

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

(In the Name of Allah, the Most Merciful, the Most Compassionate.)

COMPUTER SCIENCE AND ENTREPRENEURSHIP

11



**PUNJAB EDUCATION, CURRICULUM,
TRAINING AND ASSESSMENT AUTHORITY**

This textbook is based on Updated/Revised National Curriculum of Pakistan 2023 and has been approved by the Punjab Education, Curriculum, Training and Assessment Authority (PECTAA).

All rights are reserved with the PECTAA.

No part of this textbook can be copied, translated, reproduced or used for preparation of test papers, guidebooks, keynotes and helping books.

Contents

Unit No.	Unit Name	Page No.
1	Introduction to Software Development	01
2	Python Programming	20
3	Algorithms and Problem Solving	41
4	Computational Structures	56
5	Data Analytics	67
6	Emerging Technologies	86
7	Legal and Ethical Aspects of Computing System	101
8	Online Research and Digital Literacy	115
9	Entrepreneurship in Digital Age	125
	Answers	141

Authors:

- **Prof. Dr. Muhammad Atif Chattha**
Dean Faculty of Computer Science, Lahore Garrison University, D.H.A, Lahore.
- **Prof. Dr. Syed Waqar ul Qounain Jaffry**
Chairman Department of IT, University of The Punjab,
Allama Iqbal Campus, Lahore.

Experimental
Edition

Reviewer:

- **Dr. Arshad Ali**
Associate Professor, Department
Head (Cyber Security), FAST School of Computing
National University of Computing and Emerging
Sciences, Lahore
- **Dr. Abdul Sattar**
Assistant Professor, Lahore Garrison
University, D.H.A Lahore.
- **Mr. Muhammad Fahim**
Associate Professor (Computer Science)
Govt. Graduate College for boys, Gulberg,
Lahore
- **Muhammad Asif Majeed Khan**
SST(IT), Govt Model High School, Jampur
- **Prof. Mahmood Ahmad Chaudhry**
The Crescent College, Shadman, Lahore
- **Dr. Mudasser Naseer**
Associate Professor(CS),
Department of CS & IT,
University of Lahore Defense Road, Lahore
- **Mr. Saqib Ubaid**
Assistant Professor (Computer Science).
Khawaja Fareed University of IT, Rahim Yar Khan
- **Mrs. Tabinda Muqaddas**
Assistant Professor, Head of Department (CS),
Govt. Associate College for Women,
Gulshan Ravi, Lahore.
- **Mr. Fahad Asif**
EST (CS),
Govt. Lab Higher Secondary School,
QAED Kasur.
- **Mr. Jahanzaib Khan**
Assistant Director (Curriculum-Sciences), PECTAA

Director (Curriculum and Compliance)

Mr. Aamir Riaz

Deputy. Director (Compliance-Sciences)

Syed Saghir-Ul-Hassnain Tirmizi

Assistant Director (Curriculum-Sciences)

Mr. Jahanzaib Khan

Incharge Art Cell

Ms. Aisha Sadiq

Design & Layout

Ms. Minal Tariq
Ms. Sameira Ismail

Illustrator

Mr. Ayat Ullah

Composer

M. Azhar Shah

UNIT 1

Introduction to Software Development

Student Learning Outcomes

By the end of this chapter, students will be able to:

- Define software development and explain its importance.
- Understand and describe key software development terminology, including Software Development Life Cycle (SDLC), debugging, testing, and design patterns.
- Explain the stages of the SDLC and the objectives and activities involved in each stage.
- Differentiate between various software development methodologies such as the Waterfall model and Agile methodology.
- Plan a software project by setting timelines, estimating costs, and managing risks.
- Recognize and apply quality assurance techniques to ensure software standards.
- Utilize Unified Modeling Language (UML) diagrams to represent software systems.
- Identify and apply common software design patterns in software design.
- Employ debugging techniques and testing strategies to ensure software reliability.
- Understand and utilize various software development tools, including Integrated Development Environment (IDEs), compilers, and source code repositories.

Introduction

Software development is a systematic process that transforms user needs into software products. It involves a series of stages, from initial analysis through design, coding, testing, and deployment. Each stage has its own importance and requires specific skills and tools. Understanding the software development process is crucial for creating reliable, maintainable, and scalable software solutions. The chapter introduces the fundamental concepts of software development, including key terminology, the Software Development Life Cycle (SDLC), software development methodologies, project planning and management, quality assurance, and software design patterns.

1.1 Software Development

Software development is the process of creating computer programs designed to perform specific tasks. It involves writing code, testing it, and addressing any issues that arise.

1.2 Introduction to Software Development Life Cycle (SDLC)

Software Development Life Cycle (SDLC) is a framework that defines the processes used by organizations to build an application from its initial conception to its deployment and maintenance. The primary purpose of SDLC is to deliver high-quality software that meets customer expectations, reaches completion within time and cost estimates, and works efficiently.

1.2.1 Framework in Software Development

In software engineering, a framework is a standardized and reusable set of concepts, practices, and tools that provides a structured foundation for developing software applications. It offers predefined components and architectures that facilitate the implementation of specific software functionalities, allowing developers to focus on writing code specific to their application rather than reinventing common solutions. Frameworks promote efficiency, consistency, and code reusability, that improve the overall quality and maintainability of software systems.

Example: Imagine you want to create a website. Instead of writing all the code from scratch, you can use a framework like Django (for websites). Django comes with ready-made features like user login, database management, and page templates.

1.2.2 Stages involved in SDLC

The SDLC is an organized method for developing software that ensures it meets quality standards and functions properly. The SDLC consists of several steps as shown in Figure 1.1. Each step has distinct tasks and goals.

1.2.2.1 Requirement Gathering

In this initial phase, the goal is to understand and collect what the software needs to achieve. This involves talking to the people who will use the software, as well as other stakeholders, to find out their needs and expectations.



Figure 1.1: System development life cycle stages



Key activities in this phase include:

- **Interviews and Surveys:** Asking questions and collecting feedback from potential users to understand their needs and preferences.
- **Observations:** Watching how users interact with current systems to identify problems and opportunities for improvement.
- **Document Review:** Looking at existing documents, such as reports and user manuals, to gather additional information about the requirements.

Functional and Non-Functional Requirements

Requirements are generally categorized into two types, functional and non-functional requirements.

Functional Requirements

Functional requirements describe the specific behaviors or functions of a system. These requirements outline what the system should do and include tasks, services, and functionalities that the system must perform.

They define the interactions between the system and its users or other systems.

Example:

Some functional requirements for a Library Management System are:

- **User Registration:** The system should allow users (students and faculty) to register and create an account.
- **Book Borrowing:** The system should enable users to search for books and borrow them.
- **Inventory Management:** Librarians should be able to add, update, and remove books from the inventory.

Non-Functional Requirements

Non-functional requirements define the quality attributes, performance criteria, and constraints of the system. These requirements specify how the system performs a function rather than what the system should do.

Example:

Some non-functional requirements for a Library Management System are:

- **Performance:** The system should handle up to 1000 simultaneous users without performance degradation.
- **Reliability:** The system should be available 99.9% of the time, ensuring high availability and minimal downtime.
- **Security:** User data should be encrypted, and access should be controlled through secure authentication mechanisms.

Differentiating Functional and Non Functional Requirements:

Functional Requirements	Non-Functional Requirements
Define specific behaviors or functions of the system	Define the quality attributes and constraints of the system
What the system should do	How the system should perform
Directly related to user interactions and system tasks	Related to system performance, usability, reliability, etc.

Table 1.1: Comparison between Functional and Non-Functional Requirements

1.2.2.2 Design

In the design phase, we plan out how the software will look and work. During this phase, we:

- **Create Diagrams:** To show how different parts of the software will connect and work together. For example, we draw a flowchart to map out the steps the program will take to complete a task.
- **Develop Models:** To represent the software's structure. This could include creating mockups of the user interface, showing what the program will look like, and how users will interact with it.
- **Plan the Architecture:** To decide the overall structure of the software, including how different components will interact. This helps ensure that the program is organized and functions smoothly.
- **Specify Requirements:** To define clearly what each part of the software needs to do, ensuring that all features are planned out and nothing is overlooked.

These steps help to ensure that the final software is well-organized, user-friendly, and meets the needs of its users.

Tidbits

Think of this phase like designing a new house. You need blueprints to show where the rooms and furniture will go before you start building.

1.2.2.3 Coding / Development

Based on the design specifications, which outline what the software should do and how it should look, programmers translate these specifications into a programming language.

1.2.2.4 Testing

Testing is the process of checking software to identify any bugs, errors, or issues. Think of it as a quality check to make sure everything works as expected. This includes:

- 
- **Functionality Testing:** Ensuring all features of the software work according to the specifications.
 - **Performance Testing:** Checking if the software performs well under different conditions, such as high traffic or heavy data.
 - **Compatibility Testing:** Making sure the software works well on various devices and operating systems.

1.2.2.5 Deployment

Deployment is the process of making software available for users to access and use. This often involves several steps:

- **Installation:** The software is installed on the user's system or server. This may involve running an installation program that copies files and sets up necessary configurations.
- **Configuration:** The software is adjusted to fit the specific needs of the user or organization. It can include setting up user preferences, network settings, and database connections.
- **Testing in the Real-World:** After installation, the software is tested in its real-world environment to ensure it works correctly with other systems and meets user needs.

1.2.2.6 Maintenance

The final phase involves ongoing maintenance and updates. This ensures the software continues to function correctly and adapts to any changes in user needs or technology.

1.3 Software Development Methodologies

Software development methodologies are structured approaches to software development that guide the planning, creation, and management of software projects. They help ensure that the development process is systematic, efficient, and produces high-quality software.

1.3.1 Introduction to Software Process Models

Software process models are abstract representations of the processes involved in the SDLC. They provide a framework for planning, structuring, and controlling the development of software systems. The importance of software process models lies in their ability to provide:

- **Predictability:** By following a defined process, teams can predict outcomes and manage risks more effectively.
- **Efficiency:** Structured methodologies streamline the development process, reducing wasted effort.
- **Quality:** Adhering to a process model ensures that quality assurance practices are integrated throughout the SDLC.

1.3.1.1 Waterfall Model

The Waterfall Model is a straightforward approach to software development where each phase of the project must be completed before the next one begins. This model is linear and sequential, meaning that you move through each phase in order, without going back to previous phases once they are completed as shown in Figure 1.2. The main phases of the Waterfall Model are:

- **Requirements:** Gather and document what the software needs to do.
- **Design:** Plan how the software will be built and how it will look.
- **Implementation:** Write the actual code to create the software.
- **Testing:** Check for and fix any problems or bugs in the software.
- **Deployment:** Release the software for users to use.
- **Maintenance:** Make updates and fix any issues that come up after the software is in use.

Benefits and Limitations

- **Benefits:**
 1. **Simple and Easy to Understand:** The Waterfall Model is easy to follow because it has clear, distinct phases
 2. **Sequential Process:** Each phase is completed one at a time, which makes it easier to manage and track progress.
 3. **Suitable for Small Projects:** Works well for projects with clear, fixed requirements where changes are unlikely.
- **Limitations:**
 1. **Inflexibility:** Once a phase is completed, going back to make changes is difficult and costly.
 2. **Not Ideal for Complex Projects:** For projects with evolving requirements or complex designs, this model can be challenging to use effectively.
 3. **Risk and Uncertainty:** The model assumes that all requirements are known from the start, which can be risky if new needs or issues arise later in the process.

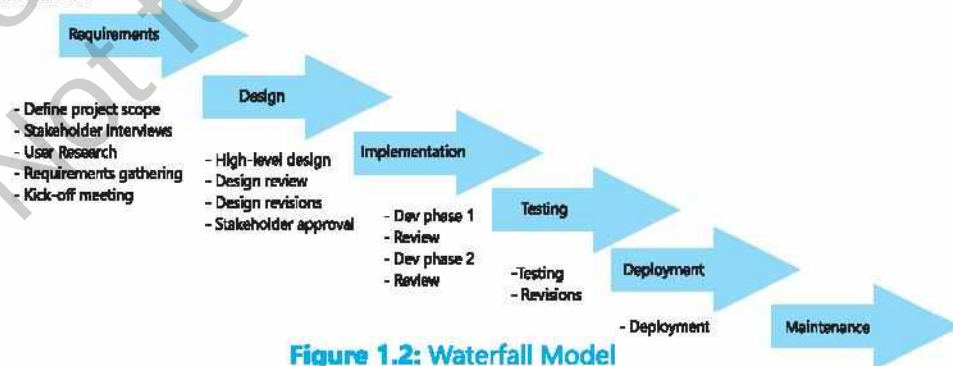


Figure 1.2: Waterfall Model

1.3.1.2 Agile Methodology

Agile Methodology is a flexible and adaptive approach to software development. Agile focuses on delivering small, functional parts of the software quickly and adapting to changes as the project progresses. The main idea is to work in short cycles, called iterations or sprints, which help teams deliver parts of the software rapidly and gather feedback early as shown in Figure 1.3. Agile methods include practices such as:

- **Continuous Integration:** Regularly merging code changes into a central repository to detect and fix issues early.
- **Test-Driven Development:** Writing tests before writing the code to ensure the software works as expected.
- **Pair Programming:** Two developers work together at one workstation, with one writing code and the other reviewing it in real-time.

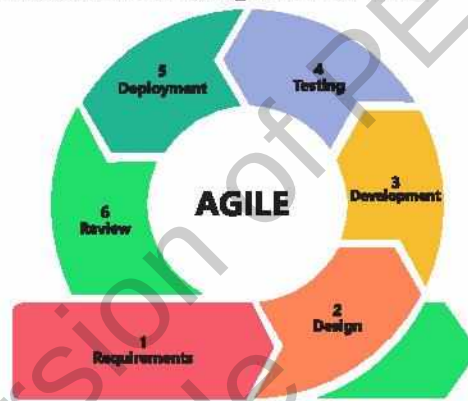


Figure 1.3: Agile Methodology

Benefits and Limitations

- **Benefits:**
 1. **High Flexibility:** Agile allows for changes in requirements even after development has started, making it easier to adapt to new needs or feedback.
 2. **Improved Customer Satisfaction:** Regular updates and frequent delivery of working software mean that customers can see progress and provide feedback more often.
- **Limitations:**
 1. **Scaling Challenges:** Managing large projects with many teams can be difficult, as it requires careful coordination and communication.
 2. **Stakeholder Involvement:** Agile requires active participation from all stakeholders, which can be challenging if some are unavailable or not fully engaged.
 3. **Less Predictable:** Since Agile projects evolve through feedback and changes, it can be harder to predict the exact timeline and scope of the final product.

1.4 Project Planning and Management

Planning a software project is like planning a trip. You need to know where you're going, how long it will take, and how much it will cost.

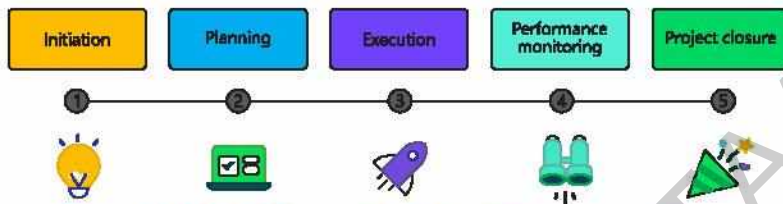


Figure 1.4: The 5 Phases of a Project Management Plan

1.4.1 Comprehensive Project Planning

Comprehensive project planning involves thinking about all the details of your project before you start. This includes understanding what needs to be done, who will do it, and how it will be done.



Big software companies are worth a lot of money, for example in 2023, Microsoft's worth was \$ 2 Trillion. This shows how important software is in today's digital world.

1.4.2 Setting Project Timelines

Setting project timelines means deciding how long each part of the project will take. This helps keep the project on track and ensures it gets done on time.

1.4.3 Estimating Costs

Estimating the cost of a software project is a critical step in project planning and management. It involves predicting the total expenses required to complete the project successfully. Accurate cost estimation helps in budgeting, resource allocation, and setting realistic expectations.

Key Factors in Cost Estimation:

- **Development Team:** The cost depends on the number of developers, their expertise, and their hourly rates.
- **Technology Stack:** The choice of technology, programming languages, and tools can affect the cost. Some technologies require more resources or specialized knowledge.
- **Project Duration:** Longer projects generally incur higher costs due to prolonged resource engagement and potential changes in scope.
- **Risk Management:** Identifying potential risks and their mitigation strategies can add to the overall cost. Contingency funds are often included to address unforeseen issues.
- **Quality Assurance:** Costs associated with testing, bug fixing, and ensuring the

- 
- software meets quality standards, are also part of the estimation.

1.4.4 Risk Assessment and Management

Risk assessment and management are crucial aspects of any software project. They involve identifying potential risks that could impact the project's success, analysing the likelihood and impact of these risks, and developing strategies to manage them.

Steps in Risk Assessment and Management:

1. **Identify Risks:** List all potential risks that could affect the project. These could be technical risks, such as technology changes; operational risks, like resource shortages; or external risks, such as market fluctuations.
2. **Analyze Risks:** Evaluate the likelihood of each risk occurring and its potential impact on the project.
3. **Develop Mitigation Strategies:** For each significant risk, develop a plan to reduce its likelihood or minimize its impact. This could involve adding buffers to the schedule, securing backup resources, or conducting additional testing.
4. **Monitor and Review:** Continuously monitor the project for new risks and review existing risks to adjust strategies as necessary.

1.4.5 Execution

This is the phase where actual development work happens. The team writes codes, creates designs, and builds the software based on the project plan, it requires team work, coordination and regular updates to stay on track.

1.4.6 Quality Assurance

Quality assurance ensures that a project meets set standards and works correctly. It involves methods such as testing, reviewing code, getting feedback from stakeholders, and regularly checking the project's progress.

1.5 Graphical Representation of Software Systems

Graphical representation of software systems involves using visual diagrams to depict various aspects of a software system's structure and behavior. This approach helps in simplifying complex systems, making it easier for developers and stakeholders to understand, communicate, and manage the system.

1.5.1 Introduction to UML

Unified Modeling Language (UML) is a standardized way to visualize the design of a software system. It helps developers understand how a system works and communicates.

1.5.2 Types of UML Diagrams

In this section, we will discuss four types of UML diagrams that are given below.

1.5.2.1 Use Case Diagrams

Use case diagrams provide a visual representation of the system's functionality from the

user's perspective, helping to identify the requirements and the interactions between the users and the system.

Definition and Purpose:

A use case is a description of a set of interactions between a user (actor) and a system to achieve a specific goal. Use cases are identified based on the functionalities that the system must support to meet the user's needs. Each use case represents a complete workflow from the user's perspective, detailing the steps involved in accomplishing a particular task.

Use Case Diagrams are used for several purposes:

1. **Capturing Functional Requirements:** They help in identifying and documenting the functional requirements of the system.
2. **Understanding User Interactions:** They illustrate how different users will interact with the system.
3. **Planning and Testing:** They aid in planning the development process and in designing test cases for validating system functionalities.

Identifying Use Cases:

The process of identifying use cases involves several steps:

1. **Identify Actors:** Determine the different types of users who will interact with the system. Actors can be human users or other systems.
2. **Define Goals:** For each actor, identify their goals or what they need to accomplish using the system.
3. **Outline Interactions:** Describe the interactions between the actors and the system to achieve these goals. Each interaction that results in a significant outcome is a potential use case.
4. **Validate Use Cases:** Review the identified use cases with stakeholders to ensure they accurately capture the required functionalities and interactions.

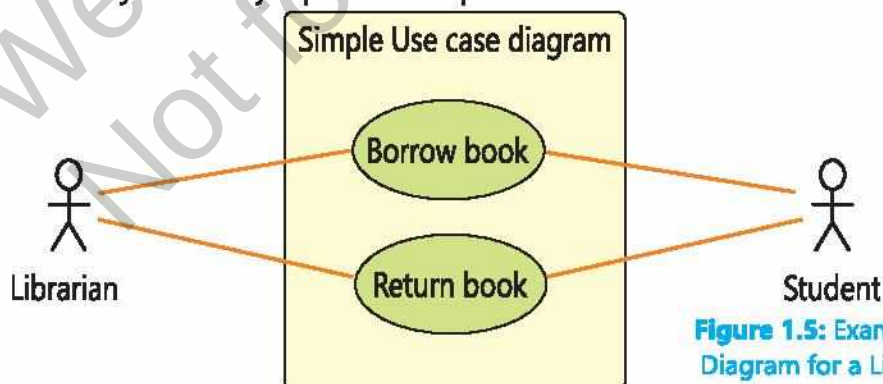


Figure 1.5: Example Use Case Diagram for a Library System

Class Activity

Statement: Imagine you are designing an online shopping platform. The platform allows customers to browse products, add items to their cart, and make purchases. Additionally, the platform includes features for administrators to manage product listings, process orders, and handle customer inquiries. There is also a feature for delivery personnel to update the status of deliveries.

In the above class activity, you can compare your findings with the following:

- **Actors:**
 - Customer
 - Administrator
 - Delivery Personnel
- **Use Cases:**
 - Browse Products
 - Add Items to Cart
 - Make Purchase
 - Manage Product Listings
 - Process Orders
 - Handle Customer Inquiries
 - Update Delivery Status

1.5.2.2 Class Diagram

A class diagram is like a map that shows how things are organized in a system.

Example:

In the example of organizing your room as shown in Figure 1.6:

- **Room:** Represents the overall space encompassing all other elements, analogous to the main structure in a class diagram.
- **Box:** Serves as a container within the room, akin to a class in a diagram.
- **Attributes:** Each box contains specific items, such as a 'ToyBox' holding toys or a 'BookBox' containing books.
- **Methods:** Boxes can perform actions like 'open' or 'close,' similar to methods in a class diagram that define what the box can do.
- **Specific Boxes:** Examples of specialized boxes include a 'ToyBox' for toys, a 'BookBox' for books, and a 'ClothesBox' for clothes, representing distinct instances of the general 'Box' class.

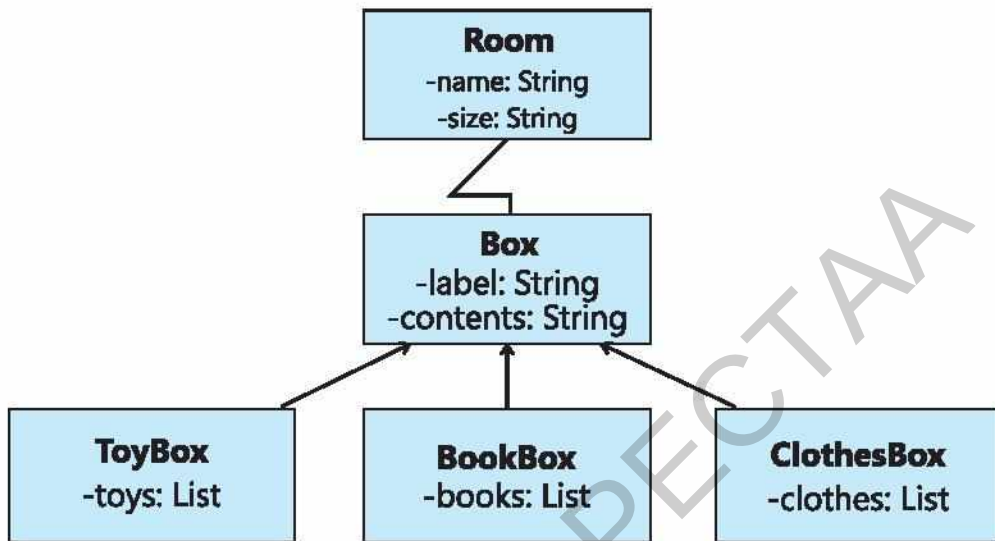


Figure 1.6: Class Diagram for Organizing Your Room

1.5.2.3 Sequence Diagrams

Sequence Diagrams show how objects in a system interact with each other in a particular sequence. They help in understanding the flow of messages between objects over time.

Interactions:

- **open()**: User opens each box.
- **put toys/books/clothes inside**: User puts the respective items into the boxes.
- **close()**: User closes each box.

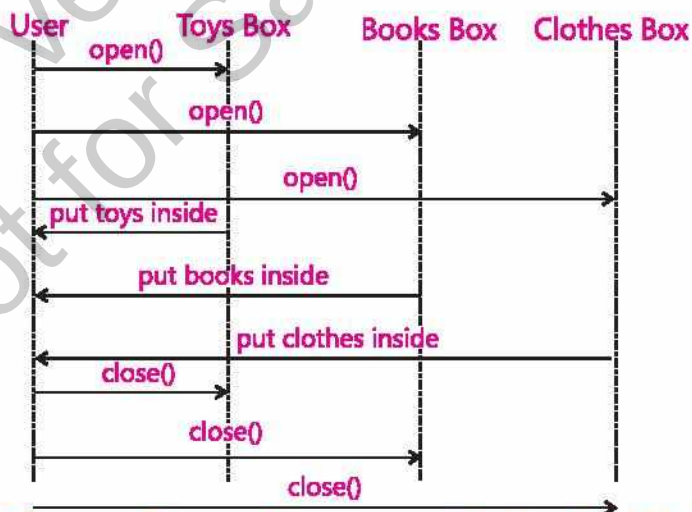


Figure 1.7: Sequence diagram of the user organizing items into labeled boxes

1.5.2.4 Activity Diagrams

Activity Diagrams illustrate the flow of activities or steps in a process. They are useful for modeling the logic of complex operations.

Example: In a restaurant management system, an activity diagram can represent the process from 'Order Placement' to 'Food Preparation' and finally to 'Order Delivery'.

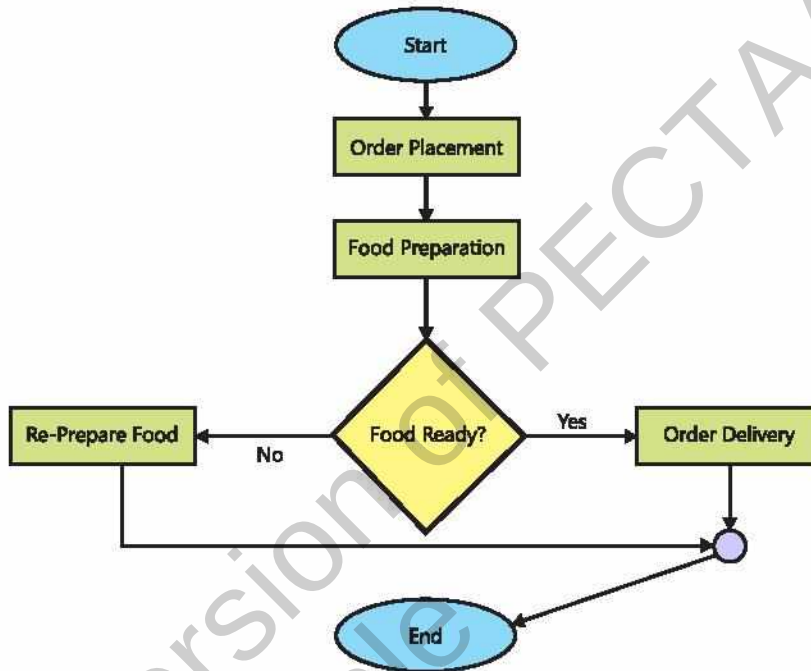


Figure 1.8: Activity Diagram with Decision and Connector Symbol

1.5.3 Using UML to Represent Software Systems

UML can be used in various stages of software development to improve understanding and communication. Here are some practical applications:

- **Planning:** Use UML diagrams to map out the system's requirements and design before writing any code.
- **Development:** Developers refer to UML diagrams to understand the structure and relationships within the system.
- **Communication:** UML diagrams help team members, including non-technical stakeholders, to understand how the system works.

1.6 Introduction to Design Patterns

Design pattern are common solutions to problem in software development, they act like templates to help make coding easier, faster and more consistent.

1.6.1 Commonly Used Design Patterns

Below are some of the most widely recognized design patterns:

1.6.1.1 Singleton Pattern

The Singleton Design Pattern is a way to make sure that a specific object or resource is created only once in a program and reused whenever needed.

1.6.1.2 Factory Pattern

The Factory Design Pattern is like having a special workshop that knows how to create different products, but you don't need to worry about the details of how those products are made. Instead, you just tell the factory what you need, and it gives you the finished product.

1.6.1.3 Observer Pattern

The Observer Design Pattern is like having a group of people who are interested in getting updates from one particular source. Whenever something important happens, the source automatically notifies all the interested people. It's a way to keep things in sync without everyone constantly checking for updates.

1.6.1.4 Strategy Pattern

The Strategy Design Pattern is like having a toolbox full of different tools, each designed for a specific job. When you face a problem, you can pick the right tool from the box based on the task at hand.

Class Activity

Identify a real-world scenario around you where you can apply one of these design patterns. Share your examples in the next class.

1.6.2 Applications of Design Patterns in Software Design

Design patterns are widely used in software development to solve common problems and create robust and maintainable code. They help in:

- Reducing code complexity by providing a clear structure.
- Enhancing code reusability by using proven solutions.
- Improving communication among developers by providing a common vocabulary.

Design patterns help create systems that are flexible, maintainable, and easy to understand.



Many popular software frameworks and libraries are built using design patterns. For example, the Model-View-Controller (MVC) pattern is used in web development frameworks like Ruby on Rails and Angular.

1.7 Software Debugging and Testing

Debugging and testing are important steps to make sure that software works correctly. They help find and fix errors so the software meets requirements and run as expected.

1.7.1 Debugging

Debugging is the process of finding and fixing bugs or errors in a software. Bugs are errors or mistakes in the software that cause it to behave unexpectedly. Identifying bugs involves observing the software's behavior and finding the source of the problem. Once identified, bugs requires making changes to the code to correct the error.

Tools and Best Practices

There are various tools and best practices for debugging, including:

- **Debuggers:** Software tools that help programmers find bugs by allowing them to step through code, inspect variables, and monitor program execution.
- **Print Statements:** Adding print statements in the code to display the values of variables at different points in the program.
- **Code Reviews:** Having other developers review your code to spot potential errors.

1.7.2 Testing

Testing is the process of evaluating the software to ensure it meets the requirements and works as expected. The testing process typically follows a hierarchy that begins with smaller components and gradually progresses to the entire system, including user acceptance. The main types of testing in this hierarchy are given below.

1.7.2.1 Unit Testing

Unit Testing is the first level of testing, where individual components or modules of the software are tested in isolation. Each "unit" is a small, testable part of the software, such as a function or method. The primary goal of unit testing is to verify that each component works correctly according to its design and performs as expected.

Class Activity

Try writing a unit test for a simple function in your favorite programming language.

1.7.2.2 Integration Testing

After unit testing, Integration Testing is performed to evaluate the interaction between different components or modules. While unit testing focuses on isolated units, integration testing ensures that these units work together correctly when combined.

This type of testing checks for interface errors, data flow between modules, and other integration-related issues.

1.7.2.3 System Testing

System Testing is a higher level of testing where the entire software system is tested as a whole. At this stage, the software is treated as a complete entity, and testers evaluate its overall functionality, performance, security, and compliance with specified requirements.

1.7.2.4 Acceptance Testing

Acceptance Testing is conducted to determine whether the software is ready for release. It is often performed by the end-users or clients to ensure that the software meets their expectations and requirements.



Acceptance testing is sometimes called User Acceptance Testing (UAT) because it is often done by the end-users of the software.

1.8 Software Development Tools

Software development tools are programs or applications that assist in various stages of software creation. They are used to write, edit, test, debug, and manage code, ensuring that software functions correctly and efficiently.

1.8.1 Language Editors

Language editors, also known as code editors, are tools that help developers write and edit code in different programming languages. Examples include:

- **Notepad++:** A simple yet powerful code editor.
- **VS Code:** A popular editor with many extensions.

1.8.2 Translators

Translators are tools that convert code written in one programming language into another language that the computer can understand. Translators convert high-level programming languages (like Python) into machine language (binary code) that computers can execute. It has two types:

- **Interpreters:** Translate code line-by-line (e.g., Python interpreter).
- **Compilers:** Translate the entire code at once (e.g., GCC for C/C++).

1.8.3 Debuggers

Debuggers are tools that help developers find and fix errors (bugs) in their code. The purpose of debuggers is to allow developers to test their code and identify where errors occur. Examples include:

GDB: GNU Debugger for C/C++.

Visual Studio Debugger: Integrated with Visual Studio IDE.



1.8.4 Integrated Development Environments (IDEs)

IDEs are comprehensive software suites that provide all the tools needed for software development in one place. IDE integrates various development tools like editors, compilers, debuggers, and version control systems to streamline the development process. An IDE offers a unified interface where developers can write, test, and debug their code efficiently. Examples include:

- **Visual Studio:** Popular for .NET and C++ development.
- **PyCharm:** Preferred for Python development.

1.8.5 Online and Offline Computing Platforms

These platforms provide environments where developers can write, run, and test their code.

- **Online Platforms:** Cloud-based platforms accessible via the internet (e.g., Repl.it, Gitpod).
- **Offline Platforms:** Local development environments on a computer (e.g., local installations of IDEs).

1.8.6 Source Code Repositories

Source code repositories are platforms where developers can store, manage, and track changes to their code. Repositories help in version control, allowing multiple developers to work on the same project without conflicts. Examples include:

- **GitHub:** Popular platform for open-source projects.
- **Bitbucket:** Used for both private and public repositories.

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

Short Questions

1. Differentiate between functional and non-functional requirements.
2. Explain why the testing phase is important in the Software Development Life Cycle (SDLC), and provide two reasons for its significance.
3. Illustrate the concept of continuous integration in Agile Methodology and

- 
- discuss its importance in software development.
4. Evaluate the main steps involved in risk assessment and management, and assess their importance in a software project.
 5. Explain the purpose of a Use Case Diagram in software development.
 6. Compare and contrast a Sequence Diagram with an Activity Diagram, highlighting the key differences.
 7. Describe the Factory Pattern and explain how it differs from directly creating objects, with an example.

Long Questions

1. Design a flowchart for a user registration process in a software application. Outline its key steps.
 2. Imagine you are managing a project to develop a simple mobile application. Describe how you would use the Agile Methodology to handle this project.
 3. Consider an online banking system. Create a Use Case Diagram to show the interactions between customers, bank staff, and the system.
 4. 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.
 5. Discuss the importance of software development tools in the software development process.
 - a) Explain the role of language editors, translators, and debuggers in creating and maintaining software.
 - b) Provide examples of each tool and describe how they contribute to the efficiency and accuracy of software development.
- 