Practical no : 01

AIM :- SRS

Software Requirement Specifications

The production of the requirements stage of the software development process is **Software Requirements Specifications (SRS)** (also called a **requirements document**). This report lays a foundation for software engineering activities and is constructing when entire requirements are elicited 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 establish different 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, constraints, attributes, or external interfaces.

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

Note: It is essential to specify the responses to both valid and invalid values.

(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 requirements described 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 in another 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 may determine that the program will multiply them.

(b) One condition may state that "A" must always follow "B," while other requires that "A and B" co-occurs.

(3). Two or more requirements may define the same real-world object but use different terms for 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 promotes consistency.

4. Unambiguousness: SRS is unambiguous when every fixed requirement has only one interpretation. 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. The requirements 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 it facilitates the referencing of each condition in future development or enhancement documentation.

There are two types of Traceability:

1. Backward Traceability: This depends upon each requirement explicitly referencing its source 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 those modifications.

9. Design Independence: There should be an option to select from multiple design alternatives for 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 test cases 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 how to do these. This means that the SRS document should define the external behavior of the system 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 merely understand it. Response to undesired events: It should characterize acceptable responses to unwanted events. These are called system response to exceptional conditions.

Verifiable: All requirements of the system, as documented in the SRS document, should be correct. This means that it should be possible to decide whether or not requirements have been met in an implementation.

Practical no : 02

AIM :- Justification for selection of suitable model

Question: Explain the various types of models which used in software Engineering.

Answer:

There are multiple models used in software development and models are choosed based on the application requirement. For Example, in the small project waterfall method can be used and for the big size projects spiral method is preferred. We selects the model for the software Engineering on following basis:

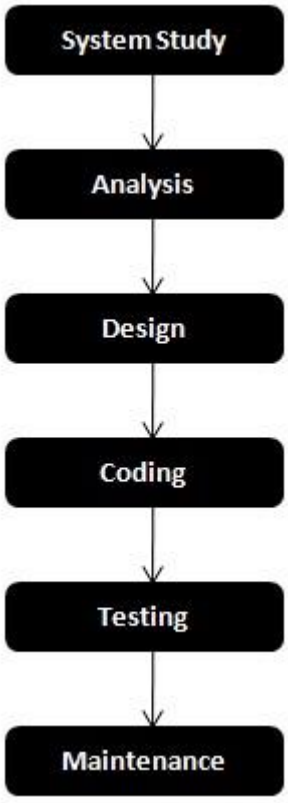
- Application and projects nature
- Use of techniques and methods

- Necessary control and dispatch

Details of various models are given below:

Waterfall methodology

This method is very old and still trendy which is normally used because simplicity is key of success of this method. In this method every action taken is planned in a process of software development. For the small size of projects waterfall method is very suitable. The specialty of this method is that every stage has a sequence. After the end of one stage, the next stage started at that point and the output of every stage converted in input for the every next steps. So with this reason the second name of this model is Sequence Model. With the study about system the first stage started and after that analysis, design, coding, testing and controlling take place one by one. Every stage completed one by one in a sequence like waterfall so this is the reason of this name.



Benefits

- With the reason of simplicity and sequence system it is very easy to use it.
- User can understand it quickly because simplicity is the main quality of this method.
- It defines the complete information about every step.
- This model is very helpful in planning and scheduling of projects.
- It reduces the cost of Error correctness.
- Importantly it increases the possibility that the system fulfill the customer needs.

Limitations

- Expectations become very quickly in this process which is not correct.
- Risk factors not involved in this model which is very necessary part for every model.
- In this method training sessions of user is not necessary.
- We cannot find any error till time testing of software not done.

Prototype Model

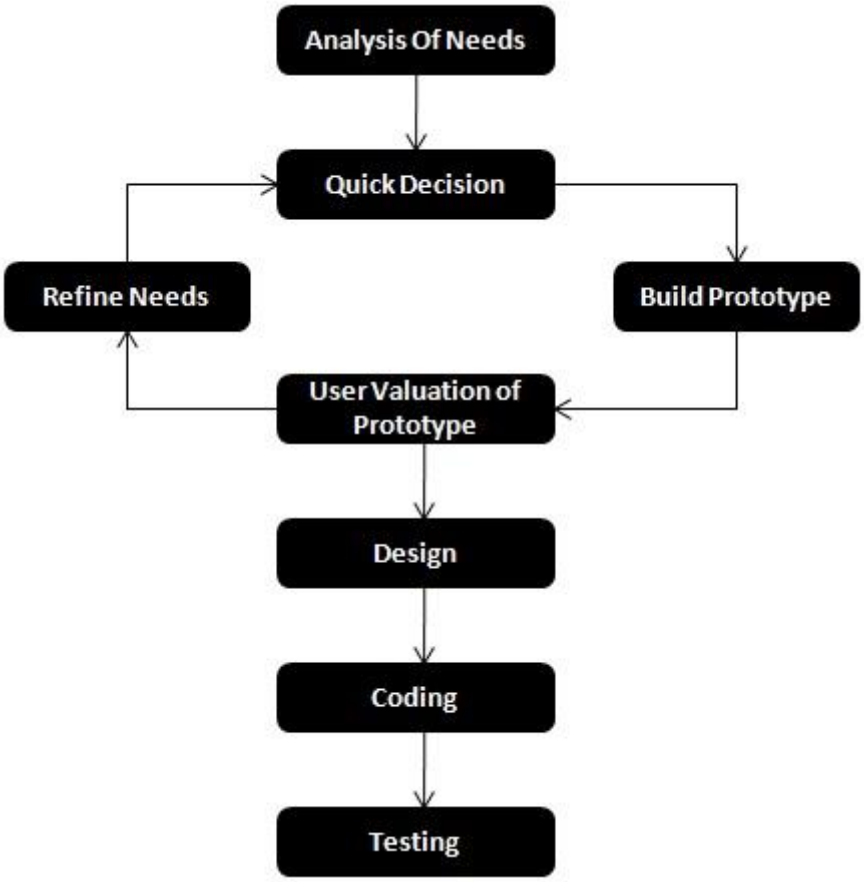
This is a working system and the objective of the model is to develop those ideas who tested regarding a new system which could be used again. Prototyping is processes of making a model for the system which can be develop. The main part of this model is before the design and coding we could not fix the need of software while we could think about the needs of customer from software system. The prototype model made on the basis of knowledge about the needs of customer. Prototype model provide the design which gives the thoughts how system did the works. The objective of prototype is to make good understanding about system needs. At the last we get that such result which will be very low variable.

The prototype after the process of development, the end user gives the chance to take the benefit of prototype and provide the important thoughts about this developer.

- What is good.
- Area of alteration
- Missing information
- Requirement which is not necessary

After getting the feedback necessary alteration made in prototype model and provides it again to user for using it. This approach is suitable in that case where two below given points covered.

- When a little amount of needs should know at the starting time.
- When a customer wants a short area testing instead of fully software.



Benefits

- This model is more suitable where customer needs not cleared.
- This Type of model use in that area where the size of problem are large.
- It decreases the cost of maintenance. The main factor of using this model is its cost reductions.
- This model is very helpful in decrease the gap of communication between those persons who use the software and that group of developed the software system.
- In the comparison of final software alteration in prototyping model alteration is quickly and cheaper.

Limitations

- To get the quick work if may possible that it ignore the quality.
- It can increase the cost of software system because most of time design and code not used.
- Expectations of users increase from the software after looking the working process of prototype model.
- When the result of prototyping is not according to customer needs in that case they can make negative thought about that system.

Integrative Enhancement Model

This model is combination of waterfall and prototyping model. The objective of this model is to start a system on a low level with needs which is necessary and introduces it as a trial in the form of version no. 1. After that getting the point of view of user some changes are made and software takes the shape of version no.2. Then this process repeated by developer one by one with necessary changes and every time a new version became ready for introducing in market. This step started with a low level which is called subsystem and after that step by step taken and a series of versions launched in market. This model has four steps:

1. **Step of needs** - In this step a list is made about the necessary items.
2. **Design phase** - The design should be according to requirement if it satisfied the needs then it could be new or existing one.
3. **Implementation phase** - In this phase, the process of software development is implemented.
4. **Valuation phase** - All types of valuations are done on this phase.

Spiral Model

Spiral model is introduced in 1986 by Boehm. For the lack of risk factor many models became fail but Spiral model include risk factor because Boehm know that the future is uncertain. In this model various types of activity take place like a spiral that has many cycles. Each cycle have four steps.

1. **Planning** - This is about the objective and getting the substitutes resources.
2. **Risk analysis** - Alternate of valuation and know about the factors which solve the Risk factors.
3. **Development** - Development for the next level products.
4. **Customer Valuation** - customer evaluates the prototyping and other models.

During the 1st cycle:

1. To get the motive of the product and make planning for the substitution.
2. Various risks are analyzed.
3. Making of prototype model.
4. The valuation of prototype are done by the user.

During 2nd cycle:

1. Valuation by customer is the base of planning.
2. Reaction of customer is the base of analysis of risk factors.
3. Make a best prototype model.
4. Involving customers for accessing a new prototype model method.

The focus of third and fourth cycle is to make an error free prototype model and to make easy risk resolution with better planning. Each stage makes a better version of software with a little time consumption. The cost is representing by the radius of spiral at every point of the project.

Benefits

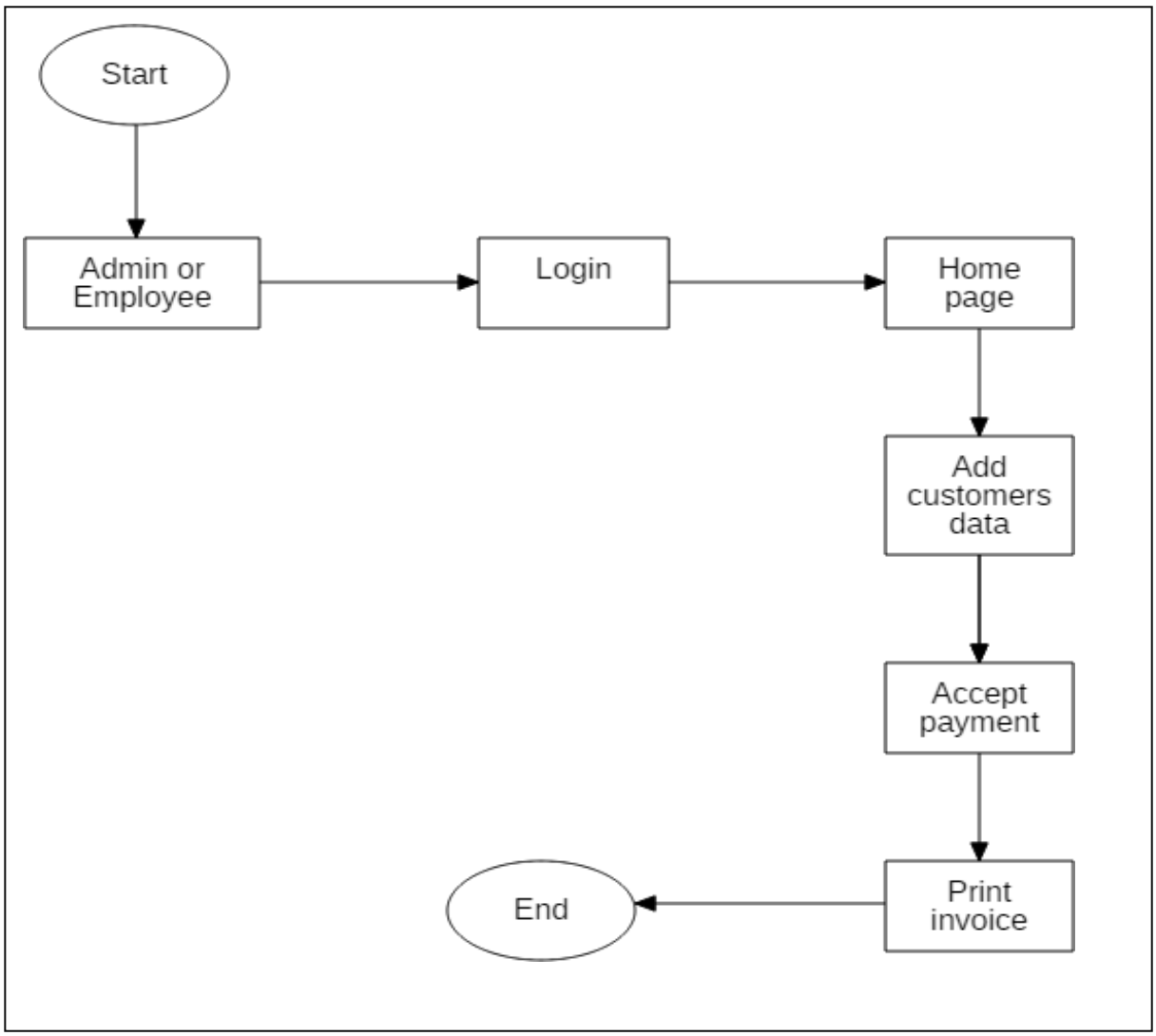
- By the evaluation of customer the each cycle of the spiral became complete.
- It gives the capacity for quick development.
- It has the Quality of reality and works for the both development type.

Practical no : 03

AIM :- DFD

Data-flow diagram :-

A Data-flow diagram(DFD) is a way of representing a flow of a data of a process or a system. The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops.



Practical no : 04

AIM :- Cost estimation using COCOMO 1

COCOMO 1

- It is used in waterfall model of software development cycle.
- It helps give estimates required for the efforts and schedule.
- It is based on the linear reuse formula.
- It is based on the assumption of reasonable stable requirements.

- The effort equation has an exponent that is figured out using 3 development modes.
- The development begins after the requirements are assigned to software.
- The number of sub-models are 3.
- It has 15 cost drivers assigned to it.
- The size of the software is measured in terms of lines of code.

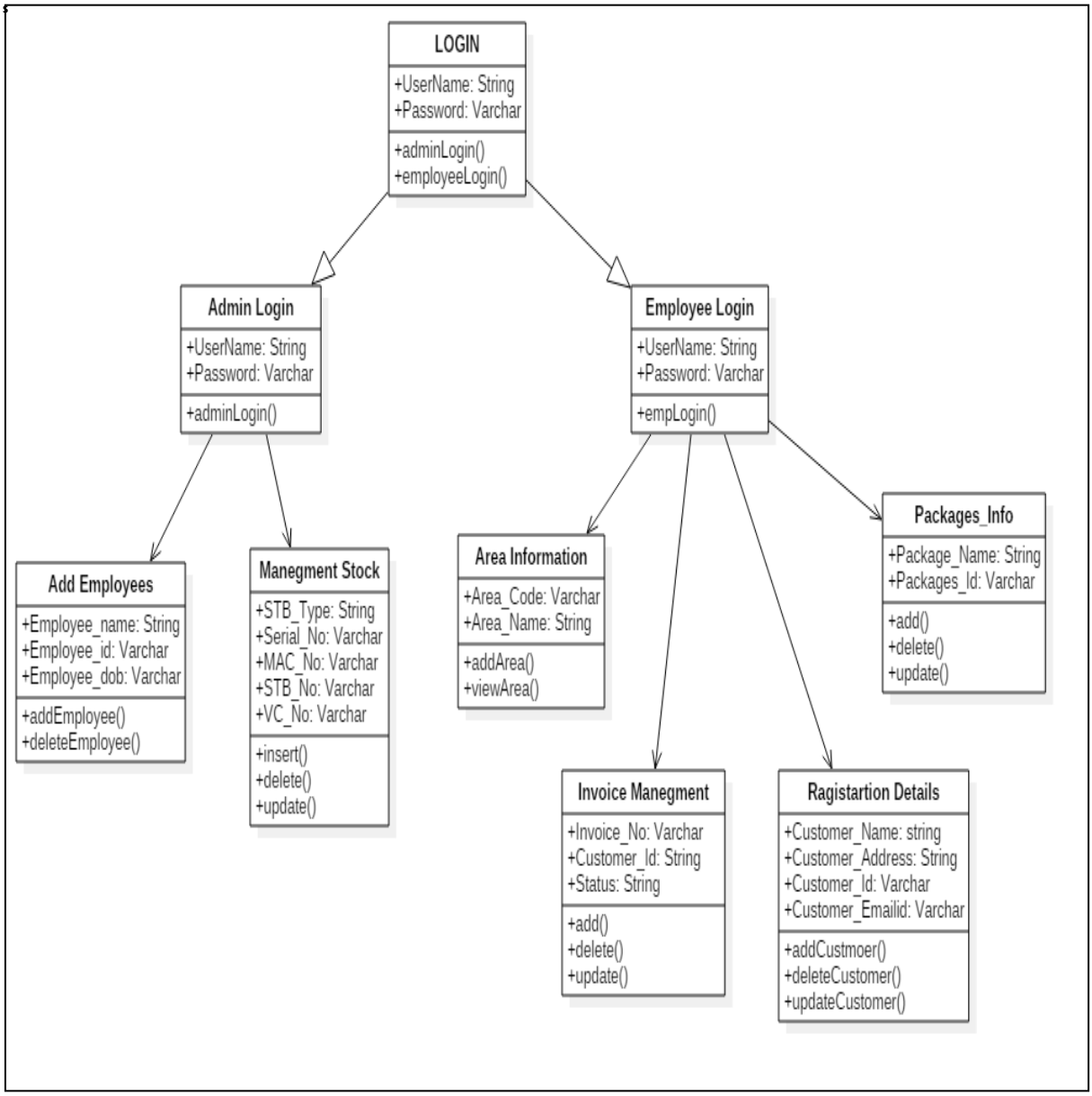
Practical no : 05

AIM :- UML Diagrams 1– Class Diagram, Use Case Diagram (Use STARUML software)

Class diagram :-

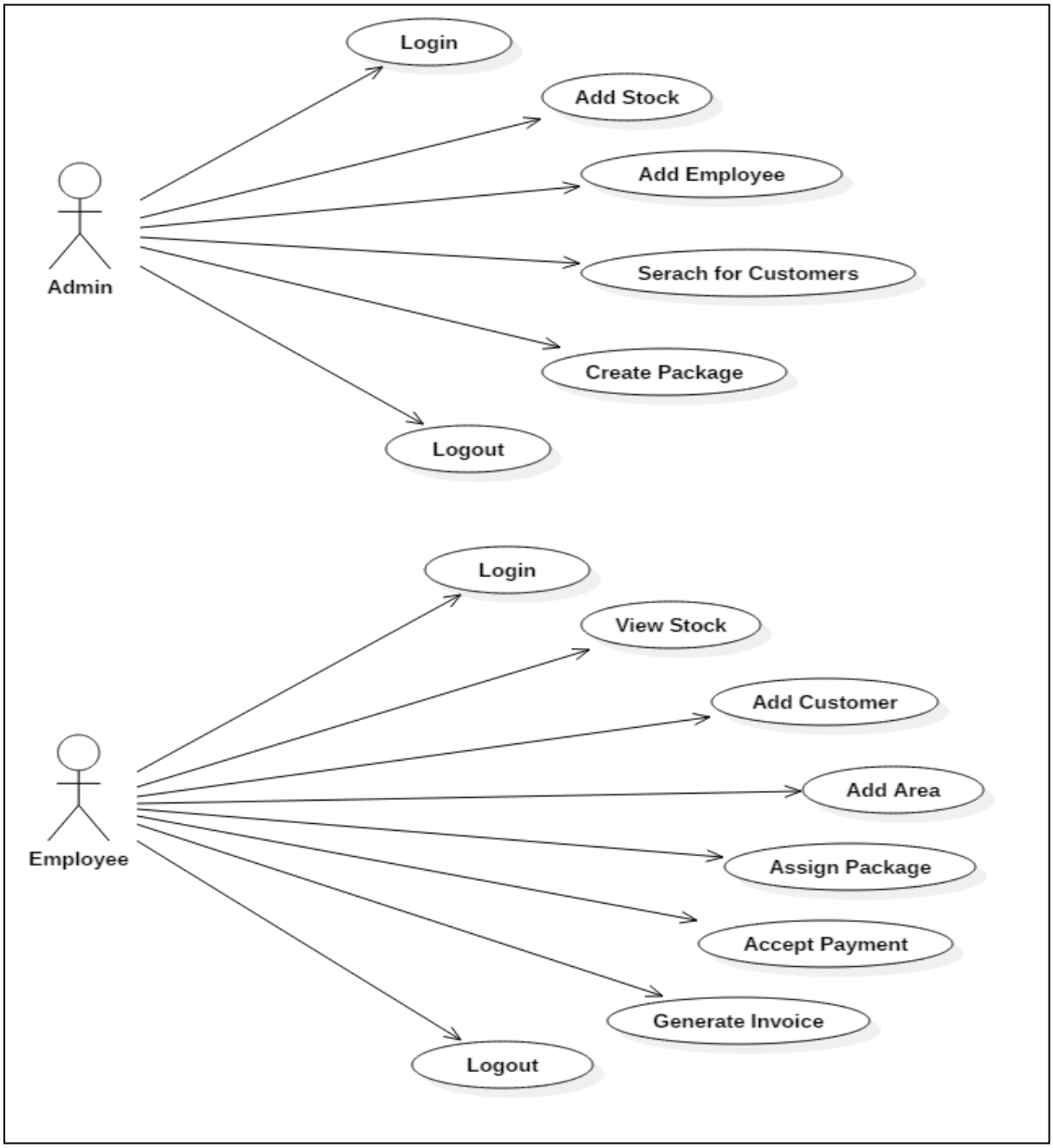
A Class diagram is the Unified Modeling Language (UML) is type of static structure diagram that describes the attributes and the constraints impose on system. It shows the collection of classes, interface, association, collaboration and constraints. They are used to show the different objects in a system, their attributes, their operations and the relationships among them. The following diagram represent the Cable Operator Management System with different operations and attributes, including the parent class login. The login class has the two child classes which is Admin login and Employee login. These both class have identical attributes and operations.

The below diagram consist of classes. The Login class is consist of two variable namely Username and Password. Login class having two child class is namely Admin Login and Employee Login. The Admin Login class is parent class of Add Employee and Management Stock. The Employee Login class also have two child class namely Area Information And Package_info. The Employee Login class is also parent class of two class i.e Invoice management and Registration Details class.



Use case diagram :-

A use case diagram at its simplest is a representation of user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by the other types of diagrams as well. The actors(end-users) involved in the use cases, a use case diagram and the detail use case description are provided. The use cases that find representation are admin and employee. The Admin can add, update, delete details of employees, stocks, etc. The employee can login to the system and edit customer information, generate invoice for customers.



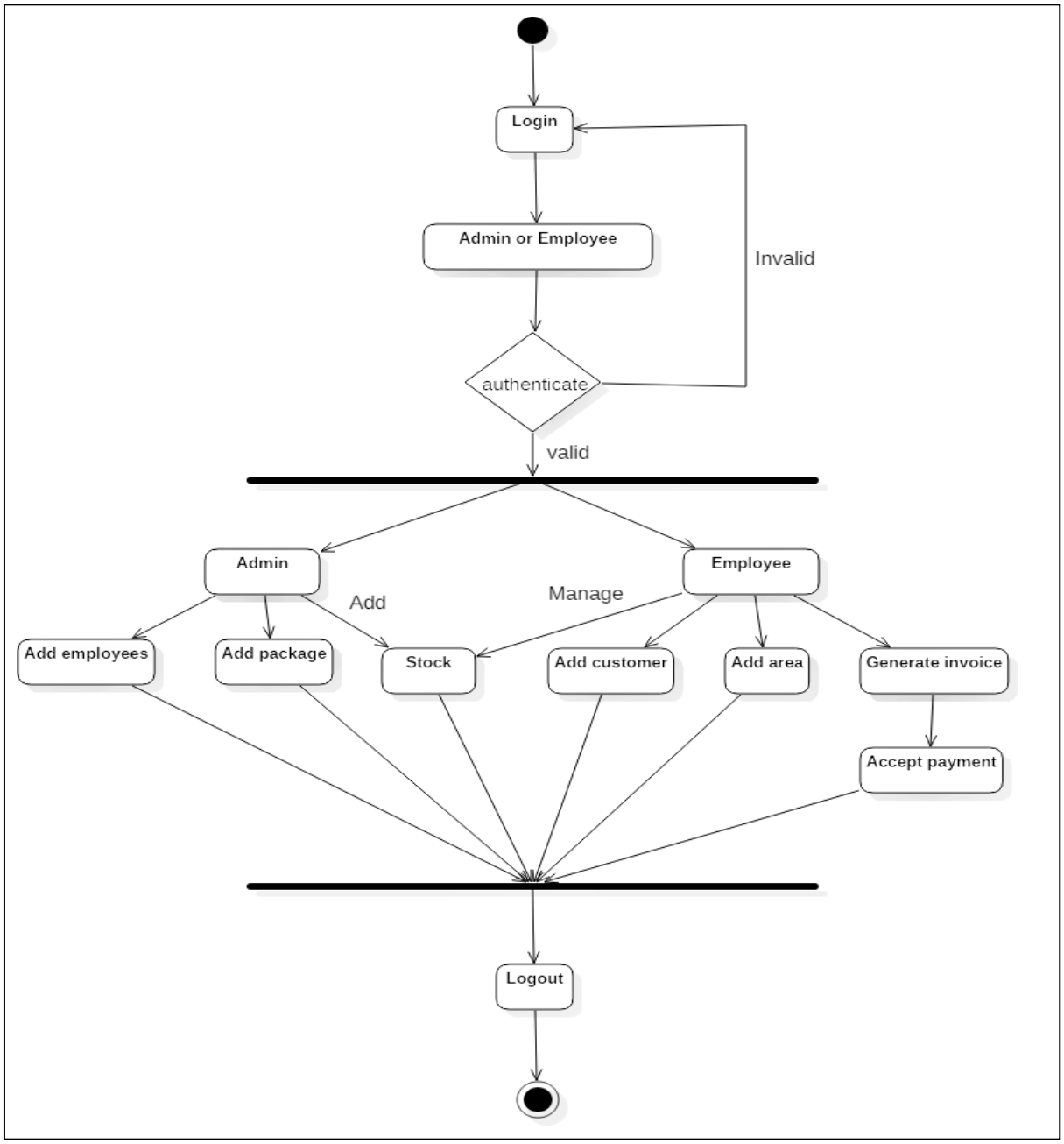
Practical no : 06

AIM :-UML Diagrams 2 – Activity Diagram, Sequence Diagram, Collaboration Diagram (Use STARUML software)

Activity diagram :-

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the

system. The control flow is drawn from one operation to another. Activities model can be sequential and concurrent. In both cases an activity diagram will have a beginning & an end. In between there are ways to depict activities, flows, decisions, guards, merge and time events and more.



From the term Interaction, it is clear that the diagram is used to describe some type of interactions among the different elements in the model. This interaction is a part of dynamic behavior of the system.

This interactive behavior is represented in UML by two diagrams known as **Sequence diagram** and **Collaboration diagram**. The basic purpose of both the diagrams are similar.

Sequence diagram emphasizes on time sequence of messages and collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.

Purpose of Interaction Diagrams

The purpose of interaction diagrams is to visualize the interactive behavior of the system. Visualizing the interaction is a difficult task. Hence, the solution is to use different types of models to capture the different aspects of the interaction.

Sequence and collaboration diagrams are used to capture the dynamic nature but from a different angle.

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.

How to Draw an Interaction Diagram?

As we have already discussed, the purpose of interaction diagrams is to capture the dynamic aspect of a system. So to capture the dynamic aspect, we need to understand what a dynamic aspect is and how it is visualized. Dynamic aspect can be defined as the snapshot of the running system at a particular moment

We have two types of interaction diagrams in UML. One is the sequence diagram and the other is the collaboration diagram. The sequence diagram captures the time sequence of the message flow from one object to another and the collaboration diagram describes the organization of objects in a system taking part in the message flow.

Following things are to be identified clearly before drawing the interaction diagram

- Objects taking part in the interaction.
- Message flows among the objects.
- The sequence in which the messages are flowing.
- Object organization.

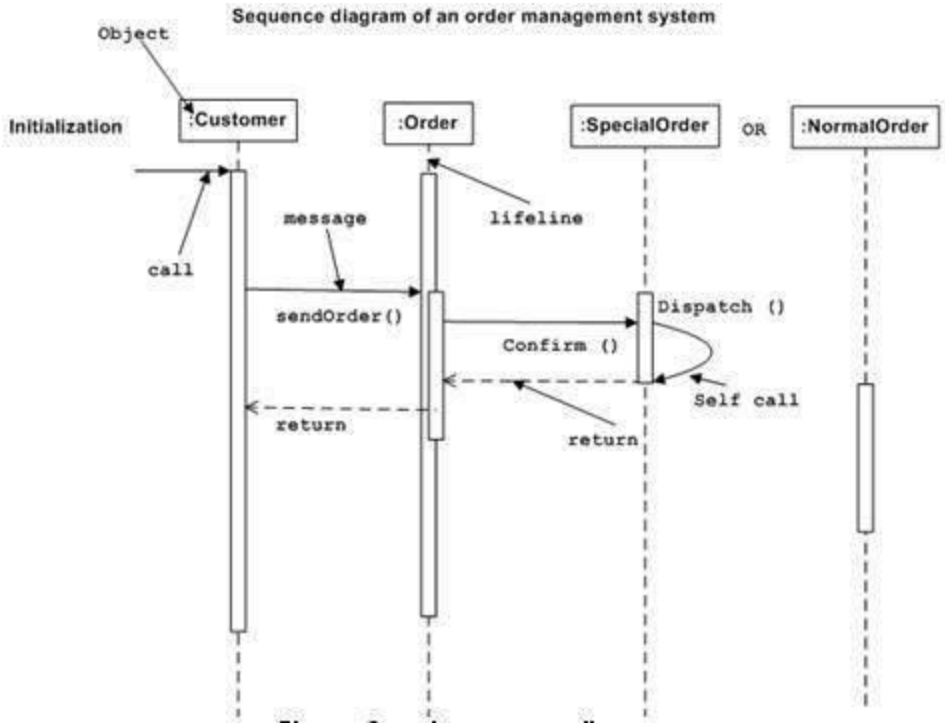
Following are two interaction diagrams modeling the order management system. The first diagram is a sequence diagram and the second is a collaboration diagram

The Sequence Diagram

The sequence diagram has four objects (Customer, Order, SpecialOrder and NormalOrder).

The following diagram shows the message sequence for *SpecialOrder* object and the same can be used in case of *NormalOrder* object. It is important to understand the time sequence of message flows. The message flow is nothing but a method call of an object.

The first call is *sendOrder ()* which is a method of *Order object*. The next call is *confirm ()* which is a method of *SpecialOrder* object and the last call is *Dispatch ()* which is a method of *SpecialOrder* object. The following diagram mainly describes the method calls from one object to another, and this is also the actual scenario when the system is running.

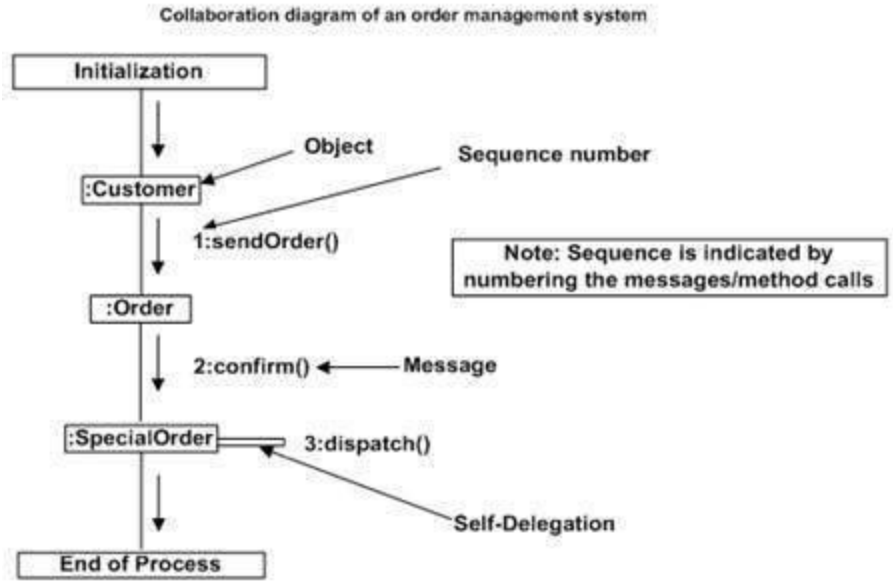


The Collaboration Diagram

The second interaction diagram is the collaboration diagram. It shows the object organization as seen in the following diagram. In the collaboration diagram, the method call sequence is indicated by some numbering technique. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram.

Method calls are similar to that of a sequence diagram. However, difference being the sequence diagram does not describe the object organization, whereas the collaboration diagram shows the object organization.

To choose between these two diagrams, emphasis is placed on the type of requirement. If the time sequence is important, then the sequence diagram is used. If organization is required, then collaboration diagram is used.



Where to Use Interaction Diagrams?

We have already discussed that interaction diagrams are used to describe the dynamic nature of a system. Now, we will look into the practical scenarios where these diagrams are used. To understand the practical application, we need to understand the basic nature of sequence and collaboration diagram.

The main purpose of both the diagrams are similar as they are used to capture the dynamic behavior of a system. However, the specific purpose is more important to clarify and understand.

Sequence diagrams are used to capture the order of messages flowing from one object to another. Collaboration diagrams are used to describe the structural organization of the objects taking part in the interaction. A single diagram is not sufficient to describe the dynamic aspect of an entire system, so a set of diagrams are used to capture it as a whole.

Interaction diagrams are used when we want to understand the message flow and the structural organization. Message flow means the sequence of control flow from one object to another. Structural organization means the visual organization of the elements in a system.

Interaction diagrams can be used −

- To model the flow of control by time sequence.
- To model the flow of control by structural organizations.
- For forward engineering.
- For reverse engineering.

Practical no : 07

AIM :-
What is meant by software testing? What are its types? Which are
the tools used for testing?

Testing Approach :-

Software Testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. It can also provide an objective, independent view of the software to allow the business to appreciate and understand the risk of software implementation. The basic purpose of testing is to detect the errors that may be present in the program. Testing as the process of executing a program with the intent of finding errors.

The Box Approach :-

Software testing methods are traditionally divided into white-box and black-box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test cases.

1) White-box Testing :-

White box testing also known as clear testing, glass testing, and transparent box testing and structural testing. In white box testing an internal perspective of the system, as well as programming skills, are used to design test case. The testers choose inputs to exercise paths through the code and determine the appropriate outputs. While white-box testing can be applied at the unit, integration and system levels of the software testing process, it usually done at the unit level.

2) Black-box Testing :-

Black box testing treats the software as a “black box”, examining functionally without any knowledge of internal implementation, without seeing the source code. The testers are only aware of what the software is supposed to do, not how it does it. Black box Testing methods include :

equivalence portioning, boundary value analysis, all-pairs testing state transition tables, decision table testing, fuzz testing , model –based testing, use-case testing, exploratory testing and specification-based testing.

Levels of Testing :-

The levels of testing are as follows :

1) Unit Testing

2) Integration Testing

3) System Testing

1. **Unit Testing :-**

Unit testing focuses verification efforts on the smallest unit of the software design, the module. This is also known as "Module Design". This testing carried out during programming stage itself. In this testing each module is found to be working satisfactorily as regard to the expected output from the module. All textboxes are having validation by which they will not remain empty and all work properly as expected. To store mobile number then the length must be 10 digits and only digit is allowed.

2. Integration Testing :-

Integration testing is systematic testing for construction the program structure while at the same time conducting tests to uncover errors associated with in the interface. The objective is to take unit tested modules and build a program structure. All the modules are combined and tested as a whole. Here correction is difficult because the isolation of cause is complicated by the vast expense of the entire program. In Integration testing I test the system by combining all modules. All the customer data including his id, name, package that assign to customer and total amount show togetherly and invoice gets generate smoothly.

3. System Testing :-

It is the stage of implementation that is aimed at ensuring that the system works accurately and efficiently for live operation commences. Testing is vital to the success of the system. System testing makes a logical assumption that if all the parts of the system are correct, then goal will be successfully achieved.

Practical no : 08

AIM :- What is meant by quality assurance?

What is Quality Assurance?

Quality Assurance is defined as the auditing and reporting procedures used to provide the stakeholders with data needed to make well-informed decisions.

It is the Degree to which a system meets specified requirements and customer expectations. It is also monitoring the processes and products throughout the SDLC.

Quality Assurance Criteria:

Below are the Quality assurance criteria against which the software would be evaluated against:

- correctness
- efficiency
- flexibility
- integrity
- interoperability
- maintainability
- portability
- reliability
- reusability
- testability
- usability

The IEEE definition for software quality assurance is as follows −

"A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements. A set of activities designed to evaluate the process by which the products are developed or manufactured."

Objectives of SQA Activities

The objectives of SQA activities are as follows −

In Software development (process-oriented)

- Assuring an acceptable level of confidence that the software will conform to functional technical requirements.
- Assuring an acceptable level of confidence that the software will conform to managerial scheduling and budgetary requirements.
- Initiating and managing activities for the improvement and greater efficiency of software development and SQA activities.

In Software maintenance (product-oriented)

- Assuring with an acceptable level of confidence that the software maintenance activities will conform to the functional technical requirements.
- Assuring with an acceptable level of confidence that the software maintenance activities will conform to managerial scheduling and budgetary requirements.
- Initiating and managing activities to improve and increase the efficiency of software maintenance and SQA activities. This involves improving the prospects of achieving functional and managerial requirements while reducing costs.

Organizing for Quality Assurance

The quality assurance organizational framework that operates within the organizational structure includes the following participants −

Managers

- Top management executives, especially the executive directly in charge of software quality assurance
- Software development and maintenance department managers
- Software testing department managers
- Project managers and team leaders of development and maintenance projects
- Leaders of software testing teams

Testers

- Members of software testing teams

SQA professionals and interested practitioners −

- SQA trustees
- SQA committee members
- SQA forum members
- SQA unit team members

Only the managers and employees of the software testing department are occupied full time in the performance of SQA tasks. The others dedicate part of their time to quality issues, whether during fulfilment of their managerial functions or professional tasks, or as volunteers in others, most often a SQA committee, a SQA forum, or as SQA trustees.