Notes by Care Link Educational Hub

Unit 1: Introduction to Software Development

Long Questions with Diagrams

1. Software Development and the SDLC

Q.1 What is software development?

Answer: Software development is the process of creating computer programs that perform particular tasks. It includes writing code, testing it, and fixing problems. Software helps solve real-life problems and makes daily life easier.

For example, software lets us chat with friends on social media, manage money through banking apps, and learn with educational games.

Q.2 What is the Software Development Life Cycle (SDLC)?

Answer: The Software Development Life Cycle (SDLC) is a step-by-step process for developing software from the first idea to its final release and maintenance. Its main goal is to create high-quality software that meets customer needs, stays within budget, is completed on time, and works efficiently.

Q.3 What is a framework in software development? Explain.

Answer: A framework is like a toolbox containing ready-made tools, rules, and structures that help developers build software faster and more efficiently. It includes pre-built components, so developers do not have to start from the beginning. This saves time, keeps the work consistent, and makes maintenance easier.

For example, Django is a web framework with ready-made features such as user login, database management, and page templates.

Q.4 Discuss the stages involved in the Software Development Life Cycle (SDLC).

Answer: The SDLC has several connected stages. Each stage has its own goal and activities, and together they help teams develop software in an organised way.

1. Requirement Gathering: Find out what the software must do and what users expect. This includes interviews, surveys, observations, and reviewing existing documents.
2. Design: Prepare a plan or blueprint for the software. Developers create diagrams, models, interface mock-ups, and the system architecture.
3. Coding or Development: Write the actual program using a language such as Python, Java, or C#.
4. Testing: Check the software for errors, confirm that its functions work, test performance under heavy use, and check compatibility with devices, operating systems, and browsers.
5. Deployment: Install the tested software, configure it for the user or organisation, and test it in the real environment.
6. Maintenance: Fix later errors, add useful features, and update the software for security or compatibility changes.

Figure 1. Software Development Life Cycle: the stages repeat when maintenance leads to new requirements.

Additional question: Why is the design phase compared with preparing blueprints for a house?

Answer: Blueprints show the structure and layout of a house before construction begins. In the same way, the design phase shows the structure, layout, and functions of software before coding starts. This gives the team a clear plan, keeps the work organised, and helps the final software meet user requirements.

2. Software Development Methodologies

Q.5 What are software development methodologies?

Answer: Software development methodologies are organised approaches for planning, creating, and managing software projects. They make the process systematic and efficient and help teams produce good-quality software. They also guide the team in managing tasks, schedules, and resources.

Q.6 Explain the importance of software process models in software development.

Answer: Software process models are simplified descriptions of the steps used in the SDLC. They help teams organise and control development so that software is built systematically and meets user needs.

They provide predictability because a defined process helps teams forecast results and manage risks. They improve efficiency by reducing wasted effort and making work smoother. They also support quality by including quality checks throughout development.

Q.7 What is the Waterfall model? Explain its benefits and limitations.

Answer: The Waterfall model is one of the earliest and simplest software-development methods. It follows a straight, step-by-step process: one phase must be completed before the next phase begins.

Main phases:

1. Requirements: Gather and record what the software must achieve.
2. Design: Plan the architecture and user interface.
3. Coding: Write the program based on the design.
4. Testing: Find and correct errors.
5. Deployment: Release the software for users.
6. Maintenance: Update the software and fix later problems.

Benefits:

- It is simple to understand because the phases are clear.
- It is easy to manage and track because work moves in order.
- It suits small projects with fixed requirements.

Limitations:

- It is inflexible because changes after a phase is finished are difficult and costly.
- It is not ideal for large or complex projects.
- It assumes that all requirements are known at the beginning, which may not be true.

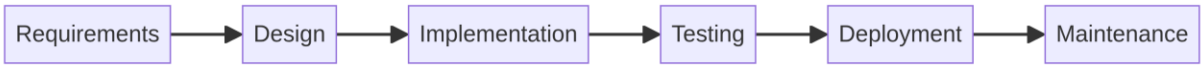

Figure 2. Waterfall model: each stage moves forward in a fixed order.

Q.8 What is Agile methodology? Explain its key practices, benefits, and limitations.

Answer: Agile is a flexible and adaptive approach to software development. It delivers small working parts quickly and adjusts to changes as the project continues. Unlike Waterfall, Agile develops software through short repeated cycles called iterations or sprints. Teams can release working software early, collect feedback, and improve the product continuously.

Key practices:

- Continuous Integration: Add code changes regularly to a shared place and test them early.
- Test-Driven Development (TDD): Write tests before the program code so that the code meets the requirements.
- Pair Programming: Two developers work together; one writes code while the other reviews it.

Benefits:

- High flexibility: The team can accept changes even after development has begun.
- Better customer satisfaction: Customers see working software regularly and give feedback.

Limitations:

- Large projects with many teams need careful coordination and communication.
- Agile requires active participation from stakeholders.
- The final timeline and scope are harder to predict because the project changes over time.

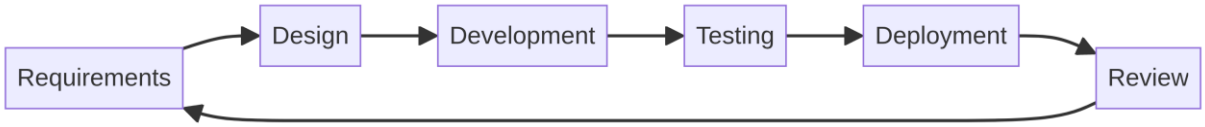

Figure 3. Agile methodology: short cycles repeat through review and feedback.

Q.9 Compare and contrast the Waterfall model and Agile methodology.

Answer: Waterfall is linear and sequential. It follows Requirements → Design → Implementation → Testing → Maintenance, with limited changes after work starts. Customer involvement is usually greater at the beginning and end, testing mainly follows development, and delivery is often one final release. It works well for fixed-scope projects.

Agile is iterative and incremental. It uses repeated sprints containing planning, development, testing, and review. It welcomes changes, involves customers throughout the project, tests continuously, and delivers small working parts frequently. It works well for projects whose needs may change.

Agile usually lowers risk through frequent feedback and early error detection, while Waterfall can have higher risk because problems may be found late. Waterfall emphasises detailed documentation and a fixed plan; Agile gives greater priority to working software and adaptation.

3. Project Planning and Risk Management

Q.10 What are the important steps in project planning and management for a software project?

Answer: Planning a software project is like planning a trip: the team needs a destination, a time plan, and a budget. Good planning considers all important details before work begins and helps reduce risks.

Important steps include:

1. Understand requirements and define the project scope.
2. Set clear goals and deliverables.
3. Divide the work into tasks and set timelines or milestones.
4. Assign responsibilities to developers, designers, testers, and other team members.
5. Define the processes, tools, and methodology to be used.
6. Estimate the cost, including staff, technology, duration, quality assurance, and risk reserves.
7. Identify possible risks and prepare ways to reduce them.
8. Monitor progress, communicate regularly, and address delays or changes.

Additional question: What makes software companies such as Microsoft highly valuable?

Answer: Software companies can be highly valuable because their products can serve many users without the cost increasing at the same rate. They may earn regular income, continue to innovate, and build strong groups of connected products and services. Their success shows how software supports digital change and economic growth.

Q.11 What are the steps involved in risk assessment and management in a software project?

Answer: Risk assessment and management help a project stay on schedule, within budget, and at the required quality level.

1. Identify risks: List possible technical, operational, and external risks, such as untested technology, a shortage of resources, or market changes.
2. Analyse risks: Estimate how likely each risk is and how much damage it could cause.
3. Prepare mitigation strategies: Plan ways to lower the chance of the risk or reduce its effect. Examples include adding extra time, arranging backup resources, and doing additional testing.
4. Monitor and review: Continue checking for new risks and update the plans when circumstances change.

Q.12 What happens during the Execution phase of software development?

Answer: During execution, the team follows the project plan to write code, create designs, and build the software. The work requires teamwork, coordination, communication, and regular progress updates so that the project stays on schedule and the product is completed successfully.

Q.13 What is Quality Assurance (QA), and how does it help a software project?

Answer: Quality Assurance (QA) makes sure that the project and its software meet the required standards and work correctly. It includes testing the software, reviewing code, collecting feedback from users or stakeholders, and checking progress regularly. QA helps prevent mistakes, improves reliability, and makes the final product easier and safer to use.

4. Graphical Representation and UML

Q.14 What is the purpose of graphical representation of software systems?

Answer: Graphical representation uses diagrams to show the structure and behaviour of a software system. Diagrams make complicated ideas easier to understand, improve communication between developers and stakeholders, and help teams plan and manage the system.

Q.15 What is UML? Describe a Use Case Diagram and explain how to identify use cases.

Answer: UML means Unified Modeling Language. It is a standard way to show the design of a software system visually. UML diagrams make it easier to understand how system parts interact.

A Use Case Diagram shows the interactions between users, called actors, and a system. A use case is a complete set of steps that an actor follows to reach a particular goal. Use cases are based on the functions the system must provide.

Uses of a Use Case Diagram:
1. It records the system's functional requirements.
2. It shows how different users interact with the system.
3. It helps plan development and design test cases.

How to identify use cases:
1. Identify actors, such as users or other systems.
2. Define the goals or tasks each actor must complete.
3. Describe the interactions needed to reach those goals.
4. Review the use cases with stakeholders to check that they are correct.

CARE LINK EDUCATIONAL HUB

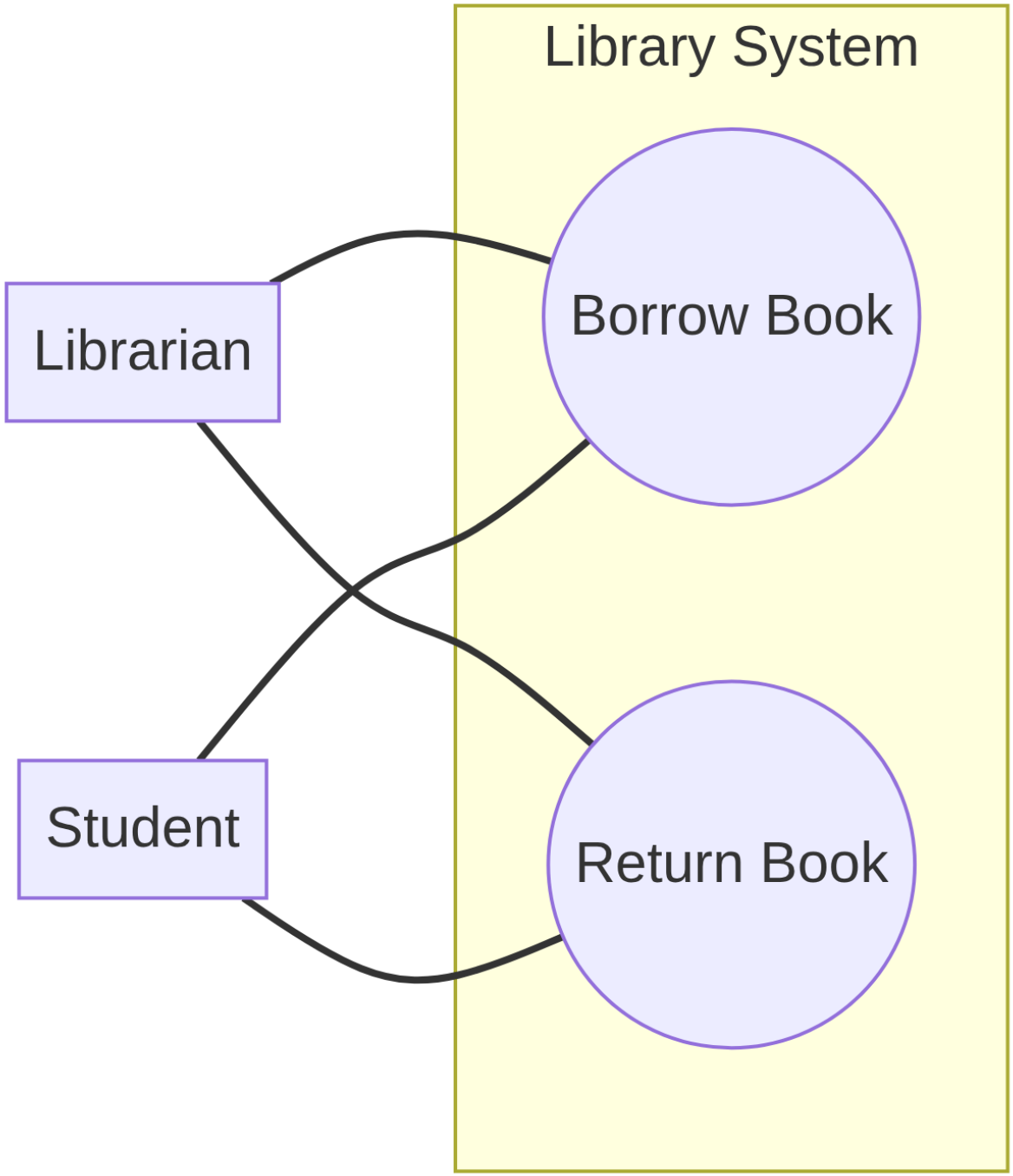

Figure 4. Library Use Case Diagram: Librarian and Student interact with Borrow Book and Return Book.

Q.16 What is a Class Diagram? Explain with an example.

Answer: A Class Diagram is like a map of the structure of a system. It shows classes, their attributes, their methods, and the relationships between them.

Example: Organising a room. Room is the larger structure and contains Box objects. Room has attributes name and size. Box has attributes label and contents and methods such as open() and close(). Toys Box, Book Box, and Clothes Box are specialised types of Box, each with its own list of items.

EDUCATIONAL HUB

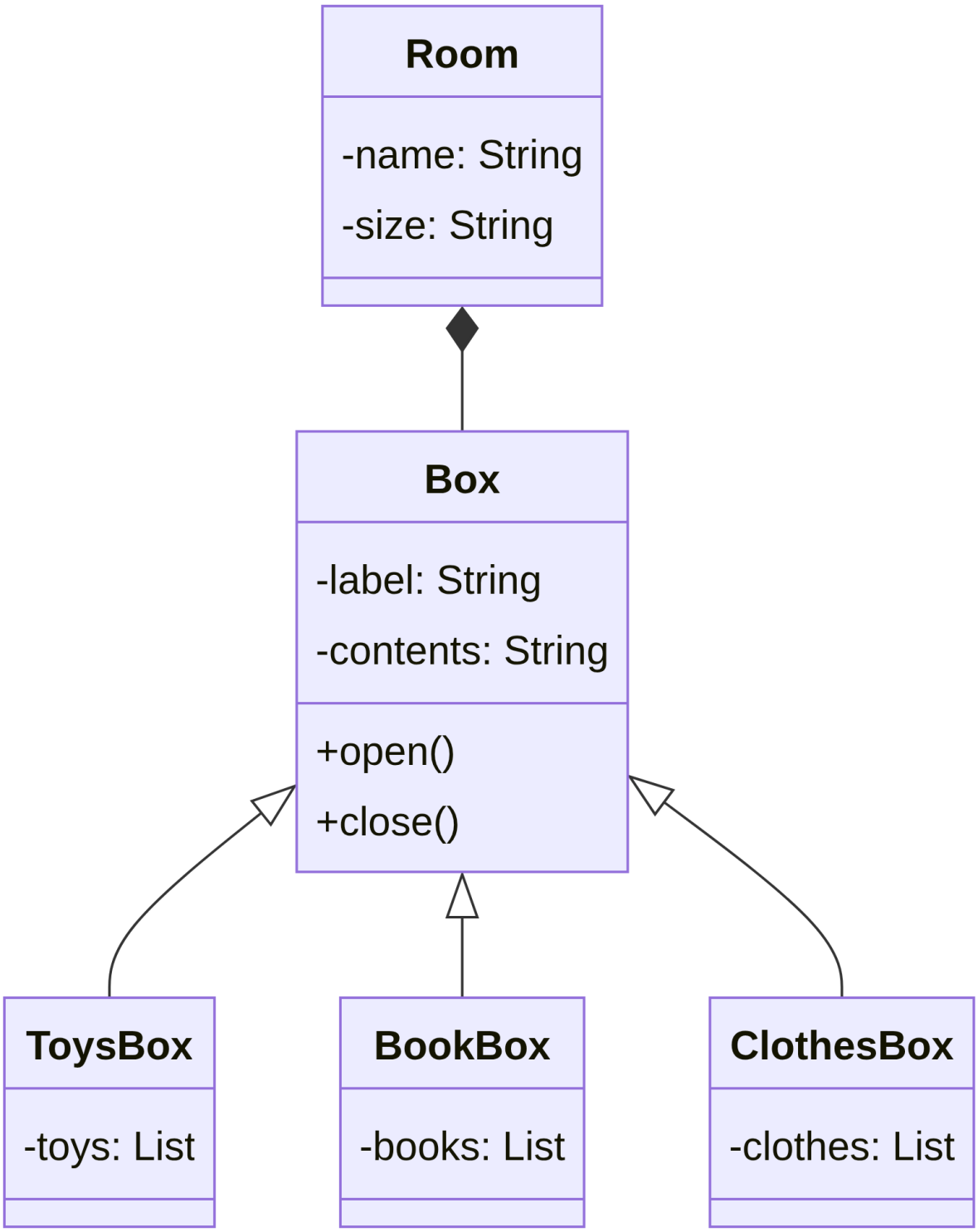

Figure 5. Class Diagram for organising a room.

Q.17 What are Sequence Diagrams? Explain with an example.

Answer: A Sequence Diagram shows how objects interact in a particular order. It focuses on the messages or actions exchanged between objects over time.

Example: A user organising items into boxes. The user opens the Toys Box, Book Box, and Clothes Box, puts the correct items into each box, and then closes the boxes. The diagram shows the order of these messages.

EDUCATIONAL HUB

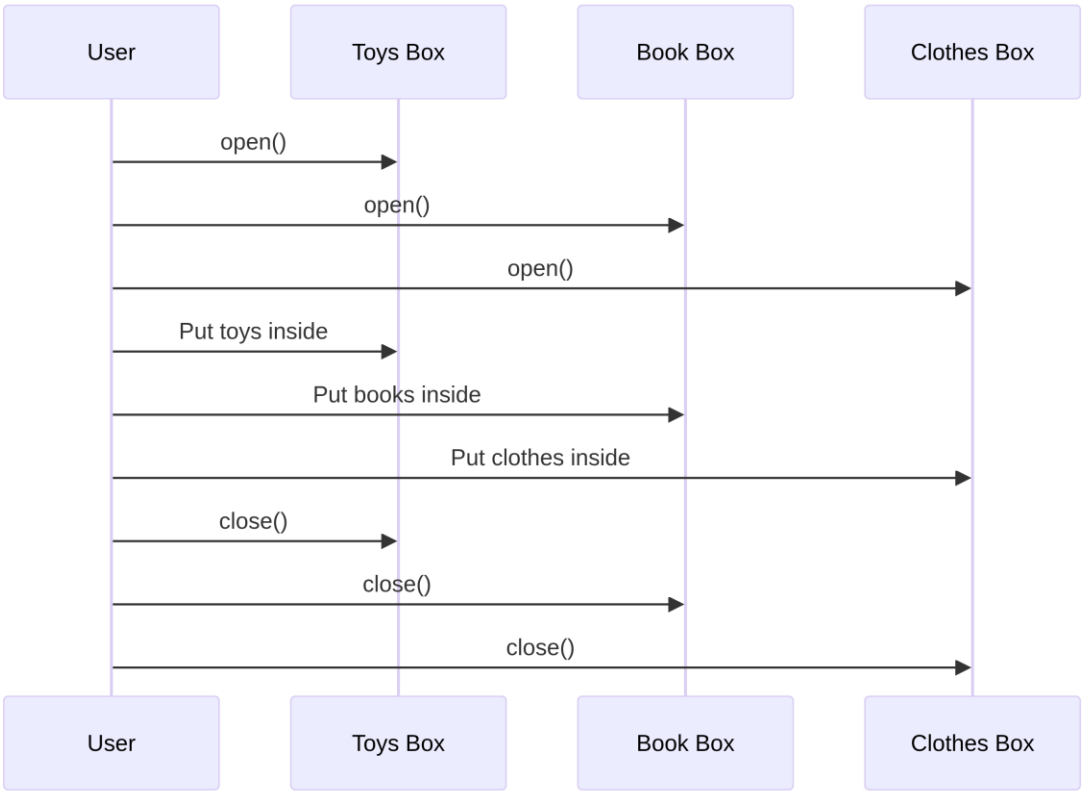

Figure 6. Sequence Diagram for putting toys, books, and clothes into labelled boxes.

Q.18 What are Activity Diagrams? Explain with an example.

Answer: An Activity Diagram shows the flow of activities or steps in a process. It is useful for explaining the logic of a complex operation from beginning to end.

Example: Restaurant order process. The process starts when the customer places an order. The kitchen prepares the food, and the system checks whether the food is ready. If it is not ready, preparation continues. If it is ready, the order is delivered and the process ends.

EDUCATIONAL HUB

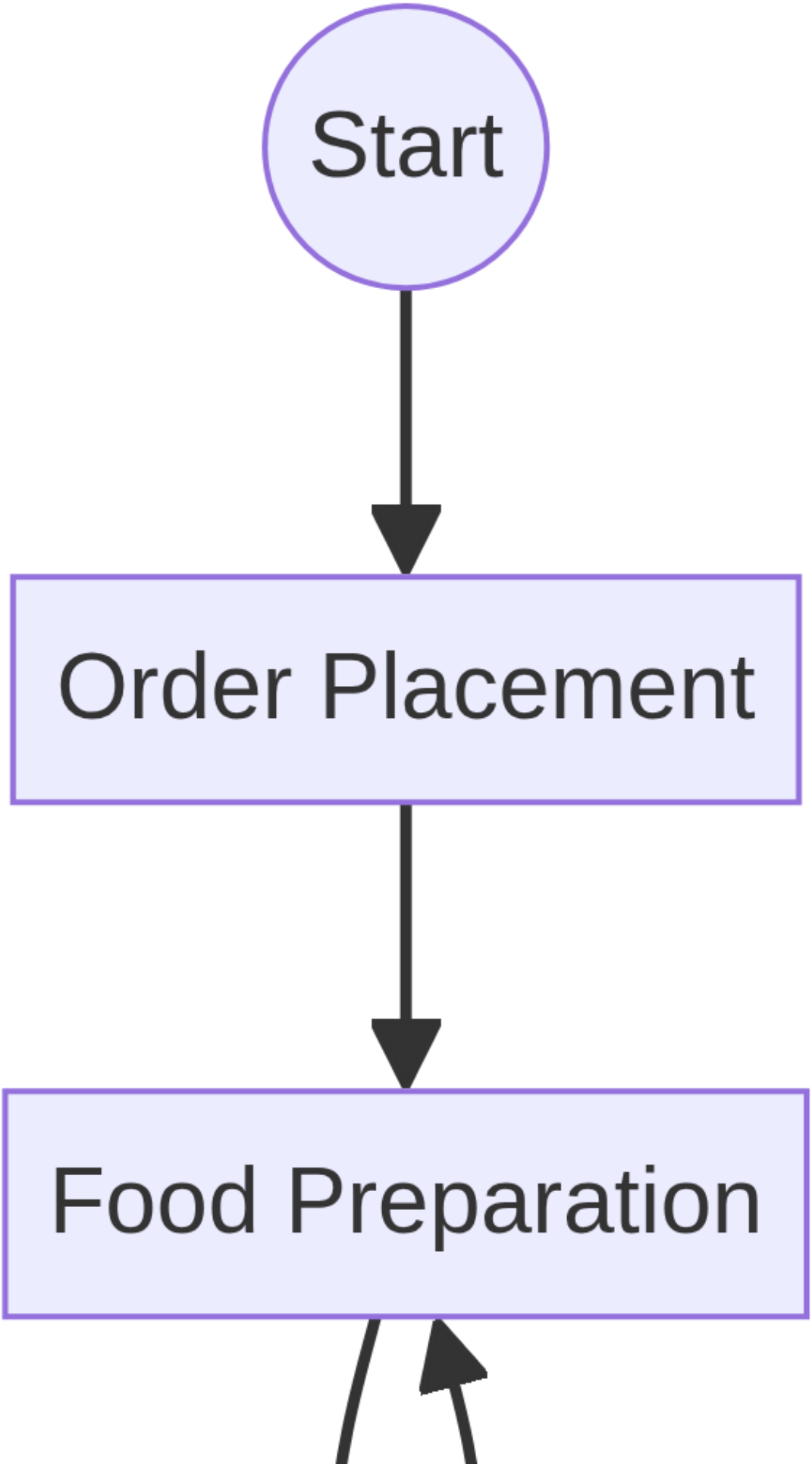
Start
Order Placement
Food Preparation

Figure 7. Activity Diagram for a restaurant order with a Food Ready? decision.

Q.19 How are UML diagrams used in different stages of software development to improve understanding and communication?

Answer: During planning, UML diagrams map requirements and the system design before code is written. This helps developers and stakeholders see the structure and functions early.

During development, UML diagrams act as a guide to the system structure, relationships between components, and interactions between parts.

For communication, UML provides a common visual language for technical and non-technical people. Clients, managers, and developers can discuss how the system works without everyone needing deep programming knowledge.

5. Design Patterns

Q.20 Explain design patterns and discuss some commonly used patterns.

Answer: Design patterns are common, reusable solutions to common software-design problems. Developers can adapt them as templates to make systems more flexible, easier to maintain, and easier to understand.

1. Singleton Pattern: Ensures that one object or resource is created only once and reused. Example: one database connection for an application.
2. Factory Pattern: Creates the required object through a special factory while hiding the creation details. Example: a vehicle factory creates the type of vehicle requested.
3. Observer Pattern: Automatically informs interested objects when a source changes. Example: subscribers receive news updates from a publisher.
4. Strategy Pattern: Provides different methods for solving a task, allowing the program to choose the suitable method. Example: a shopping cart chooses a payment strategy such as card or PayPal.

Additional activity: Identify a real-world situation where a design pattern can be used.

Answer: The Observer pattern can be used in an online food-delivery application. The order status is the subject, while the customer and driver are observers. Whenever the order is confirmed, prepared, or delivered, the application sends an update to the interested users. This keeps everyone informed without each user repeatedly checking the system.

Q.21 What are the applications of design patterns in software design?

Answer: Design patterns are used to reduce code complexity by giving the program a clear structure. They improve code reuse because developers can adapt proven solutions instead of starting over. They also improve communication because developers can use common names such as Singleton or Observer when discussing design decisions. Overall, patterns help create robust, maintainable, and scalable systems that can adapt to changing requirements.

Do You Know? Give an example of a design pattern used in popular software frameworks.

Answer: The Model–View–Controller (MVC) pattern is widely used in software frameworks such as Ruby on Rails and Angular. It separates the data and rules, the user interface, and the control logic so that the application is easier to manage.

6. Software Debugging and Testing

Q.22 Why are software debugging and testing critical stages in development?

Answer: Debugging and testing are critical because they help confirm that software meets its requirements, works as intended, and does not contain serious errors. Testing finds problems, while debugging investigates their causes and corrects the code. Together, they improve software quality, reliability, and user satisfaction.

Q.23 What is debugging, and why is it important in software development?

Answer: Debugging is the process of finding and fixing bugs or errors in software. A bug is a mistake that makes a program behave unexpectedly. Developers observe the program, locate the source of the problem, and change the code to correct it.

Useful debugging methods include debuggers, print statements that show variable values, and code reviews in which

another developer checks the code. Debugging is important because it prevents crashes and incorrect results and helps the software meet user requirements.

Practical activity: Write a unit test for a simple function that adds two numbers.

Answer: The following Python example defines an add_numbers function and tests it with positive, negative, and zero values.

```
# math_functions.py
def add_numbers(a, b):
    return a + b

# test_math_functions.py
import unittest
from math_functions import add_numbers

class TestMathFunctions(unittest.TestCase):
    def test_add_numbers(self):
        self.assertEqual(add_numbers(2, 3), 5)
        self.assertEqual(add_numbers(-1, 1), 0)
        self.assertEqual(add_numbers(0, 0), 0)

if __name__ == '__main__':
    unittest.main()
```

Explanation: The function is tested with several inputs, and assertEqual checks whether the actual answer matches the expected answer. Save the two files and run test_math_functions.py to see whether the tests pass.

Q.24 What is testing? Explain its types.

Answer: Testing is the process of checking whether software works correctly and meets its requirements. It begins with small parts and gradually moves to the complete system, including feedback from users.

1. Unit Testing: Checks individual components, such as a function or method, separately.
2. Integration Testing: Checks whether different components work correctly together and whether their data and interfaces connect properly.
3. System Testing: Checks the complete application as one system, including its functions, performance, security, and required standards.
4. Acceptance Testing: Checks whether the software is ready for release and meets the expectations of end users or clients.

Acceptance testing is also called User Acceptance Testing (UAT) because it is normally performed by the end users to confirm that the software meets their needs.

7. Software Development Tools and Platforms

Q.25 What are software development tools?

Answer: Software development tools help developers create, test, and maintain software. They support writing code, finding and correcting errors, organising projects, and managing the development process effectively.

Q.26 Discuss the purpose and examples of language editors, translators, debuggers, and IDEs.

Answer: Language editors, also called code editors, provide a convenient place to write and change code. Features such as syntax highlighting, auto-completion, and code formatting improve productivity, reduce typing mistakes, and make code easier to read. Examples include Notepad++ and Visual Studio Code.

Translators change high-level programming code into machine instructions that a computer can run. An interpreter translates and runs code one line at a time, while a compiler translates the complete program at once. Examples include a Python interpreter and GCC for C or C++. Translators allow programs to run, identify syntax errors, and may improve performance.

Debuggers help developers find and correct errors. They can run code step by step, inspect variables, and identify run-time or logic errors. Examples include GDB and the Visual Studio Debugger.

An Integrated Development Environment (IDE) combines tools such as an editor, compiler, debugger, and version-control support in one application. Examples include Visual Studio for .NET and C++ development and PyCharm for Python development.

Q.27 What are online and offline computing platforms?

Answer: Online platforms are cloud-based environments that developers access through the internet to write, run, and test code. Examples include Repl.it and Gitpod.

Offline platforms are local development environments installed on a computer. A locally installed IDE is an example. It can be used without depending on an online coding platform.

Q.28 Define source-code repositories.

Answer: Source-code repositories are platforms that store, manage, and track changes to program code. They support version control, so several developers can work on one project with fewer conflicts. Repositories keep a history of changes, making it possible to review or restore earlier versions. Examples include GitHub, which is widely used for open-source projects, and Bitbucket, which supports private and public repositories.

CARE LINK
EDUCATIONAL HUB