Module 1(Fundamental)

1. **What is SDLC?**

SDLC stands for Software Development Life Cycle. It's a process used by software developers to design, create, test, and deploy software. You can think of it as a roadmap or a plan that helps ensure software is built correctly and efficiently. Here are the key steps in SDLC, explained in simple terms.

* 1. **Planning:** Deciding what the software needs to do and making a plan for how to build it.
  2. **Analysis:** Understanding the requirements and making sure they are clear and achievable.
  3. **Design:** Creating a blueprint for how the software will work, including its look and feel.
  4. **Development:** Writing the actual code to build the software.
  5. **Testing:** Checking the software to make sure it works correctly and fixing any problems.
  6. **Deployment:** Releasing the software for people to use.
  7. **Maintenance:** Making updates and improvements to the software over time.

1. **What is software testing?**

Software testing is the process of checking if a software program works as it should. Think of it like proofreading a book to find and fix any mistakes before publishing. Here's a simple breakdown.

* 1. **Finding Bugs:** Just like looking for typos in a book, testers find problems (bugs) in the software.
  2. **Verifying Functionality:** Ensuring that the software does what it's supposed to do, like making sure a calculator app correctly adds and subtracts numbers.
  3. **Ensuring Performance:** Checking if the software runs smoothly and quickly, like making sure a game doesn't lag or crash.
  4. **User Experience:** Making sure the software is easy to use and looks good, like making sure a website is easy to navigate.

In short, software testing helps ensure the software is reliable, works properly, and provides a good experience for users.

1. **What is agile methodology?**

Agile methodology is a way of managing projects, especially in software development, that focuses on flexibility, collaboration, and customer satisfaction. Imagine building a Lego model, but instead of completing it all at once, you build it in small sections, check if everything is correct, and make adjustments as needed.

Here's a simple breakdown of agile methodology,

* 1. **Small Steps (Iterations):** The project is broken down into small, manageable chunks called iterations or sprints, usually lasting a few weeks.
  2. **Continuous Feedback:** After each iteration, the team reviews the progress, gathers feedback from customers or stakeholders, and makes necessary changes.
  3. **Collaboration:** Team members work closely together and with customers to ensure everyone is on the same page and working towards the same goals.
  4. **Flexibility:** Agile allows for changes and adjustments throughout the project, making it easier to respond to new information or changing requirements.
  5. **Quick Delivery:** By delivering small, working parts of the project regularly, customers get to see progress and use parts of the product sooner.

1. **What is SRS ?**

SRS stands for Software Requirements Specification. It's a detailed document that describes what a software should do and how it should perform. Think of it as a blueprint or a guide for building the software.

Here's a simple breakdown:

* 1. **What the Software Should Do**: It explains all the functions and features the software needs to have. For example, if it's a

weather app, the SRS would list features like showing current weather, forecasting, and alerting users about severe weather.

* 1. **How the Software Should Work:** It details how the software should behave in different situations, including any specific requirements or constraints. For example, the weather app should update every hour and work on both Android and iOS devices.
  2. **User Interface:** It describes how the software should look and feel for users. This includes things like screen layouts, buttons, and navigation.
  3. **Performance and Security:** It outlines how fast the software should run, how much data it can handle, and any security measures it needs to protect user information.

**5.What is oops?**

OOPS stands for Object-Oriented Programming System. It's a way of writing software that organizes code into reusable blocks, called objects. Each object represents a real-world thing or concept and contains both data (attributes) and methods (functions) that work on the data. Here are the key concepts in simple terms:

* 1. **Class:** A blueprint for creating objects, like a recipe for making cakes.
  2. **Object:** An instance of a class, like an actual cake made from the recipe.
  3. **Inheritance:** A way for one class to use the properties and methods of another class, like a child inheriting traits from a parent.
  4. **Encapsulation:** Keeping the data inside an object safe from outside interference, like having a protective cover on a phone.
  5. **Polymorphism**: Allowing objects to be treated as instances of their parent class, even if they behave differently, like a smartphone with different apps that all work differently but use the same interface.
  6. **Abstraction**: Abstraction is the representation of the essential features of an object these are ‘encapsulated’ into an abstract data type.

**6.Write basic concepts of oops?**

Here are the basic concepts of Object-Oriented Programming (OOP) in simple and short terms:

1**.Class:** A blueprint for creating objects.

2**.Object:** An instance of a class.

3**.Inheritance:** One class inherits properties and methods from another.

4.**Encapsulation:** Hiding data within an object, accessible only through methods.

5.**Polymorphism:** Same method can behave differently in different classes.

6.**Abstraction:** Hiding complex details, showing only the essential features.

**7. What is object ?**

In software testing, an object refers to an element in the software being tested, such as a button, text box, or menu. It is something that users can interact with and testers need to check to ensure it works correctly.

**8.What is class ?**

In software testing, a class refers to a blueprint for creating objects. These objects can be elements in the software, like buttons, text boxes, or any user interface components. A class defines the properties and behaviors that these objects will have, helping testers to understand and verify how different parts of the software should function.

**9.What is encapsulation ?**

Encapsulation is a concept in programming where an object's data (attributes) is kept private and can only be accessed or modified through specific methods (functions). Think of it like a protective cover on a smartphone that keeps the internal components safe and only lets you interact with them through the screen and buttons.

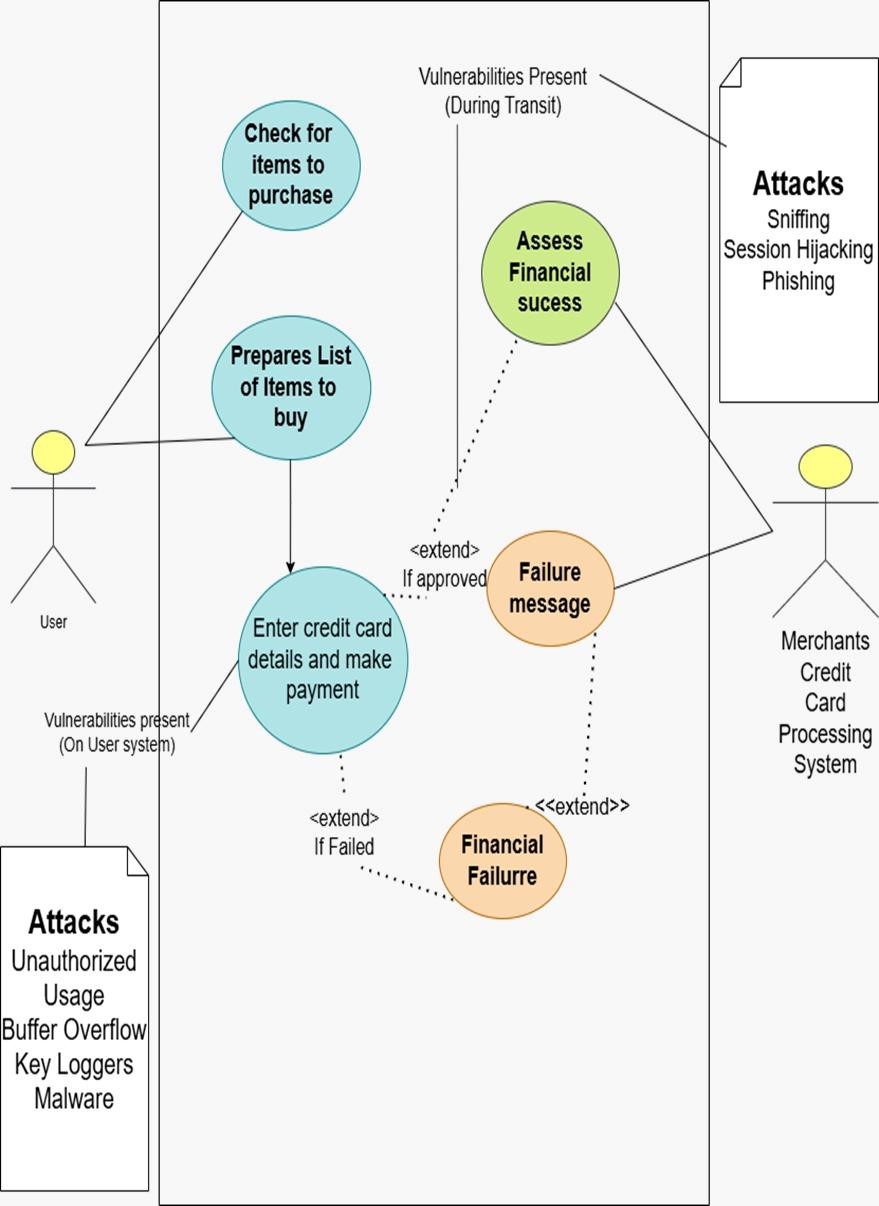
**10.What is inheritance ?**

In software testing, inheritance refers to the practice where a test case or test suite can inherit properties and methods from another test case or suite. This allows testers to reuse common test scenarios and functionalities without rewriting them

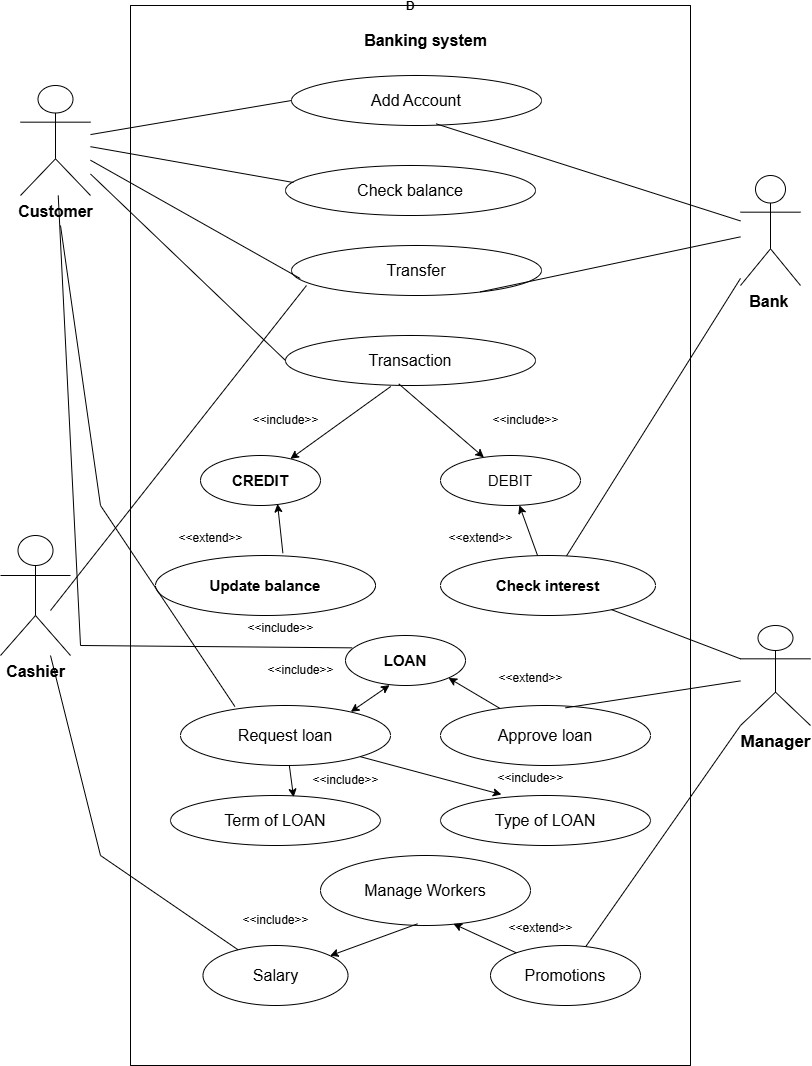
**11.What is polymorphism ?**

In software testing, polymorphism refers to the ability of different objects to be tested using the same interface or method. This is particularly useful when testing software that uses inheritance and different implementations of methods.

**12.Draw Use case on online bill payment system (Paytm)**



**13.Draw Use case on banking system for customers**



**14.Draw Use case on broadcasting system**



**15.Write SDLC phases with basic introduction**

The Software Development Life Cycle (SDLC) is a process used by software developers to design, develop, test, and deploy software applications. It helps ensure that software is created systematically and efficiently. There are typically six phases in the SDLC:

# 1. Planning

* What it is: This is the first phase where the project is discussed and planned.
* What happens: Developers and stakeholders meet to understand what the software needs to do. They decide on the goals, resources, time, and costs involved. It’s like making a blueprint for the software.

# 2. Feasibility Study

* What it is: This phase analyzes whether the project is possible and practical.
* What happens: The team checks if the project can be done within the available resources (time, budget, technology). They also assess the potential risks and challenges.

# 3. Design

* What it is: In this phase, the software’s structure and components are designed.

What happens: Developers create detailed plans for how the software will look and work. This includes the system architecture, user interfaces, and database design. It's like creating the detailed plan for building the software.

# 4. Development

* What it is: This is where the actual coding of the software happens.
* What happens: Developers write the code based on the design.

The software's features and functionalities are built in this phase.

# 5. Testing

* What it is: This phase ensures the software works as expected.
* What happens: The software is tested for bugs, errors, and issues. Testers check if everything works correctly and meets the original requirements. If problems are found, developers fix them.

# 6. Deployment

* What it is: This phase involves releasing the software for use.
* What happens: Once the software is fully tested and ready, it is deployed to the users. This could mean installing it on computers, servers, or making it available for download.

**16.Explain Phases of the waterfall model**

The Waterfall Model is a traditional software development process where each phase is completed before moving on to the next one. It’s called "Waterfall" because progress flows down in a linear way, like a waterfall. Here are the phases explained in simple terms:

# 1.Requirement Gathering and Analysis

* What it is: In this first phase, the team gathers all the information about what the software needs to do.
* What happens: Developers talk to the client or stakeholders to understand their needs and document the requirements. This is like writing a list of everything the software should be able to do.

# 2. System Design

* What it is: This phase focuses on planning how the software will work.
* What happens: Based on the requirements, developers and designers create detailed plans about how the software will be built. This includes creating blueprints for the structure, user interface, and system components.

# 3.Implementation (Coding)

* What it is: Now, the actual work of building the software begins.
* What happens: Developers write the code based on the designs. They turn the plans into a working program. This is where the software starts to take shape.

# 4.Testing

* What it is: After coding, the software is tested to make sure it works correctly.
* What happens: The software is checked for bugs, errors, or anything that doesn’t work right. Testers make sure the software meets all the requirements that were gathered earlier. If there are problems, developers fix them.

# 5. Deployment

* What it is: The software is ready to be used by the end users.
* What happens: Once everything is tested and fixed, the software is released for use. This could mean installing it on computers or making it available for download.

# 6.Maintenance

* What it is: After deployment, the software may need updates and fixes.
* What happens: The software is monitored to make sure it keeps working smoothly. If any issues come up or users need new features, they are addressed in this phase. Maintenance ensures the software stays useful and up-to-date

**17.Write phases of spiral model**

The Spiral Model combines elements of both design and prototyping. It focuses on repeated cycles of development. Here are the phases explained in simple terms:

# 1. Planning

* What it is: Identify goals, requirements, and risks for the project.
* What happens: The team figures out what needs to be done and plans how to do it.

# 2. Risk Analysis

* What it is: Analyze potential risks and find solutions.
* What happens: The team looks at possible problems and creates plans to avoid or manage them.

# 3. Engineering (Development & Design)

* What it is: Build and design the software.
* What happens: The team creates the software, designs it, and builds the prototype or working model.

# 4.Testing and Evaluation

* What it is: Test the software and get feedback.
* What happens: The software is tested, and feedback is gathered to see if it meets goals and requirements.

# 5.Planning for Next Iteration

* What it is: Plan the next cycle or update.
* What happens: Based on the feedback, the team plans for the next round of improvements or changes.

**18.Write agile manifesto principles**

* Customer first – Keep customers happy with fast and regular updates.
* Welcome change – Be ready to adjust plans anytime.
* Deliver often – Release small, working parts quickly.
* Work together – Business and tech teams must collaborate daily.
* Trust teams – Give them what they need and let them do their job.
* Face-to-face talk – Talking directly is the best way to share ideas.
* Working software – A working product matters more than paperwork.
* Keep steady – Move at a pace that your team can keep up with.
* Focus on quality – Keep improving and making things better.
* Keep it simple – Do just enough, avoid unnecessary work.
* Self-organizing teams – Let teams decide how to do their work.
* Reflect and improve – Regularly review and get better.

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

Agile Model – Working Methodology

* 1. **Plan** – Break big work into small tasks.
  2. **Design** – Think about how to build the product.
  3. **Develop** – Create small parts of the product.
  4. **Test** – Check if it works well.
  5. **Release** – Give it to users and get feedback.
  6. **Repeat** – Improve and build more in short cycles.

**Pros of Agile**

Fast delivery – Small updates come quickly.

Flexible – Easy to change plans anytime.

Better teamwork – Teams and customers work closely.

Higher quality – Continuous testing and improvements.

Happy customers – Regular feedback makes them satisfied.

**Cons of Agile**

Needs strong teamwork – Works best when teams communicate well.

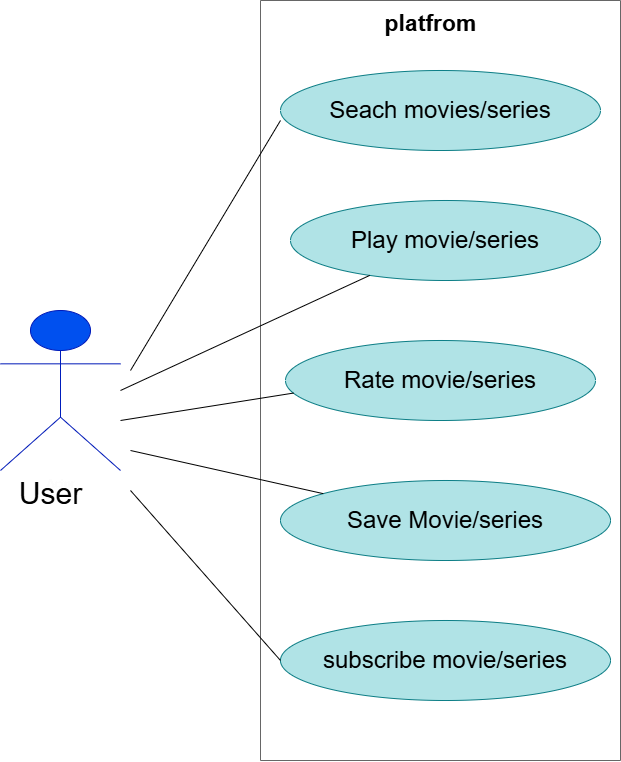
Less predictability – Hard to plan everything in detail.

Too many meetings – Frequent discussions can take time.

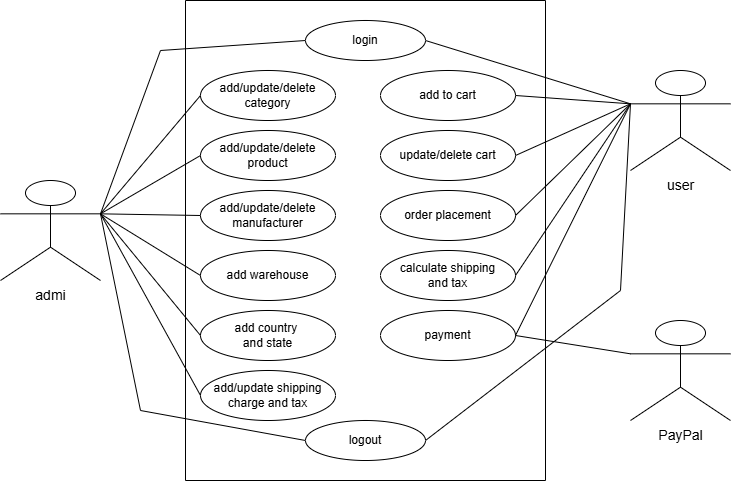
Not for all projects – Big, fixed projects may not fit Agile.

Requires experienced team – Needs skilled people to manage change.

**20.Draw Use case on OTT platform**



**21.Draw Use case on E-commerce application**



**22.Draw Use case on online shopping product using payment gateway**

