**OBJECTIVES:**

* To capture the requirements specification for an intended software system
* To draw the UML diagrams for the given specification
* To map the design properly to code
* To test the software system thoroughly for all scenarios
* To improve the design by applying appropriate design patterns.

Draw standard UML diagrams using an UML modeling tool for a given case study and map design to code and implement a 3 layered architecture. Test the developed code and validate whether the SRS is satisfied.

1. Identify a software system that needs to be developed.
2. Document the Software Requirements Specification (SRS) for the identified system.
3. Identify use cases and develop the Use Case model.
4. Identify the conceptual classes and develop a Domain Model and also derive a Class Diagram from that.
5. Using the identified scenarios, find the interaction between objects and represent them using

UML Sequence and Collaboration Diagrams

1. Draw relevant State Chart and Activity Diagrams for the same system.
2. Implement the system as per the detailed design
3. Test the software system for all the scenarios identified as per the usecase diagram
4. Improve the reusability and maintainability of the software system by applying appropriate design patterns.
5. Implement the modified system and test it for various scenarios

**SUGGESTED DOMAINS FOR MINI-PROJECT:**

* 1. Passport automation system.
  2. Book bank
  3. Exam registration
  4. Stock maintenance system.
  5. Online course reservation system
  6. Airline/Railway reservation system
  7. Software personnel management system
  8. Credit card processing
  9. e-book management system
  10. Recruitment system
  11. Foreign trading system
  12. Conference management system
  13. BPO management system
  14. Library management system
  15. Student information system

**CASE TOOLS**

CASE tools known as Computer-aided software engineering tools is a kind of component-based development which allows its users to rapidly develop information systems. The main goal of case technology is the automation of the entire information systems development life cycle process using a set of integrated software tools, such as modeling, methodology and automatic code generation. Component based manufacturing has several advantages over custom development. The main advantages are the availability of high quality, defect free products at low cost and at a faster time. The prefabricated components are customized as per the requirements of the customers. The components used are pre-built, ready-tested and add value and differentiation by rapid customization to the targeted customers. However the products we get from case tools are only a skeleton of the final product required and a lot of programming must be done by hand to get a fully finished, good product.

# CHARACTERISTICS OF CASE:

Some of the characteristics of case tools that make it better than customized development are;

* It is a graphic oriented tool.
* It supports decomposition of process.

Some typical CASE tools are:

* Unified Modeling Language
* Data modeling tools, and
* Source code generation tools

# INTRODUCTION TO UML (UNIFIED MODELING LANGUAGE):

The UML is a language for specifying, constructing, visualizing, and documenting the software system and its components. The UML is a graphical language with sets of rules and semantics. The rules and semantics of a model are expressed in English in a form known as OCL (Object Constraint Language). OCL uses simple logic for specifying the properties

of a system. The UML is not intended to be a visual programming language. However it has a much closer mapping to object-oriented programming

languages, so that the best of both can be obtained. The UML is much simpler than other methods preceding it. UML is appropriate for modeling systems, ranging from enterprise information system to distributed web based application and even to real time embedded system. It is a very expensive language addressing all views needed to develop and then to display system even though understand to use. Learning to apply UML effectively starts forming a conceptual mode of languages which requires learning.

Three major language elements:

* UML basic building blocks
* Rules that dictate how this building blocks put together
* Some common mechanism that apply throughout the language The primary goals in the design of UML are:
  + 1. Provides users ready to use, expressive visual modeling language as well so they can develop and exchange meaningful models.
    2. Provide extensibility and specialization mechanisms to extend the core concepts.
    3. Be independent of particular programming languages and development processes.
    4. Provide formal basis for understanding the modeling language.
    5. Encourage the growth of the OO tools market.
    6. Support higher-level development concepts.
    7. Integrate best practices and methodologies.

Every complex system is best approached through a small set of nearly independent views of a model. Every model can be expressed at different levels of fidelity. The best models are connected to reality. The UML defines nine graphical diagrams:

1. Class diagram
2. Use-case diagram
3. Behavior diagram
   1. Interaction diagram
      1. sequence diagram
      2. collaboration diagram
   2. state chart diagram
   3. activity diagram
4. Implementation diagram

4.1component diagram 4.2deployment diagram

# UML class diagram:

The UML class diagram is also known as object modeling. It is a static analysis diagram. These diagrams show the static structure of the model. A class diagram is a connection of static model elements, such as classes and their relationships, connected as a graph to each other and to their contents.

# Use-case diagram:

The functionality of a system can be described in a number of different use-cases, each of which represents a specific flow of events in a system. It is a graph of actors, a set of use-cases enclosed in a boundary, communication, associations between the actors and the use-cases, and generalization among the use-cases.

# Behavior diagram:

It is a dynamic model unlike all the others mentioned before. The objects of an object oriented system are not static and are not easily understood by static diagrams. The behavior of the class’s instance (an object) is represented in this diagram. Every use-case of the system has an associated behavior diagram that indicates the behavior of the object. In conjunction with the use-case diagram we may provide a script or interaction diagram to show a time line of events. It consists of sequence and collaboration diagrams.

# Interaction diagram

It is the combination of sequence and collaboration diagram. It is used to depict the flow of events in the system over a timeline. The interaction diagram is a dynamic model which shows how the system behaves during dynamic execution.

# State chart diagram:

It consists of state, events and activities. State diagrams are a familiar technique to describe the behavior of a system. They describe all of the

possible states that a particular object can get into and how the object's state changes as a result of events that reach the object. In most OO techniques, state diagrams are drawn for a single class to show the lifetime behavior of a single object.

# Activity diagram:

It shows organization and their dependence among the set of components. These diagrams are particularly useful in connection with workflow and in describing behavior that has a lot of parallel processing. An activity is a state of doing something: either a real-world process, or the execution of a software routine.

# Implementation diagram:

It shows the implementation phase of the systems development, such as the source code structure and the run-time implementation structure. These are relatively simple high level diagrams compared to the others seen so far. They are of two sub-diagrams, the component diagram and the deployment diagram.

# Component diagram:

These are organizational parts of a UML model. These are boxes to which a model can be decomposed. They show the structure of the code itself. They model the physical components such as source code, user interface in a design. It is similar to the concept of packages.

# Deployment diagram:

The deployment diagram shows the structure of the runtime system. It shows the configuration of runtime processing elements and the software components that live in them. They are usually used in conjunction with deployment diagrams to show how physical modules of code are distributed on the system.

# NOTATION ELEMENTS:

These are explanatory parts of UML model. They are boxes which may apply to describe and remark about any element in the model. They

provide the information for understanding the necessary details of the diagrams.

# Relations in the UML:

are:

These are four kinds of relationships used in an UML diagram, they

* Dependency
* Association
* Generalization
* Realization

# Dependency:

It is a semantic relationship between two things in which a change one thing affects the semantics of other things. Graphically a dependency is represented by a non-continuous line.

# Association:

It is a structural relationship that describes asset of links. A link is being connected among objects. Graphically association is represented as a solid line possibly including label.

# Generalization:

It is a specialized relationship in which the specialized elements are substitutable for object of the generalized element. Graphically it is a solid line with hollow arrow head parent.

# Realization:

It is a semantic relation between classifiers. Graphically it is represented as a cross between generalization and dependency relationship.

# Where UML can be used:

UML is not limited to modeling software. In fact it is expressive to model non-software such as to show in structure and behavior of health case system and to design the hardware of the system.

# Conceptual model be UML:

UML you need to form the conceptual model of UML. This requires three major elements:

* + UML basic building blocks.
  + Rules that dictate how this building blocks are put together.
  + Some common mechanism that apply throughout the language.

Once you have grasped these ideas, you may be able to read. UML create some basic ones. As you gain more experience in applying conceptual model using more advanced features of this language.

# Building blocks of the UML:

The vocabulary of UML encompasses these kinds of building blocks.

# Use CASE definition:

**Description:**

A use case is a set of scenarios tied together by a common user goal. A use case is a behavioral diagram that shows a set of use case actions and their relationships.

# Purpose:

The purpose of use case is login and exchange messages between sender and receiver (Email client).

# Main flow:

First, the sender gives his id and enters his login. Now, he enters the message to the receiver id.

# Alternate flow:

If the username and id by the sender or receiver is not valid, the administrator will not allow entering and “Invalid password” message is displayed.

# Pre-condition:

A person has to register himself to obtain a login ID.

# Post-condition:

The user is not allowed to enter if the password or user name is not

valid.

# Class diagram:

**Description:**

* A class diagram describes the type of objects in system and various kinds of relationships that exists among them.
* Class diagrams and collaboration diagrams are alternate representations of object models.

During analysis, we use class diagram to show roles and responsibilities of entities that provide email client system behaviors design. We use to capture the structure of classes that form the email client system architecture.

# A class diagram is represented as:

<<Class name>>

<<Attribute 1>>

<<Attribute n>>

<<Operation ()>>

# Relationship used:

A change in one element affects the other

# Generalization:

It is a kind of relationship

# State chart:

**Description:**

* The state chart diagram made the dynamic behavior of individual classes.
* State chart shows the sequences of states that an object goes through events and state transitions.
* A state chart contains one state ‘start’ and multiple ‘end’ states.

The important objectives are:

# Decision:

It represents a specific location state chart diagram where the work flow may branch based upon guard conditions.

# Synchronization:

It gives a simultaneous workflow in a state chart diagram. They visually define forks and joints representing parallel workflow.

# Forks and joins:

* + A fork construct is used to model a single flow of control.
  + Every work must be followed by a corresponding join.
  + Joints have two or more flow that unit into a single flow.

# State:

A state is a condition or situation during a life of an object in which it satisfies condition or waits for some events.

Transition:

It is a relationship between two activities and between states and activities.

# Start state:

A start state shows the beginning of a workflow or beginning of a state machine on a state chart diagram.

# End state:

It is a final or terminal state.

# Activity diagram Description:

Activity diagram provides a way to model the workflow of a development process. We can also model this code specific information such as class operation using activity diagram. Activity diagrams can model different types of diagrams. There are various tools involved in the activity diagram.

# Activity:

An activity represents the performance of a task on duty. It may also represent the execution of a statement in a procedure.

# Decision:

A decision represents a condition on situation during the life of an object, which it satisfies some condition or waits for an event.

# Start state:

It represents the condition explicitly the beginning of a workflow on an activity.

# Object flow:

An object on an activity diagram represents the relationship between activity and object that creates or uses it.

# Synchronization:

It enables us to see a simultaneous workflow in an activity.

# End state:

An end state represents a final or terminal state on an activity diagram or state chart diagram.

# Sequence diagram:

**Description:**

A sequence diagram is a graphical view of scenario that shows object interaction in a time based sequence what happens first what happens next. Sequence diagrams are closely related to collaboration diagram.

The main difference between sequence and collaboration diagram is that sequence diagram show time based interaction while collaboration diagram shows objects associated with each other.

The sequence diagram for the e-mail client system consists of the following objectives:

# Object:

An object has state, behavior and identity. An object is not based is referred to as an instance.

The various objects in e-mail client system are:

* + - User
    - Website
    - Login
    - Groups

# Message icon:

A message icon represents the communication between objects indicating that an action will follow. The message icon is the horizontal solid arrow connecting lifelines together.

# Collaboration diagram:

**Description:**

Collaboration diagram 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. Collaboration diagram is an interaction diagram that shows the order of messages that implement an operation or a transaction.

Collaboration diagram shows object s, their links and their messages. They can also contain simple class instances and class utility instances.

During, analysis indicates the semantics of the primary and secondary interactions. Design, shows the semantics of mechanisms in the logical design of system.

Toggling between the sequence and collaboration diagrams

When we work in either a sequence or collaboration diagram, it is possible to view the corresponding diagram by pressing F5 key.

# CONCLUSION:

Thus the study for case tools was done.

# BLOOD BANK MANAGENEMT

# AIM:

To create an automated system to perform the Passport Process.

# PROBLEM STATEMENT:

Passport Automation System is used in the effective dispatch of passport to all of the applicants. This system adopts a comprehensive approach to minimize the manual work and schedule resources, time in a cogent manner. The core of the system is to get the online registration form (with details such as name, address etc.,) filled by the applicant whose testament is verified for its genuineness by the Passport Automation System with respect to the already existing information in the database.

**(II)SOFTWARE REQUIREMENT SPECIFICATION**

# 2.1SOFTWARE INTERFACE

* + **Front End Client** - The applicant and Administrator online interface is built using

JSP and HTML. The Administrators's local interface is built using Java.

* + **Web Server** - Glassfish application server(Oracle Corporation).
  + **Back End** - Oracle database.

# 2.2HARDWARE INTERFACE

The server is directly connected to the client systems. The client systems have access to the database in the server.

# (III) USECASE DIAGRAM :

# 

# Fig.3.USECASE DIAGRAM FOR PASSPORT AUTOMATION SYSTEM

s

**RESULT:**

Thus the mini project for passport automation system has been successfully executed and codes are generated.