➢Software ➢Design

Software Design

Object Oriented Design

Object oriented design is the result of focusing attention not on the function performed by the program, but instead on the data that are to do manipulated by the program. Thus, it is orthogonal to function oriented design.

Object Oriented Design begins with an examination of the real world "things" that are part of the problem to be solved. These things (which we will call objects) are characterized individually in terms of their attributes and behavior.

Software Design

➢ Basic Concepts

Object Oriented Design is not dependent on any specific implementation language. Problems are modeled using objects. Objects have:

- Behavior (they do things)
- State (which changes when they do things)

Software Design

The various terms related to object design are:

i. Objects

The word "Object" is used very frequently and conveys different meaning in different circumstances. Here, meaning is an entity able to save a state (information) and which offers a number of operations (behavior) to either examine or affect this state. An object is characterized by number of operations and a state which remembers the effect of these operations.

Software Design

ii. Messages

Objects communicate by message passing. Messages consist of the identity of the target object, the name of the requested operation and any other operation needed to perform the function. Message are often implemented as procedure or function calls.

iii. Abstraction

In object oriented design, complexity is managed using abstraction. Abstraction is the elimination of the irrelevant and the amplification of the essentials.

Software Design

iv. Class

In any system, there shall be number of objects. Some of the objects may have common characteristics and we can group the objects according to these characteristics. This type of grouping is known as a class. Hence, a class is a set of objects that share a common structure and a common behavior.

We may define a class "car" and each object that represent a car becomes an instance of this class. In this class "car", Indica, Santro, Maruti, Indigo are instances of this class as shown in fig. 20.

Classes are useful because they act as a blueprint for objects. If we want a new square we may use the square class and simply fill in the particular details (i.e. colour and position) fig. 21 shows how can we represent the square class.

Software Design

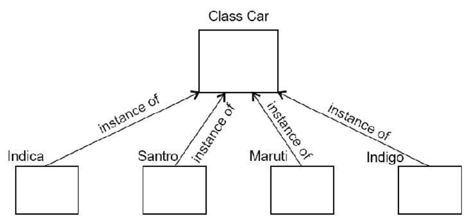


Fig.20: Indica, Santro, Maruti, Indigo are all instances of the class "car"

Software Design

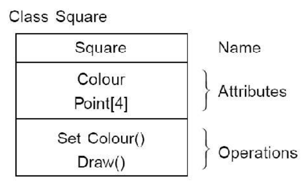


Fig. 21: The square class

Software Design

v. Attributes

An attributes is a data value held by the objects in a class. The square class has two attributes: a colour and array of points. Each attributes has a value for each object instance. The attributes are shown as second part of the class as shown in fig. 21.

vi. Operations

An operation is a function or transformation that may be applied to or by objects in a class. In the square class, we have two operations: set colour() and draw(). All objects in a class share the same operations. An object "knows" its class, and hence the right implementation of the operation. Operation are shown in the third part of the class as indicated in fig. 21.

Software Design

vii. Inheritance

Imagine that, as well as squares, we have triangle class. Fig. 22 shows the class for a triangle.

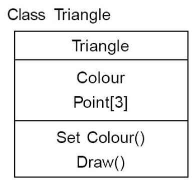


Fig. 22: The triangle class

Software Design

Now, comparing fig. 21 and 22, we can see that there is some difference between triangle and squares classes.

For example, at a high level of abstraction, we might want to think of a picture as made up of shapes and to draw the picture, we draw each shape in turn. We want to eliminate the irrelevant details: we do not care that one shape is a square and the other is a triangle as long as both can draw themselves.

To do this, we consider the important parts out of these classes in to a new class called Shape. Fig. 23 shows the results.

Software Design

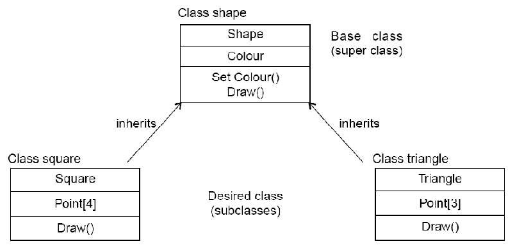


Fig. 23: Abstracting common features in a new class

This sort of abstraction is called inheritance. The low level classes (known as subclasses or derived classes) inherit state and behavior from this high level class (known as a super class or base class).

Software Design

viii. Polymorphism

When we abstract just the interface of an operation and leave the implementation to subclasses it is called a polymorphic operation and process is called polymorphism.

ix. Encapsulation (Information Hiding)

Encapsulation is also commonly referred to as "Information Hiding". It consists of the separation of the external aspects of an object from the internal implementation details of the object.

x. Hierarchy

Hierarchy involves organizing something according to some particular order or rank. It is another mechanism for reducing the complexity of software by being able to treat and express sub-types in a generic way.

Software Design

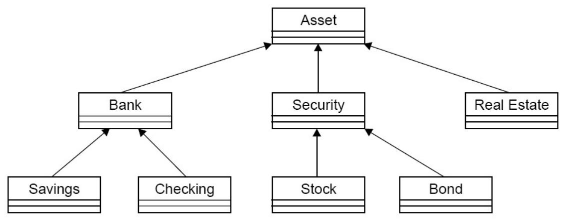


Fig. 24: Hierarchy

Software Design

➢ Steps to Analyze and Design Object Oriented System

There are various steps in the analysis and design of an object oriented system and are given in fig. 25

Software Design

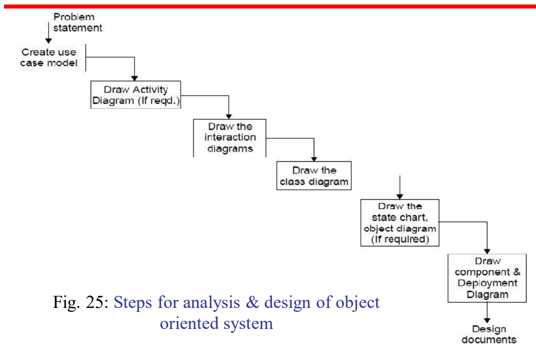

Problem statement
Create use case model
Draw Activity Diagram (If reqd.)
Draw the interaction diagrams
Draw the class diagram
Draw the state chart, object diagram (If required)
Draw component & Deployment Diagram
Design documents

Fig. 25: Steps for analysis & design of object oriented system

Software Design

i. Create use case model

First step is to identify the actors interacting with the system. We should then write the use case and draw the use case diagram.

ii. Draw activity diagram (If required)

Activity Diagram illustrate the dynamic nature of a system by modeling the flow of control form activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Fig. 26 shows the activity diagram processing an order to deliver some goods.

Software Design

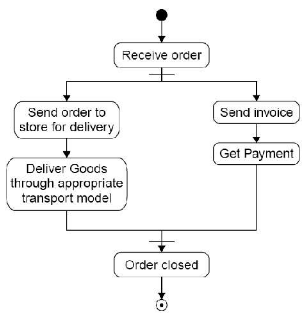

Receive order
Send order to store for delivery
Send invoice
Deliver Goods through appropriate transport model
Get Payment
Order closed

Fig. 26: Activity diagram

Software Design

iii. Draw the interaction diagram

An interaction diagram shows an interaction, consisting of a set of objects and their relationship, including the messages that may be dispatched among them. Interaction diagrams address the dynamic view of a system.

Steps to draws interaction diagrams are as under:

a) Firstly, we should identify that the objects with respects to every use case.

b) We draw the sequence diagrams for every use case.

d) We draw the collaboration diagrams for every use case.

Software Design

The object types used in this analysis model are entity objects, interface objects and control objects as given in fig. 27.

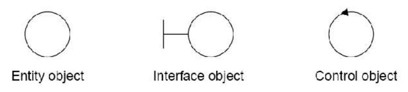


Fig. 27: Object types

Software Design

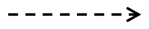
iv. Draw the class diagram

The class diagram shows the relationship amongst classes. There are four types of relationships in class diagrams.

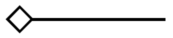
a) **Association** are semantic connection between classes. When an association connects two classes, each class can send messages to the other in a sequence or a collaboration diagram. Associations can be bi-directional or unidirectional.

Software Design

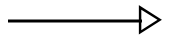
b) **Dependencies** connect two classes. Dependencies are always unidirectional and show that one class, depends on the definitions in another class.



c) **Aggregations** are stronger form of association. An aggregation is a relationship between a whole and its parts.



d) **Generalizations** are used to show an inheritance relationship between two classes.



Software Design

v. Design of state chart diagrams

A state chart diagram is used to show the state space of a given class, the event that cause a transition from one state to another, and the action that result from a state change. A state transition diagram for a "book" in the library system is given in fig. 28.

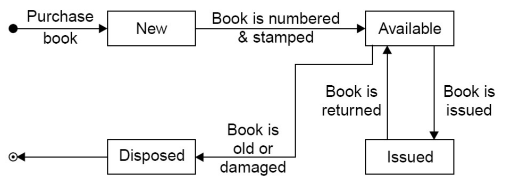


Fig. 28: Transition chart for "book" in a library system.

Software Design

vi. Draw component and development diagram

Component diagrams address the static implementation view of a system they are related to class diagrams in that a component typically maps to one or more classes, interfaces or collaboration.

Deployment Diagram Captures relationship between physical components and the hardware.

Software Design

A software has to be developed for automating the manual library of a University. The system should be stand alone in nature. It should be designed to provide functionality's as explained below:

Issue of Books:

- ❖ A student of any course should be able to get books issued.
- ❖ Books from General Section are issued to all but Book bank books are issued only for their respective courses.
- ❖ A limitation is imposed on the number of books a student can issue.
- ❖ A maximum of 4 books from Book bank and 3 books from General section is issued for 15 days only.The software takes the current system date as the date of issue and calculates date of return.

Software Design

- ❖ A bar code detector is used to save the student as well as book information.
- ❖ The due date for return of the book is stamped on the book.

Return of Books:

- ❖ Any person can return the issued books.
- ❖ The student information is displayed using the bar code detector.
- ❖ The system displays the student details on whose name the books were issued as well as the date of issue and return of the book.
- ❖ The system operator verifies the duration for the issue.
- ❖ The information is saved and the corresponding updating take place in the database.

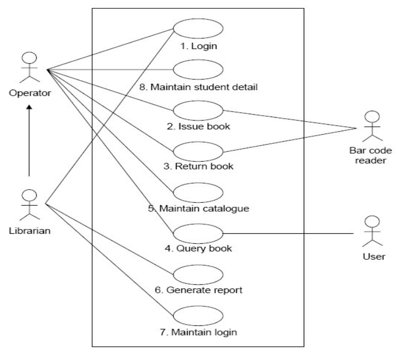
Software Design

Query Processing:

- ❖ The system should be able to provide information like:
- ❖ Availability of a particular book.
- ❖ Availability of book of any particular author.
- ❖ Number of copies available of the desired book.

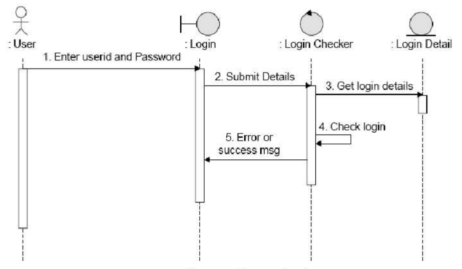
The system should also be able to generate reports regarding the details of the books available in the library at any given time. The corresponding printouts for each entry (issue/return) made in the system should be generated. Security provisions like the 'login authenticity should be provided. Each user should have a user id and a password. Record of the users of the system should be kept in the log file. Provision should be made for full backup of the system.

Software Design



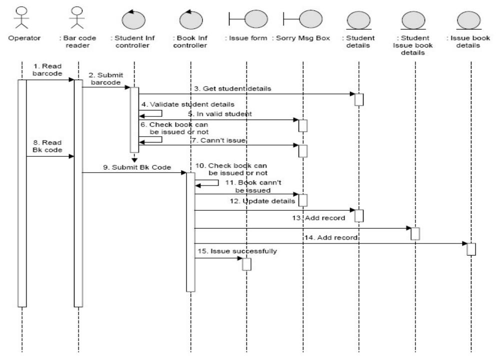
Use case diagram for library management system

Software Design



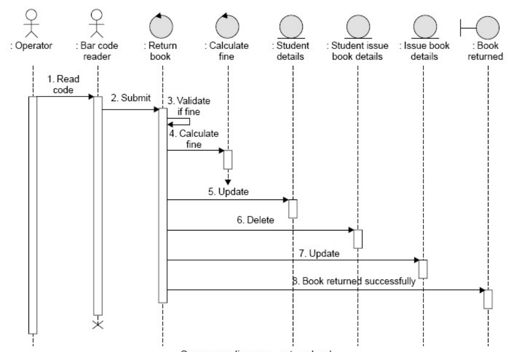
Sequence diagram—Login

Software Design



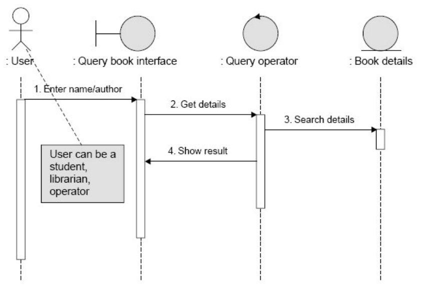
Sequence diagram—issue book

Software Design



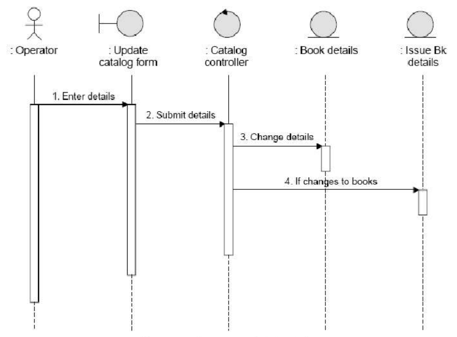
Sequence diagram—return book

Software Design



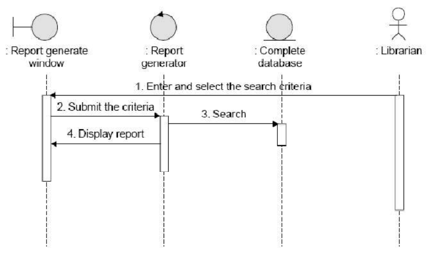
Sequence diagram—query book

Software Design



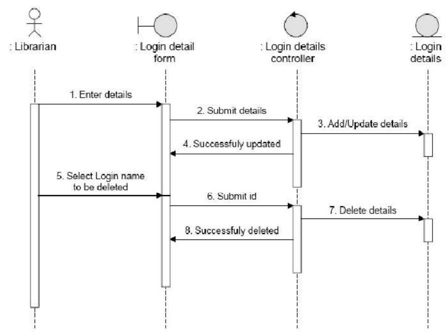
Sequence diagram—maintain catalog

Software Design



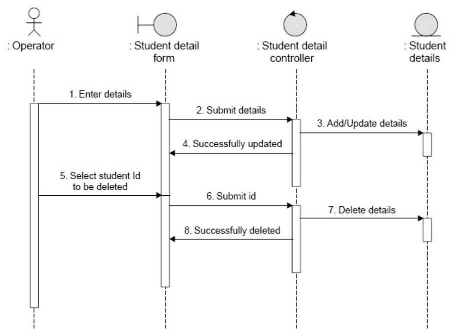
Sequence diagram—generate reports

Software Design



Sequence diagram—maintain login

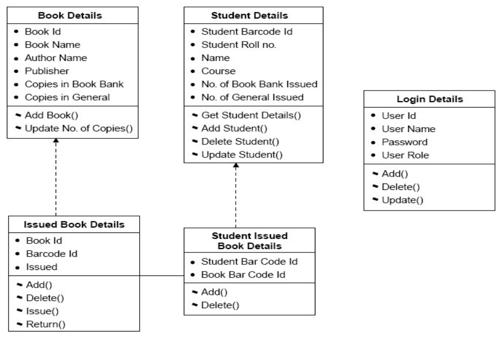
Software Design



Sequence diagram—maintain student details

Software Design

Class diagram of entity classes



Class diagram of entity classes

3