DESIGN AND IMPLEMENTATION

CHAPTER OUTLINE

After studying this chapter, the reader will be able to understand the

- Introduction
- Object Oriented Design using the UML
- Design Models
- Design Patterns
- Implementation Issues
- Open Source Development

INTRODUCTION

Software design and implementation is the stage in the software engineering process at which we design the software either using structured design or an object oriented design approach and only then an executable software system is developed. These activities are invariably interleaved. Software design is a creative activity in which you identify software components and their relationships, based on a customer's requirements. Implementation is the process of realizing the design as a program. The level of detail in the design depends on the type of system being developed and methods such as plan driven or agile approach that you are using. Design and implementation are closely linked and you should take implementation issues into account when developing a design. While implementing a system, the most important decisions that have to be made at an early stage of software project is whether or not you should buy or build application. It is now possible to buy off the shelf system that can be adapted to the user's requirement. When you are developing a system using COTS components then design process becomes concerned with how to use the configuration features of that system to deliver the system requirements.

OBJECT ORIENTED DESIGN USING THE UML

Object oriented design is a means of designing software so that the fundamental components in the design represent objects with their own private states and operations rather than functions. In this design, the executing system is made up of interacting objects that maintain their own local state and provide operations on that state. This process involves designing the objects classes and the relationship between these classes. It is a part of object oriented development where an object oriented strategy is used throughout the development process.

Software that are developed using object oriented design are easier to change than the systems developed using functional approaches. In this design objects include both data and operations to manipulate that data so they may be easy to understood and modified as standalone entities.

There are a variety of different object-oriented design processes that depend on the organization using the process. Common activities in these processes include:

- Define the context and modes of use of the system;
- Design the system architecture;
- Identify the principal system objects;
- Develop design models;
- Specify object interfaces.

System Context and Interactions

In this step clear understanding of the system being designed and its external environment is identified to know how to provide the required system functionality and how to structure the system to communicate with its environment. This gives you better understanding of system boundary. System context normally show that the environment includes several other automated systems however they do not show the systems in the environment and system that is being specified. External system might produce data or consume data from the system. When we model the interaction of system with its environment, you should use an abstract approach that does not include too much detail and for this we design the use case diagram.

Use case Model

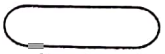
A use cases diagram is a graphical depiction of the interactions among the elements of a system and its external entities called actors. A use case is a methodology used in system analysis to identify clarifies & organize system requirements.

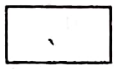
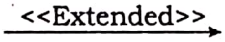
Use case diagram are consists of actors use cases and their relationships. The diagram is used to model of system/sub system of an application. A single use case diagram captures a particular functionality of a system.

Purposes of use case diagram

- Used together requirement of a system.
- Used to get an outside view of a system.
- Identify external and internal factors influencing the system.
- Show the interacting among the requirements are actors.
- It is focus on functional requirement.

Symbols

1.  <Role/Actor> (Sticky figure)	An actor is an external entity that interacts with the system. It can be human or machine eg: paypal, e-sewa, sms server etc.
2.  [Use case] or action	A use case is an action that the user performs within the system or the system performs on the user.
3. [Straight line]	Straight line is used to show communication (interaction between the actor and user.

4.  [Bounary]	System boundary is used to represent the system scope.
5.  <include>	The dependency might exist among a main use case and other use case such that the later use case is the outcome of the main use case. It is shown using
6.  <<Extended>>	The new use case is called the extended use case. The extended use case usually arguments the base use case with additional of behavior. It is shown using .

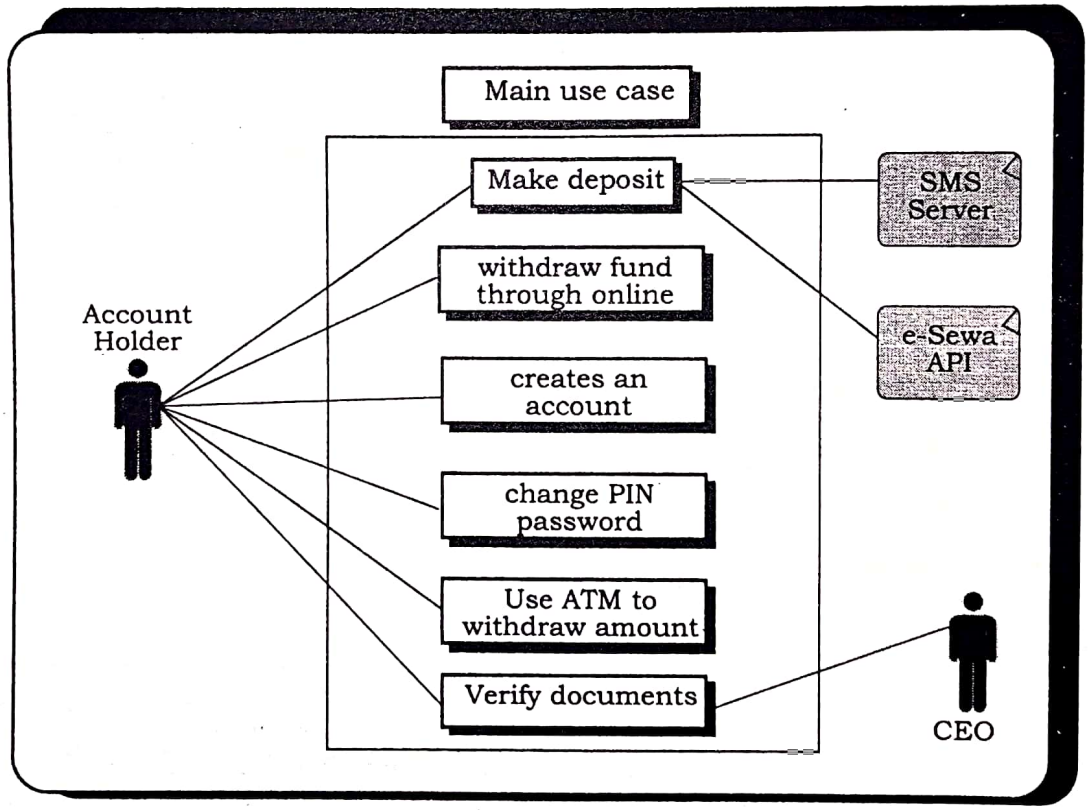


Fig: Bank ATM system use case diagram

Architectural Design

After identifying the interactions between the software system and its environment, that information is used for designing systems architecture. Architectural design is the process of decomposing the system into subsystems and establishing the relationship between subsystems. For this any of the architectural pattern is chosen from repository architecture, client server architecture or layered architecture.

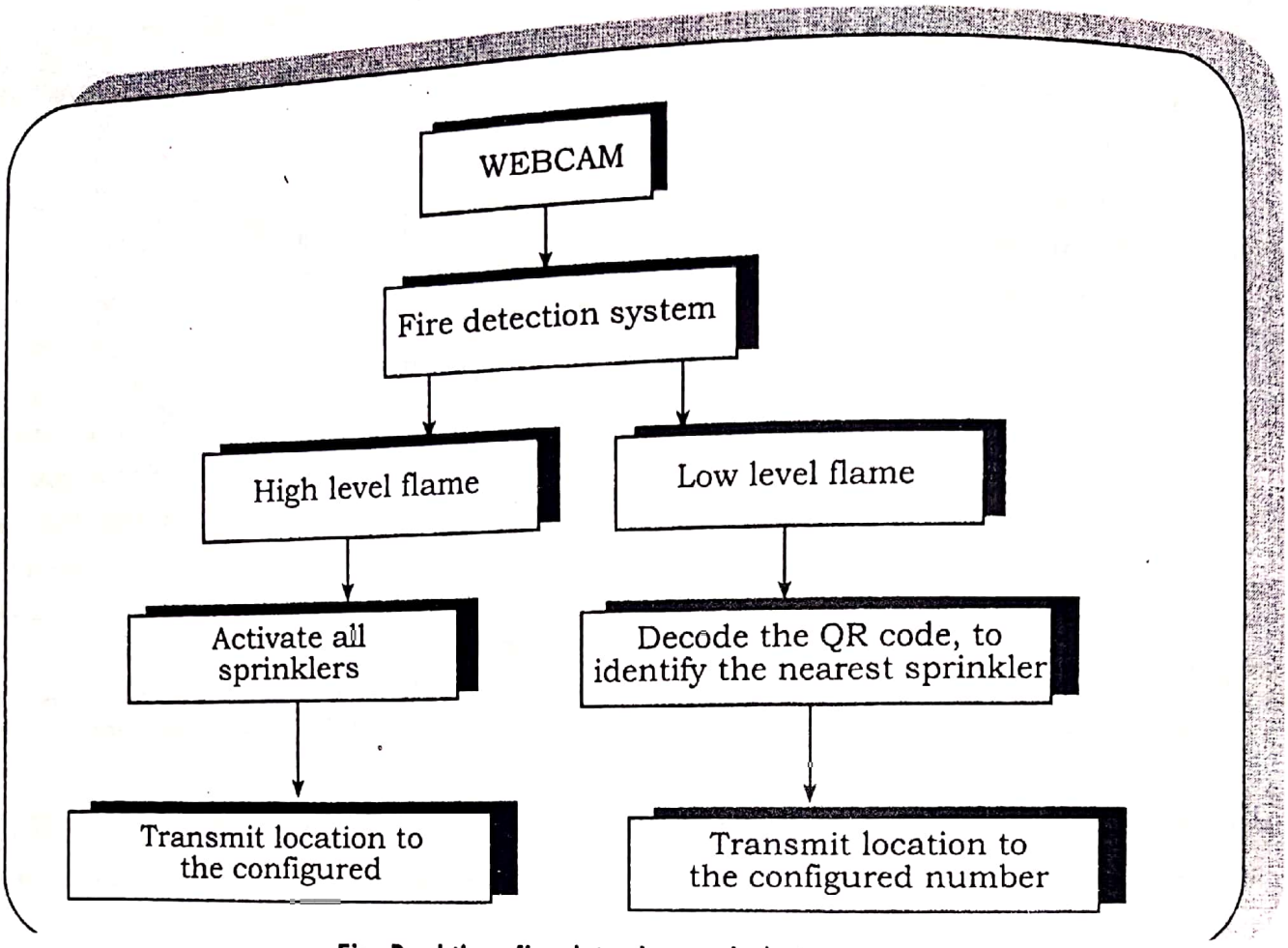


Fig: Real time fire detection and alert system

Object Class Identification

From architectural design you already have idea about the necessary objects in the system. Object identification is the important and critical issue in the object oriented design. There is no 'magic formula' for object identification. It relies on the skill, experience and domain knowledge of system designers. Object identification is an iterative process. The objects identified in the object-oriented analysis phases are grouped into classes and refined so that they are suitable for actual implementation. To identify object classes different methods can be used such as: grammatical analysis of natural language description of a system, tangible entities in the software application under consideration are identified, Use a scenario-based analysis and the objects, attributes and methods in each scenario are identified.

The functions of this stage are

- Identifying and refining the classes in each subsystem or package
- Defining the links and associations between the classes
- Designing the hierarchical associations among the classes, i.e., the generalization/specialization and inheritances
- Designing aggregations

This can be modeled using object and class diagram of the system.

DESIGN MODELS

A design model in Software Engineering is an object-based picture or pictures that represent the system. It shows the object or object classes in a system. Design models show the associations or relationship between system entities that act as bridge between requirement specification and implementation of system. The design model that we choose to design a system depends on the type of system being developed. For example we design a sequential data processing system in a different way than real time systems. In object oriented design we use UML diagram. In UML, a range of different models may be produced. These includes, static models such as: class models, generalization model, association model and dynamic models such as: sequence models, state machine models etc.

1. **Static /Structural Model:** Structural diagrams, as the name suggest show how the system is structured, including the classes, objects, packages, components, etc. in the system and the relationships between those elements. The structural diagrams represent the static aspect of the system. These static aspects represent those parts of a diagram, which forms the main structure and are therefore stable. This model describes the static structure of the system in terms of object classes and their relationships. Class diagram is the best example of static model.

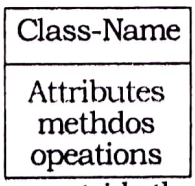
   Class Diagram: The class diagram is a static diagram which represents the static view of an application calls diagram is not only used for visualizing, describing and documenting different aspects of a system but also for constructing executable code of software application. The class diagram describes the attributes & method (operations) of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object oriented system because they are the only UML diagrams which can be mapped directly with object oriented languages. The class diagram shows a collection of classes' interfaces associations, collaborations and construction. It is also knows as structural diagram.

Purpose of class diagram

- Analysis and design of the static view of application.
- Describe responsibilities of a system.
- Base for component & development diagrams.
- Forward and reserve engineering.

Symbols

i. **Class symbol**



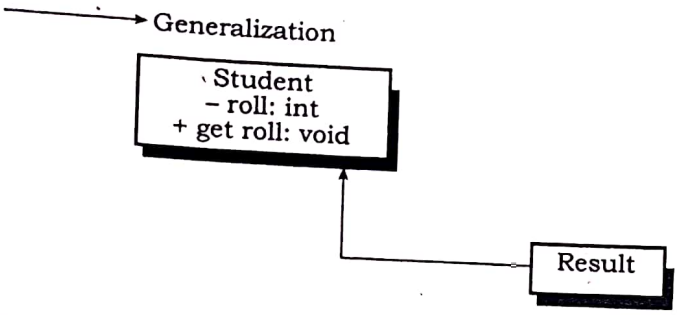
ii. **Visibility**

+ : public: accessed from outside the class

- : private: only be accessed from within the class

: protected: accessed from with in the class or any descendant

iii. **Relationships**



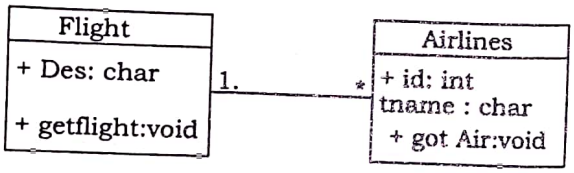
iv. ..... →

Dependency

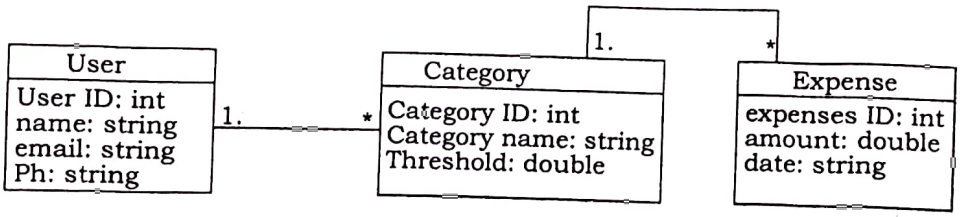
If main class has change than it change into child class but if child class has change but could not change in main class.

v. 1. *

Collaboration

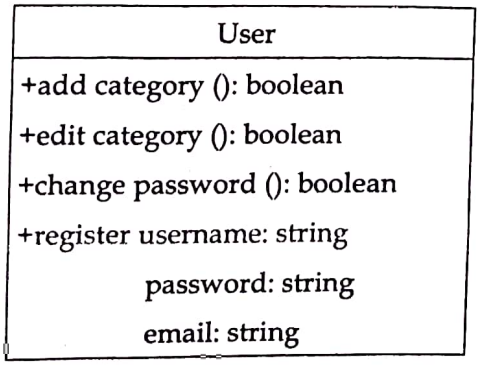


- **Attributes:** The attributes are the characteristics of any object properly qualifier unique objects. It is the property of an object. They are represented each on a new line.



- **Methods:** Methods are specified using the form:

Method.name(): type



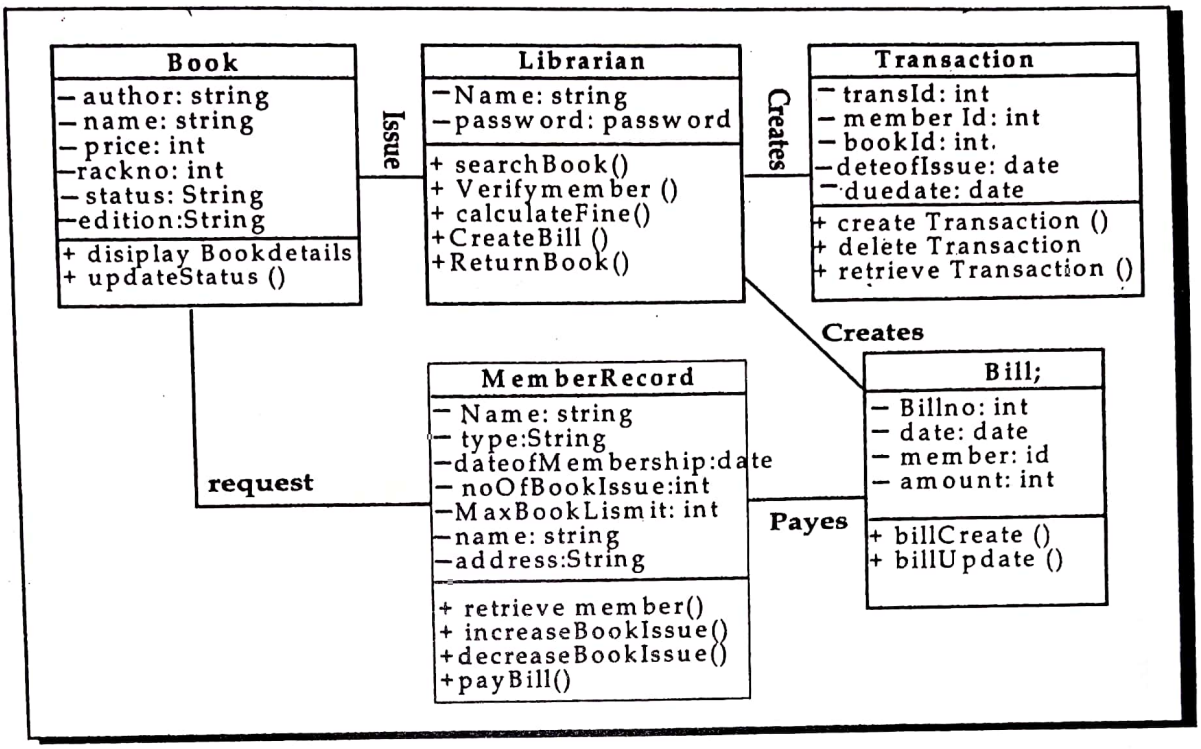


Fig: Class diagram of Library Management System

Component diagram

Component diagram are used to model physical aspects of a system. Physical aspects are the elements like executables, libraries, files, documentations etc. which resides in a node. So component diagrams are used to visualize the organization and relationships among components in a system. these diagrams are also used to make executable system. It has a higher level of abstraction then a class diagram usually a components is implemented by one or more classes of run time.

Purpose of component diagram

- Visualize the component of a system.
- Construct executables by using forward and reverse engineering.
- Describes the organizations and relationships of the component.

Component communicable with each other using interface. The interfaces are liked using connectors.

Example:

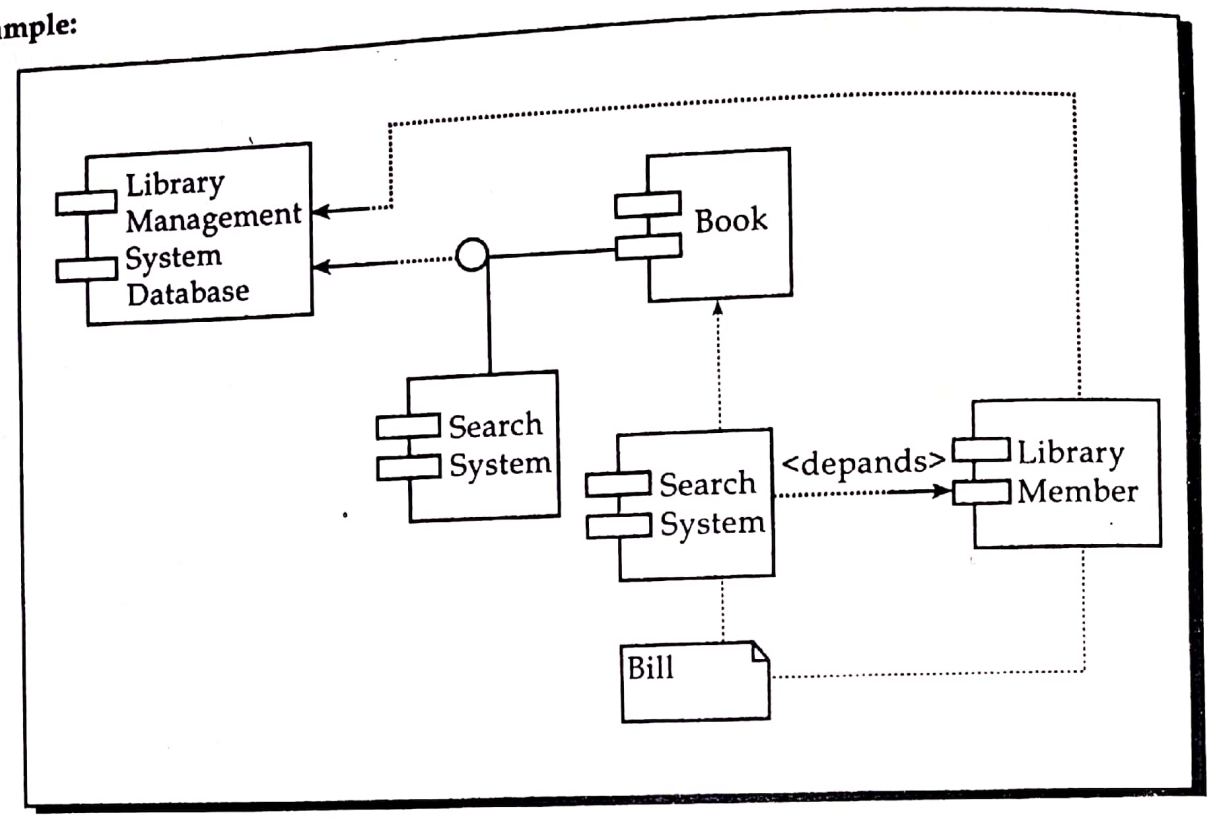


Fig: Component Diagram of Library Management System

2. **Dynamic/ Behavioral Model:** This model shows the dynamic structure of the system that shows the interactions between the system objects. Dynamic aspect can be described as the changing/moving parts of a system. Behavioral diagram assist in understanding and communicating how elements interact and collaborate to provide the functionality of a system. Use case diagram, activity diagram, sequence diagram etc. are examples of dynamic models.

Activity Diagram: Activity diagram are typically used for business process modeling. For modeling the logic captured by a single use case or usage scenario or for modeling the detailed logic of a business rule.

Activity diagram is basically a flow chart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. So the control flow is drawn from one activity to another, this flow can be sequential or concurrent.

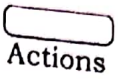
Purpose of activity diagram:

- Draw the activity flow of a system.
- Describe the sequence from one activity to another.
- Describe the parallel, branched and concurrent flow of a system.

Symbols

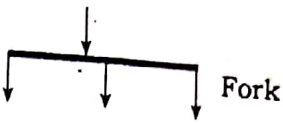
i.

ii.



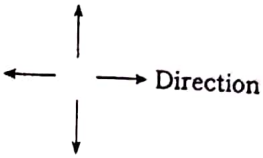
iii.

iv.



v.

vi.



vii.

End node

Example:

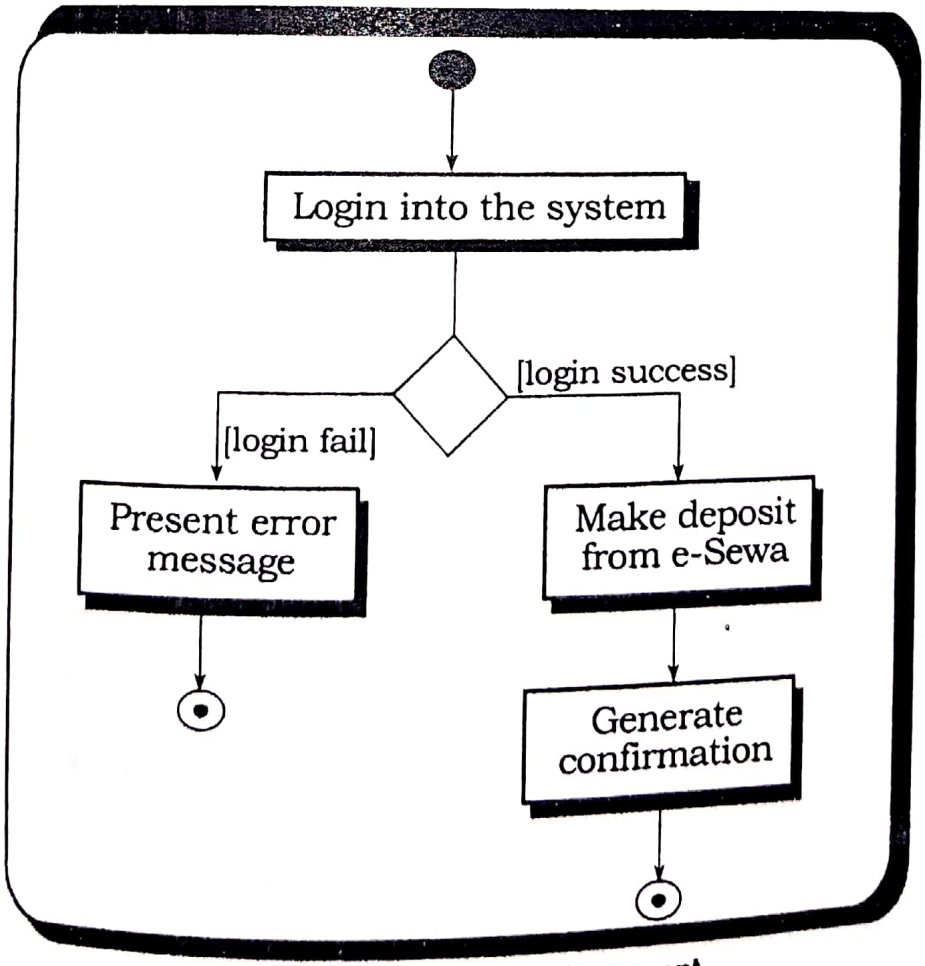


Fig: Activity Diagram of E-payment

Sequence Diagram

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop inside the rectangle in brackets. A sequence diagram is a form of interaction diagram which shows objects as lifelines running down the page with their interaction overtime represented message drawn as arrows from the source lifelines to the target lifeline. Sequence diagram are good at showing which objects communicate with which other objects and what messages trigger those communications sequence diagram are not intended for showing complex procedural logic.

Purpose of sequence diagram

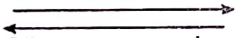
- Used primarily to show the interactions between objects in the sequential orders.
- Transition from requirements expressed to the neat level of refinement.
- Used to document how object in existing software currently interact.

Symbols

i.

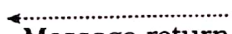
Object lifeline

ii. Synchronous/Asynchronous



Message passing

iii.



Message return

Example:

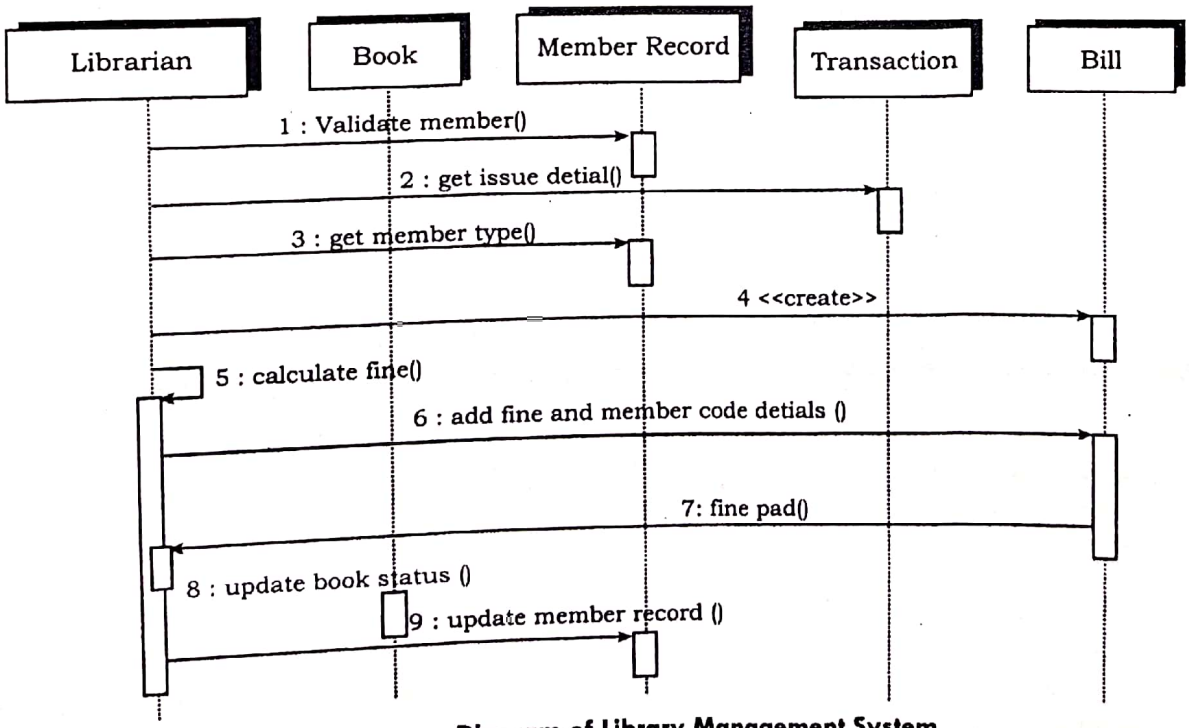


Fig: Sequence Diagram of Library Management System

3. **Interface specification:** This is an important concept of object oriented design process. Interface specifications provide the standardized mechanism in which subsystems can effectively communicate with each other and enable them to operate as independent modules. In this interface between the components in the design is specified so that objects and sub systems can be designed in parallel and they can communicate with each other. Interface design concerned with specifying the detail of the interface to an object or to a group of objects. It defines the signatures and semantics of the services that are provided by the object or by a group of objects. The UML uses class diagrams for interface specification but Java may also be used.

DESIGN PATTERNS

Design patterns are typical solutions to common problems in software design. Each pattern is like a blueprint that you can customize to solve a particular design problem in your code. Patterns are a toolkit of solutions to common problems in software design. They define a common language that helps your team communicate more efficiently. It is a description or template for how to solve a problem that can be used in many different situations. Patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.

Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved.

Design patterns can speed up the development process by providing tested, proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Reusing design patterns helps to prevent subtle issues that can cause major problems and improves code readability for coders and architects familiar with the patterns. Design patterns are usually associated with object oriented design that relies on object characteristics such as inheritance and polymorphism. Hence pattern is a way of reusing abstract knowledge about a problem and its solution.

IMPLEMENTATION ISSUES

A critical stage of SDLC is the system implementation in which we create an executable code of the software using high level programming language from scratch or using off the shelf systems to meet the specific requirements of the user. Some aspect of implementation issues are:

1. **Reuse:** Now days, modern software are constructed by reusing existing components or systems. When you are developing software, you should make as much use as possible of existing code. Software was the reuse of functions and objects in programming language libraries. Software reuse is possible at different levels.

- The object level: At this level, you directly reuse objects from a library rather than writing the code yourself.
- The component level: Components are collections of objects and object classes that you reuse in application systems. You can reuse the component by adding some code of your own.
- The system level: At this level, you reuse entire application systems.

2. **Configuration Management:** Configuration management is the name given to the general process of managing a changing software system. During the development process, you have to keep track of the many different versions of each software component in a configuration management system. The aim of configuration management is to support the system integration process so that all developers can access the project code and documents in a controlled way, find out what changes have been made, and compile and link components to create a system.
3. **Host-target Development:** Generally, the software is developed in one computer and execute on the same computer as the software development environment. Host target development is the methodology in which software is developed on one computer but runs on a separate computer.

OPEN SOURCE DEVELOPMENT

The term open source was coined within the software development industry and refers to something that anyone can inspect, modify, and share. It represents a specific approach to creating computer programs which celebrates the values of collaboration, transparency, and community-oriented development. Open source software is usually a free software product, where developers have access to the source code. They can enhance the program's performance, add some features, and fix errors.

Open source development is an approach to software development in which the source code of a software system is published and volunteers are invited to participate in the development process. It is the technique which advocates that source code should not be proprietary but rather should always be available for users to examine and modify as they wish. Open source software extended this idea by using the Internet to recruit a much larger population of volunteer developers. Examples of open source product are Linux, apache web server, MYSQL database management system, Java etc.

A fundamental principle of open-source development is that source code should be freely available; this does not mean that anyone can do as they wish with that code.

- Legally, the developer of the code (either a company or an individual) still owns the code. They can place restrictions on how it is used by including legally binding conditions in an open source software license.
- Some open source developers believe that if an open source component is used to develop a new system, then that system should also be open source.
- Others are willing to allow their code to be used without this restriction. The developed systems may be proprietary and sold as closed source systems.

Licensing issues are important because if you use open source software as part of a software product then you may be obliged by terms of license to make your own product open source. Most open source licenses are derived from one of three general models:

- The GNU General Public License (GPL). This is a so-called 'reciprocal' license that means that if you use open source software that is licensed under the GPL license, then you must make that software open source.
- The GNU Lesser General Public License (LGPL) is a variant of the GPL license where you can write components that link to open source code without having to publish the source of these components.

The Berkley Standard Distribution (BSD) License. This is a non-reciprocal license, which means you are not obliged to re-publish any changes or modifications made to open source code. You can include the code in proprietary systems that are sold.

1. What do you mean by object oriented design? Explain the importance of UML in object oriented design.
2. Explain common activities in object oriented design process.
3. What is object class identification? Explain with example.
4. What do you mean by design models? Differentiate between structural model and behavioral model.
5. What are design patterns? Explain with examples.
6. Explain different implementation issues in brief.
7. What is open source software? Explain the concept of open source development and open source licensing.

❑❑❑