Unit -II – REQUIREMENT ANALYSIS AND SPECIFICATION

Requirement analysis and specification – Requirements gathering and analysis – SoftwareRequirement Specification – Formal system specification – Finite State Machines – Petri nets – Object modelling using UML – Use case Model–Class diagrams – Interaction diagrams–Activitydiagrams–State chart diagrams –Functional modelling–Data Flow Diagram

2.1 Requirement analysis and specification

The Requirement Analysis and Specification phase starts after the feasibility study stage iscomplete and the project is financially viable and technically feasible. This phase ends whenthe requirements specification document has been developed and reviewed. It is usually called the Software Requirements Specification (SRS) Document.These activities are usually carried out by a few experienced members of the development teamand it normally requires them to spend some time at the customer site. The engineers who gather and analyse customer requirements and then write the requirements specification document are known as system analysts in the software industry. System analysts collect data about the product to be developed and analyse the collected data to conceptualize what exactly needs to be done. After understanding the precise user requirements, the analysts analyse the requirements to weed out inconsistencies, anomalies and incompleteness. We can conceptually divide the requirements gathering and analysis activity into two separate tasks:

- Requirements gathering
- Requirements analysis

Requirements Gathering:

It is also known as Requirements Elicitation.

The primary objective of the requirements-gathering task is to collect the requirements from the stakeholders. A stakeholder is a source of the requirements and is usually a person or a group of persons who either directly or indirectly are concerned with the software.

1. Studying existing documentation:

The analyst usually studies all the available documents regarding the system to be developed before visiting the customer site. Customers usually provide a statement of purpose (SoP) document to the developers. Typically, these documents might discuss issues such as the context in which the software is required.

2. Interview:

Typically, there are many different categories of users of the software. Each category of users typically requires a different set of features from the software. Therefore, it is important for the analyst to first identify the different categories of users and then determine the requirements of each.

3. Task analysis:

The users usually have a black-box view of software and consider the software as something that provides a set of services. A service supported by the software is also called a task . We can therefore say that the software performs various tasks for the users.

In this context, the analyst tries to identify and understand the different tasks to be performed by the software. For each identified task, the analyst tries to formulate the different stepsnecessary to realise the required functionality in consultation with the users.

4. Scenario analysis:

A task can have many scenarios of operation. The different scenarios of a task may take place when the task is invoked under different situations. For different types of scenarios of a task, the behaviour of the software can be different.

5. Form analysis:

Form analysis is an important and effective requirement-gathering activity that is undertaken by the analyst when the project involves automating an existing manual system. During the operation of a manual system, normally several forms are required to be filled up by the stakeholders, and in turn, they receive several notifications. In form analysis, the existing forms and the formats of the notifications produced are analysed to determine the data input to the system and the data that are output from the system.

2.1.2 Requirement Analysis and Specification:

After requirements gathering is complete, the analyst analyses the gathered requirements to form a clear understanding of the exact customer requirements and to weed out any problems in the gathered requirements. It is natural to expect the data collected from various stakeholders to contain several contradictions, ambiguities, and incompleteness. The main purpose of the requirements analysis activity is to analyse the gathered requirements to remove all ambiguities, incompleteness, and inconsistencies from the gathered customer requirements and to obtain a clear understanding of the software to be developed. During requirements analysis, the analyst needs to identify and resolve three main types of problems in the requirements:

- Anomaly
- Inconsistency
- Incompleteness

Anomaly:

It is an anomaly is an ambiguity in a requirement. When a requirement is anomalous, several interpretations of that requirement are possible. Any anomaly in any of the requirements can lead to the development of an incorrect system, since an anomalous requirement can be interpreted in several ways during development.

Inconsistency:

Two requirements are said to be inconsistent if one of the requirementscontradicts the other.

Incompleteness:

An incomplete set of requirements is one in which some requirements have been overlooked. The lack of these features would be felt by the customer much later, possibly while using the software. Often, incompleteness is caused by the inability of the customer to visualise the system that is to be developed and to anticipate all the features that would be required.

2.2 Software Requirement Specification

The production of the requirements stage of the software development process is SoftwareRe quirements Specifications (SRS) (also called a requirements document). This report lays afou ndation for software engineering activities and is constructing when entire requirements areel icited and analyzed. SRS is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements. Also, it comprises user requirements for a system as well as detailed specifications of the system requirements. The SRS is a specification for a specific software product, program, or set of applications
that perform particular functions in a specific environment. It serves several goals depending on who is writing it. First, the SRS could be written by the client of a system. Second, the SRS could be
written by a developer of the system. The two methods create entirely various situations and e stablishdifferent purposes for the document altogether. The first case, SRS, is used to define the needs and expectation of the users. The second case, SRS, is written for various purposes and serves as a contract document between customer and developer.

Software Requirement Specification

The production of the requirements stage of the software development process is SoftwareRe quirements Specifications (SRS) (also called a requirements document). This report lays afou ndation for software engineering activities and is constructing when entire requirements areel icited and analyzed. SRS is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements. Also, it comprises user requirements for a system as well as detailed specifications of the system requirements. The SRS is a specification for a specific software product, program, or set of applications
that perform particular functions in a specific environment. It serves several goals depending on who is writing it. First, the SRS could be written by the client of a system. Second, the SRS could be
written by a developer of the system. The two methods create entirely various situations and e stablishdifferent purposes for the document altogether. The first case, SRS, is used to define the needs and expectation of the users. The second case, SRS, is written for various purposes and serves as a contract document between customer and developer.

Characteristics of good SRS

Following are the features of a good SRS document:1. Correctness:

User review is used to provide the accuracy of requirements stated in the SRS.SRS is said to be perfect if it covers all the needs that are truly expected from the system.

2. Completeness:

The SRS is complete if, and only if, it includes the following elements:

(1). All essential requirements, whether relating to functionality, performance, design,constrai nts, attributes, or external interfaces.

(2).Definition of their responses of the software to all realizable classes of input data in allavailable categories of situations.

(3).Full labels and references to all figures, tables, and diagrams in the SRS and definitions of all terms and units of measure.

3. Consistency:

The SRS is consistent if, and only if, no subset of individual requirementsdescribed in its conflict. There are three types of possible conflict in the SRS:

(1).The specified characteristics of real-world objects may conflicts. For example,

(a) The format of an output report may be described in one requirement as tabular but inanother as textual.

(b) One condition may state that all lights shall be green while another states that all lights shall be blue.

(2).There may be a reasonable or temporal conflict between the two specified actions. For example,(a) One requirement may determine that the program will add two inputs, and another maydetermine that the program will multiply them.(b) One condition may state that "A" must always follow "B," while other requires that "A andB" co-occurs.

(3).Two or more requirements may define the same real-world object but use different termsfor that object. For example, a program's request for user input may be called a "prompt" in one
requirement's and a "cue" in another. The use of standard terminology and descriptions prom otes consistency.

4. Unambiguousness:

SRS is unambiguous when every fixed requirement has only oneinterpretation. This suggests that each element is uniquely interpreted. In case there is a method used with multiple definitions, the requirements report should determine the implications in the SRS so that it is clear and simple to understand.

5. Ranking for importance and stability:

The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement. Typically, all requirements are not equally important. Some prerequisites may be essential, especially for life-critical applications, while others may be desirable. Each element should be identified to make these differences clear and explicit. Another way to rank requirements is to distinguish classes of items as essential, conditional, and optional

6. Modifiability:

SRS should be made as modifiable as likely and should be capable of quickly obtain changes to the system to some extent. Modifications should be perfectly indexed and cross-referenced.

7. Verifiability:

SRS is correct when the specified requirements can be verified with a cost-effective system to check whether the final software meets those requirements. Therequirements are verified with the help of reviews.

8. Traceability:

The SRS is traceable if the origin of each of the requirements is clear and if itfacilitates the referencing of each condition in future development or enhancementdocumentation.

There are two types of Traceability:

1. Backward Traceability:

This depends upon each requirement explicitly referencing itssource in earlier documents.

2. Forward Traceability:

This depends upon each element in the SRS having a unique name or reference number. The forward traceability of the SRS is especially crucial when the software product enters the operation and maintenance phase. As code and design document is modified, it is necessary to be able to ascertain the complete set of requirements that may be concerned by thosemodifications.

9. Design Independence:

There should be an option to select from multiple design alternativesfor the final system. More specifically, the SRS should not contain any implementation details.

10. Testability:

An SRS should be written in such a method that it is simple to generate testcases and test plans from the report.

11. Understandable by the customer:

An end user may be an expert in his/her explicit domain but might not be trained in computer science. Hence, the purpose of formal notations and symbols should be avoided too as much extent as possible. The language should be kept simple and clear.

12. The right level of abstraction:

If the SRS is written for the requirements stage, the details should be explained explicitly. Whereas, for a feasibility study, fewer analysis can be used. Hence, the level of abstraction modifies according to the objective of the SRS.

Properties of a good SRS document

The essential properties of a good SRS document are the following:

Concise:

The SRS report should be concise and at the same time, unambiguous, consistent, and complete. Verbose and irrelevant descriptions decrease readability and also increase error possibilities.

Structured:

It should be well-structured. A well-structured document is simple to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the user requirements. Often, user requirements evolve over a period of time. Therefore, to make the modifications to the SRS document easy, it is vital to make the report well-structured.

Black-box view:

It should only define what the system should do and refrain from stating howto do these. This means that the SRS document should define the external behavior of thesystem and not discuss the implementation issues. The SRS report should view the system to be developed as a black box and should define the externally visible behavior of the system .For this reason, the SRS report is also known as the black-box specification of a system.

Conceptual integrity:

It should show conceptual integrity so that the reader can merelyunderstand it. Response to undesired events: It should characterize acceptable responses tounwanted events. These are called system response to exceptional conditions.

Verifiable:

All requirements of the system, as documented in the SRS document, should becorrect. This means that it should be possible to decide whether or not requirements have beenmet in an implementation.

Software Requirement Specification (SRS) Format

In order to form agood SRS, here you will see some points which can be used and should beconsidered to form a structure of good SRS. These are as follows :

1. Introduction

(i) Purpose of this document

(ii)Scope of this document

(iii)Overview

2. General description

3. Functional Requirements

4. Interface Requirements

5. Performance Requirements

6. Design Constraints

7. Non-Functional Attributes

8. Preliminary Schedule and Budget

9. Appendices

Software Requirement Specification (SRS) Format as name suggests, is complete specification and description of requirements of software that needs to be fulfilled for successful development of software system. These requirements can befunc tional as well as non-functional depending upon type of requirement. The interaction between different customers and contractor is done because its necessary to fully understand needs of customers. Depending upon information gathered after interaction, SRS is developed which describes requirements of software that may include changes and modifications that is needed to be done to increase quality of product and to satisfy customer's demand.

Document title

Author(s)

Affiliation

Address

Date

Document version

Introduction :

(i) Purpose of this Document –

At first, main aim of why this document is necessary and what's purpose of document is explained and described.

(ii) Scope of this document –

In this, overall working and main objective of document and what value it will provide to customer is described and explained. It also includes a description of development cost and time required.

(iii) Overview –

In this, description of product is explained. It's simply summary or overall review of product.

2.General description :

In this, general functions of product which includes objective of user, a user characteristic, features, benefits, about why its importance is mentioned. It also describes features of user community.

3.Functional Requirements :

In this, possible outcome of software system which includes effects due to operation of program is fully explained. All functional requirements which may include calculations, data processing, etc. are placed in a ranked order.

4.Interface Requirements :

In this, software interfaces which mean how software program communicates with each other or users either in form of any language, code, or message are fully described and explained. Examples can be shared memory, data streams, etc.

5.Performance Requirements :

In this, how a software system performs desired functions under specific condition is explained. It also explains required time, required memory, maximum error rate, etc.

6.Design Constraints :

In this, constraints which simply means limitation or restriction arespecified and explained for design team. Examples may include use of a particular algorith m, hardware and software limitations, etc.

7.Non-Functional Attributes :

In this, non-functional attributes are explained that are required by software system for better performance. An example may include Security, Portability, Reliability, Reusability, Application compatibility, Data integrity, Scalability capacity, etc.

8.Preliminary Schedule and Budget :

In this, initial version and budget of project plan are explained which include overall time duration required and overall cost required for development of project.

9.Appendices :

In this, additional information like references from where information is gathered, definitions of some specific terms, acronyms, abbreviations, etc. are given and explained.

Formal system specification

Formal system specification in software engineering refers to the process of precisely describingthe behavior, properties, and requirements of a software system using formal meth ods andmathematical notations. It aims to eliminate ambiguity, clarify system requirements, and enable formal analysis and verification of system properties.

Here are some key aspects of formal system specification:

Formal Methods: Formal methods are mathematical techniques used to specify, model, andreason about software systems. They provide a rigorous and precise approach to system specification, ensuring clarity and unambiguous representation. Common formal methods used for system specification include Z notation, B method, and Specification and Description Language(SDL).

Mathematical Notations: Formal system specification often involves using mathematicalnotat ions to express system requirements, constraints, and behaviors. These notations can include predicate logic, set theory, algebraic expressions, temporal logic, and process calculi, dependingon the chosen formal method.

Language Constructs: Formal specification languages provide constructs and syntax for expre ssing system properties. For example, Z notation includes constructs for defining data types, predicates, schema definitions, and invariants. These language constructs allow for a sy stematicand structured representation of the system.

Requirements Capture: Formal system specification helps in capturing and refining systemreq uirements. It enables the identification of inconsistencies, ambiguities, and gaps in Requirements early in the development process.

Formal methods facilitate the translation of informal requirements into precise and unambiguous specifications.

Verification and Validation: Formal system specification enables formal analysis, verification, and validation of system properties. Formal methods support the use of

automated tools and theorem provers to check the consistency, correctness, and completeness of system specifications. This helps in identifying design flaws, detecting logical inconsistencies, and verifying that the system meets its intended requirements.

Refinement: Formal specification allows for stepwise refinement of system specifications. System requirements can be progressively refined into more detailed specifications, ensuring that the design and implementation faithfully adhere to the original requirements.

Documentation:

Formal system specifications serve as a valuable source of documentation. They provide a clear and unambiguous representation of system requirements, constraints, and behavior, which aids in system understanding, maintenance, and future enhancements.

Formal system specification plays a crucial role in software engineering by enhancing the quality, reliability, and maintainability of software systems. It supports rigorous analysis, verification, and validation, leading to more dependable and trustworthy software designs theorem provers to check the consistency, correctness, and completeness of system specifications. This helps in identifying design flaws, detecting logical inconsistencies, and verifying that the system meets its intended requirements.

Refinement:

Formal specification allows for stepwise refinement of system specifications. System requirements can be progressively refined into more detailed specifications, ensuring that the design and implementation faithfully adhere to the original requirements.

Documentation:

Formal system specifications serve as a valuable source of documentation. They provide a clear and unambiguous representation of system requirements, constraints, and behavior, which aids in system understanding, maintenance, and future enhancements. Formal system specification plays a crucial role in software engineering by enhancing the quality, reliability, and maintainability of software systems. It supports rigorous analysis, verification, and validation, leading to more dependable and trustworthy software designs

Complex application can be visualized as a collection of different states processing a fixed set of events and producing a fixed set of outputs.

Loose coupling: An application can be divided into multiple behaviors or state machines, and each unit can be tested separately or could be reused in other applications.

Easy debugging and easy code maintenance.

Scalable

Narrow down the whole application completely to state-level complexity, analyze and implement.

Different types of state machines:

1.Mealy machine

2.Moore machine

3.Harel state charts

4.UML state machines

Some of these state machines are used for software engineering, and some state machines arestill being used in digital electronics, VLSI design, etc.

UML Modelling tool and code generator

Rhapsody® by IBM®

QM™ Model-based design tool by Quantum Leaps, LLC

Visual State by IAR

Yakindu state chart tools by Itemis AG Some are paid tools, and some are open-source. These tools can interpret your model and canauto-generate the code. Types of Finite State Machines Mealy machines are a type of simple state machine where the next state is determined by the current state and the given input.The next state is determined by checking which input generates the next state with the current state. Imagine a button that only works when you're logged in. That button is a state machine with login as the current state.

The new state is determined by logging in. Once you're logged in, the next state is determined by the current state which is logged in.

Moore Machines

A Moore State Machine is a type of state machine for modeling systems with a high degree of uncertainty. In these systems, predicting the exact sequence of future events is difficult. As a result, it is difficult to determine the next event. A Moore model captures this uncertainty by allowing the system to move from one state to another based on the outcome of a random event. A Moore model has many applications in both industry and academia.

Turing Machine

The Turing Machine consists of an input tape (with symbols on it), an internal tape (whichcorresponds to memory), and an output tape (which contains the result).A Turing Machine operates through a series of steps: it scans its input tape, reads out onesymbol at a time from its internal tape, and then applies this symbol as a command (or decision) to its output tape. For example: "If you see 'X' on the input tape, then print 'Y' onthe output tape."The input tape can be considered a finite set of symbols, while the internal and output tapes areinfinite. The Turing Machine must read an entire symbol from its internal tape before it canmove its head to the next symbol on the input tape.Once it has moved its head to

the next symbol, it can read that symbol out of its internal tapeand then move to the next symbol on its input tape.

This process continues until no more input or output symbols are left in the Turing Machine'sinternal or external tapes (at which point, it stops).

State Machine Use Cases

State machines are helpful for a variety of purposes. They can be used to model the flow of logic within a program, represent the states of a system, or for modeling the flow of events in a business process.There are many different types of state machines, ranging from simple to highly complex. Afew common use cases include:

Modeling Business Workflow Processes

State machines are ideal for modeling business workflows. This includes account setup flows,order completion, or a hiring process.These things have a beginning and an end and follow some sort of sequential order. Statemachines are also great for modeling tasks that involve conditional logic.

Business Decision-Making

Companies pair FSMs with their data strategy to explore the cause and effect of businessscenarios to make informed business decisions.Business scenarios are often complex and unpredictable. There are many possible outcomes,and each one impacts the business differently.A simulation allows you to try different business scenarios and see how each plays out. Youcan then assess the risk and determine the best course of action.

2.4 Petri nets

Petri nets are a mathematical modeling tool used in software engineering to analyze and describethe behavior of concurrent systems. They were introduced by Carl Adam Petri in the 1960s andhave since been widely used in various fields, including software engineering.Petri nets provide a graphical representation of a system's state and the transitions between thosestates. They consist of places, transitions, and arcs. Places represent states, transitions represent events or actions, and arcs represent the flow of tokens (also called markings) between places andtransitions.In software engineering, Petri nets can be used for various purposes, including:

System Modeling

: Petri nets are used to model the behavior of complex software systems,especially concurrent and distributed systems. They help in understanding and visualizing howdifferent components of the system interact and how their states change over time.

Specification and Verification

: Petri nets can be used to specify the desired behavior of asoftware system. By modeling the system using Petri nets, it becomes possible to formally
verify properties such as safety, liveness, reachability, and deadlock-
freeness. Verification techniques based on Petri nets help in identifying design flaws, potentia l deadlocks, or other issues early inthe development process.

Performance Analysis:

Petri nets can be used to analyze the performance of software systems,including their throughput, response time, and resource utilization. By modeling the system's behavior and introducing timing annotations, it becomes possible to perform quantitative analysisand predict system performance under different conditions.

Workflow Modeling

: Petri nets are often used to model business processes and workflows. Insoftware engineering, this is particularly useful for modeling the flow of tasks and activities insoftware development processes, such as agile methodologies or continuous integration/continuousdeployment (CI/CD) pipelines.

Software Testing:

Petri nets can be utilized in software testing to generate test cases and
verifythe correctness of the system. By modeling the system's behavior and different scenarios, it becomes possible to systematically generate test cases that cover various paths and states, helpingin identifying potential bugs and ensuring the software's reliability.

2.5 Object modelling using UML

UML (Unified Modeling Language) can be called a

graphical modeling language

that is usedin the field of software engineering. It specifies, visualizes, builds, and documents the softwaresystem's artifacts (main elements). It is a visual language, not a programming language. UMLdiagrams are used to depict the behavior and structure of a system. UML facilitates modeling,design, and analysis for software developers, business owners, and system architects. A pictureis worth a thousand words. The Benefits of Using UML

Because it is a general-purpose modeling language, it can be used by any modeler.UML is a straightforward modeling approach utilized to describe all practical systems.

Because no standard methods were available then, UML was developed to systemizeand condense object-oriented programming. Complex applications demandcollaboration and

preparation from numerous teams, necessitating a clear andstraightforward means of communication among them.

The

UML diagrams

are designed for customers, developers, laypeople, or anyone whowants to understand a system, whether software or non-software. Businesspeople do notunderstand code. As a result, UML becomes vital for communicating the system'sessential requirements, features, and procedures to non-programmers. Technical details became easier to understand by non-technical people through an introduction to UML.

When teams can visualize processes, user interactions, and the system's static structure,they save a lot of time in the long run.

Characteristics of UML

The UML has the following characteristics:

It is a modeling language that has been generalized for various use cases.

It is not a programming language; instead, it is a graphical modeling language that usesdiagrams that can be understood by non-programmers as well.

It has a close connection to object-oriented analysis and design.

It is used to visualize the system's workflow.

Conceptual Modeling

Before moving ahead with the concept of UML, we should first understand the basics of theconceptual
model.A conceptual model is composed of several interrelated concepts. It makes it easy toun derstand the objects and how they interact with each other. This is the first step beforedrawing UML diagrams.Following are some object-oriented concepts that are needed to begin with UML:

Object:

An object is a real world entity. There are many objects present within a singlesystem. It is a fundamental building block of UML.

o

Class:

A class is a software blueprint for objects, which means that it defines thevariables and methods common to all the objects of a particular type.

o

Abstraction:

Abstraction is the process of portraying the essential characteristics of anobject to the users while hiding the irrelevant information. Basically, it is used toenvision the functioning of an object.

o

Inheritance:

Inheritance is the process of deriving a new class from the existing ones.

o

Polymorphism:

It is a mechanism of representing objects having multiple forms usedfor different purposes.

o

Encapsulation:

It binds the data and the object together as a single unit, enabling tightcoupling between them.

2.5.1 Use Case Diagram

A use case diagram is used to represent the dynamic behavior of a system. Itencapsulates the system's functionality by incorporating use cases, actors, and theirrelationships. It models the tasks, services, and functions required by asystem/subsystem of an application. It depicts the high-level functionality of a systemand also tells how the user handles a system.

Purpose of Use Case Diagrams

The main purpose of a use case diagram is to portray the dynamic aspect of a system. Itaccumulates the system's requirement, which includes both internal as well as externalinflue

nces. It invokes persons, use cases, and several things that invoke the actors and elementsaccountable for the implementation of use case diagrams. It represents how an entity from theexternal environment can interact with a part of the system.Following are the purposes of a use case diagram given below:1.It gathers the system's needs.2.It depicts the external view of the system.3.It recognizes the internal as well as external factors that influence the system.4.It represents the interaction between the actors.

How to draw a Use Case diagram?

It is essential to analyze the whole system before starting with drawing a use case diagram, andthen the system's functionalities are found. And once every single functionality is identified,they are then transformed into the use cases to be used in the use case diagram.After that, we will enlist the actors that will interact with the system. The actors are the personor a thing that invokes the functionality of a system. It may be a system or a private entity, suchthat it requires an entity to be pertinent to the functionalities of the system to which it is goingto interact.Once both the actors and use cases are enlisted, the relation between the actor and use case/system is inspected. It identifies the no of times an actor communicates with the system.Basically, an actor can interact multiple times with a use case or system at a particular instanceof time.Following are some rules that must be followed while drawing a use case diagram:1.A pertinent and meaningful name should be assigned to the actor or a use case of asystem.2.The communication of an actor with a use case must be defined in an understandableway.3.Specified notations to be used as and when required.4.The most significant interactions should be represented among the multiple no of i nteractions between the use case and actors.

Example of a Use Case Diagram

A use case diagram depicting the Online Shopping website is given below.Here the Web Customer actor makes use of any online shopping website to purchase online.The top-level uses are as follows; View Items, Make Purchase, Checkout, Client Register.The

View Items

use case is utilized by the customer who searches and view products.The

Client Register

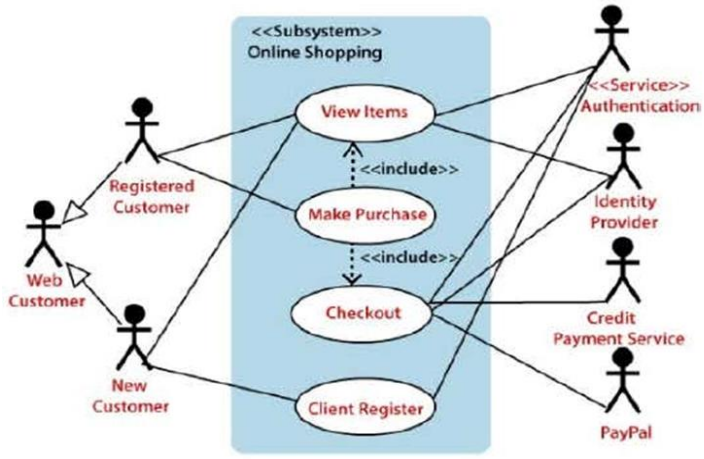
use case allows the customer to register itself with the website for availinggift vouchers, coupons, or getting a private sale invitation. It is to be noted that the

Checkout

isan included use case, which is part of

Making Purchase,

and it is not available by itself.



The

View Items

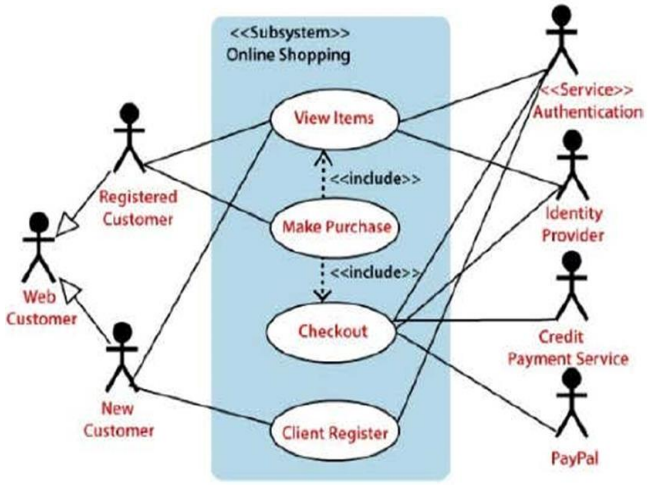
is further extended by several use cases such as; Search Items, Browse Items,View Recommended Items, Add to Shopping Cart, Add to Wish list. All of these extended usecases provide some functions to customers, which allows them to search for an item. The ViewItems is further extended by several use cases such as; Search Items, Browse Items, ViewRecommended Items, Add to Shopping Cart, Add to Wish list. All of these extended use cases provide some functions to customers, which allows them to search for an item.Both

View Recommended Item

and

Add to Wish List

include the Customer Authenticationuse case, as they necessitate authenticated customers, and simultaneously item can be added tothe shopping cart without any user authentication.



The

View Items

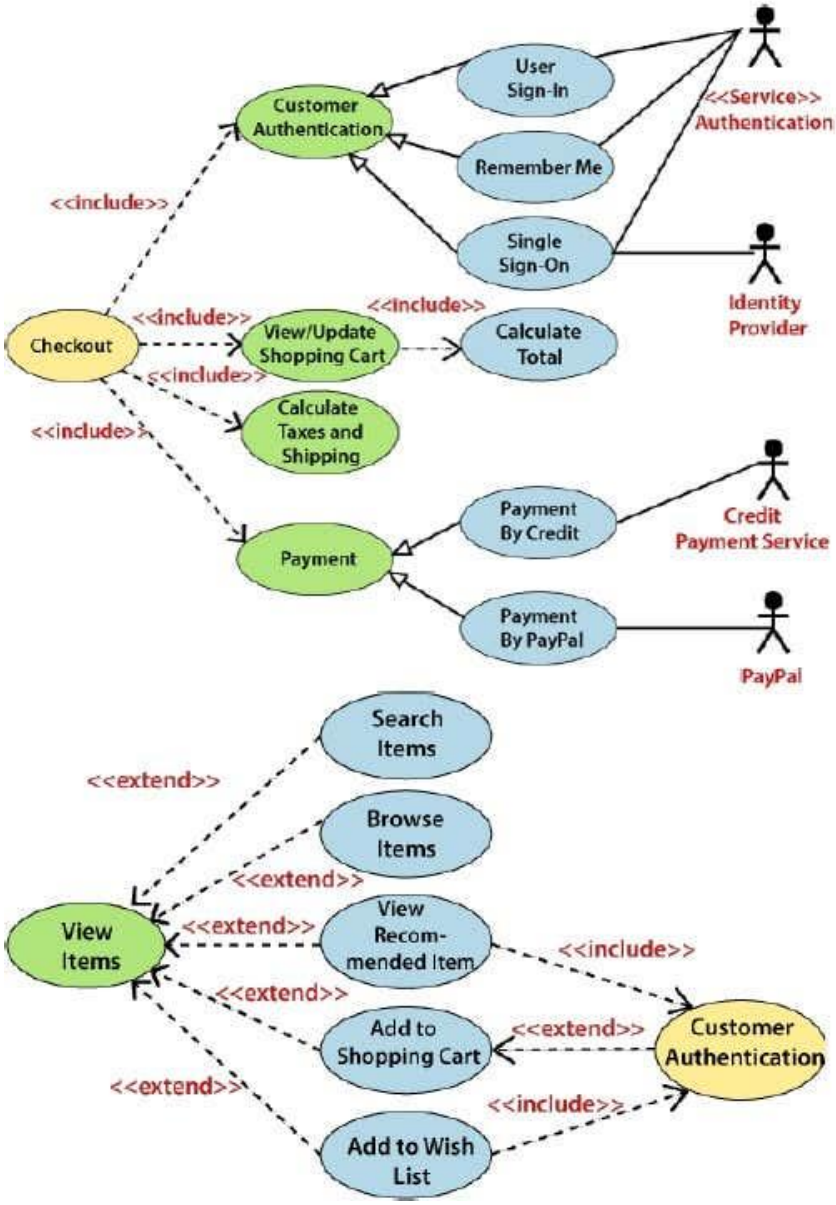
is further extended by several use cases such as; Search Items, Browse Items,View Recommended Items, Add to Shopping Cart, Add to Wish list. All of these extended usecases provide some functions to customers, which allows them to search for an item. The ViewItems is further extended by several use cases such as; Search Items, Browse Items, ViewRecommended Items, Add to Shopping Cart, Add to Wish list. All of these extended use cases provide some functions to customers, which allows them to search for an item.Both

View Recommended Item

and

Add to Wish List

include the Customer Authenticationuse case, as they necessitate authenticated customers, and simultaneously item can be added tothe shopping cart without any user authentication.



Similarly, the

Checkout

use case also includes the following use cases, as shown below. Itrequires an authenticated Web Customer, which can be done by login page, user authenticationcookie ("Remember me"), or Single Sign-On (SSO). SSO needs an external identity provider's participation, while Web site authentication service is utilized in all these use cases.The Checkout use case

involves Payment use case that can be done either by the credit cardand external credit payment services or with PayPal.

Important tips for drawing a Use Case diagram

Following are some important tips that are to be kept in mind while drawing a use casediagram:1.A simple and complete use case diagram should be articulated.2.A use case diagram should represent the most significant interaction among themultiple interactions.3.At least one module of a system should be represented by the use case diagram.4.If the use case diagram is large and more complex, then it should be drawn moregeneralized.

2.5.2 Class diagram

The class diagram depicts a static view of an application. It represents the types of objectsresiding in the system and the relationships between them. A class consists of its objects, andalso it may inherit from other classes. A class diagram is used to visualize, describe, documentvarious different aspects of the system, and also construct executable software code.It shows the attributes, classes, functions, and relationships to give an overview of the softwaresystem. It constitutes class names, attributes, and functions in a separate compartment thathelps in software development. Since it is a collection of classes, interfaces, associations,collaborations, and constraints, it is termed as a structural diagram.

Purpose of Class Diagrams

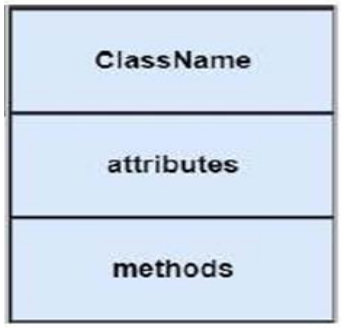
The main purpose of class diagrams is to build a static view of an application. It is the onlydiagram that is widely used for construction, and it can be mapped with object-orientedlanguages. It is one of the most popular UML diagrams. Following are the purpose of classdiagrams given below:1.It analyses and designs a static view of an application.2.It describes the major responsibilities of a system.3.It is a base for component and deployment diagrams.4.It incorporates forward and reverse engineering.

Benefits of Class Diagrams

1.It can represent the object model for complex systems.2.It reduces the maintenance time by providing an overview of how an application isstructured before coding.3.It provides a general schematic of an application for better understanding.4.It represents a detailed chart by highlighting the desired code, which is to be programmed.5.It is helpful for the stakeholders and the developers.

Vital components of a Class Diagram

The class diagram is made up of three sections:



o

Upper Section:

The upper section encompasses the name of the class. A class is arepresentation of similar objects that shares the same relationships, attributes,operations, and semantics. Some of the following rules that should be taken intoaccount while representing a class are given below:a.Capitalize the initial letter of the class name. b.Place the class name in the center of the upper section.c.A class name must be written in bold format.d.The name of the abstract class should be written in italics format.

o

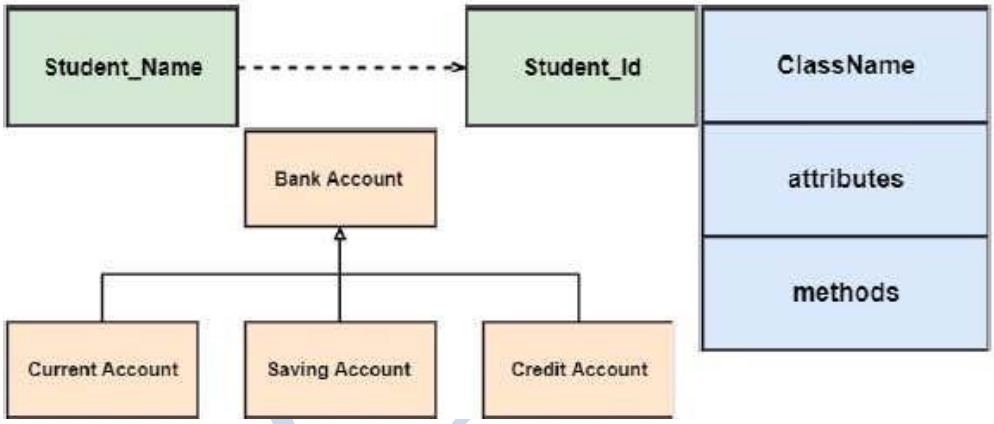
Middle Section:

The middle section constitutes the attributes, which describe thequality of the class. The attributes have the following characteristics:a.The attributes are written along with its visibility factors, which are public (+), private(-), protected (#), and package (~). b.The accessibility of an attribute class is illustrated by the visibility factors.c.A meaningful name should be assigned to the attribute, which will explain itsusage inside the class.

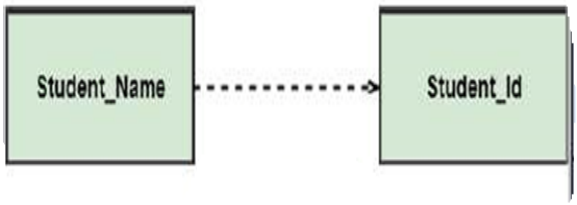
o

Lower Section:

The lower section contain methods or operations. The methods arerepresented in the form of a list, where each method is written in a single line. Itdemonstrates how a class interacts with data.



In UML, relationships are of three types:

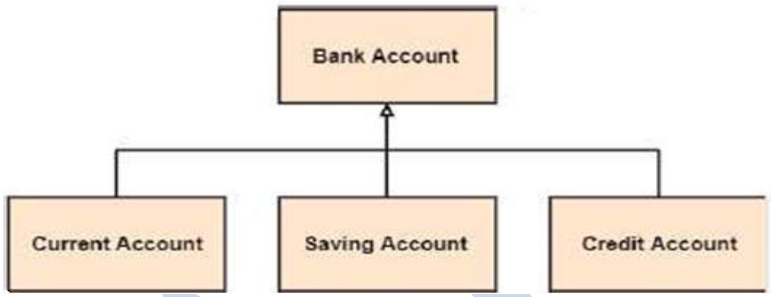


o

Dependency:

A dependency is a semantic relationship between two or more classeswhere a change in one class cause changes in another class. It forms a weaker relations hip.In the following example, Student_Name is dependent on the Student_Id

Generalization:



A generalization is a relationship between a parent class (superclass)and a child class (subclass). In this, the child class is inherited from the parent class.For example, The Current Account, Saving Account, and Credit Account are thegeneralized form of Bank Account.

o

Association:



It describes a static or physical connection between two or more objects.It depicts how many objects are there in the relationship.For example, a department is associated with the college.

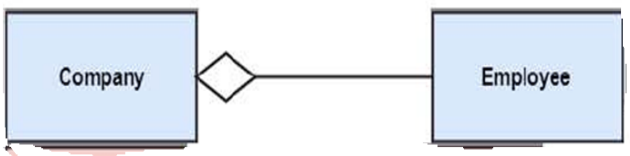
Generalization:

A generalization is a relationship between a parent class (superclass)and a child class (subclass). In this, the child class is inherited from the parent class.For example, The Current Account, Saving Account, and Credit Account are thegeneralized form of Bank Account.

o

Association:

It describes a static or physical connection between two or more objects.It depicts how many objects are there in the relationship.For example, a department is associated with the college.



Generalization:

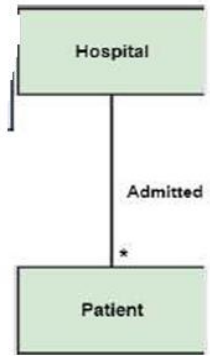
A generalization is a relationship between a parent class (superclass)and a child class (subclass). In this, the child class is inherited from the parent class.For example, The Current Account, Saving Account, and Credit Account are thegeneralized form of Bank Account.

o

Association:

It describes a static or physical connection between two or more objects.It depicts how many objects are there in the relationship.For example, a department is associated with the college

Multiplicity:



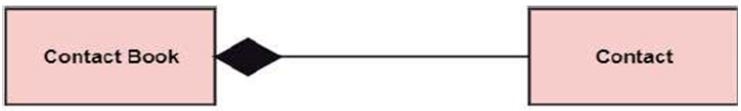
It defines a specific range of allowable instances of attributes. In case if a rangeis not specified, one is considered as a default multiplicity.For example, multiple patients are admitted to one hospital.

Aggregation:

An aggregation is a subset of association, which represents has a relationship. Itis more specific then association. It defines a part-whole or part-of relationship. In this kind of relationship, the child class can exist independently of its parent class.The company encompasses a number of employees, and even if one employee resigns, thecompany still exists.

Composition:

The composition is a subset of aggregation. It portrays the dependency betweenthe parent and its child, which means if one part is deleted, then the other part also getsdiscarded. It represents a whole-part relationship.A contact book consists of multiple contacts, and if you delete the contact book, all the contactswill be lost.



Multiplicity:

It defines a specific range of allowable instances of attributes. In case if a rangeis not specified, one is considered as a default multiplicity.For example, multiple patients are admitted to one hospital.

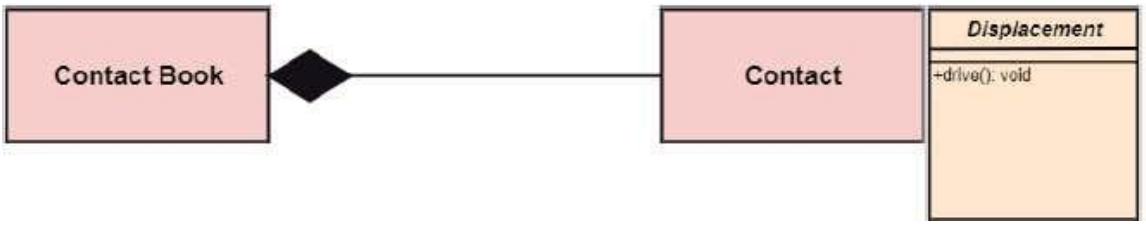
Aggregation:

An aggregation is a subset of association, which represents has a relationship. Itis more specific then association. It defines a part-whole or part-of relationship. In this kind of relationship, the child class can exist independently of its parent class.The company encompasses a number of employees, and even if one employee resigns, thecompany still exists.

Composition:

The composition is a subset of aggregation. It portrays the dependency betweenthe parent and its child, which means if one part is deleted, then the other part also getsdiscarded. It represents a whole-part relationship.A contact book consists of multiple contacts, and if you delete the contact book, all the contactswill be lost.

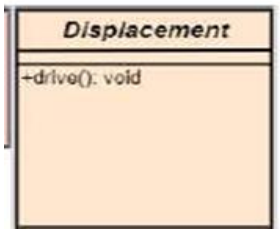
APEC



Abstract Classes

In the abstract class, no objects can be a direct entity of the abstract class. The abstract classcan neither be declared nor be instantiated. It is used to find the functionalities across theclasses. The notation of the abstract class is similar to that of class; the only difference is thatthe name of the class is written in italics. Since it does not involve any implementation for agiven function, it is best to use the abstract class with multiple objects.Let us assume that

we have an abstract class named displacement with a method declaredinside it, and that method will be called as a drive ()



. Now, this abstract class method can beimplemented by any object, for example, car, bike, scooter, cycle, etc.

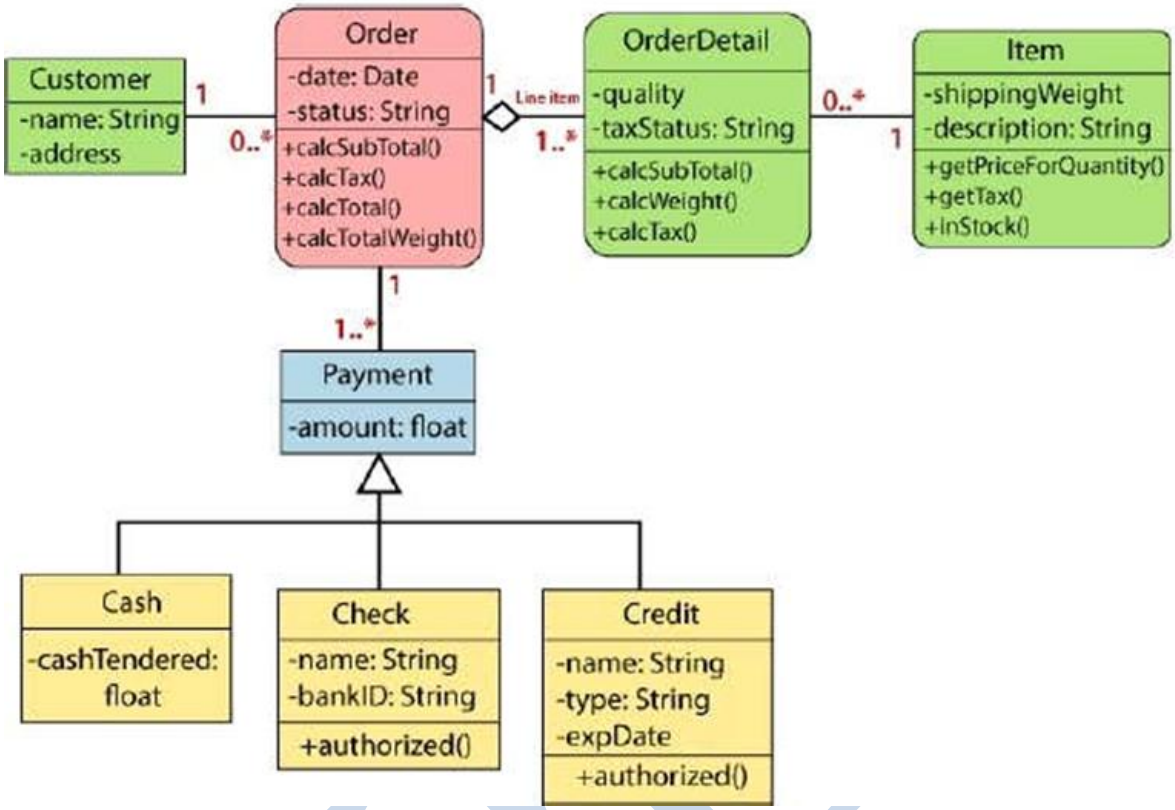
How to draw a Class Diagram?

The class diagram is used most widely to construct software applications. It not only representsa static view of the system but also all the major aspects of an application. A collection of classdiagrams as a whole represents a system.Some key points that are needed to keep in mind while drawing a class diagram are given below:1.To describe a complete aspect of the system, it is suggested to give a meaningful nameto the class diagram.2.The objects and their relationships should be acknowledged in advance.3.The attributes and methods (responsibilities) of each class must be known.4.A minimum number of desired properties should be specified as more number of theunwanted property will lead to a complex diagram.

5.Notes can be used as and when required by the developer to describe the aspects of adiagram.6.The diagrams should be redrawn and reworked as many times to make it correct before producing its final version.

Class Diagram Example

A class diagram describing the sales order system is given below



Usage of Class diagrams

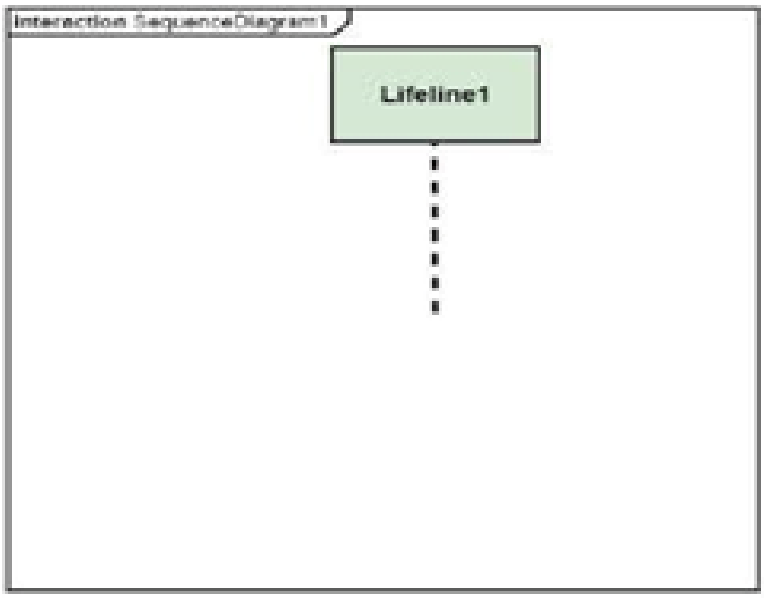
The class diagram is used to represent a static view of the system. It plays an essential role inthe establishment of the component and deployment diagrams. It helps to construct anexecu table code to perform forward and backward engineering for any system, or we can say itis mainly used for construction. It represents the mapping with object-oriented languages thatare C++, Java, etc. Class diagrams can be used for the following purposes:1.To describe the static view of a system.2.To show the collaboration among every instance in the static view.3.To describe the functionalities performed by the system.4.To construct the software application using object-oriented languages.

Interaction diagram

As the name suggests, the interaction diagram portrays the interactions between distinct entities present in the model. It amalgamates both the activity and sequence diagrams. Theco mmunication is nothing but units of the behaviour of a classifier that provides context for interactions.A set of messages that are interchanged between the entities to achieve certain specified tasksin the system is termed as interaction. It may incorporate any feature of the classifier of whichit has access. In the interaction diagram, the critical

component is the messages and the lifeline.In UML, the interaction overview diagram initiates the interaction between the objects utilizingmessage passing. While drawing an interaction diagram, the entire focus is to represent therelationship among different objects which are available within the system boundary and themessage exchanged by them to communicate with each other.The message exchanged among objects is either to pass some information or to request someinformation. And based on the information, the interaction diagram is categorized into thesequence diagram, collaboration diagram, and timing diagram.The sequence diagram envisions the order of the flow of messages inside the system bydepicting the communication between two lifelines, just like a time-ordered sequence of events.The collaboration diagram, which is also known as the communication diagram, representshow lifelines connect within the system, whereas the timing diagram focuses on that instantwhen a message is passed from one element to the other.

Notation of an Interaction Diagram



Notation of an Interaction Diagram

Purpose of an Interaction Diagram

The interaction diagram helps to envision the interactive (dynamic) behavior of any system. It portrays how objects residing in the system communicates and connects to each other. It als o provides us with a context of communication between the lifelines inside the system.Following are the purpose of an interaction diagram given below:1.To visualize the dynamic behavior of the system.2.To envision the interaction and the message flow in the system.3.To portray the structural aspects of the entities within the system.4.To represent the order of the sequenced interaction in the system.5.To visualize the real-time data and represent the architecture of an object-orientedsystem.

How to draw an Interaction Diagram?

Since the main purpose of an interaction diagram is to visualize the dynamic behavior of thesystem, it is important to understand what a dynamic aspect really is and how we can visualizeit. The dynamic aspect is nothing but a screenshot of the system at the run time.Before drawing an interaction diagram, the first step is to discover the scenario for which thediagram will be made. Next, we will identify various lifelines that will be invoked in the communication, and then we will classify each
lifeline. After that, the connections areinvestigated and how the lifelines are interrelated to each other.Following are some things that are needed:1.A total no of lifeline which will take part in the communication.2.The sequence of the message flow among several entities within the system.3.No operators used to ease out the functionality of the diagram.4.Several distinct messages that depict the interactions in a precise and clear way.5.The organization and structure of a system.6.The order of the sequence of the flow of messages.7.Total no of time constructs of an object.

Use of an Interaction Diagram

The interaction diagram can be used for:1.The sequence diagram is employed to investigate a new application.2.The interaction diagram explores and compares the use of the collaboration diagramsequence diagram and the timing diagram.3.The interaction diagram represents the interactive (dynamic) behaviour of the system.4.The sequence diagram portrays the order of control flow from one element to the other elements inside the system, whereas the collaboration diagrams are employed to get anoverview of the object architecture of the system.5.The interaction diagram models the system as a time-ordered sequence of a system.6.The interaction diagram models the system as a time-ordered sequence of a system.7.The interaction diagram systemizes the structure of the interactive elements.

2.5.4 Activity diagram

In UML, the activity diagram is used to demonstrate the flow of control within the systemrather than the implementation. It models the concurrent and sequential activities

The activity diagram helps in envisioning the workflow from one activity to another. It putemphasis on the condition of flow and the order in which it occurs. The flow can be sequential, branched, or concurrent, and to deal with such kinds of flows, the activity diagram

has come upwith a fork, join, etc.It is also termed as an object-oriented flowchart. It encompasses activities composed of a set of actions or operations that are applied to model the behavioral diagram.

Components of an Activity Diagram

Following are the component of an activity diagram:

Activities



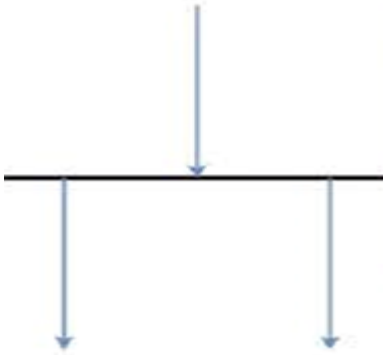
The categorization of behavior into one or more actions is termed as an activity. In other words, it can be said that an activity is a network of nodes that are connected by edges. Theedges depict the flow of execution. It may contain action nodes, control nodes, or object nodes.The control flow of activity is represented by control nodes and object nodes that illustrates theobjects used within an activity. The activities are initiated at the initial node and are terminatedat the final node.

Activity partition /swimlane



The swimlane is used to cluster all the related activities in one column or one row. It can beeither vertical or horizontal. It used to add modularity to the
activity diagram. It is notnecessary to incorporate swimlane in the activity diagram. But it is u sed to add moretransparency to the activity diagram

Forks



Forks and join nodes generate the concurrent flow inside the activity. A fork node consists of one inward edge and several outward edges. It is the same as that of various decision para meters. Whenever a data is received at an inward edge, it gets copied
and split crosswaysvarious outward edges. It split a single inward flow into multiple parallel flows.

Join Nodes

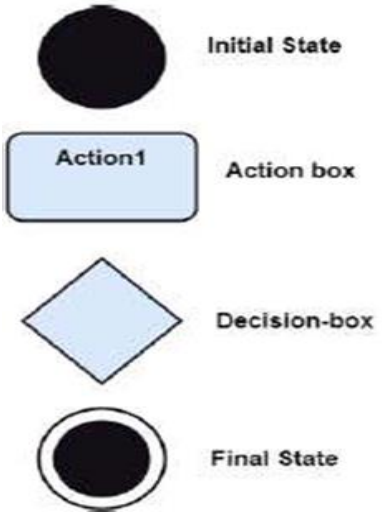
Join nodes are the opposite of fork nodes. A Logical AND operation is performed on all of theinward edges as it synchronizes the flow of input across one single output (outward) edge.



Pins

It is a small rectangle, which is attached to the action rectangle. It clears out all the messy andcomplicated thing to manage the execution flow of activities. It is an object node that preciselyrepresents one input to or output from the action.

Notation of an Activity diagram



Activity diagram constitutes following notations:

Initial State:

It depicts the initial stage or beginning of the set of actions.

Final State:

It is the stage where all the control flows and object flows end.

Decision Box:

It makes sure that the control flow or object flow will follow only one path.

Action Box:

It represents the set of actions that are to be performed

Why use Activity Diagram?

An event is created as an activity diagram encompassing a group of nodes associated withedges. To model the behavior of activities, they can be attached to any modeling element. Itcan model use cases, classes, interfaces, components, and collaborations.It mainly models processes and workflows. It envisions the dynamic behavior of the system aswell as constructs a runnable system that incorporates forward and reverse engineering. It doesnot include the message part, which means message flow is not represented in an activitydiagram.

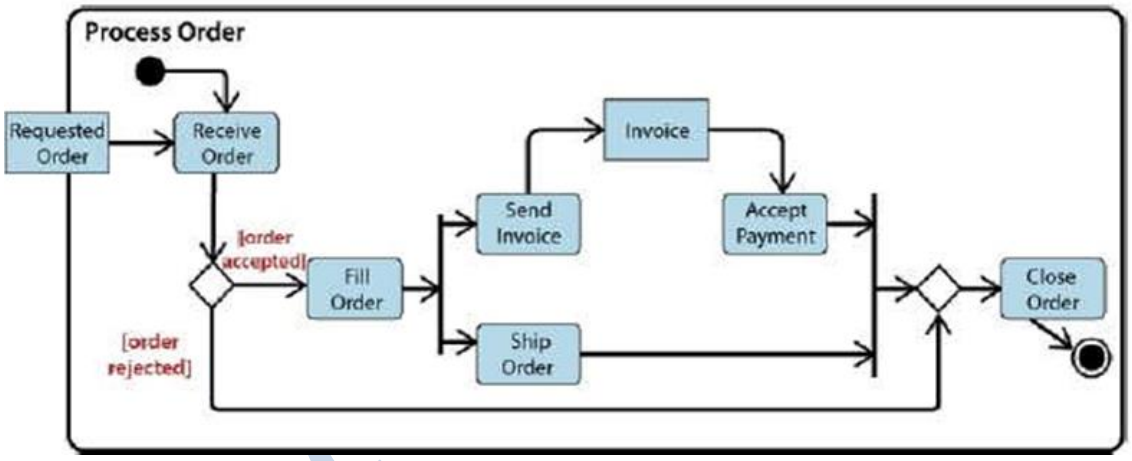
It is the same as that of a flowchart but not exactly a flowchart itself. It is used to depict theflow between several activities.

How to draw an Activity Diagram?

An activity diagram is a flowchart of activities, as it represents the workflow among variousactivities. They are identical to the flowcharts, but they themself are not exactly the flowchart.In other words, it can be said that an activity diagram is an enhancement of the flowchart,which encompasses several unique skills.Since it incorporates swimlanes, branching, parallel flows, join nodes, control nodes, and forks,it supports exception handling. A system must be explored as a whole before drawing anactivity diagram to provide a clearer view of the user. All of the activities are explored after they are properly analyzed for finding out the constraints applied to the activities. Each andevery activity, condition, and association must be recognized.After gathering all the essential information, an abstract or a prototype is built, which is thentransformed into the actual diagram.Following are the rules that are to be followed for drawing an activity diagram:1.A meaningful name should be given to each and every activity.2.Identify all of the constraints.3.Acknowledge the activity associations.

Example of an Activity Diagram

An example of an activity diagram showing the business flow activity of order processing isgiven below.Here the input parameter is the Requested order, and once the order is accepted, all of therequired information is then filled, payment is also accepted, and then the order is shipped. It permits order shipment before an invoice is sent or payment is completed.



When to use an Activity Diagram?

An activity diagram can be used to portray business processes and workflows. Also, it used for modeling business as well as the software. An activity diagram is utilized for the followings:1.To graphically model the workflow in an easier and understandable way.2.To model the execution flow among several activities.3.To model comprehensive information of a function or an algorithm employed withinthe system.4.To model the business process and its workflow.5.To envision the dynamic aspect of a system.6.To generate the top-level

flowcharts for representing the workflow of an application.7.To represent a high-level view of a distributed or an object-oriented system.

2.5.5 State chart diagram

A

state diagram

is used to represent the condition of the system or part of the system atfinite instances of time. It's a

behavioral

diagram and it represents the behavior using finitestate transitions. State diagrams are also referred to as

State machines

and

State-chartDiagrams

. These terms are often used interchangeably. So simply, a state diagram is used tomodel the dynamic behavior of a class in response to time and changing external stimuli. Wecan say that each and every class has a state but we don't model every class using Statediagrams. We prefer to model the states with three or more states.

Uses of statechart diagram

We use it to state the events responsible for change in state (we do not show what processes cause those events).

We use it to model the dynamic behavior of the system .

To understand the reaction of objects/classes to internal or external stimuli.Firstly let us understand what are

Behavior diagrams

? There are two types of diagrams inUML :1.

Structure Diagrams –

Used to model the static structure of a system, for example- classdiagram, package diagram, object diagram, deployment diagram etc.2.

Behavior diagram –

Used to model the dynamic change in the system over time. Theyare used to model and construct the functionality of a system. So, a behavior diagramsimply guides us through the functionality of the system using Use case diagrams,Interaction diagrams, Activity diagrams and State diagrams.

Basic components of a statechart diagram –

1.

Initial state –

We use a black filled circle represent the initial state of a System or aclass.

Figure –

initial state notation2.

Transition –

We use a solid arrow to represent the transition or change of control fromone state to another. The arrow is labelled with the event which causes the change instate.

Figure –

transition3.

State –

We use a rounded rectangle to represent a state. A state represents the conditionsor circumstances of an object of a class at an instant of time.

Figure –

state notation

34







4.

Fork –

We use a rounded solid rectangular bar to represent a Fork notation withincoming arrow from the parent state and outgoing arrows towards the newly createdstates. We use the fork notation to represent a state splitting into two or more concurrentstates.

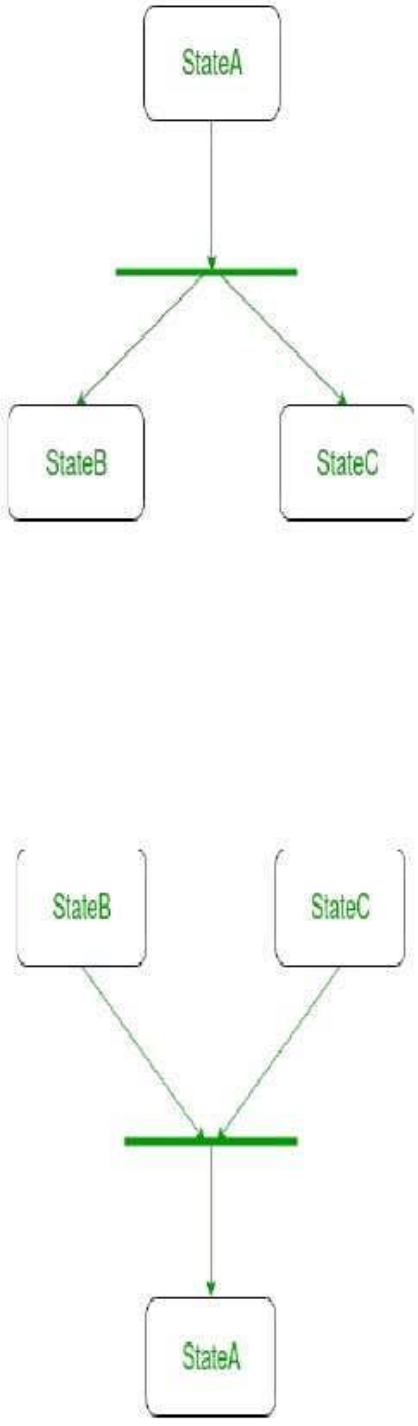


Figure –

a diagram using the fork notation5.

Join –

We use a rounded solid rectangular bar to represent a Join notation with incomingarrows from the joining states and outgoing arrow towards the common goal state. Weuse the join notation when two or more states concurrently converge into one on theoccurrence of an event or events.

Figure –

a diagram using join notation

35

Self transition –

We use a solid arrow pointing back to the state itself to represent a selftransition. There might be scenarios when the state of the object does not change upon theoccurrence of an event. We use self transitions to represent such cases.

Figure –

self transition notation7.

Composite state –

We use a rounded rectangle to represent a composite state also.Werepresent a state with internal activities using a composite state.

Figure –

a state with internal activities8.

Final state –

We use a filled circle within a circle notation to represent the final state in astate machine diagram.

Figure –

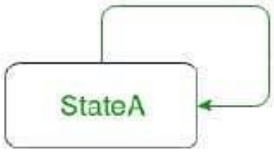
final state notation

Steps to draw a state diagram –

1.Identify the initial state and the final terminating states.2.Identify the possible states in which the object can exist (boundary values correspondingto different attributes guide us in identifying different states).3.Label the events which trigger these transitions.

Example –

state diagram for an online order


StateA


Composite State



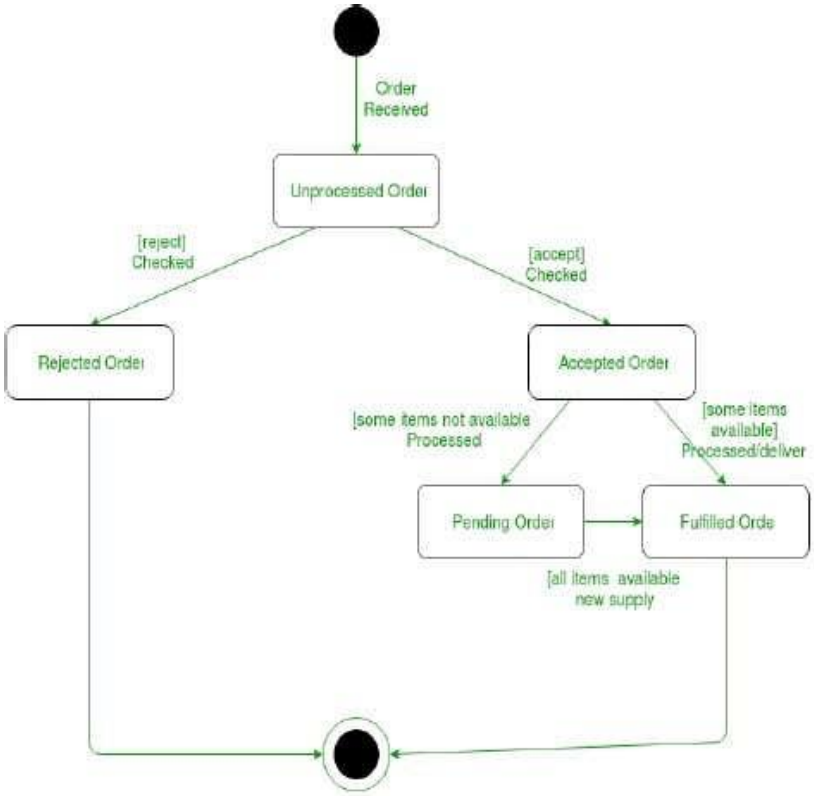


Figure –

state diagram for an online order The UMl diagrams we draw depend on the system we aim to represent. Here is just anexample of how an online ordering system might look like :1.On the event of an order being received, we transit from our initial state to Unprocessedorder state.2.The unprocessed order is then checked.3.If the order is rejected, we transit to the Rejected Order state.4.If the order is accepted and we have the items available we transit to the fulfilled orderstate.5.However if the items are not available we transit to the Pending

Order state.6.After the order is fulfilled, we transit to the final state. In this example, we merge the twostates i.e. Fulfilled order and Rejected order into one final state.

37

.6 Functional Modelling

Functional Modelling gives the process perspective of the object-oriented analysis model andan overview of what the system is supposed to do. It defines the function of the internal processes in the system with the aid of Data Flow Diagrams (DFDs). It depicts the functionalderivation of the data values without indicating how they are derived when they are computed,or why they need to be computed.

2.6.1 Data Flow Diagrams

Functional Modelling is represented through a hierarchy of DFDs. The DFD is a graphicalrepresentation of a system that shows the inputs to the system, the processing upon the inputs,the outputs of the system as well as the internal data stores. DFDs illustrate the series of transformations or computations performed on the objects or the system, and the externalcontrols and objects that affect the transformation.Rumbaugh et al. have defined DFD as, “A data flow diagram is a graph which shows the flowof data values from their sources in objects through processes that transform them to their destinations on other objects.”The four main parts of a DFD are −

-

Processes,

-

Data Flows,

-

Actors, and

-

Data Stores.The other parts of a DFD are −

-

Constraints, and

-

Control Flows.

Features of a DFDProcesses

Processes are the computational activities that transform data values. A whole system can bevisualized as a high-level process. A process may be further divided into smaller components.The lowest-level process may be a simple function.

Representation in DFD

− A process is represented as an ellipse with its name written inside itand contains a fixed number of input and output data values.

Example

− The following figure shows a process Compute_HCF_LCM that accepts twointegers as inputs and outputs their HCF (highest common factor) and LCM (least commonmultiple).Data Flows

38

Data flow represents the flow of data between two processes. It could be between an actor anda process, or between a data store and a process. A data flow denotes the value of a data item atsome point of the computation. This value is not changed by the data flow.

Representation in DFD

− A data flow is represented by a directed arc or an arrow, labelledwith the name of the data item that it carries.In the above figure, Integer_a and Integer_b represent the input data flows to the process, whileL.C.M. and H.C.F. are the output data flows.A data flow may be forked in the following cases −

The output value is sent to several places as shown in the following figure. Here, theoutput arrows are unlabelled as they denote the same value.

The data flow contains an aggregate value, and each of the components is sent todifferent places as shown in the following figure. Here, each of the forked componentsis labelled.ActorsActors are the active objects that interact with the system by either producing data andinputting them to the system, or consuming data produced by the system. In other words, actorsserve as the sources and the sinks of data.

Representation in DFD

− An actor is represented by a rectangle. Actors are connected to theinputs and outputs and lie on the boundary of the DFD.

Example

− The following figure shows the actors, namely, Customer and Sales_Clerk in acounter sales system.

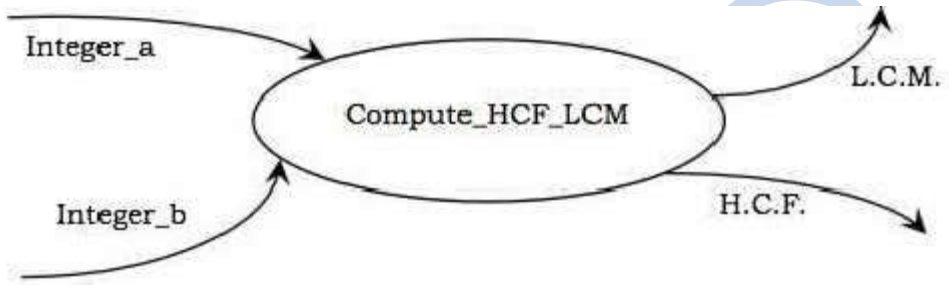
Data Stores

Data stores are the passive objects that act as a repository of data. Unlike actors, they cannot perform any operations. They are used to store data and retrieve the stored data. They representa data structure, a disk file, or a table in a database.

Representation in DFD

− A data store is represented by two parallel lines containing thename of the data store. Each data store is connected to at least one process. Input arrows

3



contain information to modify the contents of the data store, while output arrows containinformation retrieved from the data store. When a part of the information is to be retrieved, theoutput arrow is labelled. An unlabelled arrow denotes full data retrieval. A two-way arrowimplies both retrieval and update.

Example

− The following figure shows a data store, Sales_Record, that stores the details of allsales. Input to the data store comprises of details of sales such as item, billing amount, date,etc. To find the average sales, the process retrieves the sales records and computes the average.ConstraintsConstraints specify the conditions or restrictions that need to be satisfied over time. They allowadding new rules or modifying existing ones. Constraints can appear in all the three models of object-oriented analysis.

In Object Modelling, the constraints define the relationship between objects. They mayalso define the relationship between the different values that an object may take atdifferent times.

In Dynamic Modelling, the constraints define the relationship between the states andevents of different objects.

In Functional Modelling, the constraints define the restrictions on the transformationsand computations.

Representation

− A constraint is rendered as a string within braces.

Example

− The following figure shows a portion of DFD for computing the salary of employees of a company that has decided to give incentives to all employees of the salesdepartment and increment the salary of all employees of the HR department. It can be seen thatthe constraint {Dept:Sales} causes incentive to be calculated only if the department is sales andthe constraint {Dept:HR} causes increment to be computed only if the department is HR.

Control Flows

40

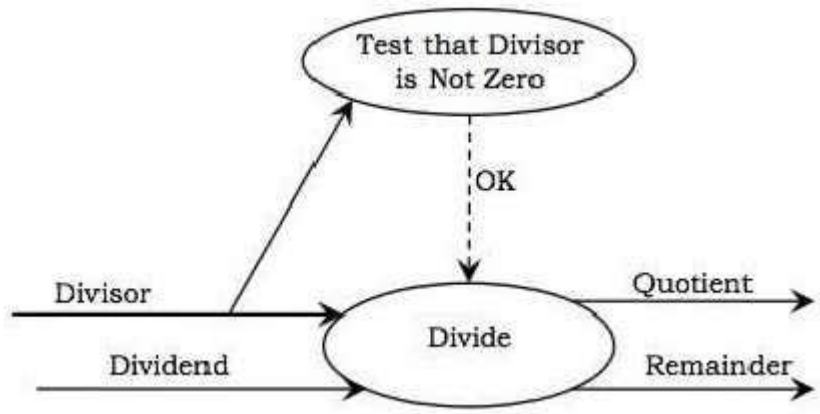
A process may be associated with a certain Boolean value and is evaluated only if the value istrue, though it is not a direct input to the process. These Boolean values are called the controlflows.

Representation in DFD

− Control flows are represented by a dotted arc from the process producing the Boolean value to the process controlled by them.

Example

− The following figure represents a DFD for arithmetic division. The Divisor istested for non-zero. If it is not zero, the control flow OK has a value True and subsequently theDivide process computes the Quotient and the Remainder.



Developing the DFD Model of a System

In order to develop the DFD model of a system, a hierarchy of DFDs are constructed. The top-level DFD comprises of a single process and the actors interacting with it.At each successive lower level, further details are gradually included. A process is decomposedinto sub-processes, the data flows among the sub-processes are identified, the control flows aredetermined, and the data stores are defined. While decomposing a process, the data flow into or out of the process should match the data flow at the next level of DFD.

Example

− Let us consider a software system, Wholesaler Software, that automates thetransactions of a wholesale shop. The shop sells in bulks and has a clientele comprising of merchants and retail shop owners. Each customer is asked to register with his/her particularsand is given a unique customer code, C_Code. Once a sale is done, the shop registers its detailsand sends the goods for dispatch. Each year, the shop distributes Christmas gifts to itscustomers, which comprise of a silver coin or a gold coin depending upon the total sales andthe decision of the proprietor.The functional model for the Wholesale Software is given below. The figure below shows thetop-level DFD. It shows the software as a single process and the actors that interact with it.The actors in the system are −

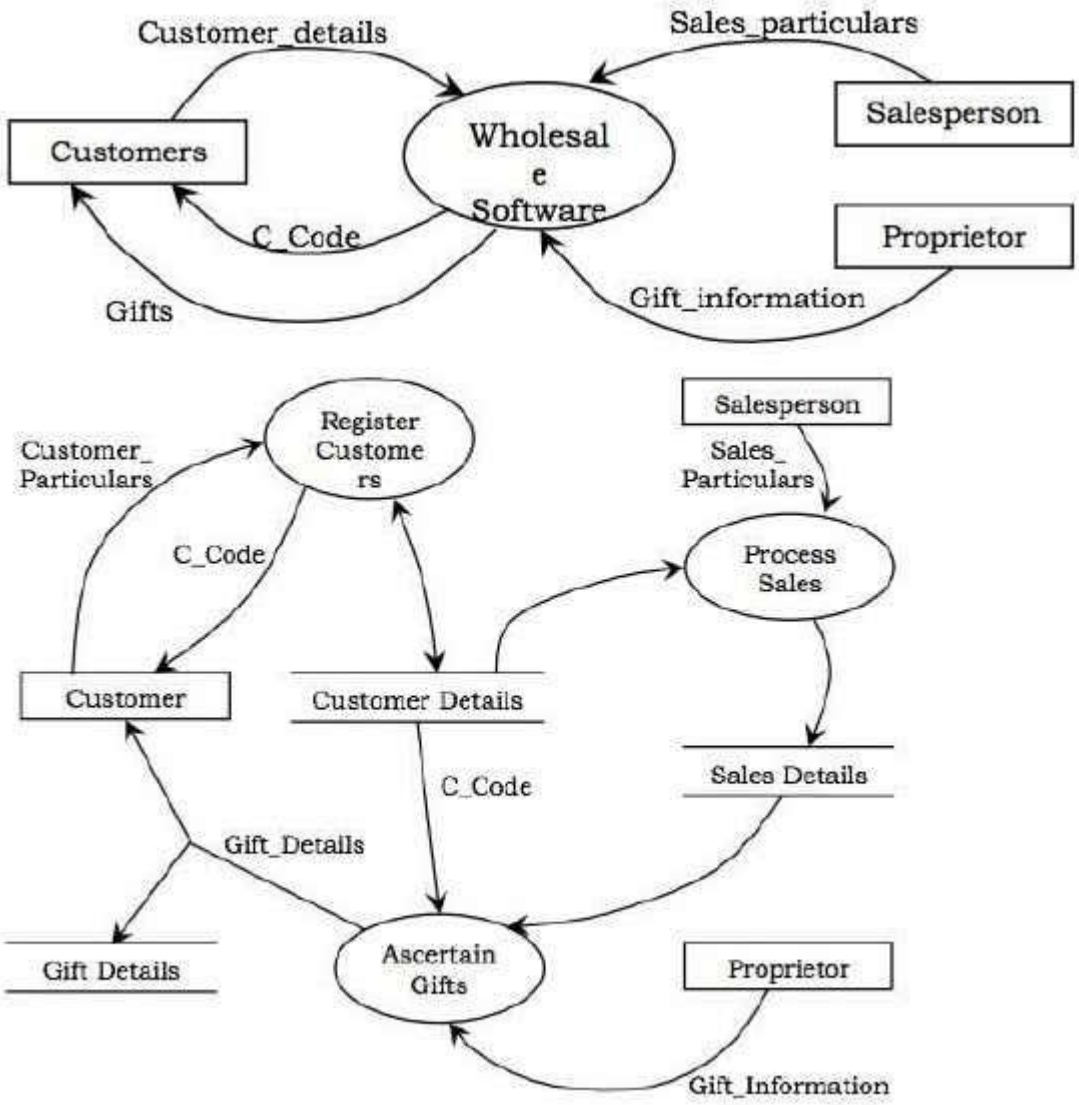
Customers

Salesperson

Proprietor

4

APEC



DFD of Wholesale Software

In the next level DFD, as shown in the following figure, the major processes of the system areidentified, the data stores are defined and the interaction of the processes with the actors, andthe data stores are established.In the system, three processes can be identified, which are –

Register Customers

Process Sales

Ascertain GiftsThe data stores that will be required are −

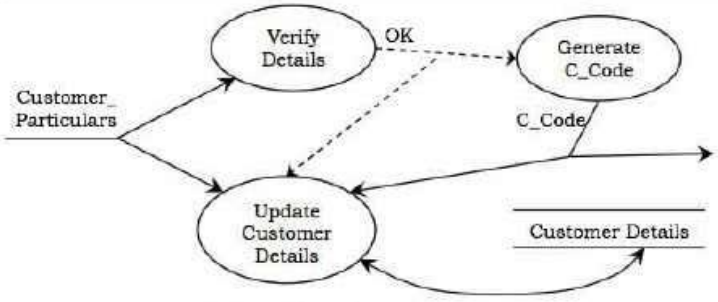
Customer Details

Sales Details

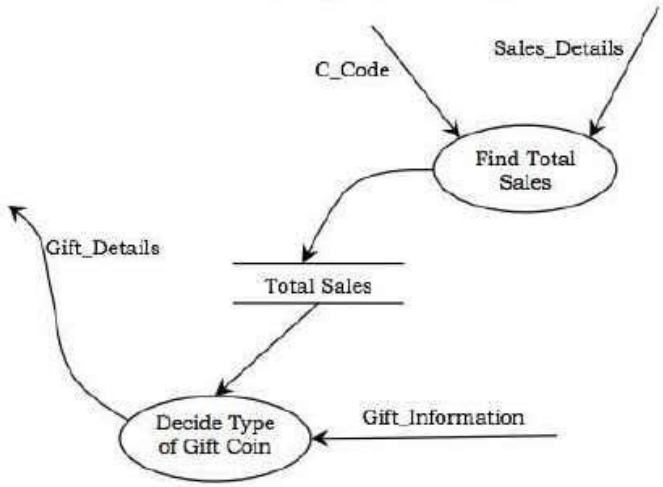
Gift Details

The following figure shows the details of the process Register Customer. There are three processes in it, Verify Details, Generate C_Code, and Update Customer Details. When t hedetails of the customer are entered, they are verified. If the data is correct, C_Code is generatedand the data store Customer Details is updated.

APEC



DFD of Customer Process



DFD of Gift Process

T he following figure shows the expansion of the process Ascertain Gifts. It has two processesin it, Find Total Sales and Decide Type of Gift Coin. The Find Total Sales process computesthe yearly total sales corresponding to each customer and records the data. Taking this recordand the decision of the proprietor as inputs, the gift coins are allotted through Decide Type of Gift Coin process.

Advantages and Disadvantages of DFDAdvantagesDisadvantages

DFDs depict the boundaries of a system andhence are helpful in portraying the relationship between the external objects and the processeswithin the system.DFDs take a long time to create, which may not be feasible for practical purposes.They help the users to have a knowledge aboutthe system.DFDs do not provide any information about thetime-dependent behavior, i.e., they do notspecify when the transformations are done.The graphical representation serves as a blueprint for the programmers to develop aThey do not throw any light on the frequency ofcomputations or the reasons for computations.

43

system.DFDs provide detailed information about thesystem processes.The preparation of DFDs is a complex processthat needs considerable expertise. Also, it isdifficult for a non-technical person tounderstand.They are used as a part of the systemdocumentation.The method of preparation is subjective andleaves ample scope to be imprecise.

Relationship between Object, Dynamic, and Functional Models

The Object Model, the Dynamic Model, and the Functional Model are complementary to eachother for a complete Object-Oriented Analysis.

Object modelling develops the static structure of the software system in terms of objects. Thus it shows the “doers” of a system.

Dynamic Modelling develops the temporal behavior of the objects in response toexternal events. It shows the sequences of operations performed on the objects.

Functional model gives an overview of what the system should do.

Functional Model and Object Model

The four main parts of a Functional Model in terms of object model are −

Process

− Processes imply the methods of the objects that need to be implemented.

Actors

− Actors are the objects in the object model.

Data Stores

− These are either objects in the object model or attributes of objects.

Data Flows

− Data flows to or from actors represent operations on or by objects. Dataflows to or from data stores represent queries or updates.

Functional Model and Dynamic Model

The dynamic model states when the operations are performed, while the functional modelstates how they are performed and which arguments are needed. As actors are active objects,the dynamic model has to specify when it acts. The data stores are passive objects and theyonly respond to updates and queries; therefore, the dynamic model need not specify when theyact.

Object Model and Dynamic Model

The dynamic model shows the status of the objects and the operations performed on theoccurrences of events and the subsequent changes in states. The state of the object as a result of the changes is shown in the object model.

APEC