe Home,* , is shown in Figure 14.5.

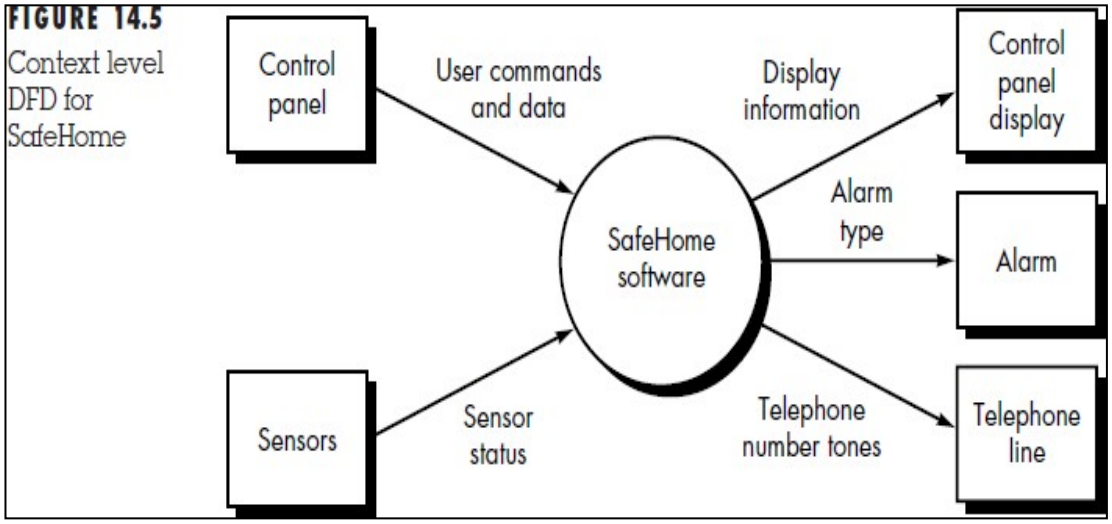


FIGURE 14.5 Context level DFD for SafeHome

14.6.1 Design Steps

The steps begin with a re-evaluation of work done during requirements analysis and then move to the design of the software architecture.

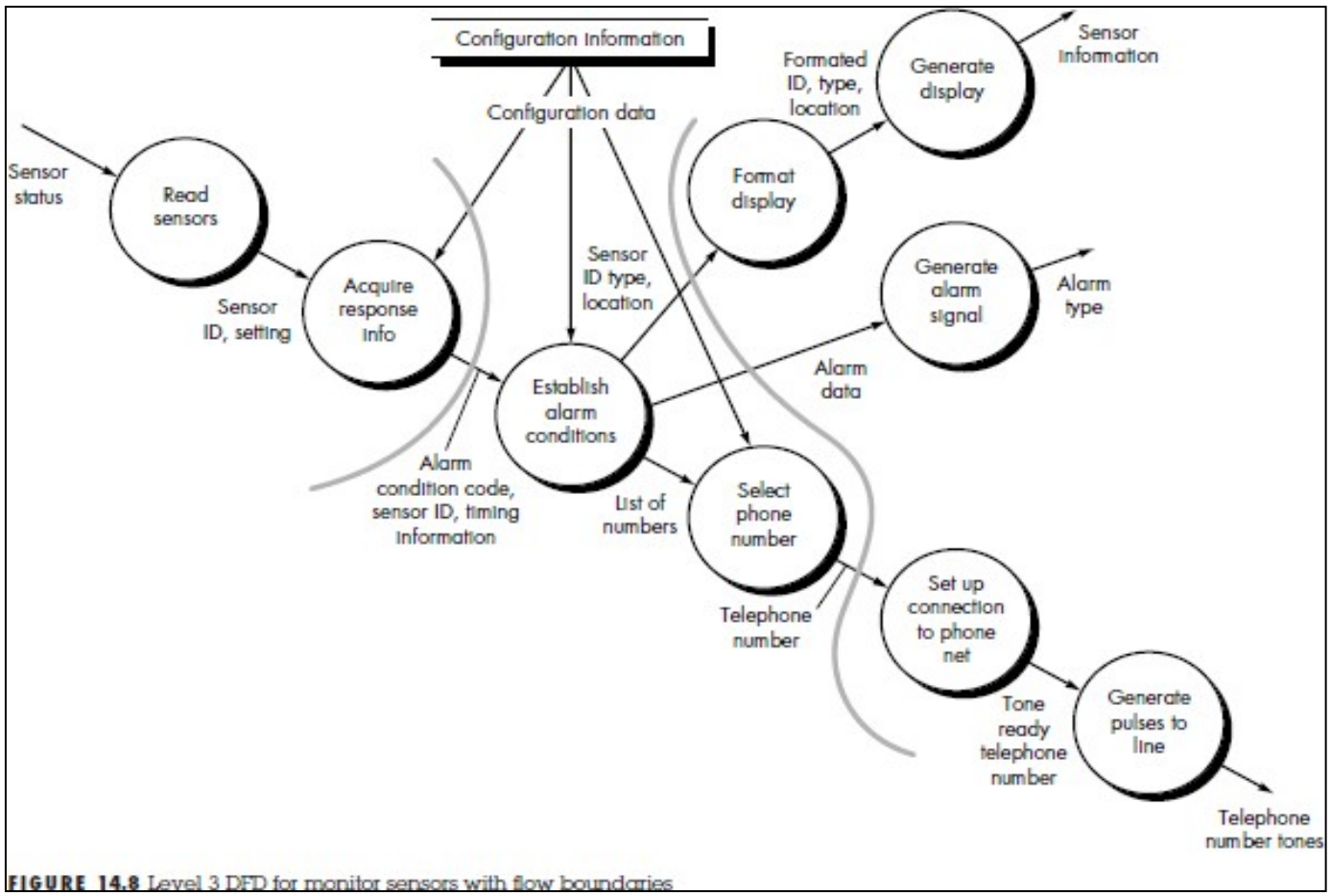


FIGURE 14.8 Level 3 DFD for monitor sensors with flow boundaries

Evaluating the DFD (Figure 14.8), we see data entering the software along one incoming path and exiting along three outgoing paths. Therefore, an overall transform characteristic will be assumed form information flow.

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

Flow boundaries for the example are illustrated as shaded curves running vertically through the flow in Figure 14.8. The transforms (bubbles) that constitute the transform center lie within the two shaded boundaries that run from top to bottom in the figure. An argument can be made to read just a boundary (e.g, any separating read sensors and acquire response info could be proposed). The emphasis in this design step should be on selecting reasonable boundaries, rather than lengthy iteration on placement of divisions.

Step 5. Perform "first-level factoring."

Factoring results in a program structure in which top-level modules perform decision making and low-level modules perform most input, computation, and output work. Middle-level modules perform some control and do moderate amounts of work.

A main controller (called monitor sensors executive) resides at the top of the program structure and coordinates the following subordinate control functions:

• An incoming information processing controller, called sensor input controller, coordinates receipt of all incoming data.

• A transform flow controller, called alarm conditions controller, supervises all operations on data in internalized form (e.g., a module that

invokes various data transformation procedures).

• An outgoing information processing controller, called alarm output controller, coordinates production of output information.

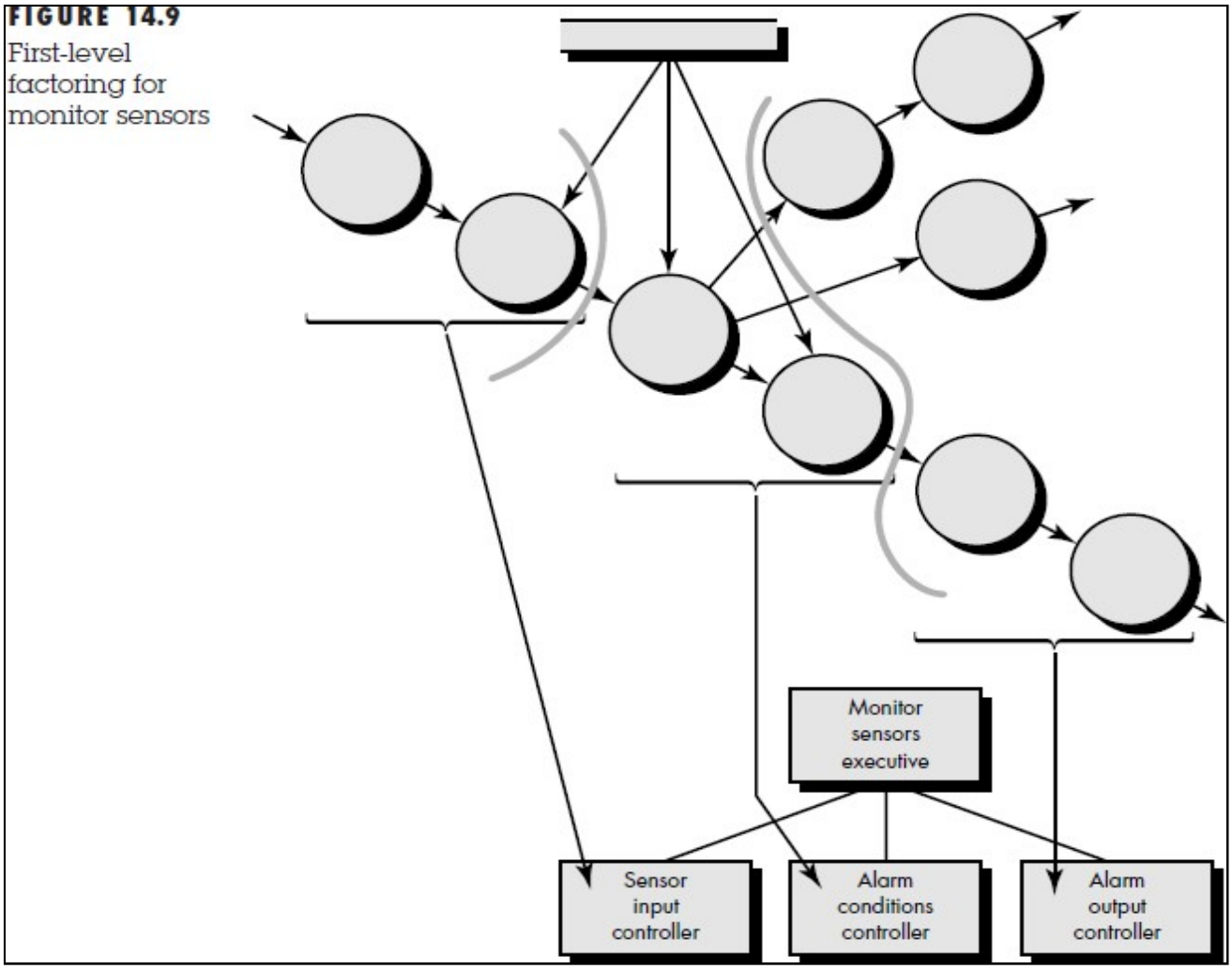


FIGURE 14.9
First-level factoring for monitor sensors

Step 6. Perform "second-level factoring."

Second-level factoring is accomplished by mapping individual transforms (bubbles) of a DFD into appropriate modules within the architecture.The general approach to second-level factoring for the SafeHome data flow is illustrated in Figure 14.10.)

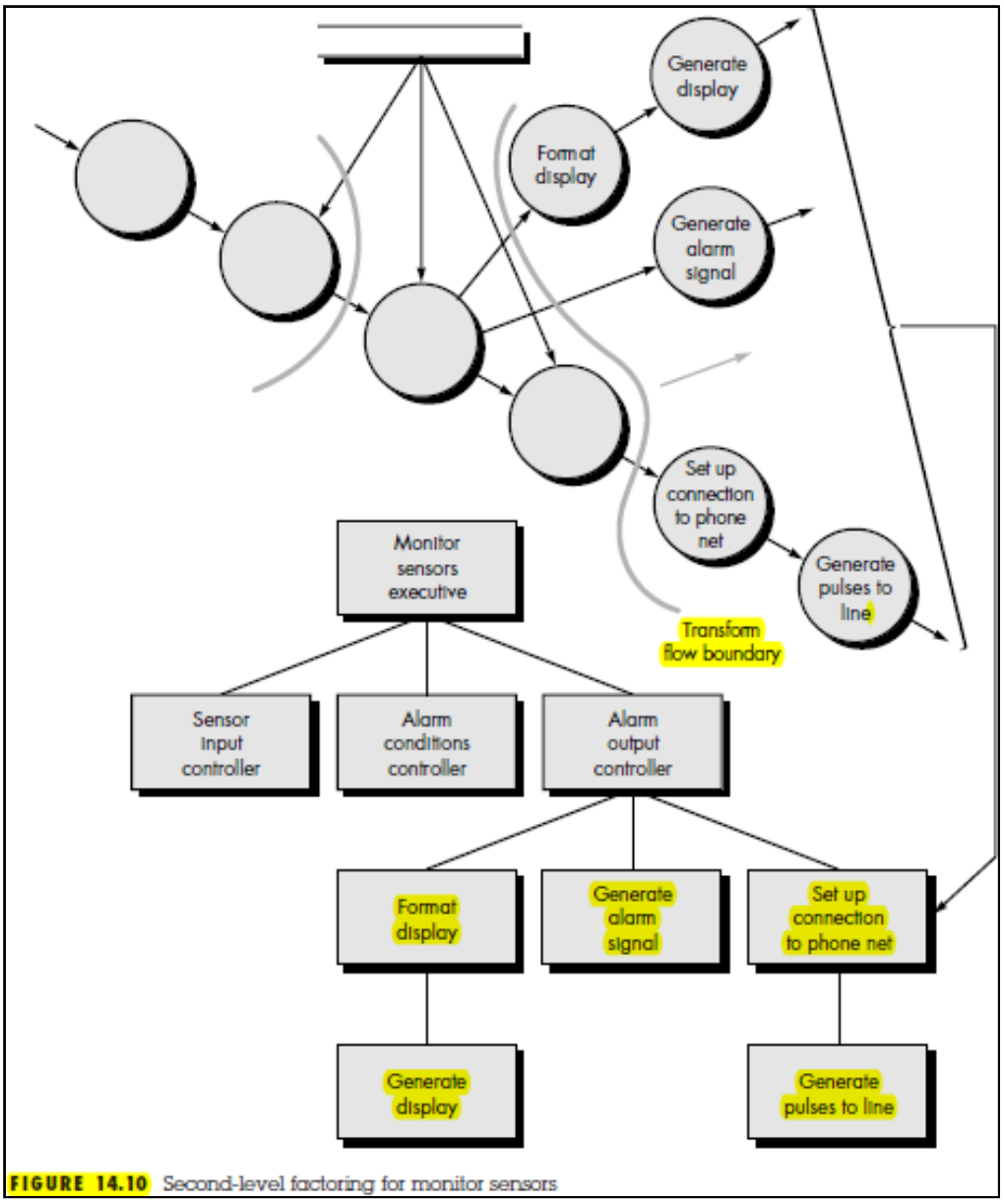


FIGURE 14.10 Second-level factoring for monitor sensors

Second-level factoring for incoming flow follows in the same manner. Factoring is again accomplished by moving outward from the transform center boundary on the incoming flow sideA completed first-iteration architecture is shown in Figure

The narrative describes

- Information that passes into and out of the module (an interface description).
- Information that is retained by a module, such as data stored in a local data structure.
- A procedural narrative that indicates major decision points and tasks.
- A brief discussion of restrictions and special features (e.g., file I/O, hardware dependent characteristics, special timing requirements).

The narrative serves as a first-generation Design Specification.

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

A first-iteration architecture can always be refined by applying concepts of module independence.

1. The incoming controller can be removed because it is unnecessary when a single incoming flow path is to be managed.

2. The substructure generated from the transform flow can be imploded into the module establish alarm conditions (which will now include the processing implied by select phone number). The transform controller will not be needed and the small decrease in cohesion is tolerable.

3. The refined program structure for monitor sensors is given by figure.

ACTUAL OUTPUT:

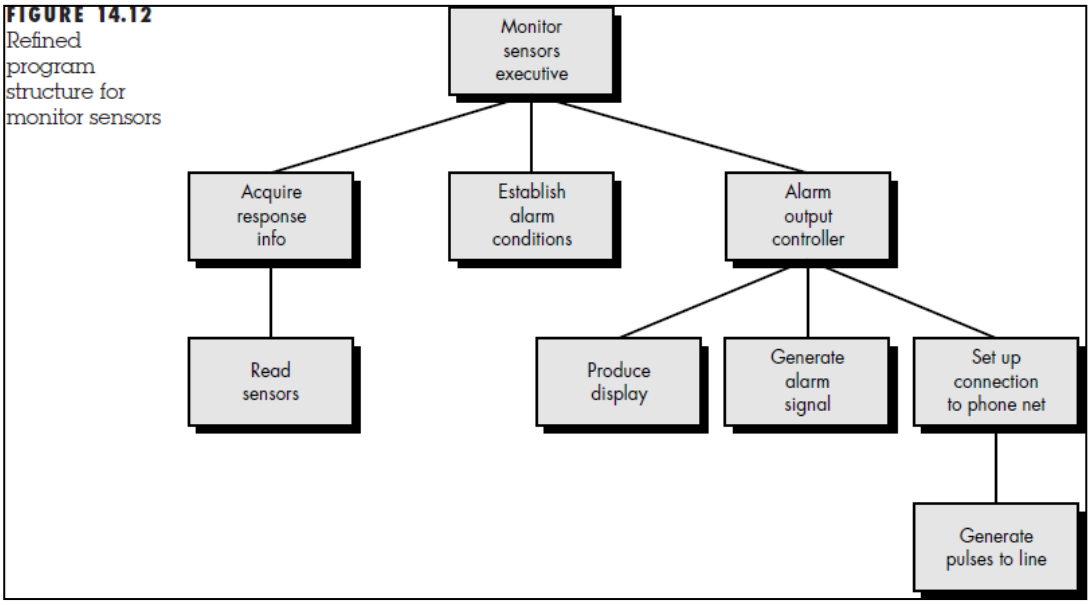


FIGURE 14.12
Refined program structure for monitor sensors

Implementation:

1. To create a new DFD select diagram> new from toolbar.
2. In new diagram widow, select DATA FLOW DIAGRAM and click NEXT.
3. We will now draw the first process. From diagram tool bar, drag process onto diagram, name new process system.
4. We use that of resource catalog to create a datastore from safe home software with bidirectional flow between them.
5. We selected data flow from resource catalog (->, <->).
6. For the diagram to look less crowded we make changes as follows:
    a. Right click on the diagram (context level DLD)
    b. Select connectors >curves.
    c. Now connectors in DFD will become curves.
    d. Move the shapes around so that diagram looks crowded.

Conclusion:

The DFD is converted into appropriate architectural style

Q16: Draw complete class diagram and object diagrams using Rational tools

AIM:-

Draw complete class diagram and object diagrams using Rational tools

PROCEDURE:-

Class Diagrams and Object Diagrams:-

Class Diagram: - A class diagram is a graph or structural diagram that shows what the objects in your system are consist of. It is a static structural diagram of classifier elements connected through different static relationships. A class diagram might be a combination of interfaces, classes, packages, and instances. The class diagram is the essential block of Object-Oriented programming. A class diagram might be used to make a non-programmer understand the system. It is significantly used for general conceptual modeling and data modeling.

Object Diagram: - The object diagram is also a structural diagram that shows how the objects in your system are reacting to their counterparts. The basics of object diagram are similar to that of the class diagram, but a class diagram defines classes and show how they relate to each other whereas, an object diagram focuses on objects and their relationship.

Purpose :-

Class Diagram: - The main purpose of class diagrams is to build a static view of an application. It is the only diagram that is widely used for construction, and it can be mapped with object-oriented languages. It is one of the most popular UML diagrams. Following are the purpose of class diagrams given below:

- It analyses and designs a static view of an application.
- It describes the major responsibilities of a system.
- It is a base for component and deployment diagrams.
- It incorporates forward and reverse engineering.

Object Diagram: - A class diagram represents an abstract model consisting of classes and their relationships. However, an object diagram represents an instance at a particular moment, which is concrete in nature.
It means the object diagram is closer to the actual system behavior. The purpose is to capture the static view of a system at a particular moment.
The purpose of the object diagram can be summarized as −

- Forward and reverse engineering.
- Object relationships of a system
- Static view of an interaction.
- Understand object behavior and their relationship from practical perspective

Uses of Class Diagram & Object Diagram:-

Class Diagram: -

- Describing the static view of the system.
- Showing the collaboration among the elements of the static view.
- Describing the functionalities performed by the system.
- Construction of software applications using object oriented languages.

Object Diagram:

- Making the prototype of a system.
- Reverse engineering.
- Modeling complex data structures.
- Understanding the system from practical perspective.

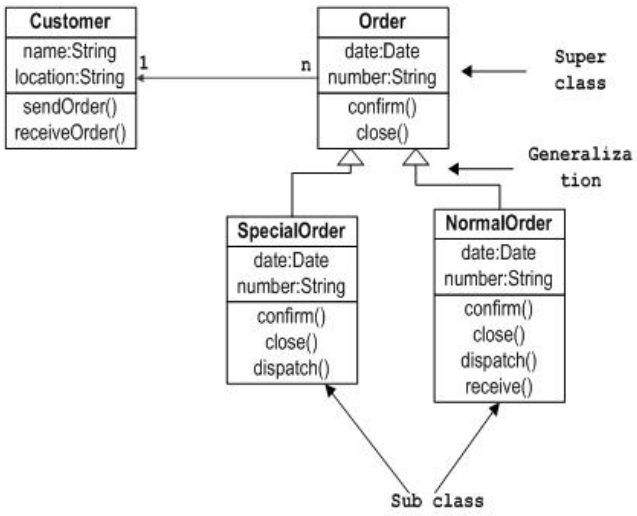
Summary:-

A Class Diagram will show what the Objects in your system consist of (members) and what they are capable of doing (methods).

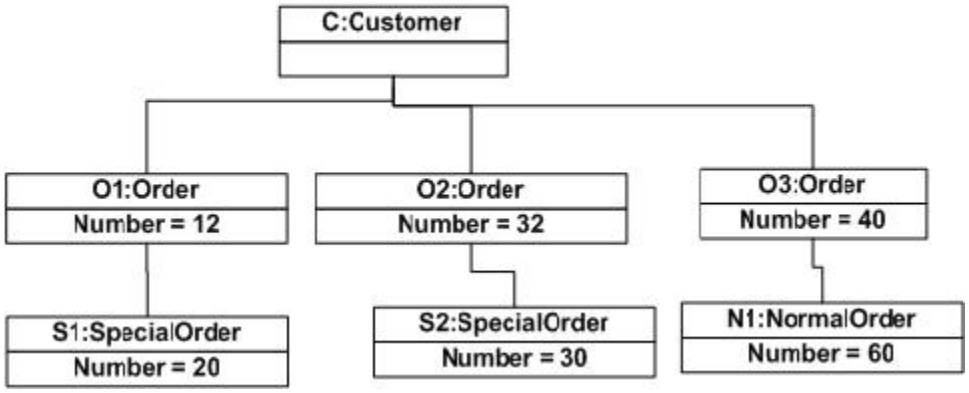
- In contrast, an Object Diagram will show how objects in your system are interacting with each other at some point in time, and what values those objects contain when the program is in this state

- The difference between them is that the class diagram represents class and its relationship with other classes while an object diagram represents objects and their relationship at a particular instance.

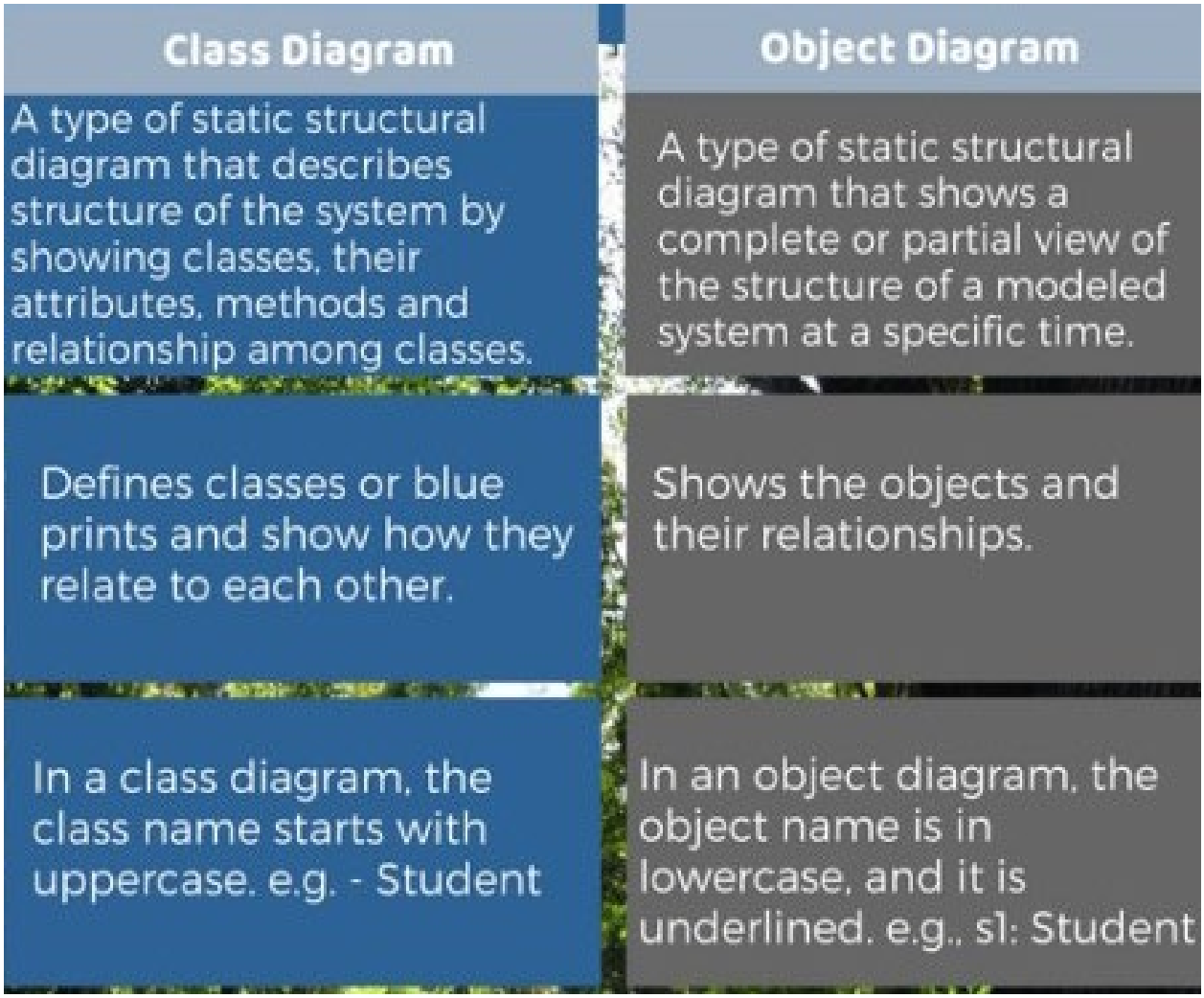
Sample Class Diagram



Object diagram of an order management system



Differences Between Class Diagram and Object Diagram:-

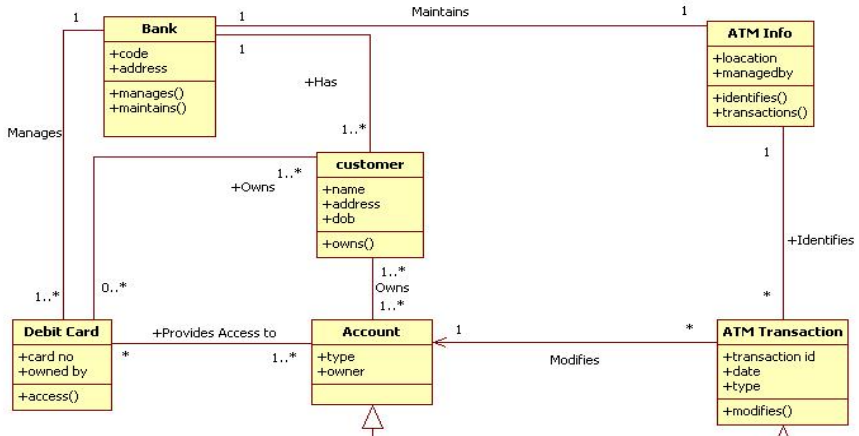


OUTPUT:-

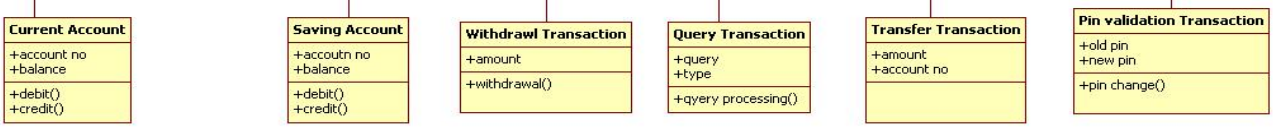
Example Of Class Diagram and Object Diagram:-

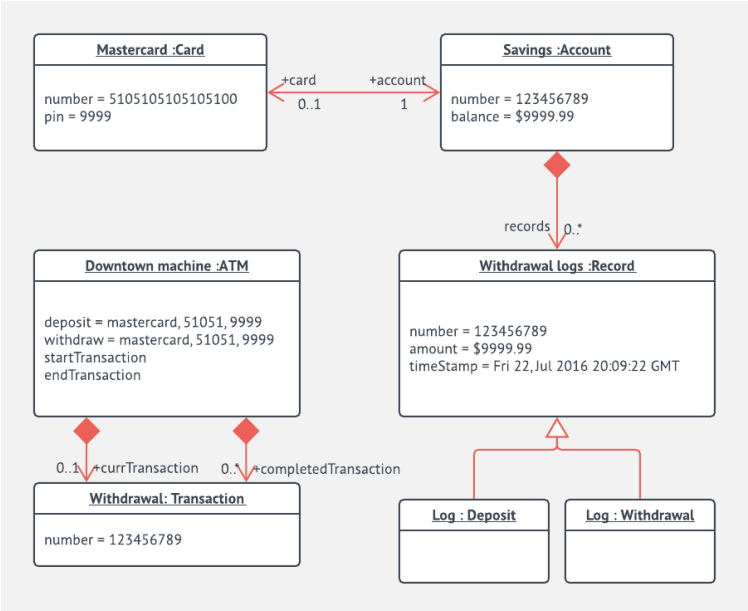
A test case for an ATM machine was built with two approaches for its class diagram and object diagram as seen below.

Class Diagram: -



Object Diagram:-





Q17) Define the design activities along with necessary artifacts using Design Document.

Aim: To define the Design Activities along with necessary Artifacts using
Design Document

DESIGN ACTIVITY:

Design activity includes collaboratively creating visual tools—rough sketches, mockups, models, and drawings—potentiating the communication, coordination, and integration between functions, and enhancing the production and circulation of knowledge within and between projects.

DESIGN DOCUMENT:

A design document is a complete high-level solution to the problem presented. It should be detailed enough that somebody who already understands the problem could go out and code the project without having to make any significant decisions. Further, if this somebody happens to be an experienced coder, they should be able to use the design document to code the solution in a few hours (not necessarily including debugging).

System Design Document:

Overview
The System Design Document describes about the system architecture, files and database design, human-machine interfaces design, detailed design and external interfaces.

1 INTRODUCTION

1 Purpose and Scope

This section provides a brief description of Design Activities purpose and scope.

2 Project Executive Summary

This section provides a description of the project from a management perspective and an overview of the framework within which the conceptual system design was prepared. If appropriate, include the information discussed in the subsequent sections in the summary.

3 System Overview

This section describes the system in narrative form using non-technical terms. It should provide a high-level system architecture diagram showing a subsystem breakout of the system, if applicable. .

4 Design Constraints

This section describes any constraints in the system design (reference any trade-off analyses conducted such, as resource use versus productivity, or conflicts with other systems) and includes any assumptions made by the project team in developing the system design.

5 Future Contingencies

This section describes any contingencies that might arise in the design of the system that may change the development direction. Possibilities include lack of interface agreements with outside agencies or unstable architectures at the time this document is produced.

6 Document Organization

This section describes the organization of the Systems Design Activities.

7 Points of Contact

This section provides the organization code and title of the key points of contact (and alternates if appropriate) for the information system development effort. These points of contact should include the Project Manager, System Proponent, User Organization, Quality Assurance (QA) Manager, Security Manager, and Configuration Manager, as appropriate.

8 Project References

This section provides a bibliography of key project references and

deliverables that have been produced before this point.

9 Glossary

Supply a glossary of all terms and abbreviations used in this document. If the glossary is several pages in length, it may be included as an appendix.

10 SYSTEM ARCHITECTURE

In this section, describe the system and/or subsystem(s) architecture for the project. References to external entities should be minimal, as they will be described in detail in Section 6, External Interfaces.

11 System Hardware Architecture

In this section, describe the overall system hardware and organization. Include a list of hardware components (with a brief description of each item) and diagrams showing the connectivity between the components. If appropriate, use subsections to address each subsystem.

12 System Software Architecture

In this section, describe the overall system software and organization. Include a list of software modules (this could include functions, subroutines, or classes), computer languages, and programming computer-aided software engineering tools (with a brief description of the function of each item). Use structured organization diagrams/object-oriented diagrams that show the various segmentation levels down to the lowest level. All features on the diagrams should have reference numbers and names. Include a narrative that expands on and enhances the understanding of the functional breakdown. If appropriate, use subsections to address each module.

Note: The diagrams should map to the FRD data flow diagrams, providing the physical process and data flow related to the FRD logical process and data flow.

13 Internal Communications Architecture

In this section, describe the overall communications within the system; for example, LANs, buses, etc. Include the communications architecture(s) being implemented, such as X.25, Token Ring, etc. Provide a diagram depicting the communications path(s) between the system and subsystem modules. If appropriate, use subsections to address each architecture being employed.

Note: The diagrams should map to the FRD context diagrams.

14 FILE AND DATABASE DESIGN

Interact with the Database Administrator (DBA) when preparing this section. The section should reveal the final design of all database management system (DBMS) files and the non-DBMS files associated with the system under development. Additional information may add as required for the particular project. Provide a comprehensive data dictionary showing data element name, type, length, source, validation rules, maintenance (create, read, update, delete (CRUD) capability), data stores, outputs, aliases, and description. Can be included as an appendix.

15Database Management System Files

This section reveals the final design of the DBMS files and includes the following information, as appropriate (refer to the data dictionary):

- Refined logical model; provide normalized table layouts, entity relationship diagrams, and other logical design information
- A physical description of the DBMS schemas, sub-schemas, records, sets, tables, storage page sizes, etc.
- Access methods (such as indexed, via set, sequential, random access, sorted pointer array, etc.)
- Estimate of the DBMS file size or volume of data within the file, and data pages, including overhead resulting from access methods and free space
- Definition of the update frequency of the database tables, views, files, areas, records, sets, and data pages; estimate the number of transactions if the database is an online transaction-based system

16Non-Database Management System Files

In this section, provide the detailed description of all non-DBMS files and include a narrative description of the usage of each file—including if the file is used for input, output, or both; if this file is a temporary file; an indication of which modules read and write the file, etc.; and file structures (refer to the data dictionary). As appropriate, the file structure information should:

- Identify record structures, record keys or indexes, and reference data elements within the records
- Define record length (fixed or maximum variable length) and blocking factors
- Define file access method—for example, index sequential, virtual sequential, random access, etc.

- Estimate the file size or volume of data within the file, including overhead resulting from file access methods
- Define the update frequency of the file; if the file is part of an online transaction-based system, provide the estimated number of transactions per unit time, and the statistical mean, mode, and distribution of those transactions

17 DETAILED DESIGN

This section provides the information needed for a system development team to actually build and integrate the hardware components, code and integrates the software modules, and interconnects the hardware and software segments into a functional product. Additionally, this section addresses the detailed procedures for combining separate COTS packages into a single system. Every detailed requirement should map back to the FRD, and the mapping should be presented in an update to the RTM and include the RTM as an appendix to this design document.

18Hardware Detailed Design

A hardware component is the lowest level of design granularity in the system. Depending on the design requirements, there may be one or more components per system. This section should provide enough detailed information about individual component requirements to correctly build and/or procure all the hardware for the system (or integrate COTS items).

If there are many components or if the component documentation is extensive, place it in an appendix or reference a separate document. Industry-standard component specification practices should be followed. Include the following information in the detailed component designs (as applicable):

- Power input requirements for each component
- Memory and/or storage space requirements
- Processor requirements (speed and functionality)
- Graphical representation depicting the number of hardware items (for example, monitors, printers, servers, I/O devices), and the relative positioning of the components to each other
- User interfaces (buttons, toggle switches, etc.)

19Software Detailed Design

A software module is the lowest level of design granularity in the system. Depending on the software development approach, there may be one or more modules per system. This section should provide enough detailed information

about logic and data necessary to completely write source code for all modules in the system.

If there are many modules or if the module documentation is extensive, place it in an appendix or reference a separate document. Add additional diagrams and information, if necessary, to describe each module, its functionality, and its hierarchy. Industry-standard module specification practices should be followed. Include the following information in the detailed module designs:

- Data elements, record structures, and file structures associated with module input and output
- Graphical representation of the module processing, logic, flow of control, and algorithms, using an accepted diagramming approach (for example, structure charts, action diagrams, flowcharts, etc.)
- Data entry and data output graphics; define or reference associated data elements; if the project is large and complex or if the detailed module designs will be incorporated into a separate document, then it may be appropriate to repeat the screen information in this section
- Report layout

20 Internal Communications Detailed Design

If the system includes more than one component there may be a requirement for internal communications to exchange information, provide commands, or support input/output functions. This section should provide enough detailed information about the communication requirements to correctly build and/or procure the communications components for the system. Include the following information in the detailed designs (as appropriate):

- The number of servers and clients to be included on each area network
- Specifications for bus timing requirements and bus control
- Format(s) for data being exchanged between components
- Graphical representation of the connectivity between components, showing the direction of data flow (if applicable), and approximate distances between components; information should provide enough detail to support the procurement of hardware to complete the installation at a given location
- LAN topology

21 EXTERNAL INTERFACES

External systems are any systems that are not within the scope of the system under development, regardless whether the other systems are managed by the State or another agency. In this section, describe the electronic interface(s)

between this system and each of the other systems and/or subsystem(s), emphasizing the point of view of the system being developed.

22 Interface Architecture

In this section, describe the interface(s) between the system being developed and other systems; for example, batch transfers, queries, etc. Include the interface architecture(s) being implemented, such as wide area networks, gateways, etc. Provide a diagram depicting the communications path(s) between this system and each of the other systems, which should map to the context diagrams in Section 1.2.1. If appropriate, use subsections to address each interface being implemented.

23 Interface Detailed Design

For each system that provides information exchange with the system under development, there is a requirement for rules governing the interface. This section should provide enough detailed information about the interface requirements to correctly format, transmit, and/or receive data across the interface. Include the following information in the detailed design for each interface (as appropriate):

- The data format requirements; if there is a need to reformat data before they are transmitted or after incoming data is received, tools and/or methods for the reformat process should be defined
- Format(s) for error reports exchanged between the systems; should address the disposition of error reports; for example, retained in a file, sent to a printer, flag/alarm sent to the operator, etc.
- Graphical representation of the connectivity between systems, showing the direction of data flow
- Query and response descriptions

24 Component design

During detailed design, these logical components are refined and their interactions are modeled to verify the validity of their structural composition.

Component design refers to modeling the internal structure and behavior of components—which includes the internal structure of both logical and physical components – identified during the software architecture phase.

25 constuction design

Construction design is not a new concept. Many other authors have proposed it as an important design activity. For example, McConnell specifies five levels of software design; one of them, being at the lowest level, deals with internal routine design. Similarly, one identifies a form of low-level design that fills the gap between detailed design and programming and deals with issues such as operation specification, including operation name, parameter types and return types among others. Other authors have highlighted the importance of designing codes at low-levels, during construction.

Construction design is the last design activity—typically conducted during the construction phase – required to support the system's quality attributes, such as performance, maintainability and testability

26 Human-computer interface design

The human-computer interface (HCI) design activity is where general principles are applied to optimize the interface between humans and computers.

Visual designs have a major role on the success or failure of software systems. Systems that meet functional requirements but that are not usable cannot succeed. The HCI design activity can be executed in parallel to the software architecture or detailed design activities.

The major concerns of the HCI designs may include the evaluation and use of modes, navigation, visual designs, response time, feedback and design modalities, such as forms and menu-driven.

HCI designs directly influence the quality of any system and are essential to understanding and addressing the factors that affect the overall usability of the system. Many design principles and evaluation techniques exist to successfully design user interfaces.

OUTPUT:

REAL TIME EXAMPLE

Design Document for Automated Telling Machine (ATM):

1 Introduction

This Software is been developed for customers of bank which allows customers to complete basic transactions without the aid of a branch representative or teller. This ATM software system also allows customers easier access to their accounts. Anyone with a debit or credit card will be able to access most ATMs. Using a machine operated by your bank is usually free, but accessing funds through a unit owned by a competing bank will usually incur a small fee

1.1 Purpose of the Document

This System Design Document (SDD) defines the requirements for the Automatic Teller Machine (ATM) System. The primary audience of this document includes, but is not limited to, project leaders, the designers and developers of the system and existing/potential end users. This document follows the conventions of IEEE standard to help ensure that the requirements were well-formed

This document is been developed for the user to read, understand and use it but cannot be modified or sale without the permission of SAP.

1.2 Scope

The objective of an ATM machine is to provide anytime or Automated Banking services to the bank customers without the customer having to make a trip to the bank. Some of the services an ATM provides are:

- Cash withdrawal
- Accept deposits
- Issue Balance Statements

- Pre-paid Mobile Charge
- Money Transfer

1.3 Relationship

This is design document that is having direct relation with Software requirement Specification

1.4 Methodology and tools:

This document having following methodology: UML, Class Diagram, State machine Diagram
Tools: Visual Paradigm , Visio studio

1.5 Policies:

Here are some policies for ATM software system:
According to branch authorization guidelines, banks are given permission to open, close and shift all categories of branches including ATMs at a time, normally once a year, and the sanction is valid for one year from the date of announcement.
A group would be constituted to review the existing framework of the branch authorization policy for providing greater elasticity and enhanced penetration.
Required views and comments on the draft circular from banks authorized ATM network operators, non-bank entities and members of public in this regard up to the 6th of next month.
ATMs will be in the nature of White Label Automated Teller Machines (WLAs) and will offer ATM services to customers of all banks.

1.6 Key Stakeholders:

Following are the stakeholder of this document:
Owner of this document: Abbas Arshad
Document approval: Sir Aqeel

2 Design Overview

2.1 Background Information

An automated teller machine (ATM) is a computerized telecommunications device that provides the customers of financial institutions with access to financial transactions in a public space without the need for a human clerk or bank teller. On most modern ATMs, the customer is indentified by inserting a plastic ATM card with a magnetic tape or a plastic smartcard with a chip, that contains a unique card number and some security information, such as a expiration date or CVC (CVV). Security is provided by the customer entering a personal identification number (PIN)

2.2 System Evolution Description

Automated teller machines (ATMs) have gained prominence as a delivery channel for banking transaction. Banks have been deploying ATMs to increase their reach. While ATMs facilitate a variety of banking transaction for customers, their main utility has been for cash withdrawal and balance enquiry.

2.3 Technology Forecast

Forecasting a new service diffusion process is critical in designing marketing strategies and analyzing the costs and benefits for service providers. It is very difficult, however, in cases that data are not available.

2.4 Application Overview

Using an ATM, customers can access their bank accounts to make cash withdrawals (or cresit card cash advances) and check their account balances.
The functions of the system are:

- Login
- Get balance information
- Withdraw Cash
- Transfer funds

The hardware, software and technology used should have following specifications:

- Ability to read the ATM card
- Ability to count currency notes

- Touch screen for convenience
- Keypad (in case touchpad fails)
- Continuous power supply
- Ability to connect to bank's network
- Ability to take input from user
- Ability to validate user

2.5 Constraints

- Login
    - Validate bank card
        - Validate for card expiration date
        - Validate that the card's expiration date is later than today's date
        - If the card is expired, prompt error message "Card is Expired"
    - Validate for Stolen card or lost
        - Validate that the card is not reported lost or stolen
        - If the card is lost, prompt error message, "Card has been reported lost"
        - If the card is stolen, prompt error message, " Card has been reported Stolen"
    - Validate for disabled card
        - Validate that the card is not disabled
        - If card is disabled, prompt error message, "Card has been disabled as of <expiration date>"
    - Validate for locked account
        - Validate that the account is not locked
        - If the account Is locked, prompt error message "Account is Locked"
    - Validate PIN
        - Validate the password is not blank
        - If the PIN is blank, prompt error message "Please Provide PIN"

        - Validate that the password entered matches the password on file
        - If the password does not match, prompt error message “Password is incorrect”
- Lock Account
    - If number of consecutive unsuccessful logins exceeds three attempts, lock account
- Maintain consecutive unsuccessful login counter
- Increment login counter
- For every consecutive login attempt, increment logic counter by 1
- Reset login counter to 0 after the login is successful
- Get balance information
- Withdraw cash
- Transfer funds

2.6 Risks

- Personal safety is a risk with ATMs in empty parking lots, poorly lighted places, or behind bank buildings. In these places, there's always the risk someone will be lurking nearby to rob you of your money when you are finished.
- There is also the risk that more clever thieves will swipe your PIN code. This risk increases with poorly maintained or unfamiliar ATMs.
- Consumer protection agencies warn that criminals attach mirrors to ATMs to capture PIN numbers or tamper with the machines in other ways to get your cash or information.
- You will also be charged fees for using ATMs not connected to your bank or those in foreign countries.

2.7 Assumptions and Dependencies

- Hardware never fails
- ATM cashing is impenetrable
- Limited number of transactions per day (Sufficient paper for receipts)
- Limited amount of money withdrawn per day (Sufficient money)

3 Scope of the Work

The software supports a computerized banking network called ATM system. The network Enables customers to complete simple bank account services via automatic teller machines (ATM) that may be located off premise and that need to be owned and operated by the customer's bank. The ATM identifies a customer by cash and password. It collects information about a simple account transaction (e.g. deposit, withdrawal, transfer, bill payment etc), communicates the transaction information to the customer's bank, and dispenses cash to the customer. The banks provide their own software for their own computers. The ATM system software requires appropriate record keeping and security provisions. The software must handle concurrent accesses to the same account correctly.

3.1 System functions

The ATM network does not work independently. It works together with the banks' computers and the software run by the network's banks. The actors of the system are:

- User
- ATM Machine
- Bank

The software should support a computerized banking network. Each bank provides its own computer to maintain its own accounts and process transactions against them. Automatic teller machines communicate with the banks, computers. An ATM accepts a cash card and interacts with the user and the user communicates with the bank computer to carry out the transactions. ATM dispenses cash and prints receipts. The system requires appropriate record keeping and security provisions. The system must handle concurrent access to the same account correctly. The banks will provide their own software for their own computers. The cost of shasred system will be apportioned to the banks according to the number of customers

3.2 User characteristics

Open to all authorized users characteristics and is dependent upon functionality:

Customers are simply members if the public with no special training. Bank security personnel need have no special education or experiences. Maintainers must be experienced network administrators, to be able to connect new ATMs to the network.

3.3 Hardware Interfaces

The hardware should have the following Specifications:

- Ability to read the ATM card
- Ability to count the currency notes
- Touch screen for convenience
- Keypad (in case touchpad fails)
- Continuous power supply
- Ability to connect to bank's network
- Ability to take input from user
- Ability to validate user

3.4 Communication Interfaces

The Commucation interfaces should have the following Specifications

- Easy to understand by the user
- Ability to carry out functions easily
- Continuous Communcation without delays and errors

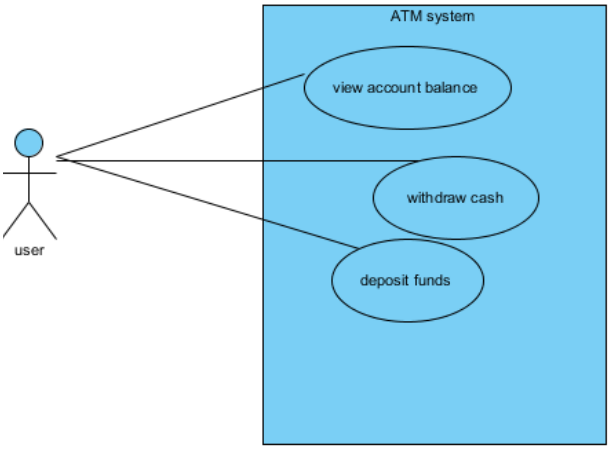
3.5 Software Interfaces

The software interfaces are specific to target banking software systems. At present two known banking systems will participate in the ATM network

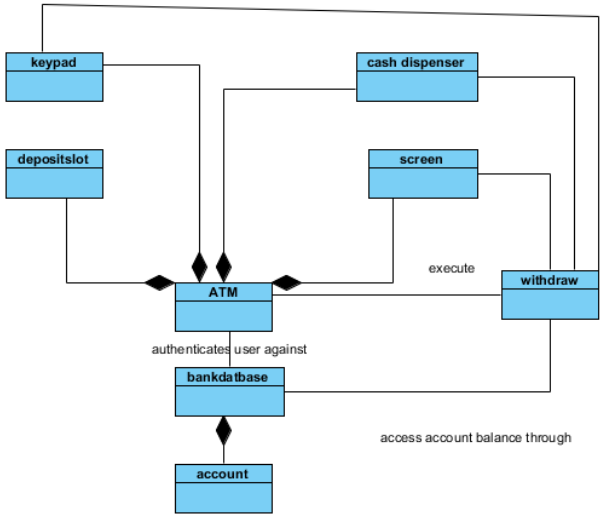
- State Bank
- Overseas Bank

4 Detailed Design

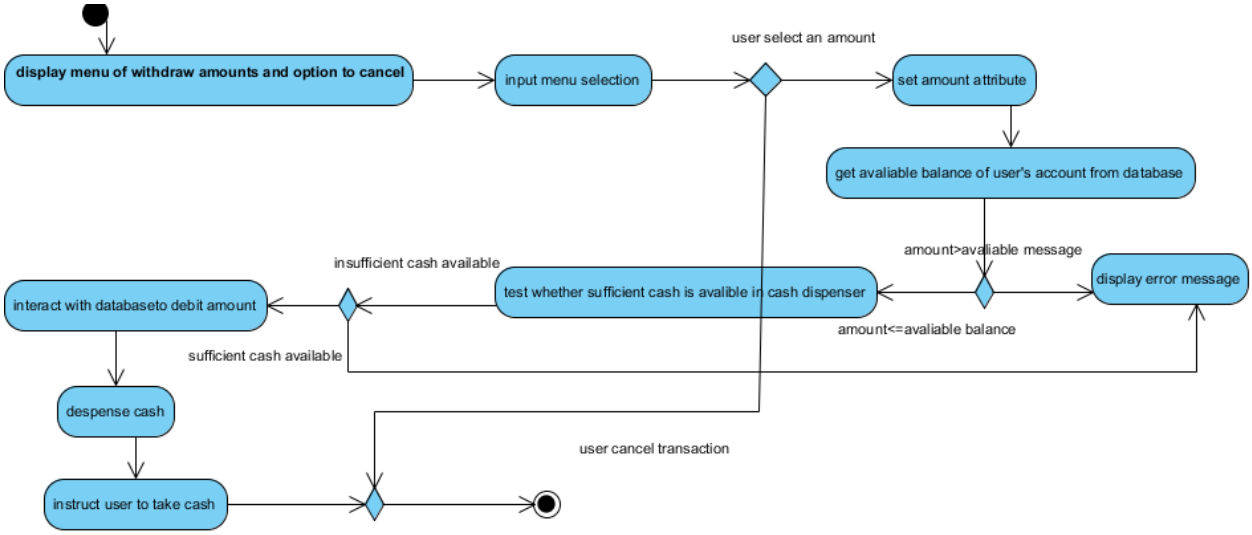
Use case diagram for the ATM system from the User's perspective.



Class diagram for the ATM system model.



Activity diagram for a withdrawal transaction



5 Interface Design

5.1 Interface Description

The customer's user interface should be intuitive, such that 99.9% of all new ATM users are able to complete their banking transactions without any assistance.
The user interface of the automated teller machine contains:

- A screen that displays messages to the user
- A keypad that receives numeric input from the user
- A cash dispenser that dispenses cash to the user and
- A deposit slot that receives deposit envelopes from the user.

A real ATM typically contains a device that reads a user's account number from an ATM card, whereas this ATM asks the user to type the account number on the keypad. A real ATM also usually prints a receipt at the end of a session, but all output from this ATM appears on the screen.

6 User Interface Design

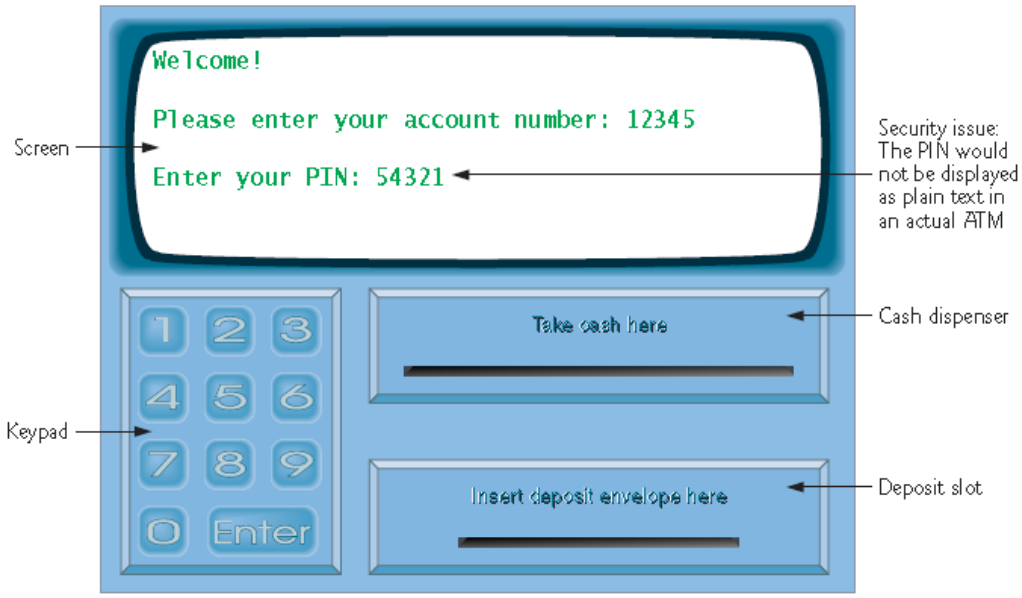
6.1 User Interfaces

User interface should be user friendly and understandable. Here are few images of user interface designs

6.1.1 Screen Images

Upon first approaching the ATM (assuming no one is currently using it), the user should experience the following sequence of events (shown in Fig.):

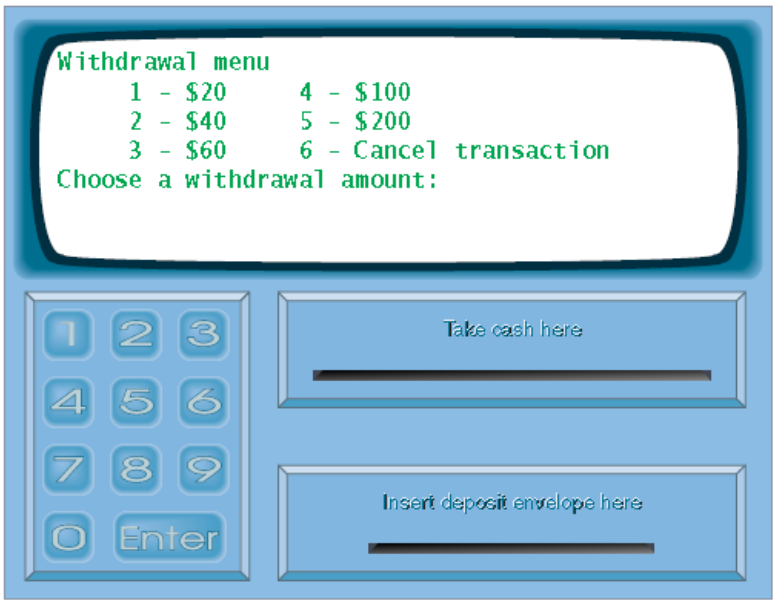
1. The screen displays Welcome! and prompts the user to enter an account number.
2. The user enters a five-digit account number using the keypad.
3. The screen prompts the user to enter the PIN (personal identification number) associated with the specified account number.
4. The user enters a five-digit PIN using the keypad.1
5. If the user enters a valid account number and the correct PIN for that account, the screen displays the main menu (Fig. 12.2). If the user enters an invalid account number or an incorrect PIN, the screen displays an appropriate message, then the ATM returns to *Step 1* to restart the authentication process



After the ATM authenticates the user, the main menu (Fig. 12.2) should contain a numbered option for each of the three types of transactions: balance inquiry (option 1), Withdrawal (option 2) and deposit (option 3). It also should contain an option to allow the user to exit the system (option 4). The user then chooses either to perform a transaction (by entering 1, 2 or 3) or to exit the system (by entering 4).

If the user enters 1 to make a balance inquiry, the screen displays the user's account balance. To do so, the ATM must retrieve the balance from the bank's database. The following steps describe what occurs when the user enters 2 to make a withdrawal:

1. The screen displays a menu (Fig. 12.3) containing standard withdrawal amounts: $20 (option 1), $40 (option 2), $60 (option 3), $100 (option 4) and $200 (option 5). The menu also contains an option to allow the user to cancel the transaction (option 6).



2. The user enters a menu selection using the keypad.

3. If the withdrawal amount chosen is greater than the user's account balance, the screen displays a message stating this and telling the user to choose a smaller amount. The ATM then returns to *Step 1*. If the withdrawal amount chosen is less

than or equal to the user's account balance (i.e., an acceptable amount), the ATM proceeds to *Step 4*. If the user chooses to cancel the transaction (option 6), the ATM displays the main menu and waits for user input.
4. If the cash dispenser contains enough cash, the ATM proceeds to *Step 5*. Otherwise, the screen displays a message indicating the problem and telling the user to choose a smaller withdrawal amount. The ATM then returns to *Step 1*.
5. The ATM debits the withdrawal amount from the user's account in the bank's database (i.e., subtracts the withdrawal amount from the user's account balance).
6. The cash dispenser dispenses the desired amount of money to the user.
7. The screen displays a message reminding the user to take the money.
The following steps describe the actions that occur when the user enters 3 (when viewing the main menu of Fig. 12.2) to make a deposit:
1. The screen prompts the user to enter a deposit amount or type 0 (zero) to cancel.
2. The user enters a deposit amount or 0 using the keypad. [*Note:* The keypad does not contain a decimal point or a dollar sign, so the user cannot type a real dollar amount (e.g., $27.25). Instead, the user must enter a deposit amount as a number of cents (e.g., 2725). The ATM then divides this number by 100 to obtain a number representing a dollar amount (e.g., 2725 ÷ 100 = 27.25).]
3. If the user specifies a deposit amount, the ATM proceeds to *Step 4*. If the user chooses to cancel the transaction (by entering 0), the ATM displays the main menu and waits for user input.
4. The screen displays a message telling the user to insert a deposit envelope.
5. If the deposit slot receives a deposit envelope within two minutes, the ATM credits the deposit amount to the user's account in the bank's database (i.e., adds the deposit amount to the user's account balance). [*Note:* This money is *not* immediately available for withdrawal. The bank first must physically verify the amount of cash in the deposit envelope, and any checks in the envelope must clear (i.e. money must be transferred from the check writer's account to the check recipient's account). When either of these events occurs, the bank appropriately updates the user's balance stored in its database. This occurs independently of the ATM system.] If the deposit slot does not receive a deposit envelope within this time period, the screen displays a message that the system has canceled the transaction due to inactivity. The ATM then displays the main menu and waits for user input.
After the system successfully executes a transaction, it should return to the main menu so that the user can perform additional transactions. If the user exits the system, the screen should display a thank you message, and then display the welcome message for the next user.

6.2 Components Available

Components that are available are:

- A screen that displays messages to the user
- A keypad that receives numeric input from the user
- A cash dispenser that dispenses cash to the user and
- A deposit slot that receives deposit envelopes from the user.

6.3 User Interface Development Description

The User interfaces are designed in such a way that it is easy for the user to understand and carry out tasks required without any other assistance. The interfaces are shown and described in the chapter above.

18. Reverse Engineer any object-oriented code to an appropriate class and object diagrams.

Aim:- Reverse Engineer any object-oriented code to an appropriate class and object diagrams.

Appropriate definition:

A Reverse engineer works backward from a chosen file to determine the design and technology used by the creator.
Reverse Engineering generally belongs to software maintenance
Why do we need reverse engineering?
Recovery of lost information
- Providing proper system documentation

Assisting with maintenance
- Identification of side effects and anomalies

Class Diagrams:

A class diagram is a diagram that shows a set of classes, interfaces, and collaborations and their relationships. Graphically, a class diagram is a collection of vertices and arcs

Common Properties:

A class diagram is just a special kind of diagram and shares the same common properties as do all other diagrams name and graphical content that are a projection into a model. What distinguishes a class diagram from other kinds of diagrams is its particular content.

Common uses of class diagrams:

You use class diagrams to model the static design view of a system. This view primarily supports the functional requirements of a system the services the system should provide to its end users. When you model the static design view of a system, you'll typically use class diagrams in one of three ways

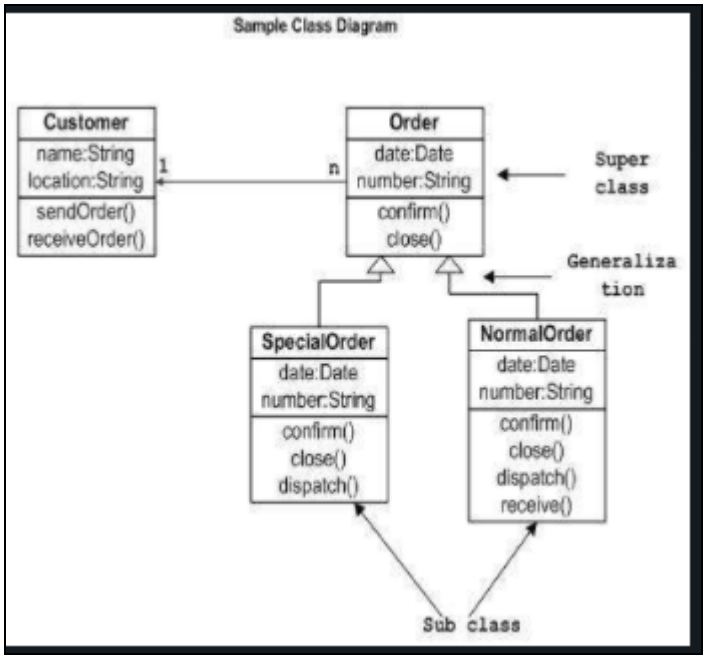
Reverse engineering on class diagrams:

- Reverse engineering is the process of transforming code into a model through a mapping from a specific implementation language.
- Reverse engineering results in a flood of information, some of which is at a lower level of detail than you'll need to build useful models.
- At the same time, reverse engineering is incomplete. There is a loss of information when forward engineering models into code, and so you can't completely recreate a model from code unless your tools encode information in the source comments that goes

beyond the semantics of the implementation language. To reverse engineer a class diagram,

- Identify the rules for mapping from your implementation language or languages of choice. This is something you'll want to do for your project or your organization as a whole.
- Using a tool, point to the code you'd like to reverse engineer. Use your tool to generate a new model or modify an existing one that was previously forward engineered.
- It is unreasonable to expect to reverse engineer a single concise model from a large body of code. You need to select portion of the code and build the model from the bottom.
- Using your tool, create a class diagram by querying the model. For example, you might start with one or more classes, then expand the diagram by following specific relationships or other neighboring classes.
- Expose or hide details of the contents of this class diagram as necessary to communicate your intent.
- Manually add design information to the model to express the intent of the design that is missing or hidden in the code.

CLASS DIAGRAMS OUTPUT:



Object Diagrams:

- Object diagrams model the instances of things contained in class diagrams. An object diagram shows a set of objects and their relationships at a point in time
- An object diagram is a diagram that shows a set of objects and their relationships at a point in time. Graphically, an object diagram is a collection of vertices and arcs.

Common Properties :

- An object diagram is a special kind of diagram and shares the same common properties as all other diagrams that is, a name and graphical contents that are a projection into a model. What distinguishes an object diagram from all other kinds of diagrams is its particular content.

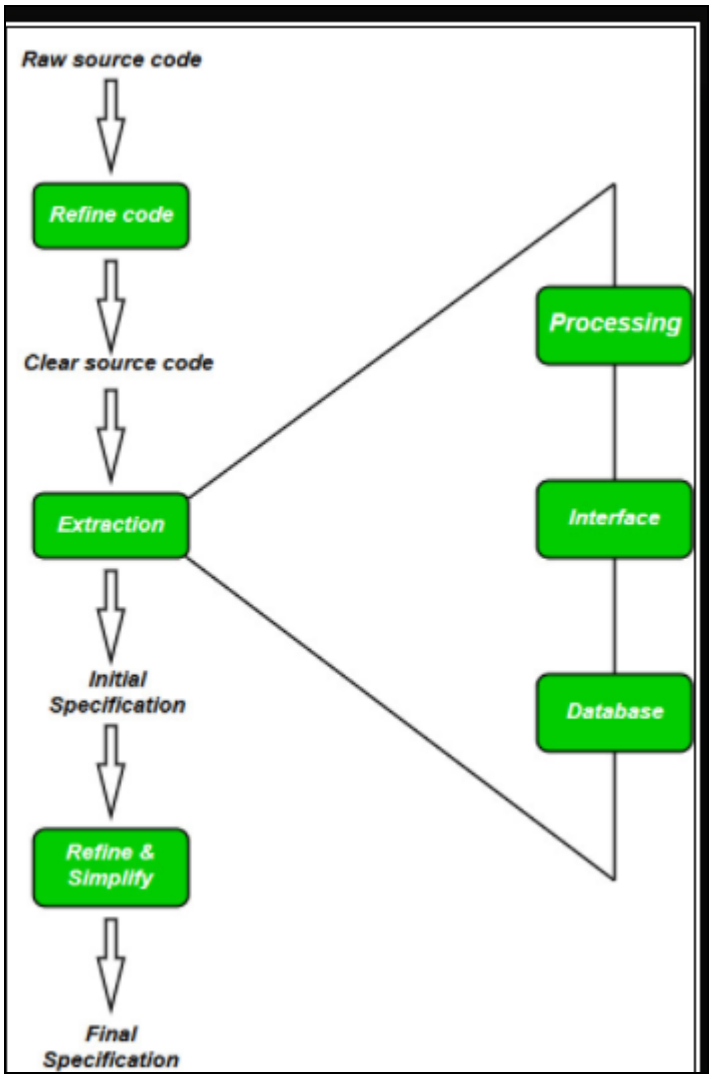
common Uses on object diagrams:

- You use object diagrams to model the static design view or static process view of a system just as you do with class diagrams, but from the perspective of real or prototypical instances.
- This view primarily supports the functional requirements of a systemthat is, the services the system should provide to its end users. Object diagrams let you model static data structures.

Reverse engineering on object diagrams:

- Reverse engineering (the creation of a model from code) an object diagram can be useful. In fact, while you are debugging your system, this is something that you or your tools will do all the time
- For example, if you are chasing down a dangling link, you'll want to literally or mentally draw an object diagram of the affected objects to see where, at a given moment in time, an object's state or its relationship to other objects is broken. To reverse engineer an object diagram
- Chose the target you want to reverse engineer. Typically, you'll set your context inside an operation or relative to an instance of one particular class
- Using a tool or simply walking through a scenario, stop execution at a certain moment in time.
- Identify the set of interesting objects that collaborate in that context and render them in an object diagram
- As necessary to understand their semantics, expose these object's states
- As necessary to understand their semantics, identify the links that exist among these objects

OBJECT DIAGRAMS OUTPUT:



Conclusion:-

- Reverse engineering is a new research area among software maintenance
- Reverse engineering includes activites of understanding the system and recovery information from system
- Program understanding is the most important subset of Reverse engineering

19. Test a piece of code which executes a specific functionality in the code to be tested and asserts a certain behavior or state using J unit

AIM:-

Test a piece of code which executes a specific functionality in the code to be tested and asserts a certain behavior or state using the JUnit.

Testing in Software Engineering:

Software testing activities can be categorized the following ways-

- Functional activities that ensure if a product meets technical requirements, all the features work as intended;
- Non-functional activities that validate performance, ease of use, security, and user satisfaction with the project.

Unit Testing

Unit Testing is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected.
Unit Testing is done during the development (coding phase) of an application by the developers.
Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object.

Here, are the key reasons to perform unit testing in software engineering:

1. Unit tests help to fix bugs early in the development cycle and save costs.
2. It helps the developers to understand the testing code base and enables them to make changes quickly
3. Good unit tests serve as project documentation
4. Unit tests help with code reuse. Migrate both your code and your tests to your new project. Tweak the code until the tests run again.

Steps to perform unit testing

In order to do Unit Testing, developers write a section of code to test a specific function in software application. Developers generally use Unit Test framework to develop automated test cases for unit testing.

Unit Testing is of two types

- Manual
- Automated

Under the automated approach-

- A developer writes a section of code in the application just to test the function. They would later comment out and finally remove the test code when the application is deployed.
- A coder generally uses a UnitTest Framework to develop automated test cases. Using an automation framework, the developer codes criteria into the test to verify the correctness of the code.
- During execution of the test cases, the framework logs failing test cases. Many frameworks will also automatically flag and report, in summary, these failed test cases. Depending on the severity of a failure, the framework may halt subsequent testing.

Workflow of Unit Testing

1) Create Test Cases
2) Review/Rework
3) Baseline
4) Execute Test Cases

Unit Testing Techniques

Black Box Testing - Using which the user interface, input and output are tested.
White Box Testing - used to test each one of those functions behaviour is tested.
Gray Box Testing - Used to execute tests, risks and assessment methods.

Introduction to JUnit

JUnit is an open source Unit Testing Framework for JAVA. It is useful for Java Developers to write and run repeatable tests. As the name implies, it is used for Unit Testing of a small chunk of code.

JUnit does not require a server for testing web applications, which makes the testing process fast. JUnit framework also allows quick and easy generation of test cases and test data.

The org.Junit package consists of many interfaces and classes for JUnit Testing such as Test, Assert, After, Before, etc.

Testing Code using Junit

In the JunitTestCaseExample.java class, we created the code which we want to test.

In the TestJunitTestCaseExample.java, we write the test cases for the JunitTestCaseEample.java class. We create an object of the JunitTestCaseExample.java class, and by using its object, we will test all its methods.

We create the TestRunner.java class to execute the test cases. It contains the main() method in which we run the TestJunitTestCaseExample.java class using the runClasses() method

Program :

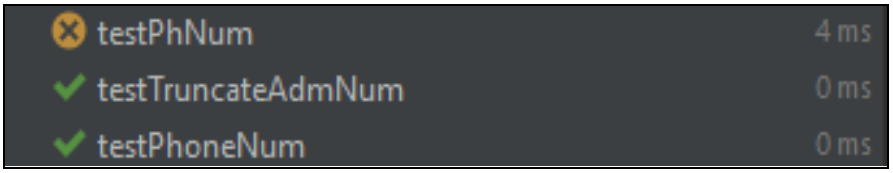
```
public class StringHelper {

         public String truncateAdmNum(String str) {
            if (str.length() <= 10)
               return str.replaceAll("19001A0", "");

      String first7Chars = str.substring(0, 7);
      String stringMinusFirst7Chars = str.substring(7);

      return first7Chars.replaceAll("19001A0", "") + stringMinusFirst7Chars;
   }

   public boolean PhoneNum(String str) {

      if (str.length() == 10)
         return true;
      else
         return false;
   }
}
```

Test cases :

```
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;

public class StringHelperTest {
   StringHelper helper;
   @Before
   public void before(){
      helper = new StringHelper();
   }
   @Test
   public void testTruncateAdmNum() {
      assertEquals("599", helper.truncateAdmNum("19001A0599"));
   }
   @Test
   public void testPhoneNum() {
      assertTrue(helper.PhoneNum("9000202069"));
   }
```

```
    @Test
    public void testPhNum() {
        assertFalse(helper.PhoneNum("9000202069"));
    }
}
```

Output:



20. Test the percentage of code to be tested by unit test using any code coverage tools.

AIM:-

Test the percentage of code to be tested by unit test using any code coverage tools.

Unit Testing:

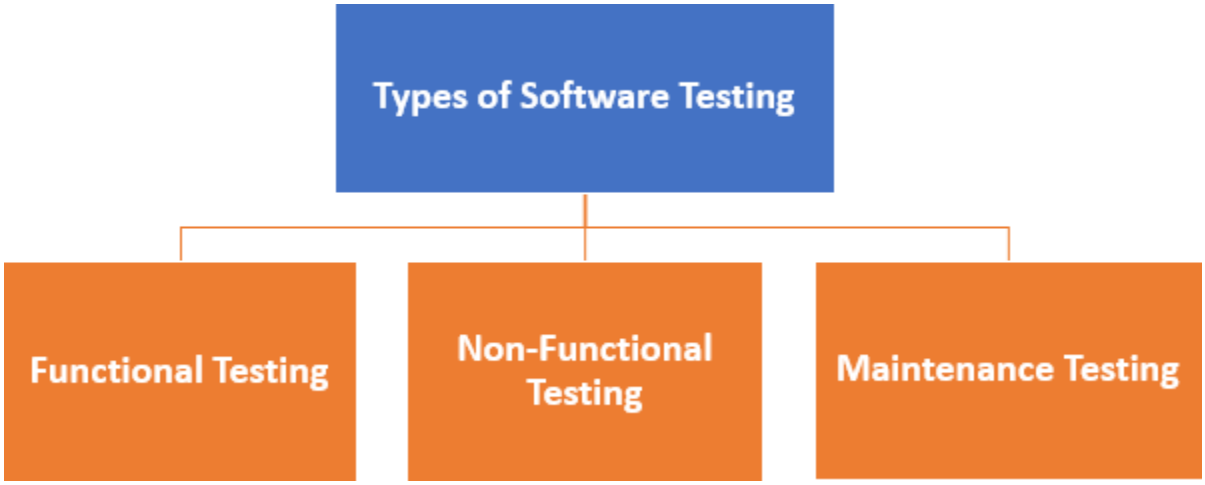
This software testing basic approach is followed by the programmer to test the unit of the program. It helps developers to know whether the individual unit of the code is working properly or not.

Testing in Software Engineering

Types of Software Testing

Typically Testing is classified into three categories.

- Functional Testing
- Non-Functional Testing or Performance Testing
- Maintenance (Regression and Maintenance)



Functionalactivitiesthatensureifaproduct meetstechnicalrequirements,allthefeatures work asintended;

Non-functional activities that validate performance, ease ofuse, security, and user satisfactionwiththeproject.

Testing Category	Types of Testing
Functional Testing	• Unit Testing • Integration Testing • Smoke • UAT(User Acceptance Testing • Localization • Globalization

	• Interoperability • So on
Non-FunctionalTesting	• Performance • Endurance • Load • Volume • Scalability • Usability • So on
Maintenance	• Regression • Maintenance

Unit Testing

UnitTestingisatypeofsoftwaretestingwhere individualunitsorcomponentsofasoftwareare tested. Thepurposeistovalidatethateachunit ofthesoftwarecodeperformsasexpected.

UnitTestingisdoneduringthedevelopment (coding phase) of an application by the developers.

UnitTestsisolateasectionofcodeandverifyits correctness. A unit may be an individual function, method, procedure, module, or object.

Here,arethekeyreasonstoperformunit testinginsoftwareengineering:

Unittestshelptofixbugsearlyinthe developmentcycleandsavecosts.

Ithelpsthedeveloperstounderstandthe

testingcodebaseandenablesthemtomake changes quickly

Goodunittestsserveasprojectdocumentation

Unittestshelpwithcodereuse.Migrateboth yourcodeandyourteststoyournewproject. Tweak thecodeuntilthetestsrunagain.

WhySoftwareTestingisImportant?

Software Testing is Important because if thereareanybugsorerrorsinthe software,it canbeidentifiedearlyandcanbesolved beforedeliveryofthe softwareproduct.

Properly tested software product ensures reliability, security and high performance whichfurtherresultsintimesaving,cost effectivenessandcustomersatisfaction.

What are the benefits of Software Testing?

- **Cost-Effective:** Itisoneoftheimportant advantagesofsoftwaretesting.Testing anyIT projectontimehelpsyoutosave yourmoneyforthe longterm.Incaseifthe bugs caught in the earlier stage of software testing, it costs less to fix.
- **Security:** Itisthemostvulnerableand sensitive benefitofsoftwaretesting. People are looking for trusted products. It helpsinremovingrisksand problems earlier.
- **Product quality:** It is an essential requirement of any software product. Testing ensuresaqualityproductis deliveredto customers.
- **Customer Satisfaction:** Themainaimof any productistogivesatisfactiontotheir

customers.UI/UXTestingensuresthebest userexperience.

ProgramTesting

Program Testing in software testing is a methodofexecutinganactual software programwiththeaimoftestingprogram behaviorandfindingerrors. Thesoftware programisexecutedwithtestcasedatato analysetheprogram behavioror responseto thetestdata.Agoodprogramtestingisone whichhashigh chancesoffindingbugs.

Program :

```
public class StringHelper {

        public String truncateAdmNum(String str){ if (str.length() <= 10)
                returnstr.replaceAll("19001A0582","");


    Stringfirst7Chars=str.substring(0,10); StringstringMinusFirst7Chars=
str.substring(10);

    return first7Chars.replaceAll("19001A0582", "") +stringMinusFirst7Chars;
  }
```

```
    publicbooleanPhoneNum(Stringstr){

        if(str.length()==10) returntrue;
        else
           return false;
    }
}
```

Test cases:

```
importstaticorg.junit.Assert.*; importorg.junit.Before;
```

```
import org.junit.Test;


publicclassStringHelperTest{ StringHelper helper;
   @Before
   public voidbefore(){
         helper = new StringHelper();
   }
   @Test
   public void testTruncateAdmNum(){ assertEquals("399",
helper.truncateAdmNum("19001A0399"));
   }
   @Test
   public void testPhoneNum() {
```

```
        assertTrue(helper.PhoneNum("8082572287"
));
   }
   @Test
   public void testPhNum() { assertFalse(helper.PhoneNum("8082572286
"));
   }
}
```

Output:

Tested TruncateAdmNum Testedphone

number

21.Define an appropriate metrics for atleast 3 Quality Attributes for any Software Application of your interest.

Software Quality: Quality software refers to a software, which is reasonably bug and defect free, delivered in time and within the specified budget, meets the requirements or expectations and is maintainable.

Metrics: Metrics are measures of quantitative assessment commonly used for assessing, comparing, and tracking performance or production.

- Software quality metrics classified into 3 types:

1. Product metrics – Describes the characteristics of the product such as size, complexity, design features, performance, and quality level.

2. Process metrics – these characteristics can be used to improve the development and maintenance activities of the software.

3. Project metrics – this metrics describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.

Each quality attribute have some metrics for to measure software.

- **Quality Attributes**: Software quality attributes (QA) characterize the usefulness of the software in a given environment. Software Quality Attributes are features that facilitate the measurement of performance of a software product by Software Testing

professionals. High scores in **Software Quality Attributes** enable software architects to guarantee that a software application will perform as the specifications provided by the client.

- Software quality attributes such as:
  - Availability
  - Interoperability
  - Correctness
  - Reliability
  - Learnability
  - Robustness
  - Maintainability
  - Readability
  - Extensibility
  - Testability
  - Efficiency
  - Portability

Software Quality Attributes Concern

Rank	Quality attribute	Concern
1	Modifiability	Reducing coupling
2	Performance	Latency
3	Interoperability	Upgrading and integrating with other System components
4	Modifiability	Designing for portability
5	Usability	Ease of operation
6	Availability	Fault detection
7	Interoperability	Ease of interfacing with other systems
8	Modifiability	Designing for extensibility
9	Availability	Fault recovery
10	Performance	Resource management
11	Deployability	Minimizing build, test, and release duration

THE THREE QUALITY ATTRIBUTES ARE:

AVAILABILITY:

Availability defines the proportion of time that the system is functional and working. It can be measured as a percentage of the total system downtime over a predefined period. System errors, infrastructure problems, malicious attacks, and system load will affect overall system availability.

Metrics:

The equation below is used to calculate the availability

A=MTBF/MTBF+MTTR

A: Availability
MTBF: Mean Time Between Failure
MTTR: Mean Time to Repair

Strategies to Achieve the quality metrics:

- Using cloud technologies based on different region as data center redundancy pattern.
- Using redundant infrastructure components.
- Availability status page, which our users can subscribe to receive availability status reports and incidents.

USABILITY:

- Usability is the degree to which a software can be used by specified consumers to achieve quantified objectives with effectiveness, efficiency, and satisfaction in a quantified context of use, for example, easy to localize and globalize, providing good access for disabled users, and resulting in a good overall user experience

Metrics:

- **Completion rate**: The rate of users who completed a task successfully.
- **Number of errors**: The number of errors the user faced while doing a task on the system.
- **Duration time**: The duration, the user spent to finish the task
- **Training:** The number of days required to train new users
- User satisfaction surveys

Strategies to achieve the quality metrics:

- The solution will use auto-completion algorithm to help the user fill the data easily. Any forms will take more than 5 minutes will be split into more than one form.
- The navigation buttons will be designed according to usability standards.
- Using behavioral analytics techniques to analyze user behavior.

PERFOMANCE:

The degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage. It refers to responsiveness, either the time required to respond to specific events or the number of events processed in a given interval of time.

Metrics:

- **Latency**: Latency is the time taken to respond to any event
- **Throughput:** It is the number of events that take place within a given amount of time
- **Capacity:** It is the total demands can be handled by the system while continuing to meet the latency and throughput requirements
- **Modes:** Having different resources or demand over time and system adaptation to that.

Strategies to achieve the quality metrics:

- There are different strategies to enhance system performance.
- For example: caching techniques, data partitioning, load balancing, scalability, First Things First (FTF)

- **Application for solving one-dimensional and two-dimensional Schrodinger equations [42].**

The application is written in the C programming language. It has 1074 LOC organized into 48 logical units.

- This application was chosen due to their ability to prove that the developers who adopt software engineering practices while creating applications produce better results for all measured quality metrics.

- The data required for the practical evaluation of the model is obtained by analysing the software applications' source codes and the software applications performance analysis.

1. Evaluation of the maintainability attribute: Software maintainability depends on the following attributes: **a) analysability**, b) **changeability,** and c) **testability**

▶ The analysability attribute is evaluated by using the following metrics:

- Volume—represented in MY. FPs are calculated when LOC is divided by 128. Volume in MYs is obtained when the total FPs are divided by the number of FPs monthly per programmer, multiplied by 12.
- Code duplication — expressed as a percentage of LOC. To detect the percentage of duplicated source code blocks, the tool PMD [47] was used.
- Size of software units — average number of code lines per logical software unit.

Table 1: Result for an analysability attribute

Volume (MY)	*Duplicated code (%)*	*Average unit size (LOC)*	*Analysability*
0.08	*5.59*	*16.48*	*7.38*

▶ Software changeability depends on the percentage of duplicate code and on the average complexity of logical software units. The calculations according to the risk categories for cyclomatic complexity of software units.

Table 2. Percentage of source code according to the risk categories for cyclomatic complexity.

Module complexity	A1 (%)
1-10	27.65
11-20	25.51
21-50	4.19
>50	16.29

To calculate the software changeability, the average complexity per unit and percentage of duplicated code are used.

Table 3: Results for changeability attribute.

Duplicated code (%)	Average unit complexity	Changeability
5.59	19.02	12.30

▶ The testability attribute depends on the average unit complexity and the average unit size.

Table 4. Results for the testability attribute

Average unit size	Average unit complexity	Testability

16.48	19.02	17.75

2. Evaluation of the portability attribute:

The metrics used to calculate portability: modularity, programming language, system architecture, and system complexity.

- **System modularity** depends on the number of software units (modules) and the average unit size (LOC. The number of modules positively affects the modularity of the system, and the average size of modules can negatively affect the system.

Table 5. Results for the modularity attribute.

Average unit size	Number of units	Modularity
16.48	48	24.03

- Applications use a programming language that is platform-independent; the system architecture value is one. The programming language metric value is two for application. The complexity of the system is calculated according to Eq. (7). The average unit cyclomatic complexity and the average unit coupling inversely affect the complexity of the system.

Table 6. Results for the system complexity attribute

Average unit complexity	Average unit coupling	System complexity
19.02	0.82	9.92

3. Evaluation of the reliability attribute: Reliability is calculated using the following metrics: MTTF, MBTF, MTTR, and AVAIL. Only, the MMTR metric inversely affects the reliability. The application was measured for an execution time of 173 min.

Table 7. Results for the reliability attribute

MTTF (minutes)	MTTR (minutes)	MTBF (minutes)	AVAIL
86	6	92	0.94

22. Define a complete call graph for any C or C++ Code...

CALL GRAPH:

- A **call graph** (also known as a call multigraph) is a **control-flow graph**, which represents calling relationships between **subroutines** in a **computer program**.
- Each node represents a **procedure** and each edge (f, g) indicates that procedure f calls procedure g.
- Thus, a cycle in the graph indicates recursive procedure calls.

Call graphs can be defined to represent varying degrees of precision. A more precise call graph more precisely approximates the behavior of the real program, at the cost of taking longer to compute and more memory to store. The most precise call graph is fully **context-sensitive**, which means that for each procedure, the graph contains a separate node for each **call stack** that procedure can be activated with.

Call graphs can be:

1. Dynamic
2. Static

1.DYNAMIC:

- A Dynamic call graph is a record of an execution of the program, for example as output by a profiler.

- ❖ Thus, a dynamic call graph can be exact, but only describes one run of the program. Dynamic methods actually run the program to determine the call graph.

2.STATIC:

- ❖ A Static call graph is a call graph intended to represent every possible run of the program. The exact static call graph is an undecidable problem, so static call graph algorithms are generally over approximations.
- ❖ That is, every call relationship that occurs is represented in the graph, and possibly also some call relationships that would never occur in actual runs of the program.

Call graphs can be used in different ways:

- One simple application of call graphs is finding procedures that are never called.
- Call graphs can act as documentation for humans to understand programs.
- They can also serve as basis for further analyses, such as an analysis that tracks the flow of values between procedures, or change impact prediction.
- Call graphs can also be used to detect anomalies of program execution or code injection attacks.

We have some tools to represent any code in the form of call graphs:

Graphiz (short for Graph visualization software) is a package of opensource tool. Graph Visualization is a way of representing structural information as diagrams of abstract graphs and networks.

.

Egypt is a simple tool for creating call graphs of C programs. Egypt neither analyzes source code nor lays out graphs. Instead, it leaves the source code analysis to

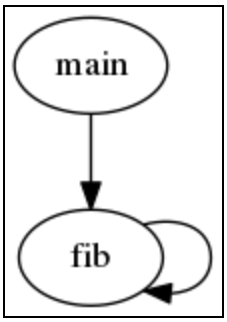
GCC and the graph layout to **Graphviz**, both of which are better at their respective jobs than egypt itself could ever hope to be.

Example 1 –

Representing a call graph for a simple c program to find n^{th} Fibonacci number.

```
#include<stdio.h>
int fib (int n) {
if(n<=2) return 1;
return fib(n-1) +fib(n-2);
}

int main () {
int n=10, x=0;
x=fib (10);
printf ("%d", x);
return 0;
}
```

We will get the following call graph.


main
fib