Roll No.: 1

What is the composition of SRS.

Components of the **SRS**

Functional requirements specify what output should be produced from the given inputs. Performance Requirements (Speed Requirements) This part of an **SRS** specifies the performance constraints on the software system. Design Constraints. External Interface Requirements

Non-Functional Requirements.

Any one function of your own project was to be explained with its i/p and o/p.
Explain where (on which level) in the DFD diagram is the above function.

Roll No.: 2

Group of 2 called, our questions:

All questions revolved around SRS only basically. Kuch aur exp /topic pe nahi pucha.

1) What design approach is used in software engineering?

Software design is a process to conceptualize the software requirements into software implementation. Software design takes the user requirements as challenges and tries to find optimum solution. While the software is being conceptualized, a plan is chalked out to find the best possible design for implementing the intended solution.

There are multiple variants of software design. Let us study them briefly:

Structured Design

Structured design is a conceptualization of problem into several well-organized elements of solution. It is basically concerned with the solution design. Benefit of structured design is, it gives better understanding of how the problem is being solved. Structured design also makes it simpler for designer to concentrate on the problem more accurately.

Structured design is mostly based on ‘divide and conquer’ strategy where a problem is broken into several small problems and each small problem is individually solved until the whole problem is solved.

The small pieces of problem are solved by means of solution modules. Structured design emphasis that these modules be well organized in order to achieve precise solution.

These modules are arranged in hierarchy. They communicate with each other. A good structured design always follows some rules for communication among multiple modules, namely -

Cohesion - grouping of all functionally related elements.

Coupling - communication between different modules.

A good structured design has high cohesion and low coupling arrangements.

Function Oriented Design

In function-oriented design, the system is comprised of many smaller sub-systems known as functions. These functions are capable of performing significant task in the system. The system is considered as top view of all functions.

Function oriented design inherits some properties of structured design where divide and conquer methodology is used.

This design mechanism divides the whole system into smaller functions, which provides means of abstraction by concealing the information and their operation.. These functional modules can share information among themselves by means of information passing and using information available globally.

Another characteristic of functions is that when a program calls a function, the function changes the state of the program, which sometimes is not acceptable by other modules. Function oriented design works well where the system state does not matter and program/functions work on input rather than on a state.

Design Process

- The whole system is seen as how data flows in the system by means of data flow diagram.
- DFD depicts how functions changes data and state of entire system.
- The entire system is logically broken down into smaller units known as functions on the basis of their operation in the system.
- Each function is then described at large.

Object Oriented Design

Object oriented design works around the entities and their characteristics instead of functions involved in the software system. This design strategies focuses on entities and its characteristics. The whole concept of software solution revolves around the engaged entities.

Let us see the important concepts of Object Oriented Design:

- **Objects -** All entities involved in the solution design are known as objects. For example, person, banks, company and customers are treated as objects. Every entity has some attributes associated to it and has some methods to perform on the attributes.
- **Classes -** A class is a generalized description of an object. An object is an instance of a class. Class defines all the attributes, which an object can have and methods, which defines the functionality of the object.

  In the solution design, attributes are stored as variables and functionalities are defined by means of methods or procedures.

- **Encapsulation -** In OOD, the attributes (data variables) and methods (operation on the data) are bundled together is called encapsulation. Encapsulation not only bundles important information of an object together, but also restricts access of the data and methods from the outside world. This is called information hiding.
- **Inheritance -** OOD allows similar classes to stack up in hierarchical manner where the lower or sub-classes can import, implement and re-use allowed variables and methods from their immediate super classes. This property of OOD is known as inheritance. This makes it easier to define specific class and to create generalized classes from specific ones.

- **Polymorphism** - OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned same name. This is called polymorphism, which allows a single interface performing tasks for different types. Depending upon how the function is invoked, respective portion of the code gets executed.

Design Process

Software design process can be perceived as series of well-defined steps. Though it varies according to design approach (function oriented or object oriented, yet It may have the following steps involved:

- A solution design is created from requirement or previous used system and/or system sequence diagram.
- Objects are identified and grouped into classes on behalf of similarity in attribute characteristics.
- Class hierarchy and relation among them is defined.
- Application framework is defined.

Software Design Approaches

Here are two generic approaches for software designing:

Top Down Design

We know that a system is composed of more than one sub-systems and it contains a number of components. Further, these sub-systems and components may have their on set of sub-system and components and creates hierarchical structure in the system.

Top-down design takes the whole software system as one entity and then decomposes it to achieve more than one sub-system or component based on some characteristics. Each sub-system or component is then treated as a system and decomposed further. This process keeps on running until the lowest level of system in the top-down hierarchy is achieved.

Top-down design starts with a generalized model of system and keeps on defining the more specific part of it. When all components are composed the whole system comes into existence.

Top-down design is more suitable when the software solution needs to be designed from scratch and specific details are unknown.

Bottom-up Design

The bottom up design model starts with most specific and basic components. It proceeds with composing higher level of components by using basic or lower level components. It keeps creating higher level components until the desired system is not evolved as one single component. With each higher level, the amount of abstraction is increased.

Bottom-up strategy is more suitable when a system needs to be created from some existing system, where the basic primitives can be used in the newer system.

Both, top-down and bottom-up approaches are not practical individually. Instead, a good combination of both is used.

2) What SRS comprises of? Explain in detail.

A **software requirements specification** (SRS) is a detailed description of a software system to be developed with its functional and non-functional requirements. The SRS is developed based the agreement between customer and contractors. It may include the use cases of how user is going to interact with software system. The software requirement specification document consistent of all necessary requirements required for project development. To develop the software system we should have clear understanding of Software system. To achieve this we need to continuous communication with customers to gather all requirements.
A good SRS defines the how Software System will interact with all internal modules, hardware, communication with other programs and human user interactions with wide range of real life scenarios. Using the *Software requirements specification* (SRS) document on QA lead, managers creates test plan. It is very important that testers must be cleared with every detail specified in this document in order to avoid faults in test cases and its expected results.
It is highly recommended to review or test SRS documents before start writing test cases and making any plan for testing. Let's see how to test SRS and the important point to keep in mind while testing it.

3) Does SRS contain classes and implementation parts??

Yes

4) What are the non functional requirements? Explain in detail.

NON-FUNCTIONAL REQUIREMENT (NFR) specifies the quality attribute of a software system. They judge the software system based on Responsiveness, Usability, Security, Portability and other non-functional standards that are critical to the success of the software system. Example of nonfunctional requirement, *"how fast does the website load?"* Failing to meet non-functional requirements can result in systems that fail to satisfy user needs.

Non-functional Requirements allows you to impose constraints or restrictions on the design of the system across the various agile backlogs. Example, the site should load in 3 seconds when the number of simultaneous users are > 10000. Description of non-functional requirements is just as critical as a functional requirement.

- Usability requirement
- Serviceability requirement
- Manageability requirement
- Recoverability requirement
- Security requirement
- Data Integrity requirement
- Capacity requirement
- Availability requirement
- Scalability requirement
- Interoperability requirement
- Reliability requirement
- Maintainability requirement

- Regulatory requirement
- Environmental requirement

Examples of Non-functional requirements

Here, are some examples of non-functional requirement:

1. Users must change the initially assigned login password immediately after the first successful login. Moreover, the initial should never be reused.
2. Employees never allowed to update their salary information. Such attempt should be reported to the security administrator.
3. Every unsuccessful attempt by a user to access an item of data shall be recorded on an audit trail.
4. A website should be capable enough to handle 20 million users with affecting its performance
5. The software should be portable. So moving from one OS to other OS does not create any problem.
6. Privacy of information, the export of restricted technologies, intellectual property rights, etc. should be audited.

Functional vs Non Functional Requirements

Parameters	**Functional Requirement**	**Non-Functional Requirement**
What is it?	Verb	Attributes
Requirement	It is mandatory	It is non-mandatory
Capturing type	It is captured in use case.	It is captured as a quality attribute.
End-result	Product feature	Product properties
Capturing	Easy to capture	Hard to capture
Objective	Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Area of focus	Focus on user requirement	Concentrates on the user's expectation.
Documentation	Describe what the product does	Describes how the product works
Type of Testing	Functional Testing like System, Integration, End to End, API testing, etc.	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc.
Test Execution	Test Execution is done before non-functional testing.	After the functional testing
Product Info	Product Features	Product Properties

Advantages of Non-Functional Requirement

Benefits/pros of Non-functional testing are:

- The nonfunctional requirements ensure the software system follow legal and compliance rules.
- They ensure the reliability, availability, and performance of the software system
- They ensure good user experience and ease of operating the software.
- They help in formulating security policy of the software system.

Disadvantages of Non-functional requirement

Cons/drawbacks of Non-function requirement are:

- None functional requirement may affect the various high-level software subsystem
- They require special consideration during the software architecture/high-level design phase which increases costs.
- Their implementation does not usually map to the specific software sub-system,
- It is tough to modify non-functional once you pass the architecture phase.

5) What are the functional requirements? Explain in detail.(take any one function, explain that in details, what is the i/p and o/p of that function, to elaborate the i/p o/p part, DFD was asked to be made, explain the level which you are drawing, and how it relates to your problem statement, don't present the DFD, you just need to draw it on a paper, and explain it verbally)

A **Functional Requirement** (FR) is a description of the service that the software must offer. It describes a software system or its component. A function is nothing but inputs to the software system, its behavior, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform. Functional Requirements are also called **Functional Specification**.

In software engineering and systems engineering, a Functional Requirement can range from the high-level abstract statement of the sender's necessity to detailed mathematical functional requirement specifications. Functional software requirements help you to capture the intended behaviour of the system.

What should be included in the Functional Requirements Document?

Functional Requirement No.	Function Requirement Description
FR 1	User should be able to enter Sales Data
FR 2	Sales Reports should be generated every 24 hours
FR 3	API interface to Invoice System

Example Functional Requirements

Functional Requirements should include the following things:

- Details of operations conducted in every screen
- Data handling logic should be entered into the system
- It should have descriptions of system reports or other outputs
- Complete information about the workflows performed by the system
- It should clearly define who will be allowed to create/modify/delete the data in the system
- How the system will fulfill applicable regulatory and compliance needs should be captured in the functional document

Benefits of Functional Requirement

Here, are the pros/advantages of creating a typical functional requirement document-

- Helps you to check whether the application is providing all the functionalities that were mentioned in the functional requirement of that application
- A functional requirement document helps you to define the functionality of a system or one of its subsystems.
- Functional requirements along with requirement analysis help identify missing requirements. They help clearly define the expected system service and behavior.
- Errors caught in the Functional requirement gathering stage are the cheapest to fix.
- Support user goals, tasks, or activitie

Roll no 3:

who are the actors in your system

An actor in use case modeling specifies a role played by a user or any other system that interacts with the subject. An Actor models a type of role played by an entity that interacts with the subject (**e.g.**, by exchanging signals and data), but which is external to the subject.

How do you gather requirements for srs

Requirement Gathering

If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user. Analysts and engineers communicate with the client

and end-users to know their ideas on what the software should provide and which features they want the software to include.

Software Requirement Specification

SRS is a document created by system analyst after the requirements are collected from various stakeholders.

SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.

The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be comprehended and useful by the software development team.

SRS should come up with following features:

- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

How did you get requirements for Ferry System

what are types of actors

Actors can be primary or secondary actors. Primary actors initiate a use case, while secondary actors support a use case or receive something of value from the use case. While this answer might score you some points in the interview, there is another way to classify actors that is important to know and can show that you understand some of the finer points of use case diagramming.

Actors can be:

1. Human
2. Systems/Software
3. Hardware
4. Timer/Clock

Many analysts miss key actors during the use case diagramming process because they only identify human actors. Categorizing use case actors in this ways helps the analyst ensure they haven't overlooked any critical actors within the use case diagram.

how do you represent them

What are the nouns pronouns verbs in the srs

what do they represent

Ishita is a proper noun Student is a noun
what do they mean
explain programmatically
My answer) Class Student

object Ishita
private members name,class,branch
constructor to initialize values
getters setters to get and set values

Roll No. 4:

What are the types of actors, symbolic notation to represent actors?

Design a class diagram (just mention the class names with relationships) for a scenario (in my case, Prathamesh learns SAD).

What is the type of association between the Student and the Course class?

Explain primary and secondary actors.

Primary Actors are actor(s) using the system to achieve a goal. The Use Case documents the interactions between the system and the actors to achieve the goal of the primary actor. Secondary Actors are actors that the system needs assistance from to achieve the primary actor's goal.

Roll No. 5 (Questions asked after Roll No.6):

What is the extension point in the use case diagram? Give example of the same.

An extension point is a feature of a **use case** which identifies (references) a **point** in the behavior of the **use case** where that behavior can be **extended** by some other (**extending**) **use case**, as specified by **extend** relationship. The optional explanation is some description usually **given** as informal text.

Scenario - One student can enroll in a maximum of 5 courses in Management System. How will you represent this in a use case diagram?
Ans: Add use case to check number of courses as <<include>> to the enroll use case

Scenario - If there is a particular deadline for submission of assignment how will you show this in use case diagram?
Ans: Using external entity of Calendar System

Roll No. 6:

What is a use case ?

The purpose of use case diagram is to capture the dynamic aspect of a system. However, this definition is too generic to describe the purpose, as other four diagrams (activity, sequence, collaboration, and Statechart) also have the same purpose. We will look into some specific purpose, which will distinguish it from other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. Hence, when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified.

When the initial task is complete, use case diagrams are modelled to present the outside view.

In brief, the purposes of use case diagrams can be said to be as follows −

- Used to gather the requirements of a system.
- Used to get an outside view of a system.
- Identify the external and internal factors influencing the system.
- Show the interaction among the requirements are actors.

Purpose of Use Case Diagram

Use case diagrams are typically developed in the early stage of development and people often apply use case modeling for the following purposes:

- Specify the context of a system
- Capture the requirements of a system
- Validate a systems architecture
- Drive implementation and generate test cases
- Developed by analysts together with domain experts

What are the use cases in your project ?

What is use case description and its contents

The use case description is a narrative document that describes, in general terms, **the** required functionality of **the use case**. Typically it describes **the use case** goal and gives a general **description** of what usually happens, **the** normal course of events, adding a brief **description** of any minor variations.

What is alternative flow with example ?

An **alternate flow** describes a scenario other than the basic **flow** that results in a user completing his or her goal. It is often considered to be an optional **flow**. It implies that the user has chosen to take an **alternative** path through the system.

What is include and extend with example

Extend is used when a use case conditionally adds steps to another first class use case. For **example**, imagine "Withdraw Cash" is a use case of an ATM machine. ... **Include** is used to extract use case fragments that are duplicated in multiple use cases.

If A includes B, then A contains all of the functionality of B without modifying that functionality of B.

If A extends B, then A takes the functionality of B and will customize it in some way.

If you are familiar with OOP - then ‘includes’ is where class A inherits class B, but does not override any of the methods in class B; while ‘extends’ is where class A inherits class B and overrides at least one method of class B.

Roll No 7(Questions asked after Roll No. 8):

What should be the order of objects in the sequence diagram?

What is difference between activity diagram and sequence diagram

SEQUENCE DIAGRAM	ACTIVITY DIAGRAM
The Sequence diagram represents the UML, which is used to visualize the sequence of calls in a system that is used to perform a specific functionality.	The Activity diagram represents the UML, which is used to model the workflow of a system.
The Sequence diagram shows the message flow from one object to another object.	The Activity diagram shows the message flow from one activity to another.
Sequence diagram is used for the purpose of dynamic modelling.	Activity diagram is used for the purpose of functional modelling.
Sequence diagram is used to describe the behavior of several objects in a single use case	Activity diagrams is used to describe the general sequence of actions for several objects and use cases.
Sequence diagram is mainly used to represent the time order of a process.	Activity diagram is used to represent the execution of the process.

Sr. No.	Key	Sequence Diagram	Activity Diagram
1	Definition	Sequence diagram is the diagram in which main representation is of the sequence of messages flowing from one object to	On other hand Activity diagram is a diagram in which main representation is of the control flowing from one

Sr. No.	Key	Sequence Diagram	Activity Diagram
		another also main emphasis is on representing that how the messages/events are exchanged between objects and in what time-order.	activity to another implementing the logic behind these activity with the use of conditional structures, loops, concurrency, etc.
2	Main focus	Sequence diagram mainly focuses to represent interaction between different objects by pictorial representation of the message flow from one object to another object. It is time ordered means exact interactions between objects is represented step by step.	On other hand Activity diagram focuses to represent the work flow of a system by pictorial representation of the message flow from one activity to another.
3	Type	As Sequence diagram models the sequential logic, ordering of messages with respect to time so it is categorised as Dynamic modelling diagram.	On other hand Activity diagram mainly represents process flows captured in system so it is not classified as Dynamic modelling diagram.
4	UseCase	Sequence diagram as already mentioned is used to describe the behaviour of several objects in a particular single use case with implementation of all possible logical	However on other hand Activity diagrams is used to describe the general sequence of actions for several objects in several use cases.

Sr. No.	Key	Sequence Diagram	Activity Diagram
		conditions and flows.	

Roll No 8:

How many use cases are there in your use case diagram?

In **UML**, there are five **diagrams** available to model the dynamic nature and **use case diagram** is one of them. Now as we have to discuss that the **use case diagram** is dynamic in nature, there should be some internal or external factors for making the interaction. These internal and external agents are known as actors.

What are boundary objects, controller objects and entity objects in a sequence diagram?

Entities are **objects** representing system data: Customer, Transaction, Cart, etc. **Boundaries** are **objects** that interface with system actors: user interfaces, gateways, proxies, etc. **Controllers** are **objects** that mediate between **boundaries** and **entities**. They orchestrate the execution of commands coming from the **boundary**.

Do you always need a controller class in the sequence diagram?

What are the types of messages in the sequence diagram?

We represent **messages** using arrows. Lifelines and **messages** form the core of a **sequence diagram**. Synchronous **messages** – A synchronous **message** waits for a reply before the interaction can move forward. ... Asynchronous **Messages** – An asynchronous **message** does not wait for a reply from the receiver

What is the sequence of object creation in your sequence diagram? (wrt your project)

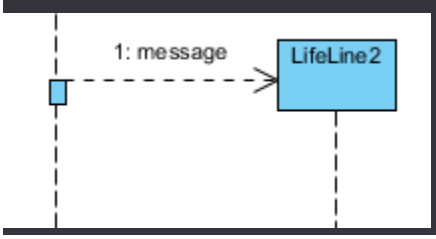
I had 5 use cases in my use case diagram. So he asked me how many sequence diagrams will be there for these use cases?

Difference between Synchronous message and asynchronous message in a sequence diagram.

A **synchronous message** requires a response before the interaction can continue. It's usually drawn using a line **with a** solid arrowhead pointing from one object to another. **Asynchronous messages** don't need a reply **for** interaction to continue.

How is a create message and destroy message represented?

Create Message



Definition

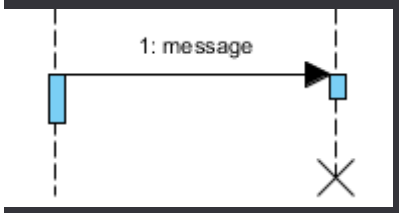
A message defines a particular communication between Lifelines of an Interaction.

Create message is a kind of message that represents the instantiation of (target) lifeline.

Properties

Name	The name of message.
Sequence No.	The number of message indicates the order of message within an interaction.
Documentation	Description of message.
Asynchronous	Determines whether the message is an asynchronous or a synchronous message.

Destroy Message



Definition
A message defines a particular communication between Lifelines of an Interaction.

Destroy message is a kind of message that represents the request of destroying the lifecycle of target lifeline.

Properties

Name	The name of message.
Action Type	Type action type of the message.
Return Value	The information to return to caller.
Sequence No.	The number of message indicates the order of message within an interaction.
Documentation	Description of message.
Asynchronous	Determines whether the message is an asynchronous or a synchronous message.

Roll No. 9

How do you represent association in code?

Association Example

```
class CarClass{
  String carName;
  int carId;
  CarClass(String name, int id)
  {
```

```
            this.carName = name;
            this.carId = id;
    }
}
class Driver extends CarClass{
    String driverName;
    Driver(String name, String cname, int cid){
            super(cname, cid);
            this.driverName=name;
    }
}
class TransportCompany{
    public static void main(String args[])
    {
            Driver obj = new Driver("Andy", "Ford", 9988);
            System.out.println(obj.driverName+" is a driver of car Id: "+obj.carId);
    }
}
```

Output:

```
Andy is a driver of car Id: 9988
```

In the above example, there is a one to one relationship(**Association**) between two classes: CarClass and Driver. Both the classes represent two separate entities.

Although all three are related terms, there are some major differences in the way they relate two classes. **Association** is a relationship between two separate classes and the association can be of any type say one to one, one to may etc. It joins two entirely separate entities.

Aggregation is a special form of association which is a unidirectional one way relationship between classes (or entities), for e.g. Wallet and Money classes. Wallet has Money but money doesn't need to have Wallet necessarily so its a one directional relationship. In this relationship both the entries can survive if other one ends. In our example if Wallet class is not present, it does not mean that the Money class cannot exist.

Composition is a restricted form of Aggregation in which two entities (or you can say classes) are highly dependent on each other. For e.g. Human and Heart. A human needs heart to live and a heart needs a Human body to survive. In other words when the classes (entities) are dependent on each other and their life span are same (if one dies then another one too) then its a composition. Heart class has no sense if Human class is not present.

How do you represent aggregation and composition in code? (Class A is an aggregation of class B, how to write this in java)

Association vs Aggregation vs Composition

1. **Aggregation** implies a relationship where the child can exist independently of the parent. Example: Class (parent) and Student (child). Delete the Class and the Students still exist.

2. **Composition** implies a relationship where the child cannot exist independent of the parent. Example: House (parent) and Room (child).

Example of Composition

```
//Car must have Engine
public class Car {
  //engine is a mandatory part of the car
  private final Engine engine;

  public Car () {
    engine = new Engine();
  }
}

//Engine Object
class Engine {}
```

Example of Aggregation

```
//Team
public class Team {
  //players can be 0 or more
  private List players;

  public Car () {
    players = new ArrayList();
  }
}
//Player Object
class Player {}
```

In restaurants are items and order composition or aggregation?

Roll No. 20 and 21

How do you identify classes?

1. Find the nouns. Read through the problem statement and the associated documentation and highlight the nouns. ...
2. Evaluate the nouns to find **classes**. Questions to ask to evaluate nouns to find **classes**: ...
3. Define the purpose.

How will you identify classes and their relations from a problem statement?

Identifying the Classes in a System

The first step of our OOD process is to identify the classes in our model. We will eventually describe these classes in a formal way and implement them in Java. First, we review the problem statement and locate all the *nouns*; it is likely that these include most of the classes (or instances of classes) necessary to implement the elevator simulation. Figure 3.19 is a list of these nouns (and noun phrases) in the order of their appearance.

Nouns (and noun phrases) in the problem statement		
company	elevator system	graphical user interface (GUI)
office building	elevator shaft	elevator car
elevator	display	person
software-simulator application	model	floor (first floor; second floor)
passenger	bell inside the elevator	**First Floor** GUI button
floor door	light on that floor	**Second Floor** GUI button
user of our application	energy	audio
floor button	capacity	elevator music
elevator button		

What is the next step for making a class diagram?

Class diagrams are the most popular UML diagrams used for construction of software applications. It is very important to learn the drawing procedure of class diagram.

Class diagrams have a lot of properties to consider while drawing but here the diagram will be considered from a top level view.

Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. A collection of class diagrams represent the whole system.

The following points should be remembered while drawing a class diagram −

- The name of the class diagram should be meaningful to describe the aspect of the system.
- Each element and their relationships should be identified in advance.
- Responsibility (attributes and methods) of each class should be clearly identified
- For each class, minimum number of properties should be specified, as unnecessary properties will make the diagram complicated.
- Use notes whenever required to describe some aspect of the diagram. At the end of the drawing it should be understandable to the developer/coder.
- Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.

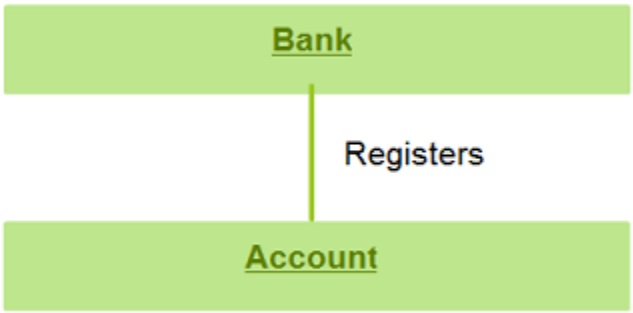
The following diagram is an example of an Order System of an application. It describes a particular aspect of the entire application.

- First of all, Order and Customer are identified as the two elements of the system. They have a one-to-many relationship because a customer can have multiple orders.
- Order class is an abstract class and it has two concrete classes (inheritance relationship) SpecialOrder and NormalOrder.
- The two inherited classes have all the properties as the Order class. In addition, they have additional functions like dispatch () and receive ().

How do you identify the relation between classes?- Here i gave an example of inheritance relation from my project so he asked questions about the inheritance classes from my project here after.

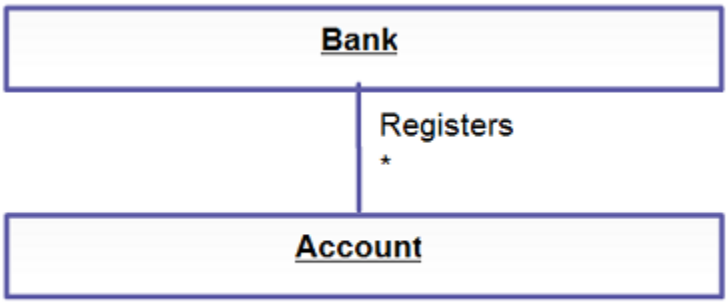
Association

An association relation is established when two classes are connected to each other in any way. For example: A “bank registers account” association can be shown as follows.



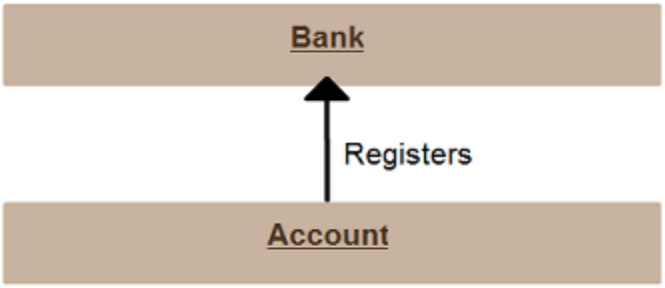
2. Multiplicity

An example of this kind of association is many accounts being registered by the bank. Hence, the relationship shows a star sign near the account class (one to many and many to many etc). When it comes to class diagram relationship this is one of the most misunderstood relationships.



3. Directed Association

By default, an association that exists between classes is bi-directional. Ideally, you may illustrate the flow of the association by utilizing a directed association. The arrowhead indicates the container-contained relationship.

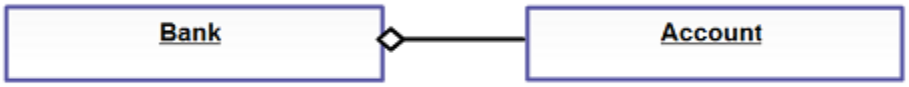


4. Reflexive Association

An example here is when a class has many different types of responsibilities. For example, an employee of a company can be an executive, assistant manager, or a CEO. There is no symbol that can be used here, however, the relation will point back at the same class.

5. Aggregation

When a class is formed as a collection of other classes, it is called an aggregation relationship between these classes. It is also called a "**has a**" relationship.

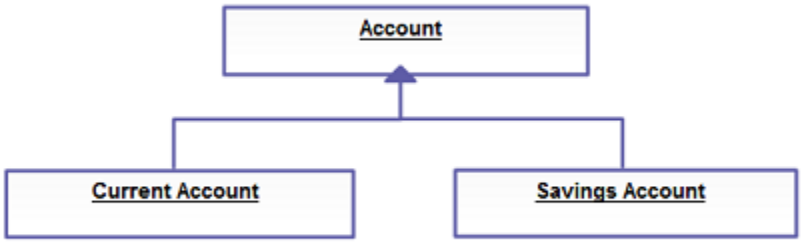


6. Composition

The composition is a variation of the aggregation relationship. Composition illustrates that a strong life cycle is present between the classes. Another class diagram relationship that not many are aware of and few really understand.

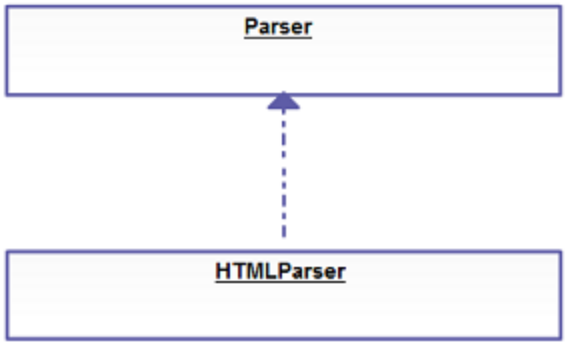
7. Generalization/Inheritance

Known as an "**is a**" relationship since the child class **is a** type of the parent class. Generalization is the ideal type of relationship that is used to showcase reusable elements in the class diagram. Literally, the child classes "inherit" the common functionality defined in the parent class.



8. Realization

In a realization relationship, one entity (normally an interface) defines a set of functionalities as a contract and the other entity (normally a class) "realizes" the contract by implementing the functionality defined in the contract.



What are the base and derived classes?

Base class :

A base class is a class from which other classes are derived in an object-oriented programming language.

It used for creation of other classes that can reuse the code implicitly inherited from the base class (except constructors and destructors).

It is also called as **parent class** or **Super class.**

Properties :

1) Base classes are automatically instantiated before derived classes.

2) Base class members can be accessed from the derived class.

3) The derived class can communicate to the base class during instantiation by calling the base class constructor.

Derived Class :

A derived class is a class derived from another existing class i.e base class.

A derived class acquired the properties of base class.

While inheriting from base class, the derived class implicitly inherits all the members (except constructors and destructors) which it reuses, extends and modifies the behavior of the base class.

It Is also called as **Child class** .

Example : lets take a real life example of tree and mango tree, banana tree

here:

tree is base class

and

mango tree and banana tree are child class.

What are their attributes?
Why did you make separate derived classes?
Can classes be associated with themselves? Give examples.

An association describes discrete connections among objects or other instances in a system. An association relates an ordered list (tuple) of two or more classifiers, with repetitions permitted. The most common kind of association is a binary association between a pair of classifiers. An instance of an association is a link. A link comprises a tuple (an ordered list) of objects, each drawn from its corresponding class. A binary link comprises a pair of objects.

Self Association also called reflexive association, that is, A single object may be associated with itself if the same class appears more than once in an association. If the same class appears twice in an association, the two instances do not have to be the same object, and usually they are not.

Roll No. 22 and 23
(cross-questioned every answer)
What are non functional requirements?
4-5 explained with examples.

NON-FUNCTIONAL REQUIREMENT (NFR) specifies the quality attribute of a software system. They judge the software system based on Responsiveness, Usability, Security, Portability and other non-functional standards that are critical to the success of the software system. Example of nonfunctional requirement, *"how fast does the website load?"* Failing to meet non-functional requirements can result in systems that fail to satisfy user needs.

Non-functional Requirements allows you to impose constraints or restrictions on the design of the system across the various agile backlogs. Example, the site should load in 3 seconds when the number of simultaneous users are > 10000. Description of non-functional requirements is just as critical as a functional requirement.

Here, are some examples of non-functional requirement:

1. Users must change the initially assigned login password immediately after the first successful login. Moreover, the initial should never be reused.
2. Employees never allowed to update their salary information. Such attempt should be reported to the security administrator.
3. Every unsuccessful attempt by a user to access an item of data shall be recorded on an audit trail.
4. A website should be capable enough to handle 20 million users with affecting its performance
5. The software should be portable. So moving from one OS to other OS does not create any problem.

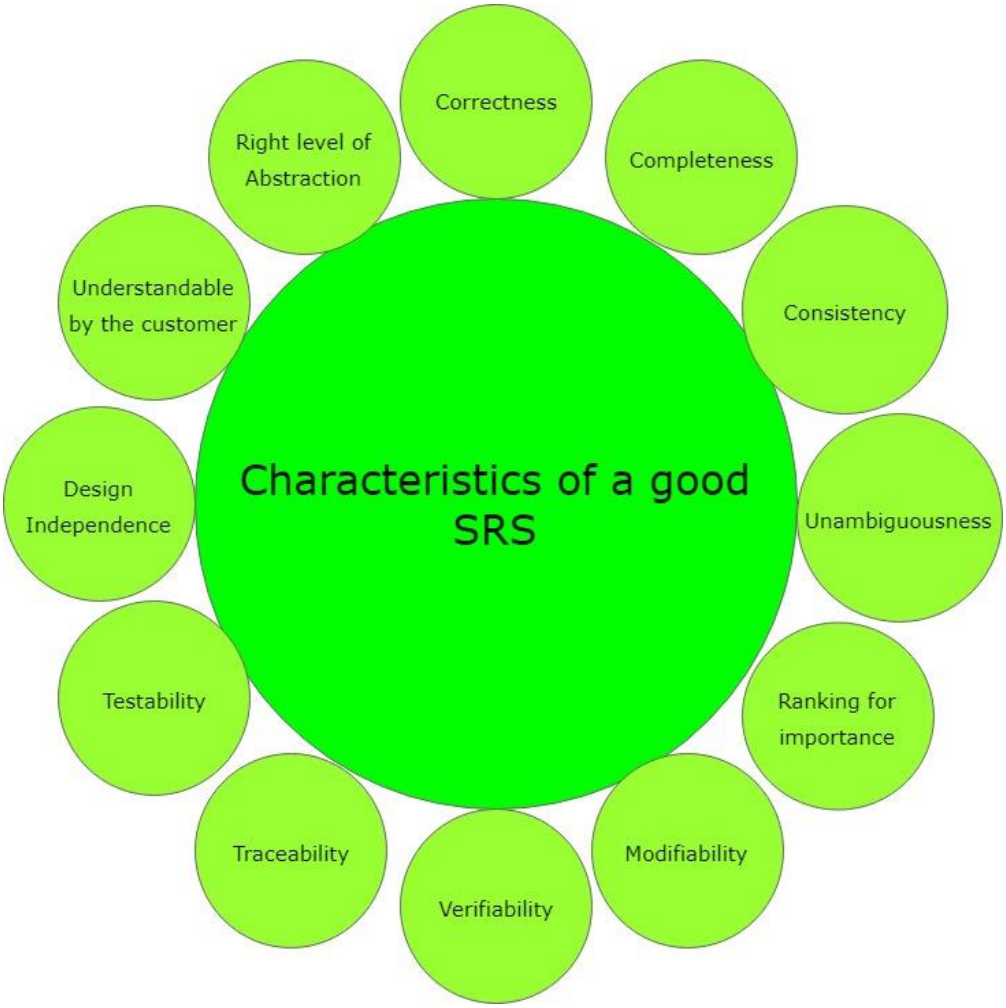
6. Privacy of information, the export of restricted technologies, intellectual property rights, etc. should be audited.

What are qualities of good SRS?
3-4 with examples

Following are the characteristics of a good SRS document:

1. **Correctness:**
   User review is used to ensure the correctness of requirements stated in the SRS. SRS is said to be correct if it covers all the requirements that are actually expected from the system.
2. **Completeness:**
   Completeness of SRS indicates every sense of completion including the numbering of all the pages, resolving the to be determined parts to as much extent as possible as well as covering all the functional and non-functional requirements properly.
3. **Consistency:**
   Requirements in SRS are said to be consistent if there are no conflicts between any set of requirements. Examples of conflict include differences in terminologies used at separate places, logical conflicts like time period of report generation, etc.

4. **Unambiguousness:**
   A SRS is said to be unambiguous if all the requirements stated have only 1 interpretation. Some of the ways to prevent unambiguousness include the use of modelling techniques like ER diagrams, proper reviews and buddy checks, etc.
5. **Ranking for importance and stability:**
   There should a criterion to classify the requirements as less or more important or more specifically as desirable or essential. An identifier mark can be used with every requirement to indicate its rank or stability.
6. **Modifiability:**
   SRS should be made as modifiable as possible and should be capable of easily accepting changes to the system to some extent. Modifications should be properly indexed and cross-referenced.



7. **Verifiability:**
   A SRS is verifiable if there exists a specific technique to quantifiably measure the extent to which every requirement is met by the system. For example, a requirement stating that the system must be user-friendly is not verifiable and listing such requirements should be avoided.
8. **Traceability:**
   One should be able to trace a requirement to design component and then to code segment in the program. Similarly, one should be able to trace a requirement to the corresponding test cases.
9. **Design Independence:**
   There should be an option to choose from multiple design alternatives for the final system. More specifically, the SRS should not include any implementation details.
10. **Testability:**
    A SRS should be written in such a way that it is easy to generate test cases and test plans from the document.
11. **Understandable by the customer:**
    An end user maybe an expert in his/her specific domain but might not be an expert in computer science. Hence, the use of formal notations and symbols should be avoided to as much extent as possible. The language should be kept easy and clear.
12. **Right level of abstraction:**
    If the SRS is written for the requirements phase, the details should be explained

explicitly. Whereas, for a feasibility study, fewer details can be used. Hence, the level of abstraction varies according to the purpose of the SRS.

How to gather information for the SRS?
2-3 techniques

Techniques for Requirements Gathering

An ideal business analyst must follow some techniques for the requirements gathering process. Some of them have been discussed briefly below.

#1 Interview

Interview turns out to be one of the most effective techniques for requirement gathering. In this method, the business analyst talks to the user and clients who are unable to give out detailed information as they are not aware of the system development and related functionalities.

It is the responsibility of the business analyst to extract relevant information from them which can be achieved by interview.

#2 Survey

Survey is another effective method to collect information and requirements within a short frame of time. Under this technique, it is advisable to first ascertain the goal of the survey and thereafter draft the questionnaire.

Once your questions list is ready, it should be delivered to the user as well as the stakeholder for answers. A responsible business analyst would study the answers and then document them.

#3 Brainstorming

SMEs or subject matter experts are the responsible people to conduct brainstorming sessions. They discuss and find out solutions to complex issues.

They are further responsible for requirement prioritization post they collect all the requirements which are related to the software.

#4 Joint Application Method

This method follows a prototyping method, under which all the stakeholders like developers, end users, SMEs, business analysts and software engineers come together and attend workshops for working on a system in greater detail.

These stakeholders attend the workshops till the time the desired goal is accomplished.

#5 Observation

Under the observation method, the responsible person observes the team in working environment and gets ideas about the software and subsequently document the observation.

Observation can be invisible, the person simply observes the working and does not interact or makes himself visible, and hence, the concerned person observes and asks relevant questions.

#6 Focus Group

In this method, the business analyst bases his/her requirement gathering process by interacting with the representative of the client and users.

The representative here has a broad idea pertaining to the needs of the users and clients. Under focus group, the idea is to collect information from representatives to understand the software idea clearly.

#7 Interface Analysis

Interface analysis is a specialized technique in which specific requirements related to application development are determined and their interaction with other software components is measured.

#8 Prototyping

This is a technique of building a model of software which helps in uncovering and capturing software requirements from client. The output can be broad mockups or sketch formats of software.

#9 *Use case diagram*

Use case diagram is a technique that shows how people interact with software. It shows what a system does.

#10 Problem Reports and Suggestion Analysis

Requirements analysis can also be done from change suggestions and user issues. A direct method to look for requirements is to see the suggestions and problems that are described in the document first.

Some organizations have forms to report and record system problems; one can look through such a report and sort the problems into some key areas that are troubling the client.

The users can be asked questions on these to clarify doubts of the users.

Who prepares the SRS?

A person who has experience in software development and professional knowledge in software architecture and applied technologies. The role may vary: business analyst, technical officer, architect, coder.

Who uses the SRS?
Is it publicly available?
Why do they use it?