UNIT-II

Software Requirement Specification (SRS)

Software Requirement Specification (SRS) Format as the name suggests, is a complete specification and description of requirements of the software that need to be fulfilled for the successful development of the software system. These requirements can be functional as well as non-functional depending upon the type of requirement. The interaction between different customers and contractors is done because it is necessary to fully understand the 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.

- **Purpose of this Document** – At first, main aim of why this document is necessary and what's purpose of document is explained and described.
- **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.
- **Overview** – In this, description of product is explained. It's simply summary or overall review of product.

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.

1. **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. Functional requirements specify the expected behaviour of the system-which outputs should be produced from the given inputs. They describe the relationship between the input and output of the system. For each functional requirement, detailed description all the data inputs and their source, the units of measure, and the range of valid inputs must be specified.

2. **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 or explained. Examples can be shared memory, data streams, etc.

3. 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. The performance requirements part of an SRS specifies the performance constraints on the software system. All the requirements relating to the performance characteristics of the system must be clearly specified. There are two types of performance requirements: static and dynamic. Static requirements are those that do not impose constraint on the execution characteristics of the system. Dynamic requirements specify constraints on the execution behaviour of the system.

4. Design Constraints

In this, constraints which simply mean limitation or restriction are specified and explained for design team. Examples may include use of a particular algorithm, hardware and software limitations, etc. There are a number of factors in the client's environment that may restrict the choices of a designer leading to design constraints such factors include standards that must be followed resource limits, operating environment, reliability and security requirements and policies that may have an impact on the design of the system. An SRS should identify and specify all such constraints.

5. 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.

6. 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.

Uses of SRS document

- Development team require it for developing product according to the need.
- Test plans are generated by testing group based on the describe external behaviour.
- Maintenance and support staff need it to understand what the software product is supposed to do.

- Project manager base their plans and estimates of schedule, effort and resources on it.
- customer rely on it to know that product they can expect.
- As a contract between developer and customer.
- in documentation purpose.

❖ use case scenario

A use case scenario describes how a user might interact with a system to achieve a goal. It's typically used in software development as part of the functional requirements document/plan, but use case scenarios aren't just for developers.

Let's take a closer look at what a use case is and how it can help you.

use case

Explaining how a user will interact with a system is a must if you're going to a) get buy-in from stakeholders and b) create something the user wants to use.

Like a user story, a use case describes how a user will interact with a system to achieve a goal but on a slightly more granular level. It's typically written as a sequence of steps, each representing a different action the user takes.

For example, let's say you're designing a new e-commerce website. A use case for this project might be something like this:

- A user goes to the website and browses through the product catalog.
- The user adds a product to their shopping cart.
- The user checks out and pays for the product.
- The system sends a confirmation email to the user.

As you can see, a use case is still quite high-level and isn't meant to be a detailed design document.

create a use case

Use cases are a great way to start a project because they help you understand the user's needs and how the system will fit into the user flow. By creating a use case, you can:

- Gather requirements from users.
- Define the scope of a project.

- Create a roadmap for development.

Everyone from project managers to developers can use them to understand better how a project works and what needs to happen behind the scenes to make it all run smoothly. You can also use them to create a shared understanding of the project among stakeholders.

write a use case

Writing a use case scenario is relatively simple. You just need to answer three questions:

1. Who is going to use the product?
2. What are they going to use it for?
3. How are they going to do it?

Let's say you're designing a new software application. Your first step is to identify the user. In this case, it's someone who wants to book a hotel room.

Next, you need to understand what they want to do with the product. In this case, the user wants to find a hotel room that meets their needs and book it.

Finally, you need to explain how the user will achieve their goal. In this case, they'll use the software to search and book.

include in a use case scenario

The use case can be detailed or basic, depending on the intended audience and system. Either way, the document should establish and identify a few key components:

- System: a system is a collection of hardware, software, and people that work together to achieve a specific goal. In this case, the system is the software application you're designing.
- Actor: an actor is someone who interacts with the system. In this case, the actor is the user who wants to book a hotel room.
- Goal: a goal is something that the actor wants to achieve by interacting with the system. In this case, the goal is to find and book a hotel room.
- Preconditions: preconditions are conditions you need to meet to complete the use case. In our current example, the precondition is the user's need for lodging.

- **Postconditions**: postconditions are conditions that need to be met after completing the use case. In our example, the postcondition is the user confirming a hotel reservation.
- **Extensions**: extensions are alternative sequences of events that can happen during the use case. In this example, an extension might be something like the user leaving the site before booking or inquiring about the room.
- **Use case**: the use case outlines the success and failure scenarios that can happen when the actor interacts with the system. During this section, you'll establish the main success scenario (MSS) and alternative paths that explain what happens in the event of a failure.

create a use case scenario

Creating a use case scenario is a four-step process:

1. Identify the actors.
2. Describe the use case.
3. Outline the success and failure scenarios.
4. Diagram the use case scenario.

Let's go through each of these steps in more detail.

1. Identify the actors

The first step is to identify the different actors that will be interacting with the system. An actor is anyone or anything that interacts with the system. For example, for an e-commerce site, the actors will be the customers. You can make this as granular as you like, such as customers who use TikTok or men in the state of California.

2. Describe the use case

The next step is to describe the use case. This description should be a high-level overview of what the use case is and what it involves.

For example, the description for the customer adding a product to their shopping cart might be: "the customer browses the product catalog and adds a product to their shopping cart."

Look at things from the user's perspective to keep your system customer-focused. A typical format to follow might look like this:

- **As a <type of user or role>, I want <goal> so that <reason>.**

3. Outline the success and failure scenarios

Once you've described the use case, you'll need to outline the success and failure scenarios. A success scenario is a sequence of events that results in the successful completion of a use case. A failure scenario is a sequence of events that prevent the use case from completing successfully.

For example, the success scenario for the customer adding a product to their shopping cart might run as follows:

- The customer browses the product catalog and finds the product they want to buy.
- The customer adds the product to their shopping cart.
- The customer checks out and pays for the product.
- The customer service team ships the product to the customer's address.
- The customer receives the product.

On the other hand, a failure scenario might run like this:

- The customer browses the product catalog and doesn't find the product they want to buy.
- The customer abandons the cart and leaves the website.

Or

- The customer adds a product to their shopping cart.
- The customer checks out and pays for the product.
- The customer decides they don't want the product before it ships.
- The customer service team cancels the order.

As you can see, some failure scenarios are out of your hands. However, your website or app must be prepared to handle all eventualities.

4. Diagram the use case scenario

The last step is to diagram the use case scenario. This will help you visualize the different steps involved in the scenario and see how the actors interact with each other.

You can do this in a few different ways:

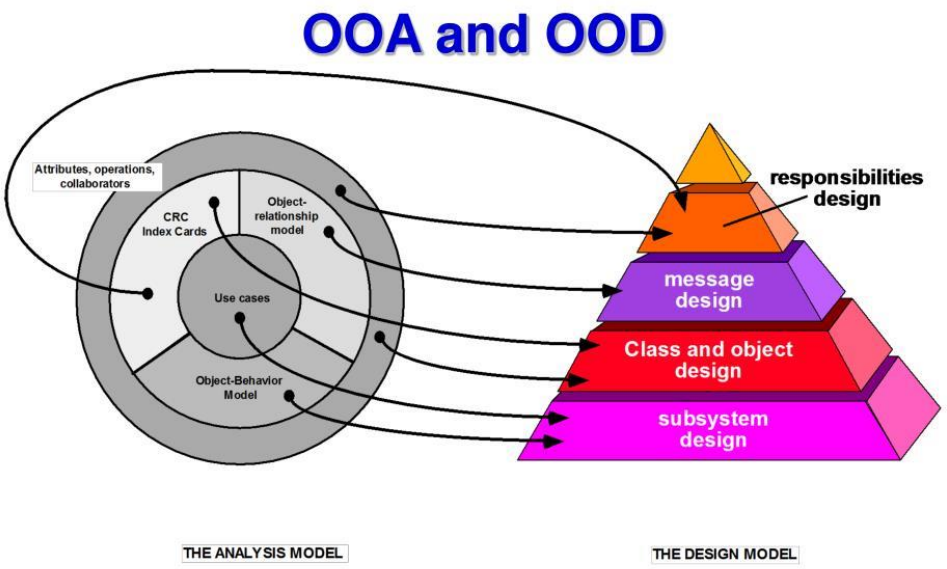
- Drawing sketches by hand
- Using the shapes and lines in word processing or slideshow software

- Using a diagramming tool, like Cacoo (the best option, in our opinion)

Why is diagramming software the best option? You can grab ready-made templates and drag-and-drop shapes to build your diagram. You can also share the use case scenario with the rest of the team in real-time and collaborate in the same file. Not to mention, your entire team can add comments and attach files from one convenient hub.

❖ Object oriented Analysis and Design

Object-oriented analysis and design (OOAD) is a technical approach for analyzing and designing an application, system, or business by applying object-oriented programming, as well as using visual modelling throughout the software development process to guide stakeholder communication and product quality.



OOAD in modern software engineering is typically conducted in an iterative and incremental way.

The outputs of OOAD activities are analysis models (for OOA) and design models (for OOD) respectively.

Object-Oriented Analysis(OOA):

- The purpose of any analysis activity in the software life-cycle is to create a model of the system's functional requirements that is independent of implementation constraints.
- The main difference between object-oriented analysis and other forms of analysis is that by the object-oriented approach we organize requirements around objects, which integrate both behaviors (processes) and states (data) modeled after real world objects that the system interacts with.
- In other or traditional analysis methodologies, the two aspects: processes and data are considered separately. For example, data may be modelled by ER diagrams, and behaviors by flow charts or structure charts.
- Common models used in OOA are use cases and object models. Use cases describe scenarios for standard domain functions that the system must accomplish.
- Object models describe the names, class relations (e.g. Circle is a subclass of Shape), operations, and properties of the main objects. User-interface mockups or prototypes can also be created to help understanding.

Object-Oriented Design(OOD):

- During object-oriented design (OOD), a developer applies implementation constraints to the conceptual model produced in object-oriented analysis.
- Such constraints could include the hardware and software platforms, the performance requirements, persistent storage and transaction, usability of the system, and limitations imposed by budgets and time.
- Concepts in the analysis model which is technology independent, are mapped onto implementing classes and interfaces resulting in a model of the solution domain, i.e., a detailed description of how the system is to be built on concrete technologies.
- Important topics during OOD also include the design of software architectures by applying architectural patterns and design patterns with the object-oriented design principles.

Benefits of Object-Oriented Analysis and Design(OOAD)

- It increases the modularity and maintainability of software by encouraging the creation of tiny, reusable parts that can be combined to create more complex systems.
- It provides a high-level, abstract representation of a software system, making understanding and maintenance easier.
- It promotes object-oriented design principles and the reuse of objects, which lowers the amount of code that must be produced and raises the quality of the program.

- Software engineers can use the same language and method that OOAD provides to communicate and work together more successfully in groups.
- It can assist developers in creating scalable software systems that can adapt to changing user needs and business demands over time.

Challenges of Object-Oriented Analysis and Design(OOAD)

- Because objects and their interactions need to be carefully explained and handled, it might complicate a software system.
- Because objects must be instantiated, managed, and interacted with, this may result in additional overhead and reduce the software's speed.
- For beginner software engineers, OOAD might have a challenging learning curve since it requires a solid grasp of OOP principles and methods.
- It can be a time-consuming process that involves significant upfront planning and documentation. This can lead to longer development times and higher costs.
- OOAD can be more expensive than other software engineering methodologies due to the upfront planning and documentation required.

Real world applications of Object-Oriented Analysis and Design(OOAD)

Some examples of OOAD's practical uses are listed below:

- **Banking Software:** In banking systems, OOAD is frequently used to simulate complex financial transactions, structures, and customer interactions. Designing adaptable and reliable financial apps is made easier by OOAD's modular and scalable architecture.
- **Electronic Health Record (EHR) Systems:** Patient data, medical records, and healthcare workflows are all modeled using OOAD. Modular and flexible healthcare apps that may change to meet emerging requirements can be made through object-oriented principles.
- **Flight Control Systems:** OOAD is crucial in designing flight control systems for aircraft. It helps model the interactions between different components such as navigation systems, sensors, and control surfaces, ensuring safety and reliability.
- **Telecom Billing Systems:** In the telecom sector, OOAD is used to model and build billing systems. It enables the modular and scalable modeling of complex subscription plans, invoicing rules, and client data.
- **Online Shopping Platforms:** E-commerce system development frequently makes use of OOAD. Product catalogs, user profiles, shopping carts, and payment procedures are all modeled, which facilitates platform maintenance and functionality expansion.

❖ Design Patterns in Software Engineering

A Software Design Pattern is a general, reusable solution to a commonly occurring problem within a given context in software design.

Efficient and effective problem-solving is critical in software development. Design patterns are tried-and-true remedies for common problems that arise during the development process. These patterns provide best practices, ideas, and methods that programmers can use to create scalable, reliable, and maintainable software systems.

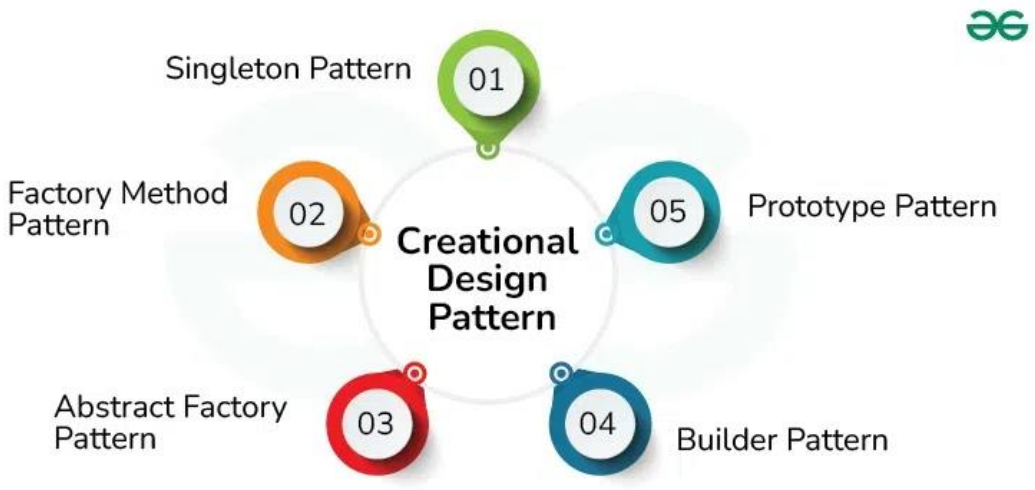
Types of Design Patterns in Software Engineering

Design patterns vary in complexity, level of detail, and scale of applicability to the entire system being designed. Let us take the analogy to road construction: you can make an intersection safer by either installing some traffic lights or building a whole multi-level interchange with underground passages for pedestrians.

As per the book 'Design Patterns - Elements of Reusable Object-Oriented Software,' there are 23 design patterns that can be classified into three categories: Creational, Structural, and Behavioral. Let us look at each one of them in detail.

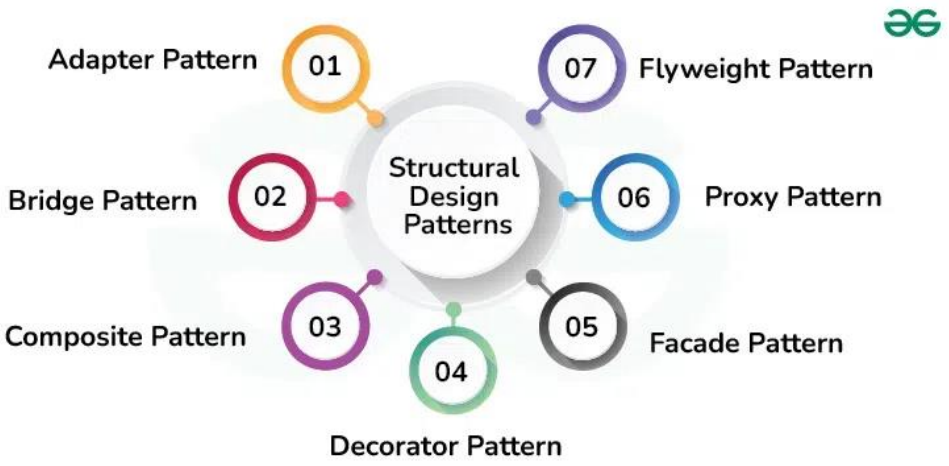
Creational Patterns

These patterns provide various object creation mechanisms, which increase flexibility and reuse of existing code.



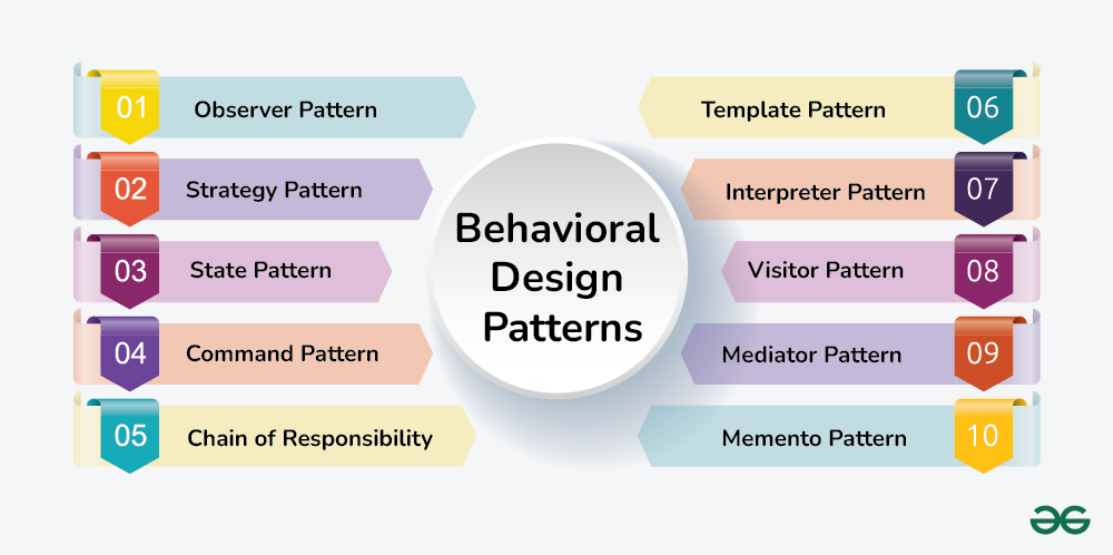
These design patterns are all about instantiating classes or creating objects. These patterns are further divided into two types: **class-creational** patterns and **object-creational** patterns. While class-creation patterns effectively employ inheritance in the instantiation process, object-creation patterns effectively use delegation. The Factory Method, Abstract Factory, Builder, Singleton, Object Pool, and Prototype are examples of creational design patterns.

Structural Patterns



- These patterns provide various object creation mechanisms, which increase flexibility and reuse of existing code.
- These design patterns are concerned with grouping various classes and objects into larger structures that give new functionality. Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Private Class Data, and Proxy are some examples of structural design patterns.

Behavioral Patterns



- Behavioral patterns are concerned with algorithms and the assignment of responsibilities between objects.
- These design patterns are about recognizing and realizing common communication patterns between objects. Chain of duty, Command, Interpreter, Iterator, Mediator, Memento, Null Object, Observer, State, Strategy, Template method, and Visitor are examples of behavioral patterns.

Advantages of Design Patterns

1. Reusable in Multiple Projects

Developers can use design patterns to apply solutions to common issues in a reusable fashion across different projects.

2. Define System Architecture

They provide a lucid and well-organized architecture by offering blueprints for specifying the composition and relationships within a system.

3. Capture Software Engineering Experiences

Software engineers' combined expertise and experience are embodied in design patterns, which provide tried-and-true answers to common design problems.

4. Transparency in Design

The architectural and design choices become more apparent and understandable to other developers when they adhere to recognized design patterns.

5. Well-Proven Solutions

Over time, design patterns have undergone extensive testing and refinement, yielding dependable solutions supported by industry knowledge.

6. Enhance System Flexibility

By separating components and encouraging modular design, they increase flexibility and facilitate system expansion and adaptation.

7. Promote Maintainability

Through the promotion of best practices like loose coupling, encapsulation, and separation of concerns, design patterns help write more maintainable code.

8. Facilitate Better System Design

Design patterns address typical design issues and trade-offs, guiding developers towards better system designs even while they do not provide perfect answers.

❖ Unified Modeling Language (UML)

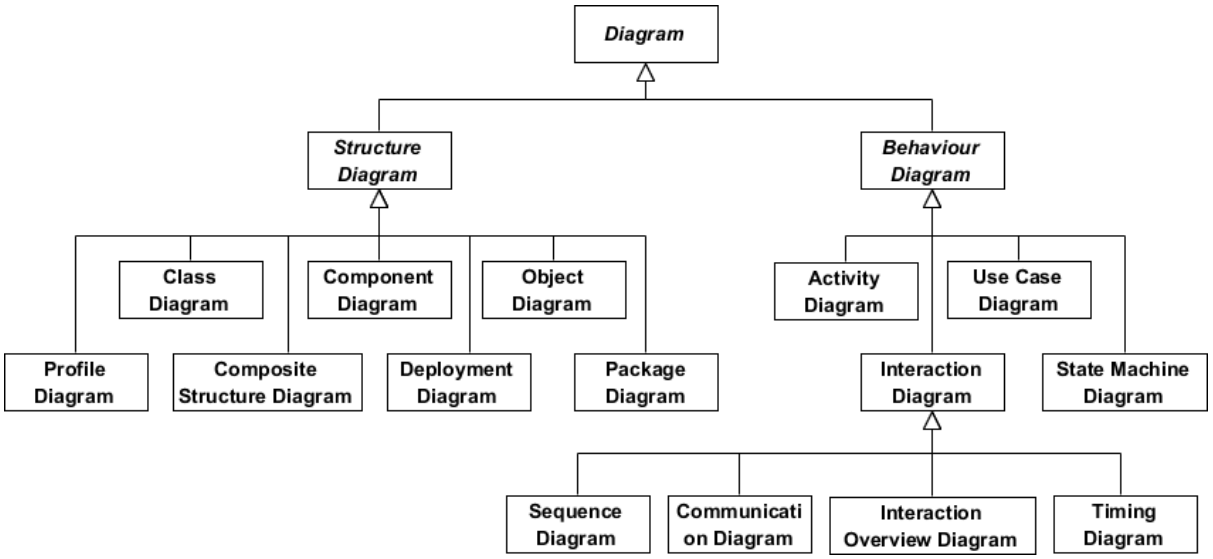
UML, short for Unified Modeling Language, is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing object oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software. In this article, we will give you detailed ideas about what is UML, the history of UML and a description of each UML diagram type, along with UML examples.

Structure diagrams show the static structure of the system and its parts on different abstraction and implementation levels and how they are related to each other. The elements in a structure diagram represent the meaningful concepts of a system, and may include abstract, real world and implementation concepts, there are seven types of structure diagram as follows:

- Class Diagram
- Component Diagram
- Deployment Diagram
- Object Diagram
- Package Diagram
- Composite Structure Diagram
- Profile Diagram

Behavior diagrams show the **dynamic behavior** of the objects in a system, which can be described as a series of changes to the system over **time**, there are seven types of behavior diagrams as follows:

- Use Case Diagram
- Activity Diagram
- State Machine Diagram
- Sequence Diagram
- Communication Diagram
- Interaction Overview Diagram
- Timing Diagram



❖ Class Diagram

The class diagram is a central modeling technique that runs through nearly all object-oriented methods. This diagram describes the types of objects in the system and various kinds of static relationships which exist between them.

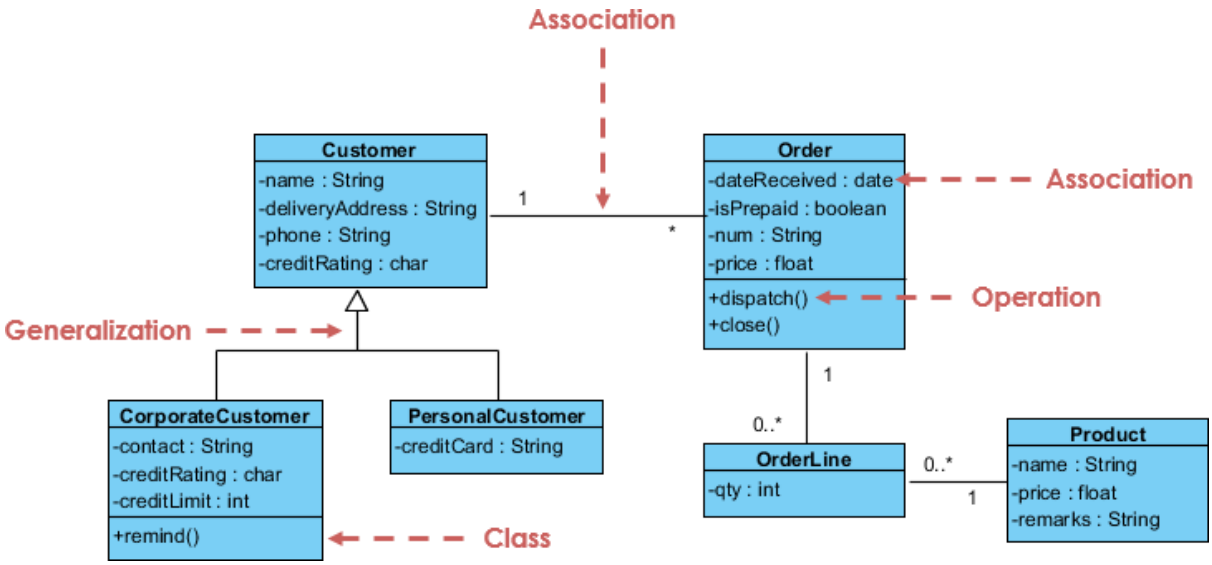
Relationships

There are three principal kinds of relationships which are important:

1. **Association** - represent relationships between instances of types (a person works for a company, a company has a number of offices.
2. **Inheritance** - the most obvious addition to ER diagrams for use in OO. It has an immediate correspondence to inheritance in OO design.

3. **Aggregation** - Aggregation, a form of object composition in object-oriented design.

Class Diagram Example



❖ Object Diagram

An object diagram is a graph of instances, including objects and data values. A static object diagram is an instance of a class diagram; it shows a snapshot of the detailed state of a system at a point in time. The difference is that a class diagram represents an abstract model consisting of classes and their relationships. However, an object diagram represents an instance at a particular moment, which is concrete in nature. The use of object diagrams is fairly limited, namely to show examples of data structure.

Class Diagram vs Object Diagram - An Example

Some people may find it difficult to understand the difference between a UML Class Diagram and a UML Object Diagram as they both comprise of named "rectangle blocks", with attributes in them, and with linkages in between, which make the two UML diagrams look similar. Some people may even think they are the same because in the UML tool they use both the notations for Class Diagram and Object Diagram are put inside the same diagram editor - Class Diagram.

But in fact, Class Diagram and Object Diagram represent two different aspects of a code base. In this article, we will provide you with some ideas about these two UML diagrams, what they are, what are their differences and when to use each of them.

Relationship between Class Diagram and Object Diagram

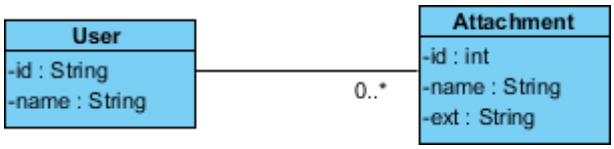
You create "classes" when you are programming. For example, in an online banking system you may create classes like 'User', 'Account', 'Transaction', etc. In a classroom management system you may create classes like 'Teacher', 'Student', 'Assignment', etc. In each class, there are attributes and operations that represent the characteristic and behavior of the class. Class Diagram is a UML diagram where you can visualize those classes, along with their attributes, operations and the inter-relationship.

UML Object Diagram shows how object instances in your system are interacting with each other at a particular state. It also represents the data values of those objects at that state. In other words, a UML Object Diagram can be seen as a representation of how classes (drawn in UML Class Diagram) are utilized at a particular state.

If you are not a fan of those definition stuff, take a look at the following UML diagram examples. I believe that you will understand their differences in seconds.

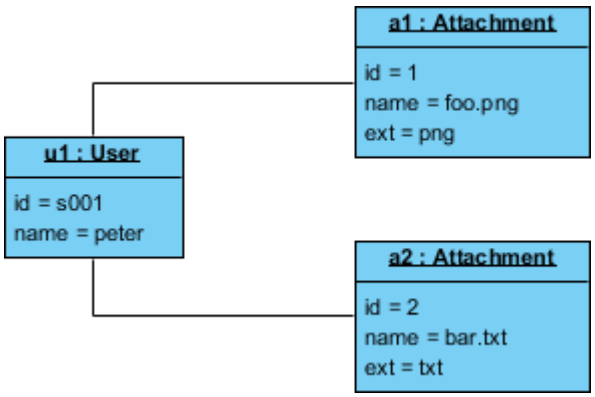
Class Diagram Example

The following Class Diagram example represents two classes - User and Attachment. A user can upload multiple attachment so the two classes are connected with an association, with 0..* as multiplicity on the Attachment side.



Object Diagram Example

The following Object Diagram example shows you how the object instances of User and Attachment class "look like" at the moment Peter (i.e. the user) is trying to upload two attachments. So there are two Instance Specification for the two attachment objects to be uploaded.



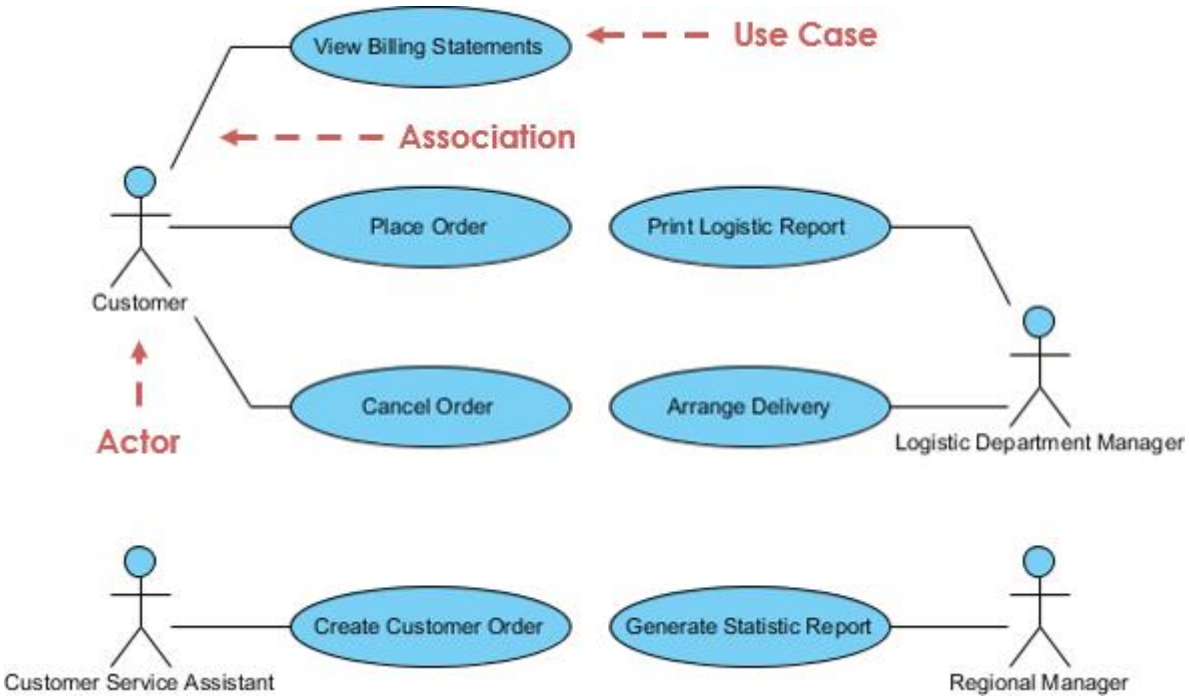
- Use Case Diagram

A use-case model describes a system's functional requirements in terms of use cases. It is a model of the system's intended functionality (use cases) and its environment (actors). Use cases enable you to relate what you need from a system to how the system delivers on those needs.

Think of a use-case model as a menu, much like the menu you'd find in a restaurant. By looking at the menu, you know what's available to you, the individual dishes as well as their prices. You also know what kind of cuisine the restaurant serves: Italian, Mexican, Chinese, and so on. By looking at the menu, you get an overall impression of the dining experience that awaits you in that restaurant. The menu, in effect, "models" the restaurant's behavior.

Because it is a very powerful planning instrument, the use-case model is generally used in all phases of the development cycle by all team members.

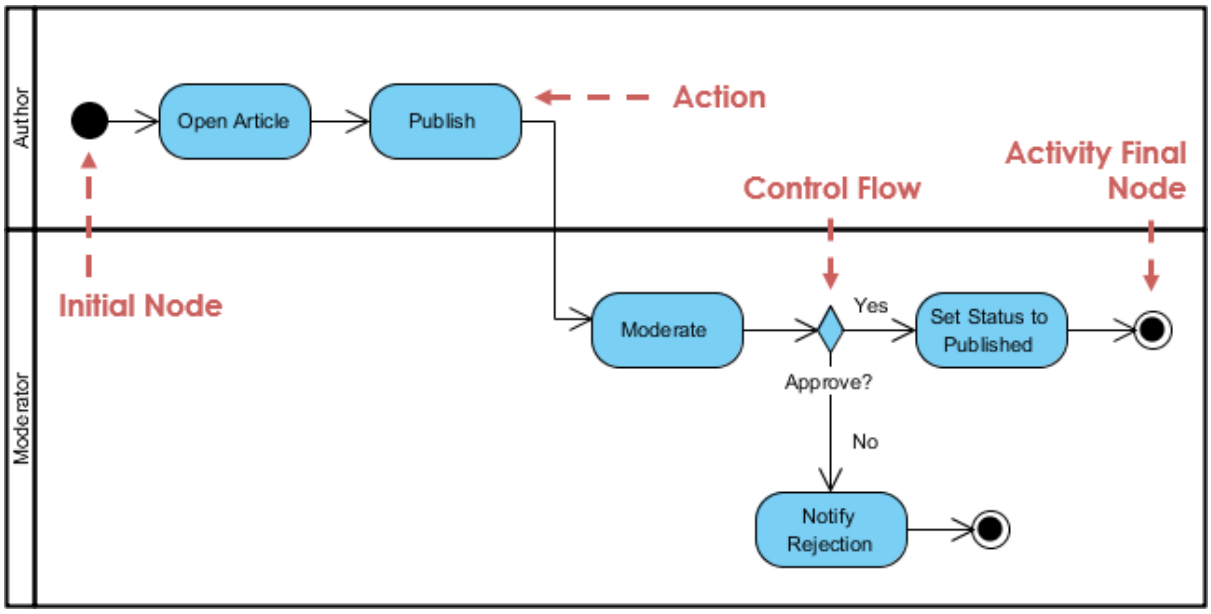
Use Case Diagram Example



❖ Activity Diagram

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. It describes the flow of control of the target system, such as the exploring complex business rules and operations, describing the use case also the business process. In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e. workflows).

Activity Diagram Example



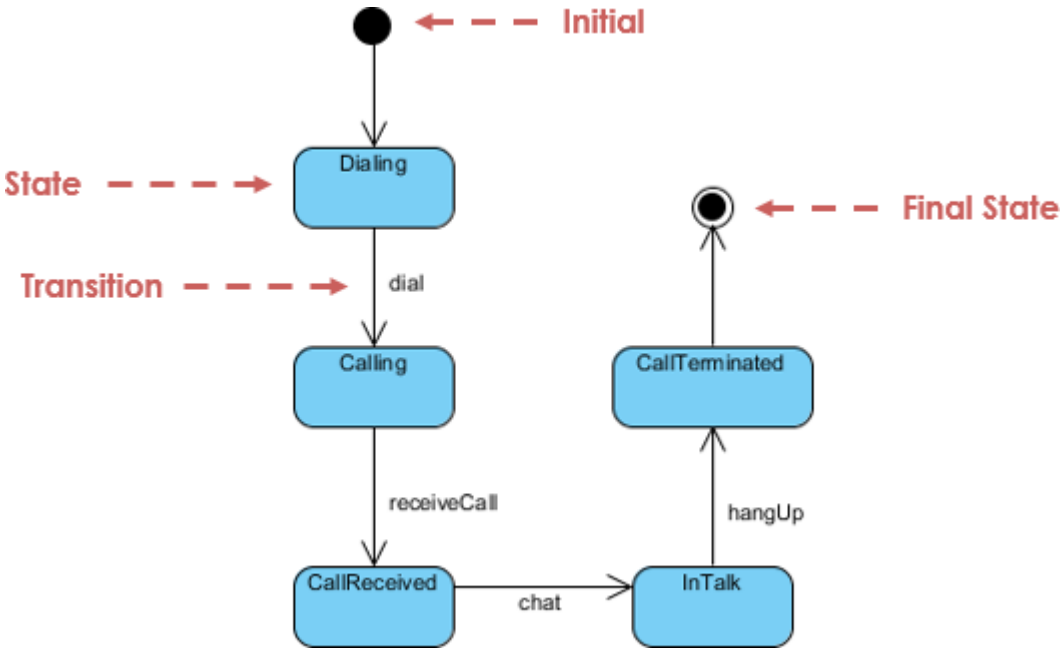
❖ State Machine Diagram

A state machine is any device that stores the status of an object at a given time and can change status or cause other actions based on the input it receives. States refer to the different combinations of information that an object can hold, not how the object behaves. In order to understand the different states of an object, you might want to visualize all of the possible states and show how an object gets to each state, and you can do so with a UML state diagram.

Each state diagram typically begins with a dark circle that indicates the initial state and ends with a bordered circle that denotes the final state. However, despite having clear start and end points, state diagrams are not necessarily the best tool for capturing an overall progression of events. Rather, they illustrate specific kinds of behavior—in particular, shifts from one state to another.

A state diagram is a type of diagram used in UML to describe the behavior of systems which is based on the concept of state diagrams by David Harel. State diagrams depict the permitted states and transitions as well as the events that effect these transitions. It helps to visualize the entire lifecycle of objects and thus help to provide a better understanding of state-based systems.

State Machine Diagram Example



❖ Sequence Diagram

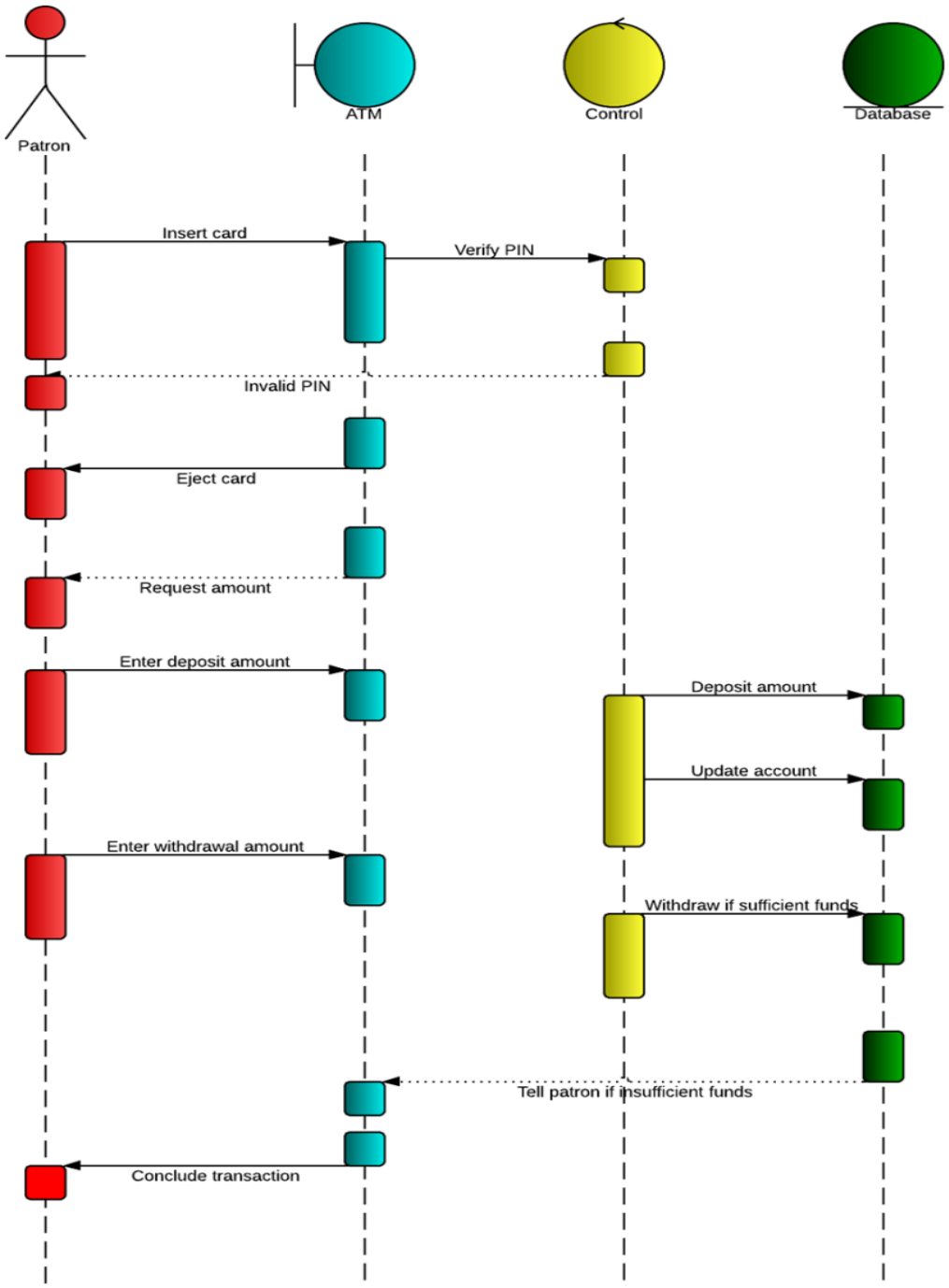
A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Sequence diagrams are sometimes known as event diagrams or event scenarios.

The Sequence Diagram models the collaboration of objects based on a time sequence. It shows how the objects interact with others in a particular scenario of a use case. With the advanced visual modeling capability, you can create complex sequence diagram in few clicks. Besides, some modeling tool such as Visual Paradigm can generate sequence diagram from the flow of events which you have defined in the use case description.

Sequence Diagram Example

Sequence diagram for ATM systems

An ATM allows patrons to access their bank accounts through a completely automated process. You can examine the steps of this process in a manageable way by drawing or viewing a sequence diagram. The example below outlines the sequential order of the interactions in the ATM system.


Patron
ATM
Control
Database
Insert card
Verify PIN
Invalid PIN
Eject card
Request amount
Enter deposit amount
Deposit amount
Update account
Enter withdrawal amount
Withdraw if sufficient funds
Tell patron if insufficient funds
Conclude transaction