Experiment-I: Software Requirements Specification

(Saloon management system)

1. **Problem Statement:** To collect software requirements and create software requirement specification.

SOFTWARE REQUIREMENTS: Ms Word.

1. Introduction

Purpose Basic Description of Problem

The Saloon management system is an important part in our day-day life. There are many advantages of a management system in a saloon.

Scope

The purpose of this specification is to document requirements for a system to manage the Saloon. The specification identifies what such a system is required to do. The specification is written in a format conforming to the IEEE Standard 830- 1984. Subject to approval, the specification will complete the Requirements phase and will be followed by detailed design, implementation, and testing.

Definitions, Acronyms, and Abbreviations

CID	Customer ID Number
SMS	Saloon Management System
GUI	Graphical User Interface
SID	Staff ID

References

Saloon Management System is a complete Saloon management softwareand system with features like: waiting time, Token number, Auto Billing and a lot more.

Overview

This specification includes a brief product perspective and a summary of the functions the software will provide. User characteristics are discussed and any general constraints or assumptions and dependencies are listed.

Requirement statements are categorized as functional requirements, performance requirements, non-functional requirements, or design constraints. Functional requirements are further categorized in terms of customer enrolment, staff enrolment, data updation of customer and staff. Non-functional requirements are further categorized in terms of security, maintainability, and scalability.

2. General Description

Product Perspective

The SMS is designed to help the Saloon administrator to handle customer and staff information. The current design goal is to build an internal system to achieve the functionality outlined in this specification.

Product Functions

The SMS will allow the user to manage information about Customer and staff. Customer enrolments provide customer sign up procedure. Same is for the staff. The SMS will also support the automatic backup and protection of data.

User Characteristics

There are three different types of users for the SMS system:

Type 1.	The Owner, who is most familiar with the problem domain. He holds some knowledge of basic computer operations and applications such as Microsoft Word, Excel, and PowerPoint. He manages the whole shop and has good practice. The Owner is the only person who will have the authority.
Type 2 .	Staffs, who are in charge of taking orders. Of the three user types, the Staffs have least computer knowledge. There is a requirement for staff to perform some data entry. Staff will need to make regular scheduling changes, and so the interface should be easy to use.
Type 3 .	Customer, who use this whole system but cannot modify any information.

Based on the above categorizations, in order to meet user's needs the following precautions should be taken:

- the interface should be designed with the computer novice in mind
- data entry masks should recognize and correct improperly entered data
- for deleting or revising a record the system should ask the users for confirmation
- error messages should be provided.
- the interface should be easy to understand.

General Constraints

The following constraints will limit the developer's options for designing the system:

implementation is required within 1 week.

3. Specific Requirements

Functional Requirements

R l. Customer enrolment Subsystem

The customer management subsystem requirements are concerned with the management of customer information.

R 1.1 The SMS shall allow the user type 1 and 2 to updatecustomer personal information (name, address, phone number, email, , …).

R 1.2 The SMS shall store all customer history information.

R 1.3 The SMS shall allow the user type 1 and 2 to add new Customer into the system.

R 1.3.1 The SMS shall assign a unique ID and passwords to new customer.

R l .4 The SMS shall allow all user types to retrieve customer personal information by customer ID.

R l.4.1 The SMS shall allow all user types to retrieve customer personal information.

R1.5 Customer account deletion

R1.5.1 The SMS shall allow user types 1 and 2 to delete customer account by entering the customer id and password.

R1.5.2 Ask for confirmation on deletion.

R2. Staff Management Subsystem

The staff management subsystemrequirements are concerned with the management of staff information. They specify how staff information can be managed and manipulated.

R 2.1 The SMS shall allow the user type 1 or 2 to add new staffs to the system.

R2.1.1 Staff details are added by taking the following information like name, address, phone number, email, …).

R2.1.2 On submission, a unique staff id and password is generated to the new staff.

R 2.2 The SMS shall allow the user type 1 or 2 to remove staffs from the system.

R2.2.1 Staff id and password is required.

R2.2.2 Confirm again before submission

R 2.3 The SMS shall allow the user type 1 or 2 to update staff information.

R 2.3.1 To update staff personal information (name, address, phone number, email, …).

R 2.4 The SMS shall update staff available information when a staff is assigned to a customer.

Performance Requirements

R4The SMS shall respond to user's retrieving information quickly. The waiting time for any retrieve operation must be under 2 seconds.

3.2 Non-functional Requirements

R5. Security.

The security requirements are concerned with security and privacy issues. All customer information is required by law to be kept private

R 5.1 The SMS shall support different user access privileges.

R 5.2 The SMS shall protect customer information.

R6. Maintainability

The maintainability requirements are concerned with the maintenance issues of the system.
R 6.1 The maintenance time of SMS shall be done regularly.
R 6.2 System down time for maintenance should be less than 6 hours per quarter of a year.

R7. Scalability

The scalability requirements are concerned with the scalable issues of the system.

Design Constraints

R 8. The SMS shall have a graphical user interface.

R 9. The SMS must run in Administrative office.

R 10. The SMS must be written in Java.

Experiment-II: Data Flow Diagrams

1. **Problem Statement:** To design dataflow diagram for software requirements

SOFTWARE REQUIREMENTS: Ms Word.

Data flow diagram (DFD)

A picture is worth a thousand words. A Data Flow Diagram (DFD) is traditional visual representation of the information flows within a system. A neat and clear DFD can depict a good amount of the system requirements graphically. It can be manual, automated, or combination of both.

It shows how information enters and leaves the system, what changes the information and where information is stored. The purpose of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communications tool between a systems analyst and any person who plays a part in the system that acts as the starting point for redesigning a system.

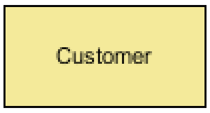
It is usually beginning with a context diagram as the level 0 of DFD diagram, a simple representation of the whole system. To elaborate further from that, we drill down to a level 1 diagram with lower level functions decomposed from the major functions of the system. This could continue to evolve to become a level 2 diagram when further analysis is required. Progression to level 3, 4 and so on is possible but anything beyond level 3 is not very common. Please bear in mind that the level of details for decomposing particular function really depending on the complexity that function.

Diagram Notations

Now we'd like to briefly introduce to you a few diagram notations which you'll see in the tutorial below.

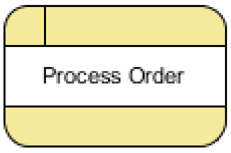
External Entity

An external entity can represent a human, system or subsystem. It is where certain data comes from or goes to. It is external to the system we study, in terms of the business process. For this reason, people used to draw external entities on the edge of a diagram.



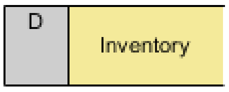
Process

A process is a business activity or function where the manipulation and transformation of data takes place. A process can be decomposed to finer level of details, for representing howdata is being processed within the process.



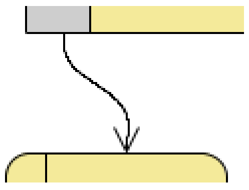
Data Store

A data store represents the storage of persistent data required and/or produced by the process. Here are some examples of data stores: membership forms, database table, etc.



Data Flow

A data flow represents the flow of information, with its direction represented by an arrow head that shows at the end(s) of flow connector.



Level-0:

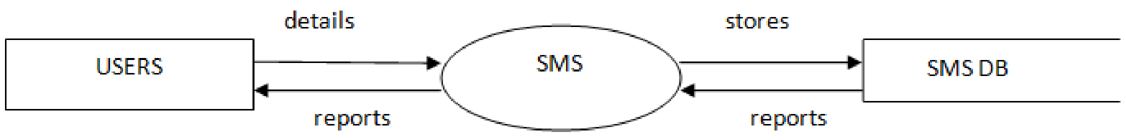


Figure 2.1: Level 0 of SMS

Level-1:

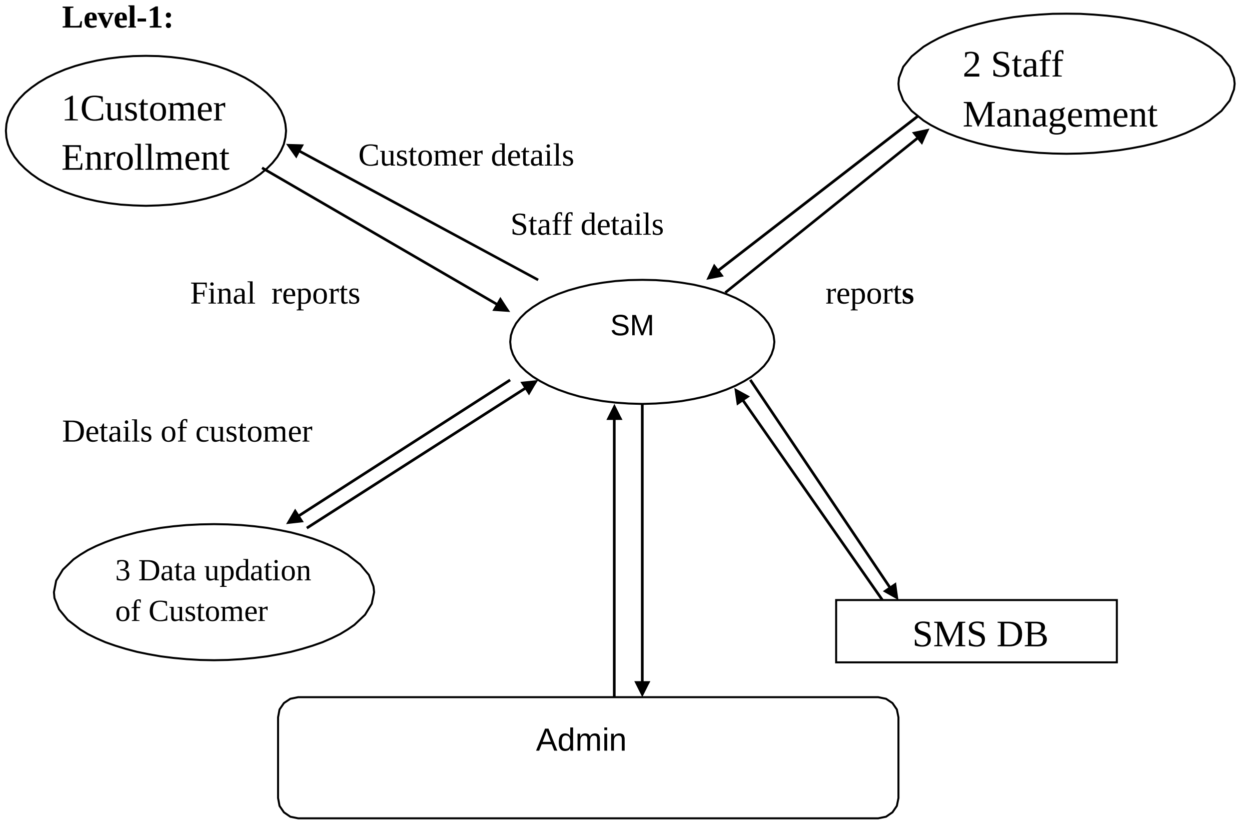


Figure 2.2: Level1 of SMS

Level:1 Customer Enrollment

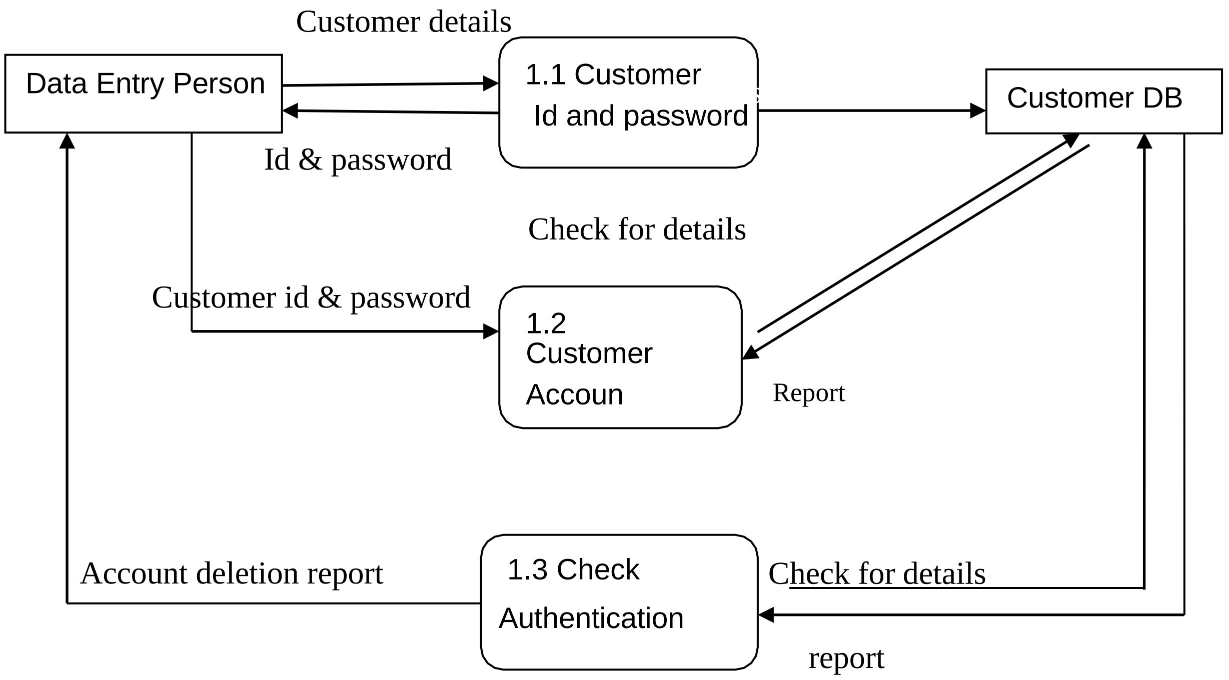


Figure 2.3:Level2 of Customer Enrollement

2. Staff Management

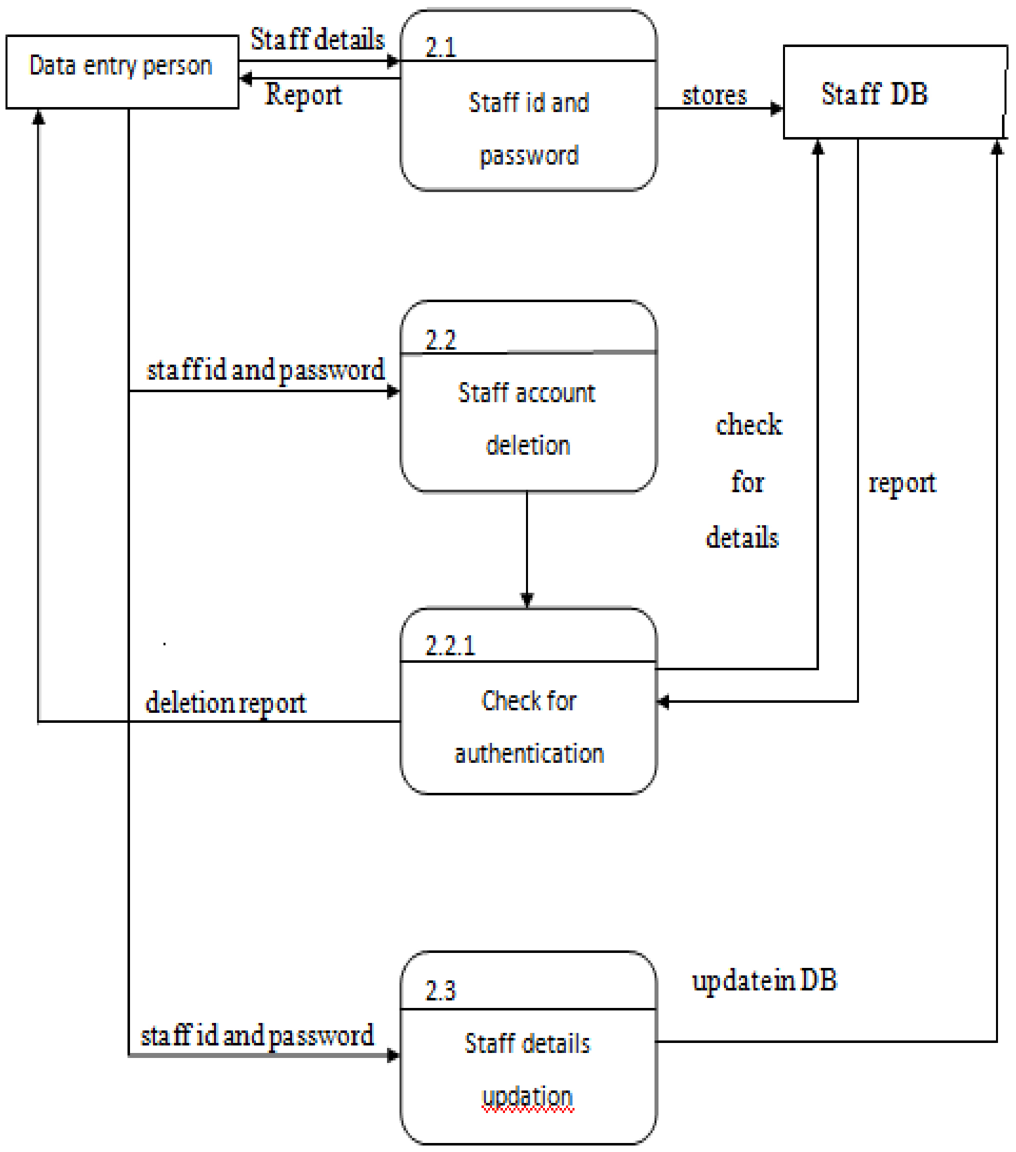


Figure 2.4:Level2 of Staff Management

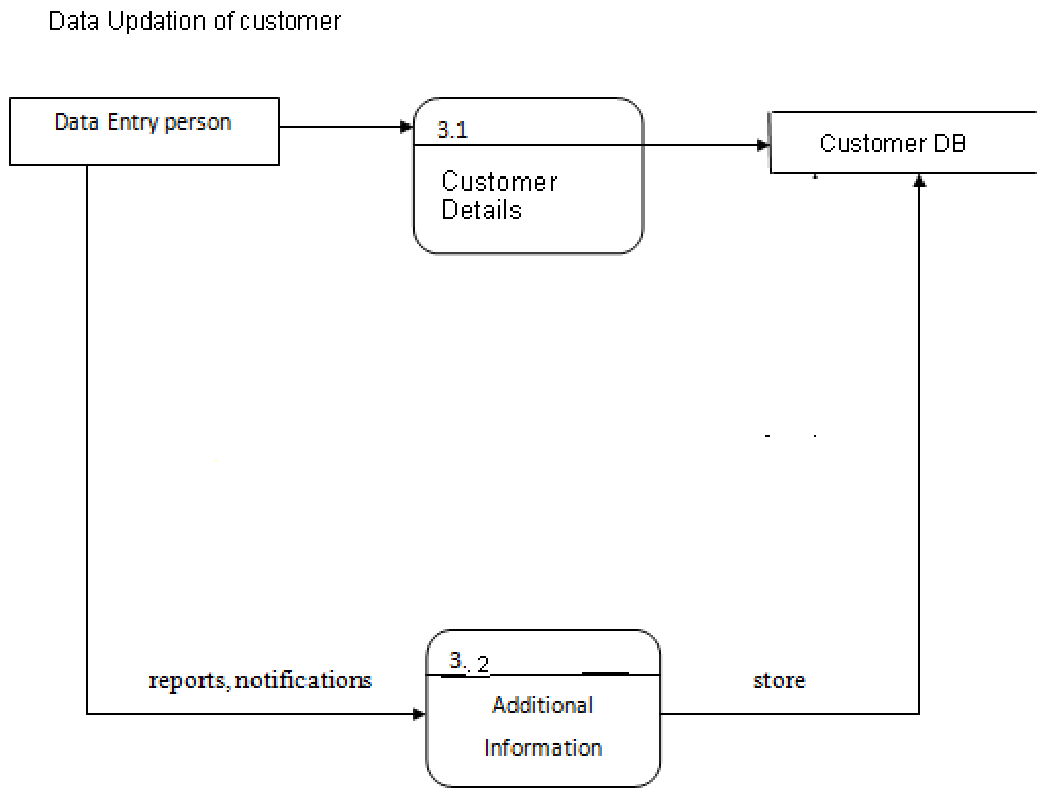


Figure 2.5:Level2 of Data Updation of customers

Experiment-III:Functional Decomposition

1. **Problem Statement:** To design functional decomposition diagram for software requirements

SOFTWARE REQUIREMENTS: Ms Word.

Theory:

Generally speaking, Decomposition is the process of breaking complex entities (processes, technology, business problems, business needs) into smaller sub-parts, and then breaking those smaller parts down even more, until the complex entity has been broken down into more discreet components with a more understandable structure.

It is a common analytical technique, and business analysts use it frequently. Among the things business analysts commonly decompose are:

- Systems – into processes, functions, rules, and decisions
- Processes – into steps, actors, and decisions

- Goals – into sub-goals and objectives
- Requirements – into functional requirements, non-functional requirements, business rules, decisions, and constraints

Decomposition is perhaps the single most-practiced technique in business analysis. A common output of decomposition is a hierarchical diagram of some sort, such as the Functional Decomposition Diagram. This is similar to a number of other Business Analysis techniques, including Organizational Analysis, Feature Tree's, Work Breakdown Structures, and Mind Mapping. The key difference between decomposition and those other analysis techniques is that the sub-components (the "children") of something that is decomposed should completely describe the component (the "parent") that has been decomposed. This may not be true of something like an Org Chart (which may not show all sub-units).

Types of Decomposition

Koopman describes three basic types of decomposition that are described below:

Structural Decomposition

Structures are physical components, logical objects, attributes, fields, or arrangements of other structures within a design. Structures typically answer the question of "what" in a design, and are typically described using nouns and adjectives. For business analyst's, structural decomposition is most likely to take the following forms:

- Logical Objects – as part of or preparation for object modeling

- Data (logical) Objects – as part of creating a data map, data dictionary, data flow, or similar activity
- Organizational Units – as part of organization modeling
- System components – either hardware or logical components, as part of solution analysis. This might include databases, data elements, hardware components, etc.

Behavioural / Functional Decomposition

Behaviours are an action, force, process, or control that is exerted on or by a structure with respect to the structure's external environment. In the case that only a portion of a design (a sub-design) is under consideration, other sub-designs constitute a portion of the external environment for the behaviour under consideration. Behaviours typically answer the questions of "how" and "when" in a design, and are typically described using verbs and adverbs. For most business analyst's, this takes the form of Functional Decomposition (where the business function is the "action" in question).

Functional Decomposition is the type of Decomposition that is described in many Business Analysis references and is one of the core business analysis techniques according to the IIBA [1]. It's also a good idea to know that Functional Decomposition exists in other disciplines, although it is often done in different ways. It exists in mathematics (for decomposing formulas and mathematical problems), signal processing, machine learning, database theory, knowledge representation, software development, and systems engineering. [2,3] Just be aware that when you refer to functional decomposition others may have different understandings of exactly what is being done than you do, so be sure you clarify.

Goal Decomposition

Goals are emergent properties that satisfy the needs which the effort or design is intended to full fill. Goals include any result that is not directly available as an "off-the-shelf" building block. Goals thus include performance targets, costs and aesthetics. For business analysts', goal decomposition can include:

- Strategic Goals – decomposed into departmental and business unit goals such as customer satisfaction, profitability measures, market share, etc.
- Project Goals – decomposed into specific (measurable) results or objectives. This includes cost limits, number of bugs, development time, etc.
- Personal Development Goals – decomposed into specific milestones. An example might be a goal to achieve CBAP certification broken down into specific sub-goals.

Cohesion and Coupling

The concepts of Cohesion and Coupling are key to well executed functional decomposition. However, they can also be applied to structural and goal decomposition as well.

Cohesion

Cohesion is simply a measure of how similar functions in a group are. The greater the cohesion between two or more functions, the more likely they should be grouped together in decomposition results. This helps to ensure that similar functions are grouped together and helps to identify potential duplicate functions that may be performed by different groups, or at different steps in a sequence. If a function does several things that have low cohesion, there is a good chance that the function needs further analysis and may need to be broken into several discreet functions at the same level as the current function.

Coupling

Coupling is a measure of interdependence between two or more functions. So a change to a function or function group with low coupling would affect very few other functions outside of its function group.

The measure of a good decomposition effort is that the sub-units have high cohesion and high coupling only within themselves, and low cohesion and low coupling with any other sub-groups.

Why do it?

Decomposition has a lot of uses. It can be used to:

- Assist in the analysis of processes by breaking them down into smaller component parts. These can be documented as sub-processes; analyzed for decisions, rules, and dependencies; or otherwise evaluated in relative isolation in order to make them more comprehensible.

- Assist in defining Scope by allowing the project team to clearly specify from a functional decomposition diagram which functions are in scope and how they relate and “roll-up” into larger feature sets.
- Assist in elicitation and analysis by identifying areas that need more elicitation work, and by providing a visual model of the functions that have been identified so far.

- As part of the elicitation and analysis processes by decomposing greater business goals and needs into actual requirements.

- Identifying current system functionality for an undocumented system that needs to be replaced.

- Strategic Planning, by breaking high-level corporate goals into lower division, department, and unit goals.

Examples of why decomposition might be used include:

- The decomposition of organization functions into sub-functions. For example, a Human Resources function could include sub-functions for Hiring Employee’s, Terminating Employee’s; Changing Employee Roles; Managing Employee Benefits, and other similar activities. Note that this is a decomposition of functions, what the business does, not how the business does it or the order it is done in. Each sub-function unit should be as independent as possible from the other units at the same level.

- The decomposition of high-level organizational functions into the processes that make up those functions.

- The decomposition of specific system functions into sub-functions. For example, a Log-In function may be broken down into sub-functions called Get User ID, Get User Password, Validate User Password, Check User Permissions, Display User Home Screen, Display Error Screen, etc.

- The decomposition of Strategic or Project Goals into the specific objectives or sub-goals that must be reached in order to meet the overall Goals.

Steps for functional decomposition

The following steps below show the general process that a Business Analyst goes through in order to conduct Functional Decomposition. Different documentation types will be shown afterwards.

Step 1

Decide the what and how of the decomposition effort. This usually involves making the following three decisions:

1. Identify the focus area that will be decomposed (the process, goal, business function, etc.).

2. Decide the level of detail that will be needed. In some cases you may only want a quick outline (such as when just starting a project to generate an initial elicitation artifact) while in others you may want substantial detail (such as when the functional decomposition results provide a significant part of the project or requirements artifacts).
3. Then decide what type of documentation will be generated to show the results of the decomposition effort. See below for examples.

Step 2

Work with the customer / SME if appropriate to identify the main components that are within the focus area.

Step 3

Review each of those main functions to identify any sub-components within them.

Step 4

Review all identified components to determine if they need further decomposition.

Step 5

Check for completeness. Are all aspects of the entity under review that should be captured represented? Are the components correctly organized into discreetly related groups? Is further decomposition needed?

Step 6

Repeat and Refine the results until all business and project team members are satisfied with the completeness and level of detail.

Step 7

If desired, each component can be documented in a consistent way so that each component:

- Has a unique name
- Describes how it fits into it's current group, and each group above it
- Identifies any business rules for that component
- Identifies any triggers (incoming or outgoing)

What Should the Results be?

In addition to differences in what is being functionally decomposed, there are different ways to document the results of the decomposition analysis, and different levels of detail that are used. The different methods of documentation include:

- A Functional Decomposition Diagram
- An outline structure
- A table structure

Advantages

- Functional Decomposition is an intuitive process for most people and is readily understood by the customer.
- Helps to discover duplicate or overlapping activities.
- Breaks complex systems into relatively separate components, which can help with scope, development, and planning.

Disadvantages

- Is internally focused (what are we currently doing versus what should we be doing)

- It can be easy to define too much detail.
- There is no way to be sure that every necessary component has been captured and properly related.
- It is very easy to conflate a functional diagram with an organizational diagram (especially for stakeholders). This can make it easy to overlook interdependencies where there may be high levels of coupling to functions that were not diagrammed on other organizational units, or where multiple organizational units are involved in the function that was diagrammed.
- A criticism of Functional Decomposition from a systems design perspective was put forth by Murray Cantor of Rational Software in 2003. It is called: Thoughts on Functional Decomposition

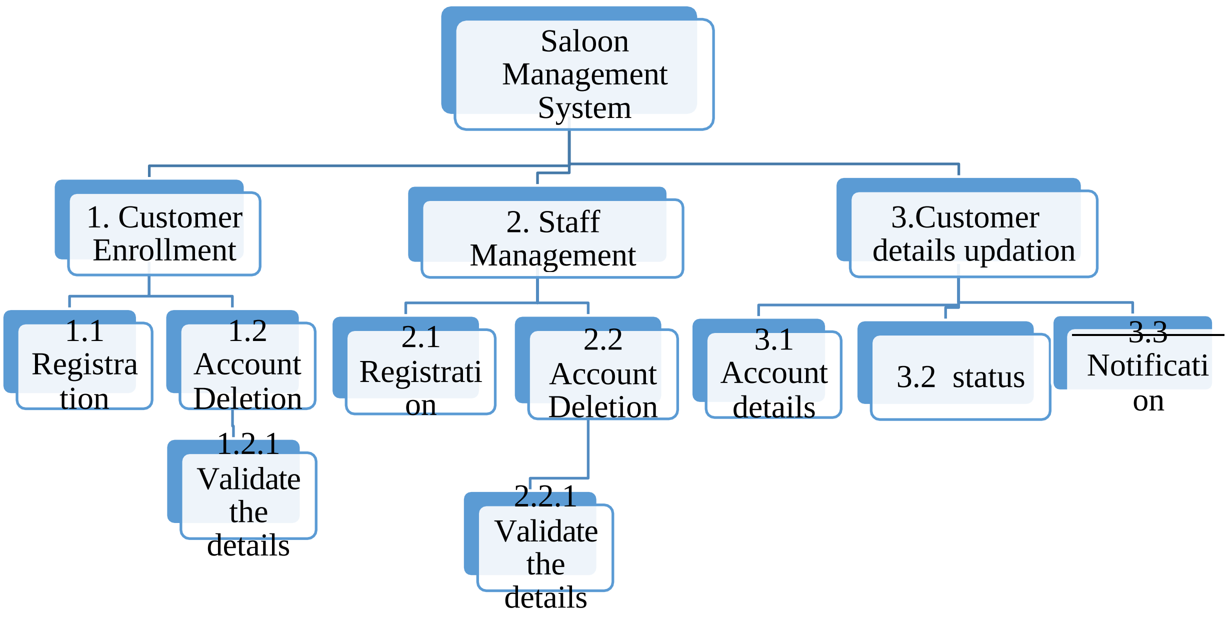


Figure 3.1:Functional Decomposition

Experiment-IV: Use Case Diagram

1. **Problem Statement:** To design use case diagram for software requirementsof School Management system

SOFTWARE REQUIREMENTS: Rational rose.

Use case diagram is a behavioural **UML diagram type** and frequently used to analyze various systems. They enable you to visualize the different types of roles in a system and how those roles interact with the system.

Importance of Use Case Diagrams

As mentioned before use case diagram are used to gather a usage requirement of a system. Depending on your requirement you can use that data in different ways. Below are few ways to use them.

- **To identify functions and how roles interact with them** – The primary purpose of use case diagrams.
- **For a high level view of the system** – Especially useful when presenting to managers or stakeholders. You can highlight the roles that interact with the system and the functionality provided by the system without going deep into inner workings of the system.
- **To identify internal and external factors** – This might sound simple but in large complex projects a system can be identified as an external role in another use case.

Use Case Diagram objects

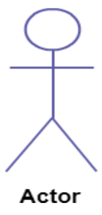
Use case diagrams consist of 4 objects.

- Actor
- Use case
- System
- Package

The objects are further explained below.

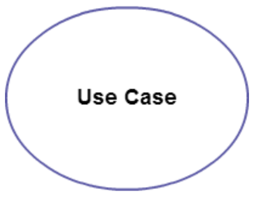
Actor

Actor in a use case diagram is **any entity that performs a role** in one given system. This could be a person, organization or an external system and usually drawn like skeleton shown below.



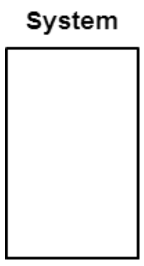
Use Case

A use case **represents a function or an action within the system**. Its drawn as an oval and named with the function.



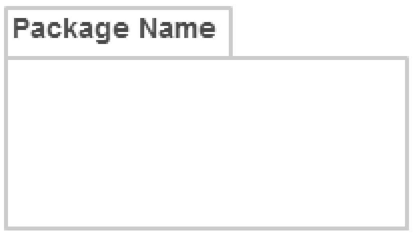
System

System is used to **define the scope of the use case** and drawn as a rectangle. This an optional element but useful when your visualizing large systems.



Package

Package is another optional element that is extremely useful in complex diagrams. Similar to **class diagrams**, packages are **used to group together use cases**. They are drawn like the image shown below.



Relationships in Use Case Diagrams

There are five types of relationships in a use case diagram. They are

- Association between an actor and a use case
- Generalization of an actor
- Extend relationship between two use cases
- Include relationship between two use cases
- Generalization of a use case

How to Create a Use Case Diagram

Let us see various processes using a Airline Reservation systems (ARS) as an example.

Identifying Actors

Actors are external entities that interact with your system. It can be a person, another system or an organization. In a ARS the most obvious actor is the passenger. Other actors can be new passenger, existing passenger.

Identifying Use Cases

Now it's time to identify the use cases. A good way to do this is to identify what the actors' needs from the system. In an ARS passenger can check availability, reserve, cancel the tickets and similar functions. So all of these can be considered as use cases.

Top level use cases should always provide a complete functions required by an actor. You can extend or include use cases depending on the complexity of the system.

Look for Common Functionality to use Include

Look for common functionality that can be reused across the system. If you find two or more use cases that share common functionality you can extract the common functions and add it to a separate use case. Then you can connect it via the include relationship to show that its always called when the original use case is executed.

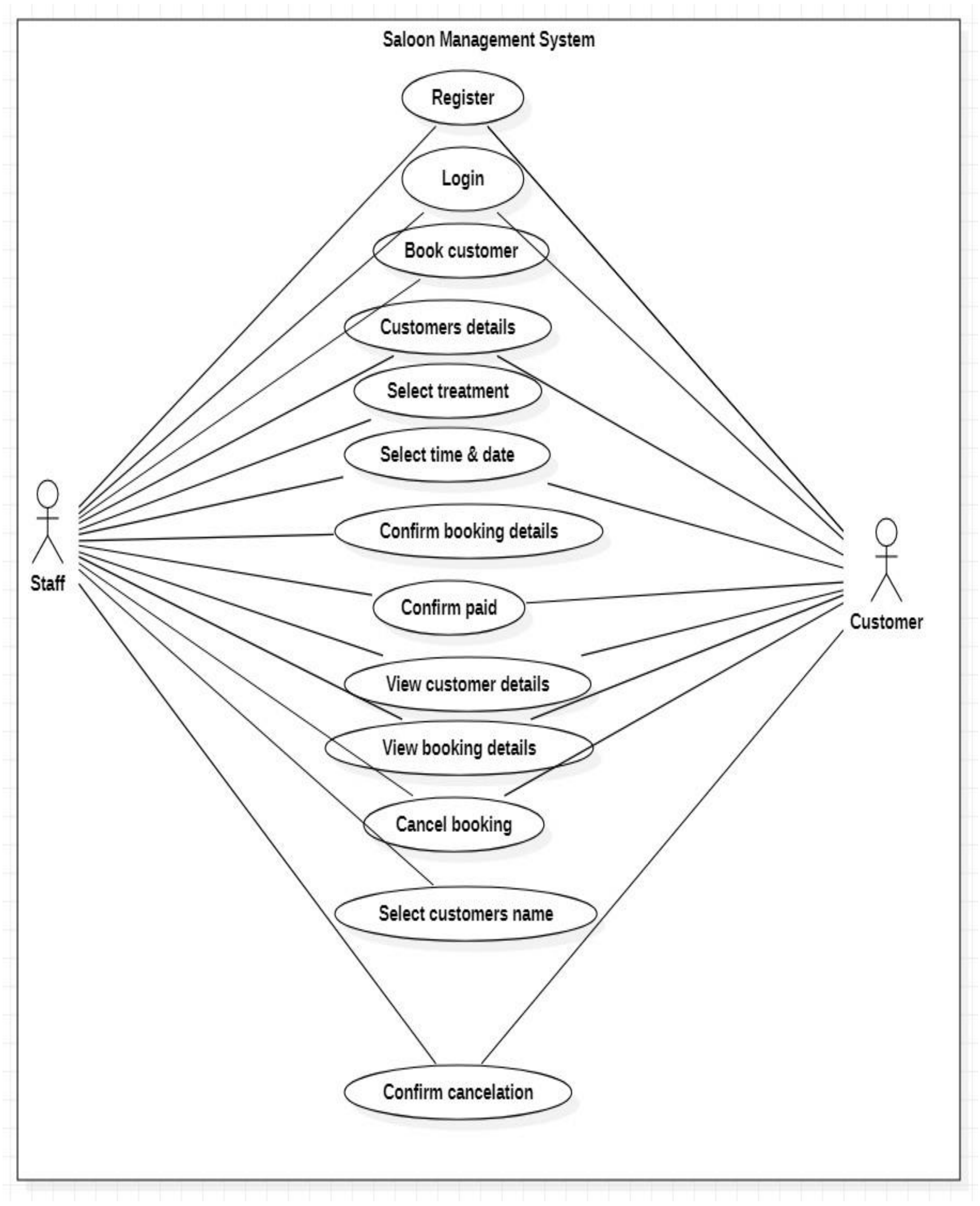
Generalize Actors and Use Cases

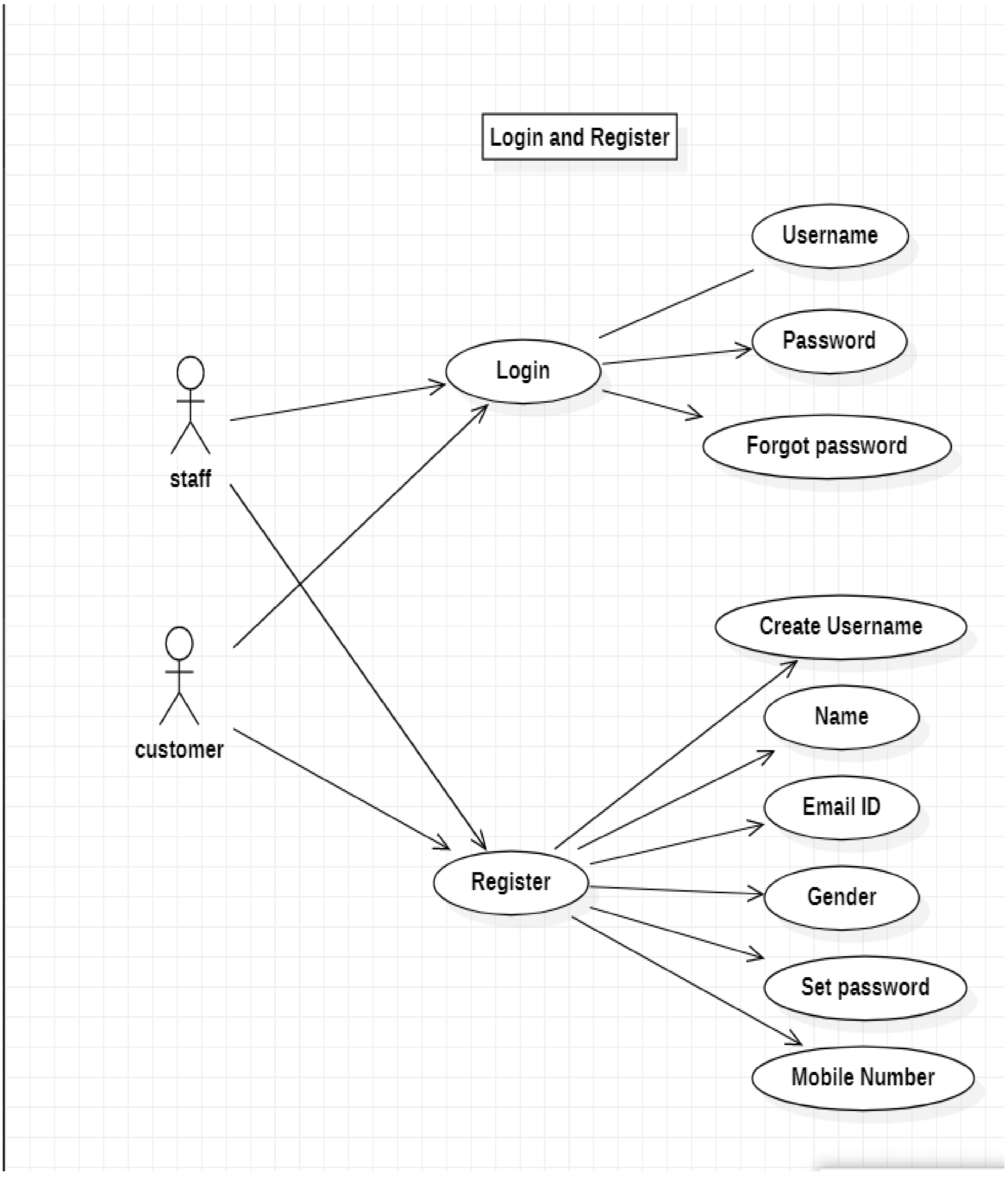
There may be instances where actors are associated with similar use cases while triggering few use cases unique only to them. In such instances you can generalize the actor to show the inheritance of functions. You can do a similar thing for use case as well.

One of the best examples of this is "Make Payment" use case in a payment system. You can further generalize it to "Pay by Credit Card", "Pay by Cash", "Pay by Check" etc. All of them have the attributes and the functionality of a payment with special scenarios unique to them.

Optional Functions or Additional Functions

There are some functions that are triggered optionally. In such cases you can use the extend relationship and attach and extension rule to it. Extend doesn't always mean its optional. Sometimes the use case connected by extend can supplement the base use case. Thing to remember is that the base use case should be able to perform a function on its own even if the extending use case is not called.


Saloon Management System
Register
Login
Book customer
Customers details
Select treatment
Select time & date
Confirm booking details
Confirm paid
View customer details
View booking details
Cancel booking
Select customers name
Confirm cancelation
Staff
Customer


Login and Register
Username
Password
Login
Forgot password
staff
customer
Create Username
Name
Email ID
Register
Gender
Set password
Mobile Number

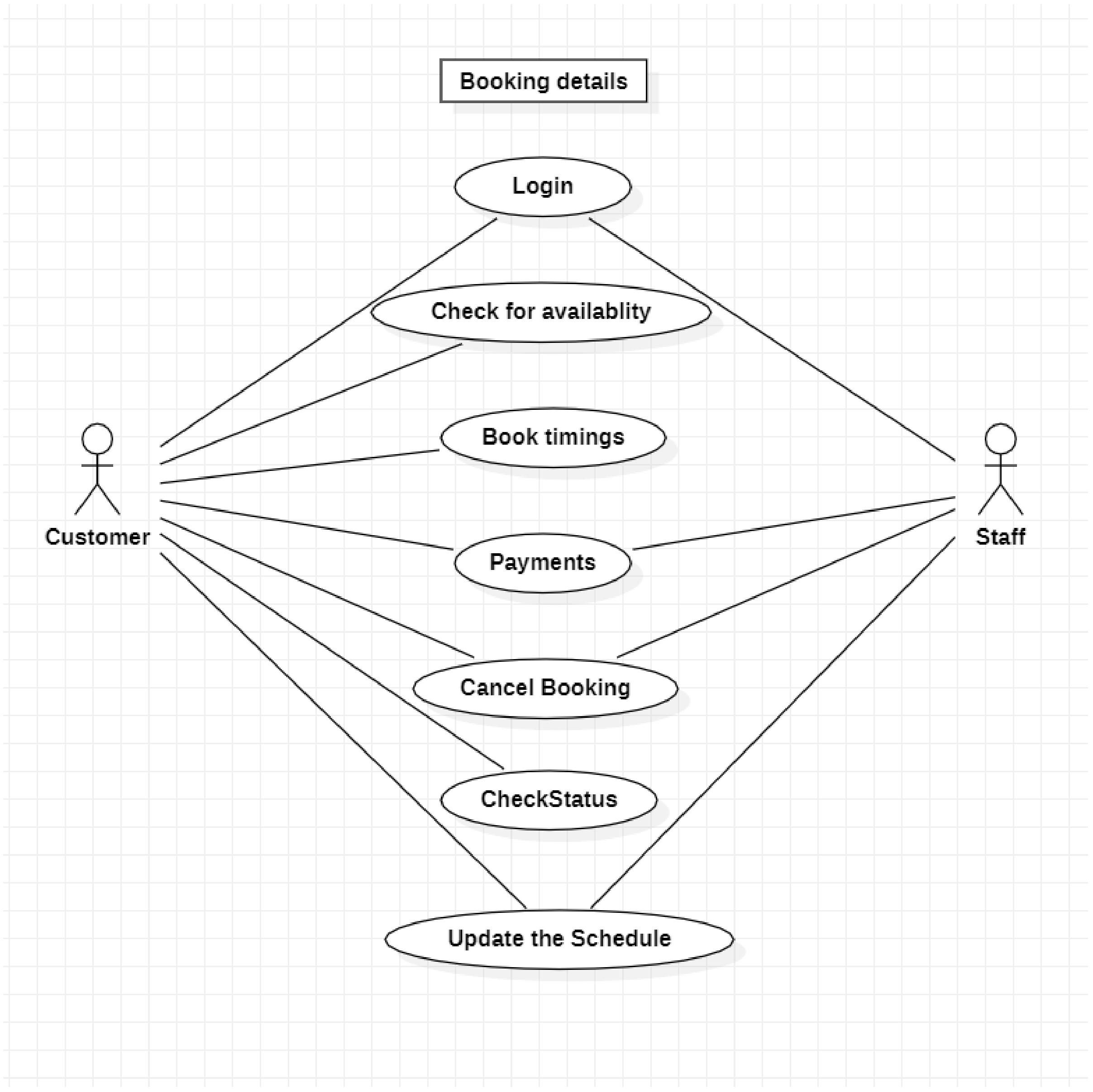


Figure 4.1: Usecase diagram

Experiment-V: Interaction Diagram

1. Sequence Diagram

Problem Statement: To draw Interaction diagram of School Management system

REQUIREMENTS:

SOFTWARE : Rational rose

DEFINITION:

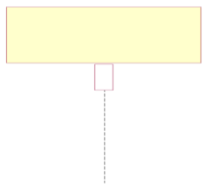
- A sequence diagram is a graphical view of a scenario that shows object interaction in a time-based sequence¾what happens first, what happens next.
- Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces.
- This type of diagram is best used during early analysis phases in design because they are simple and easy to comprehend.
- Sequence diagrams are normally associated with use cases.
- Sequence diagrams are closely related to collaboration diagrams and both are alternate representations of an interaction.
- There are two main differences between sequence and collaboration diagrams:
- Sequence diagrams show time-based object interaction
- While collaboration diagrams show how objects associate with each other.

CONTENTS:

A sequence diagram has two dimensions: typically, vertical placement represents time and horizontal placement represents different objects.

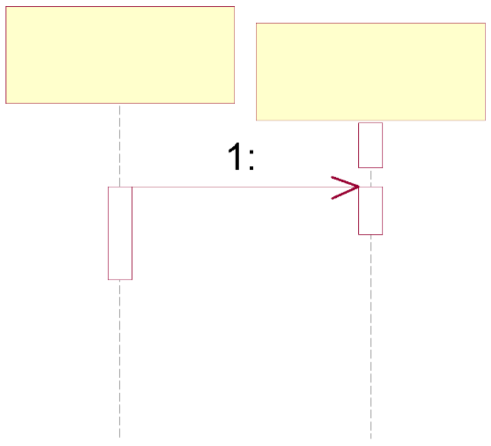
The following tools located on the sequence diagram toolbox enable we to model sequence diagrams:

· Object: An object has state, behavior, and identity. The structure and behavior of similar objects are defined in their common class. Each object in a diagram indicates some instance of a class. An object that is not named is referred to as a class instance.



If we use the same name for several object icons appearing in the same collaboration or activity diagram, they are assumed to represent the same object; otherwise, each object icon represents a distinct object. Object icons appearing in different diagrams denote different objects, even if their names are identical. Objects can be named three different ways: object name, object name and class, or just by the class name itself.

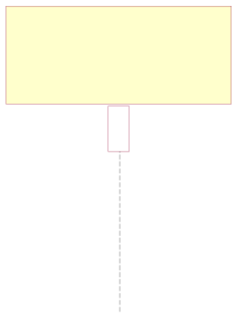
. Message Icons: A message icon represents the communication between objects indicating that an action will follow. The message icon is a horizontal, solid arrow connecting two lifelines together. The following message icons show three different ways a message icon can appear: message icon only, message icon with sequence number, and message icon with sequence number and message label.



Each message icon represents a message passed between two objects, and indicates the direction of message is going. A message icon in a collaboration diagram can represent multiple messages. A message icon in a sequence diagram represents exactly one message.

· Focus of Control: Focus of Control (FOC) is an advanced notational technique that enhances sequence diagrams. It shows the period of time during which an object is performing an action, either directly or through an underlying procedure.

FOC is portrayed through narrow rectangles that adorn lifelines. The length of an FOC indicates the amount of time it takes for a message to be performed. When we move a message vertically, each dependent message will move vertically as well. Also, we can move a FOC vertically off of the source FOC to make it detached and independent. An illustration of a sequence diagram with FOC notation follows:



· Message to Self: A Message to Self is a tool that sends a message from one object back to the same object. It does not involve other objects because the message returns to the same object. The sender of a message is the same as the receiver.

The following example shows a Message to Self labeled, 2. Turn on Light, in a sequence diagram:

· Note: A note captures the assumptions and decisions applied during analysis and design. Notes may contain any information, including plain text, fragments of code, or references to other documents. Notes are also used as a means of linking diagrams. A note holds an unlimited amount of text and can be sized accordingly.

Notes behave like labels. They are available on all diagram toolboxes, but they are not considered part of the model. Notes may be deleted like any other item on a diagram.

·Note Anchor: A note anchor connects a note to the element that it affects.

To draw a note anchor, place a note on the diagram and connect the note to an element with the note anchor icon

DIAGRAM:

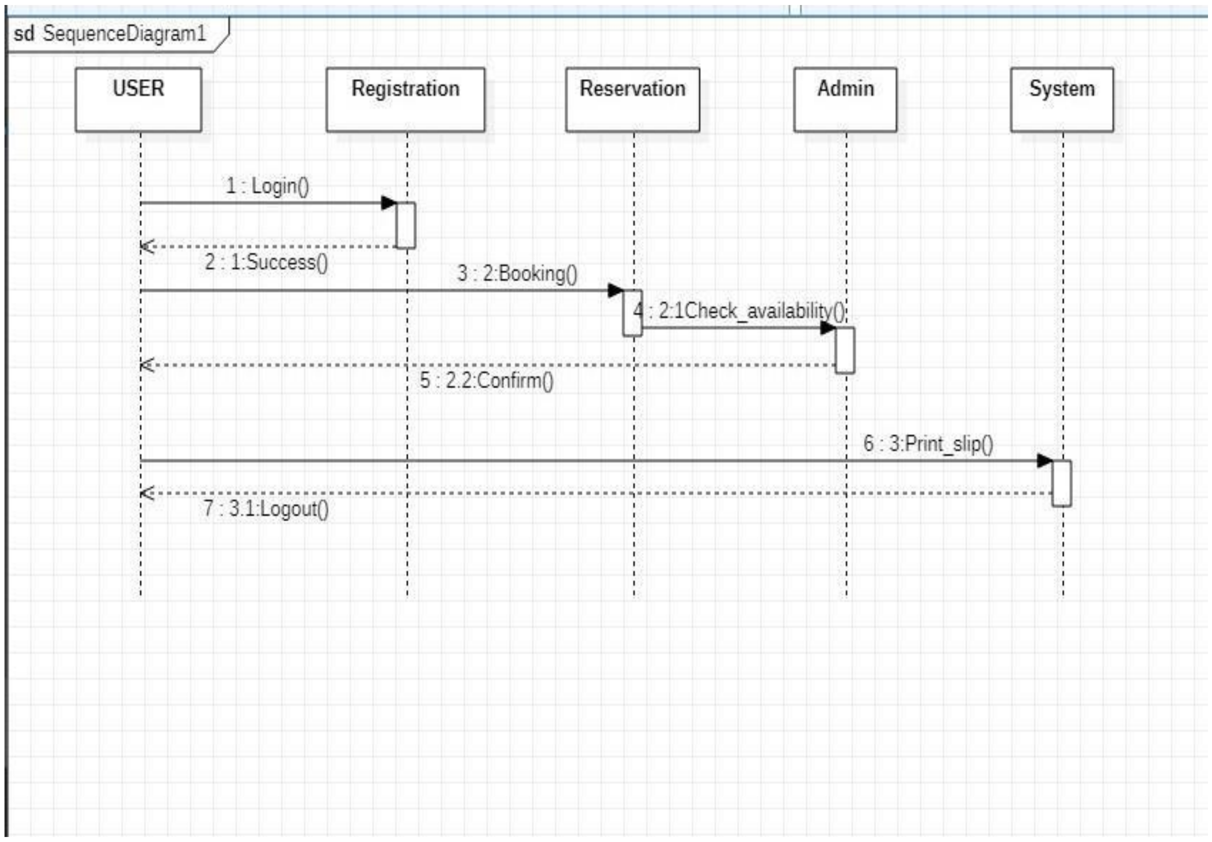


Figure 5.1 Sequence diagrams

2. Collaboration Diagrams

Problem Statement: To draw Collaboration diagram for School Management System

REQUIREMENTS:

SOFTWARE **:** Rational rose

THEORY:

Collaboration diagrams and sequence diagrams are alternate representations of an interaction. A collaboration diagram is an interaction diagram that shows the order of messages that implement an operation or a transaction. A sequence diagram shows object interaction in a time-based sequence.

Collaboration diagrams show objects, their links, and their messages. They can also contain simple class instances and class utility instances. Each collaboration diagram provides a view of the interactions or structural relationships that occur between objects and object-like entities in the current model.

The Create Collaboration Diagram Command creates a collaboration diagram from information contained in the sequence diagram. The Create Sequence Diagram Command creates a sequence diagram from information contained in the interaction's collaboration diagram. The Goto Sequence Diagram and Goto Collaboration Diagram commands traverse between an interaction's two representations.

Collaboration diagrams contain icons representing objects. We can create one or more collaboration diagrams to depict interactions for each logical package in your model. Such collaboration diagrams are themselves contained by the logical package enclosing the objects they depict.

An Object Specification enables you to display and modify the properties and relationships of an object. The information in a specification is presented textually. Some of this information can also be displayed inside the icons representing objects in collaboration diagrams.

We can change properties or relationships by editing the specification or modifying the icon on the diagram. The associated diagrams or specifications are automatically updated.

During:	Use Collaboration Diagrams To:
Analysis	Indicate the semantics of the primary and secondary interactions
Design	Show the semantics of mechanisms in the logical design of the system

Use collaboration diagrams as the primary vehicle to describe interactions that express your decisions about the behavior of the system.

Object:

An object has state, behavior, and identity. The structure and behavior of similar objects are defined in their common class. Each object in a diagram indicates some instance of a class. An object that is not named is referred to as a class instance.

If we use the same name for several object icons appearing in the same collaboration or activity diagram, they are assumed to represent the same object; otherwise, each object icon represents a distinct object. Object icons appearing in different diagrams denote different objects, even if their names are identical. Objects can be named three different ways: object name, object name and class, or just by the class name itself.

For more information on objects that appear on activity diagrams, click here.

If we specify the name of the object's class in the Object Specification, the name must identify a class defined in the model.

Graphical Depiction

The object icon is similar to a class icon except that the name is underlined:

If you have multiple objects that are instances of the same class, you can modify the object icon by clicking Multiple Instances in the Object Specification. When you select this field, the icon is changed from one object to three staggered objects:

Adornments

Concurrency

An object's concurrency is defined by the concurrency of its class.

You can display concurrency by clicking Show Concurrency from the object's shortcut menu. The adornment is displayed at the bottom of the object icon.

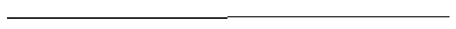
Persistence

You can explicitly set the persistence of an object in the Object Specification. You can display this value as an adornment by clicking Persistence from the shortcut menu. If you display both concurrency and persistence, the object's persistence is listed second:

Link:

Objects interact through their links to other objects. A link is an instance of an association,analogous to an object being an instance of a class. A link should exist between two objects, including class utilities, only if there is a relationship between their corresponding classes. The existence of an relationship between two classes symbolizes a path of communication between instances of the classes: one object may send messages to another.Links can support multiple messages in either direction. If a message is deleted, the link remains intact.

Graphical Depiction



The link is depicted as a straight line between objects or objects and class instances in a collaboration diagram. If an object links to itself, use the loop version of the icon.

Adornments

You can document how objects see one another by specifying a visibility type on the General tab of the Link Specification or by selecting an item from the shortcut menu. You can select global, parameter, field, or local visibility. You can also specify whether object visibility is shared.

When you specify a visibility type, a visibility adornment is displayed on the supplier end of link. Use this adornment only when you need to document an important tactical decision.

Message/Event:

A message is the communication carried between two objects that trigger an event. A message carries information from the source focus of control to the destination focus of control.

A message is represented on collaboration diagrams and sequence diagrams by a message icon which visually indicates its synchronization. The synchronization of a message can be modified through the message specification. The following synchronization types are supported:

· Simple

- Synchronous
- Balking
- Timeout
- Asynchronous
- Procedure Call
- Return

In a collaboration diagram, a message icon can represent several messages. If all messages represented by a message icon do not have the same synchronization, the "simple" message icon is displayed. You can change the synchronization of the message by editing the message specification.

The message to self icon appears as a message that returns to itself.

The collaboration diagram toolbox contains two message tools. The forward message tool, bearing an arrow pointing "northeast," places a message icon from client to supplier

The reverse message tool, bearing an arrow pointing "southwest," places a message icon from supplier to client. The default synchronization for a message is "simple."

Scripts may be attached to messages to enhance the messages.

If a message is deleted, the link remains intact.

Message Icons:

A message icon represents the communication between objects indicating that an action will follow. The message icon is a horizontal, solid arrow connecting two lifelines together. The following message icons show three different ways a message icon can appear: message icon only, message icon with sequence number, and message icon with sequence number and message label.

Each message icon represents a message passed between two objects, and indicates the direction of message is going. A message icon in a collaboration diagram can represent multiple messages. A message icon in a sequence diagram represents exactly one message.

Displaying Operation Names

You may specify the name of a message through the message specification, or by right-clicking on the message icon to display the shortcut menu of a message.

You can adjust the position of a message icon by selecting it and dragging it to a new location. This action moves the icon, the names of any operations, and the sequence numbers to the new location.

You can reposition the message labels by selecting one and dragging to a new location. This action moves the message labels only; the message icon is not moved.

You can enhance a message by attaching a script to the message. The script will remain aligned with the message regardless of any changes in the position of the message. Scripts are attached to a message by clicking Edit > Attach Script.

Changing the Synchronization Type

The default message synchronization icon is "simple." To change the synchronization type, select a type from the synchronization field in the message specification. The following synchronization types are supported:

- Simple
- Synchronous
- Balking
- Timeout
- Asynchronous
- Procedure Call
- Return

DIAGRAM:

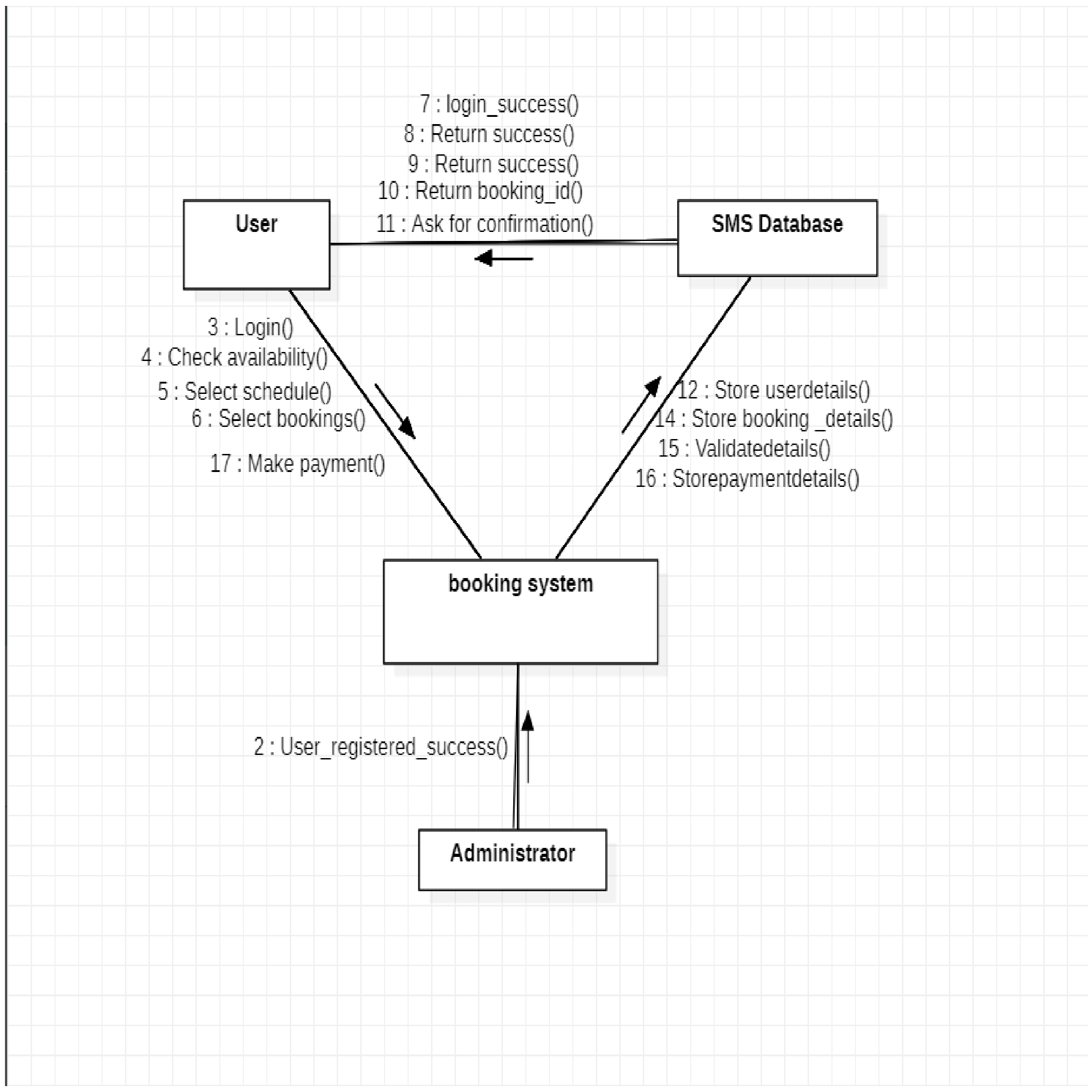


Figure 5.2: Collaboration diagram.

Experiment-VI: State Machine Diagrams

1. **Activity Diagrams**

Problem Statement: To draw Activity diagram for School Management System

REQUIREMENTS:

SOFTWARE **:** Rational rose

DEFINITION:

Activity diagrams provide a way to model the workflow of a business process.

We can also use activity diagrams to model code-specific information such as a class operation.

Activity diagrams are very similar to a flowchart because wecan model a workflow from activity to activity.

An activity diagram is basically a special case of a state machine in which most of the states are activities and most of the transitions are implicitly triggered by completion of the actions in the source activities.

The main difference between activity diagrams and state charts is activity diagrams are activity centric, while state charts are state centric.

An activity diagram is typically used for modelling the sequence of activities in a process, whereas a state chart is better suited to model the discrete stages of an object's lifetime.

CONTENTS:

Activity Diagram Tools

We can use the following tools on the activity diagram toolbox to model activity diagrams:

·Activities: An activity represents the performance of task or duty in a workflow. It may also represent the execution of a statement in a procedure. An activity is similar to a state, but expresses the intent that there is no significant waiting (for events) in an activity. Transitions connect activities with other model elements and object flows connect activities with objects.

·Decisions: A decision represents a specific location on an activity diagram or state chart diagram where the workflow may branch based upon guard conditions. There may be more than two outgoing transitions with different guard conditions, but for the most part, a decision will have only two outgoing transitions determined by a Boolean expression.

·End State: An end state represents a final or terminal state on an activity diagram or state chart diagram. Place an end state when you want to explicitly show the end of a workflow on an activity diagram or the end of a state chart diagram. Transitions can only occur into an end state; however, there can be any number of end states per context.

·Object: Rational Rose allows objects on activity, collaboration, and sequence diagrams. Specific to activity diagrams, objects are model elements that represent something you can feel and touch. It might be helpful to think of objects as the nouns of the activity diagram and activities as the verbs of the activity diagram. Further, objects on activity diagrams allow you to diagram the input and output relationships between activities. In the following diagram, the Submit Defect and Fix Defects can be thought of as the verbs and the defect objects as the nouns in the activity diagram vocabulary. Objects are connected to activities through object flows.

·Object Flow: An object flow on an activity diagram represents the relationship between an activity and the object that creates it (as an output) or uses it (as an input).

Rational Rose draws object flows as dashed arrows rather than solid arrows to distinguish them from ordinary transitions. Object flows look identical to dependencies that appear on other diagram types.

·Start State: A start state (also called an "initial state") explicitly shows the beginning of a workflow on an activity diagram or the beginning of the execution of a state machine on a state chart diagram. Wecan have only one start state for each state machine because each workflow/execution of a state machine begins in the same place. If you use multiple activity and/or state chart diagrams to model a state machine, the same start state can be placed on the multiple diagrams. When wemodel nested states or nested activities, one new start state can be created in each context.

Normally, only one outgoing transition can be placed from the start state. However, multiple transitions may be placed on a start state if at least one of them is labelled with a condition. No incoming transitions are allowed.

·States: A state represents a condition or situation during the life of an object during which it satisfies some condition or waits for some event. Each state represents the cumulative history of its behaviour.

·Swim lanes: Swim lanes are helpful when modelling a business workflow because they can represent organizational units or roles within a business model. Swim lanes are very similar to an object because they provide a way to tell who is performing a certain role. Swim lanes only appear on activity diagrams. Weshould place activities within swim lanes to determine which unit is responsible for carrying out the specific activity. For more information on swim lanes, look at the swim lane sample.

When a swim lane is dragged onto an activity diagram, it becomes a swim lane view. Swim lanes appear as small icons in the browser while a swim lane views appear between the thin, vertical lines with a header that can be renamed and relocated.

<u>Synchronizations</u>: Synchronizations enable you to see a simultaneous workflow in an activity diagram or state chart diagram. Synchronizations visually define forks and joins representing parallel workflow.

<u>Transitions:</u> A state transition indicates that an object in the source state will perform certain specified actions and enter the destination state when a specified event occurs or when certain conditions are satisfied. A state transition is a relationship between two states, two activities, or between an activity and a state.

We can show one or more state transitions from a state as long as each transition is unique. Transitions originating from a state cannot have the same event, unless there are conditions on the event.

DIAGRAM:

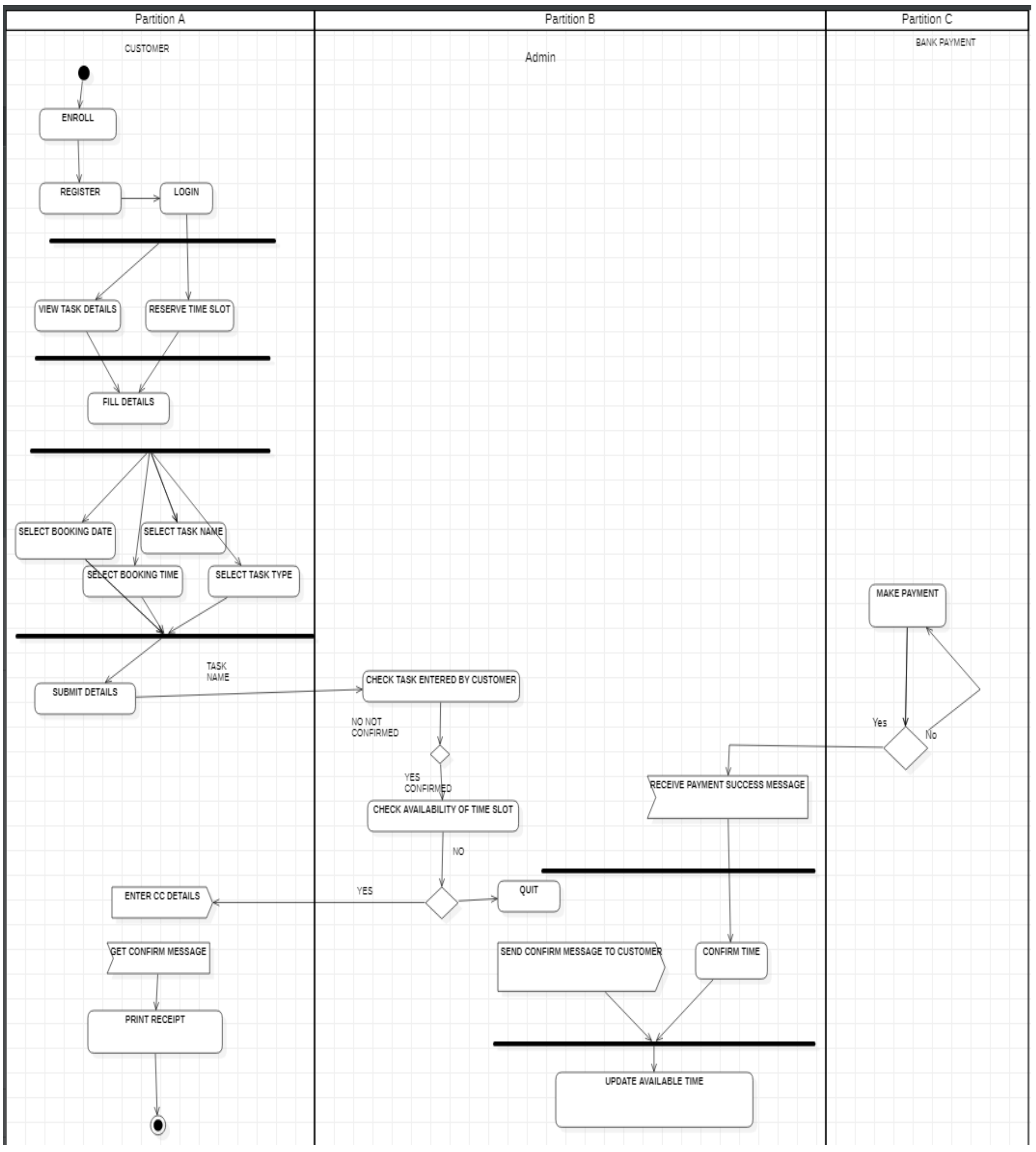


Figure6.1:Activity Diagram

2. State Chart Diagram

Problem Statement: To implement State chart diagram for School Management System

REQUIREMENTS:

HARDWARE : PIII Processor, 512 MB RAM, 80GB

SOFTWARE : Rational Rose software state chart diagram tools

THEORY:

State chart diagrams model the dynamic behaviour of individual classes or any other kind of object. They show the sequences of states that an object goes through, the events that cause a transition from one state to another and the actions that result from a state change.

State chart diagrams are closely related to activity diagrams. The main difference between the two diagrams is state chart diagrams are state centric, while activity diagrams are activity centric. A state chart diagram is typically used to model the discrete stages of an object's lifetime, whereas an activity diagram is better suited to model the sequence of activities in a process.

Each state represents a named condition during the life of an object during which it satisfies some condition or waits for some event. A state chart diagram typically contains one start state and multiple end states. Transitions connect the various states on the diagram. As with activity diagrams, decisions, synchronizations, and activities may also appear on state chart diagrams.

We use the following tools on the state chart diagram toolbox to model state chart diagrams:

Decisions: A decision represents a specific location on an activity diagram or state chart diagram where the workflow may branch based upon guard conditions. There may be more than two outgoing transitions with different guard conditions, but for the most part, a decision will have only two outgoing transitions determined by a Boolean expression.

·Synchronizations: Synchronizations enable you to see a simultaneous workflow in an activity diagram or state chart diagram. Synchronizations visually define forks and joins representing parallel workflow.

Forks and Joins: A fork construct is used to model a single flow of control that divides into two or more separate, but simultaneous flows. Every fork that appears on an activity diagram should ideally be accompanied by a corresponding join. A join consists of two of more flows of control that unite into a single flow of control. All model elements (such as activities and states) that appear between a fork and join must complete before the flow of controls can unite into one.

·States: A state represents a condition or situation during the life of an object during which it satisfies some condition or waits for some event. Each state represents the cumulative history of its behaviour.

·Transitions: A state transition indicates that an object in the source state will perform certain specified actions and enter the destination state when a specified event occurs or when certain conditions are satisfied. A state transition is a relationship between two states, two activities, or between an activity and a state.

You can show one or more state transitions from a state as long as each transition is unique. Transitions originating from a state cannot have the same event, unless there are conditions on the event.

Start States: A start state (also called an "initial state") explicitly shows the beginning of a workflow on an activity diagram or the beginning of the execution of a state machine on a state chart diagram. Wehave only one start state for each state machine because each workflow/execution of a state machine begins in the same place. If weuse multiple activity and/or state chart diagrams to model a state machine, the same start state can be placed on the multiple diagrams. When wemodel nested states or nested activities, one new start state can be created in each context.

Normally, only one outgoing transition can be placed from the start state. However, multiple transitions may be placed on a start state if at least one of them is labelled with a condition. No incoming transitions are allowed.

You can label start states, if desired. State Specifications are associated with each start state.

·End States: An end state represents a final or terminal state on an activity diagram or state chart diagram. Place an end state when you want to explicitly show the end of a workflow on an activity diagram or the end of a state chart diagram. Transitions can only occur into an end state; however, there can be any number of end states per context.

You can label end states, if desired. State Specifications are associated with each end state.

DIAGRAM:

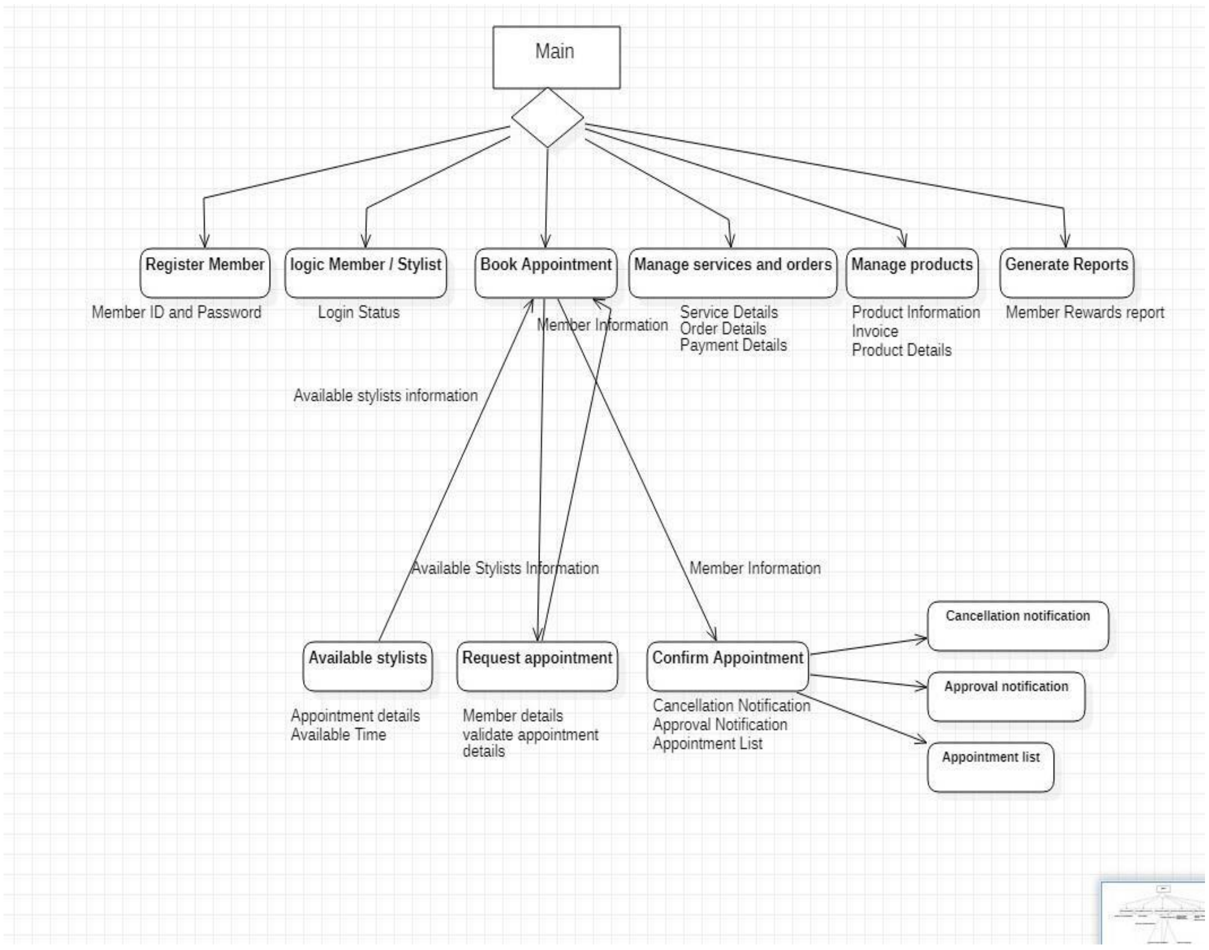


Figure 6.2:State chart diagram

Experiment-VII:Class Diagram

1. **Problem Statement:**To draw a Class Diagram for School ManagementSystem.

REQUIREMENTS:

**SOFTWARE : **Rational rose

THEORY:REPRESENTING BEHAVIOUR AND STRUCTURE:

A Class embodiesa set of responsibilities that define the behaviour of the objects in the class. The responsibilities are carried out by the operations defined for the class.

The structure of an object is described by the attributes of the class. Each attributes a data definition held by objects of the class. Objects defined for the class have value for every attribute of the class. The attributes and operations defined for a class are the ones that have meaning and utility within the application that is being developed.

A style need to be followed (like all operations must be in lower case) while defining attributes and operations which leads to more readable and maintainable models and code.

A class should have a major theme.

CREATING OPERATIONS:

Messages in interaction diagrams typically are mapped to operations of the receiving class. However, there are some special cases where the message does not become an operation: messages to and from actors representing people and message to and from classes representing GUI classes (boundary classes).

The message received by a boundary class is implemented as some type of GUI control (like button). And the behaviour is carried out by the control itself.

If the message is to or from actors representing a Physical person, the message Is statement of human procedure and is incorporated into a user manual, but not to an operation.

If the message is to or from an actor that represents an external system, a class is created to hold the protocol that Is used to perform the communication with the external system. In this case the message is mapped to an operation of the class.

Example: 1. The Professor actor must enter a password to start the 'Add a Course Offering' Scenario. This is represented as a message to the boundary class Professor Course Options. This never be an operation of the user Interface class- it will likely be a text field on a window.

2. The professor must have some sort of password to activate the system is important requirement that should be captured in the user manual. Operations should be named in terms of the class performing the operation, not the class asking for the functionality to be performed.

Example: the operation to add a student to a course offering is called add student ().

Operation names should not reflect the implementation of the operation since the implementation change over time.

Example: get Number of Students 0 rather that n Calculate Number Of Students.

Operations may also be created independently of interaction diagrams, since not all scenarios are represented in a diagram. This is also true for the operations that are created to "help" another operation.

Example: A course might have to determine if a specified professor is certified to teach it before it adds the professor as a teacher. An operation called validateProfessor0 could be an operation created to perform this behavior.

DOCUMENTING OPERATIONS:

The documentation should state the functionality provided by the operation. It should also state the input and output values for the operation. They form the signature of the operation.

Example: For set Professor () operation, documentation would be:

Add a professor as a teacher for a course offering of the course.

I/P: Professor, Course Offering

O/P: Success Indicator.

RELATIONSHIPS AND OPERATION SIGNATURES:

The signature of an operation may indicate a relationship.

Example: The two inputs to the set Professor () operation of the Course class are Professor (a class) and course offering (a class). This implies that relationships exist between:

1. Course and Professor

2. Course and Course Offering

Relationships based on operation signatures initially are modelled as associations that may be refined into dependency relationships as the design of the system matter

Package relationships should also be refined as relationships based on operation signatures are added to the modelled.

Example: Now we have added a relationship betweenCourse and Professor. Based on this one can say that there is dependency relationship b/w University Artefacts and People Packages.

CREATING ATTRIBUTES:

Many of the attributes of a class are found in the problem statement, the set of system requirements and the flow of events documentation.

The attributes may also be discovered when supplying the definition of a class.

Finally, domain expertise is also a good source of attributes for a class.

Example: The requirements state that information such as course name, description, no. of credit hours is available in the Course Catalog for a semester. This implies that name, description and no. of credit hours are attributes of the Course class. Document each attribute with a clear, concise definition. The definition should state that the purpose of the attribute, not the structure of the attribute.

ASSOCIATION CLASSES:

A relationship may also have structure and behavior. This is true when the information deals with a link between two objects and not with one object by itself.

The structure and behavior belonging to a relationship is held in an association class.

Example: A student may take up to four course offerings, and course offering may have between three and ten students. Each student must receive a grade for the course offering. Where is the grade held? This information belongs to the link b/w a student and a course offering. This is modeled as an association class. Since an association class behaves like any other class, it too can have relationships. Grade is associated with a report card received by the student each semester.

CLASS DIAGRAM STRUCTURE:

DEFINITION:

A class diagram shows the existence of classes and their relationships in the logical design of a system.A class diagram may represent all or part of the class structure of a system.

CONTENTS:

A class diagram contains

1. Classes
2. Relationships
3. Interfaces

1) Classes: Definition

A class is a set of objects that share a common structure and common behavior (the same attributes, operations, relationships and semantics). A class is an abstraction of real-world items. When these items exist in the real world, they are instances of the class and are referred to as objects.

2) Relationships:

Aggregate Relationship

Definition

Use the aggregate relationship to show a whole and part relationship between two classes.

The class at the client end of the aggregate relationship is sometimes called the aggregate class. An instance of the aggregate class is an aggregate object. The class at the supplier end of the aggregate relationship is the part whose instances are contained or owned by the aggregate object.

Association Relationship

An association provides a pathway for communication. The communication can be between use cases, actors, classes or interfaces. Associations are the most general of all relationships and consequentially the most semantically weak. If two objects are usually considered independently, the relationship is an association

Dependency Relationship

A dependency is a relationship between two model elements in which a change to one model element will affect the other model element. Use a dependency relationship to connect model elements with the same level of meaning.

Generalize Relationship

Definition: A generalize relationship between classes shows that the subclass shares the structure or behaviour defined in one or more super classes. Use a generalize relationship to show a "is-a" relationship between classes.

<u>Realize Relationship</u>

Definition:

A realization is a relationship between classes, interfaces, components, and packages that connects a client element with a supplier element. A realization relationship between classes and interfaces and between components and interfaces shows that the class realizes the operations offered by the interface.

3) <u>Interfaces</u>

Definition

An interface specifies the externally-visible operations of a class and/or component, and has no implementation of its own. An interface typically specifies only a limited part of the behavior of a class or component.

Interfaces belong to the logical view but can occur in class, use case and component diagrams.

An interface in a component diagram is displayed as a small circle with a line to the component that realizes the interface:

<u>Adornments</u>

You can further define an interface using the Class Specification. Some class specification fields correspond to adornment and compartment information that you can display in the class diagram.

Type Shortcut Menu Item Description

Abstract : Defines the interface as a base class with no instances.

Attributes Edit CompartmentAttributes not recommended by UML.

Cardinality Show CardinalityNumber of instances for the interface.

Concurrency Show ConcurrencySemantics in the presence of multiple threads of control.

Operations Edit CompartmentServices specified by the interface.

Visibility Show VisibilitySpecifies how the interface is seen outside of the package in which it is defined; shows import status of the interface.

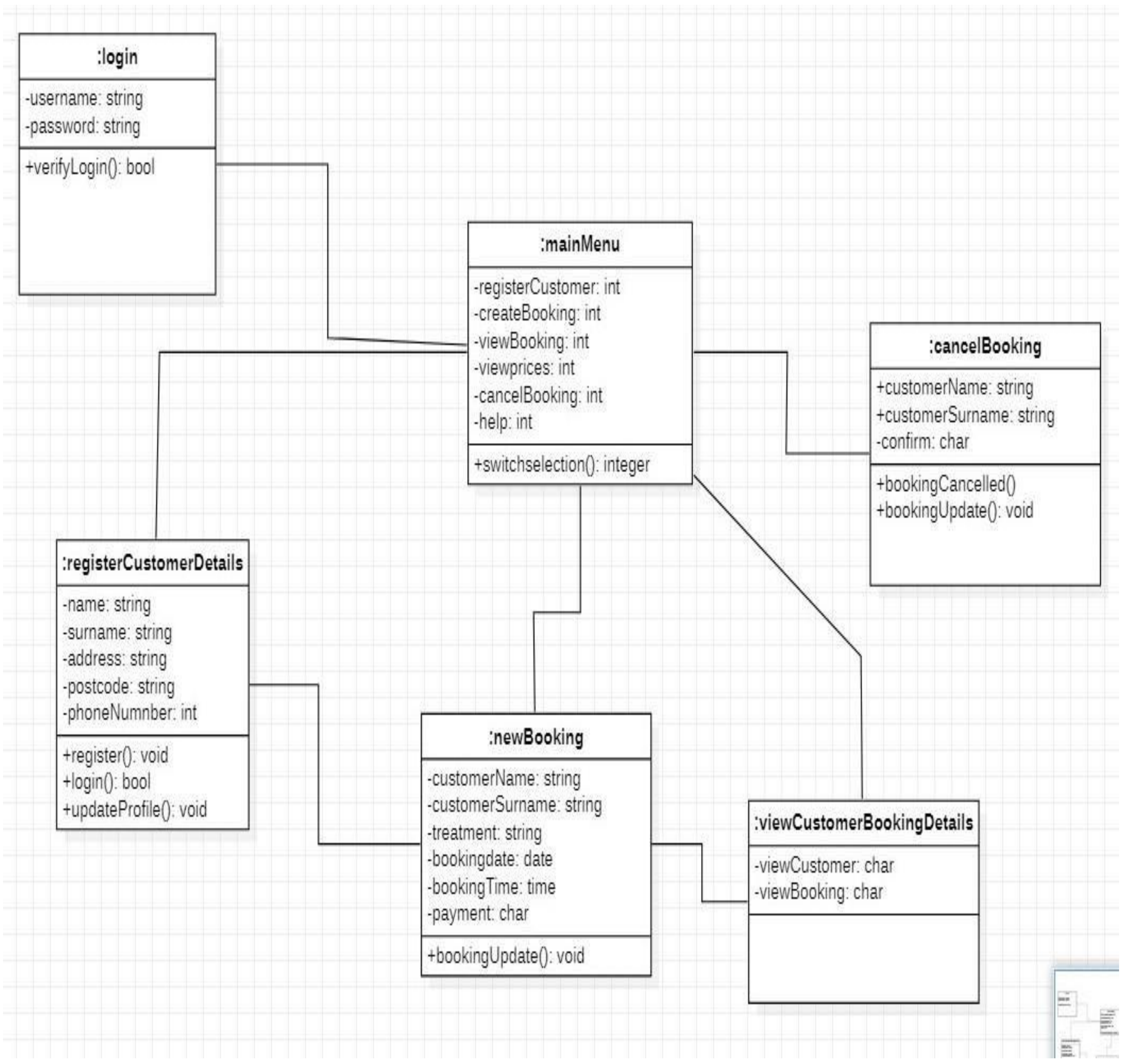


Figure 7.1:Class diagram

Experiment-VIII:Forward and Reverse Engineering

1. **Problem Statement:**To do forward and reverse engineering using class diagram

SOFTWARE REQUIREMENTS: Rational rose

THEORY

Modelling is important, but you have to remember that the primary product of a development team is software, not diagrams. Of course, the reason for creating models is to be able to deliver software that satisfies the evolving goals of its users and the business at the right time. For this reason, it's important that the models you create and the implementations you deploy map to one another and do so in a way that minimizes or even eliminates the cost of keeping your models and your implementation in sync with one another.

For some uses of the UML, the models you create will never map to code. For example, if you are modelling a business process using activity diagrams, many of the activities you model will involve people, not computers. In other cases, you'll want to model systems whose parts are, from your level of abstraction, just a piece of hardware (although at another level of abstraction, it's a good bet that this hardware contains an embedded computer and software)

In most cases though, the models you create will map to code. The UML does not specify a particular mapping to any object-oriented programming language, but the UML was designed with such mappings in mind. This is especially true for class diagrams, whose contents have a clear mapping to all the industrial-strength object-oriented languages, such as Java, C++, Smalltalk, Eiffel, Ada, ObjectPascal, and Forte. The UML was also designed to map to a variety of commercial object-based languages, such as Visual Basic.

Forward engineering is the process of transforming a model into code through a mapping to an implementation language. Forward engineering results in a loss of information, because models written in the UML are semantically richer than any current object-oriented programming language. In fact, this is a major reason why you need models in addition to code. Structural features, such as collaborations, and behavioral features, such as interactions, can be visualized clearly in the UML, but not so clearly from raw code.

To forward engineer a class diagram,

- Identify the rules for mapping to your implementation language or languages of choice. This is something you'll want to do for your project or your organization as a whole.
- Depending on the semantics of the languages you choose, you may want to constrain your use of certain UML features. For example, the UML permits you to model multiple inheritance, but Smalltalk permits only single inheritance. You can choose to prohibit developers from modeling with multiple inheritance (which makes your models language-dependent), or you can develop idioms that transform these richer features into the implementation language (which makes the mapping more complex).
- Use tagged values to guide implementation choices in your target language. You can

do this at the level of individual classes if you need precise control. You can also do so at a higher level, such as with collaborations or packages.
- Use tools to generate code.

Person.java

```
//Source file: C:\\CSE84\\Person.java

public class Person

{

   private int name;

   private int id;



   /**

   @roseuid 580EF81A0237

    */

   public Person()

   {



   }

}
```

Staff.java

```
//Source file: C:\\CSE84\\Staff.java

public class Staff extends Person

{

   private int classIncharge;
   private int subject;

   public SMSDatabase theSMSDatabase;

   public Customer();


   /**

   @roseuid 580EF81A0321

    */

   public Staff()

   {


   }


   /**

   @roseuid 57F3508C0245

    */

   public void updateDetails()

   {


   }


   /**

   @roseuid 57F350970060

    */

   public void updateReports()
```

```
  {


  }


  /**
  @roseuid 57F3509C01BF
   */
  public void updateNotifications()
  {


  }
}
/**


void Staff.generateReports(){


  }
void Staff.updateDetails(){



  }


 */
```

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 neighbouring 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.

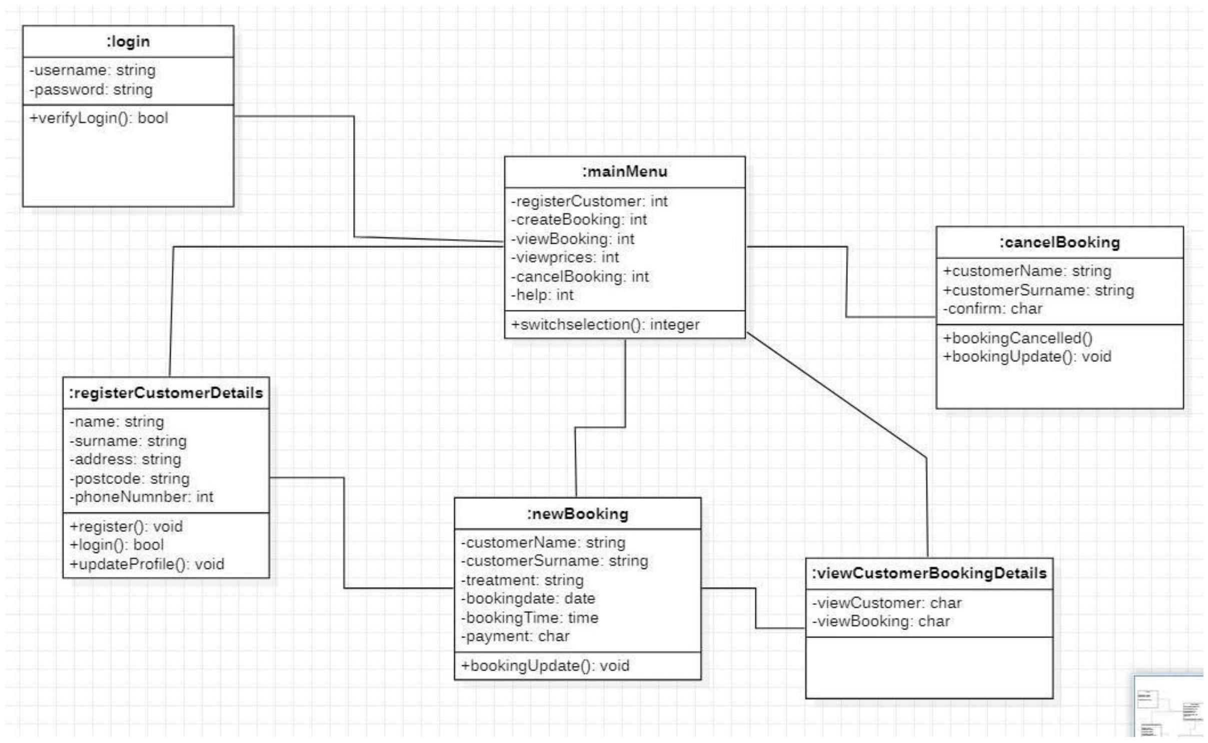


Figure 8.1:Class diagram

Experiment-IX:Component Diagram

1. Component Diagram

ProblemStatement: To draw Component diagram for School ManagementSystem

REQUIREMENTS:

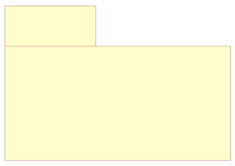
SOFTWARE **:** Rational rose

THEORY:

Component diagrams provide a physical view of the current model. A component diagram shows the organizations and dependencies among software components, including source code components, binary code components, and executable components. These diagrams also show the externally-visible behavior of the components by displaying the interfaces of the components. Calling dependencies among components are shown as dependency relationships between components and interfaces on other components. Note that the interfaces actually belong to the logical view, but they can occur both in class diagrams and in component diagrams.

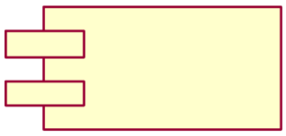
Component diagrams contain:

· Component packages: Component packages represent clusters of logically related components, or major pieces of your system. Component packages parallel the role played by logical packages for class diagrams. They allow you to partition the physical model of the system.

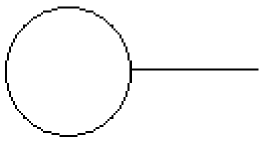


· Components: A component represents a software module (source code, binary code, executable, DLL, etc.) with a well-defined interface. The interface of a component is represented by one or several interface elements that the component provides. Components are used to show compiler and run-time dependencies, as well as interface and calling dependencies among software modules. They also show which components implement a specific class.

A system may be composed of several software modules of different kinds. Each software module is represented by a component in the model. To distinguish different kinds of components from each other, stereotypes are used.



Interfaces: An interface specifies the externally-visible operations of a class and/or component, and has no implementation of its own. An interface typically specifies only a limited part of the behavior of a class or component.



· Dependency relationships: A dependency is a relationship between two model elements in which a change to one model element will affect the other model element. Use a dependency relationship to connect model elements with the same level of meaning. Typically, on class diagrams, a dependency relationship indicates that the operations of the client invoke operations of the supplier.

We can create one or more component diagrams to depict the component packages and components at the top level of the component view, or to depict the contents of each component package. Such component diagrams belong to the component package that they depict.

A Component Package Specification enables you to display and modify the properties of a component package. Similarly, a Component Specification and a Class Specification enables you to display and modify the properties of a component and an interface, respectively. The information in these specifications is presented textually. Some of this information can also be displayed inside the icons representing component packages and components in component diagrams, and interfaces in class diagrams.

You can change properties of, or relationships among, component packages, components, and interfaces by editing the specification or modifying the icon on the diagram. The affected diagrams or specifications are automatically updated.

DIAGRAM:

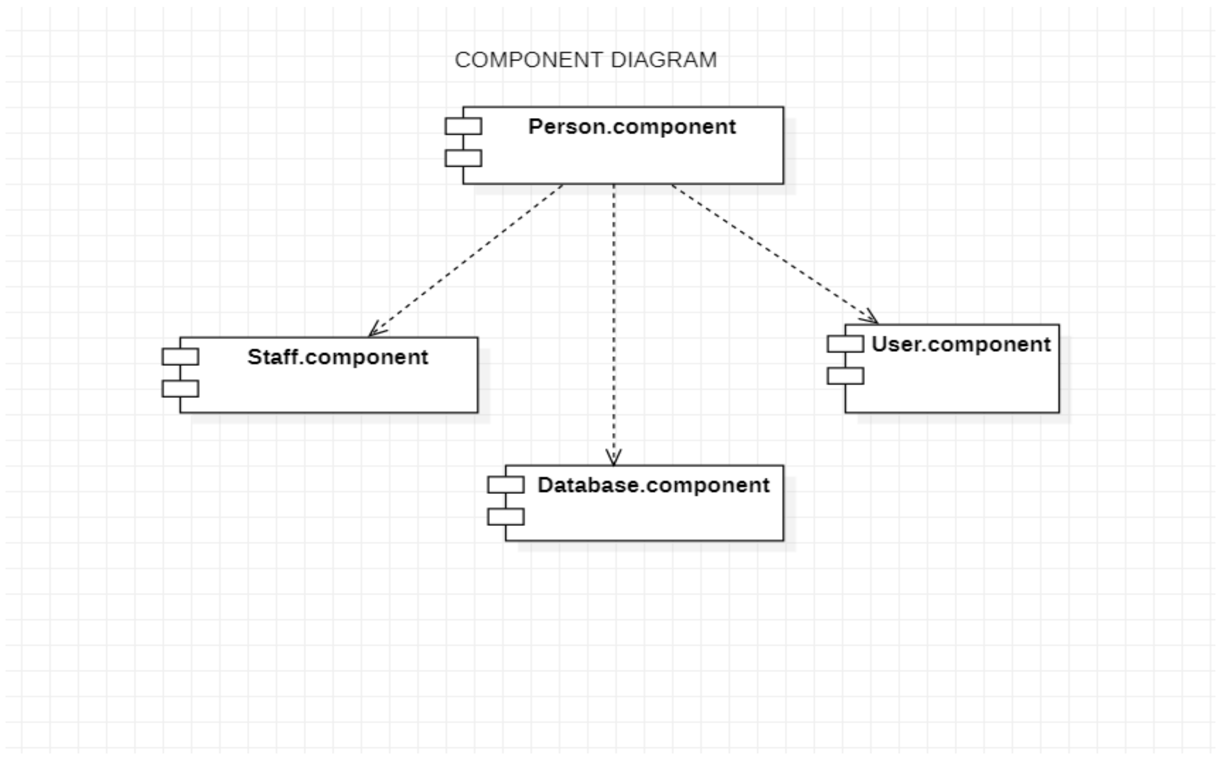


Figure 9.1: Component Diagram.

2. Deployment Diagram

ProblemStatement: To draw Deployment diagram for School ManagementSystem

REQUIREMENTS:

HARDWARE **:** PIII Processor, 512 MB RAM, 80GB

SOFTWARE **:** In Rational rose software using Deployment diagram tools

THEORY:

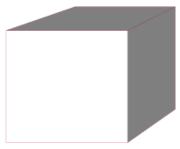
A deployment diagram shows processors, devices, and connections. Each model contains a single deployment diagram which shows the connections between its processors and devices, and the allocation of its processes to processors.

Processor Specifications, Device Specifications, and Connection Specifications enable you to display and modify the respective properties. The information in a specification is presented textually; some of this information can also be displayed inside the icons.

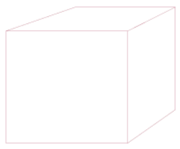
You can change properties or relationships by editing the specification or modifying the icon on the diagram. The deployment diagram specifications are automatically updated.

CONTENTS:-

Processor:-A processor is a hardware component capable of executing programs.



Devices:-A device is a hardware component with no computing power. Each device must have a name. Device names can be generic, such as "modem" or "terminal."



Connections:-A connection represents some type of hardware coupling between two entities. An entity is either a processor or a device. The hardware coupling can be direct, such as an RS232 cable, or indirect, such as satellite-to-ground communication. Connections are usually bi-directional.

DIAGRAM:

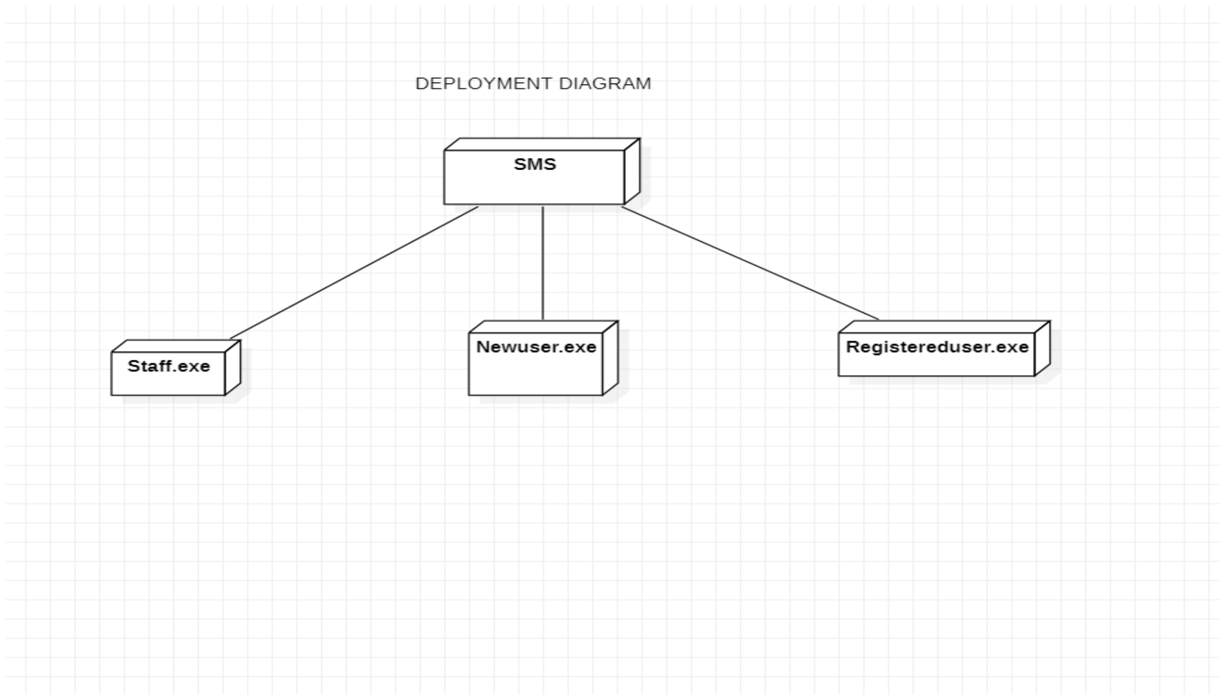


Figure 9.2: Deployment Diagram.

Experiment-X:COCOMO

1. **ProblemStatement**: To perform simple exercises on effort, cost and resource estimation

SOFTWARE REQUIREMENTS: cocomo software

Specific Instructional Objectives the student would be able to:

• Differentiate among organic, semidetached and embedded software projects.

• Explain basic COCOMO.

• Differentiate between basic COCOMO model and intermediate COCOMO model.

• Explain the complete COCOMO model.

Organic, Semidetached and Embedded software projects

Boehm postulated that any software development project can be classified into one of the following three categories based on the development complexity: organic, semidetached, and embedded. In order to classify a product into the identified categories, Boehm not only considered the characteristics of the product but also those of the development team and development environment. Roughly speaking, these three product classes correspond to application, utility and system programs, respectively. Normally, data processing programs are considered to be application programs. Compilers, linkers, etc., are utility programs. Operating systems and real-time system programs, etc. are system programs. System programs interact directly with the hardware and typically involve meeting timing constraints and concurrent processing.

Boehm's [1981] definition of organic, semidetached, and embedded systems are elaborated below.

Organic: A development project can be considered of organic type, if the project deals with developing a well understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar types of projects.

Semidetached: A development project can be considered of semidetached type, if the development consists of a mixture of experienced and inexperienced staff. Team members may have limited experience on related systems but may be unfamiliar with some aspects of the system being developed.

Embedded: A development project is considered to be of embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational procedures exist.

According to Boehm, software cost estimation should be done through three stages:

Basic COCOMO,

Intermediate COCOMO, and

Complete COCOMO.

Basic COCOMO Model

The basic COCOMO model gives an approximate estimate of the project parameters. The basic COCOMO estimation model is given by the following expressions:

Effort = a1 x (KLOC)a 2 PM

Tdev = b1 x (Effort)b 2 Months

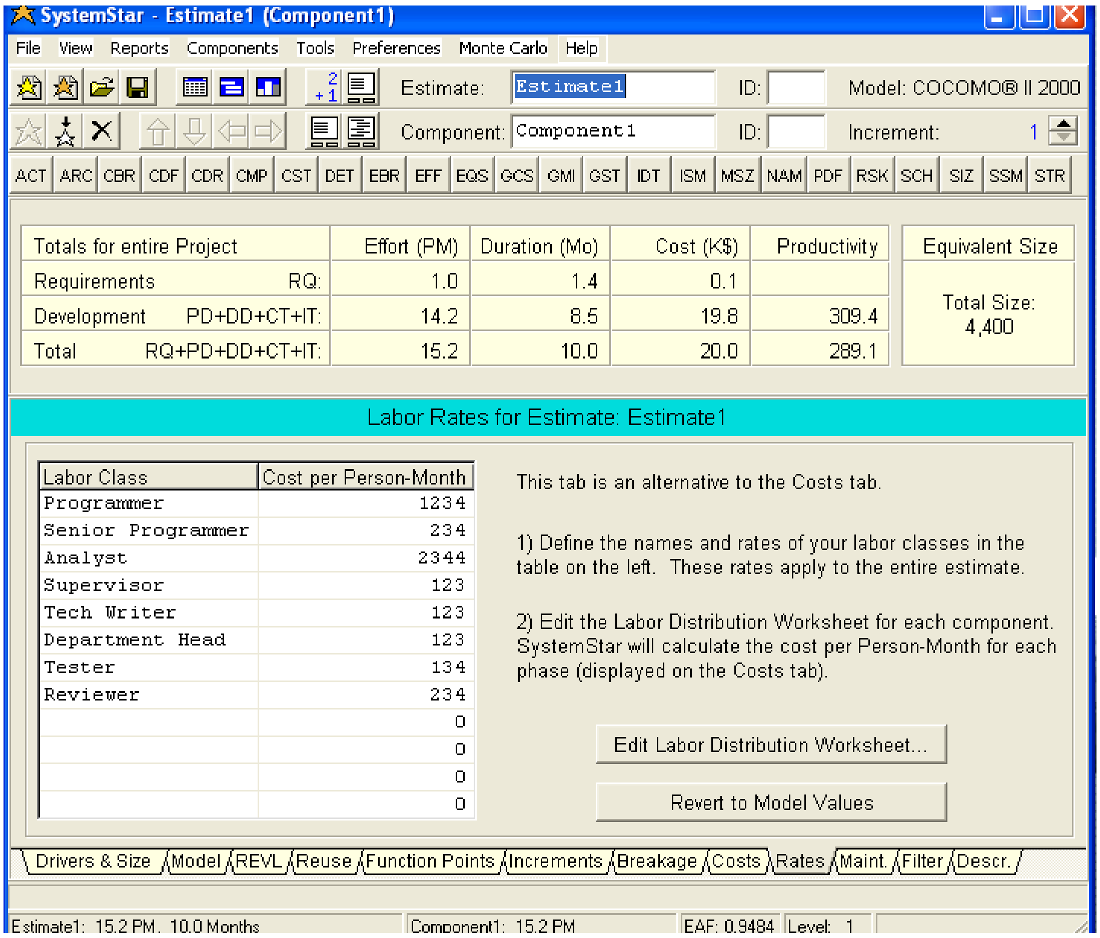
Where

• KLOC is the estimated size of the software product expressed in Kilo Lines of Code,

• a1, a2, b1, b2 are constants for each category of software products,

• Tdev is the estimated time to develop the software, expressed in months,

• Effort is the total effort required to develop the software product, expressed in person months (PMs).


SystemStar - Estimate1 (Component1)
File View Reports Components Tools Preferences Monte Carlo Help
Estimate: Estimate1 ID:
Model: COCOMO® II 2000
Component: Component1 ID:
Increment: 1
ACT ARC CBR CDF CDR CMP CST DET EBR EFF EQS GCS GMI GST IDT ISM MSZ NAM PDF RSK SCH SIZ SSM STR
Edit Labor Distribution Worksheet...
Revert to Model Values
Drivers & Size Model REVL Reuse Function Points Increments Breakage Costs Rates Maint. Filter Descr.

Totals for entire Project	Effort (PM)	Duration (Mo)	Cost (K$)	Productivity
Requirements RQ:	1.0	1.4	0.1	
Development PD+DD+CT+IT:	14.2	8.5	19.8	309.4
Total RQ+PD+DD+CT+IT:	15.2	10.0	20.0	289.1

Equivalent Size

Total Size: 4,400

Labor Rates for Estimate: Estimate1

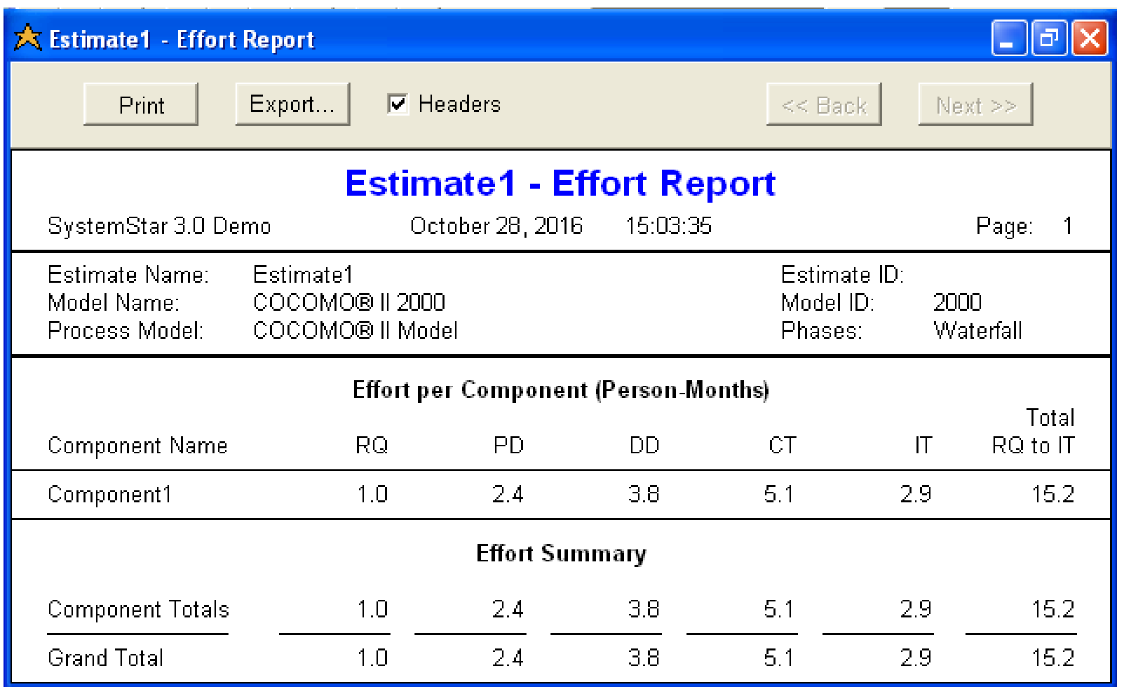
Labor Class	Cost per Person-Month
Programmer	1234
Senior Programmer	234
Analyst	2344
Supervisor	123
Tech Writer	123
Department Head	123
Tester	134
Reviewer	234
	0
	0
	0
	0

This tab is an alternative to the Costs tab.

1) Define the names and rates of your labor classes in the table on the left. These rates apply to the entire estimate.

2) Edit the Labor Distribution Worksheet for each component. SystemStar will calculate the cost per Person-Month for each phase (displayed on the Costs tab).

Estimate1: 15.2 PM, 10.0 Months | Component1: 15.2 PM | EAF: 0.9484 | Level: 1



Estimate1 - Effort Report

SystemStar 3.0 Demo October 28, 2016 15:03:35 Page: 1

Estimate Name:	Estimate1	Estimate ID:	
Model Name:	COCOMO® II 2000	Model ID:	2000
Process Model:	COCOMO® II Model	Phases:	Waterfall

Effort per Component (Person-Months)

Component Name	RQ	PD	DD	CT	IT	Total RQ to IT
Component1	1.0	2.4	3.8	5.1	2.9	15.2

Effort Summary

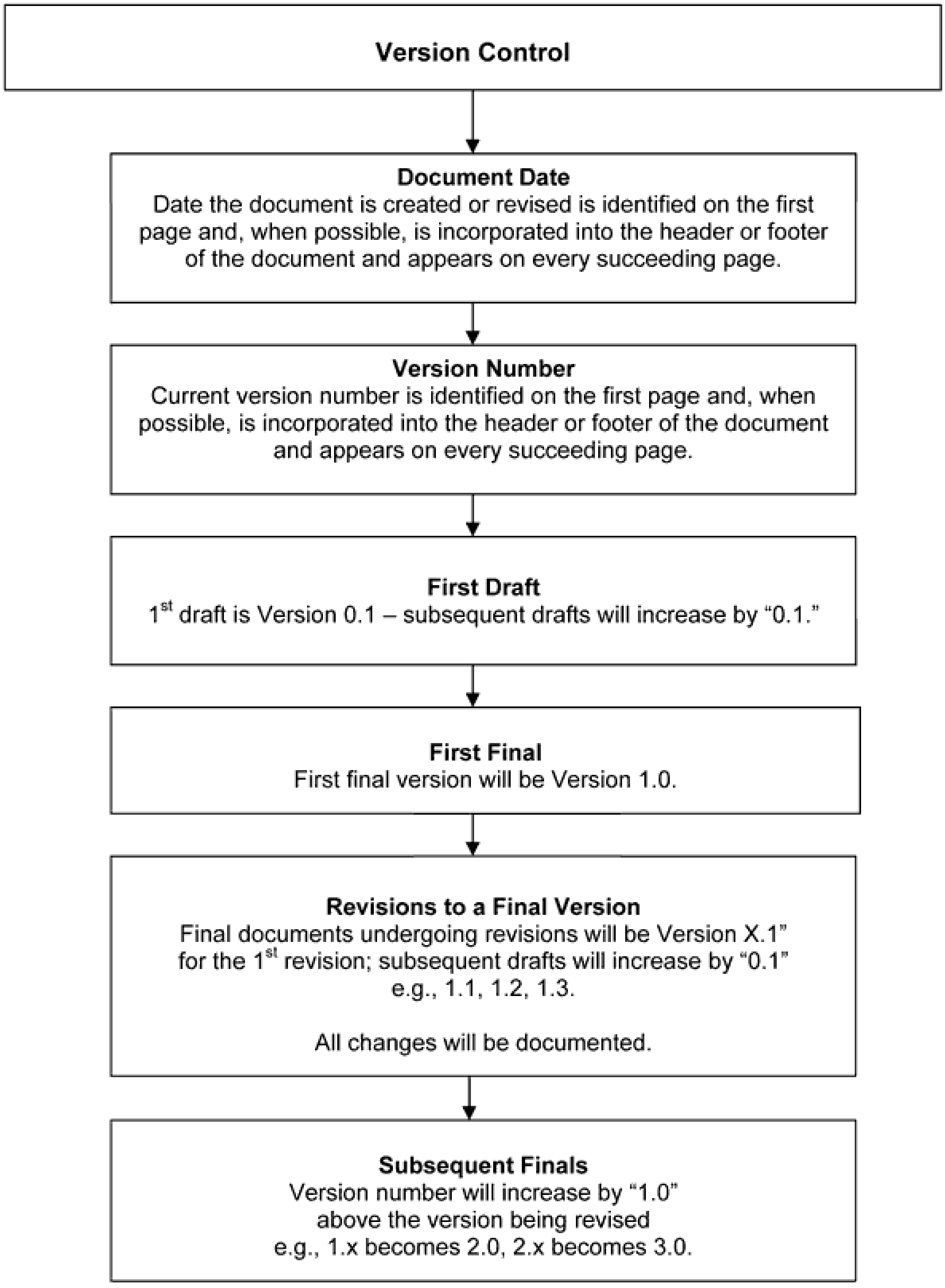
Component Totals	1.0	2.4	3.8	5.1	2.9	15.2
Grand Total	1.0	2.4	3.8	5.1	2.9	15.2

Experiment-XI:Familiarization of Software Configuration Management tool

1. **ProblemStatement**: To Familiarize with Software Configuration Management tool

- Document dates
    - The author of the document will ensure the date the document is created or revised is identified on the first page and, when possible, is incorporated into the header or footer of the document and appears on every succeeding page.
- Version numbers
    - The author of the document will ensure the current version number is identified on the first page and, when possible, is incorporated into the header or footer of the document and appears on every succeeding page.
- Draft document version number
    - The first draft of a document will be Version 0.1.
    - Subsequent drafts will have an increase of "0.1" in the version number, e.g., 0.2, 0.3, 0.4, …0.9, 0.10, 0.11.
- Final document version number and date
    - The author (or investigator) will deem a protocol or other document (consent/assent form, case report form, manual of procedures) final after all reviewers have provided final comments and the comments have been addressed.
    - The first final version of a document will be Version 1.0. Include the date when the document becomes final. Generally the final version is submitted to the Institutional Review Board and/or FDA.
    - Subsequent final documents will have an increase of "1.0" in the version number (1.0, 2.0, etc.).
- Final documents undergoing revisions
    - Final documents undergoing revisions will be Version X.1 for the first version of the revisions. While the document is under review, subsequent draft versions will increase by "0.1", e.g., 1.1, 1.2, 1.3, etc. When the revised document is deemed final, the version will increase by "1.0" over the version being revised, e.g., the draft 1.3 will become a final 2.0.
- Documenting substantive changes
    - A list of changes from the previous draft or final documents will be kept. The list will be cumulative and identify the changes from the preceding document versions. The list of changes made to a protocol and consent/assent should be submitted to the IRB with the final protocol and consent/assent documents.

Version Control Flow Chart



Version Control

Document Date
Date the document is created or revised is identified on the first page and, when possible, is incorporated into the header or footer of the document and appears on every succeeding page.

Version Number
Current version number is identified on the first page and, when possible, is incorporated into the header or footer of the document and appears on every succeeding page.

First Draft
1st draft is Version 0.1 – subsequent drafts will increase by "0.1."

First Final
First final version will be Version 1.0.

Revisions to a Final Version
Final documents undergoing revisions will be Version X.1" for the 1st revision; subsequent drafts will increase by "0.1" e.g., 1.1, 1.2, 1.3.

All changes will be documented.

Subsequent Finals
Version number will increase by "1.0" above the version being revised e.g., 1.x becomes 2.0, 2.x becomes 3.0.

Experiment-XII: Generate some examples of Test cases and verification

1. **ProblemStatement**: To create test cases and verify by comparing expected and actual results

SOFTWARE REQUIREMENT:MS Excel

Testing Test Cases and Sample Test Cases

- **Test Case** is a commonly used term for a specific test. This is usually the smallest unit of testing.
- A **Test Case** will consist of information such as requirements testing, test steps, verification steps, prerequisites, outputs, test environment, etc.
- A set of inputs, execution preconditions, and expected outcomes developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.
- A **test case** is a detailed procedure that fully tests a feature or an aspect of a feature. Whereas the test plan describes what to test, a test case describes how to perform a particular test. You need to develop a test case for each test listed in the test plan.
- A set of inputs, execution preconditions, and expected outcomes developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.
- **Test cases** should be written by a team member who understands the function or technology being tested, and each test case should be submitted for peer review.

To prepare these **Test Cases** each organization uses their own standard template, an ideal template is providing below to prepare **Test Cases**
The Name of this **Test Case Document** itself follows some name convention like below so that by seeing the name we can identify the Project Name and Version Number and Date of Release.

Project Name-----Test Cases-----Ver No-----Release Date

- The bolded words should be replaced with the actual Project Name, Version Number and Release Date. For eg. Bugzilla Test Cases 1.2.0.3 01_12_04

- On the Top-Left Corner we have company emblem and we will fill the details like Project ID, Project Name, Author of Test Cases, Version Number, Date of Creation and Date of Release in this Template.
- And we will maintain the fields Test Case ID, Requirement Number, Version Number, Type of Test Case, Test Case Name, Action, Expected Result, Cycle#1, Cycle #2, Cycle#3, Cycle#4 for each Test Case. Again this Cycle is divided into Actual Result, Status, Bug ID and Remarks.

- Test Case ID: To Design the **Test Case ID** also we are following a standard: If a test case belongs to application not specifically related to a particular Module then we will start them as TC001, if we are expecting more than one expected result for the same test case then we will name it as TC001.1. If a test case is related to Module then we will name it as M01TC001, and if a module is having a sub-module then we name that as M01SM01TC001. So that we can easily identify to which Module and which sub-module it belongs to. And one more advantage of this convention is we can easily add new test cases without changing all **Test Case Number** so it is limited to that module only
- Requirement Number: It gives the reference of Requirement Number in**SRS/FRD** for Test Case. For Test Case we will specify to which Requirement it belongs to. The advantage of maintaining this one here in Test Case Document is in future if a requirement will get change then we can easily estimate how many test cases will affect if we change the corresponding Requirement.
- Version Number: Under this column we will specify the Version Number, in which that particular test case was introduced. So that we can identify finally how many **Test Cases** are there for each Version.
- Type of Test Case: It provides the List of different type of **Test Cases like GUI, Functionality, Regression, Security, System, User Acceptance, Load, Performance** etc., which are included in the Test Plan. So while designing Test Cases we can select one of this option. The main objective of this column is we can predict totally how many GUI or Functionality test cases are there in each Module. Based on this we can estimate the resources.

- Test Case Name: This gives more specific name like particular Button or text box name, for which that particular Test Case belongs to. I mean to say we will specify the Object name for which it belongs to. For eg., OK button, Login form.
- Action: This is very important part in Test Case because it gives the clear picture what you are doing on the specific object. We can say the navigation for this Test Case. Based the steps we have written here we will perform the operations on the actual application.
- Expected Result: This is the result of the above action. It specifies what the specification or user expects from that particular action. It should be clear and for each expectation we will sub-divide that Test Case. So that we can specify pass or fail criteria for each expectation.
  Up to the above steps we will prepare the Test Case Document before seeing the actual application and based on System Requirement Specification/Functional Requirement Document and Use Cases. After that we will send this document to the concerned Test Lead for approval. He will review this document for coverage of all user Requirements in the Test Cases. After that he approved the Document. Now we are ready for testing with this Document and we will wait for the Actual Application. Now we will use the Cycle #1 parts.
  Under each Cycle#1 we are having Actual, Status, Bug ID and Remarks. Number of Cycles is based on the Organization. Some organizations document Three Cycles some organizations maintain the information for Four Cycles.
  But here I provided only one Cycle in this Template but you have to add more cycles based on your requirement.
- Actual: We will test the actual application against each Test Case and if it matches the Expected result then we will say it as **"As Expected"** else we will write the actually what happened after doing those action.
- Status: It simply indicates Pass or Fail status of that particular **Test Case**. If Actual and Expected both mismatch then the Status is **Fail** else it is **Pass**. For Passed Test Cases Bug ID should be null and for failed Test Cases Bug ID should be Bug ID in the Bug Report corresponding to that Test Case.
- Bug ID: This is gives the reference of Bug Number in Bug Report. So that Developer/Tester can easily identify the Bug associated with that Test Case.

Test cases are

Effective--Find Faults

Exemplary--represents others

Evolvable--easy to maintain

Economic--cheap to use

Test Case ID:		TC01		
Purpose:		User Registration		
Pre-condition:		User id must be at least 8 characters and max 10 characters. It must have combination of only letters and numbers. Password must be 8 characters at least and maximum 12, combination of letters, numbers and special characters.		
Input	Expected Output	Actual Output	Status	Remarks
Open browser and type the Url https://SMS.com	It should display the online SMS home page and show sign up and sign in options.	It should display the online SMS home page and show sign up and sign in options.	Pass	✓
Select sign up option.	It should display registration page and cursor should be positioned at username textbox.	It should display registration page and cursor should be positioned at username textbox.	Pass	✓
Enter valid username (say karthik123) and valid password (say bcd28@a and click on "register" button.	User should be registered successfully.	User should be registered successfully.	Pass	✓
Enter invalid username (say karu) and valid password (say bcd28@a) and click on "register" button.	User should not be registered and should be shown appropriate error message.	User should not be registered and should be shown appropriate error message.	Pass	Unsuccessful Registration. User name must have at least 1 number and minimum length 8.
Enter valid username (say karthik123) and invalid password (say bcd28@a) and click on "register" button.	User should not be registered and should be shown appropriate error message.	User should not be registered and should be shown appropriate error message.	Pass	Unsuccessful Registration. Password must have at least 1 special character and minimum length 8.
Enter valid username (say karthik123) and do not enter password and click on "register" button.	User should not be registered and should be shown appropriate error message.	User should not be registered and should be shown appropriate error message.	Pass	Unsuccessful Registration. Password is mandatory.
Do not enter user name and enter valid password (say bcd28@a) and	User should not be registered and should be shown appropriate error message.	User should not be registered and should be shown appropriate error message.	Pass	Unsuccessful Registration. User name is mandatory.

click on “register” button.				
	Executed History: Date: 03-06-2021 Result: Pass			

Test Case ID:		TC02		
Purpose:		SMS Login with valid user		
Pre-condition:		User should signup (existing user)		
Input	Expected Output	Actual Output	Status	Remarks
Open browser and type the Url https://SMS.com	It should display the online SMS home page and show sign up and sign in options.	It should display the online SMS home page and show sign up and sign in options.	Pass	✓
Select sign in option.	It should display Login page and cursor should be positioned at username textbox.	It should display Login page and cursor should be positioned at username textbox.	Pass	✓
Enter valid username (say karthik123) and valid password (say bcd28@a) and click on “Login” button.	User should be able to Login successfully.	User should be able to Login successfully.	Pass	✓
Enter invalid username (say karu) and valid password (say bcd28@a) and click on “Login” button.	User should not be able to login and should be shown appropriate error message.	User should not be able to login and should be shown appropriate error message.	Pass	Unsuccessful login.
Enter valid username (say karthik123) and invalid password (say bcd28@a) and click on “Login” button.	User should not be able to login and should be shown appropriate error message.	User should not be able to login and should be shown appropriate error message.	Pass	Unsuccessful login.
Enter valid username (say karthik123) and do not enter password and click on “Login” button.	User should not be able to login and should be shown appropriate error message.	User should not be able to login and should be shown appropriate error message.	Pass	Unsuccessful login.
Do not enter user name and enter valid password (say bcd28@a) and click on “Login” button.	User should not be able to login and should be shown appropriate error message.	User should not be able to login and should be shown appropriate error message.	Pass	Unsuccessful login.
	Executed History: Date: 03-06-2021 Result: Pass			

Test Case ID:		TC03		
Purpose:		Select Saloon and Language		
Pre-condition:		User should signup (existing user)		
Input	Expected Output	Actual Output	Status	Remarks
Open browser and type the Url https://SMS.com	It should display the online SMS home page and show sign up and sign in options.	It should display the online SMS home page and show sign up and sign in options.	Pass	✓
After Login click on “Select Saloon”.	It should display list of saloons.	It should display list of saloons.	Pass	✓
Click on saloon based on your choice by clicking drop down menu (list of all available saloons) button beside the saloon name and click on “Next”.	Then it should display “Select language”.	Then it should display “Select language”.	Pass	✓
Instead of selecting if user searches for a saloon and that saloon name is not available.	User should be shown popup message that place not found.	User should be shown popup message that place not found.	Pass	
Click on language based on your choice by clicking radio button beside the language name and click on “Next”.	User should be shown popup message that saloon and language selected.	User should be shown popup message that saloon and language selected.	Pass	✓
User did not select language and clicked on “Skip” button.	User should be shown appropriate popup message that only saloon selected.	User should be shown appropriate popup message that only saloon selected.	Pass	✓
	Executed History: Date: 03-06-2021 Result: Pass			

Test Case ID:		TC04		
Purpose:		Select Customer		
Pre-condition:		User should signup (existing user).		
Input	Expected Output	Actual Output	Status	Remarks
Open browser and type the Url https://SMS.com	It should display the online SMS home page and show sign up and sign in options.	It should display the online SMS home page and show sign up and sign in options.	Pass	✓
Click on location based on your choice clicking drop down menu (list of all locations present) button beside the location name and click on "Next".	User will be prompted with a pop up that whether he is sure to proceed with the given location.	User will be prompted with a pop up that whether he is sure to proceed with the given location.	Pass	✓
User clicks on "yes" when he is prompted with pop up.	Displays all the available Dealers and Storage godown in that location.	Displays all the available Dealers and Storage godown in that location.	Pass	✓
User clicks on "no" when he is prompted with pop up.	Go back to previous page.	Go back to previous page.	Pass	✓
	Executed History: Date: 03-06-2021 Result: Pass			

Test Case ID:		TC05		
Purpose:		Confirmation of Customer		
Pre-condition:		User should signup (existing user).		
Input	Expected Output	Actual Output	Status	Remarks
Open browser and type the Url https://SMS.com	It should display the online SMS home page and show sign up and sign in options.	It should display the online SMS home page and show sign up and sign in options.	Pass	✓
Select the appropriate Dealer and Storage and click on “Next”.	User will be prompted with a pop up that whether he is sure to proceed with the given details.	User will be prompted with a pop up that whether he is sure to proceed with the given details.	Pass	✓
User clicks on “yes” when he is prompted with pop up.	Displays the place and time of meeting and also sends message to the respective user.	Displays the place and time of meeting and also sends message to the respective user.	Pass	✓
User clicks on “no” when he is prompted with pop up.	Go back to previous page.	Go back to previous page.	Pass	✓
	Executed History: Date: 03-06-2021 Result: Pass			

Experiment-XIII: Demonstration on functional testing using Rational Functional Tester

1. ProblemStatement: To do functional testing using Rational Functional Tester

SOFTWARE REQUIREMENT: Rational functional tester tool

IBM Rational Functional Tester, referred hereafter as Functional Tester, is an advanced, automated functional and regression testing tool for testers and GUI developers who need superior control for testing Java, Microsoft Visual Studio .NET, and Web-based applications.

Functional Tester provides two user interface options: Microsoft Visual Studio and Eclipse. If your development team is building applications in Visual Studio and your test teams are more comfortable with Visual Basic .NET as a scripting language, you can install Functional Tester directly into your Visual Studio IDE shell and capture your test scripts in Visual Basic .NET.

The other option you have is to install Functional Tester into the Eclipse platform -- either your own Eclipse installation or one provided by Rational as part of the Functional Tester installation package. With the Eclipse variant of Functional Tester, test scripts are captured in Java code. Either way, you have the power of a full-up environment and the versatility of a commercial standard scripting language.

Functional Tester supports the functional regression testing of many types of applications, including those built with the following technologies:

• The Java platform

• Web (HTML)

• Microsoft® .NET

• Siebel

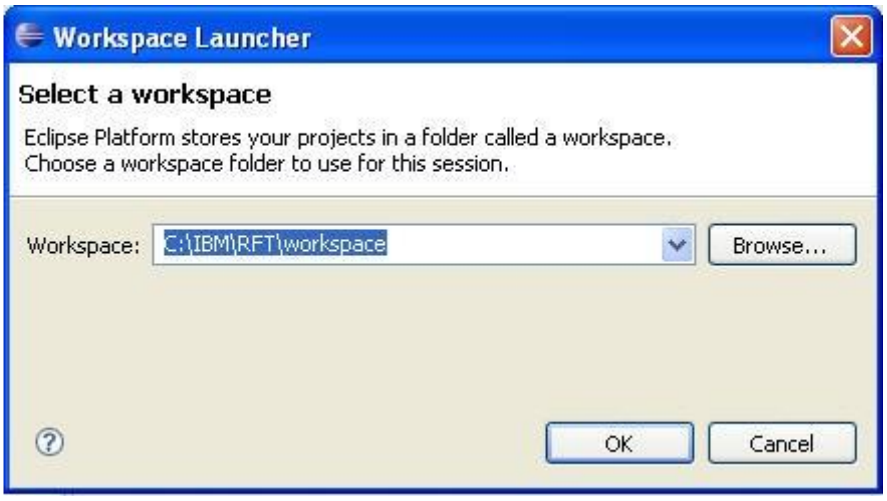
• SAPgui

• Terminal-based applications (3270, 5250, and VT100)

All of these application types can be tested with either the Eclipse-based or Visual Studio-based variant of Functional Tester. Although the Eclipse-based variant will be used for the remainder of this tutorial, everything you do here can also be done in the Visual Studio-based variant as well.

Launching Functional Tester

Functional Tester is started from the Windows Start menu by selecting Programs > IBM Rational Functional Tester > IBM Rational Functional Tester > Java Scripting. When you first start Functional Tester, you might be asked to select a workspace location, as shown in Figure 1. A workspace can be any directory location where your work is stored. If you are asked to choose a workspace when starting Functional Tester, you can choose the default or create a new one of your own.


Workspace Launcher

Select a workspace

Eclipse Platform stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\IBM\RFT\workspace

Preparing the test environment

This section walks you through the process of preparing your applicationenvironment for testing with Functional Tester. You will:

• Create a repository, or project, in which to store your Functional Testerartifacts

• Enable the application environment for testing

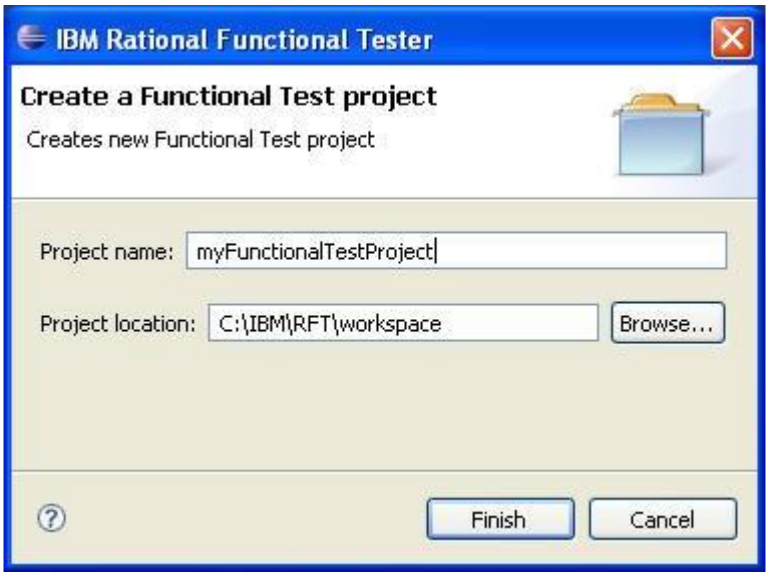
• Configure your application under test

Create a new functional test project

Begin by creating a Functional Test project named MyFunctionalTestProject tostore your test scripts, datapools, and object maps.

- Start Functional Tester and select a workspace as shown in the previous section, if you haven't already done so.
- From the workbench, select File > New > Functional Test Project. You should see the screen illustrated in Figure 3.
- Enter MyFunctionalTestProject as the project name and click Finish

Create Functional Test Project wizard


IBM Rational Functional Tester

Create a Functional Test project

Creates new Functional Test project

Project name: myFunctionalTestProject

Project location: C:\IBM\RFT\workspace

Enable the application environment

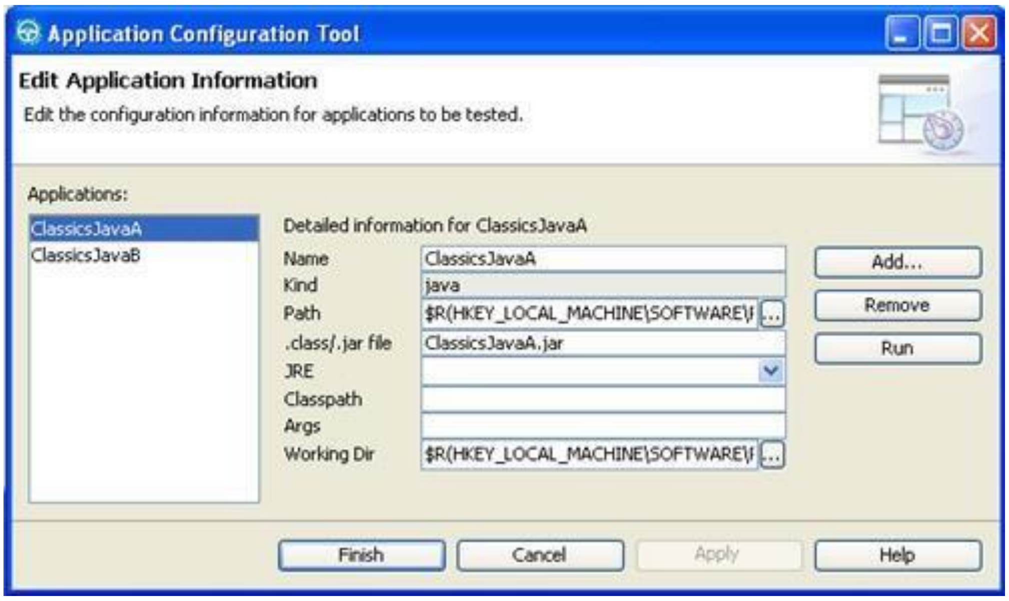
Next, you need to enable the application runtime environment. This allows Functional Tester to see inside the runtime to identify objects within the application under test.

Select Configure > Enable Environments for Testing. ... You will see three tabs in the Enable Environments window. These are the three classes or domains that might need to be enabled, depending on the type of application you are testing. Most likely, Internet Explorer is the default browser for test playback and that it is enabled.

Configure the application under test

Configuring the application under test does not actually affect or change your application in any way; it really just creates a shortcut, or pointer, in the Functional Tester environment that makes it easier to launch the application and makes the tests more portable to other test machines.

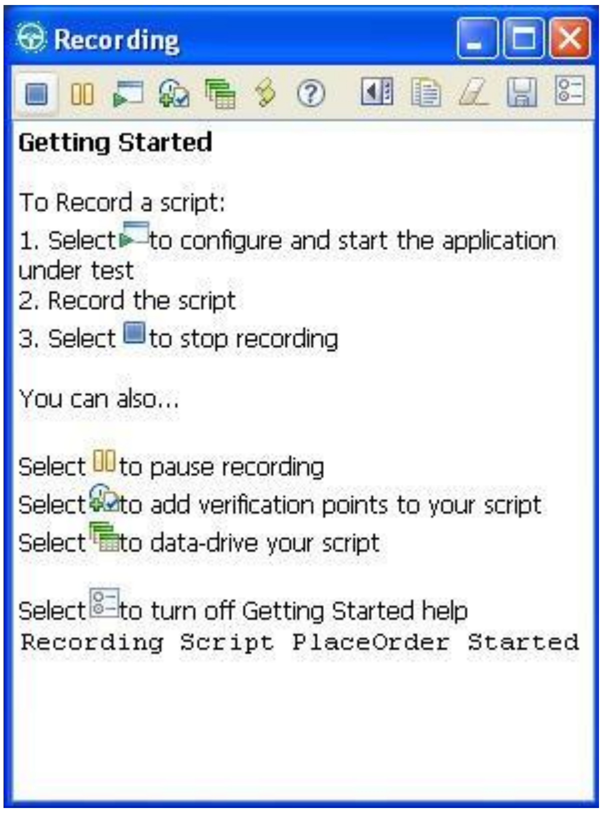
Select Configure > Configure Applications for Testing... In the Application Configuration Tool window A list of all the applications that have been configured for testing with Functional Tester is displayed. Functional Tester comes with a sample Java Swing application called ClassicsJava. Two builds of ClassicsJava are provided, and both are defined in Functional Tester automatically when it is installed. You can see here how run parameters such as the JAR file,.classpath, JRE, and working directory can be specified.


Application Configuration Tool
Edit Application Information
Edit the configuration information for applications to be tested.
Applications:
ClassicsJavaA
ClassicsJavaB
Detailed information for ClassicsJavaA
Add...
Remove
Run
Finish
Cancel
Apply
Help

Name	ClassicsJavaA
Kind	java
Path	$R(HKEY_LOCAL_MACHINE\SOFTWARE\I
.class/.jar file	ClassicsJavaA.jar
JRE	
Classpath	
Args	
Working Dir	$R(HKEY_LOCAL_MACHINE\SOFTWARE\I

Recording a test

Tests can be created by hand coding, recording, or a combination of the two. Even if you intend to do some custom coding, it is often easier to record a test first and then modify it than it is to write a test from scratch.



Getting Started

To Record a script:
1. Select to configure and start the application under test
2. Record the script
3. Select to stop recording

You can also...

Select to pause recording
Select to add verification points to your script
Select to data-drive your script

Select to turn off Getting Started help

```
Recording Script PlaceOrder Started
```

Launch the application to be tested

1. Click the Start Application button to launch the ClassicsJavaA - java application shortcut. ClassicsJava build A launches.

2. Keep in mind that anything you do within the ClassicsJava application is now being recorded. Use the Composers tree control to select Schubert > Symphonies Nos. 5 & 9.

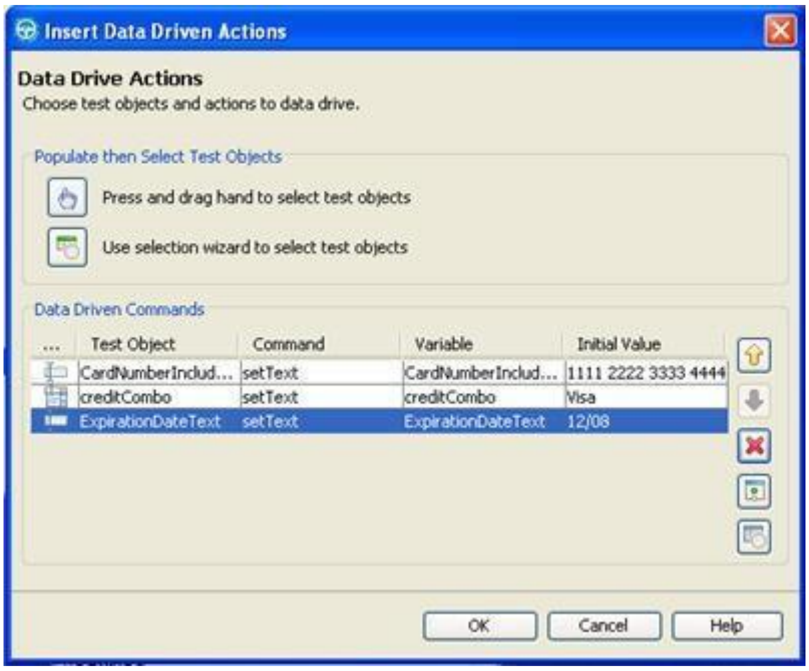
3.Click Place Order.

4.This sample application does not really check logins or passwords. For simplicity, accept the defaults when these are requested; however, check the Remember Password check box. This check box will illustrate the robustness of Functional Tester's test playback engine later in the tutorial.

5. Click OK to be taken to the Place an Order window.

Create a data-driven order

By default, all keyboard entry is captured in the test script. Functional Tester has a very powerful capability: it can separate the data entered by the user from the procedure and navigation commands of the test. The advantage in doing so is that the same test procedure can be used over and over with different sets of varied data, enabling you to reuse common tests and greatly reducing the time and effort involved in creating repetitive tests.


Insert Data Driven Actions
Data Drive Actions
Choose test objects and actions to data drive.
Populate then Select Test Objects
Press and drag hand to select test objects
Use selection wizard to select test objects
Data Driven Commands
OK
Cancel
Help

...	Test Object	Command	Variable	Initial Value
	CardNumberInclud...	setText	CardNumberInclud...	1111 2222 3333 4444
	creditCombo	setText	creditCombo	Visa
	ExpirationDateText	setText	ExpirationDateText	12/08

Verify dynamic data

Applications often respond to input with data and information that you cannot fully predict. For example, the confirmation message you'll receive in ClassicsJava contains a two-digit confirmation number. You might be able to predict that this will be a two-digit number in future orders, but the value of that number will be different each time you run the application. You need a way to verify the pattern of the message while being flexible about the actual value of the number.

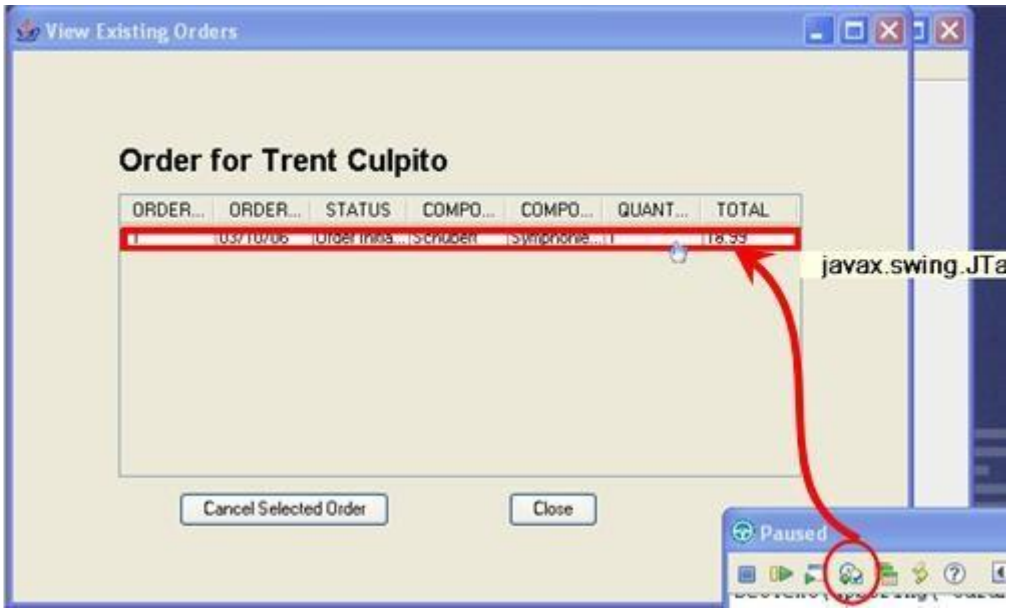
Verify static data

Finally, you need to create a verification point to verify that the order was processed correctly.

1. From the ClassicsJava menu, select Order > View Existing Order

Status. Click OK to log in again.

2. With the order displayed, click and drag the Verification Point and Action wizard onto the order information, so that the order is encased in a red square



View Existing Orders

Order for Trent Culpito

ORDER...	ORDER...	STATUS	COMPO...	COMPO...	QUANT...	TOTAL
1	03/10/06	Order Initia...	Schubert	Symphonie...	1	18.99

Add custom behavior to the test

As mentioned earlier, the test script is full Java code -- not JavaScript and not a proprietary language. This gives you extremely powerful capabilities to address unique needs in your tests. Also, Functional Tester provides a rich Application Programming Interface (API) through which you can access test objects and control test execution.

Remember that as you recorded the test, you first manually entered the credit card number and expiration date, then used the data-driven command wizard to enable Functional Tester to populate these fields from your datapool during playback. This has left some unnecessary commands in your test. These don't really hurt anything, but you can remove them at this time to see how the manual editing of script code works.

Experiment-XIV: Assessment of project quality using Rational Quality Manager

1. **ProblemStatement**: To do the assessment of project quality using Rational Quality Manager

SOFTWARE REQUIREMENT: Rational quality manager tool

Terminologies to use for RQM:

Test Plan

A document detailing a systematic approach to testing a system

Test Case:

a set of conditions or variables under which a tester will determine whether an application or software system is working correctly or not.

Test Script:

a set of instructions that will be performed on the system under test to test that the system functions as expected.

Test Environment:

a setup of software and hardware on which the testing team is going to perform the testing of the software product.

Defect:

is a condition in a software product which does not meet a software requirement or end user expectations

Plan your test

The test plan describes the overall scope of the test and the test schedule. The test plan provides a record of the test planning process. The plan also identifies test environments, entry and exit criteria, quality goals, and other aspects of the test.

When you create a test plan in Rational Quality Manager, you can base the plan on a default template or you can create a test plan. You can also define a default template or develop a new template. This flexibility makes creating test plans in Rational Quality Manager suitable for different teams that need to conduct one or numerous types of tests, such as functional verification tests, performance tests, system verification tests, globalization verification tests, and so on.

Integrate requirements from Rational RequisitePro

For many teams, requirements management helps to ensure life-cycle traceability. By linking the test cases in your test plan with requirements that product managers, program managers, or other members of the larger, cross-functional team have assigned, you can verify that all requirements are tested.

If you define your requirements in an external tool, such as Rational RequisitePro, you can import the requirements into Rational Quality Manager, where they are accessible in the Requirements view. You can also associate requirements with test plans and test cases or add requirements to your test plan manually.

When requirements are updated or deleted in the original requirements application, the status of the requirement in Rational Quality Manager is updated with the **Suspect** icon (⚠). You need to update the test plans or test cases that are associated with these suspect requirements.

Create a test case

Test cases are essential to maintaining high quality throughout the testing process. Test cases define what you need to validate to ensure that the system under test works correctly.

Rational Quality Manager provides a test-case template with some defined sections. For example, the template includes sections for preconditions and post-conditions and test execution and descriptions of expected results. You can also create a test case for a current test plan and associate your requirements with the test case. When you add a test script to a test case, it is listed in the Test Scripts section of the test case.

Associate a test script with your test case

If you have test scripts, you can add them to your test case. Of course, you can also create a script. This example provides all the steps to associate a test script with your test case.

Replace literal values with external test data in manual test scripts

To use external data in a manual test, you must associate test data with the manual test script. After you establish this association, you can use the values in the test data to replace literal values in the manual test script.

Before completing these steps, create test data and associate a test case with a manual test script.

Run a test case

Rational Quality Manager provides a variety of ways to run test cases, making the tool highly adaptable for many types of test teams. In this example, you run a Rational Functional Tester script.