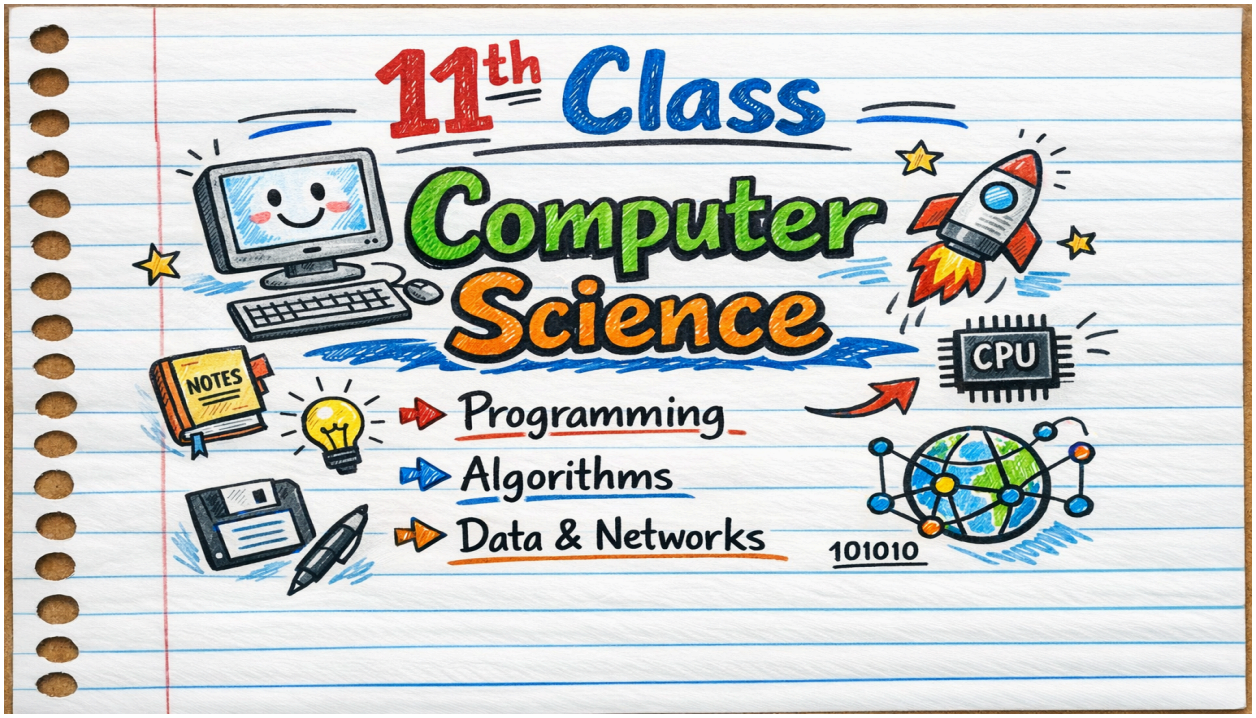


Class: 11th

Subject: Computer Science

Unit 1 Exercise: Introduction to Software Development

📘 EXERCISE

Q.1: Multiple Choice Questions

1. Primary purpose of the Software Development Life Cycle (SDLC) is to:

a) design websites

b) deliver high-quality software within time and cost estimates ✅

c) manage database systems

d) create hardware components

2. A type of requirement specifying system performance:

a) Functional Requirements

b) Non-Functional Requirements ✅

c) Technical Requirements

d) Operational Requirements

3. Role of a framework in the context of SDLC is to:

a) write code from scratch

b) provide a structured foundation with predefined components and architectures ✅

c) manage hardware

d) perform manual testing

4. Software development model involving short cycles or sprints:

a) Waterfall Model

b) Agile Methodology ✅

c) Lean Software Development

d) Scrum

5. Crucial aspect of comprehensive project planning:

a) Understanding the project scope and tasks ✅

b) Deciding the project's colour scheme

c) Hiring a large development team

d) Ignoring potential risks

6. Factor that does not influence cost estimation of a software project:

a) Scope of the project

b) Technology stack

c) Number of meetings held ✅

d) Operational costs

7. The purpose of Use Case Diagrams is to:

a) document the system's architecture

b) identify and document the system's functional requirements ✅

c) illustrate the database schema

d) define the system's user interface design

Q.2: Short Questions

1. Differentiate between functional and non-functional requirements.

Answer:

Functional requirements describe what the system should do, while non-functional requirements describe how the system should perform.

Functional Example:

A system should allow users to log in and register.

Non-Functional Example:

The system should respond within 2 seconds and support 1000 users.

2. Explain why the testing phase is important in SDLC, and provide two reasons.

Answer:

The testing phase is important because it ensures the software is error-free, reliable, and meets requirements before release.

Two reasons:

1. It helps detect and fix bugs early.
2. It improves software quality and user satisfaction.

3. Continuous Integration in Agile Methodology and its importance.

Answer:

Continuous Integration (CI) is a practice where developers frequently merge their code into a shared repository, and each integration is automatically tested.

Importance:

- Detects errors early
- Reduces integration problems
- Improves software quality and development speed

Example:

Every time a developer updates code, it is automatically tested in the system.

4. Steps of Risk Assessment and Management and their importance.

Answer:

Risk assessment and management involve identifying and controlling possible risks in a project.

Steps:

- Identify risks (technical, operational, or external).

- Analyze risks (check likelihood and impact).
- Develop mitigation strategies (reduce or control risks).
- Monitor risks continuously.

Importance:

- Reduces project failure chances
- Helps in better planning
- Ensures smooth project execution

5. Purpose of a Use Case Diagram.

Answer:

A Use Case Diagram shows the interaction between users (actors) and the system and helps identify system requirements.

Example:

In a library system, a student can search books, borrow books, and return books.

6. Sequence Diagram vs Activity Diagram.

Answer:

A Sequence Diagram shows how objects interact with each other in time order, while an Activity Diagram shows the flow of activities in a process.

Sequence Diagram Example:

User logs in ⟶ system verifies ⟶ access granted.

Activity Diagram Example:

Start ⟶ login ⟶ select option ⟶ process ⟶ end.

7. Factory Pattern and difference from direct object creation.

Answer:

The Factory Pattern is a design pattern that provides a method to create objects without exposing the creation logic to the user.

Instead of directly creating objects using new, a factory method is used.

Example:

Without Factory: Car c = new Car();

With Factory: Car c = CarFactory.getCar();

Difference:

Direct creation exposes object creation details, while Factory Pattern hides the creation process and improves flexibility.

Q.3: Long Questions

🌟 Q1: Design a flowchart for a user registration process in a software application. Explain in detail.

Answer:

A flowchart is a graphical tool used to represent a process step-by-step using standard symbols. It helps in understanding how a system works in a clear and logical way. In software development, flowcharts are commonly used during the design phase of SDLC to visualize system processes before coding.

The user registration process is one of the most basic and important functions in almost every software system such as websites, mobile apps, and online platforms.

User Registration Process:

1. Start

The process begins when the user opens the application or website and selects the “Sign Up” or “Register” option.

2. Enter Registration Information

The system asks the user to provide required details such as:

- Full Name
- Email Address

- Password
- Phone Number (optional in some systems)

This step is known as input phase, where data is collected from the user.

3. Input Validation

After entering details, the system checks whether the information is correct and complete. Validation rules may include:

- Email format should be correct
- Password should meet security requirements
- No field should be left empty

4. Decision Point (Validation Check)

At this stage, the system makes a decision:

- If information is valid: Move to account creation
- If information is invalid: Show an error message and ask the user to re-enter details

This step ensures data accuracy and prevents incorrect entries.

5. Account Creation

If all details are valid, the system creates a new user account. This involves generating a unique user ID and preparing a profile for the user.

6. Store Data in Database

The system saves user information securely in the database. This is important for future login and authentication purposes.

7. Confirmation Message

After successful registration, the system displays a message such as:

- “Registration Successful” or “Account Created Successfully”.
- This step confirms that the process is completed without errors.

8. End

The process ends after successful registration, and the user can now log in using their credentials.

Flow of Process (Logical Representation):

- Start ⟶ Enter Details ⟶ Validate Input ⟶
- If Valid ⟶ Create Account ⟶ Store Data ⟶ Success Message ⟶ End
- If Invalid ⟶ Error Message ⟶ Re-enter Details

Importance of This Flowchart:

- It helps developers understand the system clearly before coding
- It ensures proper validation of user data
- It improves system reliability and reduces errors
- It provides a clear step-by-step structure for implementation

Summary:

The user registration flowchart represents a complete process of how a system handles new user sign-up. It ensures that user data is properly collected, validated, stored securely, and confirmed, making the system efficient, reliable, and user-friendly.

🌟 Q2: Imagine you are managing a project to develop a simple mobile application. Describe how you would use the Agile Methodology to handle this project.

Answer:

Agile Methodology is a flexible and iterative software development approach used to develop software in small parts instead of completing everything at once. It focuses on continuous improvement, customer feedback, and delivering working software in short cycles called sprints or iterations.

If I am managing a project to develop a simple mobile application (for example, a to-do list app or a student app), I would use Agile in the following way:

1. Project Planning and Requirement Understanding

First, I would discuss with the client or users to understand basic requirements such as:

- Login system
- Add/edit/delete tasks
- User-friendly interface

Instead of finalizing everything at once, requirements are kept flexible for future updates.

2. Breaking Project into Sprints (Iterations)

The whole project is divided into small parts called sprints. Each sprint focuses on a specific feature.

Example:

- Sprint 1: Login and registration system
- Sprint 2: Task creation feature
- Sprint 3: Task editing and deletion
- Sprint 4: UI improvement and testing

3. Development in Small Cycles

In each sprint, the development team:

- Designs feature
- Writes code
- Tests the feature
- Fixes bugs

Each sprint produces a working version of the app.

4. Continuous Testing

Testing is done in every sprint instead of waiting until the end. This ensures:

- Early detection of bugs
- Better software quality
- Faster improvements

5. Customer Feedback

After each sprint, the working version is shown to the client or users. Their feedback is collected and used to improve the next sprint.

Example: If users say the interface is difficult, it is improved in the next iteration.

6. Continuous Integration

All small parts of the application are regularly combined into one system. This ensures:

- No integration errors
- Smooth functioning of all features together

7. Final Delivery

After completing all sprints and improvements, the final version of the mobile application is delivered to users.

Advantages of Using Agile:

- Flexible and easy to change requirements
- Early delivery of working software
- Continuous improvement through feedback
- Better customer satisfaction

Summary:

Using Agile Methodology for a mobile application project ensures that the software is developed step by step, tested continuously, and improved based on user feedback. It results in a high-quality, user-friendly, and efficient application.

🌟 Q3: Consider an online banking system. Create a Use Case Diagram to show the interactions between customers, bank staff, and the system.

Answer:

A Use Case Diagram is a type of UML diagram that shows the interaction between users (actors) and the system. It represents the functional requirements of a system from the user's perspective and helps in understanding what different users can do in the system.

In an online banking system, different users interact with the system for different purposes. The main actors are:

- Customer
- Bank Staff
- Online Banking System (itself as the system boundary)

Explanation of Actors and Their Functions:

1. Customer

The customer is the main user of the banking system.

Customer actions (use cases):

- Login to account
- Check account balance
- Transfer money
- View transaction history
- Pay bills

2. Bank Staff

Bank staff manage and monitor banking operations.

Bank staff actions (use cases):

- Manage customer accounts
- Approve transactions
- Update customer information
- Monitor system activity

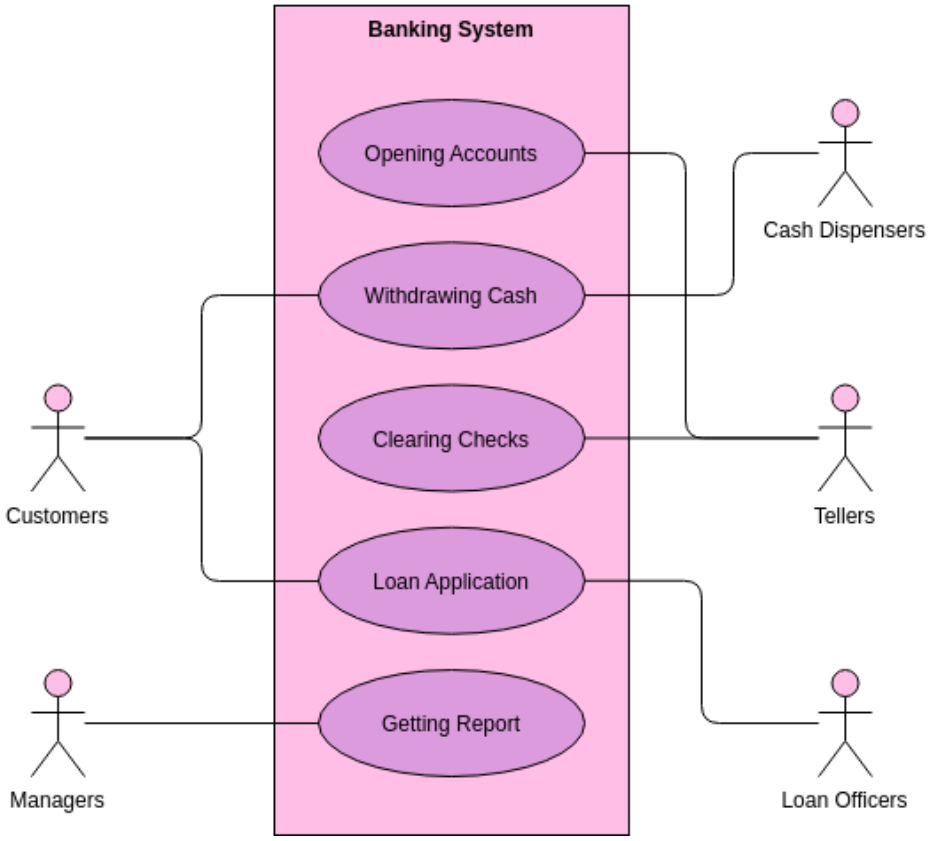
3. Online Banking System

The system provides services and processes all requests.

System functions:

- Authenticate users
- Process transactions
- Maintain database
- Generate reports

Use Case Diagram:



Simple Explanation of Diagram:

- Customers interact with the system to perform banking tasks.
- Bank staff manage and control banking operations.
- The system processes all requests and ensures security and accuracy.

Importance of Use Case Diagram:

- Shows system functionality clearly
- Helps in understanding user roles
- Useful in requirement analysis
- Improves communication between developers and clients

Summary:

The Use Case Diagram for an online banking system clearly shows how customers and bank staff interact with the system. It helps in identifying system requirements and designing the system in a structured and user-friendly way.

🌟 Q4: You are developing a food delivery application. Create a Sequence Diagram to show the process of placing an order, from the customer selecting items to the delivery of the order.

Answer:

A Sequence Diagram is a type of UML diagram that shows how different objects or users interact with each other in a step-by-step sequence over time. It focuses on the order of messages exchanged between participants in a system.

In a food delivery application, the sequence diagram helps us understand how an order moves from the customer to the restaurant and finally to the delivery process.

Main Actors (Objects) in the System:

- Customer
- Food Delivery App (System)
- Restaurant
- Delivery Person (Rider)

Step-by-Step Process of Order Placement:

1. Customer selects items

- The customer opens the app and selects food items from the menu.

2. Order request sent to system

- The app sends the order details to the system for processing.

3. System forwards order to restaurant

- The system forwards the order to the selected restaurant.

4. Restaurant confirms order

- The restaurant checks availability of items and confirms the order.

5. System updates order status

- The system updates the status as “Order Confirmed”.

6. Delivery person is assigned

- The system assigns a delivery rider to pick up the order.

7. Order pickup

- The rider collects the food from the restaurant.

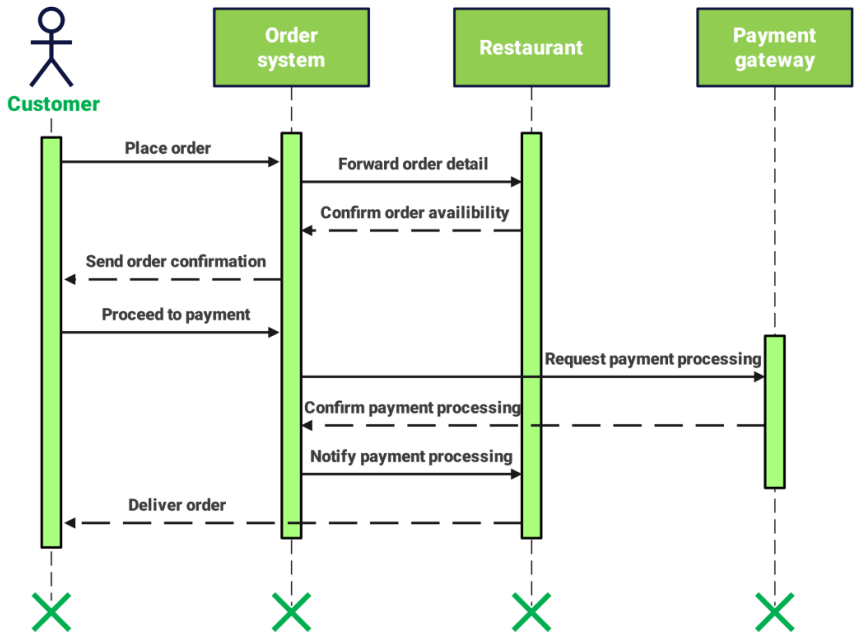
8. Delivery to customer

- The rider delivers the order to the customer’s address.

9. Order completion

- The system marks the order as delivered successfully.

Sequence Diagram:



Importance of Sequence Diagram:

- Shows step-by-step interaction clearly
- Helps understand system behavior over time
- Useful in designing real-time applications
- Helps developers implement correct workflow

Summary:

The sequence diagram for a food delivery application clearly shows how an order moves from the customer to the restaurant and then to the

delivery person. It helps in understanding the complete flow of communication between different components of the system.

🌟 Q5: Discuss the importance of software development tools in the software development process.

Answer:

Software development tools are applications that help developers create, test, debug, and maintain software efficiently. These tools are used throughout the Software Development Life Cycle (SDLC) to improve speed, accuracy, and quality of software. Without these tools, software development would be slow, complex, and error-prone.

Software development tools mainly include language editors, translators, and debuggers, each playing an important role in building reliable software systems.

(a) Role of Language Editors, Translators, and Debuggers

1. Language Editors (Code Editors)

Role:

Language editors are tools used to write and edit program code in different programming languages. They provide a comfortable environment for developers to write code easily.

Explanation:

These editors make coding easier by highlighting syntax, organizing code, and providing suggestions while typing. This helps developers write clean and error-free code.

Example:

- Visual Studio Code (VS Code)
- Notepad++

Importance:

Language editors improve efficiency because they speed up coding. They also reduce mistakes by showing errors during typing, making development easier and more organized.

2. Translators (Compiler and Interpreter)

Role:

Translators convert high-level programming language into machine language so that the computer can understand and execute the program.

Types:

- Compiler: Converts the entire program into machine code at once.
- Interpreter: Converts code line by line during execution.

Explanation:

Since computers only understand binary language (0s and 1s), translators are necessary to convert human-readable code into machine-readable form.

Example:

- GCC compiler for C/C++
- Python interpreter

Importance:

Translators ensure accuracy of execution and allow programs to run properly on computers. Without them, code written by developers cannot be executed.

3. Debuggers

Role:

Debuggers are tools used to find and fix errors (bugs) in software programs.

Explanation:

Debuggers help developers run programs step by step, check variable values, and locate errors in the code. This makes it easier to identify the exact location of a problem.

Example:

- GDB (GNU Debugger)

- Visual Studio Debugger

Importance:

Debuggers improve software reliability and correctness by removing errors before the software is released to users.

(b) Contribution of Each Tool to Efficiency and Accuracy

Language Editors:

They make coding faster and easier by providing auto-suggestions and error detection, improving overall efficiency.

Translators:

They ensure that code is correctly converted into machine language, improving accuracy and correct execution of software.

Debuggers:

They help identify and fix bugs, improving software quality, stability, and reliability.

Summary:

Software development tools are essential in modern programming. Language editors improve coding efficiency, translators ensure correct execution, and debuggers improve software quality by removing errors. Together, they make software development faster, more accurate, and more reliable.

Note:

This chapter is designed to provide a solid foundation of knowledge, with the goal of deepening understanding and encouraging further exploration of the subject. The content has been carefully selected to support effective learning and inspire students to engage with the topic more deeply.

Author: Muhammad Asghar

Purpose: To contribute to education by offering insightful, valuable content that enhances learning and understanding.

Copyright & Usage Policy