(1) What is SDLC.

→The software development lifecycle (SDLC) is the cost-effective and time-efficient process that development teams use to design and build high-quality software. The goal of SDLC is to minimize project risks through forward planning so that software meets customer expectations during production and beyond. This methodology outlines a series of steps that divide the software development process into tasks you can assign, complete, and measure.

(2) What is software testing?

→ Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

(3) What is agile methodology?

→ An agile methodology is an iterative approach to software development. Each iteration of agile methodology takes a short time interval of 1 to 4 weeks. The agile development process is aligned to deliver the changing business requirement. It distributes the software with faster and fewer changes.

The single-phase software development takes 6 to 18 months. In single-phase development, all the requirement gathering and risks management factors are predicted initially.

The agile software development process frequently takes the feedback of workable product. The workable product is delivered within 1 to 4 weeks of iteration.

(4) What is SRS.

→A software requirements specification (SRS) details the specific requirements of the software that is to be developed.

(5) What is oops.

→OOPs stands for Object-Oriented Programming. It's a programming paradigm that uses objects and classes to structure and design code. In OOP, you create objects, which are instances of classes, and these objects can have attributes (data) and methods (functions) that operate on the data. OOP helps organize and manage complex software by modeling real-world entities as objects and allowing for encapsulation, inheritance, and polymorphism to create modular and reusable code.

(6) Write Basic Concepts of oops.

→Here are some basic concepts of Object-Oriented Programming (OOP):

1. Classes and Objects:

   - A class is a blueprint or template for creating objects

   - An object is an instance of a class, representing a real-world entity.

2. Encapsulation:

   - Encapsulation refers to the bundling of data (attributes) and methods (functions) that operate on that data into a single unit (an object).

   - It helps in data hiding and restricting access to certain parts of an object.

3. Inheritance:

   - Inheritance allows a new class (subclass or derived class) to inherit properties and behaviors (attributes and methods) from an existing class (superclass or base class).

   - It promotes code reuse and hierarchy in class relationships.

4. Polymorphism:

   - Polymorphism means the ability to take on multiple forms.

   - In OOP, it allows objects of different classes to be treated as objects of a common superclass.

   - It enables method overriding and dynamic method binding.

5. Abstraction:

   - Abstraction involves simplifying complex reality by modeling classes based on the essential properties and behaviors.

   - It hides the unnecessary details while exposing the necessary features.

6. Message Passing:

   - Objects communicate by sending messages to each other.

- Methods are invoked on objects to perform actions or retrieve information.

7. Constructor and Destructor:

   - Constructors initialize objects when they are created.

   - Destructors perform cleanup when objects are destroyed (not always present in all OOP languages).

These fundamental concepts help structure and organize code in an object-oriented manner, making it more modular, maintainable, and scalable.

(7) What is object

→An object, in the context of Object-Oriented Programming (OOP), is a fundamental concept representing a real-world entity or a software entity. It is an instance of a class, which serves as a blueprint or template for creating objects.

Here are some key points about objects:

1. Instance of a Class: An object is created based on a class. The class defines the structure and behavior of the object.

2. Attributes (Data): Objects have attributes, also known as fields or properties, which store data related to the object's state. These attributes are defined by the class.

3. Methods (Functions): Objects can have methods, which are functions associated with the object. These methods allow objects to perform actions and interact with their data.

4. Encapsulation: Objects encapsulate their data and methods, meaning they bundle data and the operations that can be performed on that data into a single unit. This helps in data hiding and abstraction.

5. Identity: Each object has a unique identity, allowing it to be distinguished from other objects, even if they have the same attributes and methods.

6. Instantiation: Creating an object from a class is called instantiation. It involves allocating memory and initializing the object's attributes.

7. State: The state of an object refers to the values stored in its attributes at a given moment. Objects can change state by invoking methods that modify their data.

8. Behavior: Objects exhibit behavior through their methods. These methods define what actions the object can perform.

Objects are a central concept in OOP, and they are used to model and represent entities in software that mirror real-world objects, making it easier to design, organize, and manipulate complex systems.

(8) What is class.

→In object oriented programming, a class is a template definition of the methods and variables in a particular kind of object. Thus, an object is a specific instance of a class; it contains real values instead of variables.

(9) What is encapsulation.

→ Encapsulation is the practice of including in an object everything it needs hidden from other objects. The internal state is usually not accessible by other objects.

Encapsulation is placing the data and the functions that work on that data in the same place. While working with procedural languages, it is not always clear which functions work on which variables but objectoriented programming provides you framework to place the data and the relevant functions together in the same object.

(10) What is polymorphism.

→Polymorphism means “having many forms”. It allows different objects to respond to the same message in different ways, the response specific to the type of the object. The most important aspect of an object is its behaviour (the things it can do). A behaviour is initiated by sending a message to the object (usually by calling a method). The ability to use an operator or function in different ways in other words giving different meaning or functions to the operators or functions is called polymorphism.

(11) What is inheritance.

→Inheritance means that one class inherits the characteristics of another class. This is also called a “is a” relationship One of the most useful aspects of object-oriented programming is code reusability. As the name suggests Inheritance is the process of forming a new class from an existing class that is from the existing class called as base class, new class is formed called as derived class. This is a very important concept of object-oriented programming since this feature helps to reduce the code size. Inheritance describes the relationship between two classes. A class can get some of its characteristics from a parent class and then add unique features of its own.

(12) Write SDLC phases with basic introduction.

→The Software Development Life Cycle (SDLC) is a systematic process used in software engineering to design, develop, test, deploy, and maintain software applications. It's a series of phases and activities that guide the creation of high-quality software. Here's an overview of the basic phases in SDLC:

1. Requirements Gathering and Analysis:

- In this phase, project stakeholders, including clients and end-users, are consulted to gather and document their requirements.

- The goal is to understand what the software needs to accomplish, its functionality, and constraints.

2. System Design:

- During system design, the overall system architecture is defined. It includes high-level design decisions, data structures, and system components.

- Architects and designers create a blueprint for the software based on the gathered requirements.

3. Implementation (Coding):

- The implementation phase involves writing the actual code for the software based on the design specifications.

- Programmers and developers create the software by translating design documents into executable code.

4. Testing:

- Software testing is crucial to ensure that the software meets its requirements and functions correctly.

- Testers use various techniques to identify and fix defects or issues in the software, aiming for a high level of quality and reliability.

5. Maintenance or Support:

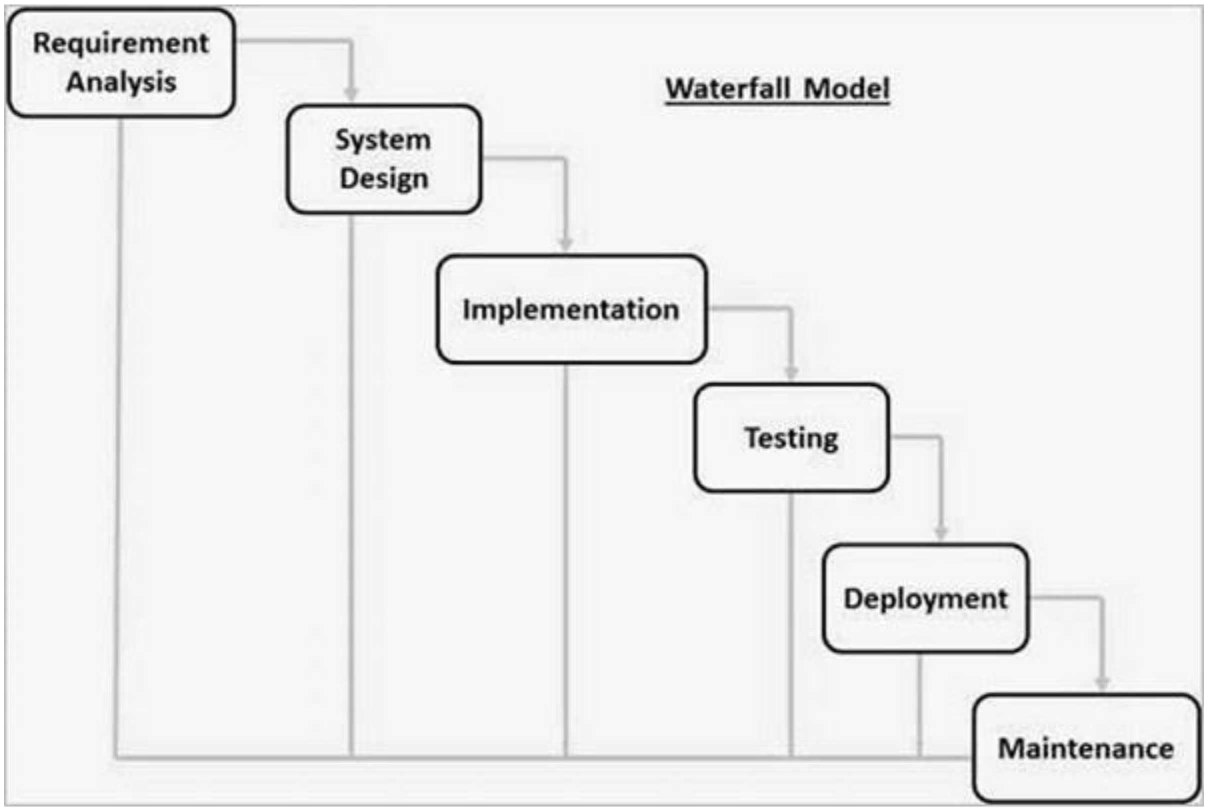
- After deployment, the software enters the maintenance phase, where it is continuously monitored and updated to address issues, add new features, and adapt to changing requirements.

- User support and troubleshooting are ongoing activities in this phase.

- Throughout the SDLC, documentation is created and updated. It includes user manuals, technical specifications, and

(13) Explain Phases of the waterfall model.

→ Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In



this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

The following illustration is a representation of the different phases of the Waterfall Model.

- **Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- **System Design** − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- **Implementation** − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next

phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.

- **Integration and Testing** − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment of system** − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
- **Maintenance** − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

(14) Write phases of spiral model

→The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.

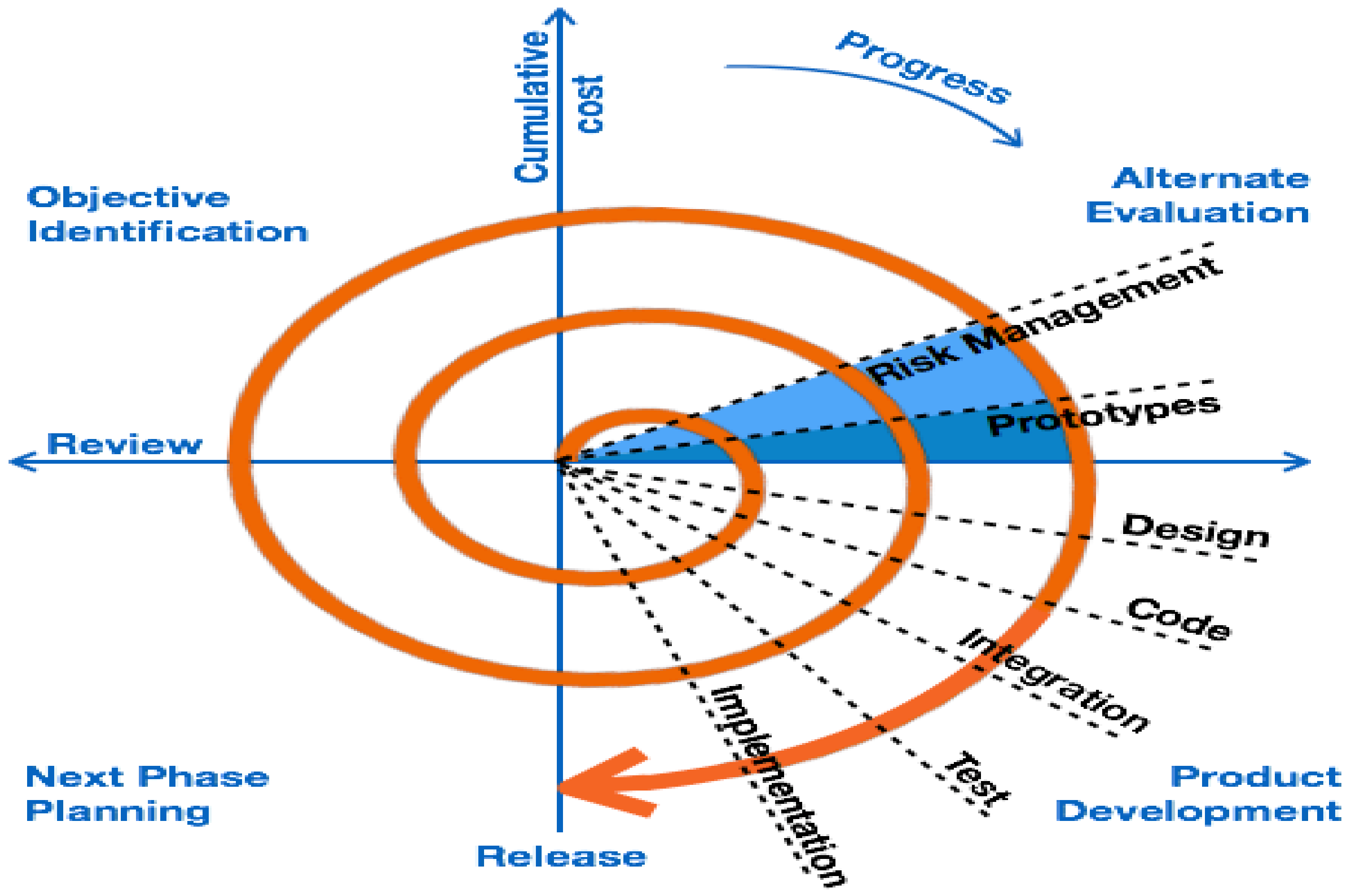
- Identification

This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

This phase also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral, the product is deployed in the identified market.

- Design

The Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and the final design in the subsequent spirals.



- Construct or Build

The Construct phase refers to production of the actual software product at every spiral. In the baseline spiral, when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to the customer for feedback.

- Evaluation and Risk Analysis

Risk Analysis includes identifying, estimating and monitoring the technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

The following illustration is a representation of the Spiral Model, listing the activities in each phase.

(15) Write agile manifesto principles.

→

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

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

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

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

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

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

7. Working software is the primary measure of progress.

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

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

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

11. The best architectures, requirements, and designs emerge from self-organizing teams.

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

(16) Explain working methodology of agile model and also write pros and cons.

→

Working Methodology of Agile:

User Stories and Backlog: Agile projects start with the creation of a product backlog, which is a prioritized list of features, enhancements, and user stories. User stories describe specific functionality from the end-user's perspective.

Sprints: Agile development is divided into fixed-length time periods called sprints, typically 2-4 weeks long. During each sprint, a subset of items from the product backlog is selected for implementation.

Planning: At the beginning of each sprint, the team holds a sprint planning meeting to select user stories to work on and create a sprint backlog. The team defines the tasks needed to complete these user stories.

Daily Standup Meetings: The team holds daily standup meetings to discuss progress, issues, and obstacles. This keeps everyone informed and ensures that any impediments are addressed promptly.

Continuous Development: Developers work on the selected user stories, making frequent small releases that can be tested and reviewed.

Testing and Quality Assurance: Testing and quality assurance are integrated into the development process. Testing is performed continuously, and issues are addressed as they arise.

Review and Retrospective: At the end of each sprint, the team conducts a sprint review to demonstrate the completed work to stakeholders. They also hold a sprint retrospective to discuss what went well and what could be improved in the next sprint.

Adaptation: Based on feedback from stakeholders and the team's experiences, adjustments are made to the product backlog and development process. This flexibility allows for changes in project direction and priorities.

❖ Pros of Agile:

Customer-Centric: Agile places a strong emphasis on customer feedback, ensuring that the product aligns with user needs and expectations.

Flexibility: Agile allows for changing requirements, making it suitable for projects in dynamic environments.

Early Delivery: Incremental development leads to early and frequent releases, allowing customers to start benefiting from the product sooner.

Improved Quality: Continuous testing and collaboration help identify and address issues early, leading to higher product quality.

Enhanced Collaboration: Cross-functional teams work closely together, promoting better communication and collaboration among team members.

❖ Cons of Agile:

Uncertainty: Agile projects may be less predictable due to changing requirements, which can be challenging for project planning and budgeting.

Resource Intensive: Agile requires active participation and dedication from team members, making it resource-intensive.

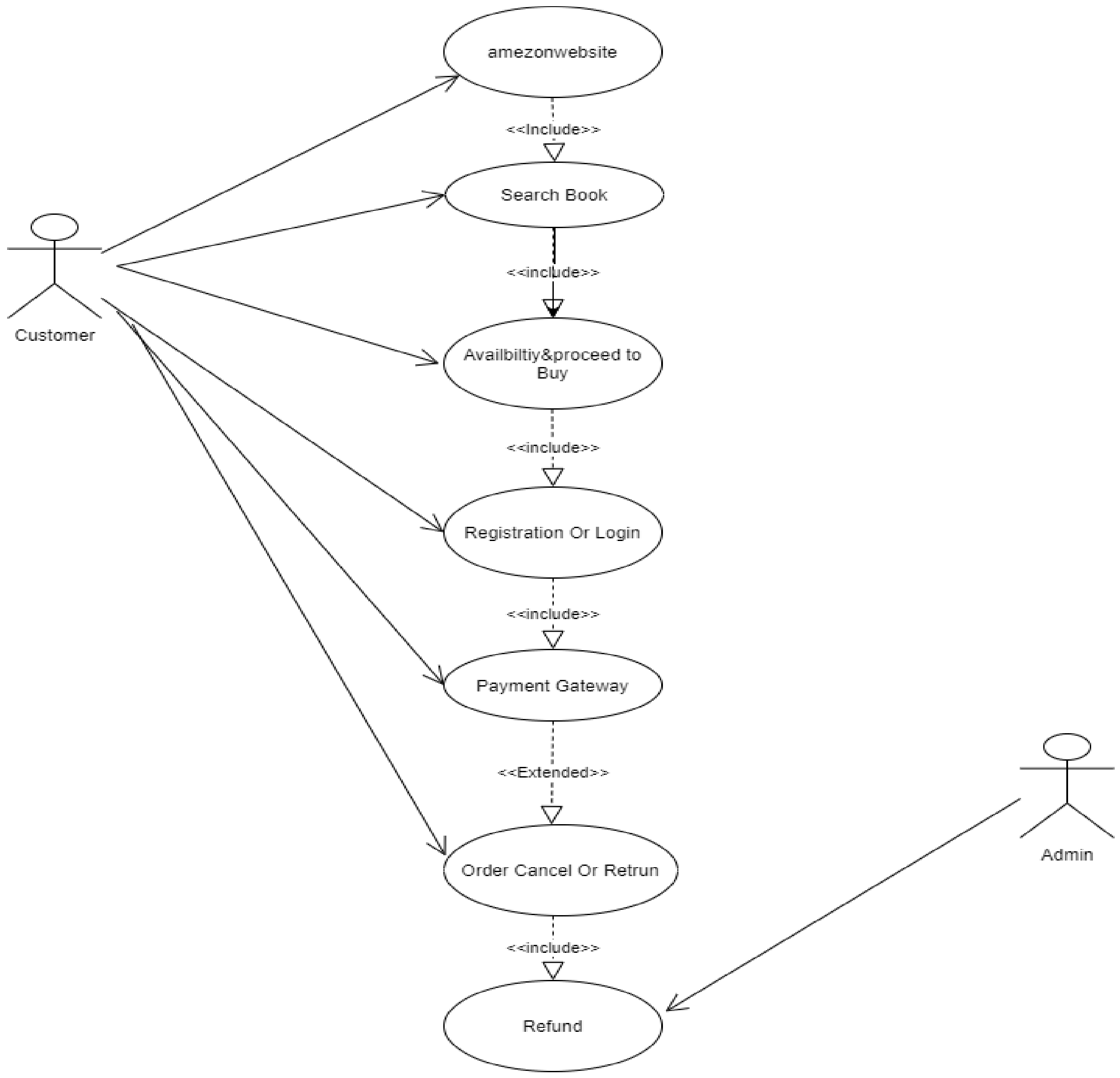
Complexity: Agile methodologies can be complex to implement and may require organizational changes.

Documentation: Some Agile approaches may have less emphasis on documentation, which can be a disadvantage in regulated industries.

Risk of Scope Creep: Frequent changes in requirements may lead to scope creep, which can affect project timelines and budgets if not managed properly.

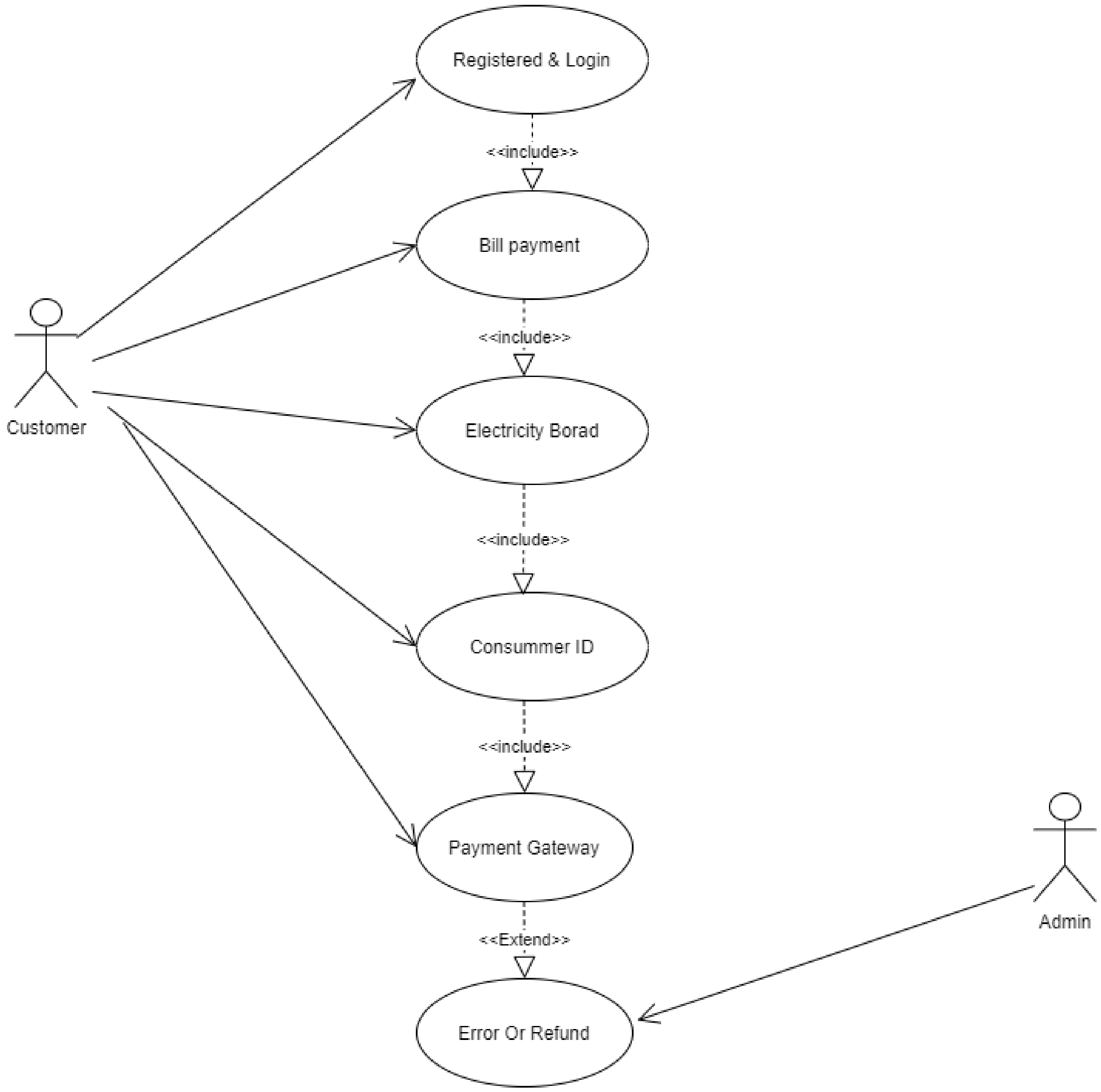
(17) Draw Usecase on Online book shopping.

→



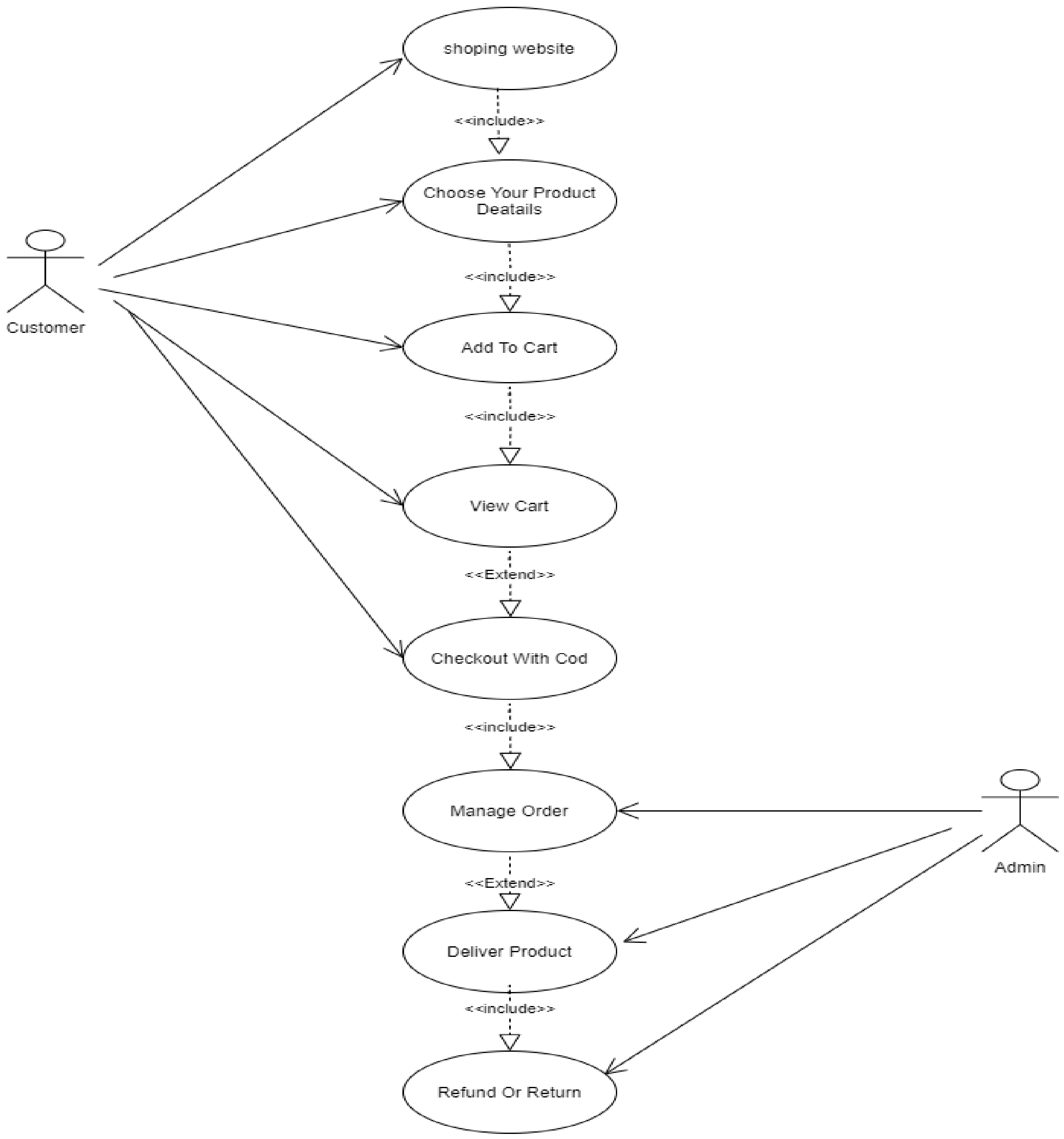
(18) Draw Usecase on online bill payment system (paytm)

→



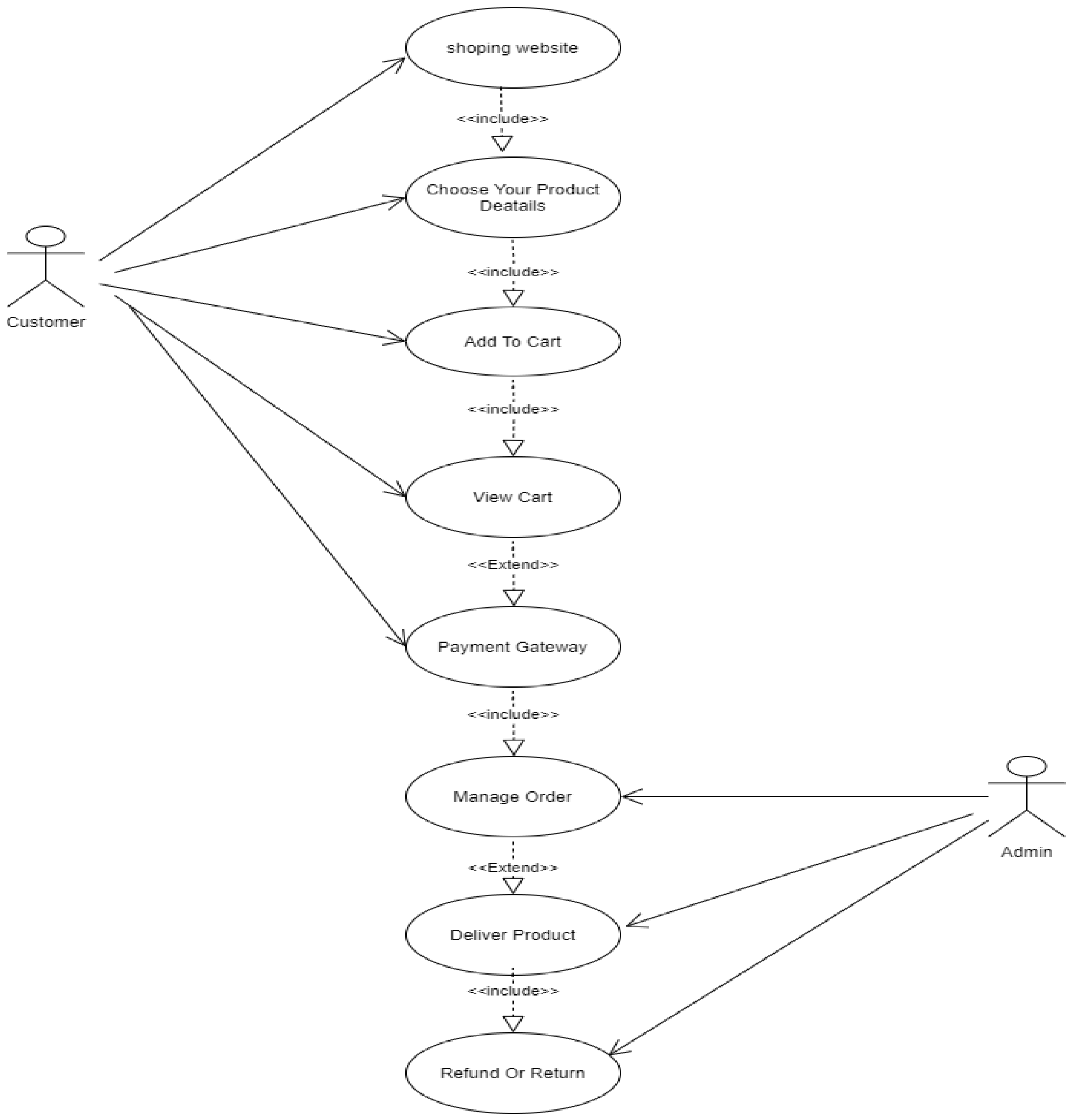
(19) Draw usecase on Online shopping product using COD.

→



(20) Draw usecase on Online shopping product using payment gateway.

→


shoping website
<<include>>
Choose Your Product Deatails
<<include>>
Add To Cart
<<include>>
View Cart
<<Extend>>
Payment Gateway
<<include>>
Manage Order
<<Extend>>
Deliver Product
<<include>>
Refund Or Return
Customer
Admin