MODULE: 1 (SDLC)

Q.2 What is software? What is software engineering?

ANS. Software:

Software refers to a collection of computer programs, data, and instructions that enable a computer or a computing device to perform specific tasks or functions. It encompasses both the tangible components of a computer system and the intangible components.

Software can take various forms, including application software and system software . It is essentially the set of instructions that tell a computer what to do and how to do it.

Software Engineering:

Software engineering is a systematic and disciplined approach to the design, development, testing, and maintenance of software. It is a field of engineering that focuses on creating reliable, efficient, and high-quality software solutions to meet specific needs or requirements.

Key aspects of software engineering include:

1. Requirements Engineering: Gathering and analyzing user and system requirements to understand what the software needs to achieve.

2. Design: Creating a detailed plan or blueprint for the software, specifying its structure, components, and interactions.

3. Implementation: Writing the actual code for the software based on the design, following coding standards and best practices.

4. Testing: Rigorous testing to identify and fix bugs, ensure the software meets its requirements, and behaves as expected.

5. Maintenance: Ongoing efforts to update and enhance the software, address issues, and adapt it to changing needs.

6. Project Management: Managing the software development process, including scheduling, resource allocation, and risk management.

Software engineering aims to produce software that is not only functional but also reliable, maintainable, and scalable. It involves a combination of technical skills, methodologies, and tools to ensure that software projects are completed on time and within budget while meeting user expectations and quality standards. Various software development methodologies, such as Agile, Waterfall, and DevOps, are employed to streamline the development process and improve collaboration among software development teams.

Q.2 Explain types of software.

ANS. Software can be broadly categorized into several types based on its functionality and purpose. Here are some common types of software:

1. System Software:

- Operating Systems (OS): These manage hardware resources and provide essential services for computer programs.

- Device Drivers: Software that enables communication between the computer's hardware components and the operating system.

2. Application Software:

- Word Processing Software: Allows users to create, edit, and format text documents (e.g., Microsoft Word, Google Docs).

- Spreadsheet Software: Used for creating, organizing, and analyzing data in tabular form (e.g., Microsoft Excel, Google Sheets).

- Presentation Software: Helps create visual presentations (e.g., Microsoft PowerPoint, Google Slides).

- Database Software: Used to store, manage, and retrieve structured data (e.g., Microsoft Access, MySQL).

- Graphics Software: Enables graphic design and manipulation (e.g., Adobe Photoshop, GIMP).

- Video Editing Software: For editing and enhancing video content (e.g., Adobe Premiere Pro, Final Cut Pro).

- Audio Editing Software: Used to edit and manipulate audio files (e.g., Adobe Audition, Audacity).

- Web Browsers: Software for accessing and navigating the internet (e.g., Google Chrome, Mozilla Firefox).

- Email Clients: Used to send, receive, and manage email messages (e.g., Microsoft Outlook, Gmail).

- Antivirus Software: Protects against viruses, malware, and other security threats (e.g., Norton, McAfee).

3. Utility Software:

- File Management Utilities: Assist in organizing and managing files and folders (e.g., Windows File Explorer, macOS Finder).

- Compression Software: Compresses and decompresses files to save storage space (e.g., WinZip, 7-Zip).

- Backup and Recovery Software: Helps in creating backups and recovering data in case of loss (e.g., Acronis True Image, Time Machine).

- System Optimization Tools: Optimize system performance by cleaning up and maintaining the computer (e.g., CCleaner, Disk Cleanup).

- Disk Defragmenters: Reorganize and optimize data on hard drives for better performance (e.g., Windows Disk Defragmenter).

4. Development Software:

- Integrated Development Environments (IDEs): Provide tools for software development, including code editors, debuggers, and compilers (e.g., Visual Studio, Eclipse).

- Text Editors: Simple software for writing and editing code (e.g., Visual Studio Code, Sublime Text).

- Version Control Software: Helps track changes in code and collaborate with others (e.g., Git, SVN).

5. Entertainment Software:

- Video Games: Interactive software for entertainment purposes (e.g., Fortnite, Minecraft).
- Media Players: Play audio and video files (e.g., VLC Media Player, Windows Media Player).
- Streaming Services: Deliver music, movies, and TV shows over the internet (e.g., Netflix, Spotify).

6. Communication Software:

- Instant Messaging and Chat Apps: Facilitate real-time text, voice, or video communication (e.g., WhatsApp, Skype).
- Social Media Apps: Enable users to interact and share content with others (e.g., Facebook, Twitter).

7. Business and Productivity Software:

- Enterprise Resource Planning (ERP) Software: Helps manage business processes and operations (e.g., SAP, Oracle).
- Project Management Software: Aids in planning, tracking, and managing projects (e.g., Microsoft Project, Trello).

8. Educational Software:

- Educational Games: Make learning fun and interactive (e.g., educational apps for children).

- E-learning Platforms: Provide online courses and educational resources (e.g., Moodle, Coursera).

These are just some of the many types of software available today, and each serves a specific purpose in computing and technology.

Q.3 What is SDLC? Explain each phase of SDLC.

ANS. SDLC, which stands for Software Development Life Cycle, is a structured approach to software development that defines a series of phases or stages to guide the development process from initial planning to software deployment and maintenance. The goal of SDLC is to produce high-quality software that meets or exceeds customer expectations while staying within budget and timeline constraints. The phases of SDLC typically include:

1. Planning :

- Requirements Gathering: The first step involves gathering and documenting the project's requirements by consulting with stakeholders, including users, clients, and developers. These requirements define what the software should do, its features, and constraints.

- Feasibility Study: Assess the technical, operational, and economic feasibility of the project. Determine if the project is worth pursuing and if it aligns with the organization's goals and resources.

- Project Planning: Create a detailed project plan that includes schedules, resource allocation, and a budget. Define the scope, objectives, and deliverables.

2. Design :

- Architectural Design: Define the software's high-level structure, including components, modules, and their interactions. Consider issues such as scalability, security, and performance.

- Detailed Design: Create detailed specifications for each component or module, specifying how they will be implemented. This includes data structures, algorithms, and interfaces.

3. Implementation:

- Coding: Develop the actual software according to the detailed design. Write the code, test it incrementally, and ensure it meets the requirements and design specifications.

- Unit Testing: Test individual units or modules of code to identify and fix bugs and ensure they work as expected.

4. Testing:

- Integration Testing: Combine the individual modules or components and test their interactions and interfaces to ensure they work together correctly.

- System Testing: Evaluate the entire system's functionality to verify that it meets the specified requirements.

- User Acceptance Testing (UAT): Involve end-users or stakeholders to validate that the software meets their needs and expectations.

- Regression Testing: Re-test the software after making changes to ensure that existing functionality has not been adversely affected.

5. Deployment :

- Installation: Deploy the software in the target environment, which may involve configuring servers, databases, and other components.

- User Training: Train end-users and support staff on how to use the software effectively.

- Data Migration: If necessary, migrate data from legacy systems to the new software.

6. Maintenance :

- Maintenance: Address and fix any issues, bugs, or problems that arise during the software's use. This may include updates, patches, and enhancements.

- Support: Provide ongoing support to users, answer questions, and address concerns.

- Monitoring and Performance Tuning: Continuously monitor the software's performance and make optimizations as needed.

7. Closure (Post-Implementation Review):

- Documentation: Update and maintain documentation to reflect the final state of the software.

- Lessons Learned: Conduct a post-implementation review to assess the project's successes and areas for improvement. Capture lessons learned for future projects.

These phases may be iterative or overlapping, depending on the specific SDLC model or methodology being used (e.g., Waterfall, Agile, DevOps). The choice of SDLC model depends on the project's requirements, complexity, and constraints. Each phase plays a crucial role in ensuring the successful development, deployment, and maintenance of software applications.

Q.4 What is DFD? Create a DFD diagram on Flipkart.

ANS. DFD stands for Data Flow Diagram. It is a graphical representation of how data flows through a system. DFDs are used to model and analyze the flow of data through a business process or software application.

A DFD diagram typically consists of four main components:

* External entities: These are the entities outside of the system that interact with it. They can be people, organizations, or other systems.
* Processes: These are the steps that the system takes to transform the input data into output data.
* Data stores: These are the repositories where the system stores its data.
* Data flows: These are the lines that connect the external entities, processes, and data stores. They show how the data moves through the system.

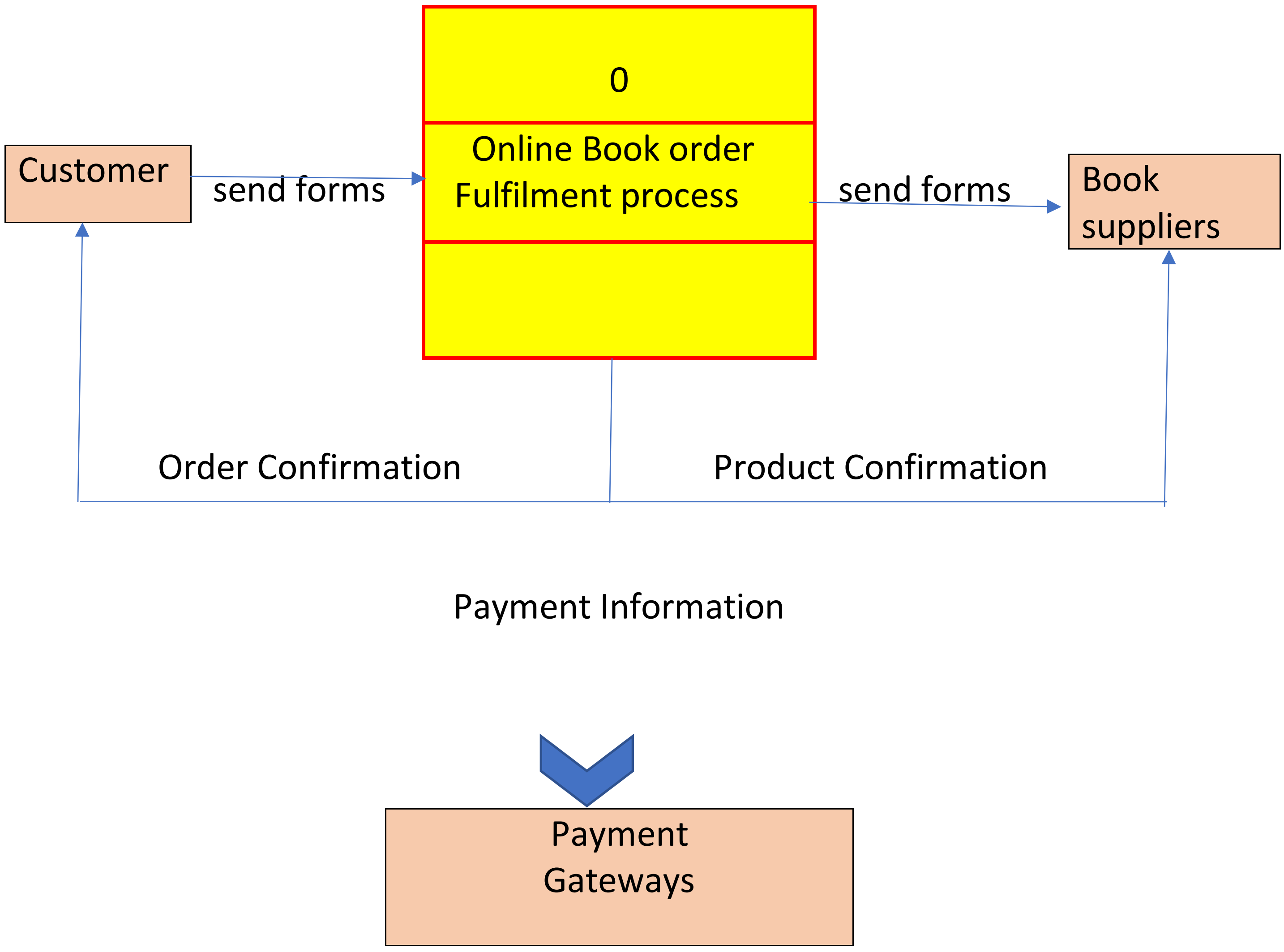
Here is a simple DFD diagram for Flipkart:

DIRECTORY

SERVICE

MFA

MFA token



External entities:

* Customer

* Seller

Processes:

* Browse products
* Add products to cart
* Checkout
* Process payment
* Ship order
* Track order

Data stores:

* Product catalog
* Customer information
* Order information

Data flows:

* Customer browses products and adds them to cart
* Customer checks out and provides payment information
* Flipkart processes the payment and ships the order
* Customer tracks the order

This is just a simple example, and a more detailed DFD diagram could be created to show all of the different processes and data flows involved in Flipkart's system.

DFD diagrams are a useful tool for understanding how a system works and identifying areas where it can be improved. They can also be used to communicate the design of a system to other stakeholders.

Q.4 What is Flow chart? Create a flowchart to make addition of two numbers.

ANS. A flowchart is a graphical representation of a process or algorithm. It uses standardized symbols to represent different steps, decisions, and actions within a process. Flowcharts are commonly used in various fields, such as software development, engineering, business, and education, to visualize and document processes.

Here's a simple flowchart to illustrate how to add two numbers:

Start

↓

Input the first number (A)

↓

Input the second number (B)

↓

Calculate the sum (A + B)

↓

Display the result

↓

End

In this flowchart:

1. The process starts with the "Start" symbol.
2. The first number (A) is input using the input symbol (usually represented as a parallelogram).
3. The second number (B) is input using another input symbol.
4. The sum of the two numbers (A + B) is calculated using a process symbol (usually represented as a rectangle).
5. The result is displayed using the output symbol (usually represented as a parallelogram).
6. Finally, the process ends with the "End" symbol.

This flowchart illustrates a simple addition operation. You would replace the placeholders (A and B) with actual numerical values when working with real numbers. Flowcharts can become much more complex when representing more intricate processes, but they serve as a valuable tool for visualizing and understanding processes step by step.

Q.5 What is Use case Diagram? Create a use-case on bill payment on paytm.

ANS. A use case diagram is a visual representation in Unified Modeling Language (UML) that depicts the functional requirements of a system or application from the perspective of its users and the specific ways they interact with the system. Use case diagrams help in understanding the system's behavior and its various use cases or scenarios.

In the context of bill payment on Paytm, let's create a simple use case diagram. In this scenario, there are two primary actors: "User" and "Paytm System."

Use Case Diagram for Bill Payment on Paytm:

```
        User
         |
         |
         V
  +------------------+
  |  Bill Payment   |
  |    on Paytm    |
  +------------------+
         |
         |
         V
     Paytm System
```

In this diagram:

1. User: This is an external actor, representing the person who interacts with the Paytm system to perform various actions, including bill payment.

2. Bill Payment on Paytm: This is the primary use case depicted as an oval shape. It represents the functionality or feature that the user wants to use within the Paytm system, which is paying bills.

3. Paytm System: This is another external actor, representing the Paytm system itself, which provides the bill payment service to the user.

The arrow between the "User" and "Bill Payment on Paytm" indicates that the "User" interacts with the "Bill Payment on Paytm" use case.

Please note that this is a simplified representation. In a real-world scenario, there may be additional actors and use cases, and you could further expand the use case diagram to include more details, such as associations between actors and use cases, and include additional use cases like "View Bill Details," "Save Payment Method," and others to provide a more comprehensive view of the system's functionality.