2024

Object Oriented Analysis and Design

Dr. Babasaheb Ambedkar Open University

Object Oriented Analysis and Design

Expert Committee

Prof. (Dr.) Nilesh K. Modi Professor and Director, School of Computer Science, Dr. Babasaheb Ambedkar Open University, Ahmedabad	(Chairman)
Prof. (Dr.) Ajay Parikh Professor and Head, Department of Computer Science Gujarat Vidyapith, Ahmedabad	(Member)
Prof. (Dr.) Satyen Parikh Dean, School of Computer Science and Application Ganpat University, Kherva, Mahesana	(Member)
M. T. Savaliya Associate Professor and Head Computer Engineering Department Vishwakarma Engineering College, Ahmedabad	(Member)
Mr. Nilesh Bokhani Assistant Professor, School of Computer Science, Dr. Babasaheb Ambedkar Open University, Ahmedabad	(Member)
Dr. Himanshu Patel Assistant Professor, School of Computer Science, Dr. Babasaheb Ambedkar Open University, Ahmedabad	(Member Secretary)

Course Writer

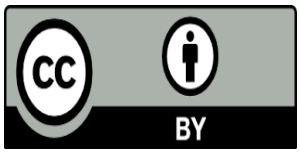
Mr. Jigneshkumar Kansara	Gujarat Vidyapith, Ahmedabad
Dr. Himanshu Patel	Assistant Professor, Department of Computer Science, KSKV Kachchh University

Content Reviewer and Editor

Prof. (Dr.) Sanjay M. Shah	Government Engineering College, Modasa

ISBN: 978-81-942146-6-3

Acknowledgement: Block-3 and Block-4 in this book is modifications based on the work created and Shared by the Saylor Academy of Unit-3for the course CS302 Software Engineering (https://learn.saylor.org/course/view.php?id=73&sectionid=18225) and used according to terms described in a Creative Commons Attribution 3.0 Unported (CC BY 3.0) License: https://creativecommons.org/licenses/by/3.0/



Printed and published by: Dr. Babasaheb Ambedkar Open University, Ahmedabad While all efforts have been made by editors to check accuracy of the content, the representation of facts, principles, descriptions and methods are that of the respective module writers. Views expressed in the publication are that of the authors, and do not necessarily reflect the views of Dr. Babasaheb Ambedkar Open University. All products and services mentioned are owned by their respective copyrights holders, and mere presentation in the publication does not mean endorsement by Dr. Babasaheb Ambedkar Open University. Every effort has been made to acknowledge and attribute all sources of information used in preparation of this learning material. Readers are requested to kindly notify missing attribution, if any.

Dr. Babasaheb Ambedkar Open University

BCAR-502

Object Oriented Analysis and Design

Block-1: Introduction to Object Oriented Analysis

Block-2: Introduction to Web Engineering

Block-3: Introduction to UML and UML Diagrams

Block-4: UML Interaction Diagram

Block-1
Introduction to Object Oriented Analysis

Unit 1: Domain Analysis

1

Unit Structure

1.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Understand Object Oriented Analysis.
- Understand system in depth and build more flexible system.
- Communicate with the stakeholders more effectively.
- Establish requirements more rapidly.
- Ensure that the solutions adoptedcan solve problem effectively.
- Demonstrate an in-depth understanding of Object Oriented paradigm and concepts.

1.2 INTRODUCTION TO OBJECT ORIENTED ANALYSIS

To understand Object Oriented Analysis, let's first understand main features and aspects of Object Oriented Programing (OOP).Mastering OOP is essential for developinga high quality software.

1.2.1 Elements of OOP

In OOP, program will be split into several small, manageable and reusable sub programs. Each sub program has its own identity, data, logic and defined communication with other sub programs.

Object

Objects could be:

- Something visible like car, spoon, banana, sparrow, bulb or ball etc.
- Something that one cannot touch or abstract like bank account, date, error or email.

Figure 1 - Objects

Each Object has its own *attributes*and *behaviour*.

*Attributes*are the characteristics or the properties of the object. For example, in case of car, the attributes are`colour` and `manufacturer`. The attributes of one object is independent of another, so one can have a white car and other may have a red car.

Behaviour is something we can do with the object(actions); in case of car, car can run. Another example, in case of bank-account, bank-account can be credited or debited.

Class

A class is the type of data structurewhere you can define properties and behaviour.

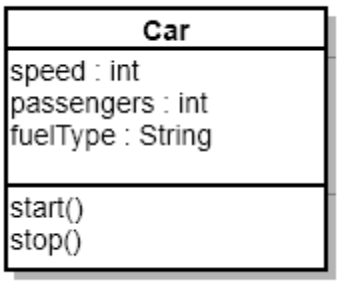


Figure 2 - Car Class

A class is a ***blueprint***, it is providing the definition (properties and behaviours) for an Object. An object is an instance of a class that typically represents a real world object and has the same types of characteristics (properties), behaviours (methods), and states (data).

1.2.2 Features of OOP

Encapsulation

Encapsulation is a process of information (data) hiding. It is simply the combination of methodsand data into a single entity. Data of an object is hidden from the rest of the system and available only through the methods(services) of the class.

Benefits of Encapsulation:

- **Information hiding:** The internal implementation details of data and methods are hidden from outside world.

- **Reuse:** Data and the Operations that manipulate data, are merged in single entity (class). This helps components reuse.
- **Interface among encapsulated objects are simplified:** An object which is using other object's operations, need not be worried about implementation or internal data structure.

Abstraction

Abstraction is the idea of focusing on the common properties and behaviours of the object within some context, and ignore what's unimportant or irrelevant within the context.

For example, we might want to create an abstraction for a food object. In a health context, its "`nutritional value`" would be part of anobject and "`cost`" can be ignored.

Let's take the **person** class. What are the essential characteristics of a person that we care about? Well, it's hard to say because person is so vague and we have notdefinedthe purpose of our person class. The abstractions you create are relative to some context. For example, if you are creating a driving app, you would care about a person in the context of a driver.You can consider `driving_licence_number` as an attribute. In another example, person in the context of student. What are some of the essential characteristics of a student? Here, you can include the`role_number, courses_name` and `grades` in each course. These are basic attributes of a person in the context of student.

The attributes do not disappear over time although their values may change since they are essential characteristics of a student. For a course, the student's `grade` value may change but student always have a `grade` attribute. This means the actual values of these attributes may change, but the attributes themselves do not.

Inheritance

Using *Inheritance,*one can create new classfrom existing class by extending and refining its capabilities. The existing classis called base class/parent class/ super-

class and the newly created class is called derived class/child class/subclass. The subclass can inherit or derive the attributes and methods of the super class, provided that the super class allows so. Besides, the subclass may add its own attributes and methods and may modify super-class methods.

Note: Inheritance defines an “is – a” relationship.

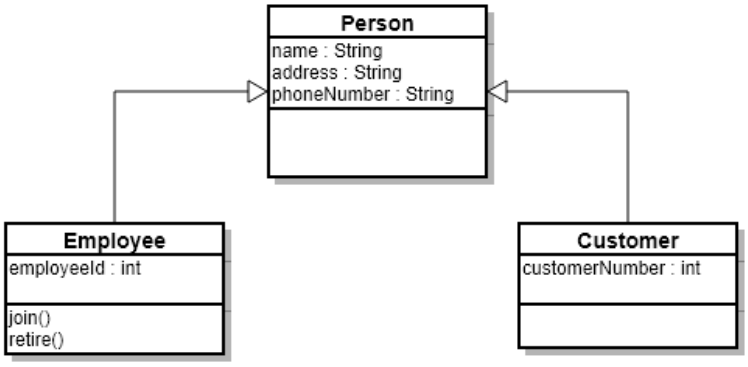


Figure 3 - Inheritance

For example, there is a**Person** class with attributes`name`, `address` and `phoneNumber`. Now, if one creates a newclass**Employee** by inheriting **Person** class. **Employee** classgetsallthe attributes and methods of**Person** classwithout having to write any codeand also can have its own additional attribute`employeeId` and methodslike `join()` and `retire()`.

Note: If one makes a change in the Person class, it will automatically cascade down and affect the inheriting classes.

Polymorphism

Polymorphism is originally a Greek word that means *the ability to take multiple forms*.

Objects of classes belonging to the same hierarchical tree (inherited from a common base class) may have functions with the same name, but each having different behaviours.

For an example, there is a base class named **Animals**and inheriting classes**Dog**, **Cat** and **Duck**. Each of the inheriting classes override the `speak()` method. With polymorphism, each subclass may have its own way of implementation of

`speak()` method. So, when the `speak()` function is called in an object of the **Dog** class, the function might respond by displaying "Bark" on the screen. On the other hand, when the same function is called onan object of the **Cat** class, "Meow" might get displayed on the screen. In case of a **Duck**, it may be "Quack".

1.2.3 Structured Approach vs. Object-Oriented Approach

The following table explains how the object-oriented approach differs from the traditional structured approach

Structured Approach	Object Oriented Approach
Program is divided into number of submodules or functions.	Program is organized by having number of classes and objects.
Function call is used.	Message passing is used.
Software reuse is not possible.	Reusability is possible.
It is suitable for real time system, embedded system and projects where objects are not the most useful level of abstraction.	It is suitable for most business applications, game development projects, which are expected to customize or extended.

1.2.4 Object Oriented Analysis

Object Oriented System Development Life Cycle consists of three processes;

1) Object Oriented Analysis (OOA)
2) Object Oriented Design (OOD)
3) Object Oriented Implementation (OOI)

In this unit, we are covering Object Oriented Analysis process.

Object Oriented Analysis is a technical approach for analysinganapplication by applying the Object Oriented Programming concepts as well as using visual modelling during the development life cycle of the software.

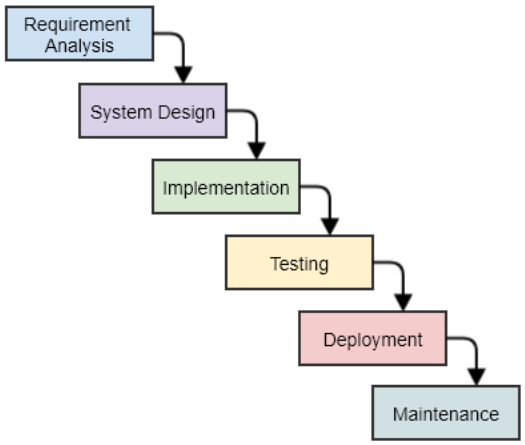


Figure 4 - The development life cycle of the software

The software development life cycle is typically starts from *requirement analysis* to *system design* to *Implementation* and *testing* and finally to *deployment*.The earliest stages of Object Oriented Analysis process are *requirement analysis* and *system design*.The difference between analysis and design is often described as “What vs How”.

In Object Oriented Analysis, developers work with **customer** and **domain experts** to define what the system is supposed to do. Implementation details are supposed to be ignored at this stage.

The purpose of OOA is to define all classes (that are required to develop system), attributes and operation associated with classes, and relationships between classes. To perform Object Oriented Analysis, a software engineer should perform the following generic steps:

1. Define customer requirements for the system:Define what does the software needs to do, and what is the problem the software is trying to solve.
2. Identify scenarios or use-cases:Describe the requirements, usually carried out by using use cases (and scenarios) or user stories.
3. Select classes and objects using basic requirements as a guide.
4. Identify attributes and operations for each system class.
5. Define structures and hierarchies that organize classes.
6. Build an Object Relationship model.
7. Build an Object Behaviour model.
8. Review these models against use-cases or scenarios.

Summary:

Analysis = Process + Models

Process	**Model Output**
1) Elicit customer requirements and identify use-cases.	Use-Case diagrams
2) Extract candidate classes, Identify attributes and methods, Define a class hierarchy.	Class Responsibility Collaborator (CRC) cards
3) Build an Object Relationship model (structural).	Conceptual Class diagram
4) Build and Object Behaviour model (dynamic).	Interaction diagram

We will cover first two processes in this unit.

1.3 DOMAIN ANALYSIS

In software engineering, **Domain Analysis**, is the process of analysing related software systems in a domain to find their common and variable parts.

Domain analysis is the first phase of **Domain Engineering**. It is a key method for realizing systematic software reuse (common part).

The **Domain** is the general field of business or technology in which the clients will use the software.A **Domain Expert** is a person who has a deep knowledge of the domain.

Benefits of performing Domain Analysis:

- Faster development
- Better system
- Anticipation of extensions

Domain analysis is generallyperformed when an organization wants to create a domain specific library of reusable classes (components).

1.3.1 Reuse and Domain Analysis

Consider a simple example to understand the benefits of reuse.

Two teams (with same skill levels and experience) have been assigned to build a new application which required to create 200 classes.

Team-One doesn't have any class library hence they have to develop all 200 classes from scratch. **Team-Two** is using robust class library and they found that 100 classes can be reused from library.

Can you guess which team can do better and rapid development?

Probable result;

1 Team-Two can finish the project much faster than Team-One.
2 Cost: Cost of Team-Two will be significantly lower than the cost of Team-One.
3 Defect: Team-Two's product will have fewer defects than Team-One's product due to use of pre tested reusable code library.

"Robust class library" can be created using domain analysis.

1.3.2 The Domain Analysis Process

Firesmith[Ref-1]describes software domain analysis in the following way;

"Software domain analysis is the identification, analysis, and specification of common requirements from a specific application domain, typically for reuse on multiple projects within that application domain."

Domain analysis is an ongoing software engineering activity that is not associated to any one software project. The role of the domain analyst is to design and build reusable components that may be used by many people working on similar but not necessarily the same applications.

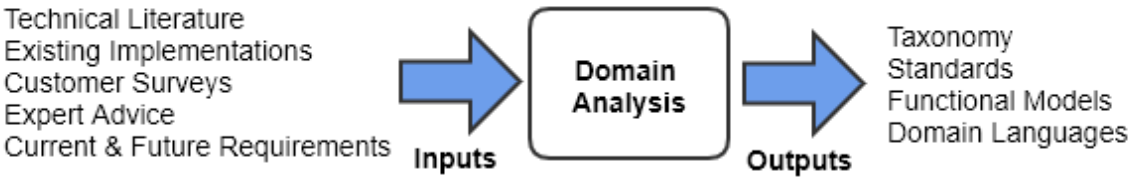


Figure 5 -Domain Analysis

Figure 5 shows key inputs and outputs for the domain analysis process. Domain analysis is quite similar to knowledge engineering.

The Knowledge engineer investigates a specific area of interest to extract *key facts* that may be of use in creating an *expert system*.

During domain analysis, source of domain knowledge isinvestigated to extract reusable objects (and class).

The domain analysis process can be characterized by a series of activities that begin with the identification of the domain to be investigated and ends with a specification of the objects and classes that characterize the domain. Berard [Ref-2] suggestedthe following Activities:

Define the domain to be investigated:

The analyst must first define the domain to be investigated. Next, both Object Oriented (OO) and non-OO components must be extracted.

- OO items include specifications, designs, existing application classes; support classes (e.g., database access classes) and test cases.
- Non-OO items include policies, procedures, plans, standards, and guidelines.

Categorize the items extracted from the domain:

The extracted items are organized into categories and naming conventions for each item are defined.

Collect a representative sample of applications in the domain:

The analyst must ensure that the application has items that fit into the categories that have already been defined.

Analyse each application in the sample: The following steps are followed by the analyst:

- Identify candidate reusable objects.
- Indicate the reasons that the object has been identified for reuse.
- Define adaptations to the object that may also be reusable.
- Estimate the percentage of applications in the domain that might make reuse of the object.
- Identify the objects by name and use configuration management techniques to control them. In addition, once the objects have been defined, the analyst should estimate what percentage of a typical application could be constructed using the reusable objects.

Develop an analysis model for the objects: The analysis model will serve as the basis for design and construction of the domain objects.

In addition to these steps, the domain analyst should also create a set of reuse guidelines and develop an example that illustrates how the domain objects could be used to create a new application.

1.4GENERIC COMPONENTS OF THE OO ANALYSIS MODEL

To develop a “precise, concise, understandable, and correct model of the real world,” a software engineer must select a notation that implements a set of generic components of an OO analysis model.

There are two types of design models:

Static Components: These are structural in nature and indicate characteristics that hold throughout the operational life of an application. These characteristics differentiate one object from other objects.

Dynamic Components: Focus on control and are sensitive to timing and event processing. They define how one object interacts with other objects over time.

The following components are identified by Monarchi [Ref-3]:

Static view of Semantic classes: As part of the analysis model, requirements are evaluated and classes are extracted. These classes persist throughout the life of the application and are used based on the customer requirements.

Static view of Attributes: Each class will be explicitly described. The attributes and operations associated with the class provide a description of the class.

Static view of Relationships: Objects are connected to each other. The analysis model must represent these relationships sothat operations (that affect these connections) can be identified and thedesign of a messaging approach can be accomplished.

Static view of Behaviours: After defining relationships, the analysis model defines behaviours that accommodate the use-cases of the system. Behaviours can be achieved by sequence of operations.

Dynamic view of Communication: Objects can communicate with each other based on series of events. These events can change system's state.

Dynamic view of Control and Time: The nature and timing of events that change states must be described.

1.5THE OBJECT ORIENTED ANALYSIS PROCESS

The Object Oriented Analysis process begins with an understanding of the manner in which the system will be used;

1) By people, if the system is human interactive
2) By machines, if the system is involved in process control
3) By other programs, if the system coordinates and controls applications.

The use cases and scenarios are two different techniques, but, usually they are used together.

Use cases identifies interactions between the system and its users(using graphical notations), while a Scenario is a textual description of one or more of these interactions.

The modelling of the software begins, oncethe scenario of usage has been defined.

1.5.1 Use Cases

Usecases model the system from the end-user's point ofview.Use-cases should achieve following objectives:

- To define the functional and operational requirements of the system by defining a scenario of usage that is agreed by the end-user and the software engineering team.
- To provide a clear and unambiguous description of how the end-user and the system interact with one another.
- To provide a basis for validation testing.

Use case symbols:

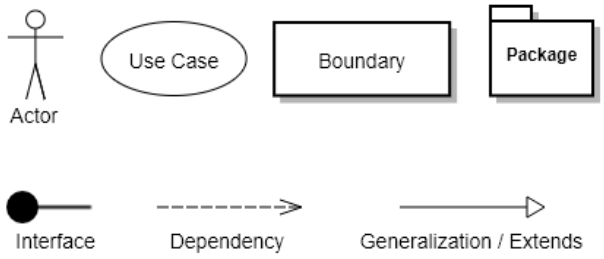


Figure 6 -Symbols of use case

1. Actors: Are those who interact with the system; human or other systems.
2. Use Case: A use case describes how actors uses a system to accomplish a particular goal. Use cases are typically initiated by a user to fulfill goals describing the activities.
3. Connection: Lines that links the actors and the interactions.

Here is the example of a use case scenario:

Name	Course Registration
Actors	Student and University System
Description	It shows how a student can register for a course and view personal information.
Pre-condition	The student is logged in
Post-condition	The student registered his/her course list for the semester.
Actions	1. Student will press on “Course Registration” from home page. 2. Select desired courses. 3. Enter personal info 4. Press on “Register” and “Course Fee Payment” 5. Confirmation message upon success. 6. Student can view his / her details. 7. Student can send comment in case of question / query.
Exceptions	User entered invalid input, hence, an error message will be displayed.

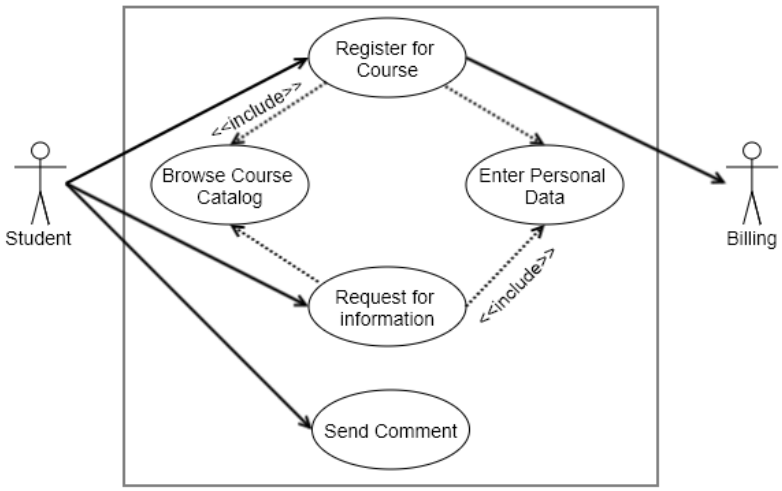


Figure 7 - Use case scenario

1.5.2 Class-Responsibility-Collaborator (CRC) Modelling

Once we have some use cases / user stories, the next thing we can do is identify candidate classes and indicate their responsibilities and collaborations.Class-Responsibility-Collaborator (CRC) modelling provides a simple means foridentifying and organizing the classes that are relevant to system or product requirements.

A CRC model is a collection of standard index cards that represent classes. A CRC card has three sections. The top of the card has the class name. On the left are the responsibilities of the class, and on the right, you list collaborators.

Responsibilities are the attributesand operations that are relevant for the class. Stated simply, a responsibility is “anythingthe class knows or does”.

Collaborators are other classes that the class interacts with to fulfil its responsibilities.

Classes / Objects

Objects will be in form of nouns. Those are the candidate objects, however, not every potential object can be considered as final object. There are six selection characteristics.

1. **Retained Information**: The potential object will be useful during analysisonly if information about it must be remembered so that the system canfunction.
2. **Needed Services:** The potential object must have a set of identifiable operationsthat can change the value of its attributes in some way.
3. **Multiple Attributes:** During requirements analysis, the focus should be on"major" information; an object with a single attribute may be usefulduring design but is probably better represented as an attribute of anotherobject during the analysis activity.
4. **Common Attributes**: A set of attributes can be defined for the potentialobject and these attributes apply to all occurrences of the object.
5. **Common Operations:** A set of operations can be defined for the potentialobject and these operations apply to all occurrences of the object.
6. **Essential Requirements:** External entities that appear in the problem spaceand produce or consume information that is essential to the operation of

anysolution for the system will almost always be defined as objects in therequirements model.

A potential object should satisfy all six of these selection characteristics if it is to beconsidered for inclusion in the CRC model.

Let's take an example:
Select following use case scenario and underline all potential objects.

Use Case Scenario: Customer verifies items in shopping cart. Customer provides payment and address to process sale. System validates payment and responds by confirming order and provides order number that Customer can use to check on order status. System will send Customer a copy of order details by email.

Figure 8 - Sample use case scenario

After underlying on candidate objects, start refining them.

Customer	**Order**
Item	~~**Order Number**~~
Shopping Cart	~~**Order Status**~~
Payment	~~**Order Details**~~
Address	**Email**
~~**Sale**~~	~~**System**~~

Figure 9 - Candidate Objects

- Remove any duplicates. We may find same objects with different names, but they actually mean the same thing.
- You may identify an attribute as an object instead. An attribute is a property or characteristic of the object. For example, `order_number` is an attribute of `Order` class.
- Youmay identify a behaviour as an object instead. A behaviour is something an object can do (responsibility). For example, when we say, "Check the Order stats", check here is a behaviour of the Order object.

After refining objects, start drawing objects.

Figure 10 - Refined Objects

Responsibilities

Guidelines for allocating responsibilities to classes:

Wirfs-Brock and her colleagues [Ref-4] suggested five guidelines for allocatingresponsibilities to classes:

1) System intelligence should be evenly distributed

Every application's intelligence defines "what the system knows and what it can do". This intelligence can be distributed across classes in a number of different ways. For example, “Dumb” classes (those that have few responsibilities) can be modelled to act as servants to a few “smart” classes (those having many responsibilities). It has a few disadvantages: (1) it concentrates all intelligence within a few classes, making changes more difficult. (2) It tends to require more classes, hence more development effort.

Hence system intelligence should be evenly distributed across the classes in an application. To determine whether system intelligence is evenly distributed, the responsibilities noted on each CRC model index card should be evaluated to determine if any class has an extraordinarily long list of responsibilities. This indicates a concentration of intelligence.

2) Each responsibility should be stated as generally as possible

General or common responsibilities (both attributes and operations) should reside high in the class hierarchy (because they are generic, they will apply to all subclasses).

3) Information and the Behaviour related to it should reside within the same class.

This can be achieved using encapsulation(as discussed in section 1.2.2). Data and the process that manipulate the data should be within the same class.

4) Information about one thing should be localized with a single class, not distributed across multiple classes.

A single class should take on the responsibility for storing and manipulating a specific type of information. This responsibility should not be shared across a number of classes. If information is distributed, software becomes more difficult to maintain and more challenging to test.

5) Responsibilities should be shared among related classes, when appropriate.

There are many cases in which a variety of related objects must all show the same behaviour at the same time. As an example, consider a video game that must display the following objects: player, player-body, player-arms, player-legs and player-head. Each of these objects has its own attributes (e.g., position, orientation, colour, speed) and all must be updated and displayed as the user manipulates a joy stick. The responsibilities update and display must therefore be shared by each of the objects. Player knows when something has changed and update is required. It collaborates with the other objects to achieve a new position or orientation, but each object controls its own display.

In our example, * indicate the relationships between objects. Object relationship means interaction with each other. For example, a customer can place an order, a student can enrol in a course, and admin can update a record, and so on.

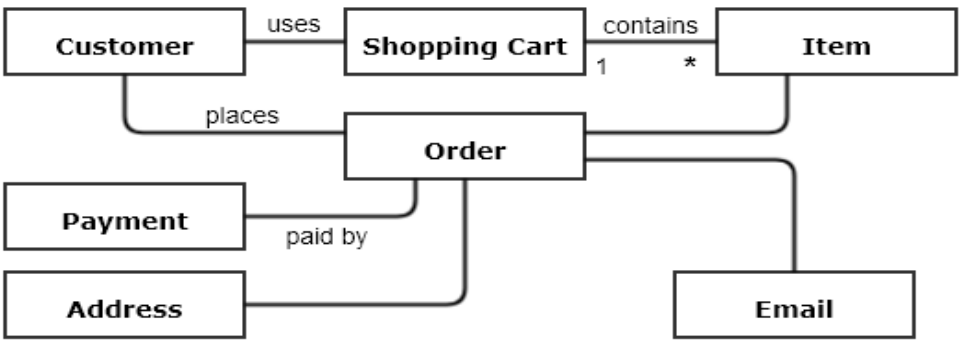


Figure 11 - Domain Analysis

Collaborations

Classes fulfil their responsibilities in one of two ways:

1) A class can use its own operations to manipulate its own attributes, thereby fulfilling a particular responsibility.
2) A class can collaborate with other classes.

Collaborations identify relationships between classes. When a set of classes all collaborate to achieve some requirement, they can be organized into a subsystem. Collaborations are identified by determining whether a class can fulfil each responsibilityitself. If it cannot, then it needs to interact with another class. Hence, a collaboration.

After relationship, identify object behaviours. Behaviours are the things/actions the object can do. In other words, the responsibilities of an object, which will become the methods in our object class.

So, we can go back to the use case and look for verbs and verb phrases to pick responsibilities.

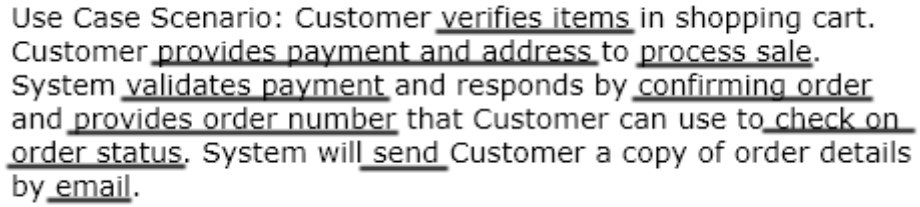


Figure 12 - Identification of Responsibilities

Things like *verifies* items, *provides* payment and address, *process* sale, *validate* payment, *confirm* order, *provide* order number, *check* order status and *send* order details email. Not all of these will become behaviours, some will be combined, some will need to be split apart, some will just not be needed or be replaced by something else, but they are a good starting point.

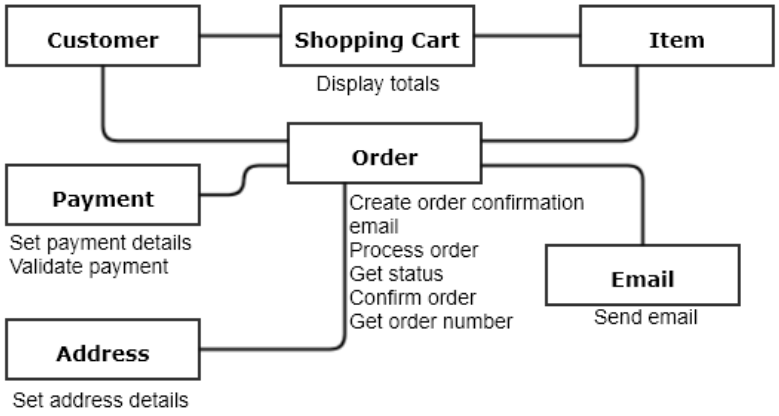


Figure 13 - Object behaviours

Class-Responsibility-Collaborator Modelling

Once basic usage scenarios have been developed for the system, it is time to identify candidate classes and indicate their responsibilities and collaborations. Class-Responsibility-Collaborator (CRC) modelling provides a simple means for identifying and organizing the classes that are relevant to system or product requirements.

A CRC is a collection of standard index cards that represent classes. The cards are divided into three sections. On the top of the card you write the name of the class. In the body of the card you list the class responsibilities on the left and the right side has the other objects (collaborators) that has a relationship with the object.

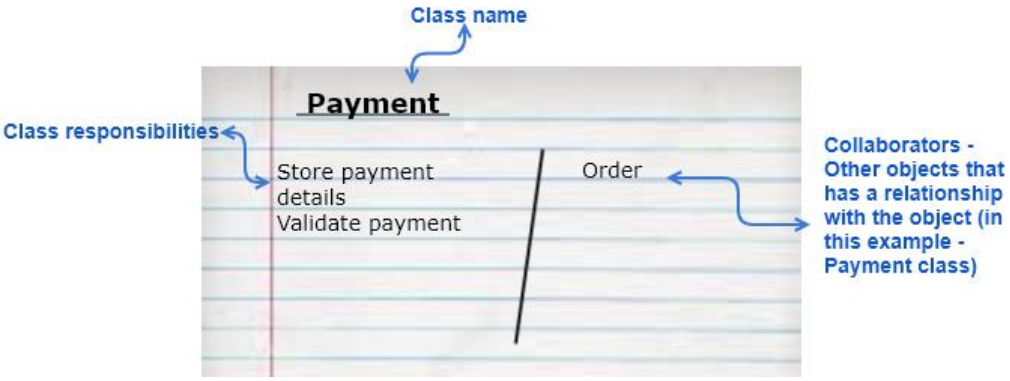


Figure 14 - Class-Responsibility-Collaborator Modelling

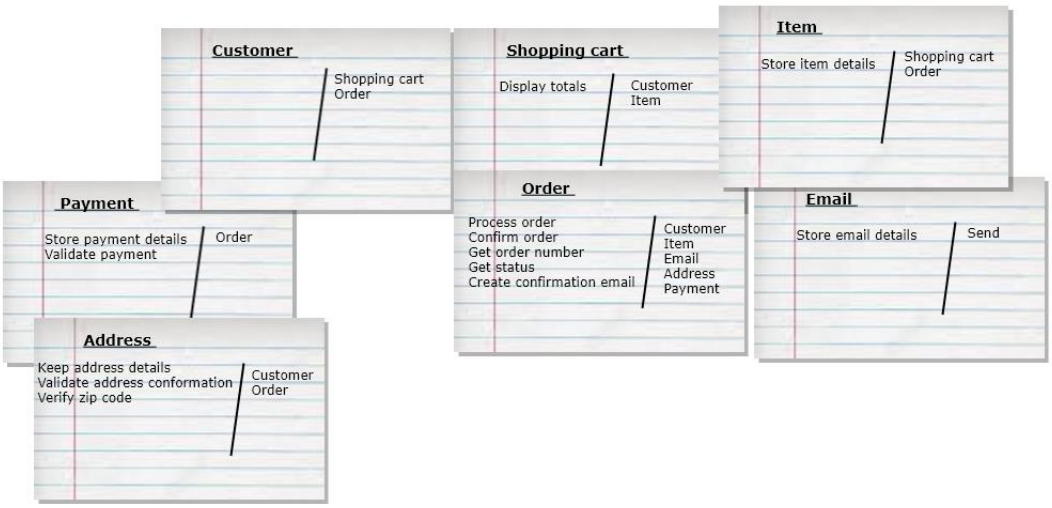


Figure 15 - Class-Responsibility-Collaborator Modelling

They are small, easy to use, you can move them around to show the relationship, you can modify them easily if there is a mistake.

This modus operandi continues until the use-case is finished. Until all use-cases (or use-case diagrams) have been reviewed, Object Oriented Analysis continues.

1.6LET US SUM UP

Object Oriented Analysis methods enable a software engineer to model a problem by representing both static and dynamic characteristics of classes and their relationships as the primary modelling components.

The Object Oriented Analysis process begins with the definition of use-cases / scenarios that describe how the Object Oriented system is to be used. The Class-Responsibility-Collaborator modelling technique is then applied to document classes and their attributes and operations. It also provides an initial view of the collaborations that occur among objects. The next step in the Object Oriented Analysisprocess is classification of objects and the creation of a class hierarchy.

1.7CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Question 1

A Car object has following properties – Manufacturer name, Model, Drive it, Lock it. Identifywhich among the above properties represent behaviour of the object.

a) Drive it, Model

b) Model, Manufacturer name

c) Manufacturer name, Lock it

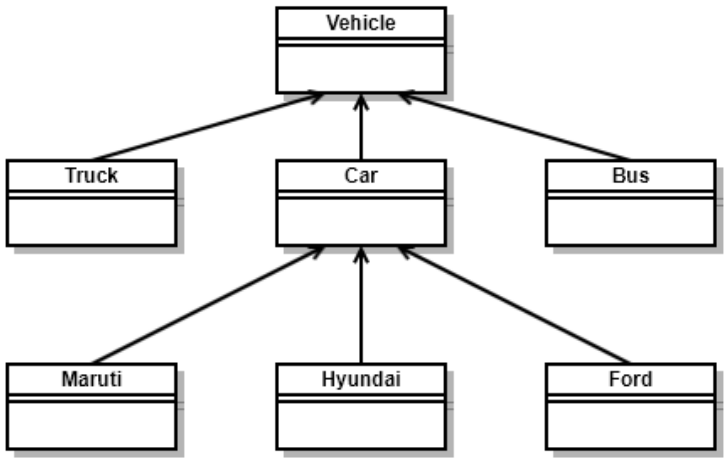
d) Lock it, Drive it

Answer: d)

Explanation: Manufacturer name and Model are the state of the object Car as they are theattributes of the Object. Drive it, Lock it describes the behaviour of the object Car. Behaviouris the set of things that an object can do on its own or are acted on.

Question 2

Identify the correct statements for the given image.



a) Maruti is the superclass of vehicle

b) It is an example of Hybrid inheritance

c) Car is subclass of Vehicle and superclass of Maruti.

d) It is an example of Hierarchical Inheritance

Answer: c), d)

Explanation: Multiple derived classes (Truck, Car and Bus) inherit from the same class Vehicle.Similarly multiple derived classes then in turn inherit from Car.

1.8FURTHER READING

Books

- Software Engineering by Roger S. Pressman Sixth Edition McGraw Hill Publications.

Web sites

- https://www.tutorialspoint.com/object_oriented_analysis_design/index.htm
- https://en.wikipedia.org/wiki/Object-oriented_analysis_and_design
- https://medium.com/omarelgabrys-blog/object-oriented-analysis-and-design-introduction-part-1-a93b0ca69d36

1.9ASSIGNMENTS

1. Write a simple, one-line definition of each of the following key terms. Try to do this without referring back to the notes if possible. For each term, give an example.
    - Polymorphism
    - Encapsulation
    - Abstraction
    - Inheritance
    - Polymorphism
    - Domain expert

1.10ACTIVITIES

Do CRC modelling for ATM machine. You insert your bank debit card into the ATM machine, the ATM machine will then ask you to enter a PIN authenticating you for access. After that, you can choose to deposit, withdraw or check your balances.

1.11CASE STUDIES

Do a domain analysis for one of the following areas:

a) Admission system of a university
b) An e-commerce application (e.g., clothes, books, electronic gear).
c) Customer services for a bank.

Be sure to isolate classes that can be used for a number of applications in the domain.

1.12 REFERENCES

- Ref-1: D.G., Object-Oriented Requirements Analysis and Logical Design, Wiley, 1993
- Ref-2:Berard-E.V., Essays on Object-Oriented Software Engineering, Addison-Wesley, 1993
- Re-3: Monarchi (D.E. and G.I. Puhr, “A Research Typology for Object-Oriented Analysis and Design,” CACM, vol. 35, no. 9, September 1992, pp. 35–47)
- Ref-4: [WIR90] Wirfs-Brock, R., B. Wilkerson, and L. Weiner, Designing Object-Oriented Software, Prentice-Hall, 1990

Unit 2: Models 2

Unit Structure

2.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Identify and model (represent) domain constraints on the objects and their relationships.
- Learn various modelling techniques to model different perspectives of object-oriented software design.

2.2 INTRODUCTION

Software development life cycle start with requirement analysis. Requirements are mostly defined in the form of tabular lists, use-cases, user stories or decision tables. The written words are good way to communicate but it is not the best way to represent the requirements for computer software. Modelling uses combination of text and diagrams to represent requirements for data, function and behaviour in a way that is easy to understand and easy to review for correctness, completeness and consistency.

Do you remember below table? (Already discussed in unit 1)

Analysis = Process + Models

Process	**Model Output**
1) Elicit customer requirements and identify use-cases.	Use-Case diagrams
2) Extract candidate classes, Identify attributes and methods, Define a class hierarchy.	Class Responsibility Collaborator (CRC) cards
3) Build an Object Relationship model (structural).	Conceptual Class diagram
4) Build and Object Behaviour model (dynamic).	Interaction diagram

In previous unit, we have already discussed first two processes. We will cover Object Relationship model and Object Behaviour model in this unit.

First, let's understand basic elements of modelling.

Data Objects, Attributes, and Relationships

The data model consists of three interconnected pieces of information.

1) The data object
2) The attributes (which describe the data object)
3) The relationships (that connect data objects to one another)

For example, a **person** or a **car** is an object. A person has attributes like; Name, Address, Age, etc. A car has attributes like; Make, Model, Colour, etc.

Data objects are connected to one another in different ways. Considertwo data objects, **book** and **bookstore**. These objects can be represented usingthe simple notation illustrated in Figure 1. A connection is established between**book** and **bookstore** because the two objects are related.

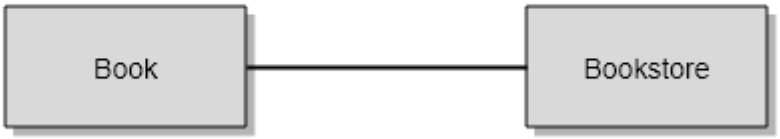


Figure 1 - A basic connection between objects

But what are the relationships? First, we must understand the role of books and bookstoreswithin the context of the software to be built. We can define a set ofobject/relationship pairs that define the relevant relationships.

For example,

- A bookstore orders books.
- A bookstore displays books.
- A bookstore stocks books.
- A bookstore sells books.
- A bookstore returns books.

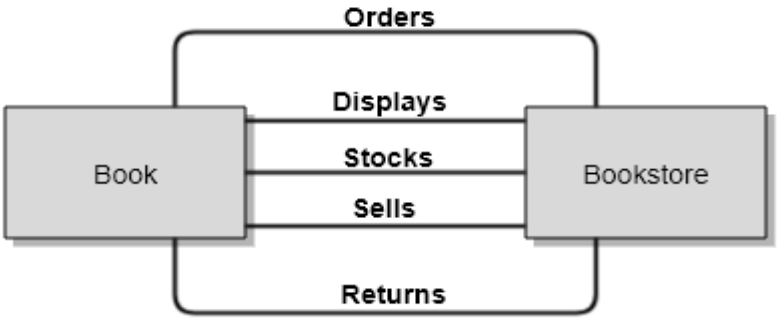


Figure 2 - Relationship between objects

Note: Object/relationship pairs are bidirectional. That is, theycan be read in either direction. A bookstore orders books or books are ordered by abookstore.

Cardinality and Modality

We have defined a set of objects and represented the object/relationship pairs thatbind them. But a simple pair that states: object-X relates to object-Y does not provideenough information for software engineering purposes. We must understandhow many occurrences of object X are related to how many occurrences of objectY. This leads to a data modelling concept called ***cardinality***.

Cardinality is: “The data model must be capable of representing the number of occurrences of objects in a given relationship”.

Cardinality is the specification of the number of occurrences of one object that can berelated to the number of occurrences of another object. Cardinality is usually expressedas simply **'one'** or **'many**.'

For example, a husband can have only **one** wife (in most cultures),while a parent can have **many** children. Taking into consideration all combinations of 'one' and 'many,' two objects can be related as;

- **One-to-one (1:1)** : An occurrence of object-A can relate to one and only one occurrence of object-B and an occurrence of object-B can relate to only one occurrence of object-A.
- **One-to-many (1:N)** : One occurrence of object-A can relate to one or many occurrences of object-B but an occurrence of object-B can relate to only one occurrence of object-A. For example, a mother can have many children, but a child can have only one mother.

- **Many-to-one (1:N)** : One occurrence of object-A can relate to one occurrences of object-B but an occurrence of object-B can relate to oneor many occurrence of object-A. For example, multiple addresses belongs to a customer.
- **Many-to-many (M:M):**An occurrence of object-A can relate to one or more occurrences of object-B, while an occurrence of object-B can relate to one or more occurrences of object-A. For example, an uncle can have many nephews, while a nephew can have many uncles.

Cardinality defines *"the maximum number of objects that can participate in a relationship"*. It does not, however, provide an indication of whether or not a particulardata object must participate in the relationship. To specify this information,the data model adds modality to the object/relationship pair.

Cardinality notation:

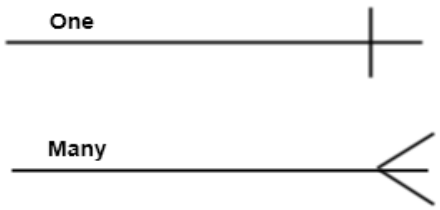


Figure 3 - Cardinality

The **modality** of a relationship is 0 if there is no explicit need for the relationshipto occur or the relationship is optional. The modality is 1 if an occurrence ofthe relationship is mandatory.

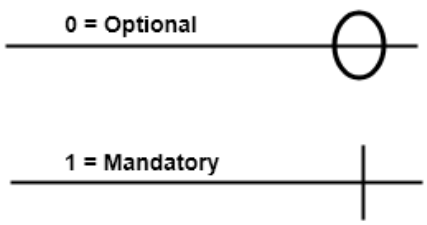


Figure 4 - Modality

Note: We can put both cardinality and modality on the same line.

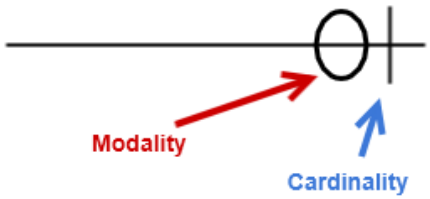


Figure 5 – Cardinality and Modality

Cardinality and modality together gives us four possible combinations:

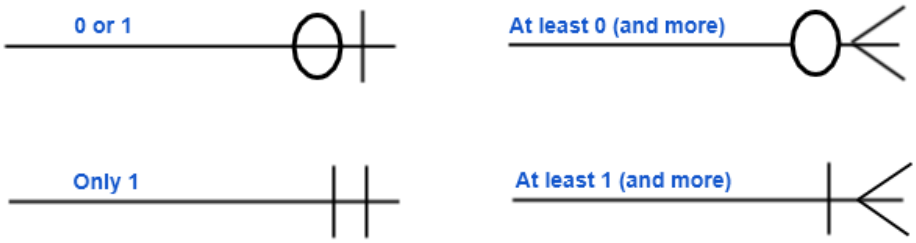


Figure 6 – Combinations of Cardinality and Modality

Figure 7 diagram shows, one manufacturer builds one or many cars.

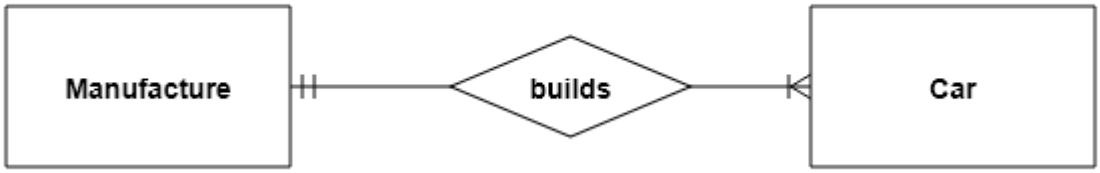


Figure 7 - Example

2.3 THE OBJECT-RELATIONSHIP MODEL

In the previous module, wehave discussed about Object Oriented Analysis and how to complete a conceptual design using CRC cards.

Below is the sample of CRC card;

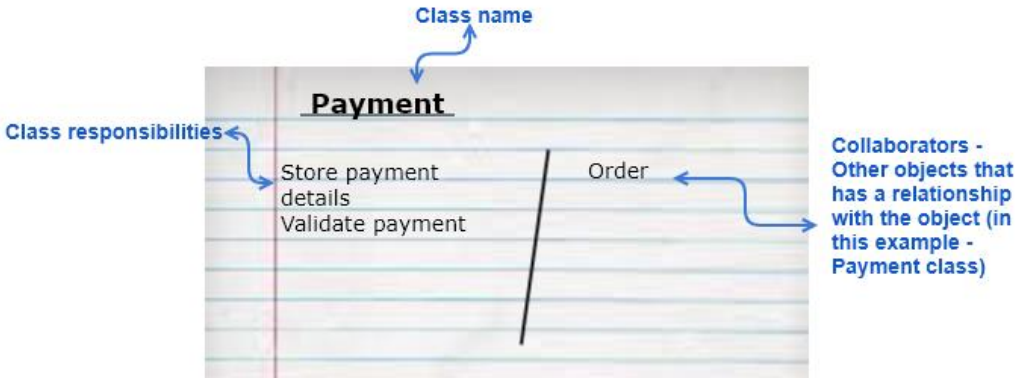


Figure 8 – CRC card

A CRC is a collection of standard index cards that represent classes. The cards are divided into three sections. On the top of the card you write the name of the class. In the body of the card you list the class responsibilities on the left and the right side has the other objects (collaborators) that has a relationship with the object.

How to define relationships of classes?

1) Need to understand the responsibilities for each class. The CRC model index card contains a list of all responsibilities.
2) Need to define those collaborator classes that help in achieving each responsibility. This establishes the “connection” between classes.

A **relationship** exists between any two classes that are connected. Therefore, collaboratorsare always related in some way. The most common type of relationship is binary (a connection exists between two classes). A binary relationship has a specific direction, which is defined based on role (client or server) of the class.

Rumbaugh and his colleagues [Ref-1] suggest that relationships can be derivedby examining the stative verbs (expressing a state) or verb phrases in the statement of scope or use-cases. Using a grammatical parse, the analyst isolates verbs that indicatephysical location or placement (next to, part of, contained in),

communications (transmitsto, acquires from), ownership (incorporated by, is composed of), and satisfactionof a condition (manages, coordinates, controls). These provide an indication ofa relationship.

The Unified Modelling Language (UML) notation for the object-relationship model has been adapted from the entity-relationship modelling techniques. Objects are connected to other objectsusing named relationships. The cardinality of the connection is specified.

The object relationship model (like the entity relationship model) can be derivedin three steps:

1. Using the CRC index cards, a network of collaborator objects can be drawn. Figure 9 represents the class connections for `SafeHome` objects. First the objects are drawn, connected by unlabelled lines that indicate some relationship exists between the connected objects.
2. Reviewing the CRC model index card, responsibilities and collaborators are evaluated and each unlabelled connected line is labelled. To avoid ambiguity, an arrow head indicates the "**direction**" of the relationship.
3. Once the labelled relationships have been established, each end is evaluated to determine cardinality. As we discussed in introduction, four options exist: 0 to 1, 1 to 1, 0 to many or 1 to many. For example, the `SafeHome` system contains a single control panel (the **1:1** cardinality notation indicates this). At least one sensor must be present for polling by the control panel. However, there may be many sensors present (the **1:m** notation indicates this). One sensor can recognize from 0 to many sensor events (e.g., smoke is detected or a break-in has occurred).

The steps just noted continue until a complete object-relationship model has beenproduced.

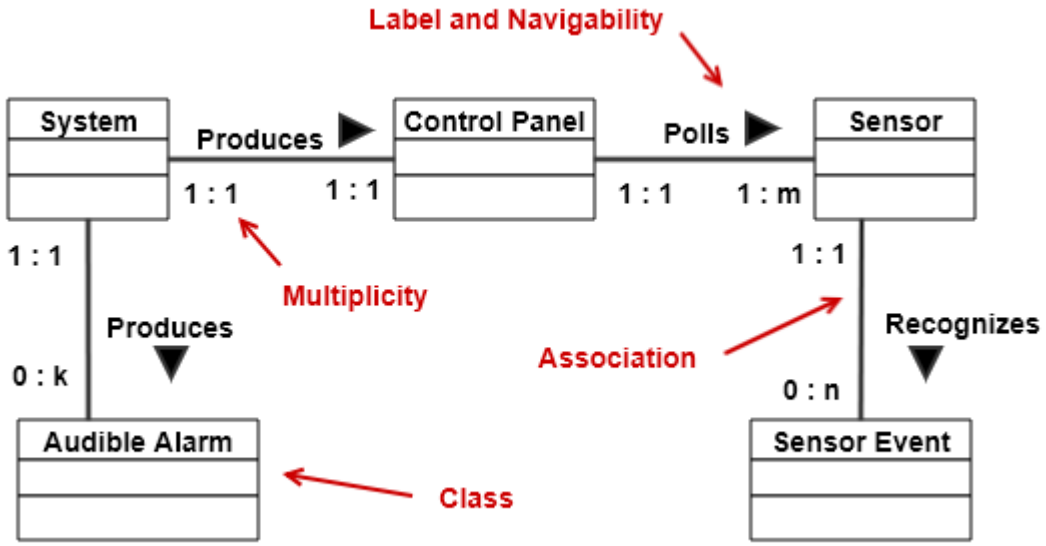
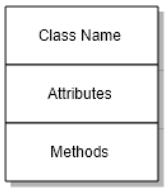


Figure 9 - SafeHome system - Relationships between objects

By developing an object-relationship model, the analyst adds another dimension to the overall analysis model and along with relationships between objects, all important message paths are also defined.

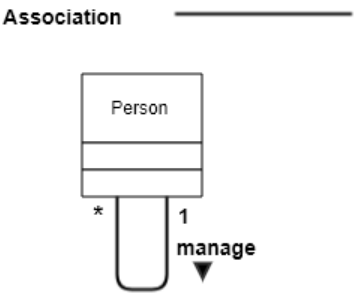
Let's understand each component of Figure 9;

Class: A blueprint of a set of objects that share the same attributes,operations, methods and relationship.

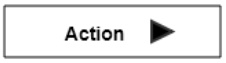


Association: A relationship between two or more classifiers that involves connections among their instances.

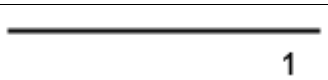
Note: Classes may have association with themselves



Navigability: Indicates that it is possible to move unidirectional across the association from objects of the server to client. Note: Indicated by an arrow head.



Multiplicity: it's not always symmetrical. The relationship of B to A may not be the same as A to B. For example, a system may create many audible alarms but each audible alarms is created by only one system.

Description	Syntax
A is associated with one B	
A is associated with zero or one B	
A is associated with one or more B	
A is associated with zero, one or more B	

By developing an object-relationship model, the analyst adds another dimension to the overall analysis model - Adding all important message path after identifying relationship between objects.

2.4 THE OBJECT-BEHAVIOR MODEL

The CRC model and the object relationship model represent static elements of theObject Oriented analysis model. It's time to understand the dynamic behaviour of the Object Oriented system.

The object behaviour model indicates how an Object Oriented system will respond to external events.

To create the behaviour model, the analyst must perform the following steps:

1. Evaluate all use cases to fully understand the sequence of **interaction** within the system.
2. Identify **events** that drive the interaction sequence and relate to specific objects.
3. Create an **interaction diagram**for each use-case.
4. Build a **state diagram** for the system.

5. Review the object-behaviour model to verify accuracy and consistency.

Let’s understand each step in detail;

2.4.1 Event Identification with Use Cases

The use case represents a sequence of activities thatinvolves actors (user) and the system. In general, an event occurs whenever an Object Orientedsystem and an actor (**an actor** can be a person, a device, or even an external system)exchange information.

Note: An event is Boolean. Event is different from information. Event shows that information is exchanged or not.

A use case is examined for points of information exchange. To understand this, reconsiderthe use-case for `SafeHome`.

1. The homeowner (actor) observes the `SafeHome` control panel to determine if the system is ready for input. If the system is not ready, the homeowner must physically close windows/doors so that the ready indicator is present. [A not-ready indicator indicates that a sensor is open, i.e., that a door or window is open.]
2. The homeowner uses the keypad to key in a four digit password. The password is compared with the valid password stored in the system. If the password is incorrect, the control panel will beep once and reset itself for additional input. If the password is correct, the control panel awaits further action.
3. The homeowner selects and keys in stay (means homeowner is inside the home) or away (means went outside) to activate the system. If Stay, system should activates only perimeter sensors (inside motion detecting sensors are deactivated). Away activate all sensors as system should monitor all scenarios.

Typical events;

- Actor (homeowner) uses the keypad to key in a four digit password.
- Event “password entered” transmitted between homeowner and control panel.

- The event “password entered” doesn’t explicitly change the flow of control, but the result of the event compare password (derived from the interaction “password is compared with the valid password stored in the system”) will have an explicit impact on the flow of control.

Once all events have been identified, they are allocated to the objects involved.Objects can be responsible for generating events (e.g., homeowner generates the “password entered” event)

2.4.2 Interaction Diagrams

Interaction diagrams are models that describe how a group of objects collaborate in a single use case. The diagrams show a number of example objects and the messages that are passed between these objects within the use-case.

The purpose of interaction diagram is −

- To capture the dynamic behaviour of a system.
- To describe the message flow in the system.
- To describe the structural organization of the objects.
- To describe the interaction among objects.

This interactive behavior is represented in UML by two diagrams known as

1) **Sequence diagram (**emphasise the sequence of events)

   Sequence models show the sequence of object interactions that take place;

   - Objects are arranged horizontally across the top
   - Time is represented vertically so models are read from top to bottom
   - Each object has a vertical life-line representing its period of existence
   - Interactions (events) are represented by labelled arrows. Different styles of arrow represent different types of interaction
   - A thin rectangle in an object lifeline represents the time when the object is the controlling object in the system

   Figure 10 is the sequence diagram of our `SafeHome` example.

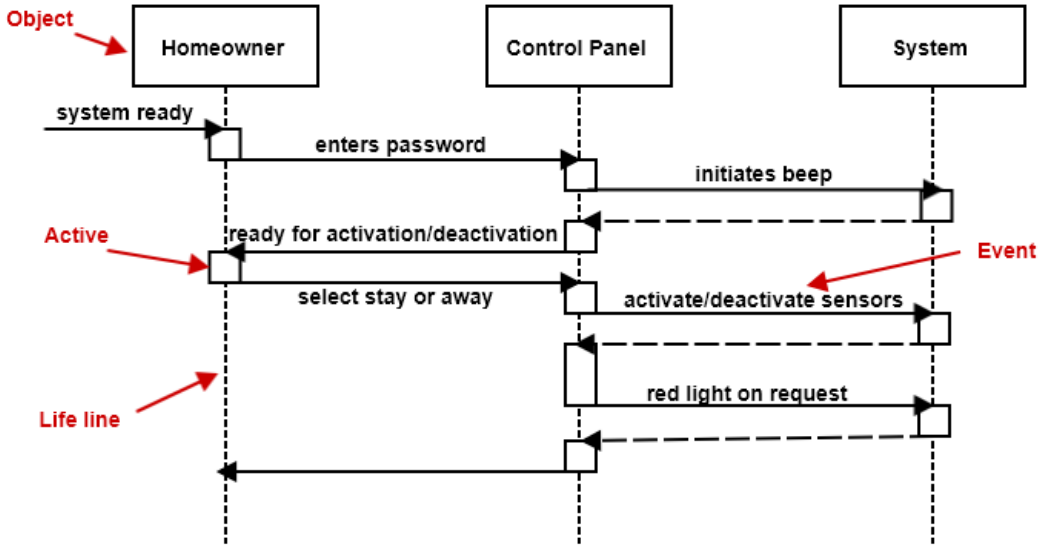


Figure 10 - sequence diagram of SafeHome

2) **Collaboration diagram** (use layout to indicate how objects are statically connected).
   - Objects are shown as icons, arrows indicate messages and sequence is indicated by a decimal numbering scheme.
   - Otherwise similar to sequence diagrams

   Figure 11 is the collaboration diagram for `SafeHome` example.

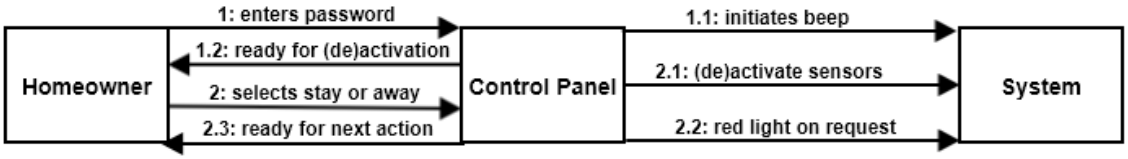


Figure 11 - collaboration diagram for `SafeHome`

2.4.3 State Representations

In the context of the Object Oriented system, Objects can be in either state;

- Passive (current state of attributes)
- Active (undergoes continuous transformation)

An event (sometimes called a trigger) must occur to force an object to make a transition from one active state to another. Figure 12 shows basic state diagram.

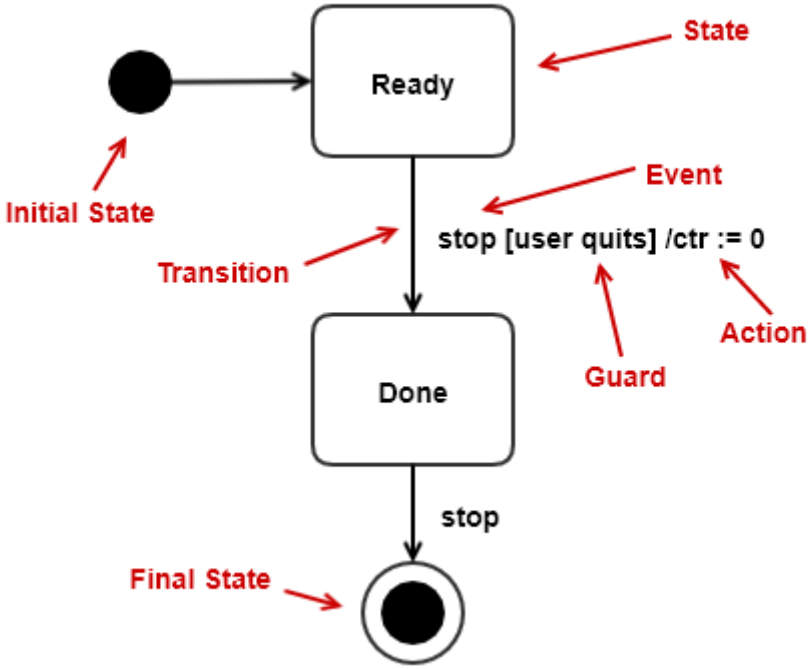


Figure 12 – Basic state diagram

Transitions:

- Three parts, all optional Event [Guard] / Action
  - Events (or triggers)
  - Guard is a logical condition returning “true” or “false”. Transition occurs only if the condition is true. Guards exiting a state must be mutually exclusive.
  - Action represents a processes which occurs quickly and is not interruptible.

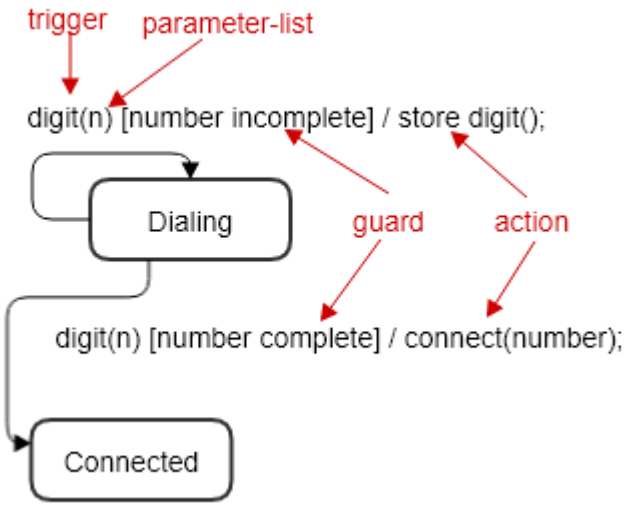


Figure 13 - Transitions

States:

- Two parts, label and activity
  - Activity (with syntax do/activity) represents a process which is longer than a transition action and can be interrupted.

Figure 14 shows a simple representation of active states for the control panel object in the `SafeHome` system.

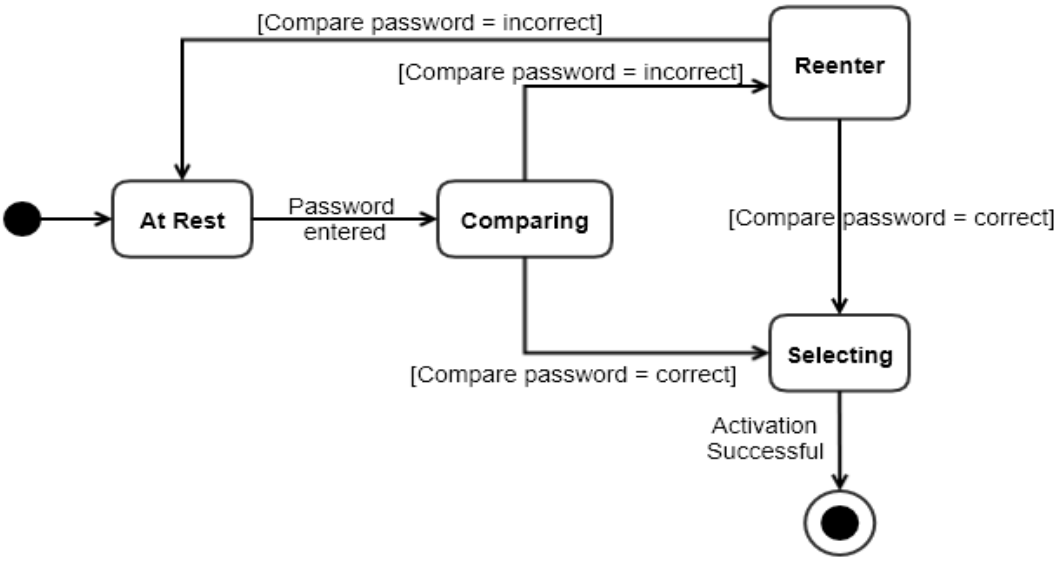


Figure 14 - Active states for the control panel object

Each arrow shown in Figure 14 represents a transition from one active state of an object to another. The labels shown for each arrow represent the event that triggersthe transition. Although the active state model provides useful insight into the "life history" of an object, it is possible to specify additional information to provide more depth in understanding the behaviour of an object. In addition to specifying the event that causes the transition to occur, the analyst can specify a guard and an action. A guard is a Boolean condition that must be satisfied in order for the transition to occur. For example, the guard for the transition from the "at rest" state to the "comparing state" in above Figure can be determined by examining the use-case:

```
if (password input = 4 digits) then make transition to comparing state;
```

In general, the guard for a transition usually depends upon the value of one or more attributes of an object. In other words, the guard depends on the passive state of the object.

An action occurs concurrently with the state transition or as a consequence of it and generally involves one or more operations (responsibilities) of the object. For example, the action connected to the *password entered* event (Figure 14) is an operation that accesses a password object and performs a digit-by-digit comparison to validate the entered password.

Once a complete event trace has been developed, all of the events that cause transitionsbetween system objects can be collated into a set of input events and outputevents (from an object). This can be represented using an event flow diagram.

UML uses a combination of state diagrams, sequence diagrams, collaboration diagrams,and activity diagrams to represent the dynamic behaviour of the objects andclasses that have been identified as part of the analysis model.

2.5 LET US SUM UP

Object oriented Analysis methods help a software engineer to model a problem byrepresenting both static and dynamic characteristics of classes and their relationships as the primary modelling components. Like earlier Object Orientedanalysis methods, theUnified Modelling Language builds an analysis model that has the following characteristics:

1) Representation of classes and class hierarchies,

2) Creation of object relationship models, and

3) Derivation of object-behaviour models.

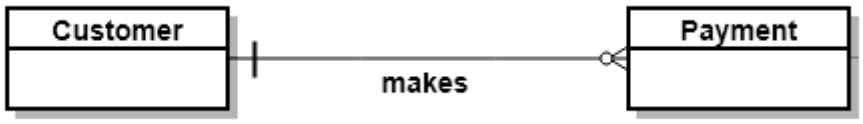
The Object OrientedAnalysis process starts with the definition of usecases (scenarios) that describe “how the Object Orientedsystem is to be used”. The class-responsibility-collaborator (CRC) modelling techniqueis then applied to document classes and their attributes and operations. It alsoprovides an initial view of the collaborations that occur among objects. The next stepin the Object OrientedAnalysis process is classification of objects and the creation of a class hierarchy.Subsystems (packages) can be used to encapsulate related objects. The object relationshipmodel provides an indication of how classes are connected to one another,and the object-behaviour model indicates the behaviour of individual objects and theoverall behaviour of the Object Orientedsystem.

2.6 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Question 1:

Use the scenario described by “A customer can make many payments, but each payment is made by only one customer” as the basis for an entity relationshipdiagram (ERD) representation.

Answer:



2.7 FURTHER READING

Books

- Software Engineering by Roger S. Pressman Sixth Edition McGraw Hill Publications.

Web sites

- OOAD - Dynamic Modelling at tutorialspoint.com. (https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_dynamic_modeling.htm).

2.8 ASSIGNMENTS

1. Write a simple, one-line definition of each of the following key terms. Try to do this without referring back to the notes if possible. For each term, give an example.
    - Cardinality
    - Modality
    - Multiplicity
    - State / transition / event / action
2. What role does cardinality play in the development of an object-relationship model?
3. What is the difference between an active and a passive state for an object?

2.9 ACTIVITIES

Develop a simple presentation on one static or dynamic modelling diagramused in UML. Present the diagram in the context of a simple example, but provideenough detail to demonstrate most important aspects of the diagrammatic form.

2.10 CASE STUDIES

Describe the difference between static and dynamic viewsof an OO system in your own words.

2.11 REFERENCES

- Ref-1: - Rumbaugh, J., et al., Object-Oriented Modelling and Design, Prentice-Hall, 1991

Unit 3: Object Oriented Design

3

Unit Structure

3.1 LEARNING OBJECTIVES

After studying this unit student should be able to:

- Use an object oriented method for designing.
- Specify, analyse and design the use case driven requirements for a system.
- Create class diagrams that model both the domain model and design model of a software system.
- Create interaction diagrams that model the dynamic aspects of a software system.
- Analyse application scenarios and design software systems using object oriented analysis and design.

3.2 INTRODUCTION

Object Oriented System Development Life Cycle consists of three processes;

1) Object Oriented Analysis (OOA)
2) Object Oriented Design (OOD)
3) Object Oriented Implementation (OOI)

In this unit, we are covering Object Oriented Design process.

Object Oriented Design (OOD) is the process of defining the objects and their interactions to solve a problem that was identified and documented during the Object Oriented Analysis (OOA). OOD is a design model that is considered as a blueprint for software construction.

Using OOD we can achieve different levels of modularity. Modularity means organizing major system components into subsystems (modules). Object is a modular form that is the building block of an Object Oriented system. Data and operations (which manipulate the data) are encapsulated into Object.

Object Oriented Design is built upon four software design concepts:

A. Abstraction

- At the highest level abstraction, a solution is stated in higher level terms using the language of the problem context.
- The lower level of abstraction provides a more detailed description of the solution.
- A collection of data that describes a data object is a data abstraction.

B. Information hiding

- Modules must be specified and designed so that the information like algorithm and data presented in a module is not accessible for other modules.

C. Functional independence

- The functional independence is the concept of separation and related to the concept of modularity, abstraction and information hiding.
- The functional independence is accessed using two criteria: Cohesion and coupling.

i. Cohesion

Cohesion is an extension of the information hiding concept.A cohesive module performs a single task and it requires a small interaction with the other components in other parts of the program.

ii. **Coupling**

Coupling is an indication of interconnection between modules in a structure of software.Coupling depends on the interface complexity between modules, the point at whichentry or reference is made to a module, and what data pass across the interface.

D. Modularity

- A software is generally divided into named and addressable components. Sometimes they are called as modules which integrate to satisfy the problem requirements.
- Modularity is the single attribute of a software that permits a program to be managed easily.

OOD is divided into two majoractivities:

1) System design

   System design is the designing the software / application as a whole (at high level) that may include analysis, modelling, architecture, components, Infrastructure etc.

2) Object design

   Object design is the set of defined rules/concepts to implement the functionalities within a software.

Let's take a simple example, a football game.

System design involves the design of football ground, goal poles, grass on the ground, location of the ground, length and width of the ground, putting line marks on the ground, scoreboard etc.

Object design focus on the: how the games needs to be played is defined by set of rules that need to be followed. Thus the players need to play game within the defined rules. So the more the player knows those rules the better they can play the game without making fowls. Similarly the rules for object oriented concept are inheritance, composition, abstraction and encapsulation. Thus the better we know these concepts the better design we can make.

3.3 DESIGN FOR OBJECT ORIENTED SYSTEMS

Figure 1 shows the four layers of the OO design pyramid.

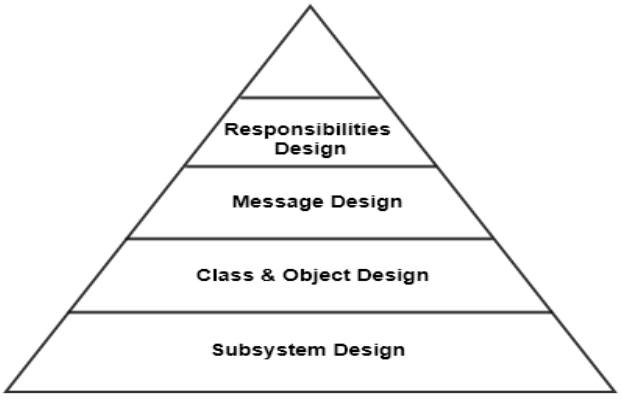


Figure 1 – The Object Oriented Design pyramid

1. **Subsystem layer** contains a representation of each of the subsystems that:
    - Enable the software to achieve its customer-defined requirements.
    - Implement the technical infrastructure that supports customer requirements.
2. **Class and object layer contains:**
    - Class hierarchies that enable system to be created using generalizations
    - Representations of each object.
3. **Message layer:**
    - Contains the design details that enable each object to communicate with its collaborators.
    - Establishes the external and internal interfaces for the system.
4. **Responsibilities layer** contains the data structure and algorithmic design for all attributes and operations for each object.

The design pyramid emphases the layers of the design for a specific product / system.

However there exist another layer of design that forms the base on which the pyramids rests. This foundation layer emphasis on the design of domain objects (known as design patterns). Domain objects play a key role to provide support for interface design, task and data management.

Relationship between the OO analysis model and design model:

OO Analysis Model	OO Design Model
Use cases + Object behavioural model	System design
CRC index cards + Attributes, Operation, Collaboration	Class & Object design
Object Relationship model	Message design
CRC index cards + Attributes, Operation, Collaboration	Responsibilities design

3.3.2 Design Issues

Bertrand Meyer [Ref-3] suggests five criteria for judging a design method's ability to achieve modularity:

1. Decomposability: A modular designmethod facilitates the task of decomposing a problem into a small number of less complex sub-problems, connected by a simple structure, and independent enough to allow further work to proceed separately on each of them.

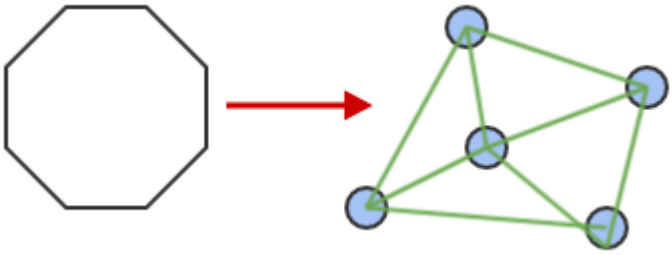


Figure 2 – Decomposability

2. Composability: A modular design method favours the production of software components (modules) which may then be freely combined with each other to produce new systems.Examples: Software libraries (or packages)

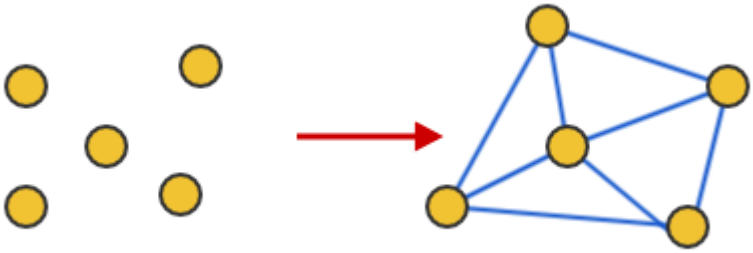


Figure 3 - Composability

3. Understandability: The ease with which a program component can be understood without reference to other information or other modules.
4. Continuity: A method satisfies modular continuityif, a small change in problem specification will trigger a change in just one module or few modules.

5. Protection: A method satisfies modular protection if, the effect of an abnormal condition occurring at run time in a module will remain limited to that module. Example: exception handling

From these criteria, Meyer [Ref-3] suggests five basic design principles that can bederived for modular architectures:

1. Linguistic modular units: Programming language should be capable of supporting the modularity.
2. Few interfaces (low coupling): This principle states that the overall number of communication between modules shouldbe as small as possible.
3. Small interfaces (weak coupling): If any two modules communicate, they should exchange as little information as possible.
4. Explicit interfaces: If two modules communicate, it should be in an obvious and direct way. If we change a module, we need to see what other modules may be affected by these changes. A traceability matrix can be used for this purpose.
5. Information hiding: All information (implementation) about a component should be hidden from outside access.

3.3.3 Steps of Object Oriented Design

To perform object-oriented design, a software engineer should perform the following generic steps:

1. Describe each subsystem and allocate it to tasks.
2. Choose a design strategy for implementing data management, interface support,and task management.
3. Design an appropriate control mechanism for the system.
4. Perform object design by creating a procedural representation for each operationand data structures for class attributes.
5. Perform message design using collaborations between objects and objectrelationships.
6. Create the messaging model.
7. Review the design model and iterate as required.

These design steps are iterative and executed incrementally until a complete design is produced.

3.3.4 A Unified Approach to OOD

UML is organized into two major design activities:

1. **System Design**

   Primary objective of UML system design is to represent the software architecture:

   - ***Conceptual architecture*** is concerned with the structure of the static class model and the connections between components of the model.
   - ***Module architecture*** describes the way the system is divided into subsystems or modules and how they communicate by exporting and importing data.
   - ***Code architecture*** defines how the program code is organized into files and directories and grouped into libraries.
   - ***Execution architecture*** focuses on the dynamic aspects of the system and the communication between components as tasks and operations execute.

2. **Object Design**

   - UML object design focuses on a description of objects and their interactions with one another.
   - Detailed specification of attribute data structures and a procedural design of all operations are created.
   - Visibility for all class attributes is defined and interfaces between objects are elaborated to define the details of a complete messaging model.

3.4SYSTEM DESIGN PROCESS

System design develops the architectural detail which is required to build a system or product.

The system design process includes the following activities:

1. Partition the analysis model into subsystems.
2. Identify concurrency that is dictated by the problem.
3. Allocate subsystems to processors and tasks.
4. Develop a design for the user interface.
5. Choose a basic strategy for implementing data management.
6. Identify global resources and the control mechanisms required to access them.
7. Design an appropriate control mechanism for the system, including taskmanagement.
8. Consider how boundary conditions should be handled.
9. Review and consider trade-offs.

Let's understand each activities in detail;

3.4.1 Partitioning the Analysis Model

Partitioning is one of the fundamental analysis principles. Partitioning the analysis model into interconnected collections of classes, relationships and behaviour.

These design elements are packaged into **subsystem**.

- All of the elements of a subsystem share some property in common.
- All may be involved in achieving the same function.

- They may reside within the same product hardware, or they may manage the same class of resources.

Subsystems are identified by their responsibilities (the *services* that it provides) [Ref-4].

Service: a service is a collection of operations that perform a specific function (like managing file).

Subsystems should follow below design criteria:

- Subsystem should have a well-defined interface through which all communication with the rest of the system occurs.
- Classes within a subsystem should collaborate only with other classes within the subsystem.
- The number of subsystems should be kept low.
- A subsystem can be partitioned internally to help reduce complexity.

When two subsystems communicate with one another, they can establish a client/server link or a peer-to-peer link. **Client / server** link - each subsystem takes on one of the roles (client or server). Service flows from server to client in only one direction. In **peer-to-peer** link - services may flow in either direction.

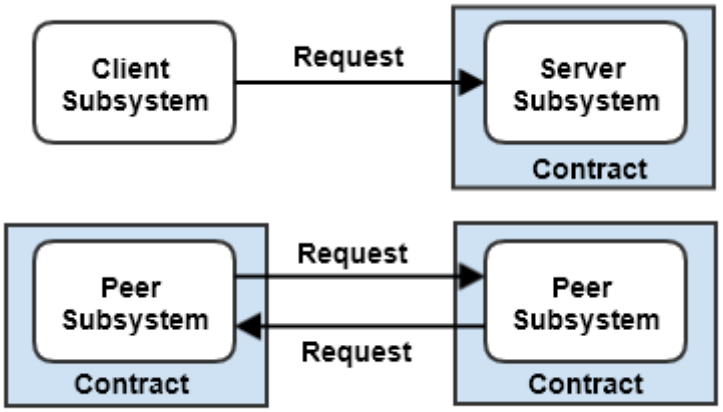


Figure 4 - A model of collaboration between subsystems

Layering occurs when a system is partitioned into subsystems.Each layer of an OO system contains one or more subsystems and represents a different level of abstraction of the functionality [Ref-5]. The levels of abstraction are defined by visibility of subsystemto an end-user.

Example: Four-layer architecture might include:

1. Presentation layer (the subsystems associated with the user interface).
2. Application layer (the subsystems that perform the processing associated with the application).
3. Data formatting layer (the subsystems that prepare the data for processing).
4. Database layer (the subsystems associated with data management).

3.4.2 Concurrency and Subsystem Allocation

Dynamic aspect of the Object Behaviour model provides an indication of concurrency among classes or subsystem.

- If classes (or subsystems) are not active at the same time:
    - No need for concurrent processing.
    - Classes (or subsystems) can be implemented on the same hardware.
- If classes (or subsystems) must act on events at the same time, they are viewed as concurrent.

When subsystems are concurrent, there are two allocation options:

1. Allocate each subsystem to an independent processor (hardware level concurrency).
2. Allocate the subsystems to the same processor and provide concurrency support through operating system features (software level concurrency).

Concurrent tasks are defined [Ref-4] by examining the state diagram for each object. Applications must handle multiple tasks in a manner that simulates parallelism.

3.4.3 The Task Management Component

Coad and Yourdon [Ref-6] suggest the following strategy for the design of the objects that manage concurrent tasks:

- The characteristics of the task are determined.
    - By understanding how the task is initiated.
    - Event-driven and clock-driven tasks are most common.
    - Both activated by an interrupts.
- A coordinator task and associated objects are defined.
- The coordinator and other tasks are integrated.

The priority and criticality of the task must also be determined.

- High-priority tasks must have immediate access to system resources.
- High-criticality tasks must continue to operate even if resource availability is reduced or the system is operating in a degraded state.

Once the characteristics of the task have been determined, need to define coordinator task and associated objects.

The basic task template form: [Ref-6]

Task name—the name of the object

Description—a narrative describing the purpose of the object

Priority—task priority (e.g., low, medium, high)

Services—a list of operations that are responsibilities of the object

Coordinates by—the manner in which object behaviour is invoked

Communicates via—input and output data values relevant to the task

3.4.4 The User Interface Component

The user interface is an important subsystem for most modern applications.Object OrientedAnalysis model containsusage scenarios (called use-cases) and descriptions of the roles that users play (called actors) as they interact with the system.

Once the actor and its usage scenario are defined, a command hierarchy is identified.

1. The command hierarchy defines major system menu categories (the menu bar or tool bar) and all sub-menus.
2. The command hierarchy is refined iteratively until every use-case can be implemented by navigating the hierarchy of functions.

3.4.5 The Data Management Component

Data management includes two distinct areas of concern:

1. The management of data that are critical to the application itself.Usually handled by DBMS
2. The creation of an infrastructure for storage and retrieval of objects.

A Database management system is often used as a common data store for all subsystems. The objects required to manipulate the database are:

1. Members of reusable classes that are identified using domain analysis.
2. Supplied directly by the database vendor.

3.4.6 The Resource Management Component

Subsystem may use different resources like disk drive, processor, database etc.

Software engineer should design a "*Guardian Object*". Rumbaugh and his colleagues [Ref-4] suggest that each resource should be owned by a "guardian object." The **Guardian object** is the gatekeeper for the resource, controlling access to it and moderating conflicting requests for it.

3.4.7 Intersubsystem Communication (Contract Design)

Once each subsystem has been specified, it is necessary to define the collaborations that exist between the subsystems. Figure 4 illustrates a collaboration model. Communication can occur by establishing a client/server link or a peer-to-peer link. Referring to the figure 4, we must specify the contract that exists between subsystems. **Contract:** provides an indication of the ways in which one subsystem can interact with another.

The following design steps can be applied to specify a contract for a subsystem

[Ref-7]:

1. **List each request that can be made by collaborators of the subsystem.**
   Organize the requests by subsystem and define them within one or more appropriate contracts. Be sure to note contracts that are inherited from super classes.
2. **For each contract, note the operations (both inherited and private) that are required to implement the responsibilities implied by the contract.**
   Be sure to associate the operations with specific classes that reside within a subsystem.

3. **Considering one contract at a time, create a table of the form shown below table.**

   For each contract, the following entries are made in the table:

   Type—the type of contract (i.e., client/server or peer-to-peer).

   Collaborators—the names of the subsystems that are parties to the contract.

   Class—the names of the classes (contained within a subsystem) that support services implied by the contract.

   Operation—the names of the operations (within the class) that implement the services.

   Message format—the message format required to implement the interaction between collaborators.

Contract	Type	Collaborators	Class	Operation	Message Format

Draft an appropriate message description for each interaction between the subsystems.

3.5OBJECT DESIGN PROCESS

Bennett and his colleagues [BEN99] discuss object design in the following way:

Object design is concerned with the detailed design of the objects and their interactions. It is completed within the overall architecture defined during system design and according to agreed design guidelines and protocols. Object design is particularly concerned with the specification of attribute types, how operations function, and how objects are linked to other objects.

3.5.1 Object Descriptions

A design description of an object (an instance of a class or subclass) can take one of two forms [Ref-9]:

1. *A protocol description* that establishes the interface of an object by defining each message that the object can receive and the related operation that the object performs when it receives the message.
2. *An implementation description* that shows implementation details for each operation implied by a message that is passed to an object. Implementation details include information about the object's private details; like internal details about the data structures that describe the object's attributes and procedural details that describe operations.

Protocol description is a set of messages and a corresponding comment for each message:

```
//Call a method associated with a buffer object that returns the next value
//   in the buffer
                    v = circularBuffer.Get () ;


//Call the method associated with a thermostat object that sets the
// temperature to be maintained
                    thermostat.setTemp (20) ;
```

Implementation description of an object provides the internal ("hidden") details that are required for implementation but are not necessary for invocation.

An implementation description is collection of the following information:

1. A specification of the object's name and reference to class.
2. A specification of private data structure with indication of data items and types.
3. A procedural description of each operation or pointers to such procedural descriptions.

3.5.2 Designing Algorithms and Data Structures

Representations contained in the analysis model and the system design provide a specification for all operations and attributes.

An algorithm is created to implement the specification for each operation.Data structures (the attributes of a class) are designed concurrently with algorithms.

Operations can generally be divided into three broad categories:

- Operations that manipulate data in some way (like adding, deleting, reformatting, selecting).
- Operations that perform a computation.
- Operations that monitor an object for the occurrence of a controlling event.

Once the basic object model is created, optimization should occur. Rumbaugh and his colleagues [Ref-4] suggest three major thrusts for OOD design optimization:

- Review the object-relationship model to ensure that the implemented design leads to efficient utilization of resources and ease of implementation. Add redundancy where necessary.
- Revise attribute data structures and corresponding operation algorithms toenhance efficient processing.
- Create new attributes to save derived information, thereby avoiding recomputation.

3.5.3 Program Components and Interfaces

An important aspect of software design quality is ***modularity***;

- Object-oriented approach defines the object as a program component that is linked to other components (e.g., private data, operations).
- During design identify the interfaces between objects and the overall structure of the objects.

A program component is a design abstraction, it should be represented in the context of the programming language used for implementation.

Program should be implemented in an object-oriented language like C++, Java, and ADA etc.

3.6 LET US SUM UP

The Object Oriented Design processcan be explained as a pyramid composed of four layers. The *foundation layer* focuseson the design of subsystems that implement major system functions. The *class layer*specifies the overall object architecture and the hierarchy of classes required to implementa system. The *message layer* indicates how collaboration between objects willbe realized, and the responsibilities layer identifies the attributes and operations thatcharacterize each class.

Like Object Oriented Analysis, there are many different Object Oriented Design methods. UML is an attempt to providea single approach to OOD that is applicable in all application domains. UML and other methods approach the design process through two levels of abstraction—design ofsubsystems (architecture) and design of individual objects.

During system design, the architecture of the object-oriented system is developed.

In addition to developing subsystems, their interactions, and their placementin architectural layers, system design considers the user interactioncomponent, a task management component, and a data management component.These subsystem components provide a design infrastructure that enablesthe application to operate effectively. The object design process focuses on thedescription of data structures that implement class attributes, algorithms thatimplement operations, and messages that enable collaborations and object relationships.

3.7 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Question 1: Identify the correct statement(s).

a) OOA is concerned with translating the initial model into a specific model that can be implemented by a software.
b) OOD is concerned with translating OOA model into a specific model that can be implemented by a software.

Answer: b)

Question 2: Which of the following is a concurrency problem?

a) Sales and Vendor sends out messages to another system Stock simultaneously.
b) A sales person and a manager updates the total value of sales for Item A.
c) user A and user B, both readthe Wikipedia page of “Design Pattern”

Answer: a), b)

Explanation: Sales and Vendor sends out messages to system Stock simultaneously. Hencethe messages should reach in the required proper order to ensure correct working of the Sales System. Similarly for option b.

Question 3: Identify the correct statements regarding Coupling and Cohesion.

a) Strongly coupled classes produce better design.
b) Measure of the degree of interdependence between the two modules.
c) Measure of functional strength of modules.
d) None of the above.

Answer: b), c)

Question 4: Use Case diagrams are composed of?

a) Uses, cases
b) People, classes and objects
c) People, computer
d) Actors, use cases

Answer: d)

Explanation: Major components of Use cases are use-cases specifying behaviours expected from the systemand the actor who will be using the system. Actors can be human or non-human.

3.8 FURTHER READING

Books

- Software Engineering by Roger S. Pressman 6thEdition McGraw Hill Publications.

Web sites

- https://en.wikipedia.org/wiki/Object-oriented_design
- https://en.wikipedia.org/wiki/Object-oriented_analysis_and_design
- https://www.tutorialspoint.com/object_oriented_analysis_design/

3.9 ASSIGNMENTS

Question 1: Discuss how the use-case can serve as an important source of information for design.

Question 2: How does a designer recognize tasks that must be concurrent?

3.10 ACTIVITIES

Write a simple, one-line definition of each of the following key terms. Try to do this without referring back to the notes if possible. For each term, give an example.

- Cohesion
- Coupling
- Decomposability
- Continuity

3.11 CASE STUDIES

Describe the difference System Design process and Object design process in your own words

3.12 REFERENCES

- Ref-1: Gamma, E., et al., Design Patterns, Addison-Wesley, 1995.

- Ref-2: Fichman, R. and C. Kemerer, "Object-Oriented and Conceptual Design Methodologies," Computer, vol. 25, no. 10, October 1992, pp. 22–39.
- Ref-3: Meyer, B., Object-Oriented Software Construction, 2nd ed., Prentice-Hall, 1988.
- Ref-4: Rumbaugh, J., et al., Object-Oriented Modeling and Design, Prentice-Hall, 1991.
- Ref-5: Buschmann, F., et al., A System of Patterns: Pattern Oriented System Architecture, Wiley, 1996.
- Ref-6: Coad, P. and E. Yourdon, Object-Oriented Design, Prentice-Hall, 1991.
- Ref-7: Wirfs-Brock, R., B. Wilkerson, and L. Weiner, Designing Object-Oriented Software, Prentice-Hall, 1990.
- Ref-8: Bennett, S., S. McRobb, and R. Farmer, Object Oriented System Analysis and Design Using UML, McGraw-Hill, 1999.
- Ref-9: Goldberg, A. and D. Robson, Smalltalk-80: The Language and Its Implementation, Addison-Wesley, 1983.

Unit 4: Design Patterns

4

Unit Structure

4.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Understand the concept of Design patterns and its importance.
- Make life easier by not reinventing the wheel.
- Understand the behavioural knowledge of the problem and solutions.
- Improve object-oriented skills.
- Recognize patterns in libraries and languages.
- Relate the Creational, Structural, Behavioural Design patterns.
- Apply the suitable design patterns to refine the basic design for given context.

4.2 INTRODUCTION

Each of us has encountered a design problem and always thought: *I wonder if anyone has developed a solution for this?* The answer is almost always— *yes*!

The problem is finding the solution; ensuring that it does fit the problem we have encountered; understanding the constraints that may restrict the manner in which the solution is applied; and finally, translating the proposed solution into design environment.

A design pattern can be characterized as “a three-part rule which expresses a relation between a certain ***context***, a ***problem***, and a ***solution***” [Ref-1].

For software design, ***context*** allows the reader to understand the environment in which the ***problem*** resides and what ***solution*** might be appropriate within that environment.

To understand relationship between context, problem and solution; let’s take a simple example.

A person wanted to travel from Ahmedabad to Mumbai. Here the context is, travelling within India and using available transportation infrastructure (like roads, railway, flight). What are all constrains that will affect this travel problem? Let’s point out some of them.

- Money - person can spend for travelling.

- Time - how fast the person wants to reach Mumbai.
- Purpose - will trip include site-seeing? Or business trip.
- Others - like person wants to travel by his own car.

Based on above constrains, we can define the problem (travelling from Ahmedabad to Mumbai) more effectively. Let's consider different scenarios.

1) If travelling purpose is business trip, travelling time must be minimized hence person should travel by aeroplane.
2) If person has less money, he has only a bicycle, he has more time and trip purpose is to raise money for charity activity. The appropriate solution to the problem based on constrains (in given context) might be a cycle trip from Ahmedabad to Mumbai.

What are Design Patterns?

In 1994, four authors ***Erich Gamma, Richard Helm, Ralph Johnson*** and ***John Vlissides***published a book titled "**Design Patterns - Elements of Reusable Object-Oriented Software"** which initiated the concept of Design Pattern in Software development.

These authors are collectively known as ***Gang of Four (GOF)***. According to these authors design patterns are primarily based on the following principles of object orientated design.

- **Program to an *interface* not an *implementation***
    - Think of an interface as a contract between an object and its clients. That is the interface which specifies the actions that an object can do, and the signatures for accessing those actions.
    - Implementations are the actual behaviours. For example, you have a method `sort()`. You can implement `QuickSort` or `MergeSort`,which should not matter to the client code calling sort as long as the interface does not change.
    - Libraries like the Java API and the .NET Framework make heavy use of interfaces because millions of programmers use the objects provided. The creators of these libraries have to be very careful that they do not change the interface to the classes in these libraries

because it will affect all programmers using the library. On the other hand they can change the implementation as much as they like.

- As a programmer, you should code with interface rather than concrete implementation. So any change in code will not break integration. So benefits of the interface are;
    1. It hides the things you do not need to know thus making the object simpler to use.
    2. It provides the contract of how the object will behave so you can depend on that.

- **Favour object composition over inheritance**
    - Inheritance is referred as “is-a” relationship, and composition is referred as “has-a” relationships.
    - In composition, a class, which desire to use functionality of an existing class, doesn't inherit, instead it holds a reference of that class in a member variable.

Wikipedia [Ref-4] definitionof Design Patterns: “A design pattern is a general repeatable solution to a commonly occurring problem in software design”

Christopher Alexander [Ref-5] first introduced patterns to encode knowledge and experience in designing buildings. He said: *“Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem in such a way that you can use this solution many times over, without ever doing it the same way twice”*.

Characteristics of an effective design pattern;

- *It solves a problem*: Patterns capture solutions, not just abstract principles or strategies.
- *It is a proven concept*: Patterns capture solutions with a track record, not theories or assumption.
- *It describes a relationship*: Patterns don’t just describe modules, but describe deeper system structures and mechanisms.

- *The pattern has a significant human component (minimize human intervention).* All software serves human comfort or quality of life; the best patterns explicitly appeal to aesthetics and utility.

A design pattern saves us from “reinventing the wheel,” or worse, inventing a “new wheel” that is slightly out of round. If used effectively, Design patterns will always make us a better software designer.

4.3 KINDS OF PATTERNS

Human beings are good at pattern recognition, that way we are learning from our past experience. Design patterns are categorised based on the problem it is solving.

Architectural design patterns describe high level design problems that are solved using a structural approach.

Data patterns describe data modelling solutions that can be used to solve frequent data-oriented problems.

*Component design patterns*solve problems related to development ofsubsystems and components. Communicationbetween components and placement within a larger architecture.

Interface design patterns describe common problems related to user interface and the solution within context of specific characteristics of end users

*WebApp design patterns*solve a problem whichis encountered during building a Web Application.

In design patterns book, Gamma and his colleagues (GoF) [Ref-1] focuses on three categories of patterns in which various design patterns can be categorized. (in context of Object Oriented Design): Creational design patterns, Structural design patterns and Behavioural design patterns.

“Creational design patterns”deals with object creation mechanism. Creational patterns are combination of two ideas;

1) Encapsulate knowledge about which concrete classes the system uses.

2) Hide how instances of concrete classes are created and combined.

"Structural design patterns" focus on problem and solution related to how classes and objects are organized and integrated to build a larger application. Structural pattern helps to define relationship between entities.

"Behavioural design patterns" focus on problemrelated to assignment of responsibility and communication between objects.

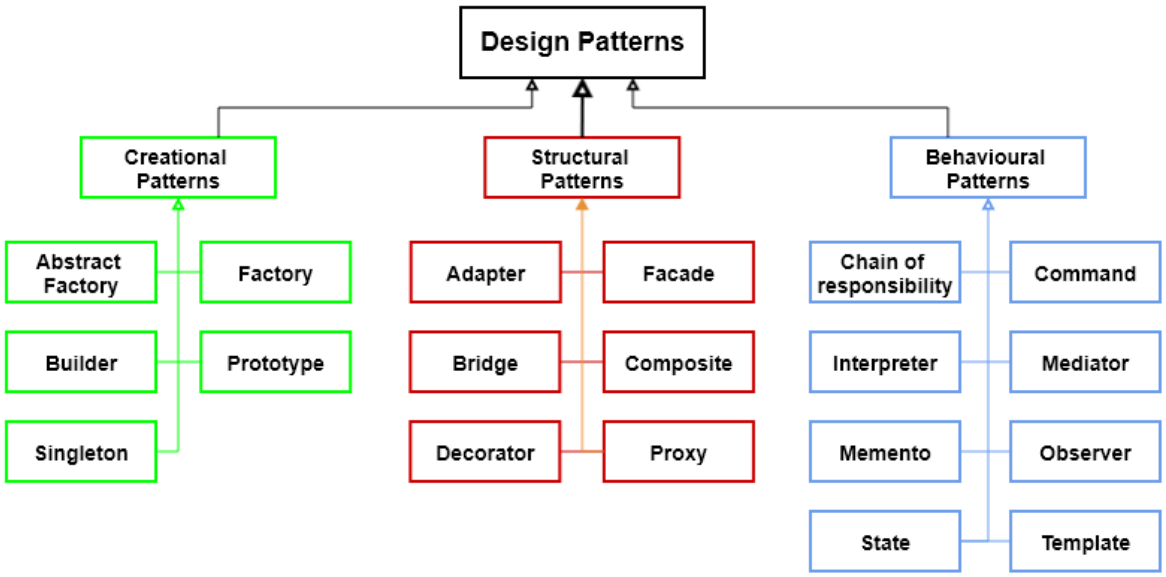


Figure 1 - Types of Design Patterns

Design patterns can be classified by scope, which specifies whether the pattern applies primarily to classes or to objects:

- Class pattern deals with relationship between classes and their subclasses. These relationship are established through inheritance, so they are static.
- Object pattern deals with object relationship, which can be changed at run-time and are more dynamic.

4.3.1 Creational Patterns

Creational patterns emphasize the automatic creation of objects within code, rather than requiring you to instantiate objects directly. So here we are abstracting object creation code from business logic (caller function). It will be easy to modify object

creation whenever it is required without any change or modification in business logic or calling function.

Factory:

Define an interface for creating an object, but let subclasses decide which class to instantiate. The Factory method lets a class defer instantiation it uses to subclasses.

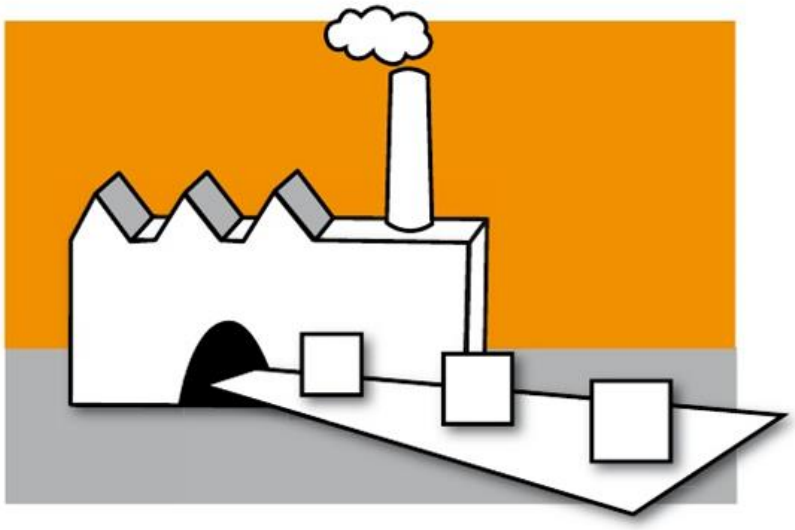


Figure 2 – Factory

Real life example: Vehicle factory, which returns car models based on manufacturer.

Abstract Factory (A factory for factories):

Centralize decision of which factory to instantiate. Encapsulates groups of factories based on common themes. Often uses polymorphism, the concept in object-oriented programming that allows one interface to serve as a basis for multiple functions of different types.

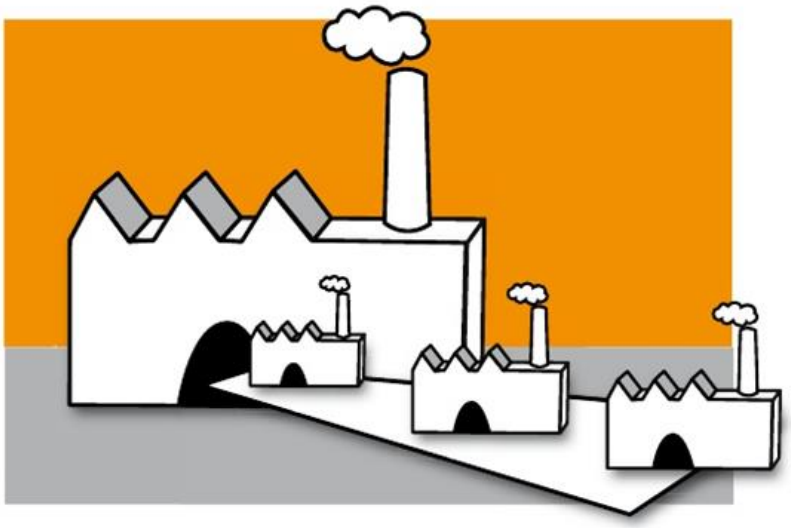


Figure 3- Abstract Factory

Real life example: Providing data access to two different data sources (e.g. a SQL Database and a XML file). You have two different data access classes (a gateway to the data store). Both inherit from a base class that defines the common methods to be implemented (e.g. Load, Save, Delete).Which data source shall be used shouldn't change the way client code retrieves its data access class. Your Abstract Factory knows which data source shall be used and returns an appropriate instance on request. The factory returns this instance as the base class type.

Builder:

Builder pattern is used to construct an object step by step and the final object is returned in last step. The object constructed by Builder patterns are usually a complex object. The process of constructing an object should be generic so that it can be used to create different behaviour of the same object runtime.

GoF [Ref-1] says,“Separate the construction of a complex object from its representation so that the same construction process can create different representations”.

Real life example: Consider construction of a house. House is the final end product (object) that is return as the output of the construction process. It will

have many steps, like floor construction, wall construction, windows and door construction. Finally the whole house object is returned. Here using the same process we can build houses with different properties.

Prototype:

When creation of an object is time consuming, costly and if we have similar object instance in hand, then we should go for prototype **pattern**. Instead of going through a time consuming process to create a complex object, just copy the existing similar object and modify it according to our needs.

Real life example: There is basic bike object with four gears. If suppose someone wants to make a different object say super bike with 6 gears, Instead of creating object from the scratch one can copy the existing instance and make required changes to the copied instance. Thus the prototype design pattern is implemented.

Singleton:
There are two points in the definition of a singleton design pattern,

1) There should be only one instance allowed for a class.
2) Should allow global point of access to that single instance.

As per GoF[Ref-1], “Ensure a class has only one instance, and provide a global point of access to it”.

Real life example:`ApplicationState` is simple example of Singleton design pattern. There will be only one instance of ApplicatoinState which hold information about the state of application.

4.3.2 Structural patterns

As per Wikipedia [Ref-6], “*structural design patterns are design patterns that ease the design by identifying a simple way to realize relationships among entities”.*

The structural design patterns simplifies the structure by identifying the relationships. Structural design patterns are concerned with how *classes* and *objects* can be composed, to form larger structures.

Adapter:(=Wrapper)

Adapter pattern works as a link between two incompatible interfaces. This pattern combines the capability of two autonomous interfaces.Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

Usually this pattern consist of a single class which is responsible to join functionalities of independent or incompatible interfaces. Typically, this is performed by creating a new`ClassNameAdapter` class that implements the interface, allowing for compatibility.

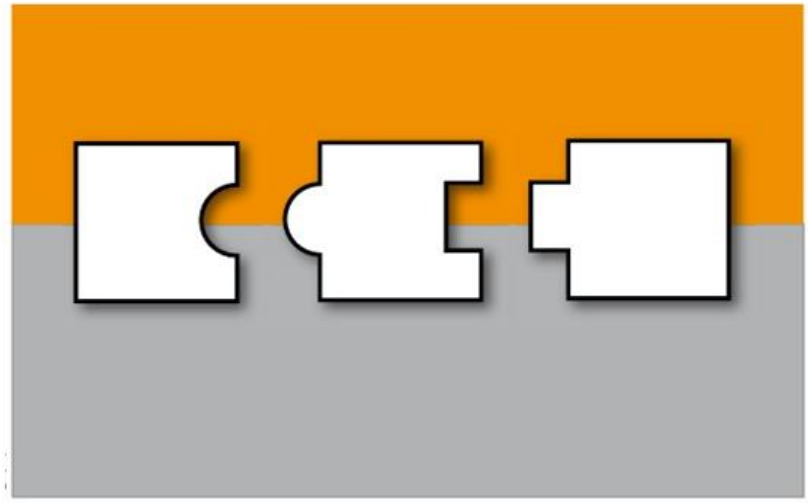


Figure 4 - Adaptor

Real life example: The travel power adapter.American socket and plug are different from British one. Their interface are not compatible with one another. British plugs are cylindrical and American plugs are rectangular. You can use an **adapter** in between to fit an American (rectangular) plug in British (cylindrical) socket.

Bridge:

As per GoF [Ref-1], "Decouple an abstraction from its implementation so that the two can vary independently". Bridge design pattern is a modified version of the notion of "*prefer composition over inheritance*".

Composite:

As per GoF [Ref-1], "*Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.*"

A system made up from subsystems or components. Components can further be divided into smaller components. Further smaller components can be divided into smaller elements. This is a part-whole hierarchy.

> **Real life example:**Graphic editors. Every shape that can be created by graphic editor can be a basic or complex. The example of a simple shape is a line. Complex shape consists of many simple shapes. This shapes (complex and simple) have a lot of operations in common. One of the operations is rendering of a shape to the screen
>
> Other example is Lego building blocks. Using primitive blocks, we can construct multiple windows, doors, walls, and floor first (small structures). Then use all these structure to create a complete entity (house).

Decorator:

Decorator pattern allows a user to add new functionality to an existing object without altering its structure.This pattern creates a decorator class which wraps the original class and provides additional functionality keeping class methods signature intact.

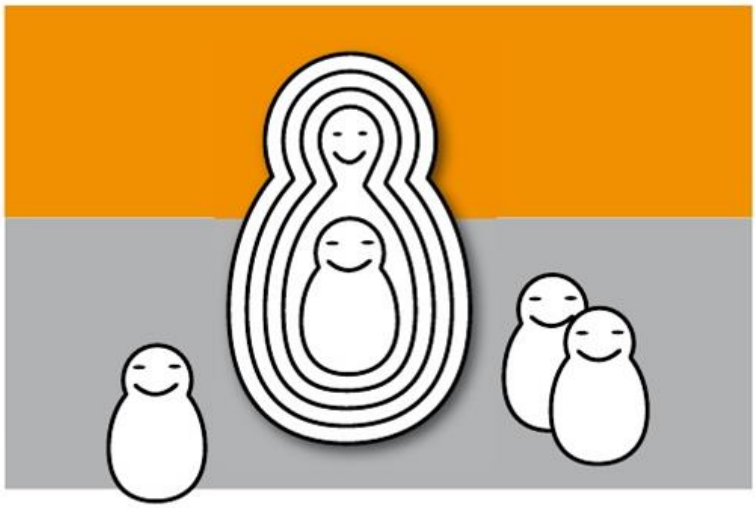


Figure 5 - Decorator

Real life example:A Pizza billing. Pizza shop sell few pizza varieties and provide toppings in the menu. As per the decorator pattern, we need to implement toppings as decorators and pizzas will be decorated by those toppings' decorators. Practically each customer would want toppings of his desire and final bill-amount will be composed of the base pizzas and additionally ordered toppings. Each topping decorator would know about the pizzas that it is decorating and its price. `GetPrice()` method of Topping object would return cumulative price of both pizza and the topping.

Facade:

As per GoF [Ref-1], definition for facade design pattern is, “Provide a unified interface to a set of interfaces in a subsystem. Facade Pattern defines a higher-level interface that makes the subsystem easier to use.”

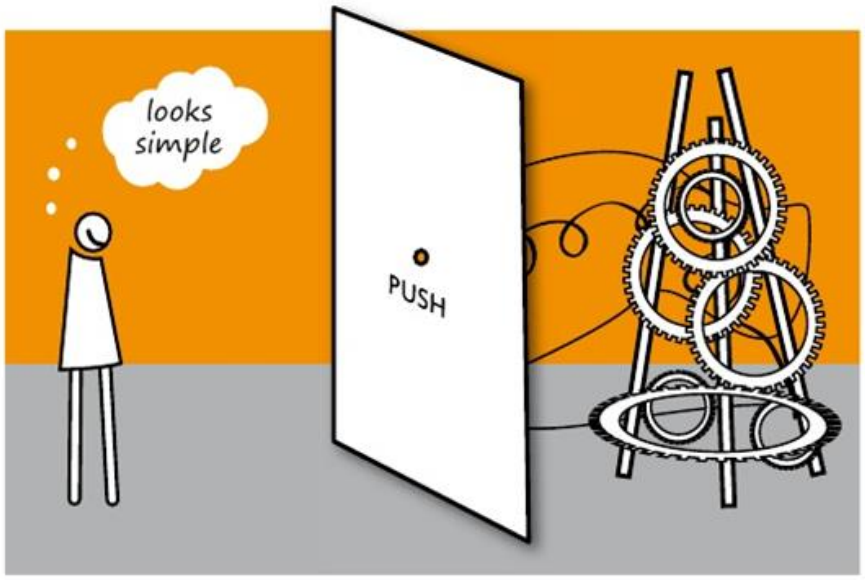


Figure 6 - Facade

Real life example: Start button / key of car. Starting a car involves multiple steps. Imagine how it would be if we had to adjust multiple valves and controllers. The facade we have is just a key hole. On turn of a key it send instruction to multiple subsystems and executes a sequence of operation and completes the objective. All we know is a key turn which acts as a facade and simplifies our job.

Proxy:

GoF [Ref-1] definition of Proxy: “Provide a surrogate or placeholder for another object to control access to it”.

Proxy means ‘in place of’ or ‘Representing’ or ‘in place of’ or ‘on behalf of’ are literal meanings of proxy and that directly explains proxy design pattern.

Real life example:ATM. ATM is holding proxy objects for bank information that exists in the remote server.

4.3.3 Behavioural Patterns

Behavioural patterns are concerned with communication and assignment between objects.

Chain of Responsibility:

Chain of responsibility helps to decouple sender of a request and receiver of the request with some adjustments. Chain of responsibility is a design pattern where a sender sends a request to a chain of objects, where the objects in the chain decide themselves who act on the request. If an object in the chain decides not to serve the request, it forwards the request to the next object in the chain.

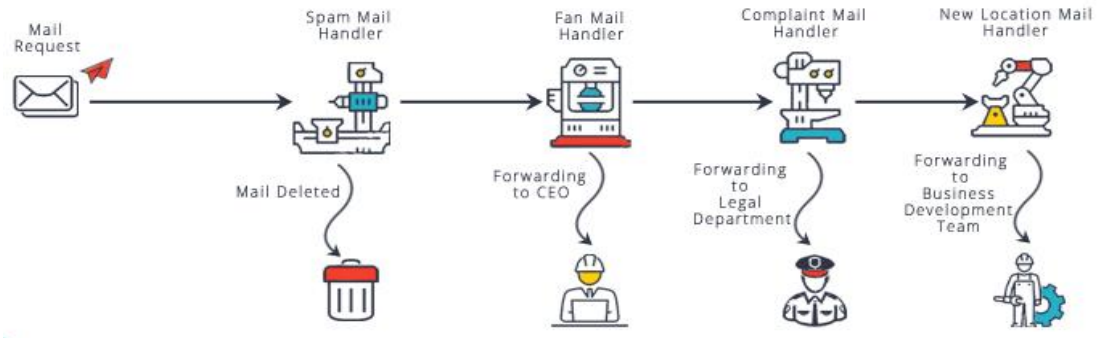


Figure 7 – Chain of responsibility

Real life example: Email classifier. On each email request, classifier will classify email in different categories and act based on configured responsibility.

Command:

Command design pattern is used to encapsulate a request as an object and pass to an invoker, wherein the invoker does not know how to service the request but uses the encapsulated command to perform an action.

Real life example: Universal Remote.

Interpreter:

Interpreter design pattern gives the ability to define a language grammar with an interpreter, where in that interpreter uses that definition to interpret sentences of that language.Typically, each symbol is defined by one class, and then a syntax tree is used to parse (interpret) the overall sentence.

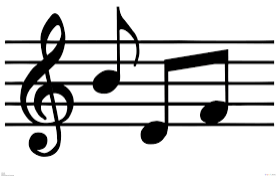


Figure 8 - Interpreter

Real life example: Musicians are examples of Interpreters. The pitch of a sound and its duration can be represented in musical notation on a staff. This notation provides the language of music. Musicians playing the music from the score are able to reproduce the original pitch and duration of each sound represented.

Mediator:

Generates a third party object (mediator) that acts as a go-between for interactions between two other similar objects (colleagues). Commonly, this is used when multiple objects need to communicate, but do not (or should not) be aware of the others respective implementation or behaviour.

Real life example: Air traffic controller (ATC). ATC is a mediator between flights. It helps in communication between flights and co-ordinates/controls landing and take-off. Two flights need not interact directly and there is no dependency between them. This dependency is solved by the mediator ATC.

If ATC is not there all the flights have to interact with one another and managing communication becomes quite difficult.

Memento:

Stores the state of an object, allowing for restoration (rollback) of the object to a previous state. This behaviour is well-known when using word processors that implement the undo (Ctrl + Z) feature.

Real life example: Undo or backspace or ctrl+z is one of the most used operation in an editor. Memento design pattern is used to implement the undo operation. This is done by saving the current state of the object as it changes state.

Observer:

In observer design pattern multiple observer objects registers with a subject for change notification. When the state of subject changes, it notifies the observers. Objects that listen or watch for change are called observers and the object that is being watched for is called subject.

Real life example: Newspaper subscription and Stock system which provides data from several companies.

State:

Allows for the behaviour of a class to change based on the current state. While these states are often changed throughout execution, the implementation of each possible state is typically defined by a unique class interface.

Real life example: The DVD player. Let's assume that the DVD player has two buttons: play and menu. The DVD player will behave differently when you push these button according to its current state. When we press play button, the state of button change and now it will act as "stop" button.

Template:

Template method design pattern is to define an algorithm as skeleton of operations and leave the details to be implemented by the child classes. The overall structure and sequence of the algorithm is preserved by the parent class.

Real life example: Order processing flow. The OrderProcessTemplate class is an abstract class containing the algorithm skeleton but individual operations are performed vary depending on the subclass.

4.4 DESCRIBING A DESIGN PATTERN

Each pattern has four important elements:

- The ***pattern name*** is a handle we can use to describe a design problem, its solutions and consequences in a word or two. Naming a pattern immediately increases the design vocabulary of programmers. Having a vocabulary for patterns enables to talk about patterns with other programmers, in the documentation, etc.
- The ***problem*** describes when to apply the pattern. It explains the problem and its context. It might describe specific design problems such as how to represent algorithms as objects. Sometimes the problem includes also a list of conditions that must be met before it makes sense to apply the pattern.
- The ***solution*** describes the elements that make up the design, their relationships, responsibilities, and collaborations. The solution doesn't describe a particular concrete design or implementation, because a pattern is like a template that can be applied in many different situations. Instead, the pattern provides an abstract description of a design problem and how a general arrangement of elements (classes and objects) solves it.
- The ***consequences*** are the results and trade-offs of applying the pattern. They may address language and implementation issues as well. Since reuse is often a factor in object-oriented design, the consequences of a pattern include its impact on a system's flexibility, extensibility, or portability.

The Gang of Four (GoF) [Ref 2] used a consistent format to describe patterns. They developed a template for describing a design pattern. The template provides a uniform structure to the information and make design patterns easier to learn, compare and use. This template describes a design pattern with:

Design Pattern Template:

Pattern name—describes the essence of thepattern in a short but expressive name

Problem—describes the problem that the pattern solve

Motivation—provide a typical scenario (example) that illustrates the design problem

Context—describes the environment in which the problemresides including the application domain

Solution—provides a detailed description of the solutionproposed for the problem

Intent—describes the pattern and what it does

Consequences—describes the potential trade-offs that mustbe considered when the pattern is implemented andthe consequences of using the pattern

Implementation—implementation steps along with sample code

Related patterns—cross-references related design patterns

We should be careful when we define pattern name. Because biggest technical issue is finding existing pattern from pattern repository. If we give meaningful pattern name, it will help us to find “right” pattern.

Pattern template provides a standard for describing a design pattern and capture characteristics which can be searched easily.

4.6 LET US SUM UP

Design patterns represent the best practices used by experienced object-oriented software developers. Design patterns are solutions to general problems that software

developers faced during software development. These solutions were obtained by trial and error by many software developers over quite an extensive period of time.

Design patterns provide an organised mechanism for describing problems and their solutions in given context. Design patterns help the software engineering community to understand design knowledge and way to reuse it.

Note: It is very critical to use correct design patterns to solve the specific problem. Design patterns are a double-edge sword, if used in the wrong context, they can make things worse. If use correctly, they help us to solve problem effectively.

4.7 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Question 1: A design pattern used to enhance the functionality of an object is

(a) Adapter

(b) Decorator

(c) Delegation

(d) Proxy

Answer: b

Question 2: A design pattern often used to restrict access to an object is

(a) Adapter

(b) Decorator

(c) Delegation

(d) Proxy

Answer: d

Question 3: You have a class that accepts and returns values in unit kilometre, but you need to use mile as unit. The design pattern that would best solve your problem is ;

(a) Adapter

(b) Decorator

(c) Delegation

(d) Proxy

Answer: A

Question 4: Why should we use Design Patterns?

Design patterns are, by principle, well-thought out solutions to programming problems. Many programmers have encountered these problems before, and have used these 'solutions' to solve them. If you encounter these problems, why recreate a solution when you can use an already proven one.

4.8 FURTHER READING

Books

- Software Engineering by Roger S. Pressman 5th, 6th, 7thEdition McGraw Hill Publications.

Web sites

- https://www.tutorialspoint.com/design_pattern/
- https://en.wikipedia.org/wiki/Software_design_pattern
- https://javapapers.com/design-patterns/builder-pattern/

4.9 ASSIGNMENTS

Question 1: Write three “parts” of a design pattern and provide an example of each parts.

Question 2: How do Creational design patterns are different from Behavioural design pattern?

4.10 ACTIVITIES

Activity 1: Develop a complete pattern description for any one design pattern (you can use design pattern template discussed in section 11.4).

4.11 CASE STUDIES

Build a pattern-organizing table for the patterns we learn in this unit.

4.12 REFERENCES

- Ref-1: Gamma, E., et al., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.
- Ref-2: Alexander, C., The Timeless Way of Building, Oxford University Press, 1979.
- Ref-3: Ambler, S., Process Patterns: Building Large-Scale Systems Using Object Technology, ambridge University Press/SIGS Books, 1998.
- Ref-4: https://en.wikipedia.org/wiki/Software_design_pattern
- Ref-5: https://en.wikipedia.org/wiki/Design_pattern
- Ref-6: https://en.wikipedia.org/wiki/Structural_pattern
- Ref-7: Software Engineering by Roger S. Pressman 5th, 6th, 7th Edition McGraw Hill Publications.

Block-2
Introduction to Web Engineering

Unit 1: Introduction to Web Engineering

1

Unit Structure

1.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Understand what is Web-Engineering (WebE) is. Learn techniques and evaluation metrics for ensuring the proper operability, maintenance and security of a web application.
- Independently learn and apply new web technologies.
- Recognize the fundamentals of web technologies.
- Understand the advantages and disadvantages of the web technologies.
- Analyse and design comprehensive Web application.
- Learn techniques and evaluation metrics for ensuring the proper operability, maintenance and security of a web application.

1.2 INTRODUCTION

The World Wide Web (WWW) and the Internet that empowers it are perhaps the most significant developments in the history of computing. These technologies have drawn all of us (with many more who will sooner or later follow) into the information age. WWW and Internet has great impact on our lives and became integral part of our life. As the web became widespread the span of application also increased from normal static web pages to advanced dynamic applications and arise the need of approach that engineer such applications efficiently and thus Web engineering evolved.

In this unit we will discuss *Web Application* and *Web Engineering*. We will explore Web Engineering and understand concepts behind it. We will also explore various aspects and attributes of Web Engineering, Web Engineering layers and Web engineering process. We will also discuss about Web-Engineering team in details.

1.3 WEB ENGINEERING

A Web Application (also known as **WebApp**) is an application program that is stored on a *server* (in most cases remote server) and delivered over the Internet through a *web browser* interface.

World Wide Web was originally designed to present information to Web surfers using simple web sites that contained primarily of hyperlinked text documents. However, as web 2.0 evolves, WebApps are evolving into sophisticated computing environments that not only provides stand-alone features, computing functions and content to the end users, but also are integrated with corporate databases and business applications. Modern WebApps are mostly large-scale software applications for e-commerce, information distribution, entertainment, collaborative working and numerous other activities. WebApps run on distributed hardware platforms, implemented in several languages and styles, interfaced with other websites and databases.

As WebApps becomes increasingly integrated in business strategies for small and large companies, the need to build reliable, usable, and adaptable systems has become very critical. That's why a disciplined and systematic approach to WebApp development was necessary. This approach is **web engineering.** Web Engineering (**WebE**) define the approaches, methodologies, techniques, and tools that are the basis of WebApp Development and which support their design, development and evaluation.

Note: Even though WebE uses Software Engineering principles, concepts and methods, it involves modified, improved or different methodologies, tools, techniques, and guidelines to meet the unique requirements of WebApps Development.

1.3.1 Attributes of Web-Based Systems and Applications

It is important to note that even though most of the WebApps are unique and different from each other, there are a few attributes that any good WebApps architecture should display. According to Powell the following attributes are encountered in the vast majority of WebApps [Ref-2].

- **Network intensiveness.**

  A WebApp works on a network and must serve to different types of clients. A WebApp may work on Internet (global communication accessed through the

Web), Intranet (communication within a single organization) or an Extranet (inter-intranet communication).

- **Concurrency.**

Modern WebApps are simultaneously used by many users and provide increasingly interactive features. A large number of users may access the WebApp at the same time and in many cases, the patterns of usage among end users will vary greatly, hence concurrency is very critical for WebApp's performance.

- **Unpredictable load.**

This attribute is also known as scalability. Scalability is the ability of a WebApps to handle increases in load without impacting the performance. For example, 200 users may use the application on Tuesday and 1000 user may use the system on Sunday. So, WebApps should be capable of handling such variation in load.

- **Performance.**

Performance is a measure of the responsiveness of an application to execute specific actions in a particular time interval. It can be evaluated in terms of latency or throughput. *Latency* is the time taken to respond to any event. *Throughput* is the number of events that take place in a given amount of time. WebApp user should get response in reasonable time. WebApps should support an appropriate level of performance.

- **Availability.**

WebApps user always asks for 100% availability (24 hours in a day / 7 days in week / 365 days in year). In order to serve different range of customers, WebApps should have high availability.

- **Data driven.**

WebApps commonly process or use information that exists on databases that are not an integral part of the Web-based environment. WebApps like e-commerce or financial applications commonly use databases to access information.

- **Content sensitive.**

An important element of WebApps is quality of content. The quality and artistic (look and feel) nature of content is an important factor of the quality of a WebApp.

- **Continuous evolution.**

Conventional application software evolves over a series of planned and timely releases. But WebApps evolve continuously. In practice we can come across many WebApps whose contents gets updated on every minute. On the other end there are some applications in which for every request content are independently created or generated. A good initial design and architecture should allow continuous growth to occur in a fairly controlled and consistent manner.

- **Immediacy.**

Web application has an attribute immediacy that is rarely found in conventional types of software. Due to market compaction, company need to get application to market quickly. It simply means that the time to market a complete web application can be a matter of a few days or weeks but the web engineer must use the efficient methods for planning, analysis, design, implementation and testing and develop the web application in a short and tight time schedule.

- **Security.**

Security is the capability of a web application to minimise the chance of malicious or accidental actions outside of the intended usage of the web application, and prevent disclosure or loss of information. As Web Applications are available via network access, it is difficult, to limit the end-users who may access the application. In order to safeguard sensitive information and provide secure modes of data transmission, strong security measures must be implemented and followed within the application throughout the infrastructure that runs or supports a WebApp. WebApps must follow security best practices and adhere to security standards wherever possible.

- **Aesthetics.**

Aesthetics is WebApp's ***look and feel***. Look and feel are equally important as technical design. This attribute gets affected by user's cultural background, education, class or personal choice, and these may influence his or her judgments of aesthetic value. One need to consider these aspects in order to

match them with the appropriate web application design. Thus, as aesthetics involves multiple dimensions of a target users, it demands insight into diverse areas relevant to an applications distribution, including foreign market considerations. Here look and feel does not only mean the use of images, graphics or the audio-visual interface intensively but interactivity and easiness to the user, which makes the application easy to access and provides value added features.

Above discussed are some general attributes applicable to the web applications but their influence and impact may vary from application to application.

1.3.2 Application categories

According to Dart, the following application categories are most commonly encountered in WebE work [Ref-3],

- ***Informational*** - Read-only content is provided with simple navigation and links.
- ***Download*** - A user downloads information from the appropriate server.
- ***Customizable*** - The user customizes content to specific needs.
- ***Interaction*** - Communication among a community of users occurs via chatroom, bulletin boards, or instant messaging.
- ***User input*** - Forms-based input is the primary mechanism for communicating need.
- ***Transaction-oriented*** - The user makes a request (like places an order) that is fulfilled by the WebApp.
- ***Service-oriented -*** The application provides a service to the user, like assists the user in making a loan payment.
- ***Portal*** - The application connect the user to other Web content or services outside the domain of the portal application.
- ***Database access*** - The user queries a large database and extracts information.
- ***Data warehousing*** - the user queries a collection of large databases and extracts information.

1.3.3 WebApp Engineering Layers

The development of Web-based systems and application required specialized process models. WebE is not a perfect clone of Software engineering but WebE adapts many of software engineering's basic concepts and principles.

Process, methods and technologies (tools) provide a layered approach to WebE.

- **Process**

In Web Application even when cycle time is rapidly changing and it dominate development thinking, it is important to understand that the problem must still need to be analysed carefully, an appropriate design should be developed, implementation should advance in an incremental fashion, and well-planned testing strategy must be initiated. In short, process is often *agile* and is almost always incremental. However, these framework activities must be defined within a process that:

A. Accept change.
B. Encourages the creativity and independence of development staff and strong interaction with WebApp stakeholders.
C. Builds systems using small development teams.
D. Emphasizes incremental development using short development cycles [Ref-4].

- **Methods**

The WebE methods are fundamentally a set of technical tasks that helps a Web engineer to conceptualize, characterize, and then build a high-quality WebApp.

At a high level, these methods can be categorized in the following types:

- **Communication methods**

  Define the approach used to ease communication between Web engineers and all other WebApp stakeholders (for example, team leaders, project managers, end-users, business clients, domain experts, content designers). Communication methods are mainly important during requirements gathering and whenever a WebApp increment is to be assessed.

- **Requirements analysis methods**

These methods provides a basis for understanding “what needs to be done”. By this methods one can understand content to be delivered by a WebApp, and the methods of interaction that individual class of user will entail as navigation through the WebApp occurs.

- **Design methods**

  Design methods incorporates a series of design techniques that address WebApp content, application and information architecture design, interface design, and navigation structure.

- **Testing methods**

  Testing methods incorporate formal technical reviews of the content and design model and a wide range of testing techniques that tests and validates component level and architectural issues. Testing methods also includes various types of testing, for example, usability testing, navigation testing, security testing, performance testing, configuration testing etc.

- **Tools and Technology**

These are the tools and technologies covering a wide array of content description and modelling languages (for example, HTML, XML), component based development resources, programming languages, browsers, multimedia tools, database connectivity tool, servers and servers utilities, site management & analysis tools and many others used by WebE teams.

1.3.4 The Web Engineering Process

The attributes of Web applications have a deep impact on the WebE process that is chosen. The *Web engineer* chooses a process model based on the attributes of the web application that is to be developed.

For example an *agileprocess* model is more suitable in case of web application where immediacy and continuous evolution attributes are principal attributes because agile process has frequent releases and rapid iterations. On the other hand, *incremental process* model might be chosen if a WebApp is to be developed over a longer time period (like a major e-commerce application). For these application, architecture needs to be specialized (so need to emphasize more on requirements

analysis, modelling and design). Many a times a parallel activities are carried out to save time and expedite development process that involve a team of both *technical* and *non-technical* members (like web engineer, copywriters and graphic designers).

Defining the Framework

Before proceeding Let's understand Agile process, An Agile process is driven by customer requirements, work on short-lived plans, and develops application iteratively and delivers multiple increments and it adapts as changes occur. Any one of the agile process models (e.g., Extreme Programming, Adaptive Software Development, SCRUM) can be applied successfully as a WebE process.

The web engineering process should be adaptability. Before we define a process framework for WebE, as mentioned in [ref 1] we must understand following points:

1. *WebApps are often delivered incrementally*. That is, framework activities will occur repeatedly as each increment is engineered and delivered.
2. *Changes will occur frequently*. These changes may occur as a result of the evaluation of a delivered increment or as a consequence of changing business conditions.
3. *Timelines are short*. This mitigates against the creation and review of large engineering documentation, but it does not prevent the simple reality that critical analysis, design, and testing must be recorded in some manner.

With these points in mind, let's discuss the WebE within generic process framework.

Customer communication: As the name suggest this framework activity include communication the customer and other stakeholders of the application. This activity includes requirements gathering and integration between the WebApp and other business application and databases identification.

Planning: Establishes an incremental plan for the WebE project. The plan consists of a task definition and a timeline schedule for the time period which is usually in weeks.

Modelling: This include the construction of the models that assist the developer and the customer to better understand WebApp requirement and the design. The main intention of the task is to develop rapid analysis and design model that define requirements.

Construction: Here WebApp is constructed using web engineering tools and technologies as modelled (for example, this combines both the generation of HTML, XML, Java and similar code). Once the web increment is constructed a round of rapid testing is carried out to uncover errors in the code or design.

Deployment: In deployment, WebApp is configured for operational environment and setup; then delivered to customer who evaluates it and provides feedback based on evaluation. Upon receiving feedbacks WebE team modifies incremental as required.

Below Figure 1, depicts WebE framework activities that are applied using an incremental process flow.

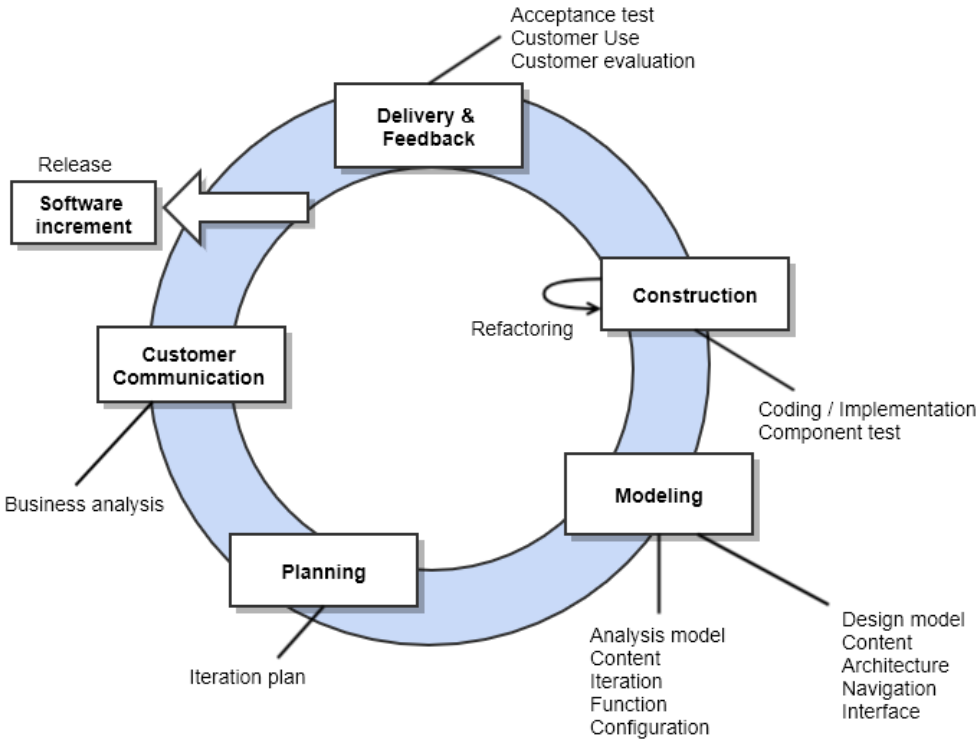


Figure 1 – WebE process

1.3.5 Web Engineering Best Practices

Web engineering teams sometimes take short-cuts due to time pressure and result in more development effort. To build industry quality WebApps, we need to define a set of best practices.

1. **Take time to understand business requirement.**

Many WebApp developers starts development even requirements / objectives of WebApp is not clear. It is not a good practice to start work when requirements are not clearly understood.

2. **Describe how users will interact with the WebApp using a scenario-based approach.**

   Customer should be convinced to describe use-cases to reflect actual business requirements. These scenarios can be used for project planning and analysis & design modelling.

3. **Always develop a Project Plan, even if it is very brief.**

   The project should be scheduled and tracked on a daily basis.

4. **Spend some time for modelling – what you are going to build.**

   UML diagrams (like class diagrams, sequence diagrams and state diagrams) may provide valuable vision.

5. **Review the models for consistency and quality**

   To develop high quality WebApp, formal technical reviews should be conducted throughout a WebE project.

6. **Use tools and technologies that enable to construct reusable components**

   Reusable components help developer to do rapid development and less errors.

7. **Test first, then deploy**

   Design comprehensive tests and execute them before releasing the project to customer.

1.6 THE WEB ENGINEERING TEAM

Teamwork is very essential in almost every arena of life and web engineering is not different from it. A successful Web engineering team is made up of a wide variety of talents which work together as a team in a high-pressure and high-performance project environment. Each individual in the web engineering team have their own specialty and role, and they all work towards a common goal. In web projects timelines are short, changes are frequent, and the technology keeps changing hence creating a team that performs utmost is very critical and tough task. And now as you

begin understanding web engineering, it is essential that you know about some important roles that will be part of web engineering team.

A lot of cutting-edge software technologies, high-speed hardware, and huge financial capital go in the making of a web development project. However, none of it will be executed well, if web development team is not well-skilled & qualified, and failed to deliver value to the end-user, hence the team is the greatest asset.

1.6.1 The Players

As we understand from above discussion that the creation of a successful Web application demands a wide-ranging skill set.

At high level Web engineering teams are organized in much the same way as conventional software teams. However, the players and their roles are often quite different. Some of the skill sets that WebE team must possess as a whole and must be distributed across WebE team members are component-based software engineering, networking, architectural and navigational design, Internet standards/languages, human interface design, graphic design, content layout, and WebApp testing.

The following roles should be distributed among the members of the WebE team;

- **Content developers/providers.** Content developers are fundamentally involved in the creation, development, and editing of content for WebApps typically using content management system tools (CMS). WebApps are basically content driven and content based. There must be at least one role in WebE team which must focus on the generation and/or collection of content. Here it's important to recall that content usually comes from a wide range of data objects, content developers / providers may come from diverse and many a times non software backgrounds. Content developers should not only have strong familiarity with application content but also with the overall business or organization behind the web app. A web content developer is responsible for creating original content for WebApps based on the requirements of a client/organization. The web content specialist should also have strong copywriting and editing skills. Web content specialists may also work with media files such as audio, video or interactive (Flash), and therefore may need those skills or may work with another media specialist external to the core web team to develop rich media content.

- **Web publisher**. As we discussed above, content is created by content developer/provider. Web publisher incorporates these content in to web app in structured and organized way. In addition, Web Publisher act as bridge between engineering staff who have technical background and non-technical content developer/providers. So, Web Publisher must have good understanding of Web technologies and content.
- **Web engineer.** It's believed that "From the minute someone enter in to web application, until the moment they complete their work with web application, they are in the web engineer's world". Web engineer is a role which is using various technological knowledge to improve application and user experience. A Web engineer usually gets involved in a broad range of activities during the development of a WebApp, some of them are eliciting requirements, analysing models, various design including navigational, architectural and interface designs, actual implementation or coding and testing. To carry out these works efficiently web engineer must possess good technical knowledge and conceptual understanding of component technologies, client/server architectures, HTML/XML, Client side scripting (for example, JavaScript, or JavaScript-based tools like jQuery) and server side scripting (for example, PHP, Python, Ruby, ASP.NET, or compiled languages like C++ or Java), database technologies and most of the time current trends in web app development. In addition, it is desirable for web engineer to possess basic understanding of hardware and software platforms, security, multimedia concepts and Web app sustaining issues.
- **Business domain experts**. A business domain expert is a person with special knowledge or skills in a particular area of business domain. Domain expert possess deep understanding of the real business and issues related to it. Theoretically Business domain experts should be able to answer all the questions related to the business like, business goals, objectives and requirements associated with the WebApp.
- **Support specialist.**In general terms responsibilities of support specialist include resolving customer queries, recommending solutions and guiding product users through features and functionalities. This role is responsible for continuing WebApp support. As we know that WebApps evolves continuously, the support

specialist is responsible for rectifications, adaptations, and enhancements to the WebApp, including content updates, implementation of new procedures and forms, and changes to the navigation pattern.

- **Administrator.** Behind the successful running of WebApp are experts who carries out maintenance tasks and other essential tasks. A general term that would refer to these specialists is a “WebMaster”. This role has responsibility for the day-to-day operation of the WebApp, some of them are development and implementation of policies for the operation of the WebApp, establishment of proper support and feedback procedures, implementation of security and access rights, measurement and analysis of Web-site traffic, harmonization of change control procedures. Significant portion of a professional webmaster’s time is dedicated to maintaining the WebApp. Many a times webmaster may get involved in the technical activities along with web engineer and support specialist.

1.6.2 Building the Team

Following are the Guidelines for building successful web engineering teams;

- **A set of team guidelines should be established.**These are the norms that team should follow to ensure productivity and success. They can be simple directives (for example, Team members are to be on time for meetings or what is expected of each person) or general guidelines (for example, Every team member has the right to offer ideas and suggestions or how problems are to be dealt with), but it is necessary to make sure that the team creates and follows these ground rules by consensus and commits to them, both as a group and as individuals.
- **Strong leadership is a must.** Team leader is the person who has ability to lead and influence others as well as skill to relate and interact with peers, subordinates, and superiors. The team leader must lead by example and by contact. A leader must show a passion that influence team to commit oneself to the work that challenges them. Effective leaders can get stuff done in a timely manner and makes sure their team members are getting the work-life balance they need to stay energized.

- **Respect for individual talents is critical.** Teams are made up of diversified talents and not everyone is good at everything. The best teams make is the one which makes use of individual's strengths and help each other to grow and realize project goals. The best team promotes and improves creativity and talent.
- **Effective Communication is essential.** Efficient teams must hold open lines of communication. Communication must be truthful and flow amid all team members equally. Team members who accepts each other's unique communication styles, or who decide on a single style of communication from the starting, are more likely to move the team in a positive and productive direction that every team member appreciates and supports. Team members must never be hesitant to communicate with any other members about difficulties, issues and concerns, as well as novel ideas or personal observations.
- **Every member of the team should be committed.**
- **It's easy to get started, but it's very hard to sustain momentum.** The best teams never let an "insurmountable" problem stop them. Team members develop an optimum solution and proceed, hoping that the momentum of forward progress may lead to an even better solution in the long term. In practical situation Team strategies, goals, workflows, tasks, technologies and even members can change over the life of the team. Team members should be able to rally together and face new challenges head-on.

1.7 LET US SUM UP

The evolution of Web-based applications and systems is perhaps the most significant event in the history of computing. With time more and more mature and disciplined approaches are evolved. Some are completely new and some are adapted from software engineering principles, concepts, process, and methods.

Web based systems are essentially different from other categories of computer software. They are content driven, network intensive and continuously evolving. The immediacy that drives their development, the superseding need for security in operation, and high demand for aesthetic as well as functional content delivery are

additional characteristics. There are various criteria based on which one can evaluate web based systems some of them are quite similar to conventional software systems like usability, reliability, functionality, efficiency, maintainability, security, availability, scalability, and time to market.

There are three layers of WebE, Process, methods and tools or technology. The WebE process adopts the agile development philosophy that accentuates a "lean" engineering approach that leads to the incremental delivery of the system to be built. The generic process framework includes communication, planning, modelling, construction, and deployment is applicable to WebE processes. Further these activities are sub-divided into a set of WebE tasks that are adapted to the needs of each project. Also a set of umbrella activities similar to those applied during software engineering work - SQA, SCM, project management can be applied to all WebE projects.

1.8 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Question 1: Which web app attribute is defined by the statement: “A large number of users may access the WebApp at one time”?

a) Unpredictable load
b) Performance
c) Concurrency
d) Network intensiveness

Answer: c)

Question 2: Which web app attribute is defined by the statement: “The quality and aesthetic nature of content remains an important determinant of the quality of a WebApp”?

a) Availability
b) Data driven
c) Continuous evolution
d) Content sensitive

Answer: d)

Question 3: Which process model should be used in virtually all situations of web engineering?

a) Incremental Model
b) Waterfall Model
c) Spiral Model
d) None of the mentioned

Answer: a)

Explanation: The web engineering process must accommodate incremental delivery, frequent changes and short timeline.

Question 4: Web development and software development are one and the same thing. True or False?

a) True
b) False

Answer: b)

Explanation: They are different due to the nature and distinct requirements of Web-based systems.

Question 5: Which of the following statements are incorrect with reference to web-based systems?

a) Web-based systems should be un-scalable.
b) Web-based systems must be able to cope with uncertain, random heavy demands on services.
c) Web-based systems must be secure.
d) Web-based systems are subject to assorted legal, social, and ethical scrutiny.

Answer: a)

Explanation: Web-based systems should be scalable.

Question 6: What category of web-based system would you assign to electronic shopping?

a) Informational
b) Interactive
c) Transaction-oriented
d) Workflow-oriented

Answer: c)

Explanation: It involves usage of transaction management of database systems.

1.9 FURTHER READING

Books

- Software Engineering by Roger S. Pressman 6th Edition McGraw Hill Publications.

Web sites

- https://en.wikipedia.org/wiki/Web_engineering
- https://www.tutorialspoint.com/web_developers_guide/web_basic_concepts.htm
- https://www.tutorialspoint.com/web_development_tutorials.htm

1.10 ASSIGNMENTS

Question 1: What is a Web Application? List the attributes of WebApps.

Question 2: What are the categories of WebApp?

Question 3: What are the WebE methods within the process Framework?

1.11 ACTIVITIES

Define Web Engineering.

What is a WebApp Increment?

1.12 CASE STUDIES

Describe the role of the "Web Publisher" in your own words.

Write any four best practices of Web Engineering with simple example.

1.13 REFERENCES

- [Ref-1] Software Engineering By Roger S. Pressman Sixth Edition McGraw Hill Publications
- [Ref-2] Powell, T. A., Web Site Engineering, Prentice-Hall, 1998.
- [Ref-3] Dart, S., "Containing the Web Crisis Using Configuration Management," Proc. First ICSE Workshop on Web Engineering, ACM, Los Angeles, May 1999. (The Proceedings of the First ICSE Workshop on Web Engineering are published on-line at http://fistserv.macarthuf.uws.e'du.au/san/icse99-WebE/ICSE99-WebE-Proc/ default.htm)
- [Ref 4] McDonald, A., and R. Welland, Agile Web Engineering (AWE) Process, Department of Computer Science, University of Glascow, Technical Report TR-2001 -98, 2001 , downloadable from http://www.dcs.gla.ac.uk/~andrew/TR-200 1 -98.pdf.

Unit 2: Analysis for Web Application

2

Unit Structure

2.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Analyse and evaluate software architectures of a real-world WebApp.
- Learn conceptual knowledge of web application.
- Perform analysis modelling for web applications.
- Identify candidate tools and technologies for developing web application.

2.2 INTRODUCTION

The development process of web applications is quite different from the process of developing traditional software. In traditional software development, we are using different *software process models* for software development. These models cannot be used for the development of web application because the content of web application frequently changes. The traditional software engineering process needs some changes for the development of web applications.

The web applications are developed to deliver functionalities for the large group of end users and the modifications are done frequently to adapt new changes. It is not possible to collect all the requirements at the beginning itself. The web applications always keep its information up-to-date and hence making the changes is a continuous process.

WebApps have ashort development cycle and a volatility so Web-Engineer has less time to do detailed analysis and designing.

Comparison between Software Engineering and Web Engineering.

Software Engineering	Web Engineering
Traditional Software System has small user range.	WebApps has large user range.
User categories are relatively limited.	Larger user categories.
User Requirements are specified	User Requirements are changes frequently with time.
Growth and change is small	Fast growth and changing rapidly.
Development budgets varies in a wide range according to the size of the company	Development budgets are small.
Development time is longer	Development time is small.

A Web engineering team should include analysis modelling when one or more of the following conditions are met:

- The WebApp to be built is large and/or complex.
- The number of stakeholders is large.
- The number of Web engineers and other contributors is large.
- The goals and objectives for the WebApp will affect the business.

If these conditions are not present, it is possible to decrease the importance of analysis modelling. In such conditions, limited analysis modelling may occur. Web Engineering team can do design modelling based on information obtained during requirements gathering.

2.3 REQUIREMENTS ANALYSIS FOR WEBAPPS

Requirements analysis for WebApps includes three major tasks:

1) ***Formulation***: During formulation, the basic goalsand objectives are identified and the categories of users are defined for the WebApp.

2) ***Requirements gathering:*** During requirements gathering, functional requirements and content are listed and Use-case scenarios developed.
3) ***Analysis modelling:*** The purpose of analysis modelling is, to establish a basic understanding of *why* the WebApp is to be built, *what* problems it will solve for end-users and *who* will use it.

2.3.1 The User Hierarchy

As part of the formulation and requirements gathering tasks, the categories of end-users (communicate with the WebApp) are identified.

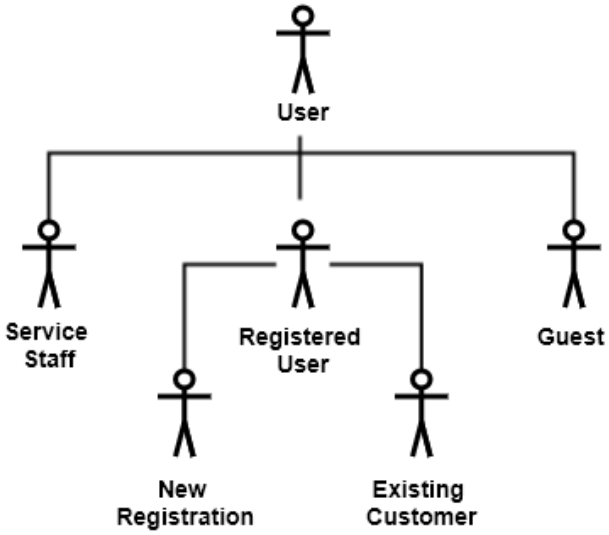


Figure 1 – User hierarchy

Figure 1 shows sample user hierarchy. General users for web site are:

*Guest*user: Visits the site but doesn't register. Guest users are often search common information / comparison of content / interested in free content.

First-time Customer: User who are doing first time registration and consume WebApp services which required registration.

Existing Customer: A registered user who own the services and can contact customer care or take technical support from WebApp.

Customer service staff: They are special users, they assist customers who have contacted for support.

Use-Cases Model development and Refining

We have already discussed about Use-case in Block#1_Unit1#Section 1.5.1.

Use-case diagrams are created for each user category explained in the user hierarchy.

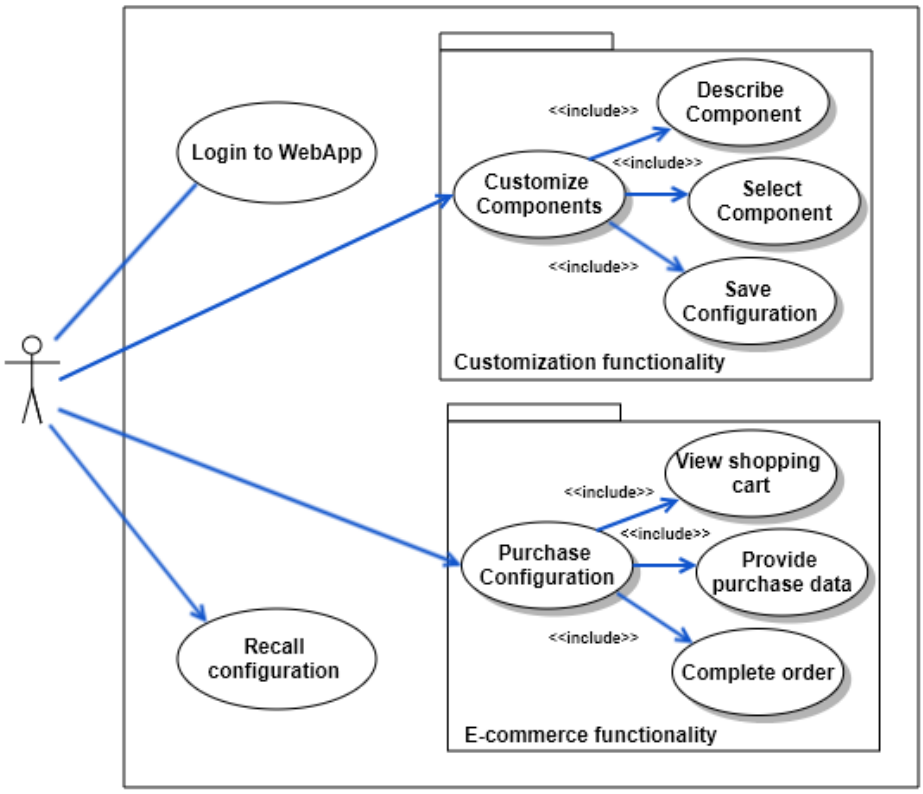


Figure 2 – Use case diagram for first-time customer

Figure 2 shows a UML use case diagram for first time customer. Each oval represent a use case that explain a precise communication between first-time customer and the WebApp. First communication is defined by the log-in (to web-site) use case.

In figure 2, key WebApp functionality and all relevant use cases arecombined into boxes. These boxes are called a "packages" inUML. Figure 2 has two packages: customization and e-commerce.

Use-cases are structured into functional packages.

Each package should be: [Ref-2]

- *Comprehensible*: All stake-holders should know the motive of the package.
- *Cohesive*: The package combined functions which are related to one another.
- *Loosely coupled*: Functions or classes inside the package collaborate with each other. But collaboration outside the package needs to be minimum.
- *Hierarchically shallow*: Deep functional hierarchies should be avoided. It will make navigation difficult for end user.

Requirements analysis and modelling is iterative activity:

- Newly defined use cases will be included into packages.
- Existing use cases will be modified based on requirement.
- Some of the use cases might be moved to other packages.

2.4 THE ANALYSIS MODEL FOR WEBAPPS

Use cases is the key input for WebApp analysis model.

Use case descriptions are analysed to identify:

- Potential classes along with its operations and attributes.
- Content to be accessible by the WebApp.
- Function to be executed.
- Implementation related requirements; for building environment and infrastructure for supporting the WebApp.

For creating complete analysis model, it is required to understand four analysis activities:

- *Content analysis:* Identifies required content (text, graphics, images, video and audio data) to be produced by the WebApp.
- *Interaction analysis:*Defines the way the user interacts with WebApp.
- *Functional analysis:*Defines the operations of WebApp.
- *Configuration analysis:*Defines the environment and infrastructure where WebApp executes.

The data collected in above four analysis tasks should be reviewed properly. After required modification, it should be organized into a model and can be handed over to WebApp designer.

The model include structural and dynamic elements.

- Structural elements: The analysis classes and content objects.
- The dynamic elements: Describe the way structural elements interact with each other.

2.5 THE CONTENT MODEL

The content model consists of structural elements. These structural elements includes content objects (like text, images, videos, audio etc.) and analysis classes. An analysis class includes attributes, operations (behaviour) and collaborations (communication with other class).

The content model (content objects and analysis classes) is obtained using use cases.

2.5.1 Defining / Identify Content Objects

Web application present information to an end user which is known as *Content*. Below are high level example of Content Objects.

External entities (e.g., other systems, databases, people) that produce or consume information to be used by the WebApp.

Things (e.g., reports, displays, video images) that are part of the information domain for the problem.

Occurrences or events (e.g., a quote or an order) that occur within the context of a user's interaction with a WebApp.

Roles (e.g., retail purchasers, customer support, salesperson) played by people who interact with the WebApp.

Organizational units (e.g., division, group, and team) that are relevant to an application.

Places (e.g., manufacturing floor or loading dock) that establish the context of the problem and the overall function of the WebApp.

Structures (e.g., sensors, monitoring devices) that define a class of objects or related classes of objects.

2.5.2 Content Relationship and Hierarchy

Majority time, a list of content objects (with brief description) is enough to define the requirements for content. Sometime, we require entity relationship diagram or data tree [Ref-3] to show the relationships between content objects.

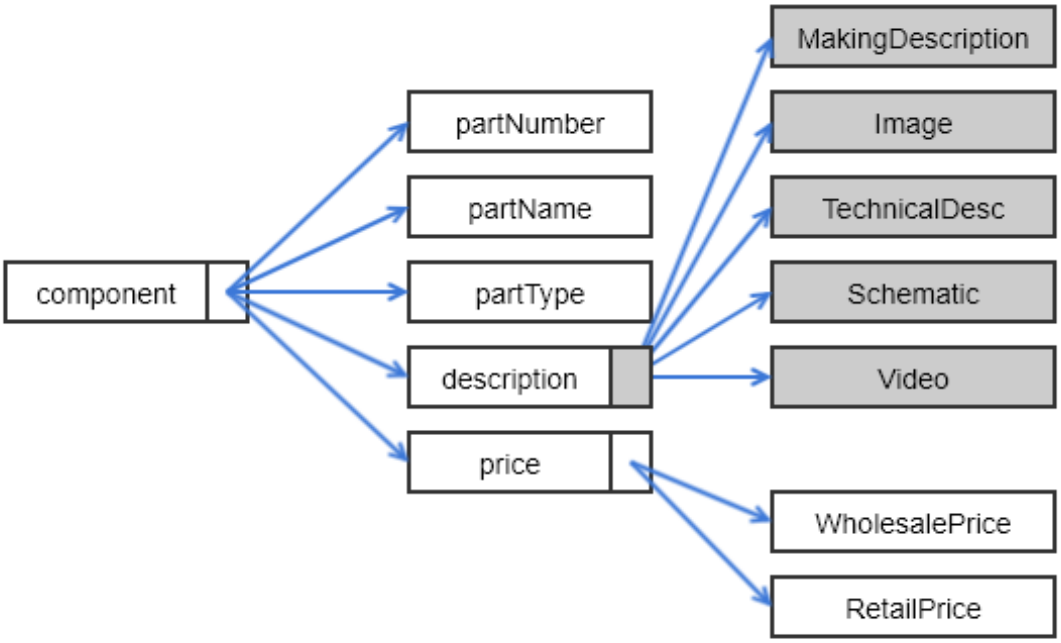


Figure 3 – Data tree

Figure 3 shows sample data tree diagram. As per above data, “description” id defined by five content objects (darker rectangle).

2.5.3 Analysis classes for WebApps

As we know, analysis classes are derived from use case. Let’s take an example:

Select following use case scenario and underline all potential objects.

Use Case Scenario: Customer verifies items in shopping cart. Customer provides payment and address to process sale. System validates payment and responds by confirming order and provides order number that Customer can use to check on order status. System will send Customer a copy of order details by email.

Figure 4 - Sample use case scenario

After underlining on candidate objects, we are getting below candidate classes.

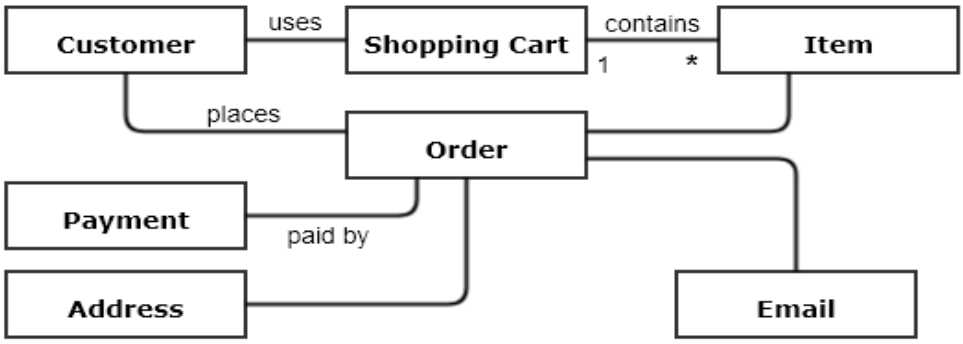


Figure 5 – Analysis classes for use case

This process repeat for all use cases to get analysis classes and content object.

2.6 THE INTERACTION MODEL

WebApp enable end user to interact with application content and functionality. This interaction model is collection of four elements:

1. Use cases,
2. Sequence diagrams,
3. State diagrams
4. A user interface prototype

The interaction is also represented inthe navigation model (Section 1.9).

Use-cases: Use-cases arekey element of the interaction model for WebApps. Use cases define the major interaction between end users (actors) and the system.

Sequence diagrams: Provide representation of user actions and collaborate with analysis classes. User actions is the dynamic elements of a system defined by use

cases. So sequence diagrams define a link among the actions mentioned in the use case and the analysis classes (structural entities).

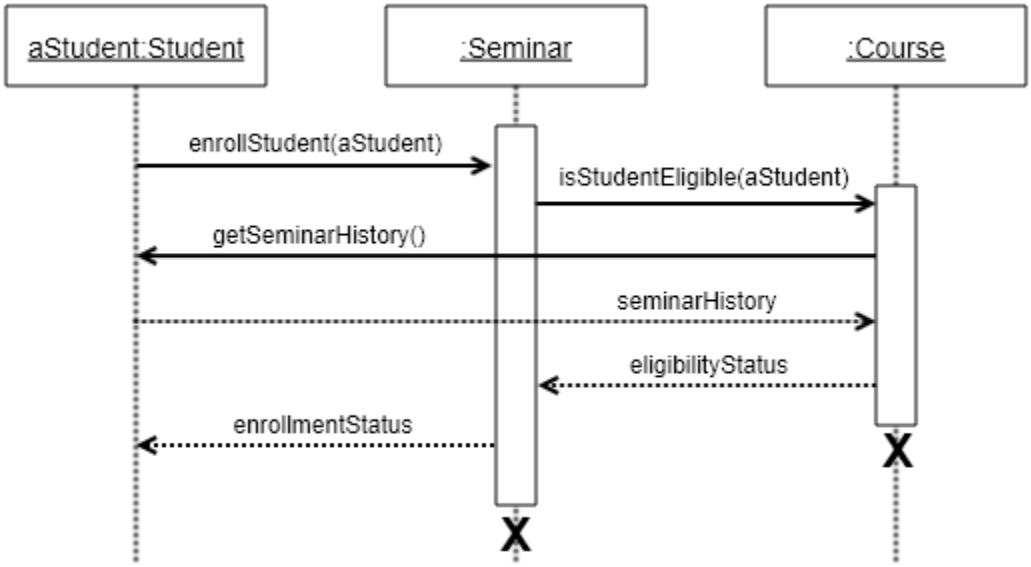


Figure 6 – Sequence diagram for use case: Seminar enrolment

Figure 6 shows a sequence diagram for the seminar enrolment use case. The vertical line of the diagram represents actions that are defined within the use-case.The horizontal line identifies the analysis classes. Objects are labelled in the standard UML format `name:ClassName`, where “name” is optional (objects without name are called *anonymous objects*). For example, in Figure 6, we can see the Student object has the name `aStudent`, this is called a *named object*, whereas the instance of Seminar is an anonymous object. The instance of Student was given a name because it is used in several places as a parameter in messages, whereas the instance of the Seminar did not need to be referenced anywhere else in the diagram and thus could be anonymous. The X at the bottom of an activation box, an example of which is presented in Figure 6, is a UML convention to indicate an object has been removed from memory. In languages such as C++ where we need to manage memory and need to invoke an object's destructor, typically modelled a message with the stereotype of <<destroy>>. In languages such as Java or C# where memory is managed for us and objects that are no longer needed are automatically removed from memory (referred to as garbage collection).

Sequence diagrams can be created for each use-case once analysis classes are defined for the use-case.

State diagrams

The *State diagram* define dynamic behaviour of the WebApp. The state diagram can be represented at various level of abstraction. Figure 7 shows high level state diagram for the Bank Automated Teller Machine (ATM).

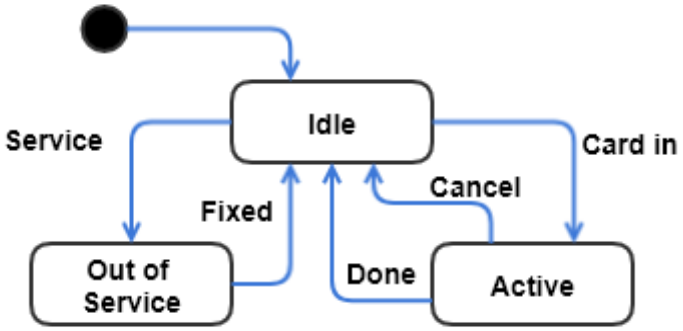


Figure 7 - High level state diagram for bank ATM

ATM is initially turned off. After the power is turned on, ATM performs start up action and enters into **Idle** state.In this state ATM waits for customer for interaction.The ATM state changes from **Idle** to **Active** when the customer inserts banking or credit card in the ATM's card reader. The transition from **Active** state back to the **Idle** state could be triggered once customer is done with the transaction. In case of service / technical issue, the ATM state changes from **Idle** to **Out of Service**. After fixing the issue, state changes back to **Idle**.

All three diagrams (use case, sequence and state) represent related information, then why we need these diagrams? In some scenarios, all three diagrams are not essential. Only use cases are enough. But use cases define one dimensional view of the interaction. Sequence diagrams provide another dimension that is more dynamic in nature. State diagrams gives third dimension which is more behavioural and give information about navigation. So it is always beneficial for large / complex WebApps to define interaction model that includes all three diagrams.

User interface prototype

Below points are very critical for user satisfaction and acceptance of the WebApp:

- The layout of the user interface.
- The content it presents.
- The interaction mechanisms it implements.
- Overall aesthetic of the user WebApp connections.

Prototype is creating user interfaces without implementation or coding. So end user can get a feel of actual WebApp before development start. The creation of user interface prototype is a design activity but it is always a great idea to create prototype during the creation of the analysis model. Using prototype, end user can do review of WebApp during analysis and define precise requirement based on review of user interface.

2.7 THE FUNCTIONAL MODEL

The Functional model defines two processing elements of the WebApp, each representing a different level of abstraction:

1. *User visible functionality that is provided by the WebApp to end-users.:*
   User visible functionality include any processing functions that are started directly by the end-user. For example, a financial Web site might implement a different typeof financial functions (e.g., Interest calculation of saving bank account or a retirement savings calculator). These functions may internally implemented using operations within analysis classes. But end user point of view, the function is visible to end-user.
2. *The operations enclosed within analysis classes that implement behaviours associated with the class.*
   At a lower level of abstraction, the analysis model defines the functional model of analysis classes. These operations manipulate class attributes.

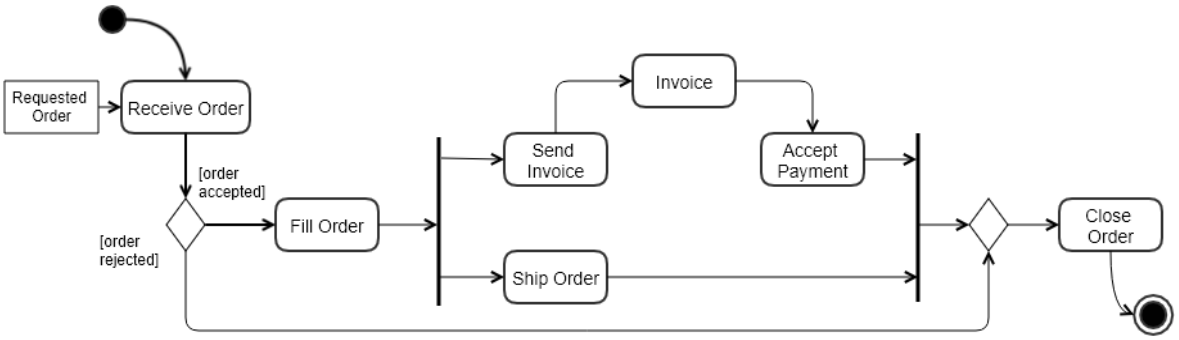


Figure 8 Activity diagram for process shopping order

The UML activity diagram defines processing details. Figure 8 shows an activity diagram for the*processing shopping order*. *Requested order* is input parameter of the activity. Once order is accepted and all essential details is filled in, payment is accepted and order is shipped.

Note, that this business flow allows order shipment before invoice is sent or payment is confirmed.

The activity diagram is similar to the flowchart. It is showing the processing flow and logical decisions with the flow.

2.8 THECONFIGURATION MODEL

The design and implementation of WebApps must accommodate different environments (both server and client). The server hosts the WebApp, and allow multiple users to access the WebApp through a network. Using the Client (like browser), end user can access WebApp on the desktop. The WebApp can be deployed on a server and provides access over the internet, an Intranet, or an Extranet. Server hardware and operating system must be specified for the WebApps. If the WebApp need database access or interface with business applications at server side, appropriate interfaces, communication protocols and related collaborative information must be specified.

Client side software (like browser) offers the infrastructure which allows access of the WebApp from the client's location. Normally, browser is used to access the

WebApp content and functionality from the server. So, the WebApp must be properly tested within every browser which is mentioned into configuration model.

Sometime, the configuration model is just a list of Server side and Client side attributes. But, for complex WebApps, different type of configuration required (like load balancing with multiple servers, caching mechanism, multiple databases, distributed application components etc.).

2.9 RELATIONSHIP-NAVIGATION ANALYSIS

We are using Analysis model to identify content and functional elements. These elements used to implement user interaction. As analysis progress into design, these elements become components of the WebApp architecture. In the context of Web applications, each architectural element can be linked to all other architectural elements. However as the number of links grows, navigation difficulty also rises. If so, how to create the proper links within content objects and the functions?

Relationship-navigation analysis (RNA) define analysis steps to identify not-covered relationships between the elements as part of the analysis model.

Yoo and Bieber [Ref-4] describe RNA in the following manner:

RNA provides systems analysts with a systematic technique for determining the relationship structure of an application, helping them to discover all potentially useful relationships in application domains. These later may be implemented as links. RNA also helps determine appropriate navigational structures on top of these links. RNA enhances system developers' understanding of application domains by broadening and deepening their conceptual model of the domain. Developers can then enhance their implementation by including additional links, meta-information, and navigation.

The RNA approach is organized into five steps:

- *Stakeholder analysis*:Identifies the different user categories and createssuitable stakeholder hierarchy.
- *Element analysis*: Identifies the content objects and functional elements that are of interest to end-users.

- *Relationship analysis*: Describes the relationships that exist among the WebApp elements.
- *Navigation analysis*: Examines how users might access individual elements or groups of elements.
- *Evaluation analysis*: Considers practical issues (e g., cost/benefit) associated with implementing the relationships defined earlier.

We have already discussed first two steps in this unit. Let's discuss remaining three steps.

2.9.1 Relationship Analysis — Key Questions

Yoo and Bieder [Ref-4]have suggested a list of questions that a Web engineer or systems analyst should ask about each element (content object or function) that has been identified within the analysis model.

- Is the element a member of a broader category of elements?
- What attributes or parameters have been identified for the element?
- Does descriptive information about the element already exist? If so, where is it?
- Does the element appear in different locations within the WebApp? If so, where?
- Is the element composed of other smaller elements? If so, what are they?
- Is the element a member of a larger collection of elements? If so, what is it and what is its structure?
- Is the element described by an analysis class?
- Are other elements similar to the element being considered? If so, is it possible that they could be combined into one element?
- Is the element used in a specific ordering of other elements? Does its appearance depend on other elements?
- Does another element always follow the appearance of the element being considered?
- What pre- and post -conditions must be met for the element to be used?
- Do specific user categories use the element? Do different user categories use the element differently? If so, how?

- Can the element be associated with a specific formulation goal or objective? With a specific WebApp requirement?
- Does this element always appear at the same time as other elements appear? If so, what are the other elements?
- Does this element always appear in the same place (e.g., same location of the screen or page) as other elements? If so, what are the other elements?

The answers of above questions help the Web engineer to arrange the elements within the WebApp and to create relationships within elements.

2.9.2 Navigation Analysis

After relationships have been developed within elements defined, the Web engineer must analyse that how each user category will navigate from one element to another element. As part of design, developer should consider overall navigation requirements.

The following questions should be asked as part of Navigation analysis.

- Should certain elements be easier to reach (require fewer navigation steps) than others? What is the priority for presentation?
- Should certain elements be emphasized to force users to navigate in their direction?
- How should navigation errors be handled?
- Should navigation to related groups of elements be given priority over navigation to a specific element?
- Should navigation be accomplished via links, via search-based access, or by some other means?
- Should certain elements be presented to users based on the context of previous navigation actions?
- Should a navigation log be maintained for users?
- Should a full navigation map or menu (as opposed to a single "back" link or directed pointer) be available at every point in a user’s interaction?
- Should navigation design be driven by the most commonly expected user behaviours or by the perceived importance of the defined WebApp elements?

- Can a user "store" his previous navigation through the WebApp to expedite future usage?
- For which user category should optimal navigation be designed?
- How should links external to the WebApp be handled? Overlaying the existing browser window? As a new browser window? As a separate frame?

The Web engineering team and stakeholders should also define overall requirements for navigation. Navigation helps user to understand entire WebApp structure (example “Site Map”). Navigation analysis help Web engineer to create “guided tour” of WebApp which will highlight the most important elements.

2.10 LET US SUM UP

Requirements analysis for WebApps includes three major tasks: Formulation, requirements gathering, and analysis modelling. The objective of these activities are: (1) define the basic motivation (goals) and purposes for the WebApp. (2) Define the categories of users; (3) define the content and functional requirements for the WebApp; and (4) establish a basic understanding of why the WebApp is to be built, who will use it, and what problem(s) it will solve for its users.

Use cases are the most critical for all requirements analysis and modelling activities. Use cases can be structured into functional packages, and each package is evaluated to make sure that it is comprehensible, cohesive, loosely coupled and hierarchically shallow.

For creating complete analysis model, required to understand four analysis activities: 1) content analysis identifies the full range of content to be delivered by the WebApp; 2) interaction analysis describes the way in which the user interacts with the WebApp; 3) functional analysis defines the operations that will be used by WebApp content and describes other processing functions that are independent of content but essential to the end-user, and 4) configuration analysis describes the environment and infrastructure in which the WebApp resides.

The content model define the range of content objects that are to be formed into a WebApp. These content objects must be developed or acquired for integration into the WebApp architecture. A data tree can be used to represent a content object

hierarchy. Analysis classes are derived from use case. Analysis classes provide another way to representing key objects that the WebApp will manipulate.

The interaction model is created from use-cases, UML sequence diagrams, and UML state diagrams to describe the “conversation'' between the user and the WebApp. In addition, an interface prototype may be built to assist in developing layout and navigation requirements.

The functional model describes user observable functions and class operations using the UML activity diagram. The configuration model describes the environment that the WebApp will require on both the server side and the client side of the system.

Relationship navigation analysis recognises relationships between the content and functional elements defined in the analysis model. Relationship navigation analysis forms requirements for defining appropriate navigation links throughout the system. A series of questions help to establish relationships and identify characteristics that will have an influence on navigation design.

2.11 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Question 1: Write a simple one-line definition of each of the following key terms. Try to do this without referring back to the notes if possible.

- Guest user
- Customer service staff user
- Content object

Question 2: Note down key differences of Software Engineering and Web Engineering.

Question 3: What is prototype? Explain benefits of prototype.

2.12 FURTHER READING

Books

- Software Engineering by Roger S. Pressman Sixth Edition McGraw Hill Publications.

Web sites

- https://www.uml.org/
- Wikipedia : https://en.wikipedia.org/wiki/Web_engineering
- Web basic concepts: https://www.tutorialspoint.com/web_developers_guide/web_basic_concepts.htm
- UML standard diagrams: https://www.tutorialspoint.com/uml/uml_standard_diagrams.htm

2.13 ASSIGNMENT

- Explain three major tasks of Requirements analysis.
- Describe four analysis activities for creating analysis model.

2.14 ACTIVITIES

- In which scenarios, Web engineering team should include analysis modelling? Explain with a simple example.

2.15 CASE STUDIES

- Take any simple user case scenario and create Analysis classes (refer section 1.5.3).

2.16 REFERENCES

- Ref-1: Software Engineering by Roger S. Pressman 6thEdition McGraw Hill Publications.
- Ref-2: Conallen , J., Building Web Applications with UML, Addison-Wesley, 2000.
- Ref-3: Sridhar, M., and N. Mandyam, "Effective Use of Data Models in Building Web Applications," 2001, available at http://www2002.org/CDROM/alternate/698/.
- Ref-4: Yoo and M. Bieber. "Toward a Relationship Navigation Analysis," Proc. 33rd Hawaii Conf. On System Sciences, vol. 6., IEEE, January 2000, download from www.cs.njit.edu/~bieber/pub/hicssO0/INWEBO2.pdf.

Unit 3: Design for Web Application

3

Unit Structure

3.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Learn about web design standards and its importance.
- Understand design issues for Web Engineering.
- Measure quality of Web application.
- Understand design goal for WebApps.

3.2 INTRODUCTION

As stated by Jakob Nielsen [Ref-2]: "There are essentially two basic approaches to design: the artistic ideal of expressing yourself and the engineering ideal of solving a problem for a customer."

During the first decade of Web development, developers used to choose artistic design. This kind of designs were usually carried out in form of HTML pages and occurred in ad hoc manner. Design evolved out of an artistic vision that itself evolved as WebApp construction occurred.

In current times, supporters of agile software development consider WebApp as poster children for “limited design”. They argue that development immediacy and volatility play important role in WebApp as compared to formal design and design evolves as an application is built (coded), here the design is less likely to get separately planned and penned down but it evolves in continuation as application is being build.So, for WebApp it is fine to spare little time creating design model. For relatively simple WebApp this argument is worth. However, there are situations where design should not be taken lightly, few of them are listed below:

- When content and function are complex.
- When the WebApp incorporates large number of content objects, functions, and classes.
- When the success of the WebApp will have a straight effect on the success of the business.

So in simple words, we can state that Design matters, because it can reflect how clients and potential customers see and feel about WebApp, Design gives an overall

impression about web application; make a bad impression and you lose a potential conversion; Make a good one and you gain a customer; Make a great impression, and you can potentially gain and retain a customer for life.

This reality leads us to Nielsen's second approach— "the engineering ideal of solving a problem for a customer." Web engineering adopts this philosophy, and a more rigorous method to WebApp design enables developers to attain it.

In simple words,Design for WebApp is combination of technical and non-technical activities. The overall look and feel of content are developed as part of graphic design; the aesthetic layout of the UI (user Interface) is created as part of interface design; and the complete technical structure of the WebApp is exhibited as part of architectural and navigational design. Like in almost every design first a design, model is created and then actual creation, but in fact a good Web engineer knows that the design matures as more is learned about stakeholder requirements as the WebApp is constructed.

3.3 Design Issues for Web Engineering

There are several generic and specific issues one must consider while applying design within web engineering context. As we know that effective design results in better model and better WebApp so design is a guide to WebApp construction.At the basic level any design modelshould exhibit sufficient information to reflect how requirements from customer or stakeholder are rendered into content and executable code. At the same time design should also be specific and must incorporate key features of WebApp such that web engineer can effectively build and test such features.

3.3.1 Design and WebApp Quality

As a basic concept we know that better design leads to a high-quality product. This leads us to a recurring question that is encountered in all engineering disciplines: What is quality? In this section, we will examine the answer within the Web engineering context.

The quality is the parameter that has two aspects in this context. Quality from user’s perspective and quality from technical perspective. As we know users are the actual

evaluators of the application and every user is different from other, users may have different choices about look and feel, about simple and complex content, about simple or flashy graphics and many other aspects. Every user has his or her own criteria about what actually makes a “good” WebApp and that is what should matter most, as users are likely to accept web application which is “good” in their criteria. So, perception of user might be more important than technical quality aspect of WebApp.

Now the obvious question comes that how this quality can be stated? What are the attributes that makes WebApp good for end-users and at the time shows technical characteristics of quality so that Web engineer can correct, adapt, enhance, and support the application for long time period?

Following are some software quality characteristics which are also relevant for measuring quality of Web application.

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability

As displayed in Figure 1, Olsina and colleagues [Ref-3] have proposed a "quality requirement tree" that recognizes a set of technical attributes— usability, functionality, reliability, efficiency, and maintainability that contributes in high-quality WebApps. The criteria in the Figure 1 provides base for Web engineers to design, build, and maintain WebApps.

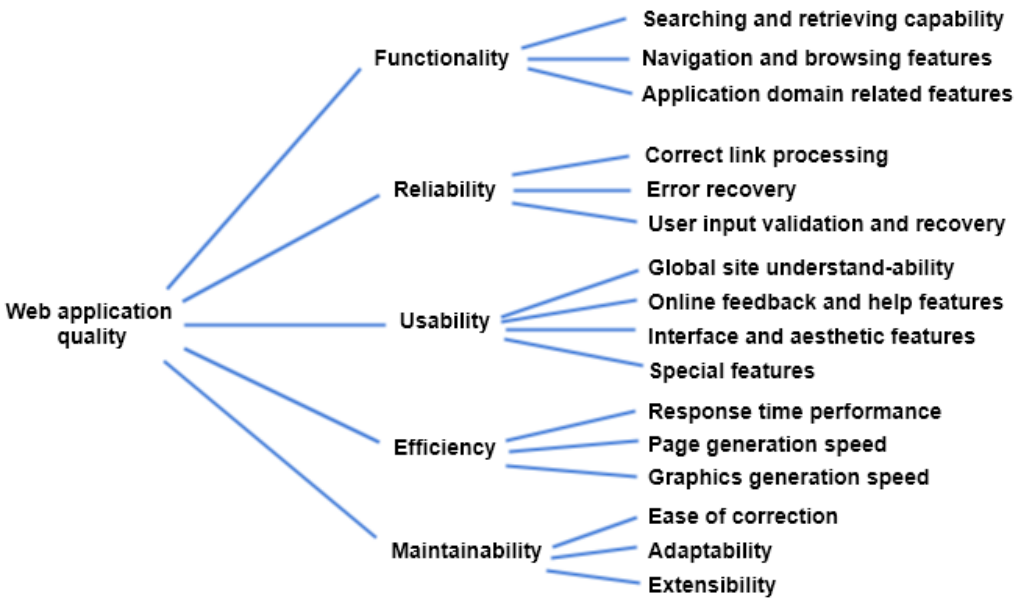


Figure 1 – Quality requirement tree [Ref-3]

Offutt [Ref-4] further extended quality attributes noted in Figure 1and added few more attributes:

Security. As we saw in earlier discussion that in last few years WebApps grown from simple to complex one. Many of such WebApps have become heavily integrated with critical corporate and government databases. Many WebApps functions involves financial transactions. E-commerce applications extract and then store sensitive customer information. For all of these and many other reasons, WebApp security is extremely critical in many circumstances. The important measure of security is the ability of the WebApp and its server environment to prohibit unauthorized access and to prevent malicious attacks or activities.As security is very deep and broad subject, detail discussion is not included here.

Availability. User always expect WebApp to be available always. So Even the best WebApp will not meet users' needs if it is unavailable. From a technical perspective availability is measure of the percentage of time that a WebApp is available for use. The typical end-user expects WebApps to be available 24 X 7 X 365. But "up-time" is not the only indicator of availability. For example, you made application which is up all the time but application is compatible to only ‘X’ browser/platform but what if user access your application using browser/platform ‘Y’, in this case your application is

not available for user who is using browser/platform ‘Y’. This all should be considered while considering Availability.

Scalability. Scalability is one important quality measure for WebApp,It is the variation in volume handled significantly by the web applications. As there are no fix rules of how many requests comes to WebApp at specific time, there can be 10 requests, 100 and even 1000 requests come at any given time and WebApp should be capable of handling all these requests with same efficiency and accuracy. So WebApp and its server environment should be scalable to handle increased load.

Time-to-market. Basically, time to market (TTM) is the length of time taken by WebApp from being conceived until its being available for users. TTM is important in Web Applications as they are outmoded quickly. A common assumption is that TTM matters most for first-of-a-kind WebApp. From technical perspective this attribute is not a potential measure of quality but from business perspective TTM is a measure of quality. The first WebApp of its kind in the market often captures a disproportionate number of end-users.

As we know that billions of Web pages are available on the World Wide Web. Even well-framed and well-targetedweb searches result in a huge number of contents. With so many sources of information, how does the one measure the qualityof the content that is presented within a WebApp? Tillman [Ref-5] suggests a set of criteria for measuring the quality of content:

Scope: Can the scope and depth of content be easily determined to ensure that it meets the user's needs?

Authority: Can the background and authority of the content's authors be easily identified?

Currency: Currency of the WebApp refers to: how current the information presented is?And how oftenthe site is updated or maintained?

Stability: Is the content and its location stable (i.e., will it remain at the referenced URL)?

Creditability: Is content credible?

Uniqueness: Is content unique? That is, does the WebApp provide some unique benefit to those who use it?

Relevance: Is content valuable to the targeted user community?

Purpose: Is content well-organized? Indexed? Easily accessible?

The above checklist is just an example quality related questions but in reality, web engineer should consider all the aspects of quality and develop a WebApp that positively address all these quality related questions.

3.3.2 Design Goals

Jean Kaiser[Ref-6] suggests a set of design goals that are applicable to almost every WebApp irrespective of application domain, size, or complexity:

Simplicity.The best user experience is served through simplicity. The WebApp should be moderate and not too much in any aspect. Content should be informative but to the point and should use a text, graphics, video, audio etc. appropriate to the information that is being delivered.WebApp objectives should be realized in the simplest possible way.Navigation should be straightforward and navigation mechanisms should be naturally understandable to the end user.Functions should be easy to understand and use.

Consistency. This design goal applies to virtually every element of the design model and this design goal may sound like a simple concept, but there are many examples out there that exhibit a lack of consistency in their designs and result in to customer dissatisfaction. Content should be constructed consistently, this can be achieved by using consistent layouts, using same text formatting and text styles, creating consistent visual elements, using standardized colour and style etc. Graphic design should ensure a consistent look across all parts of the WebApp.Architectural design should establish templates that lead to a consistent hypermedia structure. Interface design should outline consistent navigation, content display and mode of interaction.

Identity. Entire design of a WebApp must be in line with the application domain for which it is to be built. An e-commerce WebApp will definitely have different design as well as look and feel than the WebApp that provides financial services.Graphical design, navigational design, Interface design should be compatible and appropriate for Objective of WebApp. A Web engineer along with all other design contributorsshould work to establish an identity for the WebApp through the design.

Robustness. From any reliable WebApp user expect robust content and function that are relevant to user and able to fulfil user's need.Robustness is an important design goal and should be attained efficiently.

Navigability. This is like guiding the user on the webpage without even compelling them to do so. As we already discussed that navigation should be simple and yet interesting. At the same time intuitive for users to figure out how to get around a WebApp easily and efficiently. This goal emphasis on the design where user should be able to move about the WebApp with minimum possible instruction.

Visual appeal. Visual Appeal is the colours, shapes, pictures, fonts, white space, and overall visual balance of a WebApp. Web applications are indisputably the most visual, the most dynamic, and the most aesthetic amongst all software categories. As we all know beauty is contextual but there are certain parameters like style, graphics and aesthetics that increase visual appeal of the WebApp. Visual Appeal is important in making initial impression of a WebApp.

Compatibility. As we discussed earlier, WebApp will be used on a variety of browsers, platforms, operating systems, devices etc. and must be compatible with each of them. It's prerequisite for acceptance of WebApp by users.

3.4The WebE Design Pyramid

One of the most basic and complex question in Web Engineering is, what is design? The answer of this question is more difficult than one might believe. Asdesign is a base for creating a model for WebApp which is a combination of content, technology and aesthetics.Design activity is highly dependent on type and nature of WebApp. And as design varies the content, technology and aesthetics also varies. Creation of design require diverse set of skills from multiple technical/non-technical domains.

Figure 2 depicts a design pyramid for Web engineering. Each level of the pyramid represents one of the following design activities:

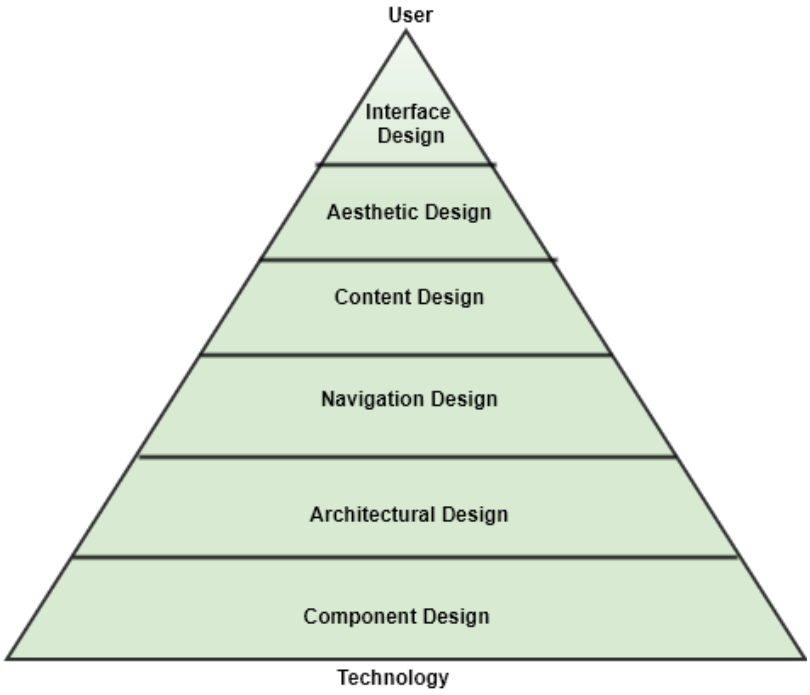


Figure 2 - The WebE Design Pyramid

- *Interface design* — Interface design outlines how user interface is going to be structured and organized. Interface design usually contains a depiction of screen layouts, a description of the modes of interaction, and a description of navigation mechanisms.
- *Aesthetic design*— Aesthetic design also sometimes termed as graphic design, it describes the overall "look and feel" of the WebApp. Includes colour schemes, geometric layout, texture, patterns, text size, font and placement, the use of graphics, and related aesthetic decisions. The design consistently makes WebApp aesthetically pleasing across the whole application and user journey.
- *Content design*— Content design defines layout, structure, and outline for all the content that is presented as part of the WebApp. This design also representsthe relationships between content objects.
- *Navigation* design— Navigation design encompasses navigational flow within different functions of WebApp and between various content objects. Complete Navigation of the WebApp is defined during Navigational Design.
- Architecture design—Architecture design identifies the overall hypermedia structure of the WebApp.

- Component design — Component design describes detailed processing logic essential to implement functional components.

We will discuss these all design in details in subsequent unit.

3.5 LET US SUM UP

In this Unit we understood WebE design from a very high level. We discussed about WebE Design Issues in details. The quality of a WebApp defined in terms of usability, functionality, reliability, efficiency, maintainability, security, scalability, and time-to-market is introduced during design. To achieve these quality attributes, a good WebApp design should exhibit simplicity, consistency, identity, robustness, navigability, visual appeal and compatibility.

Then we discussed WebE design Pyramid and various design from very high level. Interface design mainly describes the structure and organization of the user interface. Aesthetic design describes the overall "look and feel" of the WebApp. Content design defines layout, structure, and outline for all the content. Navigation design encompasses navigational flow within different functions and content of WebApp.Architecture design identifies the overall hypermedia structure of the WebApp,and Component design describes detailed processing logic.

Below table shows details list of quality characteristics of WebApp:

Functionality	Suitability	The capability of the software to provide an adequate set of functions for specified tasks and user objectives.
	Accuracy	The capability of the software to provide the right or agreed-upon results or effects.
	Interoperability	The capability of the software to interact with one or more specified systems.
	Security	The capability of the software to prevent unintended access and resist deliberate attacks intended to gain unauthorized access to confidential information or to make unauthorized modifications to information or to the program so as to provide the attacker with some advantage or so as to deny service to legitimate users.
Reliability	Maturity	The capability of the software to avoid failure as a

		result of faults in the software.
	Fault Tolerance	The capability of the software to maintain a specified level of performance in case of software faults or of infringement of its specified interface.
	Recoverability	The capability of the software to reestablish its level of performance and recover the data directly affected in the case of a failure.
Usability	Understandability	The capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use.
	Learnability	The capability of the software product to enable the user to learn its applications.
	Operability	The capability of the software product to enable the user to operate and control it.
	Attractiveness	The capability of the software product to be liked by the user.
Efficiency	Time Behavior	The capability of the software to provide appropriate response and processing times and throughput rates when performing its function under stated conditions.
	Resource Utilization	The capability of the software to use appropriate resources in an appropriate time when the software performs its function under stated condition.
Maintainability	Analyzability	The capability of the software product to be diagnosed for deficiencies or causes of failures in the software or for the parts to be modified to be identified.
	Changeability	The capability of the software product to enable a specified modification to be implemented.
	Stability	The capability of the software to minimize unexpected effects from modifications of the software.
	Testability	The capability of the software product to enable modified software to be validated.

3.6 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Question 1: Write a simple one-line definition of each of the following key terms. Try to do this without referring back to the notes if possible.

- Interface design
- Aesthetic design
- Fault Tolerance
- Scalability
- Time-to-market

Question 2: Differentiate analysis and design of WebApps.

3.7 FURTHER READING

Books

- Software Engineering by Roger S. Pressman Sixth Edition McGraw Hill Publications.

Web sites

- https://en.wikipedia.org/wiki/Web_engineering

3.8 ASSIGNMENTS

Explain Design Issues of Web Engineering in details.

3.9 ACTIVITIES

What is meaning of Agile? Describe advantages and disadvantages of Agile?

3.10 CASE STUDIES

Describe quality attributes of Web application in your own words.

3.11 REFERENCES

- Ref-1: Software Engineering by Roger S. Pressman 6thEdition McGraw Hill Publications.
- Ref-2: [NIEOO] Nielsen, J.. Designing Web Usability, New Riders Publishing, 2000.
- Ref-3: [OFF02] Offutt, J., "Quality Attributes of Web Software Applications," IEEE Sofhvare, March/April, 2002, pp. 25-32.
- Ref-4: [OLS99] Olsina, L. et al., "Specifying Quality Characteristics and Attributes for Web Sites," Proc. 1st ICSE Workshop on Web Engineering, ACM, Los Angeles, May 1 999,
- Ref-5: [TIL00] Tillman, H N, "Evaluating Quality On the Net," Babson College, May 30, 2000, available at http://www.hopetillman.eom/findqual.html#2.
- Ref-6: [KAI02] Kaiser, J., "Elements of Effective Web Design," About, Inc., 2002, available at http://webdesign.about.com/library/weekly/aa091998.htm.

Unit 4: Web Design 4

Unit Structure

4.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Understand the principles of creating an effective web design.
- Understand in-depth understanding of information architecture.
- Learn graphic design principles that relate to web design.
- Analyse the usability of a web site.

4.2 INTRODUCTION

WebApp design is an essential activity. It decides what and how the information presented to the users.It includes several different aspects, including webpage layout, content production and graphic design. While the terms web design and web development are often used interchangeably, web design is technically a subset of the broader category of web development.

Let’s learn Web Designing.

4.3 WEBAPP INTERFACE DESIGN

Every user interface should demonstrate the following characteristics, so end user feels rewarded and satisfied with WebApp:

- Easy to use
- Easy to learn
- Easy to navigate
- Intuitive
- Consistent
- Efficient
- Error-free
- Functional.

Interface design begins with a careful examination of the end user requirements. A user hierarchy is createdduring analysis modellingfor WebApp. Each user category may have separate needs. Each category may need to communicate with the

WebApp in different ways, and may require exclusive functionality or different content.

Web engineer must design an interface so that it answers three primary questions for the end-user [Ref-8]:

Where am I?

The interface should provide information related to current location in the content hierarchy. So end user can bookmark a Website page for revisiting later.

What can I do now?

The interface should always guide the user to recognise his current options - what functionalities are available, what links are live and what content is relevant.

Where have I been; where am I going?

The interface must simplify navigation. Hence, it must provide a "site-map" of where the user has been and what paths are available within the WebApp.

An effective WebApp interface must give answers of above three questions.

4.3.1 Interface Design Principles and Guidelines

Bruce Tognozzi [Ref-2] identifiesa set of fundamental characteristics, every WebApp interface designer should follow.

Effective interfaces are visually attractive and simple to use. Effective interfaces should not expose the internal working to the user. End-user's work continuously saved, give undo option for the user to reverse any activity any time. Effective application and services perform maximum work and requesting minimum of information from users.

Tognozzi identified below design principles [Ref-2].

Anticipation*:* A WebApp should be designed so that it predict the user's next move. For example, a user is searching for printer driver. The designer of the WebApp should anticipate that the user might request a download of the driver.

Communication*:* The interface should communicate the status of activity initiated by the user.

Consistency*:* The use of navigation controls, menus, icons and aesthetics (like colour, shape and layout) should be consistent throughout the WebApp. For example, if underlined blue text indicates a navigation link, content should never incorporate blue underlined text for any other purpose.

Controlled autonomy: The interface should facilitate user movement throughout the WebApp, but it should manage authentication and authorization effectively. For example, secure web-pages should be accessed after valid user id and password. There must not be any navigation path which bypass this validation.

Efficiency:The design of the WebApp and its interface should optimize the user's work efficiency.

Flexibility:The interface should be flexible to enable some users to complete task directly. It should allow other user to explore the WebApp in random fashion. In each case, it should allow the user to understand where he is and provide functionality that can undo mistakes.

***Focus*:**The WebApp interface and the content it presents should stay focused on the user task(s) at hand.

***Fitt's Law*:** This is a principle that can be applied to user interface design because his calculation included factors such as the *time* it takes the average person to select a control in relation to *size* and *motion*. In web design terms, this means that by increasing the size of an icon, it is possible to decrease the time it takes to select an item from a menu. Regular sized icons are faster than smaller icons and there is an obvious benefit in placing icons at the top of a page than at the bottom.

Human interface objects:Human-interface objects are different from the objects found in object-oriented systems. Objects include folders, documents, buttons, menus, and the recycle-bin. They appear within the user's environment and may or may not map directly to an object-oriented program's object. Human interface objects that can be seen are quite familiar in graphic user interfaces. Objects that are recognised by another sense such as hearing or touch are less familiar or are not necessarily recognized by us as being objects. For example, ring tones are auditory objects.

Latency reduction:

Latency can often be hidden from users using multi-tasking techniques.Letting user continue with their work while transmission and computation (like downloading a complex graphical image) take place in the background.

In addition to reducing latency, delays must be acknowledged so that the user understands what is happening. This includes (1) providing audio feedback (e.g., a click or bell tone). (2) displaying an animated clock or progress bar to indicate that processing is under way; (3) provide some entertainment (e.g., an animation or text presentation) while lengthy processing occurs.

Learnability:A WebApp interface should be designed to minimize learning time. User can understand interface easily.

***Metaphors*:**On the web, we use images and icons to symbolize different things. When we visit a webpage, we scan to try and find what we need as fast as possible, and images are used to help speed up that process. We can interpret something much faster with familiar styling and images. For example, we can instantly recognize an error when there's something like an exclamation mark, or a yellow or red colour.

Figure 1 – Error sign

***Maintain work product integrity*:**A form completed by the user must be automatically saved so that it will not be lost if any error occurred. Each of us has

experienced this kind of frustrating content lost due to an error. To avoid this, a WebApp should be designed to auto-save all user specific data.

Readability:All information offered using the interface should be readable by young and old users. The interface designer should emphasize readable type styles, font sizes, and colour background choices that improve readability.

Track state:When appropriate, the state of the user interaction should be traced and stored. That way user can continue his/her work after login again. In general, cookies can be designed to store state information.

Visible navigation:Web navigation refers to the process of navigating a network of information resources in the World Wide Web, which is organized as hypertext or hypermedia. The user interface that is used to do so is called a web browser. A well-designed WebApp interface offers rich experience to end user.

In addition to these design principles, Nielsen and Wagner [Ref-3] suggest a few realistic interface design guidelines:

- Reading speed on a monitor is around 25% slower than reading speed for printed copy. Hence, we should not force end-user to read huge amounts of text explaining operation or in navigation.
- Avoid *"under construction"*pages.They increase expectations and create disappointing link.
- *Users prefer not to scroll.* Key information should be positioned within the sizes of a browser window.
- Navigation menus and head-bars should be designed consistently and should be available on all pages that are available to the end-user. The design should not depend on browser.
- Aesthetics should never be substitute of functionality. For example, a simple button might be a better navigation option than an attractive, but ambiguous image or icon whose intent is unclear.
- Navigation options should be clear for all user. The user shouldn't have to search the page for navigation.

A well-designed interface enhance the user's interpretation of the content. Interface should always be well structured and user friendly.

4.3.2 Interface Control Mechanisms

The objectives of a WebApp interface are to:

1) Establish a consistent window to user.
2) Guide the user over a series of interactions with the WebApp.
3) Organize the navigation options and content available to the user.

To design a consistent interface, the designer must use aesthetic design (Section 4.4) to create a clear look for the interface. Then, required to highlight the layout and navigation. For navigation options, the designer should select any interaction mechanism from below list:

- **Navigation menus:** Vertical or horizontal menus display key content or functionality.
- **Graphic icons:** Radio-button, switches or images that allow the user to select some property or decision.
- **Graphic images:** Some selectable graphical symbol/image which implements a link to a content object or WebApp functionality.

Note: one or more of these control mechanisms should be provided at every level of the content hierarchy.

4.4AESTHETIC DESIGN

Aesthetic design is also known as “Look and Feel”. It is making a WebApp more attractive.

4.4.1 Layout Issues

Each Web-page has limited amount of space that can be used for information content, navigation features (menu), non-functional aesthetics etc. The design of this space is planned during aesthetic design.

There are no defined rules for screen layout design. However, we should consider general layout guidelines:

Don’t be afraid of white space: It is not advisable to fill up each pixel of a Web-page with content. It creates confusion for the user.

Emphasize content: Nielsen [Ref-4] recommends that the Web-page should have 80% content. Remaining space dedicated to navigation and other features.

Organize layout elements from top-left to bottom-right:Majority of users scan a Web-page similarly as they scan pages of a book. High priority content elements should be positioned in the upper left part of the page.

Group navigation, content, and function geographically within the page: Humans look for patterns in almost all things. User frustration is likely, if there are no visible patterns in the Web-page.

Don't extend your real estate with the scrolling bar:Users would prefer not to scroll. It is always better to reduce page content to avoid scrolling bar. We should use multiple pages if required to present more content.

*Consider resolution and browser window size when designing layout:*The design should be responsive rather than defining fixed sizes within a layout.

4.4.2 Graphic Design Issues

Graphic design inspects each and every characteristic of the look and feel of a WebApp. The graphic design process includes:

1) Layout of WebApp (section 4.4.1)
2) Define global colour scheme, fonts (size and styles).
3) All media like audio, video and animation.
4) All other aesthetic elements of WebApp.

4.5CONTENT DESIGN

Content design solves two design issues. Each issue required different skill sets to resolve it.

1) Develops a design representation for *content objects* and define *relationships* to one another. This design activity is conducted by Web engineers,
2) Representation of *information* specific to content object. This design activity is conducted by copywriters, graphic designers etc.

4.5.1 Content Objects

The relationship between content objects defined as part of the WebApp analysis model (Block#2-Unit#2-Figure-3). A content objects has attributes which include content specific information.

Below is the simple representation of content objects.

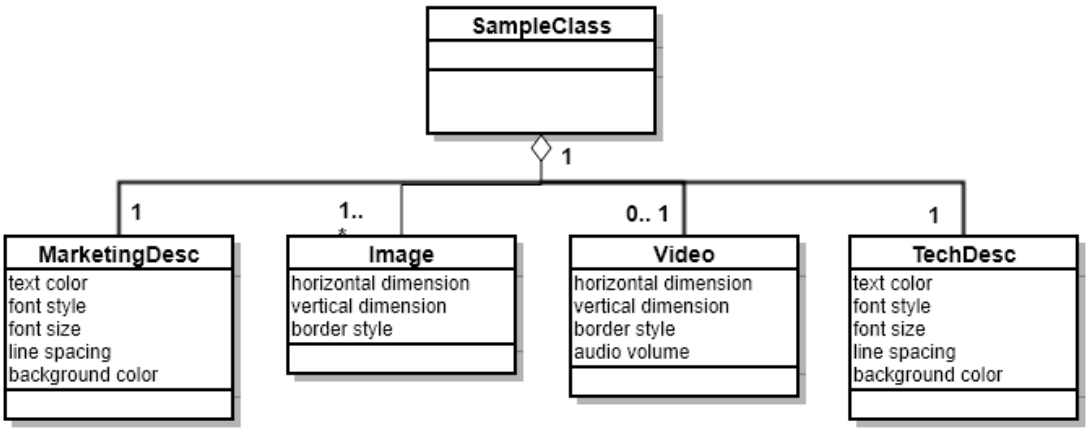


Figure 2 – Design representation of content objects

In this example, `SampleClass` composed of four content objects. Information contained within the content object is noted as attributes. For example, `Image` has the attributes `horizontal dimension`, `vertical dimension`, and `border style`. UML association and aggregation can be used to show relationships between content object. As per Figure 2, the multiplicity notation indicates that `Video` are optional (0 occurrence are possible), one `MarketingDesc` and `TechDescr` is required, and one or more instances of `Image` is used.

4.5.2 Content Design issues

When all content objects are modelled:

- Each object is authored for the information delivery.
- Then, formatted to best meet the customer's requirements.
- Aesthetic design applied to represent the proper look and feel for the content.

As content objects are designed, they are "chunked" (divide into multiple files) [Ref-5] to form WebApp pages due to:

- The number of content objects incorporated into a single page is a function of user needs.
- Constraints forced by download speed of the Internet connections.
- Restrictions forced by the amount of scrolling that the user will tolerate.

4.6 ARCHITECTURE DESIGN

Architecture design is attached with*:*

- The *goals* established for a WebApp*.*
- The *content* to be presented.
- The *users* who will visit.
- The *navigation*paths that has been established.

The architectural designer must identify:

1) *Content architecture*: Content architecture emphases on the content objects or composite objects (group of content objects, like Web pages) are structured for presentation and navigation.
2) *WebApp architecture*: WebApp architecture focuses on the application structure which manage user interaction, handle internal processing tasks, effect navigation, and present content.

In most cases, architecture design is executed in parallel with interface, aesthetic, and content design. The WebApp architecture may have a strong impact on navigation.

4.6.1 Content Architecture

The design of *content* architectureconcentrates on the overall hypermedia structure of the WebApp.

The design can choose from four different content structures [Ref-5]:

Linear structures: Asshown in Figure 3, a linear website is a type of website that has a linear-style method of navigation. Website navigation refers to the order of the web-pages that user access when they visit that site. A simple example is a tutorial or e-book presentation.

Figure 3 – Linear Structure

*Grid structures:*In architecture design, a grid is a two dimensional structure made up of a series of crossing horizontal and vertical lines. The grid serves as a framework on which a designer can organize design elements. Figure 4, shows grid structures.

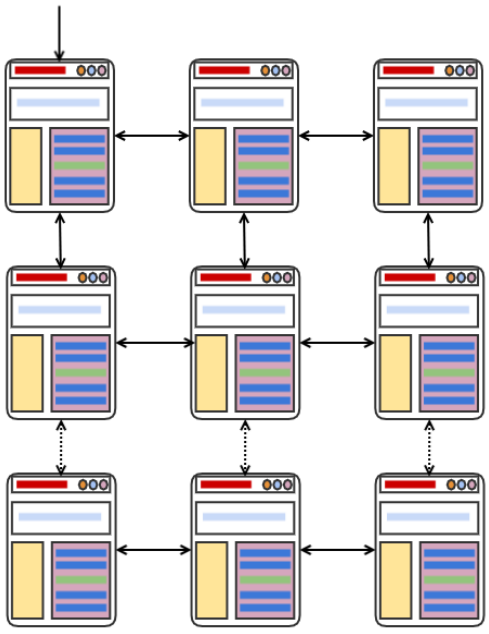


Figure 4 – Grid structure

For example, consider a scenario where an e-commerce site selling holiday packages. The horizontal dimension of the grid sows the travel agencies. The vertical dimension shows the holiday packages. So, a user can navigate the grid horizontally to find the travel agency and then vertically to search the offering provided by travel agency.

Hierarchical structures: These are the most common types of website structures. Hierarchical structuresbegin with a high level of information display at parent pages.Then, filters down into more detailed information shown at child pages. Sometimes these structures are called trees, and they are very similar to organizational charts in corporations. Figure 5, shows Hierarchical structures.

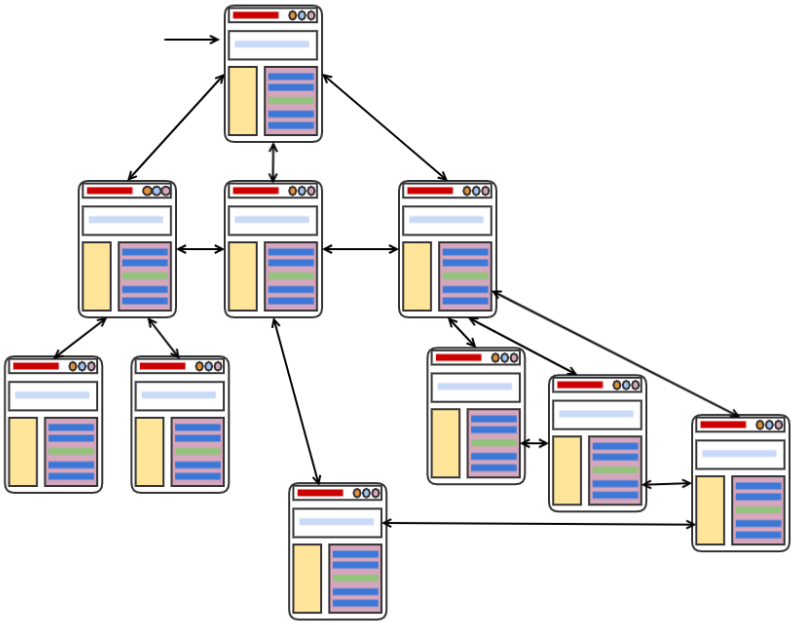


Figure 5 - Hierarchical structures

4.6.2 WebApp Architecture

WebApp architecture defines an infrastructure that enables a Web-based application to achieve its business objectives. Jacyntho and his colleagues [Ref-11] describe the basic characteristics of this infrastructure in the following manner:

Applications should be built using layers in which different concerns are taken into account; in particular, application data should be separated from the page's contents

(navigation nodes) and these contents, in turn, should be clearly separated from the interface look-and-feel (pages).

Jacyntho and his colleagues [Ref-11] has suggested a three layer design architecture. It decouples user interface from navigation controlling and application behaviour. Three layer architecture simplifies implementation and improves reuse.

The Model-View-Controller (MVC) architecture [Ref-12] is WebApp infrastructure models that decouples the user interface from the WebApp functionality and informational content. It has three layers:

- *The model:*Model contains all application specific content and processing logic, including all content objects, access to external data/information sources.
- *The view:*Viewcontains all interface specific functions and enables the presentation of content and processing logic.
- *The controller:*Controller manages access to the model and the view and coordinates the flow of data between them.

A schematic representation of the MVC architecture is shown in Figure 6.

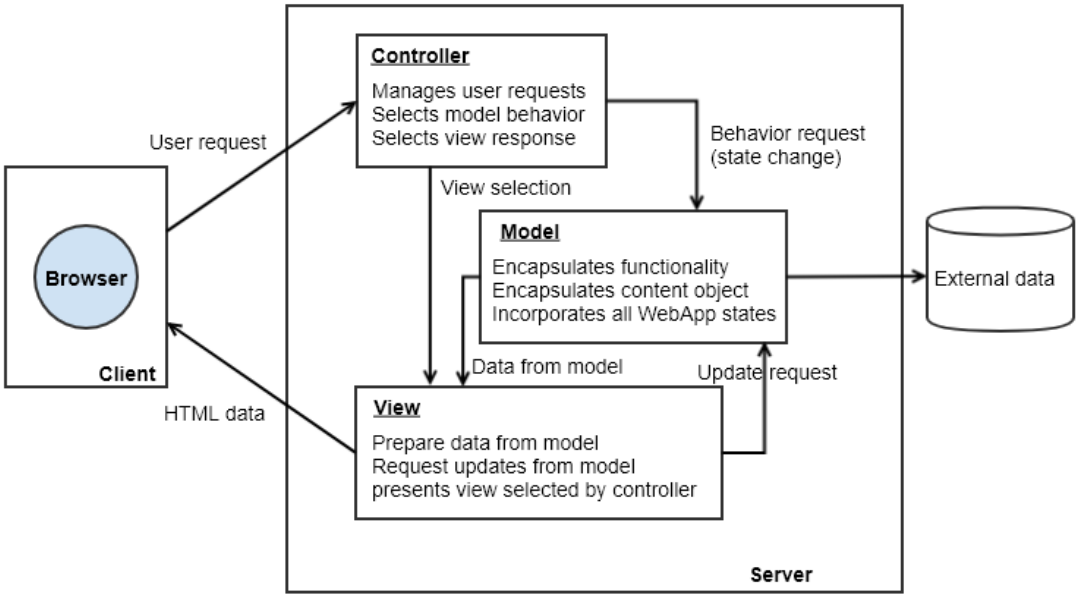


Figure 6 – The MVC architecture [Ref-11]

As shown in Figure 6, user requests are handled by the controller. Based on type of request, a behaviour request is transmitted to the model. The model object implements the functionality required to serve the request. The model object can access database. The controller selects the view object based on the user request. The data created by the model must be formatted by the respective view object. Response generated from the view is sent back to the client browser.

4.7NAVIGATION DESIGN

The designer must define navigation paths that enable users to access WebApp content and functions. For Navigation design, the designer should:

1) Identify the navigation path for different users of the web-site.
2) Define the method of achieving the navigation.

4.7.1 Navigation Semantics

Navigation design begins with consideration of the user hierarchy and related use-cases developed for each category of user (actor). Each actor may use the WebApp somewhat differently and therefore have different navigation requirements.In addition, the use-cases developed for each actorwill define a set of classes that encompass one or more content objects or WebApp functions. As each user interacts with the WebApp, she encounters a series of navigation semantic units (NSUs) - "a set of information and related navigation structures that collaborate in the fulfilment of a subset of related user requirements” [Ref-9].

4.7.2 Navigation Syntax

As design proceeds, the method of navigation are defined. Among many possible options are:

- Individual navigation link: Text-based links, icons, buttons and switches, and graphical metaphors.

- Horizontal navigation bar: Lists major content or functional categories in a bar containing appropriate links. In general, between four and seven categories are listed.
- Vertical navigation column: (l) lists major content or functional categories, or (2) lists virtually all major content objects within the WebApp. If the second option is chosen, such navigation columns can "expand" to present content objects as part of a hierarchy.
- Tabs: A metaphor that is nothing more than a variation of the navigation bar or column, representing content or functional categories as tab sheets that are selected when a link is required.
- Site maps: Provide an all-inclusive table of contents for navigation to all content objects and functionality contained within the WebApp.

In addition to choosing the method of navigation, the designer should also establish appropriate navigation agreements and helps. For example, icons and graphical links should look "clickable" by angle the edges to give the image a 3-dimensional look. Audio or visual response should be given the user with an indication that a navigation option has been chosen. For text based navigation, colour should be used to indicate navigation links and to provide an indication of links already travelled. These are few design conventions that make navigation user-friendly.

4.8 COMPONENT LEVEL DESIGN

Modern WebApp provideadvanced processing functions that:

1. Perform client-site processing to generate dynamic content and navigation capability.
2. Offer calculation or data processing capability that are suitable for the WebApp's business domain.
3. Provide advanced database query and access.
4. Initiate data interfaces (APIs) with external corporate systems.

To achieve above capabilities, the Web engineer must design and construct program components similar to software components for conventional software.

4.9 HYPERMEDIA DESIGN PATTERNS

Design patterns that are used in Web engineering includes two major classes:

1) Generic design patterns which are applicable to all types of software.
2) Hypermedia design patterns which are specific to WebApps.

Design patterns are a generic approach for solving similar small design problem. In the context of Web-based systems German and Cowan [Ref-10] suggest the following patterns categories:

Architectural patterns. These patterns help in the design of content and WebApp architecture.

Component construction patterns. These patterns suggest to combine WebApp components (e.g., content objects, functions) into composite components.

Navigation patterns. These patterns help in the design of navigation links and the overall navigation flow of the WebApp.

Presentation patterns. These patterns help in the presentation of content. Patterns in this category address how to organize user interface control functions for better usability; how to show the relationship between an interface action and the content objects it affects; how to establish effective content hierarchies; and many others.

Behaviour/user interaction patterns. These patterns help in the design of user-machine interaction. Patterns in this category address how the interface informs the user of the consequences of a specific action; how a user expands content based on usage context and user desires; how to best describe the destination that is implied by a link; how to inform the user about the status of an on-going interaction and others.

4.10 OBJECT-ORIENTED HYPERMEDIA DESIGN METHOD

Object-Oriented Hypermedia Design Method (OOHDM) was originally proposed by Daniel Schwabe and his colleagues [Ref-6], [Ref-7]. OOHDM is collection of four different design activities:

1) Conceptual design

2) Navigational design
3) Abstract interface design
4) Implementation

Let's discuss each in detail.

4.10.1 Conceptual Design for OOHDM

Conceptual design creates a representation of:

- The subsystems
- Classes
- Relationships

UML can be used to create appropriate diagrams (like class diagrams, aggregations and composite class representations, collaboration diagrams).

4.10.2 Navigational Design for OOHDM

Navigational design recognises a set of ***objects,*** *based on*classes defined in conceptual design. UML can be used to create appropriate use-cases, state charts, and sequence diagrams. These diagrams helps designer to understand navigational requirements.

OOHDM uses a predefined set of navigation classes like nodes, links, anchors, and access structures [Ref-7].

4.10.3 Abstract Interface Design and Implementation

The abstract interface design to specify the separation of the user interface from the application component. A formal model of interface objects, known as an abstract data view (ADV). ADV is used to represent the relationship between interface objects and navigation objects.

The OOHDM implementation activity define a design iteration that is particular to the environment in which the WebApp will function. Classes, navigation, and the interface are each categorised in a manner that can be constructed for the

client/server environment, operating systems, support software, programming languages, and other environmental characteristics that are relevant to the problem.

4.11 LET US SUM UP

WebApp design is an essential activity. It decides what and how the information presented to the users. It includes several different aspects, including webpage layout, content production and graphic design.

Interface design begins with a careful examination of the end user and user hierarchy. Effective interfaces are visually attractive and simple to use. The objectives of a WebApp interface are to establish a consistent window to user. To guide the user over a series of interactions with the WebApp and to organize the navigation options and content available to the user.

Aesthetic design describes the "look and feel" of the WebApp and includes colour schemes, layout, menu, text size, font and placement, the use of graphics, and related aesthetic decisions. There are no defined rules for design. However, a set of graphic design guidelines provides the basis for a design approach.

Content design defines a design representation for content objects and define relationships to one another. It is also representation of information specific to content object.

Architecture design identifies the overall hypermedia structure for the WebApp and includes both content architecture and WebApp architecture.

Navigation design represents the path between content objects and for each category of users. Navigation design begins with consideration of the user hierarchy and related use-cases developed for each category of user (actor).

Component design define the process to implement WebApp functional component.

Patterns for WebApp design include generic design patterns which apply to all types of software and hypermedia patterns which are particularly relevant for WebApps. Architecture, navigation, component, presentation, and behaviour/user design patterns have been discussed.

The Object-Oriented Hypermedia Design Method (OOHDM) includes conceptual design, navigational design, abstract interface design, and implementation.

4.12 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Question 1: Write a simple one-line definition of each of the following key terms. Try to do this without referring back to the notes if possible.

- Anticipation
- Communication
- Consistency
- Efficiency
- Flexibility
- Learnability
- Readability

Question 2: What is use of cookies?

Question 3: What is white space and how does it affect content on the web?

Question 4: What is responsive design on a web page? Explain benefits of responsive design.

Question 5: What is OOHDM?

4.13 FURTHER READING

Books

- Software Engineering by Roger S. Pressman Sixth Edition McGraw Hill Publications.

Web sites

- https://en.wikipedia.org/wiki/Web_design
- https://www.tutorialspoint.com/struts_2/basic_mvc_architecture.htm

4.14 ASSIGNMENTS

- Explain what is linear structures, Grid structures and Hierarchical structures?
- What is role and responsibility of the web designer?

4.15 ACTIVITIES

- What is web designing? Explain in your own words.

4.16 CASE STUDIES

- Explain Content architecture and WebApp architecture, in Architecture design perspective.
- Explain MVC architecture in detail.
- Should resolution and browser window have any impact on the font-size of the site content? Explain.

4.17 REFERENCES

- Ref-1: Software Engineering By Roger S. Pressman Sixth Edition McGraw Hill Publications.
- Ref-2: Tognozzi, B., "First Principles," askTOG, 2001, available at http://wvw.asktog.com/ asics/fi rstPrinciples.html.
- Ref-3: [NIE96] Nielsen, J., and A. Wagner, "User Interface Design for the WWW," Proc. CHI '96 Conf. On Human Factors in Computing Systems, ACM Press, 1996, pp. 330-331.
- Ref-4: [NIEOO] Nielsen, J., Designing Web Usability, New Riders Publishing, 2000.
- Ref-5: [POWOOl Powell, T., Web Design, McGraw-Hill/Osborne, 2000.
- Ref-6: Schwabe, D„ and G. Rossi, “The Object-Oriented Hypermedia Design Model," CACM. vol. 38, no. 8, August 1995, pp. 45-46.
- Ref-7: Schwabe, D., and G. Rossi, Developing Hypermedia Applications Using OOHDM, Proc. Workshop oil Hypermedia Development Process, Methods and Models, Hypertext ‘98, 1998, download from http://citeseer.nj.nec.com/schwabe 98developing.html.
- Ref-8: Dix, A., "Design of User interfaces for the Web,” Proc. Of User Interfaces to Data Systems Conference, September 1999, download from http:/Avmv.comp.lancs. ac.uk/computing/users/dixa/topics/webarch/.

- Ref-9: Cachero, C., et al„ "Conceptual Navigation Analysis: a Device and Platform Independent Navigation Specification," Proc. 2nd inti. Workshop on Web-Oriented Technology, June 2002, download from www.dsic.upv.es/~west/iwwost02/papers/cachero.pdf.
- Ref-10: (GER00| German, D., and D. Cowan, "Toward a Unified Catalog of Hypermedia Design Patterns," Proc. 33rd Hawaii Inti. Conf. on System Sciences, IEEE, vol. 6, Maui, Hawaii, June 2000, download from www.turingmachine.org/~dmg/research/papers/dmg_hicss2000.pdf
- Ref-11: Jacyntho, D., D. Schwabe, and G. Rossi, "An Architecture for Structuring Complex Web Applications," 2002, available at http://www2002.org/CDROM/altemate/478/.
- Ref-12: Krasner, G., and S. Pope, "A Cookbook for Using the Model-View Controller User Interface Paradigm in Smalltalk-80," Journal of Object-Oriented Programming, vol. 1, no. 3, August/September 1988, pp. 26-49.

Block-3
Introduction to UML and UML Diagrams

Unit 1: Fundamentals of UML

1

Unit Structure

1.1 Learning Objective

After studying this unit student should be able to:

- Define Unified Modelling Language, Modelling and Diagram
- Know UML background
- List goals of UML
- Describe the purpose of UML
- Differentiate between static and dynamic UML diagrams

1.2 Key Terms

UML: Unified Modelling Language (UML) is a graphical modelling language that provides syntax for describing the major elements (called artifacts in UML) of software systems.

Modelling: Modelling is the designing of software applications before coding (implementation in a particular programming language). A model is a representation or simplification of reality.

Diagram:A diagram is the graphical presentation of a set of elements, most often rendered as a connected graph of vertices (things) and arcs (relationships). You draw diagrams to visualize a system from different perspectives, so a diagram is a projection into a system.

1.3Introduction

The norm for engineering field is to model first before implementation. Modelling can be mathematically or graphically, This unit presents a Unified Modelling Language as a language to model object-oriented artifacts.

For many years, the term object oriented (OO) was used to denote a software development approach that used one of a number of object-oriented programming languages (e.g., Java, C++). Today, the OO paradigm encompasses a complete view of software engineering.

The norm for engineering analysis and design systems emphasizes modelling of the system first before implementation. Modelling is a proven and well-accepted

engineering technique. Modelling can be done mathematically or by any other common notation understood by many engineers of the same field worldwide.

Software Engineering had lacked such a notation. Between 1989 and 1994, more than 50 software modelling languages were in common use – each of them carrying their own notations. Each language contained syntax peculiar to itself, whilst at the same time, each language had elements which bore striking similarities to the other languages, and bad enough neither of these languages were complete.

1.4UML Background

In the mid 1990's, three methods emerged as the strongest. Ariadne Training (2001) state that these three methods had begun to converge, with each containing elements of the other two. Each method had its own particular strengths:

Booch was excellent for design and implementation. Grady Booch had been a major player in the development of Object Oriented techniques for the language.

Object Modelling Technique (OMT) was best for analysis and data-intensive information systems.

Object Oriented Software Engineering (OOSE) featured a model known as Use Cases. Use Cases are a powerful technique for understanding the behavior of an entire system (an area where OO has traditionally been weak).

So, the Unified Modelling Language (UML) is largely the product of three well known software engineers, - Grady Booch, Ivar Jacobson and James Rumbaugh. In 1994, James Rumbaugh, the creator of OMT joined Grady Booch at Rational Corp. The aim of the partnership was to merge their ideas into a single, unified method. By 1995, the creator of OOSE, Ivar Jacobson, had also joined Rational, and his ideas (particularly the concept of “Use Cases”) were fed into the new Unified Method – now called the Unified Modeling Language (Ariadne Training, 2001).

The Unified Modelling Language or the UML as a graphical modelling language aimed at providing syntax for describing the major elements of software systems (called artifacts in the UML). The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems. In this course we need to explore the main aspects of the UML, and describe how the UML can be applied to software development process.

1.5What is UML?

Booch et al, (2005) defines the Unified Modelling Language (UML) to be a language for: Specifying, Visualizing, Constructing and Documenting the artifacts of a software-intensive system. It is a standard language for writing software blueprints. It is a graphical language for capturing the artifacts of software development. UML is the de-facto standard for Object Oriented modelling.

a) The UML as a Language for Visualizing:

For many programmers, the distance between thinking of an implementation and then pounding it out in code is close to zero. You think it, you code it. In fact, some things are best cast directly in code. Text is a wonderfully minimal and direct way to write expressions and algorithms

b) The UML as a Language for Specifying:

Specifying means of building models that are precise, unambiguous, and complete. In particular, the UML addresses the specification of all the important analysis, design, and implementation decisions that must be made in developing and deploying a software-intensive system.

c) The UML as a Language for Constructing:

The UML is not a visual programming language, but its models can be directly connected to a variety of programming languages. This means that it is possible to map from a model in the UML to a programming language such as Java, C++, Visual Basic or PHP, or even to tables in a relational database or the persistent store of an object-oriented database.

Things that are best expressed graphically are done so graphically in the UML, whereas things that are best expressed textually are done so in the programming language. This mapping permits forward engineering: The generation of code from a UML model into a programming language.

d) The UML as a Language for Documenting:

A healthy software organization produces all sorts of artifacts in addition to raw executable code. These artifact include (but are not limited to): Requirements, Architecture, Design, Source code, Project plans, Tests, Prototypes, Releases.

Depending on the development culture, some of these artifacts are treated more or less formally than others. Such artifacts are not only the deliverables of a project, they are also critical in controlling, measuring, and communicating about a system during its development and after its deployment.

The UML addresses the documentation of a system's architecture and all of its details. The UML also provides a language for expressing requirements and for tests. Finally, the UML provides a language for modelling the activities of project planning and release management.

The UML can also be a Language for Communication. That is communication with customers has proven to be equally problematic. Graphical modelling can make people (technical and nontechnical) understand the artifacts of the software system expected. In software development, some of the things that require communication include requirements, design, implementation, and deployment. UML is a language designed to communicate these things.

As with any language, the UML has its own notation and syntax. It does not tell you how to develop software. It can be applied in any software development processes; waterfall model, spiral model, iterative, incremental frameworks. Its notation comprises a set of specialized shapes for constructing different kinds of software diagrams. Each shape has a particular meaning. UML is a generic, broad language enabling the key aspects of a software development to be captured on paper

Goals of UML are to:

- Provide modellers with an expressive, visual modelling language to develop and exchange meaningful models
- Provide extensibility and specialization mechanisms to extend core concepts
- Support specifications that are independent of particular programming languages and development processes
- Provide a basis for understanding specification languages
- Encourage the growth of the object tools market
- Supports higher level of development with concepts such as components frameworks or patterns

1.6UML with Modelling

Modelling is the designing of software applications before coding (implementation in a particular programming language). A model is a representation or simplification of reality. It provides a blueprint of a system. A model does not dictate or show how the implementation will actually be done. It just shows what, who, which, where, when etc. Model-driven analysis and design emphasizes the drawing of pictorial system models to document and validate both existing and/ or proposed systems. Ultimately, the system model becomes the blueprint for designing and constructing an improved system.

Modelling is a central part of all the activities that lead up to the deployment of good software. We build models to:

- Communicate the desired structure and behavior of our system.
- Visualize and control the system's architecture.
- Better understand the system we are building, often exposing opportunities for simplification and reuse.
- Manage risk.
- Modelling manages Complexity
- Modelling Promotes Reuse

Modelling ensures that:

- Business functionality is complete and correct,
- End-user needs are met, and
- Program design meets requirements for scalability, robustness, security, extensibility, and other characteristics, all these must be ensured before implementation in code

There are many elements that contribute to a successful software organization; one common thread is the use of modeling. Modeling is a proven and well-accepted engineering technique. There are three basic building blocks: Elements which are main "citizens" of the model, relationships i.e. relationships that tie elements together and Diagrams which are mechanisms to group interesting collections of elements and relationships. These elements are used to represent complex structures.

1.7Let us sum up

1 UML Stands for _____

   A. Uniform Modeling Language
   B. Unified Modeling Language
   C. Universal Modeling Language
   D. United Modeling Language

2 Which of the following is best for analysis and data-intensive information system?

   A. OMT
   B. OOSE
   C. Both OMT and OOSE
   D. Neither OMT nor OOSE

3 Which of the following is true?

   A. UML is a Language for Visualizing
   B. UML is a Language for Specifying
   C. UML is a Language for Constructing
   D. UML is a Language for Documenting
   E. All of the above.

4 Modelling ensures that

   A. Business functionality is complete and correct,
   B. End-user needs are met, and
   C. Program design meets requirements for scalability, robustness, security, extensibility, and other characteristics, all these must be ensured before implementation in code
   D. All of these

5 UML does not tell you how to develop software. True / False

6 The UML addresses the documentation of a system's architecture and all of its details. True / False

7 UML is a standard language for writing software. True / False

1.8Let us sum up

The purpose of the Unified Modeling Language is to visualize, specify, construct, document and communicate object-oriented systems and it is gaining adoption as a standard language. The language provides the notations to produce models. In this unit we have learn about key terms, understand UML background, what is UML, how UML is used for modelling and goals of UML.

1.9Further Reading

- https://www.duhoctrungquoc.vn/edu/en/UML
- https://shms.sa/authoring/53030-unified-modeling-language/view
- https://learn.saylor.org/mod/page/view.php?id=32969
- https://www.duhoctrungquoc.vn/edu/en/UML

1.10Assignments

- What is UML?
- Define: Modeling
- Define: Diagram
- What are the goals of UML?
- List various Open Source Tools for Modelling

1.11Answer to Check Your Progress

1. B - Unified Modeling Language
2. A – OMT
3. E – All of these
4. D – All of these
5. True
6. True
7. False

Unit 2: Introduction to UML

2

Unit Structure

2.1 Learning Objective

After studying this unit student should be able to:

- Describe History of UML
- Define Structure Diagrams
- Define Behaviour Diagrams
- Define Interaction Diagrams
- Use UML diagrams to describe static and dynamic system model
- List Open-Source UML Tools

2.2 Introduction

UML is managed as a de facto industry standard by the Object Management Group (OMG). This unit includes a review about UML History, Definition and its main diagrams.

The Unified Modelling Language is a standardized general-purpose modelling language and nowadays is managed as a de facto industry standard by the Object Management Group (OMG). UML includes a set of graphic notation techniques to create visual models of software-intensive systems.

2.3History

UML was invented by James Rumbaugh, Grady Booch and Ivar Jacobson. After Rational Software Corporation hired James Rumbaugh from General Electric in 1994, the company became the source for the two most popular object-oriented modelling approaches of the day: Rumbaugh's Object-modelling technique (OMT), which was better for object-oriented analysis (OOA), and Grady Booch's Booch method, which was better for object-oriented design (OOD). They were soon assisted in their efforts by Ivar Jacobson, the creator of the object-oriented software engineering (OOSE) method. Jacobson joined Rational in 1995, after his company, Objectory AB, was acquired by Rational.

2.4Definition

The Unified Modelling Language (UML) is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software-intensive system under development. UML offers a standard way to visualize a system's architectural blueprints, including elements such as activities, actors, business processes, database schemas, components, programming language statements, and reusable software components.

UML combines techniques from data modelling (entity relationship diagrams), business modelling (work flows), object modelling, and component modelling. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies.

2.5Models and Diagrams

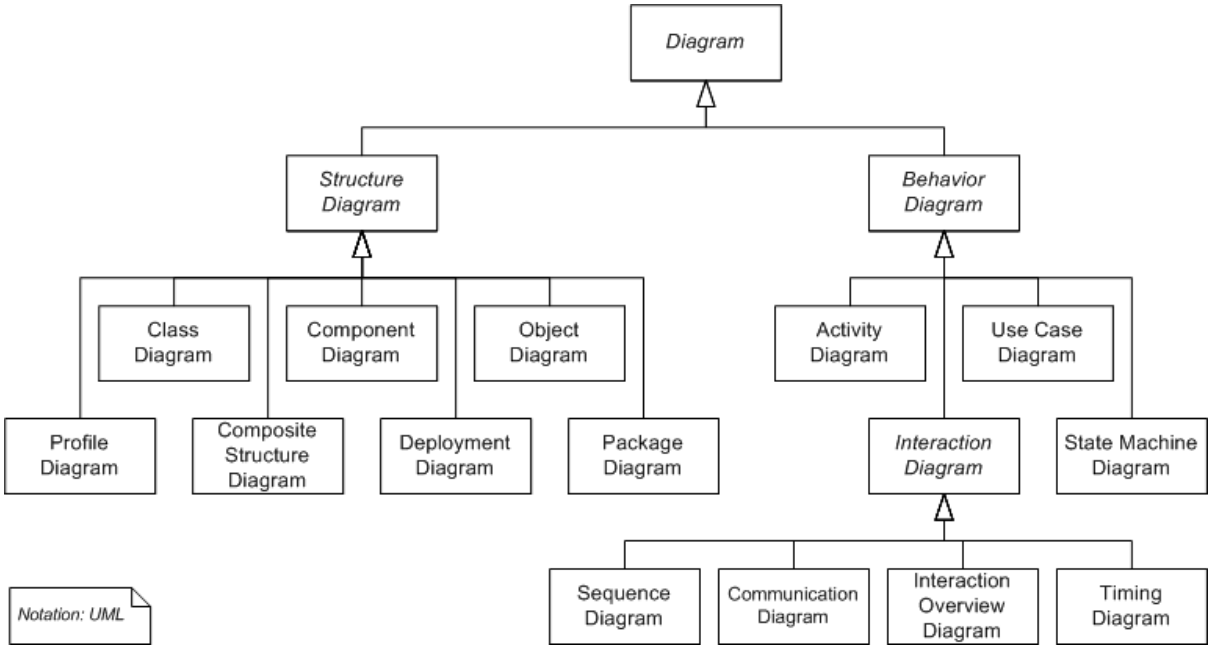
It is important to distinguish between the UML model and the set of diagrams of a system. A diagram is a partial graphic representation of a system's model. The model also contains documentation that drive the model elements and diagrams.

UML diagrams represent two different views of a system model:

- Static (or structural) view: emphasizes the static structure of the system using objects, attributes, operations and relationships. The structural view includes class diagrams and composite structure diagrams.
- Dynamic (or behavioural) view: emphasizes the dynamic behaviour of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence diagrams, activity diagrams and state machine diagrams.

2.6Diagrams Overview

In UML 2.2 there are 14 types of diagrams divided into two categories. Seven diagram types represent structural information, and the other seven represent general types of behaviour, including four that represent different aspects of interactions. These diagrams can be categorized hierarchically as shown in the following diagram:



2.7Structure Diagrams

Structure diagrams emphasize the things that must be present in the system being modelled. Since structure diagrams represent the structure, they are used extensively in documenting the software architecture of software systems.

- Class diagram: describes the structure of a system by showing the system's classes, their attributes, and the relationships among the classes.
- Component diagram: describes how a software system is split up into components and shows the dependencies among these components.
- Composite structure diagram: describes the internal structure of a class and the collaborations that this structure makes possible.
- Deployment diagram: describes the hardware used in system implementations and the execution environments and artifacts deployed on the hardware.
- Object diagram: shows a complete or partial view of the structure of a modeled system at a specific time.
- Package diagram: describes how a system is split up into logical groupings by showing the dependencies among these groupings.

- Profile diagram: operates at the metamodel level to show stereotypes as classes with the <<stereotype>> stereotype, and profiles as packages with the <<profile>> stereotype. The extension relation (solid line with closed, filled arrowhead) indicates what metamodel element a given stereotype is extending.

2.8Behaviour Diagrams

Behaviour diagrams emphasize what must happen in the system being modelled. Since behaviour diagrams illustrate the behaviour of a system, they are used extensively to describe the functionality of software systems.

- Use case diagram: describes the functionality provided by a system in terms of actors, their goals represented as use cases, and any dependencies among those use cases.
- Activity diagram: describes the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.
- state machine diagram: describes the states and state transitions of the system.

2.9Interaction Diagrams

Interaction diagrams, a subset of behaviour diagrams, emphasize the flow of control and data among the things in the system being modelled:

- Sequence diagram: shows how objects communicate with each other in terms of a sequence of messages. Also indicates the lifespans of objects relative to those messages.
- Communication diagram: shows the interactions between objects or parts in terms of sequenced messages. They represent a combination of information taken from Class, Sequence, and Use Case Diagrams describing both the static structure and dynamic behaviour of a system.
- Interaction overview diagram: provides an overview in which the nodes represent communication diagrams.

- Timing diagrams: a specific type of interaction diagram where the focus is on timing constraints.

2.10UML Modelling Tools

To draw UML diagrams, all you need is a pencil and a piece of paper. However, for a software engineer that seems a little outdated, hence most of us will use tools. The simplest tools are simply drawing programs, like Microsoft Visio or Dia. The diagrams generated this way look nice, but are not really that useful, since they do not include the code generation feature.

Hence, when deciding on a UML modelling tool (sometimes also called CASE tool) you should make sure, that it allows for code generation and even better, it should also allow for reverse engineering. Combined, these two are also referred to as round-trip engineering. Any serious tool should be able to do that. Finally, UML models can be exchanged among UML tools by using the XMI interchange format, hence you should check that your tool of choice supports this.

Since the Rational Software Corporation so to say 'invented' UML, the most well-known UML modelling tool is IBM Rational Rose. Other tools include Rational Rhapsody, Visual Paradigm, Magic Draw UML, StarUML, ArgoUML, Umbrello, BOUML, PowerDesigner, Visio and Dia. Some of popular development environments also offer UML modelling tools, i. e. Eclipse, NetBeans, and Visual Studio.

UML tools (open source or free):

- Acceleo – Eclipse and EMF template-based system for source-code generation from UML models.
- ArgoUML – a Java-based UML engineering tool.
- ATL - a QVT-tool allowing to transform (among others) UML models into other models, including UML, Java, etc. ATL provides a complete open-source solution. Available from the Eclipse GMT project (Generative Modeling Tools).
- BOUML – under GPL, written in C++/Qt
- Dia – a GTK+/GNOME diagramming tool that also supports UML (licensed under the GNU GPL)
- DOME - The DOmain Modeling Environment, written in Smalltalk.

- Eclipse – with Eclipse Modeling Framework (EMF) and UML 2.0 (meta model without GUI) projects.
- Fujaba – UML and Java development platform; Eclipse version available.
- Gaphor [3] – a GTK+/GNOME UML 2.0 modeling environment written in Python.
- GenMyModel Online UML modeler – cloud-based, UML 2.0 with code generators written in Javascript and HTML5.
- JUDE/Community – Object-Oriented Analysis and Design. JUDE/Community. though free to use, does not provide open source.
- MetaUML – Textual notation for UML. Diagram-rendering based on MetaPost, suitable for LaTeX typesetting.
- MonoUML – based on the latest Mono, GTK+ and ExpertCoder.
- NetBeans – with NetBeans IDE 5.5 Enterprise Pack.
- Poseidon for UML (Community Edition) – Commercial tool based on ArgoUML. A cost-free version can be used to view, create, and edit models, but the export options are not available without a rent subscription
- StarUML – a UML/MDA platform for Microsoft Windows, licensed under a modified version of GNU GPL, mostly written in Delphi
- Software Ideas Modeler – a universal software and data modeling platform with UML support for Microsoft Windows, free edition Standard (for non-commercial use)
- Taylor - model-driven architecture on rails (licensed under the GNU LGPL)
- Umbrello UML Modeller – part of KDE.
- UML Pad – a UML modeller written in C++/wxWidgets (licensed under the GNU GPL)
- UML Pad – a UML tool for PalmOS
- UMLet – a Java-based UML tool (licensed under the GNU GPL)
- Visual Paradigm SDE Community Edition - Visual Paradigm SDE integrates with all leading IDEs (Visual Studio®, Eclipse/WebSphere®, Borland JBuilder®, NetBeans/Sun™ ONE, IntelliJ IDEA™, Oracle JDeveloper, BEA WebLogic Workshop™)

2.11Let us Sum Up

UML is managed as a de facto industry standard by the Object Management Group (OMG). In this unitwe have learned about UML History, Definition and its main diagrams.

We have discussed about structure diagram, behaviour diagrams, interaction diagrams and UML Modelling Tools. The Unified Modeling Language is a standardized general-purpose modeling language and nowadays is managed as a de facto industry standard by the Object Management Group (OMG). UML includes a set of graphic notation techniques to create visual models of software-intensive systems.

2.12Check Your Progress

1 UML was invented by _____

A. James Rumbaugh and Grady Booch
B. James Rumbaugh, Grady Booch and Ivar Jacobson
C. James Rumbaughand Ivar Jacobson
D. Grady Booch and Ivar Jacobson

2 UML is managed as a de facto industry standard by the _____

A. OMT
B. OMG
C. OOT
D. OBC

3 _____ shows the interactions between objects or parts in terms of sequenced messages

A. Sequence diagram
B. Interaction overview diagram
C. Timing Diagram
D. Communication Diagram

4 _____provides an overview in which the nodes represent communication

diagrams

A. Sequence diagram
B. Interaction overview diagram
C. Timing Diagram
D. Communication Diagram

5 _____describes the functionality provided by a system in terms of actors, their goals represented as use cases, and any dependencies among those use cases

A. Use case diagram
B. Activity Diagram
C. State Machine Diagram
D. None of these

2.13Assignment

- What is Structure Diagram?
- What is Behaviour Diagrams?
- What is Interaction Diagrams
- List and explain UML Modelling Tools

2.14Further Reading

- https://www.duhoctrungquoc.vn/edu/en/UML
- https://shms.sa/authoring/53030-unified-modeling-language/view

2.15 Answer to Check Your Progress

1. B – JamesRumbaugh, Grady Booch and Ivar Jacobson
2. B – OMG
3. D – CommunicationDiagram
4. B – Interaction Overview Diagram
5. A – Use Case Diagram

Unit 3: Fundamentals of UML Diagrams

3

Unit Structure

3.1 Learning Objective

After studying this unit student should be able to:

- List Basic Use Case Notation
- Use Class diagram
- Use Object diagram
- Use case diagram
- Use Sequence diagram
- Use Collaboration diagram
- Use State chart diagram
- Use Activity diagram
- Use Component diagram
- Use Deployment diagram

3.2 Introduction

A diagram is the graphical presentation of a set of elements, most often rendered as a connected graph of vertices (things) and arcs (relationships). You draw diagrams to visualize a system from different perspectives, so a diagram is a projection into a system. UML has a lot of different diagrams (models). The reason for this is that it is possible to look at a system from different viewpoints. UML being a graphical language includes nine such diagrams models):

1. Class diagram
2. Object diagram
3. Use case diagram
4. Sequence diagram
5. Collaboration diagram
6. Statechart diagram
7. Activity diagram
8. Component diagram
9. Deployment diagram

UML nine diagrams can be divided into two categories

(a) Four diagram types represent static application structure:

- Class Diagram
- Object Diagram
- Component Diagram
- Deployment Diagram

(b) Five represent different aspects of dynamic behaviour

- Use Case Diagram
- Sequence Diagram
- Activity Diagram
- Collaboration Diagram
- State chart Diagram

3.3A Class Diagram

In Object Oriented design and development terms, a class has a name, a set of methods (also known as operations) and related data members (also known as attributes) as shown in Figure. Class by itself is not very useful. A large software system may have thousands of classes, Modelling the relationships (association, inheritance, composition or aggregation) between them really defines systems behaviour.

Class diagrams shows a set of classes, interfaces, and collaborations and their relationships. Class diagrams are the most common diagrams found in modelling object-oriented systems. It is an essential aspect of any Object-Oriented Design method.

Class diagrams address the static design view of a system. They are used at the analysis stage as well as design. Class diagrams that include active classes address the static process view of a system. Class Diagram syntax are being used to draw a plan of the major concepts for anyone to understand. This is called the Conceptual Model. Together with use cases, a conceptual diagram is a powerful technique in requirements analysis. Figure shows an example of a class diagram.

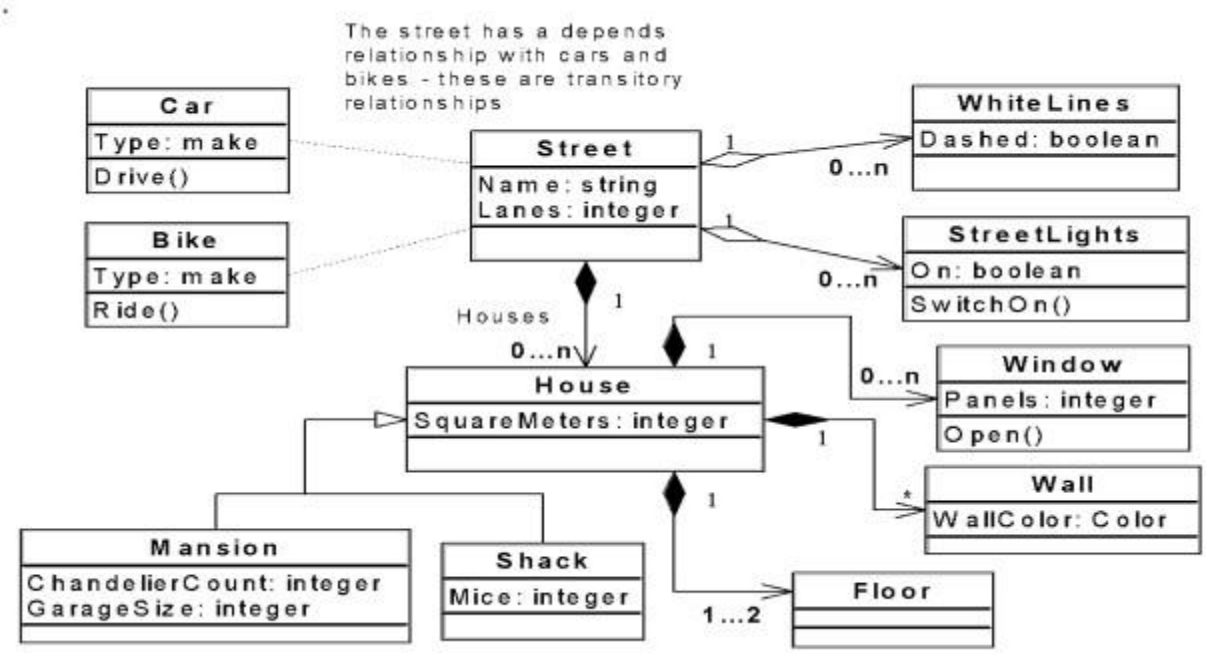


Figure: An Example of a Class Diagram

3.4Object Diagram

Object Diagrams show a set of objects and their relationships. They are static snapshots of element instances found in class diagrams. Figure 2.3 shows an example of an object diagram.

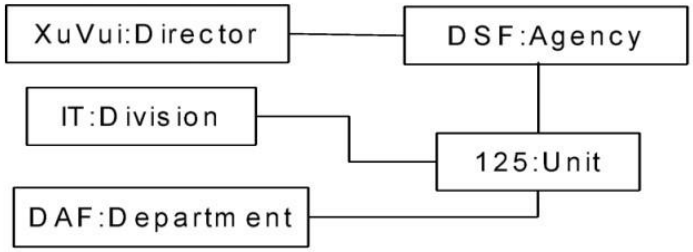


Figure 2.3: Example of a UML Object Diagram

3.5A Use Case Diagram

Use cases are versatile and valuable techniques for describing user requirements. A use case is a high-level description of a major user requirement. It represents the functionality of the system. It is a description of the system's behaviour from a user's

viewpoint and constitutes a complete interaction with the system initiated by a user or another system.

Use case diagrams address the static use case view of a system. The different types of people and/or devices (called actors) that interact with the system are identified along with the functions that they perform or initiate. A Use Case diagram shows a set of use cases and actors (a special kind of class) and their relationships.

These diagrams are especially important in organizing and modelling the behaviors of a system. They are valuable aid during analysis, since developing Use Cases helps to understand requirements. Use Cases and a Conceptual Model are the powerful techniques in requirement analysis. Notations used when representing use case diagrams include:

Basic Use Case Notation

The Actor represents a user of the system, or any external system that interacts with the system. The Usecase represents a piece of functionality that is important to the user. Mostly we see the actor as a human user, but it can also represent a system or other nonhuman artifact. Figure shows the basic notation of a use case diagram.

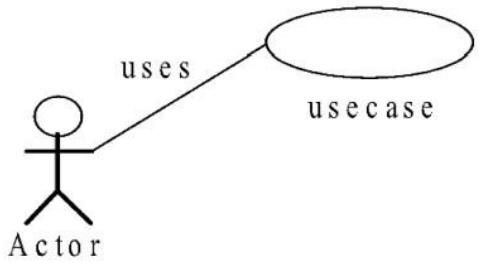


Figure : Basic Notation of a Use Case Diagram

3.6Using the <> Relationship

This feature encourages re-use. If a use-case needs the functionality of another use-case in order to perform its task, it "uses" the 2nduse case. The relationship is drawn as a line with arrowhead pointing to the use case that is being "used" as shown in Figure.

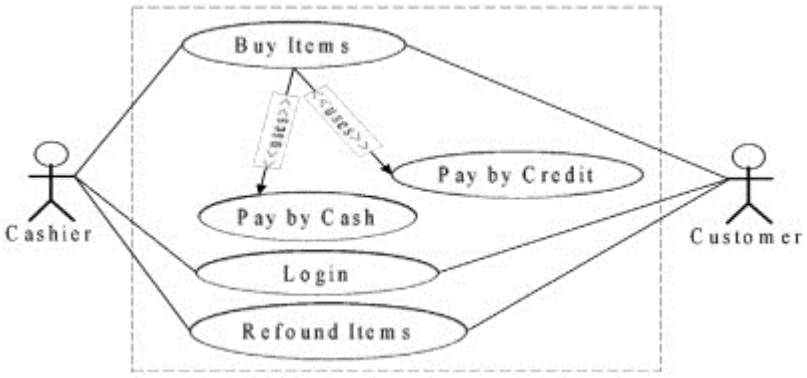


Figure: A < > Relationship in a Use Case Diagram

The "extends" notation extends the functionality of a use case to deal with errors or exceptions. "extends" relationship as shown in Figure 2.6 is being used when there is one use case that is similar to another but does a bit more. The relationship is drawn as a line with arrowhead pointing to the major use case.

The following is the essence of the "extends" relationship:

- Capture the normal simple case first
- For every step in that use case, ask "What could go wrong here" and "How might this work out differently?"
- Plot all variations as extensions of the given use case.

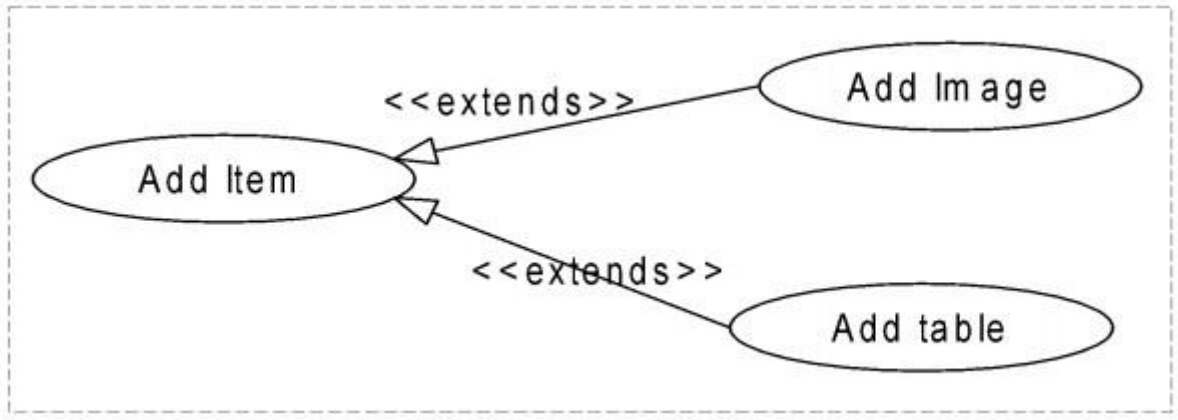


Figure: An < > Relationship in a Use Case Diagram

Using the << include >> relationship

"include" relationship as shown in Figure can be used for making up a big use case from simpler ones.

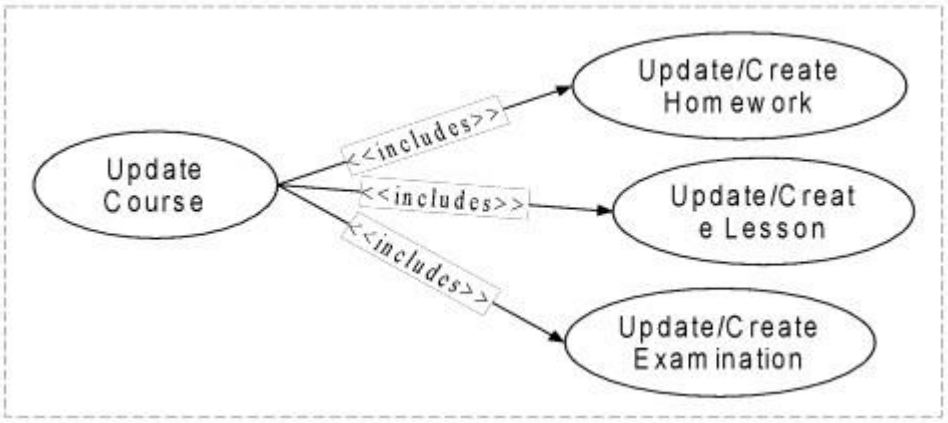


Figure: An < > Relationship in a Use Case Diagram

3.7Sequence and Collaboration Diagrams

When developing Object-Oriented software, anything our software needs to do is going to be achieved by objects collaborating. We can draw a collaboration diagram to describe how we want the objects we built to collaborate.

Both sequence diagrams and collaboration diagrams are kinds of interaction diagrams.

They both describe the flow of messages between objects.

- They are very useful for describing the procedural flow through many objects.
- Interaction diagrams consisting of a set of objects and their relationships, including the messages that may be dispatched among them
- Both are interaction diagrams which address the dynamic view of a system
- They are models that describes how a group of objects collaborate in some behaviour, typically a single use case

3.8Sequence Diagrams

A sequence diagram is an interaction diagram that emphasizes the time-ordering of messages. Sequence diagrams focus on the order in which the messages are sent. They provide a sequential map of message passing between objects over time. The Sequence Diagrams are driven by the Use Cases which are the system

requirements. In this form objects are shown as vertical lines with the messages as horizontal lines between them. The sequence of messages is indicated by reading down the page (read left to right and descending). Sequence Diagrams are about deciding and modelling "how" the system will achieve "what" we described in the Use Case model.

Although there is no fixed recipe for developing Sequence Diagrams, we can follow an approach that will result in a logical sequence diagram. This is as follows:

- Take the Use Case description and turn it into simple pseudo code running down the right-hand side of the State Diagram.
- Guess which classes you think might be involved - based on the content of the Use Case description. Simple noun analysis is as good a way to start as any.
- For each of the steps in your pseudo code decides which of the classes should have the responsibility for doing that task.
- For each of those tasks you may want to go back and decide to break them down into a number of simpler tasks.
- Add in probes that correspond to the "uses (includes)" and "extends" relationships in the Use Case diagram.
- Consider any important errors you might have to handle that perhaps weren't covered in the Use Case model.
- Consider whether anything you have discovered needs to be fed-backed into the Use Case model.

An analysis of the Sequence Diagrams shows the following (Shown in Figure):

i. A Class (Figure (a))

- Participates in a sequence by sending and/or receiving messages
- Is placed at the top of the diagram and is shown using a rectangle with a descriptive name.

ii. A Lifeline (Figure (b))

- Denotes the life of an object during a sequence
- Is a dotted vertical line below each class.

iii. A focus of control (Figure (c))

- Is a long, narrow rectangle placed atop a lifeline
- Denotes when an object is sending or receiving messages

iv. A message (Figure 2.8(d))

- Conveys information from one object to another object
- Is depicted using a horizontal arrow labeled with the message description and applicable parameters.

Example of a "Make a Cup of Tea" sequence diagram generated from its corresponding use case description is as shown in Figure below.

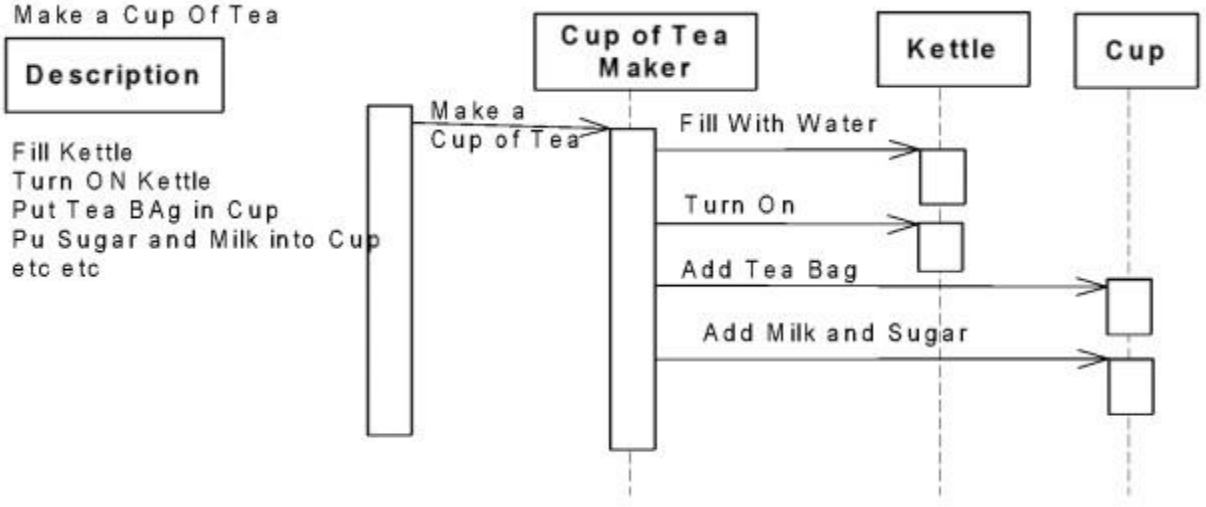


Figure: A Sequence Diagram for "Make a Cup of Tea" Use Case

3.9Collaboration Diagram

A collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages. Collaboration diagrams express both the context of a group of objects and the interaction between these objects. It focuses upon the relationships between the objects. They are very useful for visualizing the way several objects collaborate to get a job done and for comparing a dynamic model with a static model. When creating collaboration diagrams, patterns are used to justify relationships. Patterns are best principles for assigning responsibilities to objects. Figure 2.10 shows an example of a collaboration diagram.

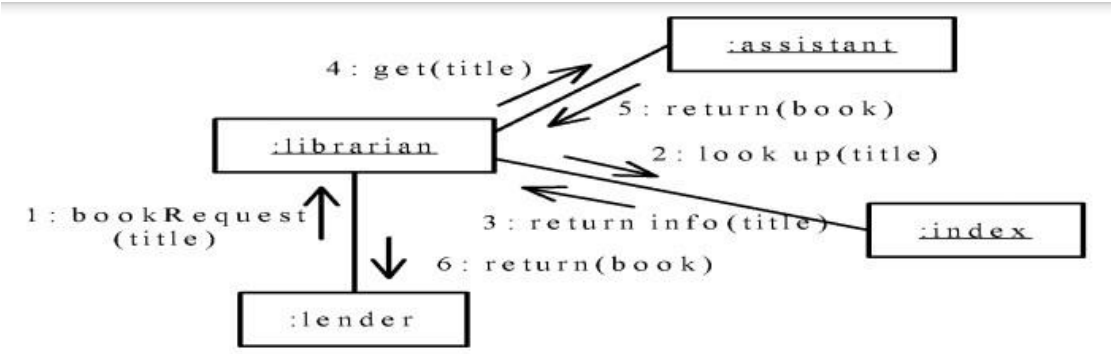


Figure: Example of a Collaboration Diagram

3.10State chart Diagrams

State chart is a diagram that shows all possible object states. Some objects can at any particular time be in a certain state. A State chart or simply a state diagram shows a state machine, consisting of states, transitions, events, and activities. State chart diagrams address the dynamic view of a system. UML State charts are not normally needed. They are needed when an object has a different reaction dependent on its state an example of a State chart diagram is shown in Figure.

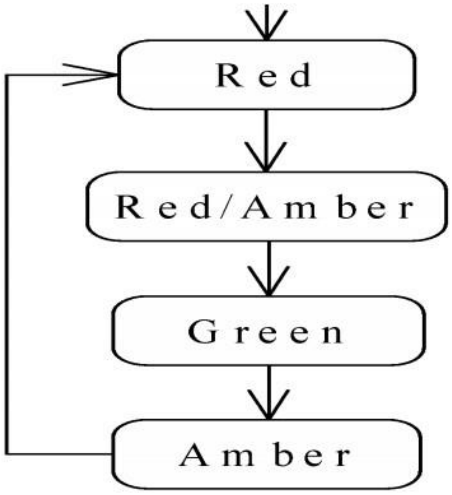


Figure: Example of a State chart Diagram Activity Diagrams

An activity diagram is a special kind of a state chart diagram that shows the flow from activity to activity within a system. They are used to show how different work flows or processes in a system are constructed, how they start, the many decision paths that can be taken from start to finish and where parallel processing may occur during execution. Activity diagrams address the dynamic view of a system. They are

especially important in modelling the function of a system and emphasize the flow of control among objects. An activity diagram as shown in Figure 2.12 generally does not model the exact internal behaviour of a software system (like a Sequence diagram does) but rather it shows the general processes and pathways at a high level.

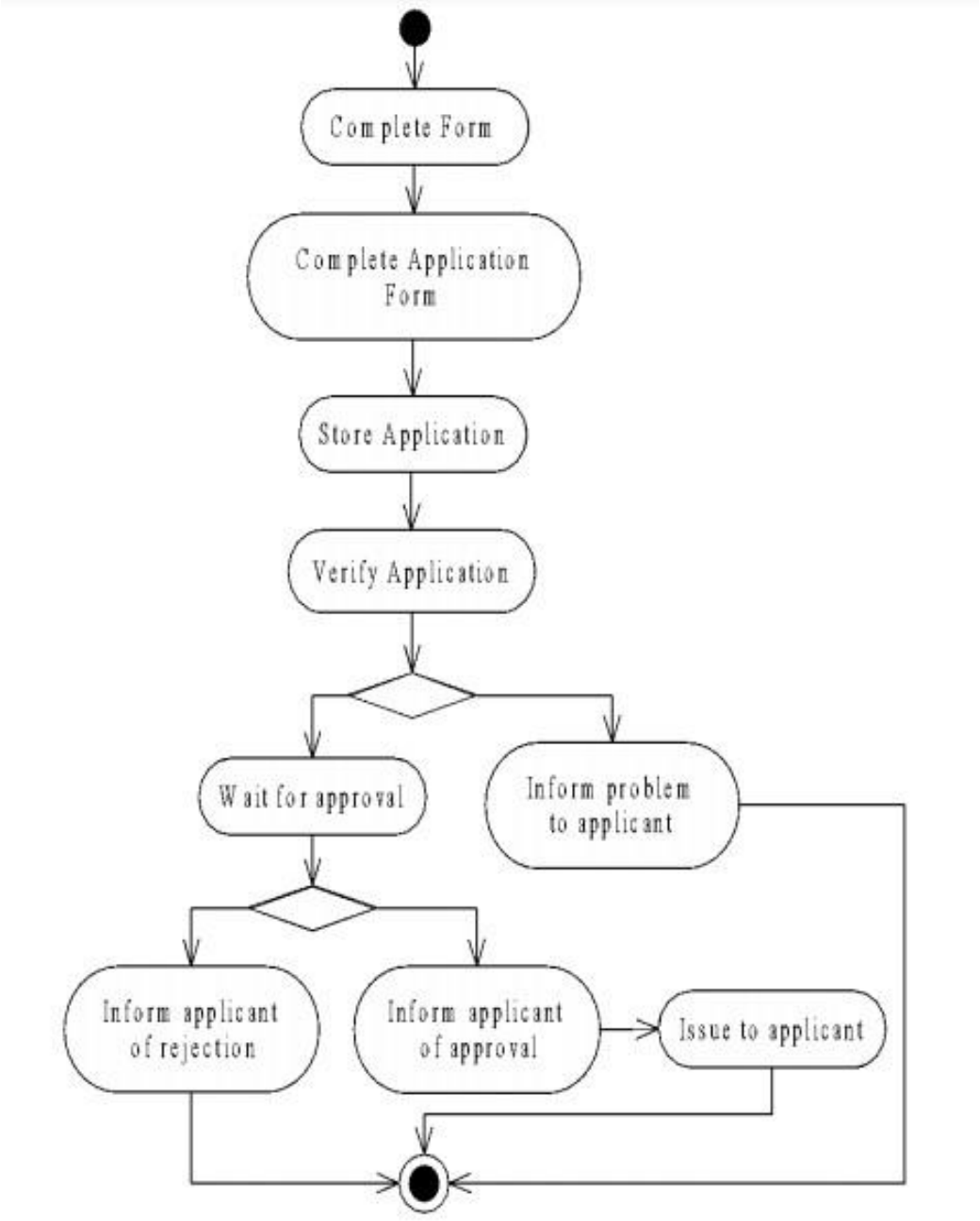


Figure: Example of Activity Diagram

3.11Package Diagrams

Any non-trial system needs to be divided up in smaller, easier to understand "chunks". A package is basically a logical container into which related elements can be placed, "like a folder or directory in an operating system". We can display groups of packages and relationships between then on the UML package diagram.

Packages does not show actually what is inside the package, it provides a very "high-level" view of the system. Some case tools allow the user to double-click on the package icon in order to open-up the package and explore the contents. The common use of a package is to group related classes together, sometimes group related use cases.

Packages can be used to:

- Group large systems into easier to manage subsystems
- Allow parallel iterative development

Package diagram in Figure shows three UML packages representing a "three-tier model"

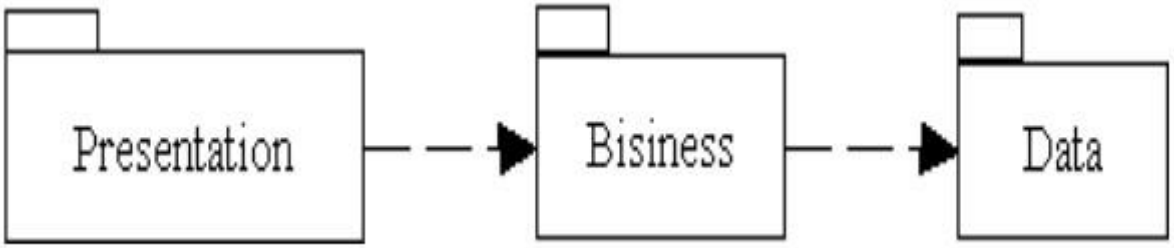


Figure: Example of a Package Diagram

3.12Component Diagrams

A component diagram is similar to the package diagram. It works in the same way as the package diagram, showing the organizations and dependencies among a set of components. Component diagrams address the static implementation view of a system. Component diagrams emphasize the physical software entity e.g. files headers, executables, link-libraries etc, rather than the logical partitioning of the package diagram. It is based heavily on the package diagram, but has added ".dll" to handle I/O, and has added a test harness executable. Not heavily used, but can be

helpful in mapping the physical, real life software code and dependencies between them. Figure 2.14 shows a symbol used for a software component.

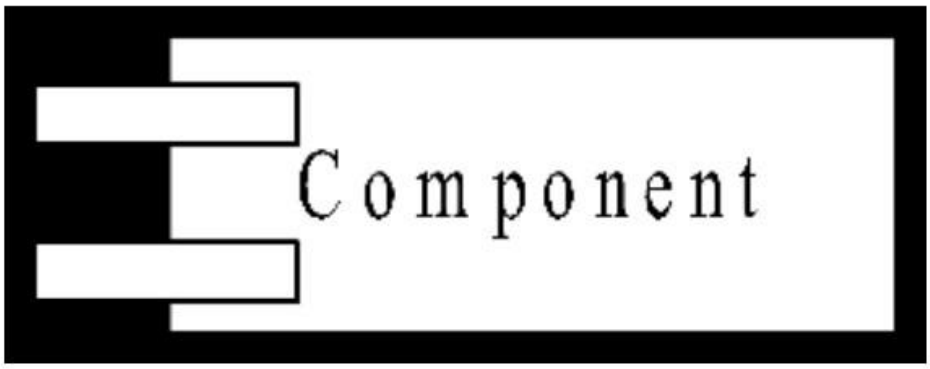


Figure: A Symbol for a Software Component

3.13Deployment Diagrams

Deployment diagram shows the configuration of run-time processing nodes and the components that live on them. Deployment diagrams address the static deployment view of an architecture. They are related to component diagrams in that a node typically encloses one or more components. Figure shows a node symbol used in deployment diagram.

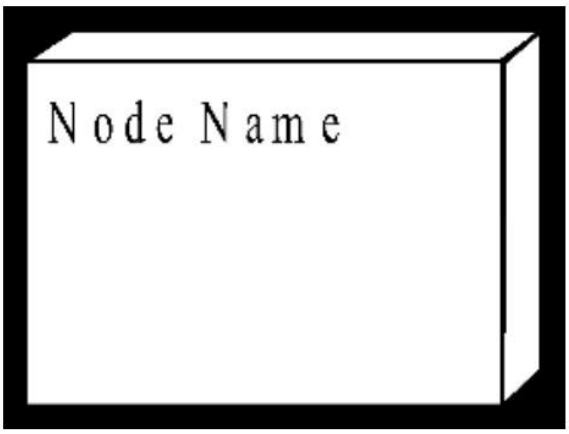


Figure: A Node Symbol for a Deployment Diagram

3.14Let us Sum Up

In this unit, you have learnt about UML diagrams and their use context. Each diagram in UML provides a certain view of the software under development. When creating a diagram, the following can be a guide question for each:

Use Cases – How will our system interact with the outside world?

Class Diagram – What objects do we need? How will they be related?

Collaboration Diagram – How will the objects interact?

Sequence Diagram – How will the objects interact?

Statechart (or state) Diagram – What states should our objects be in?

Component Diagram – How will our software components be related?

Deployment Diagram – How will the software be deployed

3.15Check Your Progress

1 Which UML Diagram are static types?

a)Use case diagrams

b)Object diagrams

c) Class diagrams

d) Both b&c

2 <<extends>>&<<include>> relationship uses in which UML diagrams?

a)Use-case

b)state

c)class

d)activity

3 Which UML diagrams are dynamic behaviour?

a) Use-case

b)Activity

c)Sequence

d)All of the above

4 _____ diagram is a special kind of a state chart diagram that shows the flow from activity to activity within a system.

a)Statechart

b)Activity

c)Use-case

d)Sequence

3.16Assignment

- What is Class diagram?
- What is Object diagram?
- What is Use case diagram?
- What is Sequence diagram?
- What is Collaboration diagram?
- What is Statechart diagram?
- What is Activity diagram?
- What is Component diagram?
- What is Deployment diagram?

3.17References

- https://www.duhoctrungquoc.vn/edu/en/UML
- https://shms.sa/authoring/53030-unified-modeling-language/view
- https://learn.saylor.org/mod/page/view.php?id=32969

3.18 Answer to Check Your Progress

1- d)Both b & c

2- a) Use – Case

3- d) All of these

4- d) Activity

Block-4
UML Interaction Diagram

Unit 1: Collaboration Diagram

1

Unit Structure

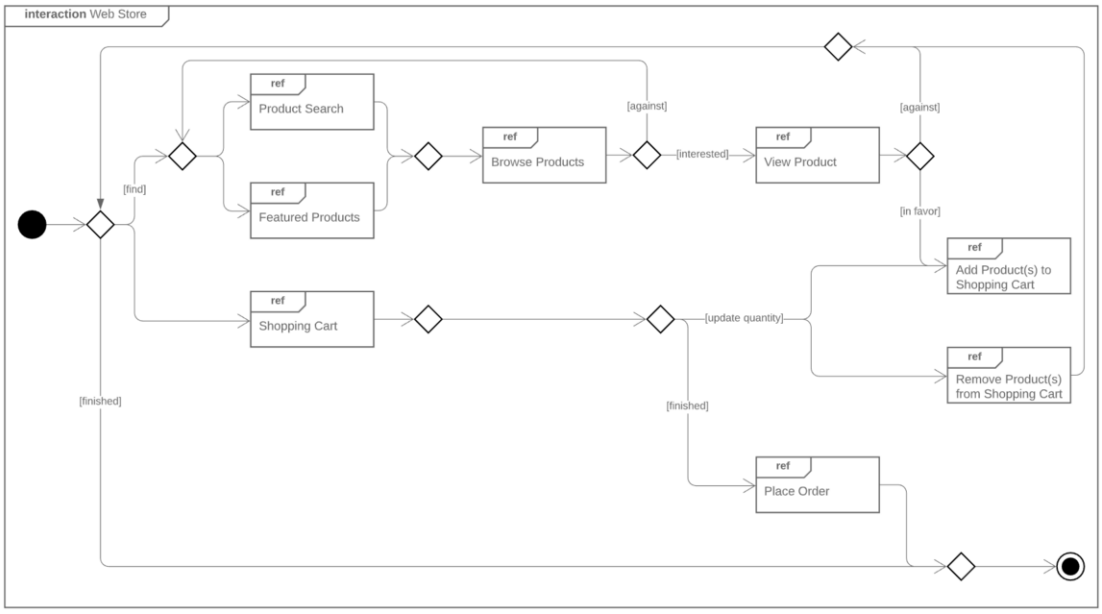
1.1 Learning Objective

After studying this unit student should be able to:

- Define Interaction Diagram
- List benefits of using an interaction diagram
- Types of interaction diagrams in UML
- Define Communication diagram
- Define Sequence diagram
- Define Timing diagram
- Define Interaction overview diagram

1.2Introduction

As its name might suggest, an interaction diagram is a type of UML diagram that's used to capture the interactive behaviour of a system. Interaction diagrams focus on describing the flow of messages within a system, providing context for one or more lifelines within a system. In addition, interaction diagrams can be used to represent the ordered sequences within a system and act as a means of visualizing real-time data via UML.



1.3Benefits of using an interaction diagram

Interaction diagrams can be implemented in a number of scenarios to provide a unique set of information. They can be used to:

- Model a system as a time-ordered sequence of events.
- Reverse- or forward-engineer a system or process.
- Organize the structure of various interactive events.
- Simply convey the behavior of messages and lifelines within a system.
- Identify possible connections between lifeline elements.

1.4Types of interaction diagrams in UML

Interaction diagrams are divided into four main types of diagrams:

- Communication diagram
- Sequence diagram
- Timing diagram
- Interaction overview diagram

Each type of diagram focuses on a different aspect of a system's behavior or structure. Take a look below for more information on the basics of each diagram and how you can benefit from them.

1.5Communication diagram (or collaboration diagram)

A Communication diagram models the interactions between objects or parts in terms of sequenced messages. Communication diagrams represent a combination of information taken from Class, Sequence, and Use Case Diagrams describing both the static structure and dynamic behavior of a system.

However, communication diagrams use the free-form arrangement of objects and links as used in Object diagrams. In order to maintain the ordering of messages in such a free-form diagram, messages are labeled with a chronological number and placed near the link the message is sent over. Reading a communication diagram involves starting at message 1.0, and following the messages from object to object.

Communication diagrams show much of the same information as sequence diagrams, but because of how the information is presented, some of it is easier to find in one diagram than the other. Communication diagrams show which elements each one interacts with better, but sequence diagrams show the order in which the interactions take place more clearly.

A collaboration diagram is a graphical representation which shows the linkage between a number of objects and the links between them. Collaboration diagrams show the messages that are passed from one object to another. Since both collaboration diagrams and object sequence diagrams can express similar constructs, we can opt to use one over the other. We shall mainly discuss and use collaboration diagrams. As an example, collaboration diagram for "makePayment(cashTendered)" operation for the POST system is shown in Figure.

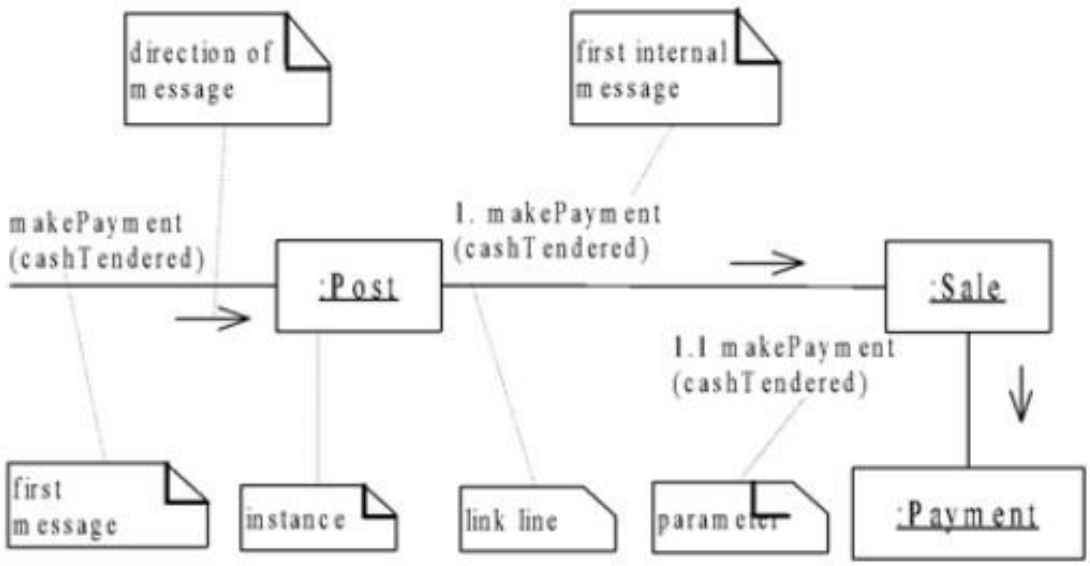


Figure: A Collaboration Diagram for make Payment (cashTendered) Operation

While in comparison with the object sequence diagram for "makePayment(cashTendered)" operation for the POST system we can have as shown in Figure.

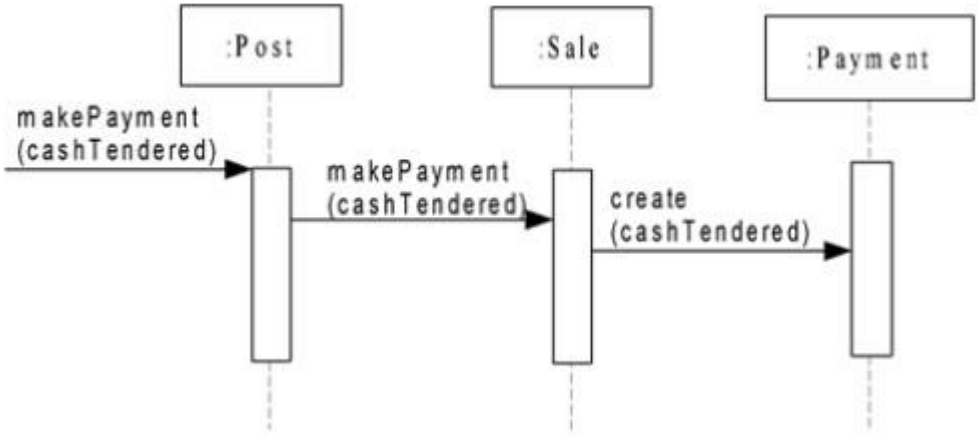


Figure: A Object Sequence Diagram for makePayment(cashTendered) Operation

Both of the two diagrams represent the following pseudo program

- POST accepts a call of it method makePayment() (from an Interface Object);
- POST calls for the method makePayment() of :Sale;
- Sale calls the constructor of class Payment to create a new:Payment

A communication diagram provides the following benefits:

- They emphasize how lifelines connect.
- They focus on elements within a system rather than message flow.
- They provide an added emphasis on organization over timing.

Communication diagrams can also have these possible downsides:

- They can become very complex.
- They make it difficult to explore specific objects within a system.
- They can be time-consuming to create.

1.6UML Notation for Collaboration Diagrams

Collaboration diagrams have basic notations as presented in Figure 4.1. Basic notations include:

- Instances: is the same object but the name is being underlined and it has to always preceded by a colon.
- Links: this is a connection path between two objects to show some form of navigation and visibility instances.

- Message: messages are represented using an arrow on the link line. Messages are numbered to show the sequential order in which the message are sent.
- Parameter: Parameters are shown within parentheses following the message.

A collaboration diagram has more notations which can be used when need comes as follows:

Representing a return value

Some message sent to an object may require a returning message. A return value may be shown by preceding the message with a return value variable name and an assignment operator (':=') as shown in Figure. The standard syntax for messages is:

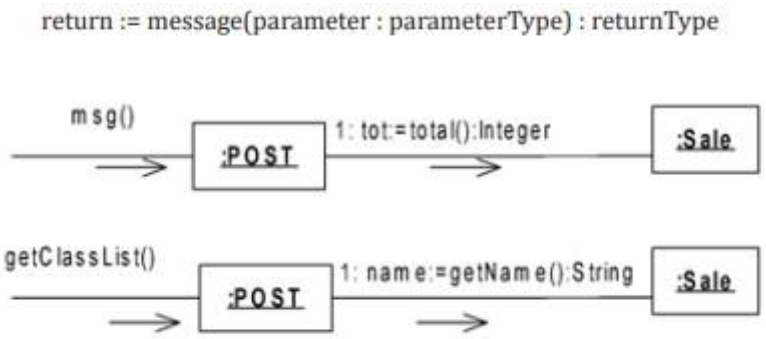


Figure 4.3: Return Values in Collaboration Diagram

Representing iteration

An object may repeatedly send a message to another object a number of times. This is indicated by prefixing the message with a start ('*') as in Figure.

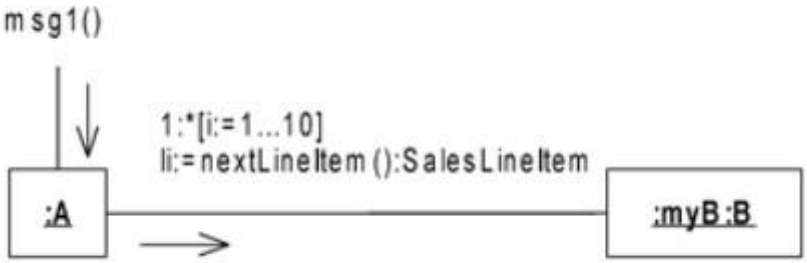


Figure : Representing Iteration in Collaboration Diagram Representing creation ofinstances

The UML creation message is create which is independent of programming languages, shown being sent to the instance being created. Optionally, the newly created instance may include a < > symbol as shown in Figure 4.5. A create

message can optionally take parameters when some attributes of the object to be created need to be set an initial value

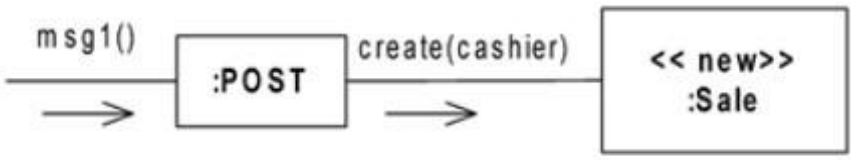


Figure: Creation of an Instance in Collaboration Diagram

Representing message number sequencing

The order of messages is illustrated with sequence numbers, as shown in Figure. The numbering scheme is:

I. The first message is not numbered. Thus, msg1() is unnumbered.

II. The order and nesting of subsequent messages is shown with a legal numbering scheme in which nested messages have appended to them a number. Nesting is denoted by pre-pending the incoming message number to the outgoing message number.

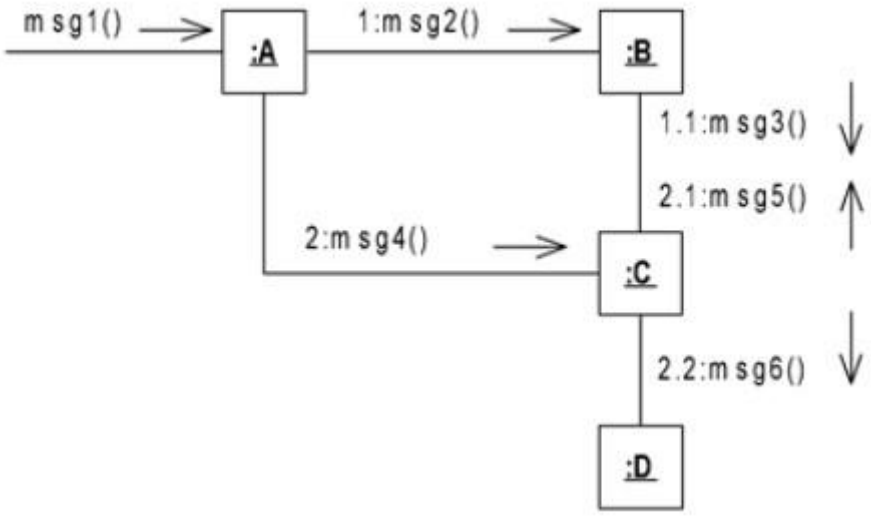


Figure: Message Number Sequencing in Collaboration Diagram

Representing conditional messages

Sometimes, a message may be guarded and can be sent from one object to another only when a condition holds. At a point during the execution of an object, a choice of several messages, guarded by different conditions, will be sent. In a sequential system, an object can send one message at a time and thus these conditions must be mutually exclusive. When we have mutually exclusive conditional messages, it is necessary to modify the sequence expressions with a conditional path letter.

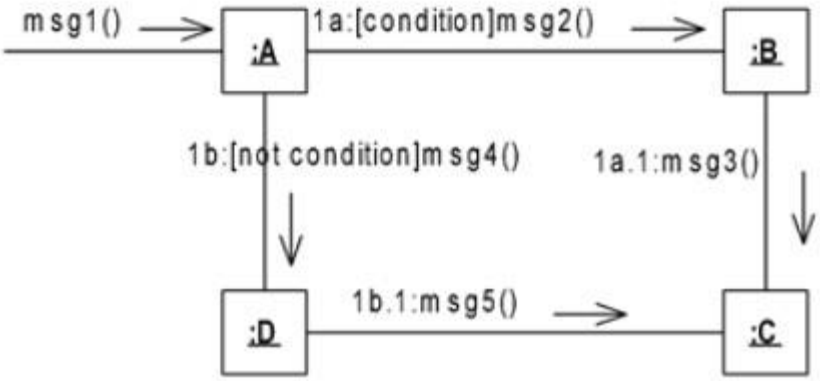


Figure : Conditional Message in Collaboration Diagram

Note that:

- Either 1a or 1b could execute after msg1() , depending on the conditions.
- The sequence number for both is 1, and a and b represent the two paths.
- This can be generalized to any number of mutually exclusive Conditional Messages.

Representing multiobjects

We call a logical set of instances/objects as a multiobject. A multiobject is an instance of a container class each instance of which is a set of instances of a given class (or type). E.g. SetOfSales is a class each instance of which is a set of sales. Each multiobject is usually implemented as a group of instances stored in a container or a collection object. In a collaboration diagram, a multiobject represents a set of objects at the "many" end of an association. In UML, a multiobject is represented as a stack icon as illustrated in Figure.

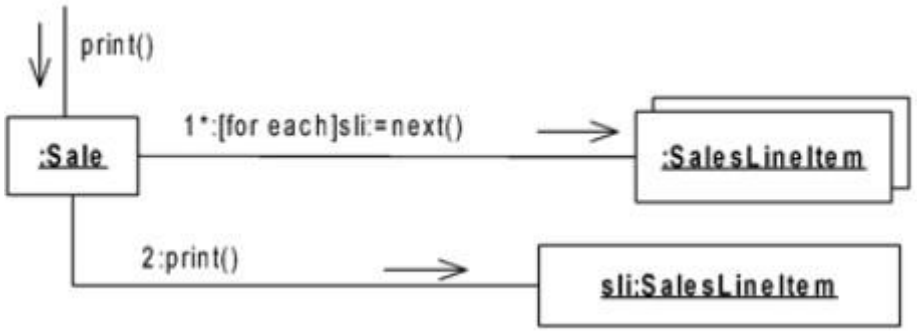


Figure 4.8: Multiobject and Message to Multiobject

1.7When to use a collaboration diagram

Collaboration diagrams should be used when the relationships among objects are crucial to display. A few examples of instances where collaboration diagrams might be helpful include:

Modelling collaborations, mechanisms or the structural organization within a system design.

- Providing an overview of collaborating objects within an object-oriented system.
- Exhibiting many alternative scenarios for the same use case.
- Demonstrating forward and reverse engineering.
- Capturing the passage of information between objects.
- Visualizing the complex logic behind an operation.

However, collaboration diagrams are best suited to the portrayal of simple interactions among relatively small numbers of objects. As the number of objects and messages grows, a collaboration diagram can become difficult to read and use efficiently. Additionally, collaboration diagrams typically exclude descriptive information, such as timing.

Collaboration vs sequence diagrams

In UML, the two types of interaction diagrams are collaboration and sequence diagrams. While both types use similar information, they display them in separate ways. Collaboration diagrams are used to visualize the structural organization of objects and their interactions. Sequence diagrams, on the other hand, focus on the

order of messages that flow between objects. However, in most scenarios, a single figure is not sufficient in describing the behavior of a system and both figures are required.

1.8Let us sum up

Collaboration diagrams are mainly used in object-oriented design phase. They are flexible to add new concepts in two dimensions hence occupy less space. In design phase objects are supposed to be provided with operations to perform. Collaborations are easily used for this purpose with the help of a Pattern which facilitate assigning responsibility to objects.

1.9Check Your Progress

1 A _____diagram models the interactions between objects or parts in terms of sequenced messages.

a) Communication diagram

b)Sequence diagram

c)Timing diagram

d)Interaction overview diagram

2 In Collaboration diagrams whichnotations are used?

a)Instances

b)Links

c)Message

d)Parameter

e)All of the above

1.10 Further Reading

- https://en.wikipedia.org/wiki/Interaction_overview_diagram
- https://www.tutorialspoint.com/uml/uml_interaction_diagram.htm
- https://www.lucidchart.com/pages/uml-interaction-diagram

- https://www.techtarget.com/searchsoftwarequality/definition/collaboration-diagram

1.11. Assignments

- What is Interaction Diagram?
- Write benefits of using an interaction diagram
- List and explain types of Interaction Diagram
- Explain communication Diagram or Collaboration Diagram with example

1.12 Possible Answers to Check Your Progress

1- a) Communication Diagram

2- e) All of these

Unit 2: Sequence Diagram 2

Unit Structure

2.1 Learning Objective

After studying this unit student should be able to:

- Define Sequence Diagrams
- List benefits of Sequence Diagrams
- Use of Sequence Diagram
- Understand notations used to draw sequence diagram

2.2 Introduction

During the requirements analysis phase, the system can be treated as a single "black box", which means that we can look at the system's behavior (what it does) without explaining how it does it. This is called a system sequence diagram.

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

This unit will guide you on how to identifying objects or concepts and make them relate to each other and finally create a conceptual or domain diagram.

2.3Use of Sequence Diagram

- Used to model and visualise the logic behind a sophisticated function, operation or procedure.
- They are also used to show details of UML use case diagrams.
- Used to understand the detailed functionality of current or future systems.
- Visualise how messages and tasks move between objects or components in a system.

2.4Sequence Diagram Notations

Actors – An actor in a UML diagram represents a type of role where it interacts with the system and its objects. It is important to note here that an actor is always outside the scope of the system we aim to model using the UML diagram.

Figure – notation symbol for actor

We use actors to depict various roles including human users and other external subjects. We represent an actor in a UML diagram using a stick person notation. We can have multiple actors in a sequence diagram. For example – Here the user in seat reservation system is shown as an actor where it exists outside the system and is not a part of the system.

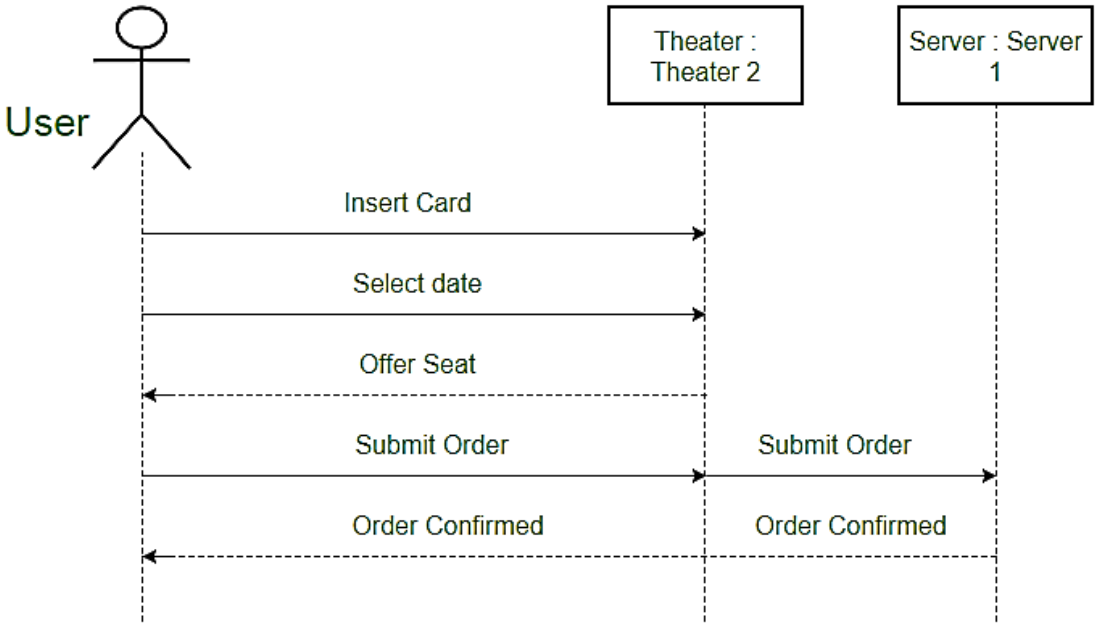
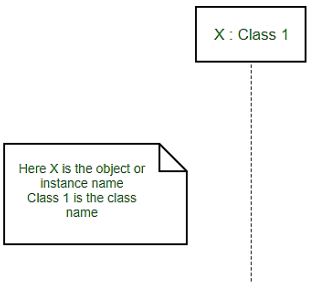


Figure – an actor interacting with a seat reservation system

Lifelines – A lifeline is a named element which depicts an individual participant in a sequence diagram. So basically each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram. The

standard in UML for naming a lifeline follows the following format – Instance Name :



Class Name

Figure – lifeline

We display a lifeline in a rectangle called head with its name and type. The head is located on top of a vertical dashed line (referred to as the stem) as shown above. If we want to model an unnamed instance, we follow the same pattern except now the portion of lifeline's name is left blank. Difference between a lifeline and an actor – A lifeline always portrays an object internal to the system whereas actors are used to depict objects external to the system. The following is an example of a sequence diagram:

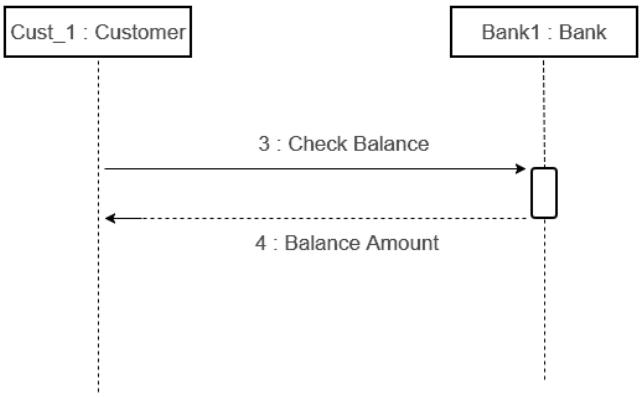


Figure – a sequence diagram

Messages – Communication between objects is depicted using messages. The messages appear in a sequential order on the lifeline. We represent messages using arrows. Lifelines and messages form the core of a sequence diagram. Messages can be broadly classified into the following categories :

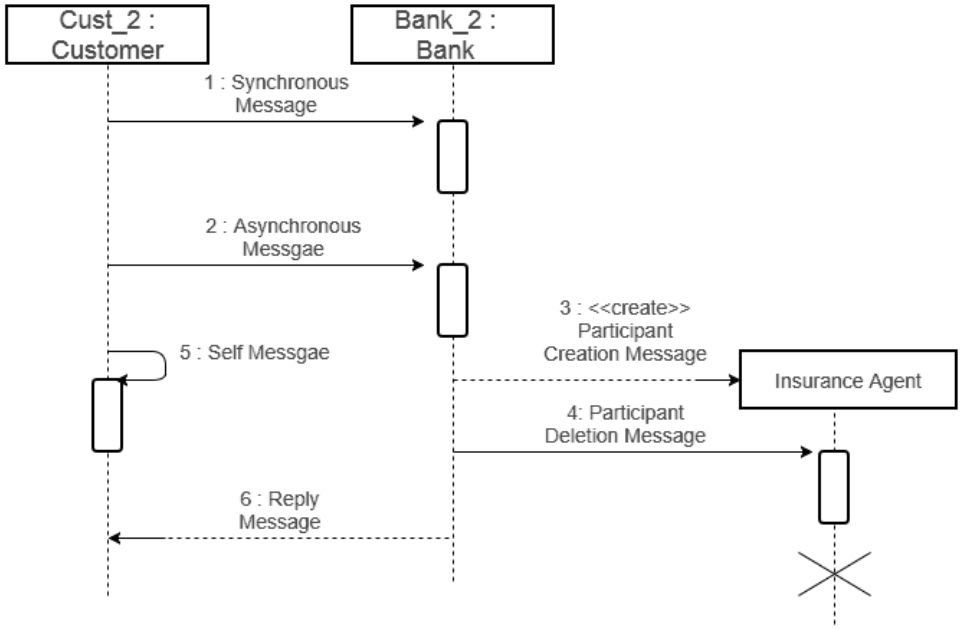


Figure – a sequence diagram with different types of messages

Synchronous messages – A synchronous message waits for a reply before the interaction can move forward. The sender waits until the receiver has completed the processing of the message. The caller continues only when it knows that the receiver has processed the previous message i.e. it receives a reply message. A large number of calls in object oriented programming are synchronous. We use a solid arrow head to represent a synchronous message.

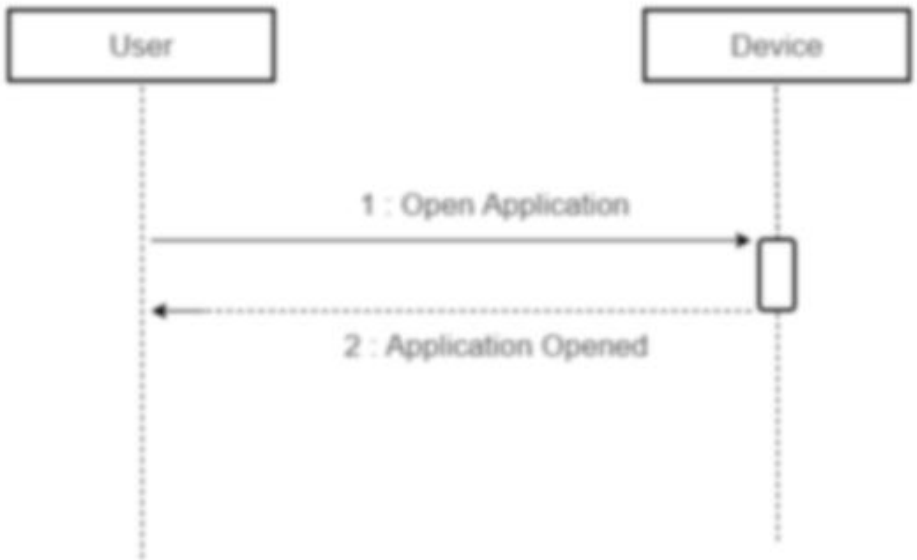
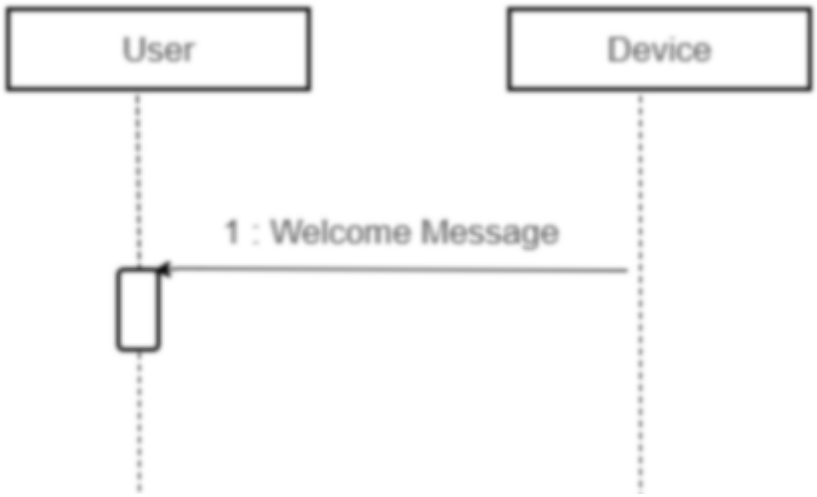


Figure – a sequence diagram using a synchronous message

Asynchronous Messages – An asynchronous message does not wait for a



reply from the receiver. The interaction moves forward irrespective of the receiver processing the previous message or not. We use a lined arrow head to represent an asynchronous message.

Create message – We use a Create message to instantiate a new object in the sequence diagram. There are situations when a particular message call requires the creation of an object. It is represented with a dotted arrow and create word labelled on it to specify that it is the create Message symbol. For example – The creation of a new order on a e-commerce website would require a new object of Order class to be created.

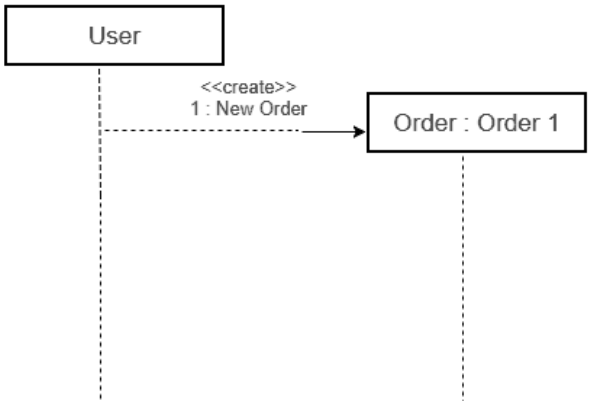


Figure – a situation where create message is used

Delete Message – We use a Delete Message to delete an object. When an object is de-allocated memory or is destroyed within the system we use the Delete Message symbol. It destroys the occurrence of the object in the system.It is represented by an arrow terminating with a x. For example – In the scenario below when the order is received by the user, the object of order class can be destroyed.

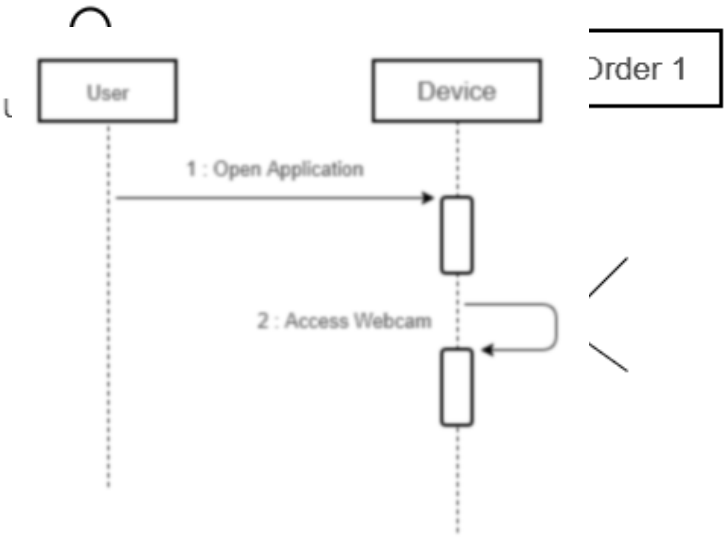


Figure – a scenario where delete message is used

Self-Message – Certain scenarios might arise where the object needs to send a message to itself. Such messages are called Self Messages and are represented with a U-shaped arrow. **Figure –** self messagefor example – Consider a scenario where the device wants to access its webcam. Such a scenario is represented using a self-message.

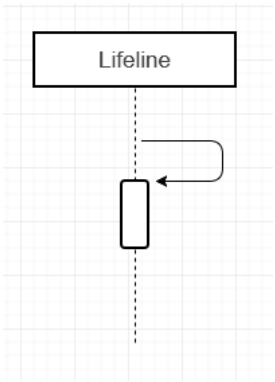


Figure – a scenario where a self-message is used

Reply Message – Reply messages are used to show the message being sent from the receiver to the sender. We represent a return/reply message using an open arrowhead with a dotted line. The interaction moves forward only when a reply message is sent by the receiver.

Figure – reply message

For example – Consider the scenario where the device requests a photo from the user. Here the message which shows the photo being sent is a reply message.

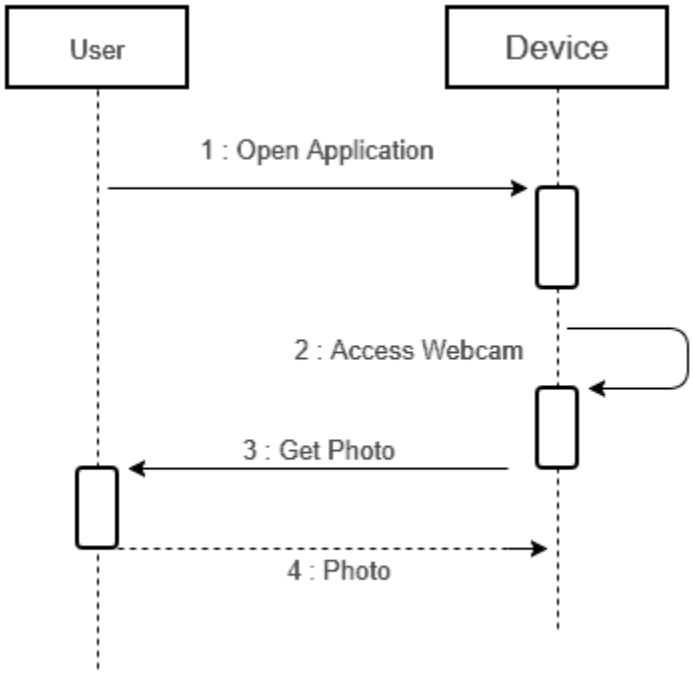


Figure – a scenario where a reply message is used

Found Message – A Found message is used to represent a scenario where an unknown source sends the message. It is represented using an arrow directed towards a lifeline from an end point. For example: Consider the scenario of a hardware failure.

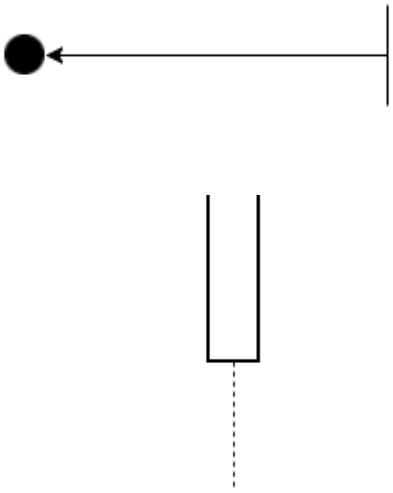


Figure – found message

It can be due to multiple reasons and we are not certain as to what caused the hardware failure.

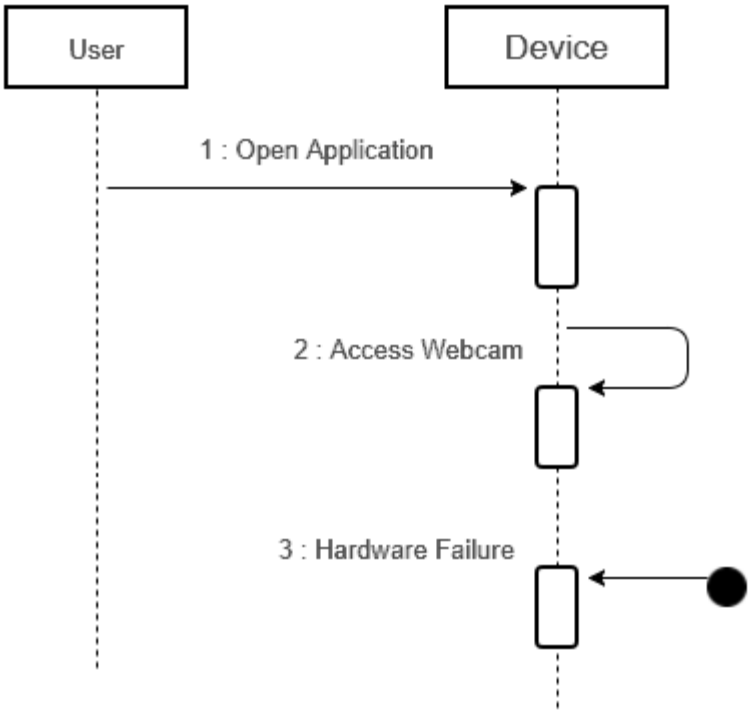


Figure – a scenario where found message is used

Lost Message – A Lost message is used to represent a scenario where the recipient is not known to the system. It is represented using an arrow directed towards an end point from a lifeline. For example: Consider a scenario where a warning is generated. **Figure –** lost message The warning might be generated for

the user or other software/object that the lifeline is interacting with. Since the destination is not known before hand, we use the Lost Message symbol.

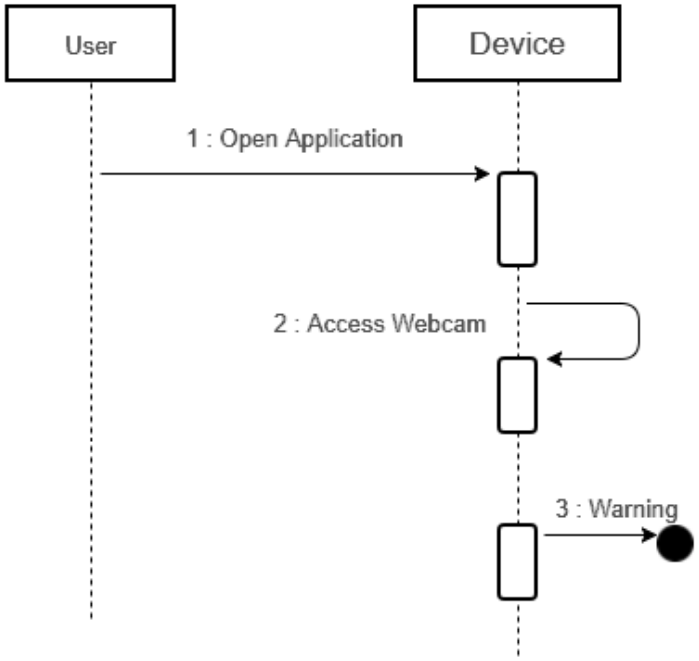


Figure – a scenario where lost message is used

Guards – To model conditions we use guards in UML. They are used when we need to restrict the flow of messages on the pretext of a condition being met. Guards play an important role in letting software developers know the constraints attached to a system or a particular process. For example: In order to be able to withdraw cash, having a balance greater than zero is a condition that must be met as shown below.

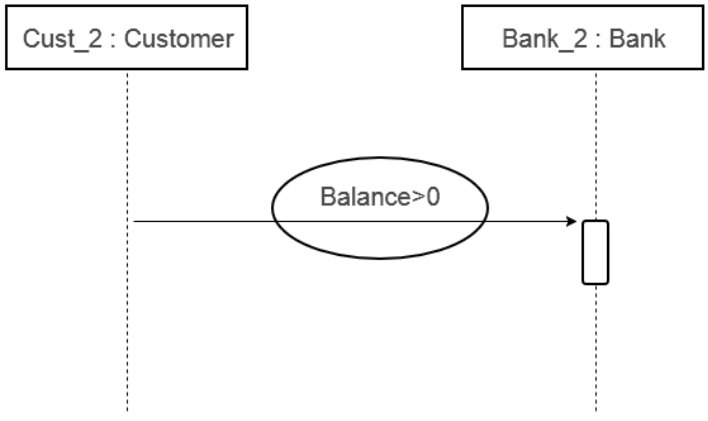


Figure – sequence diagram using a guard

2.5 Example of Sequence Diagram

A sequence diagram for an emotion-based music player –

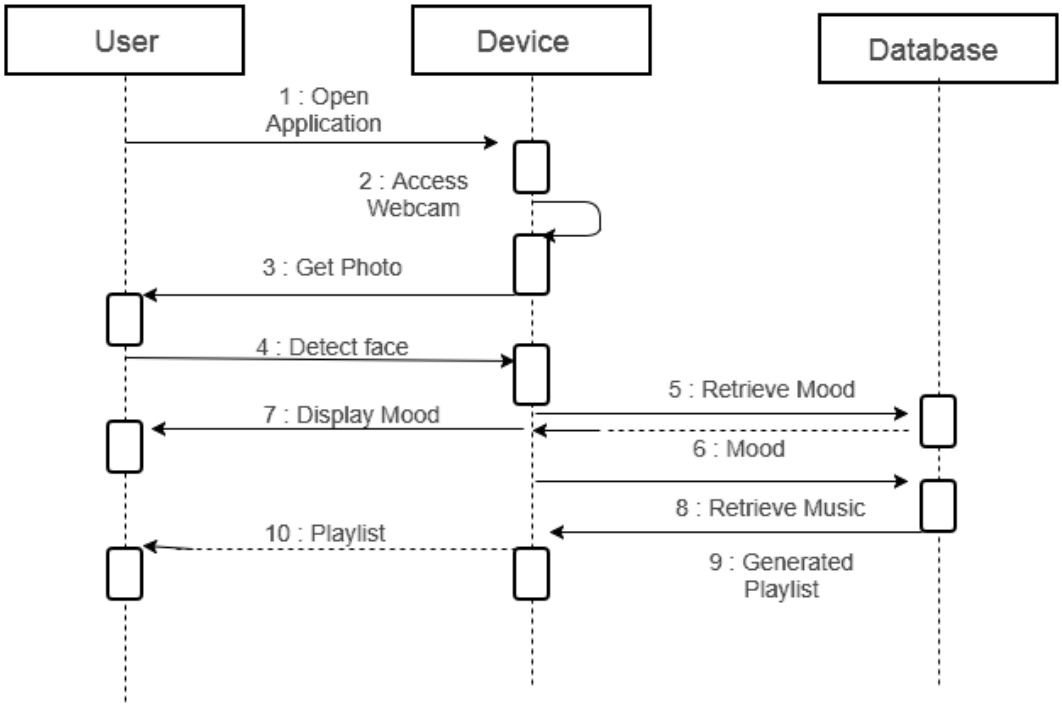


Figure – a sequence diagram for an emotion based music player

The above sequence diagram depicts the sequence diagram for an emotion-based music player:

1. Firstly, the application is opened by the user.
2. The device then gets access to the web cam.
3. The webcam captures the image of the user.
4. The device uses algorithms to detect the face and predict the mood.
5. It then requests database for dictionary of possible moods.
6. The mood is retrieved from the database.
7. The mood is displayed to the user.
8. The music is requested from the database.
9. The playlist is generated and finally shown to the user.

2.6System Sequence Diagrams and Operations

We need to identify the operations that the system needs to perform and in what order the system need to perform these operations to carry out a use case, and the effect of such an operation on the system, i.e. on the objects of the systems.

A use case defines a class of conversations between the actors and the system, and an individual conversation of this class is a realization of the use case. Obviously, there may be many realizations for a use case. A scenario of a use case is a particular instance or realized path through the use case, i.e. a particular realization of the use case. `

This topic is intending to introduce system sequence diagrams which are used to find system events and system operations. Latter we will see how to create a contract of system operations.

2.7System Input Events and System Operations

When an actor interacts with the system for a certain use case, events are being generated to a system. This event is requesting the system to perform some operations in response. Liu, indicate that events generated by actors are very tightly related to operations that the system can perform. This implies that we identify system's operations by identifying events that actors generate.

By definition: A system input event is an external input generated by an actor to a system. A system input event initiates a responding operation while a system operation is an operation that the system executes in response to a system input event.

Some system operations also generate output events to the actors to prompt the next system event that an actor can perform; and A system event is either an input event or an output event.

Therefore, a system input event triggers a system operation, and a system operation response to a system input event.

As an example given by Liu, we can formally define a scenario of a use case as a sequence of system events that occur during a realization of the use case. Consider

the use case of Make Phone Calls for a telephone system, which involves two actors, Caller (initiator) and Callee. The following sequence of events is a scenario of this use case:

Input Events	Output Events
1. Caller lifts a receiver	2. Telephone starts dial tone
3. Caller dials a number	4. Telephone rings to Callee
	5. Telephone is ringing tone to Caller
6. Callee answers the ringing phone	7. Telephone ringing tone stops
	8. Telephone ringing tone stops
	R9. Telephones connectedW6
10. Callee hangs up by returning the telephone receiver	11. connection broken
12. Caller hangs up by returning the telephone receiver	

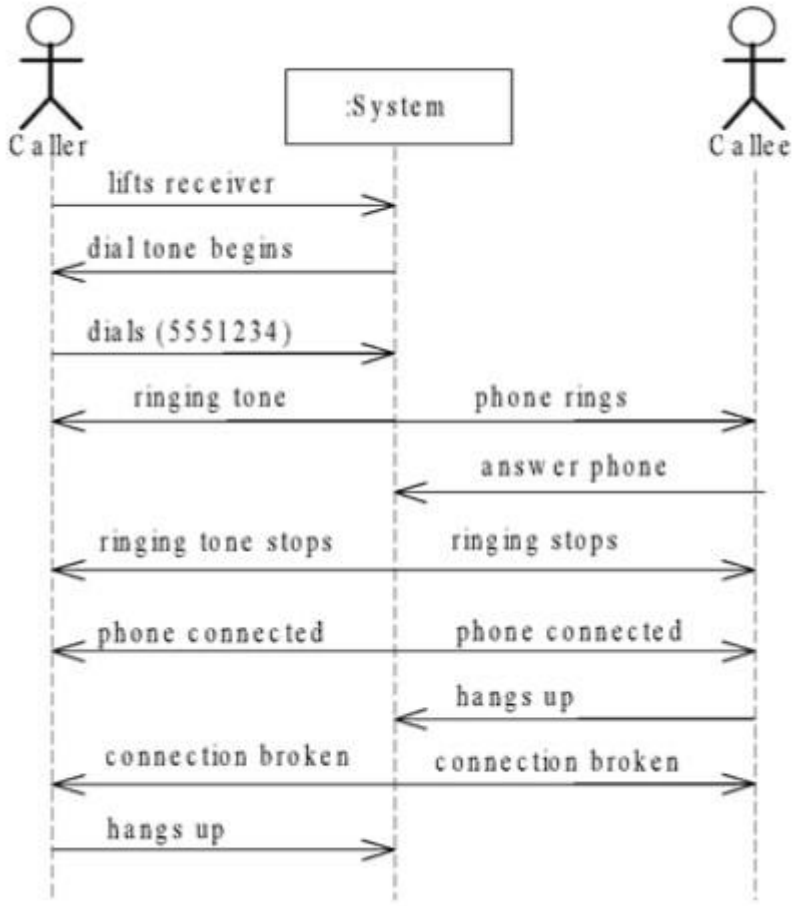


Figure: Telephone system event trace diagram for the "Make Phone Calls"

We can represent the set sequence of events is a scenario using event tracer diagram as shown in Figure:

2.8A System Sequence Diagram

During requirements analysis phase, we define the system by treating it as a single black box so that the behaviour is a description of what a system does, without explaining how it does it. In this way, only system operations that an actor requests of the system will be considered. The interest is to find input events by making use of use cases and their scenario.

For example, the typical course of events in the Make Phone Calls indicates that the caller and callee generate the system input events that can be denoted as liftReceiver, dialPhoneNumber, answersPhone, hangsUp. In general, an event takes parameters.

UML Trace diagram is very useful in identifying the system operations, as in Figure which show, for a particular course of events within a use case, the external actors that interact directly with the system, the system (as a black box), and the system input events that the actors generate. A simplified trace diagram which shows only system input events is called a system sequence diagram. A system sequence diagram (SSD) for the Make Phone Calls use case can be illustrated as in Figure.

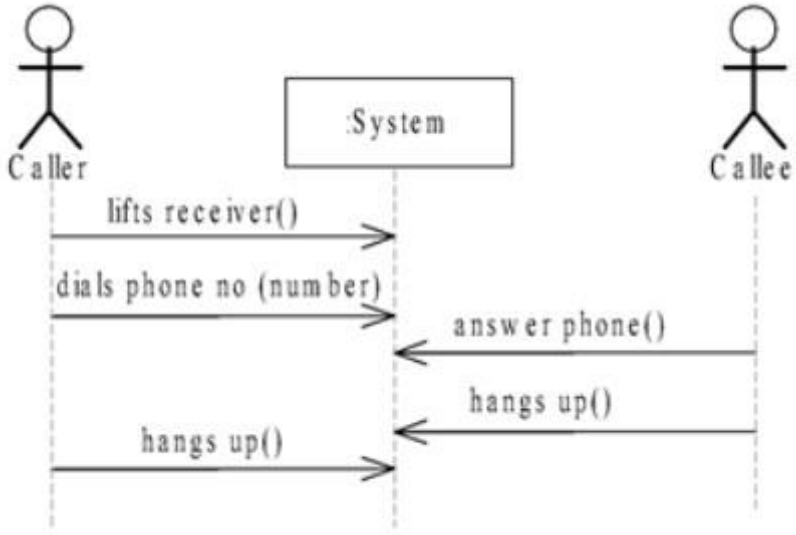
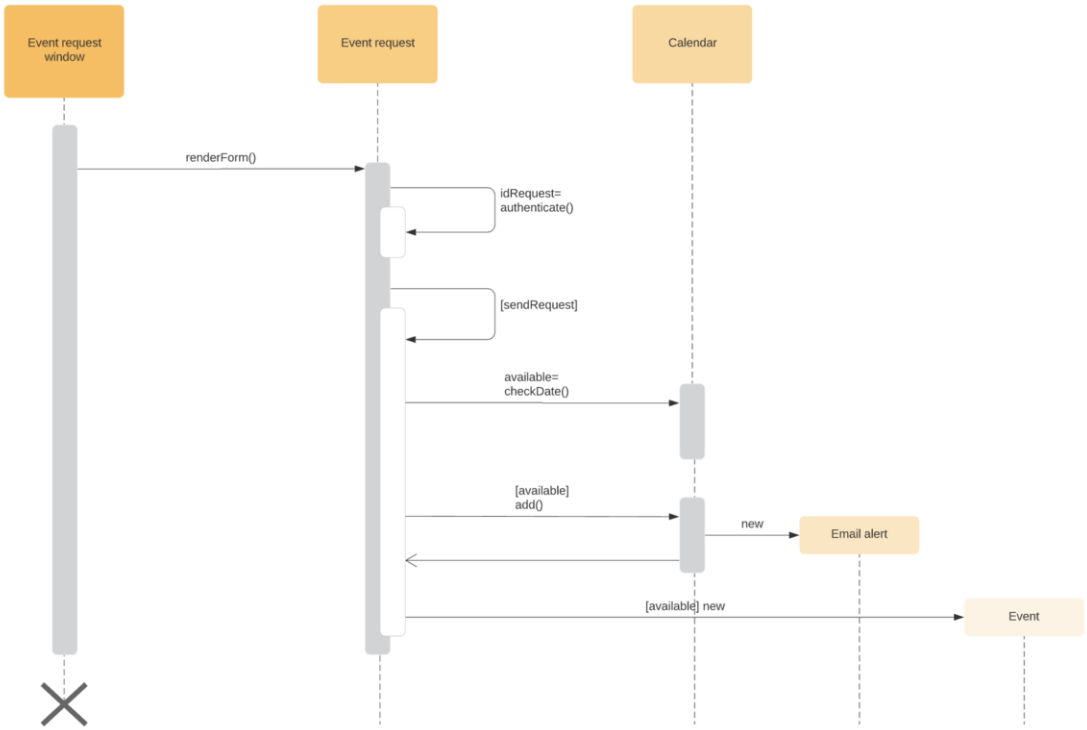


Figure: A System Sequence Diagram for the Make Phone Calls of a Telephone System

System operations are the operations that system needs to perform to carry out a use case and the effects of these operations on the system. In this case, use cases – use case diagram and use case descriptions are inputs of this activity.



If the sequence of events changes, it can cause delays, or the system may crash. It's important to select the notation that matches the particular sequence within your diagram.

2.9Benefits of Sequence Diagram

A sequence diagram provides the following benefits:

- They're easy to maintain and generate.
- They're easy to update according to changes in a system.
- They allow for reverse and forward engineering.
- Sequence diagrams can also have these possible downsides:
- They can become complex, with too many lifelines and varied notations.
- They're easy to produce incorrectly and depend on your sequence being entered correctly.

2.10Let us sum up

In this unit we have learned about what is sequence diagram, Benefits of using sequence diagram, Sequence Diagram Notations, System Sequence Diagrams and Operations and System Input Events and System Operations. How to draw a System Sequence Diagram.

2.11Check Your Progress

1 _____ in a UML diagram represents a type of role where it interacts with the system and its objects.

a) Actor

b) Message

c)Self message

d)Synchronous messages

2 A Synchronous message does not wait for a reply from the receiver. True/False.

3 _____ play an important role in letting software developers know the constraints attached to a system or a particular process.

a) Actor

b) Guards

c)Lost message

d)Found message

4 Certain scenarios might arise where the object needs to send a message to itself. Such messages are called _____Messages and are represented with a U-shaped arrow.

a)Delete message

b)Self message

c)Found message

d)Reply message

2.12 Assignment

- Explain sequence diagram with example
- Explain timing diagram with suitable example
- Define Interaction Diagram
- Explain symbols and terminology used for Interaction diagram

2.13Possible Answer to Check Your Progress

1–A) Actor

2– False

3 –B) Guards

4 – B) Self Message

Unit 3: Timing and Interaction Overview Diagram

3

Unit Structure

3.1 Learning Objective

After studying this unit student should be able to:

- Define Sequence Diagrams
- List benefits of Sequence Diagrams
- List disadvantages of Sequence Diagram
- Use of Sequence Diagram
- Understand notations used to draw sequence diagram

3.2 Introduction

In this unit we will lean about timing diagram and interaction overview diagram with example. We will also discuss advantages and disadvantages of it. We will look at the Interaction diagram symbols and terminology and How to draw an interaction diagram

3.3Timing diagram

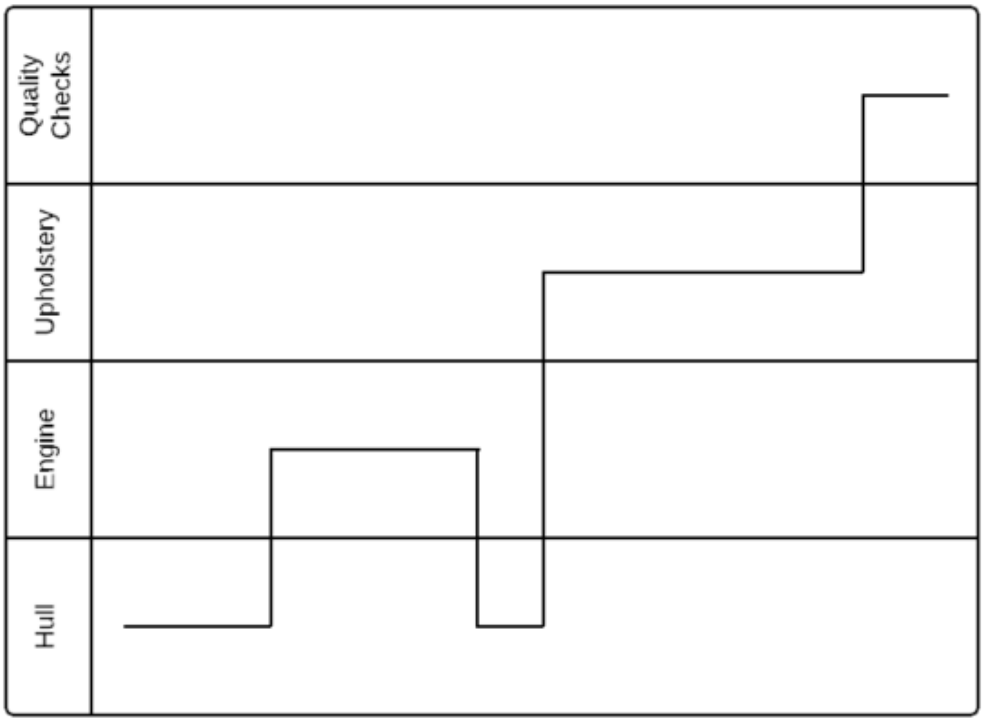
Another diagram option can be to use timing diagrams. These are visuals used to depict the state of a lifeline at any instance in time, denoting the changes in an object from one form to another. Waveforms are used within timing diagrams to visualize the flow within the software program at various instances of time.

A timing diagram offers the following benefits:

- They allow for forward and reverse engineering.
- They can represent the state of an object at an exact instance in time.
- They can keep track of any and all changes within a system.

You should also consider these potential downsides of using a timing diagram:

- They can be difficult to understand.
- They can be hard to maintain over time.



3.4Interaction overview diagram

The interaction overview diagram provides a high-level view of an interaction model. The diagram acts as an overview of the flow of control from interaction to interaction, as well as the flow of activity from diagram to diagram.

A timing diagram offers the following benefits:

- They provide an uncomplicated view of the activity within a model.
- They offer a high degree of navigability between diagrams.
- They allow the use of most annotations within an activity diagram, along with additional elements for added clarity.

Although interaction diagrams are fairly intuitive, they do require branching and interactions to follow certain behaviors, which can be restrictive.

3.5Interaction diagram symbols and terminology

Here are some common terms and symbols you'll come across in an interaction diagram:

Lifeline: A lifeline depicts a single participant in a given interaction, describing how an instance of a specific classifier participates in an interaction.

The attributes of a lifeline are as follows:

- Name: Used to refer to the lifeline in a specific interaction. Lifeline names are optional.
- Type: Names the classifier of which the lifeline represents an instance.
- Selector: Used as a Boolean condition to select a particular instance that satisfies the requirement.

Message: A message is a specific type of communication between two lifelines in an interaction. It can be used to call an operation, create or destroy an instance, or send a signal. As lifelines receive and interact with messages, it creates a focus of control that moves from lifeline to lifeline. This is referred to as a flow of control.

The messages used within an interaction diagram are as follows:

1. **Synchronous message:** The message sender keeps waiting for the receiver to return control from the message execution.
2. **Asynchronous message:** The message sender continues the execution of the next message without waiting for a return from the message receiver.
3. **Return message:** The receiver of a previous message returns the focus of control to the sender.
4. **Object creation:** The message sender creates an instance of a classifier.
5. **Object destruction:** The message sender destroys the created instance.
6. **Found message:** The message sender is outside the scope of interaction.
7. **Lost message:** The message is lost in the interaction and never reaches the destination.

Operator: An operator specifies how the operands will be executed within an operation. In UML, an operator supports operations on data in the form of branching and iterations.

The various operators within an interaction diagram are as follows:

- **Opt (option):** An operand is executed if the condition is true.
- **Alt (alternative):** An operand, whose condition is true, is executed.
- **Loop (loop):** This operator loops an instruction for a specific time period.

- **Break (break):** If the condition is true or false, the loop is broken, and the next instruction is executed.
- **Ref (reference):** This operator refers to another interaction.
- **Par (parallel):** All operands are executed in parallel with one another.

Branching: These are some of the most crucial terms in an interaction diagram. To represent branching within an interaction diagram, guard conditions are added to individual messages. These guard conditions are used to verify if a message can be sent forward or not. Only if a message's guard conditions are true can it be sent forward. A message can have multiple guard conditions, and multiple messages can carry the same guard conditions.

Iteration: An interaction expression consists of an interaction specifier and an iteration clause. An iteration expression can also be used to show iteration expression in an interaction diagram. It involves no specific syntax.

Parallel iteration specifiers are used to show that messages are being sent in parallel. This is denoted by *//. In UML, iteration is achieved by using a loop operator.

State invariants and constraints: In an interaction diagram, a state is a situation or condition during a lifetime of an object—it satisfies a constraint, performs various operations, and waits for an event. A state can be changed when an instance or lifeline receives a message, though not all messages cause a change of state.

3.6How to draw an interaction diagram

Follow these steps to ensure that you have the necessary information to begin creating your interaction diagram:

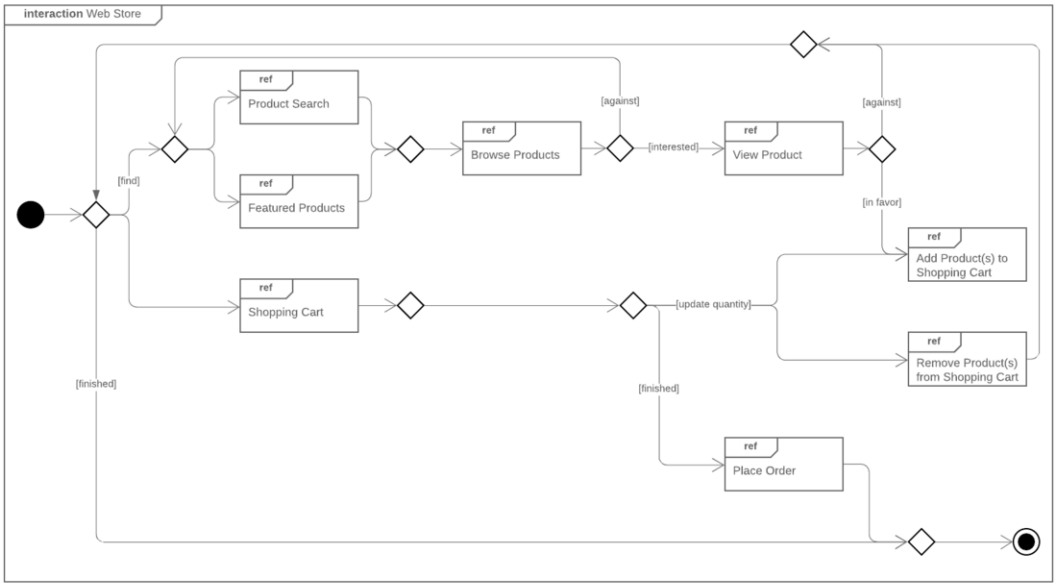
1. Determine the scenario your interaction diagram will represent.
2. Identify the lifelines that will be involved in your interaction.
3. Explore each lifeline to identify potential connections and relationships, and then categorize your lifelines.
4. Identify the sequence of message flows within your interaction.
5. Identify the various types of messages within your interaction, as well as an ordered sequence of events and time constraints of each object.

6. Within your UML diagram software, select the appropriate shapes, nodes, and lines to create the flow of activity within your diagram. Or, select a template and customize your diagram from there.
7. Label each shape, node, and line to represent each event, interaction, and message within your system.

Note that your diagram's design will vary when creating a sequence, timing, or communication diagram, as each diagram focuses on a different aspect of a system's behavior.

3.7Interaction diagram example

Here is a simple example of an interaction diagram template that can be used to model the interactions among the various elements of a basic web app. You can modify this template to visualize the control flow of a system and describe the interactions amongst objects within it.



3.8Let us sum up

In this unit we have learned about what is interaction diagram, Benefits of using an interaction diagram, Types of Interaction Diagram, Communication Diagram or Collaboration Diagram, Sequence diagram, Timing Diagram and Interaction Diagram.

We have also discussed Interaction diagram symbols and terminology, How to draw an interaction diagram, Interaction Diagram Example.

3.9Check Your Progress

1 A lifeline depicts a single participant in a given interaction, describing how an instance of a specific classifier participates in an interaction. True/False

2 Which are the attributes of lifeline?

a) Name

b) Type

c)Selector

d)All of the above

3 An operand is executed if the condition is true. In which operators within an interaction diagram?

a)Opt(option)

b)Alt(alternative)

c)Break(break)

d)Loop(loop)

4 A timing diagram offers allow for forward and reverse engineering. True/False

3.10 Assignment

- Explain sequence diagram with example
- Explain timing diagram with suitable example
- Define Interaction Diagram
- Explain symbols and terminology used for Interaction diagram

3.11 Further Reading

- https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/

3.12Possible Answers to Check Your Progress

1 – True

2 –d) All of these

3 – a) Opt(option)

4 – True