ASSIGNMENT - 1

- What is SDLC ?

--> SDLC, or Software Development Life Cycle, is a process used to design, develop, test, and deploy software. It's like a roadmap that guides teams through the stages of creating software. Here are the main stages in simple terms:

1 Planning: Define the project goals, requirements, and scope. Decide what the software needs to do.

2 Analysis: Gather detailed requirements from stakeholders and analyze them to ensure they are feasible and clear.

3 Design: Create a blueprint for the software, including architecture, user interface, and data models.

4 Development: Write the actual code based on the design.

5 Testing: Check the software for bugs and ensure it works as intended.

6 Deployment: Release the software to users.

7 Maintenance: Update and fix the software as needed after it's been deployed.

Each stage helps ensure the software is well-built and meets user needs.

- What is software testing ?

--> Software testing is the process of checking a software program to make sure it works correctly. This involves running the software to find and fix any problems or bugs. The main goal is to ensure the software is reliable, meets user requirements, and is free of errors. In simple terms, it's like proofreading a document to catch and correct mistakes before it's shared with others.

- What is agile methodology ?

--> Agile methodology is a way of managing and completing projects, especially in software development, that focuses on flexibility, collaboration, and customer satisfaction. Here's a simple breakdown:

Iterative Process: Work is divided into small, manageable pieces called iterations or sprints, usually lasting 1-4 weeks. Each iteration involves planning, designing, coding, and testing a part of the project.

Frequent Feedback: At the end of each iteration, the team reviews what they've built and gets feedback from customers or stakeholders. This feedback helps guide the next steps.

Collaboration: Agile emphasizes teamwork and regular communication among all team members, including developers, designers, testers, and customers.

Flexibility: Plans and requirements can change based on feedback and new insights. Agile allows teams to adapt quickly rather than sticking to a fixed plan.

Continuous Improvement: After each iteration, the team reflects on what went well and what can be improved, making adjustments to improve future iterations.

In simple terms, Agile is like building something piece by piece, checking each piece with others to make sure it's right, and being ready to change plans as needed to make the final product better.

- What is SRS ?

--> SRS stands for Software Requirements Specification. It's a detailed document that describes what a software application should do and how it should perform. In the context of software testing, the SRS is very important because it serves as a guide for what needs to be tested.

--> The SRS outlines all the requirements for the software, including functional requirements (what the software should do) and non-functional requirements (how the software should perform, such as speed and security).

--> It provides a clear and detailed description of the software, so everyone involved in the project understands what needs to be built and tested.

--> Testers use the SRS to create test cases, which are specific scenarios to check if the software meets its requirements. The SRS helps ensure that the software is tested thoroughly and meets the needs of the users.

--> In simple terms, the SRS is like a blueprint for the software, and testers use it to make sure the final product matches what was planned.

- What is oops ?

--> OOPS stands for Object-Oriented Programming System. It's a way of designing and writing software where everything is organized around "objects."

- **Objects**: Think of objects as self-contained units that represent things in the real world, like a car or a person. Each object has data (attributes) and actions (methods) it can perform.
- **Classes**: A class is like a blueprint for creating objects. For example, you might have a class called "Car" that defines what a car object is: it has attributes like color and model, and methods like drive and stop.
- **Encapsulation**: This means keeping the details (data and methods) inside an object hidden from the outside world. It helps to protect the object's data and makes the code easier to manage.
- **Inheritance**: This allows one class to inherit the properties and methods of another class. For example, if you have a class called "Vehicle," a "Car" class can inherit from it and have all its properties plus some additional ones.
- **Polymorphism**: This allows objects to be treated as instances of their parent class. For example, if both "Car" and "Bike" are subclasses of "Vehicle," you can write code that works with a "Vehicle" object and it will work with both cars and bikes.

In simple terms, OOPS helps programmers create software that is more organized, modular, and easier to manage by thinking of the code as a collection of interacting objects, similar to real-world items.

- Write Basic Concepts of oops.

--> The basic concepts of Object-Oriented Programming (OOP) are:

1 Classes and Objects:

- **Class**: A blueprint for creating objects. It defines a datatype by bundling data and methods that work on the data into one single unit.
- **Object**: An instance of a class. It is created using the class blueprint and represents a specific example of a class.

2 Encapsulation:

- This concept refers to the bundling of data (attributes) and methods (functions) that operate on the data into a single unit or class.
- Encapsulation helps to protect the data from being accessed directly from outside the class and provides a controlled way of accessing and modifying it through methods.

3 Inheritance:

- Inheritance allows a new class (called a subclass or derived class) to inherit properties and methods from an existing class (called a superclass or base class).
- It promotes code reuse and establishes a natural hierarchy between classes.

4 Polymorphism: Polymorphism allows objects of different classes to be treated as objects of a common superclass. It can take two forms:

- **Method Overloading**: Different methods have the same name but different parameters within the same class.
- **Method Overriding**: A subclass provides a specific implementation of a method that is already defined in its superclass.

5 Abstraction:

- Abstraction means hiding the complex implementation details and showing only the essential features of the object.
- It allows focusing on what an object does instead of how it does it, making it easier to manage complexity.

--> These concepts work together to create a system where software is more modular, reusable, and easier to maintain.

- What is object ?

--> An object in programming is a specific instance of a class, like a real-world item. It has two main characteristics:

- **Attributes (Properties)**: These are the data or characteristics of the object. For example, if you have a `Car` object, attributes might include `color`, `model`, and `year`.
- **Methods (Functions)**: These are the actions or behaviors the object can perform. For the `Car` object, methods might include `drive`, `stop`, and `honk`.

In simple terms, an object is like a specific item you can see and interact with, created using a class blueprint.

- What is class ?

--> A class in programming is like a blueprint or template used to create objects. It defines the structure and behavior that the objects created from it will have. Here's a simple explanation:

- **Attributes (Properties)**: These are the characteristics or data fields defined in the class. For example, in a `Car` class, attributes might include `color`, `model`, and `year`.
- **Methods (Functions)**: These are the actions or behaviors that the objects created from the class can perform. For instance, in the `Car` class, methods might include `drive`, `stop`, and `honk`.

--> In simple terms, a class is like a blueprint for building specific items (objects). Just as a blueprint for a house defines how the house will look and function, a class defines the attributes and methods that its objects will have.

- What is encapsulation ?

--> Encapsulation is a concept in programming where you bundle the data (attributes) and the methods (functions) that operate on the data into a single unit, called a class. It helps to protect the data from being accessed directly from outside the class and provides a controlled way of accessing and modifying it.

Here's a simple explanation:

- **Data Protection**: Encapsulation hides the internal state of the object and only exposes certain parts of it through methods. This is like having private details inside a box and only allowing access through specific openings.
- **Controlled Access**: You can control how the data is accessed and changed by using methods, also known as getters and setters. This ensures that the data remains valid and consistent.
- **Modularity**: Encapsulation helps in keeping the code modular, meaning each object manages its own state and behavior. This makes the code easier to manage and understand.

--> In simple terms, encapsulation is like putting your valuables in a safe and only giving the key to trusted people. It keeps your data secure and ensures that only authorized methods can change it.

- What is inheritance ?

--> Inheritance is a concept in programming where a new class (called a subclass or derived class) is created based on an existing class (called a superclass or base class). The new class inherits attributes and methods from the existing class, allowing for code reuse and the creation of a natural hierarchy.

Here's a simple explanation:

- **Reusing Code**: Instead of writing the same code again, you can create a new class that inherits the properties and behaviors of an existing class. This saves time and effort.
- **Adding New Features**: The new class can have additional attributes and methods, or it can modify the inherited ones. This allows for extending or customizing the functionality.
- **Hierarchy**: Inheritance helps in creating a relationship between classes. For example, you can have a general class called `Vehicle`, and more specific classes like `Car` and `Bike` that inherit from `Vehicle`.

--> In simple terms, inheritance is like a child inheriting traits from their parents. Just as a child gets some characteristics from their parents but can also have their own unique traits, a subclass inherits features from its superclass but can also have its own unique features.

- What is polymorphism ?

--> Polymorphism is a concept in programming that allows objects of different classes to be treated as objects of a common superclass. It enables a single function, method, or operator to work in different ways depending on the context.

Here's a simple explanation:

- **Method Overloading**: This is when multiple methods have the same name but different parameters within the same class. For example, you might have a `print` method that can print both text and numbers, depending on what you pass to it.

- **Method Overriding**: This occurs when a subclass provides a specific implementation of a method that is already defined in its superclass. For instance, if you have a `Vehicle` class with a `move` method, a `Car` class that inherits from `Vehicle` can override the `move` method to provide a specific way of moving.

- **Flexibility**: Polymorphism allows for code that is more flexible and easier to maintain. You can write code that works on the superclass level, and it will automatically work with any subclass as well.

--> In simple terms, polymorphism is like a Swiss Army knife: a single tool that can do different things depending on how you use it. It allows the same operation to work on different kinds of objects.

- **Write SDLC phases with basic introduction**

--> The Software Development Life Cycle (SDLC) is a structured process used for developing software. It consists of several phases, each with its own purpose and activities. Here's a basic introduction to each phase:

1 Planning :

- **Purpose**: Define the project's goals, scope, and feasibility.
- **Activities**: Gather initial requirements, estimate costs, and resources, create a project plan, and assess risks.

2 Requirement Analysis :

- **Purpose**: Gather detailed information on what the software should do.
- **Activities**: Conduct interviews, surveys, and workshops with stakeholders to understand their needs. Document requirements in a clear and detailed manner.

3 Design :

- **Purpose**: Create a blueprint for how the software will be built.
- **Activities**: Design the system architecture, user interfaces, databases, and data flow diagrams. Develop detailed specifications for each component.

4 Development :

- **Purpose**: Write the actual code based on the design specifications.

- **Activities**: Implement the software using the chosen programming languages and tools. Ensure that the code follows the design and meets the requirements.

5 Testing :

- **Purpose**: Ensure the software works correctly and is free of bugs.
- **Activities**: Perform various tests (unit tests, integration tests, system tests, and user acceptance tests) to check for errors and verify that the software meets the requirements.

6 Deployment :

- **Purpose**: Release the software to users.
- **Activities**: Install the software on user systems, provide training, and create user documentation. Deploy the software in a production environment.

7 Maintenance :

- **Purpose**: Keep the software running smoothly after deployment.
- **Activities**: Fix bugs, update the software to adapt to changes, and improve performance. Provide ongoing support to users.

--> Each phase plays a crucial role in ensuring the software is developed systematically, meets user needs, and maintains quality throughout its lifecycle.

- **Explain Phases of the waterfall model ?**

--> The Waterfall model is a straightforward, linear approach to software development where each phase must be completed before the next one begins. Here are the phases explained in simple words:

1 Requirement Gathering and Analysis :

- **Purpose**: Understand and document what the software needs to do.
- **Activities**: Talk to stakeholders to gather all the requirements, analyze them, and create a detailed requirement specification document.

2 System Design:

- **Purpose**: Plan how the software will be built.
- **Activities**: Create design documents that outline the software architecture, components, interfaces, and data flow. This serves as a blueprint for the developers.

3 Implementation (Coding):

- **Purpose**: Write the actual code for the software.
- **Activities**: Developers use the design documents to write code in the chosen programming languages. Each component is developed separately.

4 Integration and Testing:

- **Purpose**: Ensure that the software works correctly as a whole.
- **Activities**: Integrate all the components and test the entire system for bugs, errors, and any issues. Perform various tests to verify that the software meets the requirements.

5 Deployment:

- **Purpose**: Release the software to the users.

- **Activities**: Install the software on the users' systems, ensure it works in the real environment, and make it available for use. Provide necessary user training and documentation.

6 Maintenance:

- **Purpose**: Keep the software running smoothly after it has been deployed.
- **Activities**: Fix any bugs or issues that arise, make updates and improvements, and provide ongoing support to users.

In the Waterfall model, each phase must be completed before moving on to the next. It is called "waterfall" because progress flows in one direction like a waterfall, from the top (requirements) to the bottom (maintenance).

- **Write phases of spiral model.**

--> The Spiral Model is a software development process that combines iterative development with systematic, risk-driven planning. It's designed to handle large, complex projects and reduce risks through repeated cycles, or spirals. Here are the phases :

1 Planning Phase:

- **Purpose**: Determine objectives, alternatives, and constraints for the project.
- **Activities**: Gather requirements, define goals, and consider potential solutions. Create an initial plan and set up the project scope.

2 Risk Analysis Phase:

- **Purpose**: Identify and evaluate risks associated with the project.
- **Activities**: Analyze potential risks, such as technical challenges, cost overruns, and schedule delays. Develop strategies to mitigate or manage these risks. Perform feasibility studies and create prototypes if necessary to better understand the risks.

3 Engineering Phase:

- **Purpose**: Develop and test the software incrementally.
- **Activities**: Design, code, and test a part of the software based on the current plan. This phase involves actual development and testing of the software modules or components. The focus is on building a product increment that can be evaluated.

4 Evaluation Phase:

- **Purpose**: Assess the current iteration and plan the next one.
- **Activities**: Review the developed product, gather feedback from stakeholders, and evaluate the progress. Based on this evaluation, refine requirements, and update the project plan for the next spiral.

--> These phases are repeated in cycles, with each cycle building on the previous one and refining the product further. The project progresses through multiple spirals, gradually developing the complete software while continuously managing risks and incorporating feedback.

--> In simple terms, the Spiral Model is like climbing a spiral staircase where you go around the steps repeatedly, each time getting closer to your goal while carefully planning and addressing risks along the way.

- **Write agile manifesto principles.**

--> The Agile Manifesto outlines four key values and twelve principles that guide Agile software development. Here are the twelve principles explained.

1 Customer Satisfaction: Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

--> Make customers happy by frequently delivering working software that meets their needs.

2 Welcome Change: Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.

--> Be flexible and ready to make changes, even if they come late in the project.

3 Frequent Delivery: Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

--> Deliver small, functional pieces of the software regularly, ideally every few weeks.

4 Collaboration: Business people and developers must work together daily throughout the project.

--> Ensure continuous and close communication between developers and business stakeholders.

5 Motivated Individuals: Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

--> Empower and support your team members, and trust them to do their best work.

6 Face-to-Face Conversation: The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

--> Communicate directly and in person whenever possible for better understanding.

7 Working Software: Working software is the primary measure of progress.

--> Judge progress by the amount of working software produced, not by documentation or other metrics.

8 Sustainable Development: Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

--> Work at a pace that can be sustained over the long term without burnout.

9 Technical Excellence: Continuous attention to technical excellence and good design enhances agility.

--> Focus on high-quality code and design to make future changes easier.

10 Simplicity: Simplicity—the art of maximizing the amount of work not done—is essential.

--> Keep things simple by doing only what is necessary and avoiding unnecessary work.

11 Self-Organizing Teams: The best architectures, requirements, and designs emerge from self-organizing teams.

--> Let teams organize themselves and make decisions on how to accomplish their tasks.

12 Reflection and Adjustment: At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

--> Regularly review and improve your team's processes and practices to become more efficient.

These principles help Agile teams focus on delivering value, being flexible, and continuously improving their processes and products.

- **Explain working methodology of agile model and also write pros and cons.**

--> **Working Methodology of the Agile Model :**

--> The Agile model is a flexible and iterative approach to software development. Here's how it typically works:

1 Iterations (Sprints):

- The project is divided into small chunks called iterations or sprints, usually lasting 1-4 weeks.
- Each iteration is like a mini-project with its own planning, development, testing, and review phases.

2 Planning:

- At the start of each iteration, the team meets to plan what they will accomplish. They select the most important features or tasks from the project backlog.

3 Development:

- During the iteration, developers work on coding the selected features.
- They collaborate closely with other team members and stakeholders to ensure they're building what's needed.

4 Testing:

- As features are developed, they are continuously tested.
- This helps catch and fix bugs early, ensuring high-quality software.

5 Review:

- At the end of the iteration, the team reviews what they have completed.
- They demonstrate the working software to stakeholders to get feedback.

6 Retrospective:

- The team reflects on what went well and what can be improved.
- They discuss how to make the next iteration more efficient and effective.

7 Repeat:

- The cycle repeats with the next iteration, incorporating feedback and improvements from the previous one.

Pros and Cons of the Agile Model:

Pros:

1 Flexibility and Adaptability:

- Agile allows for changes even late in the project, making it easier to adapt to new requirements or market conditions.

2 Customer Collaboration:

- Continuous feedback from customers ensures the final product meets their needs and expectations.

3 Frequent Delivery:

- Regular releases of working software provide early and ongoing value to customers.

4 Improved Quality:

- Continuous testing and review help catch and fix issues early, leading to a higher-quality product.

5 Motivated Teams:

- Empowering teams and promoting collaboration often lead to higher morale and better performance.

6 Reduced Risk:

- Short iterations and frequent feedback help identify and mitigate risks early.

Cons :

1 Less Predictable:

- Due to its flexible nature, it can be hard to predict timelines and costs accurately.

2 Requires Experienced Teams:

- Agile relies on well-coordinated, self-organizing teams, which can be challenging for less experienced groups.

3 Scope Creep:

- The flexibility to change requirements can sometimes lead to scope creep, where the project grows beyond its original intent.

4 Documentation:

- Agile focuses more on working software than on comprehensive documentation, which might be an issue for some stakeholders or for future maintenance.

5 Time-Consuming Meetings:

- Regular meetings (planning, reviews, retrospectives) can be time-consuming and may slow down the actual development work if not managed properly.

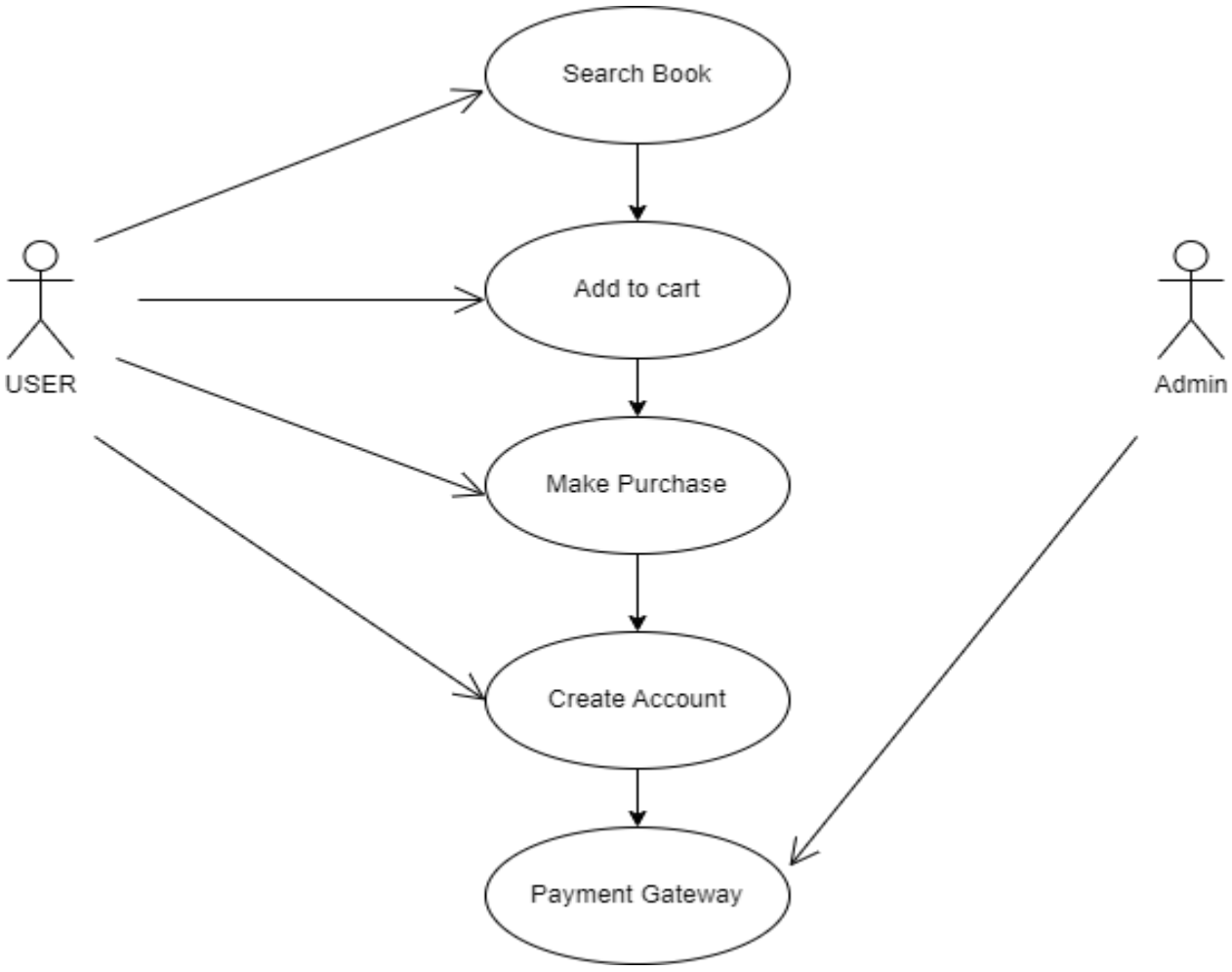
6 Requires Close Collaboration:

Agile works best when there is close collaboration between all team members and stakeholders, which can be challenging in distributed or remote teams.

--> In simple terms, the Agile model emphasizes flexibility, customer collaboration, and continuous improvement, which can lead to higher-quality software and happier customers. However, it requires experienced teams, close collaboration, and careful management to avoid potential pitfalls like scope creep and time-consuming processes.

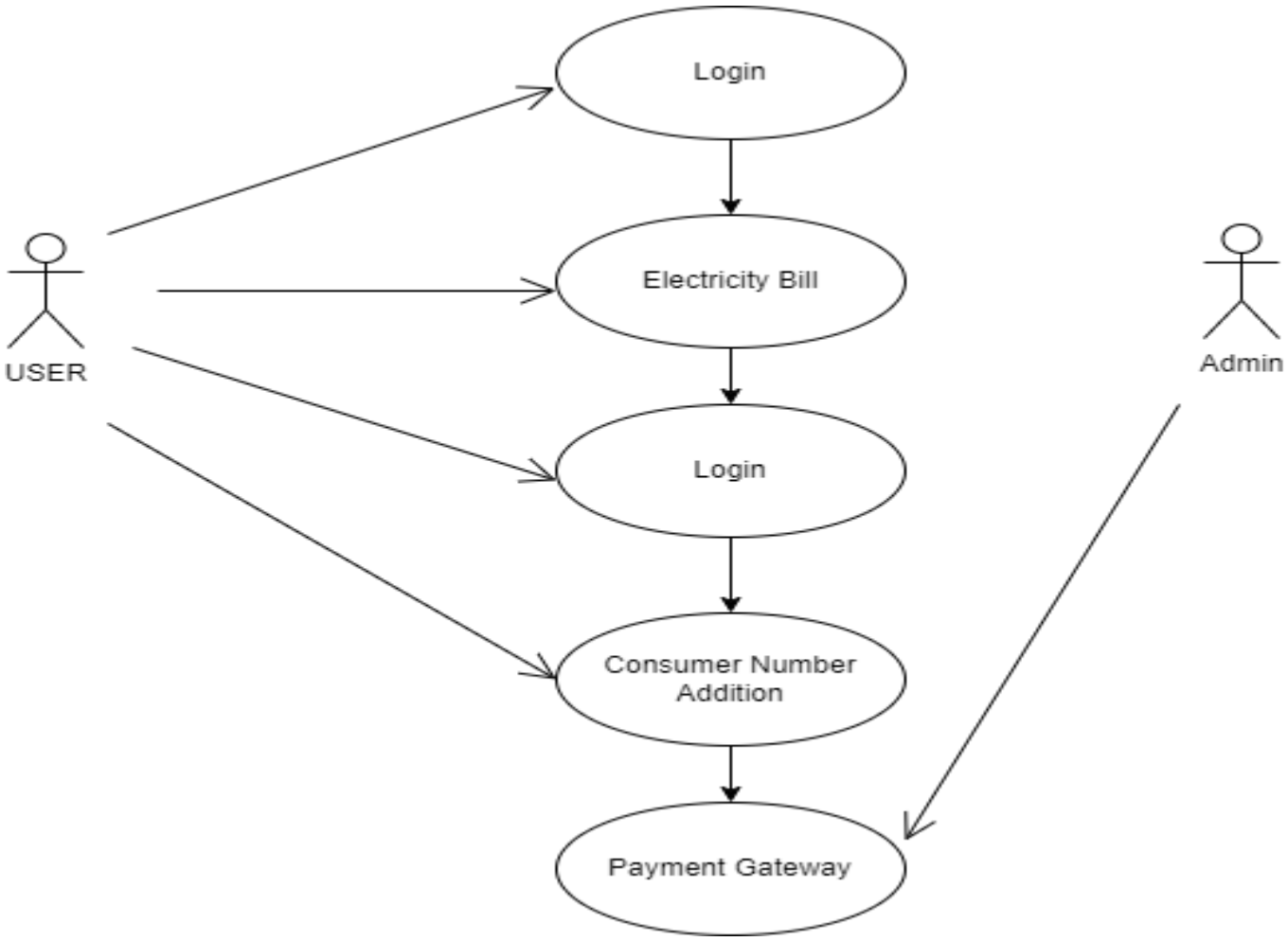
- **Draw Usecase on Online book shopping .**

-->



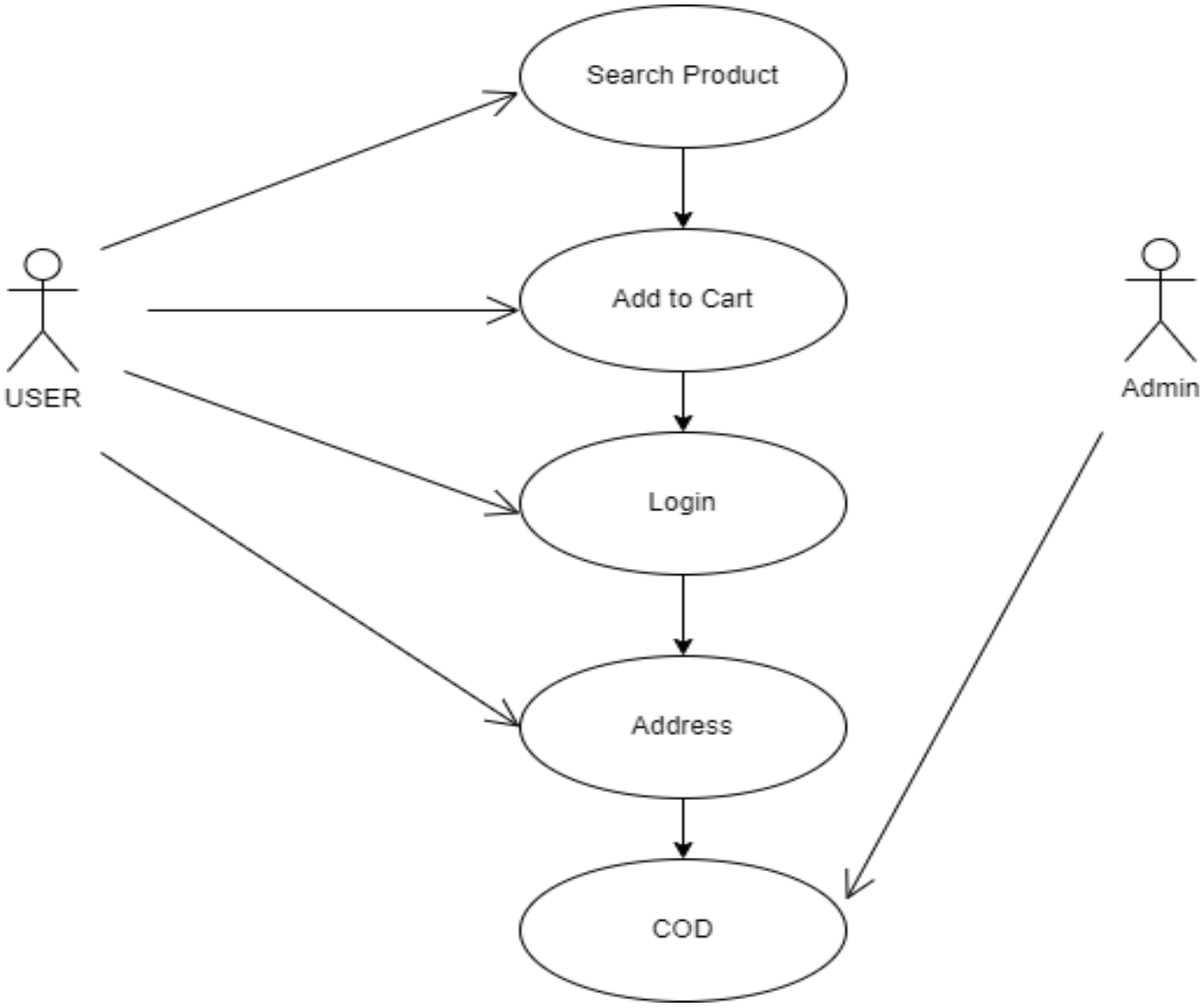
- **Draw Usecase on online bill payment system (paytm).**

-->



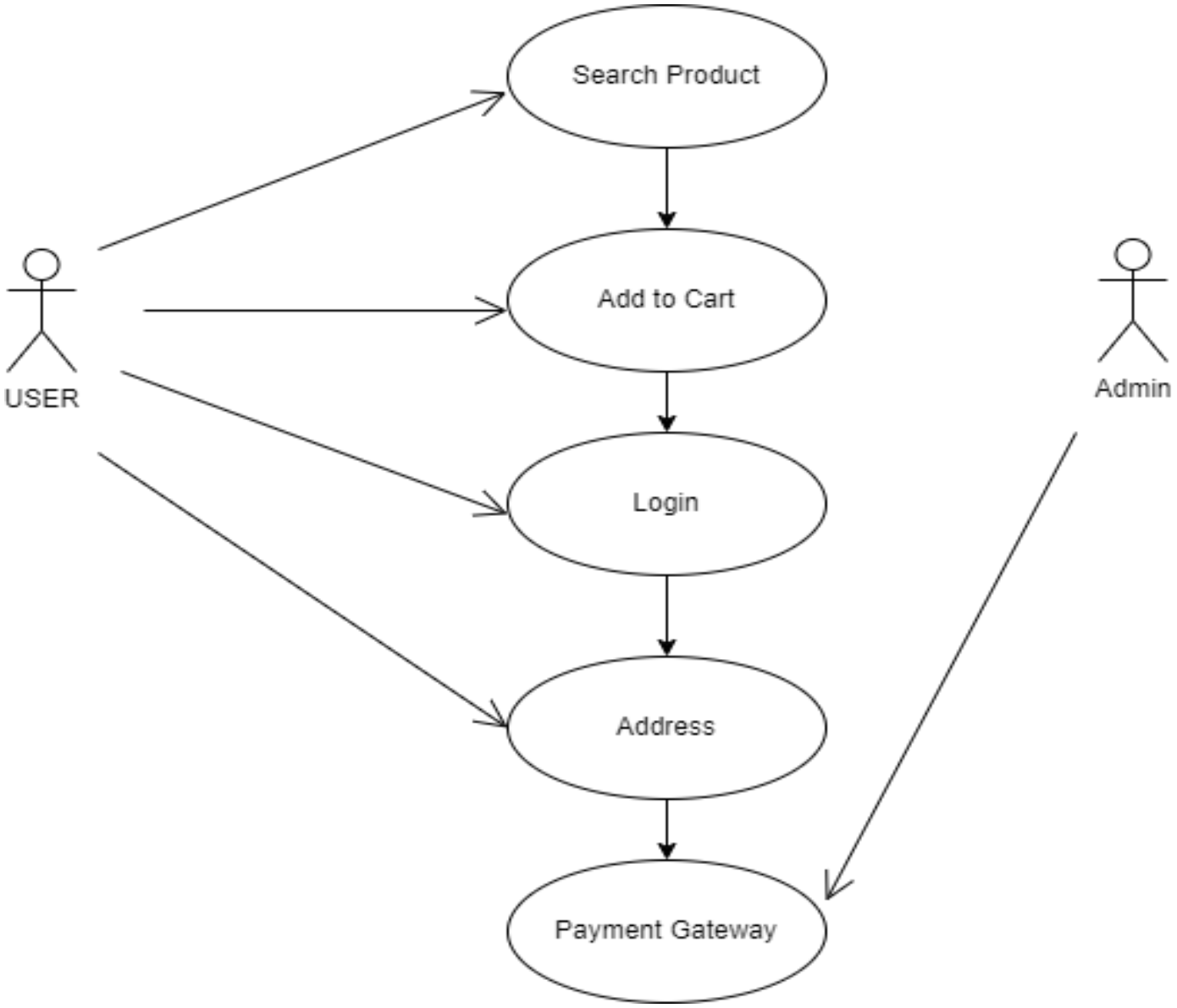
- **Draw usecase on Online shopping product using COD.**

-->



- **Draw usecase on Online shopping product using payment gateway.**

-->


Search Product
Add to Cart
Login
Address
Payment Gateway
USER
Admin