Department of Computer Science

Details of Lesson Plan

S.No.	Particulars	Details
1.	Course Name	Software Engineering
2.	Course Code	COM-603
3.	Academic Year	2024-2025
4.	Semester	6th
5.	Number of Lesson plans	39
6.	Faculty Assigned	Dr. Palvi Sharma

[illegible]

Faculty Signature

Model Institute of Engineering & Technology (Autonomous)

(Permanently Affiliated to the University of Jammu, Accredited by NAAC with "A" Grade)

Lesson Plan No. 1.1	**Course Name: Software Engineering**	**Course No.: COM-603**

Objectives	At the end of the lesson the student shall be able to: a. Understand the need and importance to study Software Engineering. b. Understand the applications of Software Engineering.
Teaching Aids (if any)	a. PPTs. b. Green board (Chalk and Talk). c. Video Lecture
Teaching Development /content summary	1. **Introduction** (5 minutes) - Brief introduction about Object oriented analysis and design. - Introduction to SE. - Applications of SE. - Role of SE in industry. - Attributes of SE. 2. **Development** (30 minutes) a.Introducing the concept of SE. b. Overview of Software. c. Discussion on major characteristics of SE. d. Main Attributes of Software Engineering. 3. **Exercise** (10 minutes) – **Video lectures**: https://www.youtube.com/watch?v=wmLFS41UeO0 Suggested reading the article on " Introduction to SE"
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. Spend 5 minutes to wrap up and consolidate the leanings.

Model Institute of Engineering & Technology (Autonomous)

(Permanently Affiliated to the University of Jammu, Accredited by NAAC with "A" Grade)

Lesson Plan No. 1.2	**Course Name: Software Engineering**	**Course No.: COM-603**

Objectives	At the end of the lesson the student shall be able to: a. Understand the need and importance of Legacy Software. b. Understand the Software crisis (myths and causes).
Teaching Aids (if any)	a. PPTs. b. Green board (Chalk and Talk). c. Video Lecture
Teaching Development /content summary	1. **Introduction** (5 minutes) - Recapitulation of previous lecture. - Introduction to Legacy software. - Software myths. 2. **Development** (30 minutes) a. Introducing the concept of - Adaptive, - Enhanced, - Interoperable and re-architected Software. b. Study software Crises and causes. - Factors involved in software crises. 3. **Exercise** (10 minutes) – c. **Activity** : (Think-Pair-Share, List the most important factors responsible for the software crisis?) d. **Video lectures**: https://youtu.be/KucbhwtZcXs - https://youtu.be/OFBK-I0rrNk e. Suggested reading the Pressman (P. No. 37-42)
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. Spend 5 minutes to wrap up and consolidate the leanings.
Evaluation	Outcome Based on Activity conducted.

Model Institute of Engineering & Technology (Autonomous)

(Permanently Affiliated to the University of Jammu, Accredited by NAAC with "A" Grade)

Lesson Plan No. 1.3	**Course Name: Software Engineering**	**Course No.: COM-603**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of different process flows in Software engineering.
Teaching Aids (if any)	a. PPTs. b. Green board (Chalk and Talk). c. Video Lecture
Teaching Development /content summary	1. **Introduction** (5 minutes) - Introducing the concept of a Software Process. - What/who/why is there the concept of process models? 2. **Development** (30 minutes) a. Introducing the concept of Iterative, - parallel and evolutionary process in Software. c. Identifying a task set. - Process Patterns etc. 3. **Exercise** (10 minutes) – b. **Activity** : (GD Discuss the difference between Iterative and parallel process) c. **Video Lectures**: https://youtu.be/GyuE47Hv60k https://youtu.be/bAEnaGG8Otc d. Suggested reading the Pressman Book (P.No. 43-50)
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. Spend 5 minutes to wrap up and consolidate the leanings.
Evaluation	Outcome based on group discussion.

Model Institute of Engineering & Technology (Autonomous)

(Permanently Affiliated to the University of Jammu, Accredited by NAAC with “A” Grade)

Lesson Plan No. 1.4	**Course Name: Software Engineering**	**Course No.: COM-603**

Objectives	At the end of the lesson the student shall be able to: a. Understand the need and importance UML Modelling.
Teaching Aids (if any)	a. PPTs. b. Green board (Chalk and Talk). c. Video Lecture
Teaching Development /content summary	1. **Introduction** (5 minutes) - Recapitulation of previous lecture. - Introduction to UML. - Need of systematic software models. 2. **Development** (30 minutes) a. Introducing the concept of the Unified modelling language. - Detailed discussion on different types of the use case diagrams.(Structure,Behaviour etc) b. Examples(Sequence diagram, context diagram,collaboration diagram,activity diagram,transition diagram,etc) c. Study Pros of each diagram. 3. **Exercise** (10 minutes) – d. **Activity** : (MCQ Questions) e. **Video lectures**: https://www.youtube.com/watch?v=b8VMFa3Cdbo e. Suggested reading the Grady Booch Book (P. No. 29-37)
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. Spend 5 minutes to wrap up and consolidate the leanings.
Evaluation	Q & A on lesson delivered.

Model Institute of Engineering & Technology (Autonomous)

(Permanently Affiliated to the University of Jammu, Accredited by NAAC with "A" Grade)

Lesson Plan No. 1.5	**Course Name: OOAD**	**Course No.: COM-702**

Objectives	At the end of the lesson the student shall be able to: a. Understand the need and importance of Software development life cycle (SDLC) and OOAD b. Understand the integration of OOAD and SE.
Teaching Aids (if any)	a. PPTs. b. Green board (Chalk and Talk). c. Video Lecture
Teaching Development /content summary	1. **Introduction** (5 minutes) - Recapitulation of previous lecture. - Introduction to SDLC. - Steps involved in the working of the model. - Integration of SE and OOAD. 2. **Development** (30 minutes) a. Introducing the working of SDLC. b. Object oriented software process development model. 3. **Exercise** (10 minutes) – c. **Activity** : Make SDLC of any application software. d. **Video lectures**: https://youtu.be/wYeugavTPS0 - https://youtu.be/_VtfqbTHLPg e. Suggested reading the article on " Importance of Software Engineering (P.No. 67-71) f. Grady Booch Book P.no 4-24
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. Spend 5 minutes to wrap up and consolidate the leanings.
Evaluation	Q & A on lesson delivered.

Lesson Plan No. 1.6	**Course Name: Software Engineering**	**Course No.: COM-603**

Objectives	At the end of the lesson the student shall be able to: a. Understand the UML Diagram depiction. b. Understand the Creation of different UML models.
Teaching Aids (if any)	a. PPTs. b. Green board (Chalk and Talk). c. Video Lecture
Teaching Development /content summary	1. **Introduction** (5 minutes) - Recapitulation of previous lecture. - Introduction to the basics Structure diagrams. - Steps involved in the working of the model. - Applications of a prototype. 2. **Development** (30 minutes) a. Introducing the working concept of structure models.(Automated Trading System). b. Variation between Structure diagrams and behaviour diagrams. 3. **Exercise** (10 minutes) – c. **Activity** :Quiz d. **Video lectures**: https://www.youtube.com/watch?v=kVV1TJ_VR0o e. Suggested reading the article on Structure Diagrams.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. Spend 5 minutes to wrap up and consolidate the leanings.
Evaluation	Based on Quiz conducted

Lesson Plan No. 1.7	**Course Name: Software Engineering**	**Course No.: COM-603**

Objectives	At the end of the lesson the student shall be able to: a. Understand the need to study agile technology. b. Understand the top methodologies of agile models.
Teaching Aids (if any)	a. PPTs. b. Green board (Chalk and Talk). c. Video Lecture
Teaching Development /content summary	1. **Introduction** (5 minutes) - Recapitulation of previous lecture. - Introduction to the basics of agile technology. - Understanding the basics of agile model. 2. **Development** (30 minutes) a. Introducing the working concept of agile models.(Scrum) b. Variation between various process models and agile models. c. Examples (Scrum, XP, feature-driven development, lean software development, adaptive software development) 3. **Exercise** (10 minutes) – c. **Activity** :5 minutes paper activity (Difference between Waterfall and Agile System) d. **Video lectures**: https://youtu.be/1xpFmmzlBQU e. Suggested reading the article on " Importance of Software Engineering (P.No. 94-100)
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. Spend 5 minutes to wrap up and consolidate the leanings.
Evaluation	Outcome based on Activity.

Model Institute of Engineering & Technology (Autonomous)

(Permanently Affiliated to the University of Jammu, Accredited by NAAC with "A" Grade)

Lesson Plan No. 2.3	**Course Name: Software Engineering**	**Course No.: COM-603**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of system modelling,behavioural models,data processing models . b. Understand the need and importance of DFD.
Teaching Aids (if any)	a. PPTs. b. Green board (Chalk and Talk). c. Video Lecture
Teaching Development /content summary	1. **Introduction** (5 minutes) - Introduction to the concept of system modelling,behavioural models,data processing models. - Introduction to the concept of DFD. - Rules for creating DFD. - Symbols used in DFD. - Levels of DFD. 2. **Development** (30 minutes) a. Diagrammatic representation of components of DFD. b. Examples (Order Processing DFD, Insulin Pump DFD). 3. **Exercise** (10 minutes) – c. **Activity** :Quiz d. Suggested reading the article on " Importance of Software Engineering https://www.mlsu.ac.in/econtents/16_EBOOK-7th_ed_software_engineering_a_practitioners_approach_by_roger_s._pressman_.pdf
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. Spend 5 minutes to wrap up and consolidate the leanings.
Evaluation	Q & A on lesson delivered.

Model Institute of Engineering & Technology (Autonomous)

(Permanently Affiliated to the University of Jammu, Accredited by NAAC with "A" Grade)

Lesson Plan No. 2.4	**Course Name: Software Engineering**	**Course No.: COM-603**

Objectives	At the end of the lesson the student shall be able to: a. Understand the Concept of SRS. b. Understand the need and importance of SRS.
Teaching Aids (if any)	a. PPTs. b. Green board (Chalk and Talk). c. Video Lecture
Teaching Development /content summary	1. **Introduction** (5 minutes) - Introduction to the concept of structure of SRS, general description,functional requirements,interface requirements,performance requirements. - Uses of SRS document. 2. **Development** (30 minutes) Rrepresentation of the concept of structure of SRS, general description,functional requirements,interface requirements,performance requirements. a. Uses of SRS document. b. Examples of making SRS for different projects. 3. **Exercise** (10 minutes) – c. **Activity** :Quiz d. Suggested reading the article on " Importance of Software Engineering https://www.mlsu.ac.in/econtents/16_EBOOK-7th_ed_software_engineering_a_practitioners_approach_by_roger_s._pressman_.pdf
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. Spend 5 minutes to wrap up and consolidate the leanings.
Evaluation	Q & A on lesson delivered.

Model Institute of Engineering & Technology (Autonomous)

(Permanently Affiliated to the University of Jammu, Accredited by NAAC with "A" Grade)

Lesson Plan No. 2.5	**Course Name: Software Engineering**	**Course No.: COM-603**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of system modelling,behavioural models,data processing models . b. Understand the need and importance of class and object modelling.
Teaching Aids (if any)	a. PPTs. b. Green board (Chalk and Talk). c. Video Lecture
Teaching Development /content summary	1. **Introduction** (5 minutes) - Introduction to the concept of system modelling,behavioural models,data processing models. - Rules for creating Data flow diagrams. - Levels of DFD. 2. **Development** (30 minutes) a. Diagrammatic representation of components of DFD. b. Examples (Order Processing DFD, Insulin Pump DFD). c. Diagrammatic Representation of Class Modelling placing orders 3. **Exercise** (10 minutes) – d. **Activity** :GD e. Suggested reading the article on " Importance of Software Engineering https://www.mlsu.ac.in/econtents/16_EBOOK-7th_ed_software_engineering_a_practitioners_approach_by_roger_s._pressman_.pdf
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. Spend 5 minutes to wrap up and consolidate the leanings.
Evaluation	Q & A on lesson delivered.

Lesson Plan No. 3.1	**Course Name: Software Engineering** **Topic: Software Project Management Process: Software Scope & Resources**	**Course No.: COM-603**

Objectives	At the end of the lesson the student shall be able to: a. Understand the importance of software project management. b. Define software scope and identify key project resources. c. Learn about different types of software resources. d. Recognize challenges in managing software scope and resources.
Teaching Aids (if any)	a. Presentation slides b. Whiteboard c. Case study
Teaching Development	1. **Introduction** (10 minutes) - Ask students: What do you think software project management involves? - Discuss the significance of project scope and resource planning. - Explain the learning objectives. 2. **Development** (30 minutes) **a. Software Scope** – Definition & importance – How scope is defined in projects – Examples of scope changes and their impact **b. Semi-structured Data:** – Types of resources: Human, technical, financial – Strategies for resource allocation – Case study discussion: Managing resources in a real project
Closure	1. Summarize the lesson correlating with learning outcomes. 2. Encourage students to read more on software project scoping.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. Why is defining software scope critical? 2. What are the key types of project resources? 3. What challenges arise when managing project resources? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 3.2	**Course Name: Software Engineering** **Topic: Software Metrics**	**Course No.: COM-603**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept and importance of software metrics. b. Learn different types of software metrics. c. Analyze how software metrics influence project success. d. Apply software metrics in project assessment.
Teaching Aids (if any)	a. PPT slides b. Whiteboard c. Example software project reports
Teaching Development	1. **Introduction** (10 minutes) – Ask: How do we measure software project success? – Define software metrics and their purpose. – Introduce the lesson objectives. 2. **Development** (30 minutes) **a. Types of Software Metrics** – Process Metrics – Product Metrics – Project Metrics **b. Key Metrics** – LOC (Lines of Code) – Function Points – Defect Density **c. Case Study** – Analyze a project’s software metrics and interpret the results.
Closure	1. Recap of key concepts. 2. Discussion on why software metrics are essential.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What are software metrics, and why are they important? 2. Name and describe three types of software metrics. 3. How do software metrics impact project management decisions? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 3.3	Course Name: Software Engineering Topic: Software Project Estimation & Decomposition Techniques	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Understand software project estimation. b. Learn different decomposition techniques. c. Explore estimation models and their practical applications. d. Apply estimation methods in software project planning.
Teaching Aids (if any)	a. Presentation slides b. Case studies c. Example estimation templates
Teaching Development	1. **Introduction** (10 minutes) – Discuss: Why is project estimation important? – Define software estimation and decomposition. 2. **Development** (30 minutes) **a. Estimation Techniques** – Expert Judgment – Analogy-based Estimation – Decomposition Techniques (Top-down, Bottom-up) **b. Hands on Activity** – Students break down a sample project into smaller components for estimation.
Closure	1. Summarize the lesson. 2. Encourage students to practice estimating small projects.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What are decomposition techniques in project estimation? 2. How does estimation impact project planning? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 3.4	Course Name: Software Engineering Topic: Empirical Estimation Model: COCOMO	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Understand the purpose of the COCOMO model. b. Learn different types of COCOMO. c. Apply COCOMO to estimate software development effort.
Teaching Aids (if any)	a. PPT slides b. Example calculations c. Online COCOMO estimation tools
Teaching Development	1. **Introduction** (10 minutes) – Define empirical estimation models. – Discuss the significance of COCOMO. 2. **Development** (30 minutes) **a. Types of COCOMO** – Basic, Intermediate, and Detailed COCOMO – Factors influencing COCOMO estimates **b. Practical Application** – Solve an estimation problem using the COCOMO model.
Closure	1. Recap of estimation principles. 2. Discuss advantages and limitations of COCOMO.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What are the three types of COCOMO models? 2. How does COCOMO help in software project estimation? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 3.5	Course Name: Software Engineering Topic: Software Project Scheduling	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Understand the importance of scheduling in software projects. b. Learn key scheduling techniques. c. Develop project schedules using industry-standard tools.
Teaching Aids (if any)	a. PPT slides b. Gantt charts, PERT diagrams
Teaching Development	1. **Introduction** (10 minutes) - Ask: What happens if a software project is not properly scheduled? - Define software project scheduling 2. **Development** (30 minutes) **a. Scheduling Techniques** – Gantt Charts – PERT and CPM Methods **b. Hands on Task** – Students create a Gantt chart for a sample project.
Closure	1. Discuss common scheduling challenges.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What is the importance of software project scheduling? 2. Name two scheduling techniques and explain their benefits. Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 3.6	Course Name: Software Engineering Topic: Risk Analysis & Software Acquisition	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Understand the fundamentals of risk analysis. b. Identify common software project risks. c. Learn about software acquisition processes.
Teaching Aids (if any)	a. PPT slides b. Risk assessment templates c. Case study
Teaching Development	1. **Introduction** (10 minutes) – Ask students: What risks do software projects face? – Define risk analysis and software acquisition. 2. **Development** (30 minutes) **a. Risk Analysis** – Types of risks (technical, financial, schedule) – Risk mitigation strategies **b. Software Acquisition** – Methods of acquiring software – Challenges in software procurement
Closure	1. Summarize risk management strategies.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What are the major risks in software projects? 2. How does software acquisition impact project management? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 4.1	Course Name: Software Engineering Topic: System Design Principles – Levels of Abstraction	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of system design principles and abstraction levels. b. Differentiate between architectural and detailed design. c. Explain the importance of abstraction in software development. d. Identify trade-offs between high-level and low-level design.
Teaching Aids (if any)	a. Presentation slides b. Whiteboard c. Case study d. Video Lecture
Teaching Development	1. **Introduction** (10 minutes) - Ask students: "Why do we need system design in software engineering?" - Explain the role of system design in software development. - Discuss the importance of abstraction in simplifying complex systems. 2. **Development** (30 minutes) **a. Levels of Abstraction** - Define Architectural Design (high-level design) and Detailed Design (low-level design). - Show a real-world example (e.g., designing a banking system at different abstraction levels). **b. Importance of Abstraction** - Types of resources: Human, technical, financial - Strategies for resource allocation - Case study discussion: Managing resources in a real project **c. Trade-offs Between High-Level and Low-Level Design** - High-Level Design: Focuses on system structure, components, and interactions. - Low-Level Design: Focuses on algorithms, data structures, and logic implementation. - Comparison of advantages and limitations of both. - Case Study: Example of abstraction in software frameworks vs. low-level coding.
Closure	1. Summarize key takeaways: • Difference between architectural and detailed design. • Why abstraction matters in software engineering. • Trade-offs in choosing high-level vs. low-level design. 2. Ask students to share real-world examples of abstraction in software development.

Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What are the key differences between architectural and detailed design? 2. Why is abstraction important in software development? 3. Can you provide an example where high-level design is preferred over low-level design? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 4.2	Course Name: Software Engineering Topic: Coupling and Cohesion	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Define coupling and cohesion in software design. b. Understand the importance of coupling and cohesion in modular software development. c. Differentiate between various types of coupling and their impact on software maintainability. d. Compare high cohesion vs. low cohesion and analyze their effect on modular design.
Teaching Aids (if any)	a. PPT slides for explanations and examples b. Chalk and Board for diagrams c. Code snippets to illustrate different types of coupling and cohesion d. Video Lecture on modular software design
Teaching Development	1. **Introduction** (10 minutes) - Start with a question: "Why do we break software into modules?" - Explain the concepts of modularity, reusability, and maintainability. - Introduce coupling and cohesion as key design principles. 2. **Development** (30 minutes) **a. Definition and Importance of Coupling and Cohesion** - Coupling: The degree of dependency between modules. - Cohesion: The degree to which a module performs a single function. - Importance of low coupling and high cohesion for maintainability and scalability. **b. Types of Coupling and their Impact** - Discuss different types of coupling from high to low (content, common, control, stamp, data, message). - Show examples in code (e.g., global variables causing high coupling). - Explain how high coupling leads to poor maintainability and difficult debugging. **c. High Cohesion vs. Low Cohesion** - Define high cohesion (focused modules) and low cohesion (scattered responsibilities). - Real-world examples: - High Cohesion: A module for user authentication (handles only login/logout). - Low Cohesion: A module that handles user authentication, report generation, and email notifications. - Demonstrate how high cohesion improves modular design.
Closure	1. Summarize key takeaways: • Low coupling = Better modularity. • High cohesion = More focused and reusable code.

	• Maintaining a balance between coupling and cohesion is essential for scalable software. 2. Ask students to share real-world examples of abstraction in software development.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What is the relationship between coupling and cohesion? 2. Why should we aim for low coupling and high cohesion in software design? 3. Can you give an example of high cohesion and low coupling in real-world applications? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 4.3	Course Name: Software Engineering Topic: Structured Design (Top-Down Functional Decomposition)	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of structured design and its role in software development. b. Explain the top-down functional decomposition approach. c. Identify the key steps involved in structured design. d. Analyze the advantages and disadvantages of structured design.
Teaching Aids (if any)	a. PPT slides for structured design principles b. Chalk and Board for function decomposition diagrams c. Case study example to demonstrate structured design d. Video Lecture on top-down functional decomposition
Teaching Development	1. **Introduction** (10 minutes) - Ask students: "How do we break down complex problems into smaller tasks?" - Explain how structured design focuses on breaking a system into smaller manageable modules. - Introduce the top-down functional decomposition approach as a key structured design method. 2. **Development** (30 minutes) **a. Top-Down Functional Decomposition Approach** - Define top-down design: • Breaking a large problem into smaller sub-problems. • Each sub-problem is further decomposed until reaching the atomic functional units. • Show a hierarchical diagram to illustrate function decomposition. - Example: ATM System • Main function: Process Transaction • Decomposed into: Authenticate User, Process Withdrawal, Process Deposit, Print Receipt, etc. **b. Steps in Structured Design** - Understand the Problem Statement - Identify the Major Functions of the System - Break Down Each Function into Sub-Functions - Create a Structure Chart to Represent the Hierarchy - Refine and Optimize the Design
Closure	1. Summarize key takeaways: • Top-down decomposition helps break down complexity. • Structured design improves clarity but may lack flexibility. • Modern designs often integrate structured and object-oriented approaches.

	2. Ask students to apply functional decomposition to a simple project (e.g., Library Management System).
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What is the main idea behind structured design? 2. How does top-down decomposition improve software development? 3. What are some real-world applications where structured design is useful?? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 4.4	Course Name: Software Engineering Topic: Object-Oriented and Event-Driven Design	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Understand the principles of Object-Oriented Design (OOD). b. Explain key concepts such as encapsulation, inheritance, and polymorphism. c. Identify and interpret UML diagrams used in object-oriented design. d. Define event-driven design and discuss its real-world applications.
Teaching Aids (if any)	a. PPT slides for OOD principles and UML diagrams b. Chalk and Board for explaining relationships in OOD c. Case study example to illustrate object-oriented and event-driven design d. Video Lecture on OOD and event-driven programming
Teaching Development	1. **Introduction** (10 minutes) - Ask students: "How do we structure software to make it reusable and maintainable?" - Explain the shift from structured design to object-oriented design (OOD). - Introduce event-driven design as an approach used in interactive applications. 2. **Development** (30 minutes) **a. Principles of OOD** - Encapsulation: Hiding data and exposing only necessary functionality • Example: A bank account class with private balance data. - Inheritance: Reusing and extending existing classes • Example: A Vehicle class inherited by Car and Bike. **b. UML Diagrams for Object-Oriented Design** - Class Diagrams: Show classes, attributes, and relationships. - Use Case Diagrams: Describe system interactions. - Sequence Diagrams: Represent object interactions over time. - Example: Library Management System UML Diagram. **c. Event-Driven Design: Concept and Real-World Applications** - **Definition**: A programming paradigm where the flow is determined by events. - **Key components**: • **Event Listener**: Detects events. • **Event Handler**: Executes code when an event occurs. • **Event Loop**: Keeps the program running and listens for new events.

	- **Real-World Applications**: • GUI Applications (e.g., Button Click in JavaScript). • IoT Devices (e.g., Sensor Triggering). • Game Development (e.g., Character Movement on Key Press).
Closure	1. Summarize key takeaways: • OOD makes software more reusable, maintainable, and scalable. • UML diagrams help in visualizing object relationships. • Event-driven design is used in interactive applications and real-time systems. 2. Ask students to create a class diagram for a Student Management System.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. How does encapsulation improve software security? 2. What are the differences between inheritance and polymorphism? 3. Can you name an application where event-driven design is crucial? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 4.5	Course Name: Software Engineering Topic: Component-Level and Test-Driven Design	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Understand the principles of component-level design and modularization. b. Identify key considerations in designing software components. c. Explain the Test-Driven Development (TDD) methodology. d. Apply TDD by writing unit tests before implementation.
Teaching Aids (if any)	a. PPT slides explaining component-level design and TDD b. Code examples demonstrating unit tests before implementation c. Live coding session for writing test cases using a testing framework (e.g., JUnit, PyTest) d. Video demonstration of TDD in action
Teaching Development	1. **Introduction** (10 minutes) - Ask students: "How do we ensure that software components are modular and testable?" - Introduce Component-Level Design as an approach to breaking software into smaller, reusable modules. - Explain the importance of testing early in the development process. 2. **Development** (30 minutes) **a. Component-Level Design: Key Considerations and Modularization** - Define component and modular design in software engineering. - Explain monolithic vs. modular approaches. - Real-world example: Designing an e-commerce platform with independent components (cart, user authentication, payment gateway, etc.). - Discuss cohesion and coupling and why they matter in component design. b. **Introduction to Test-Driven Development (TDD)** - Define TDD and explain the Red-Green-Refactor cycle. - Benefits of TDD: Bug reduction, improved design, better maintainability. - Live coding example: Writing a simple function and tests before implementation. - Case study discussion: How TDD helps in Agile software development. **c. Event-Driven Design: Concept and Real-World Applications** - **Definition**: A programming paradigm where the flow is determined by events. - **Key components**: • **Event Listener**: Detects events. • **Event Handler**: Executes code when an event occurs.

	• **Event Loop**: Keeps the program running and listens for new events.
Closure	1. Summarize key takeaways: • What is component-level design, and why is it important? • How does TDD improve software quality? • Key differences between monolithic and modular design. 2. Ask students to share examples of how they can apply component-level design and TDD in their own projects.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. How does component-level design improve software maintainability? 2. Why is test-driven development important in modern software development? 3. Can you give an example of a project where TDD would be highly beneficial? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 4.6	Course Name: Software Engineering Topic: Coding Practices – Techniques, Refactoring, and Integration Strategies	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Understand best coding techniques that improve software maintainability. b. Explain refactoring and its role in enhancing code quality. c. Apply basic refactoring techniques to improve software efficiency.
Teaching Aids (if any)	a. Presentation slides b. Whiteboard c. Live coding examples
Teaching Development	1. **Introduction** (10 minutes) - Ask: "What makes code good or bad?" - Discuss characteristics of clean code (readability, reusability, simplicity). - Explain why refactoring is necessary in software development. 2. **Development** (30 minutes) a. **Best Coding Practices** - Naming Conventions, Code Formatting, Modularization. - Use of Constants and Functions to avoid code repetition. - Example: Refactoring a complex function into multiple smaller functions. b. **Refactoring Techniques** - Define Refactoring: Improving code structure without changing functionality. - Techniques: • Extract Method (Splitting long functions into smaller ones). • Rename Variables (Enhancing code readability). • Reduce Redundant Code (Eliminating unnecessary repetitions). - Live coding: Refactor a function to follow best practices.
Closure	1. Summarize the importance of refactoring in software quality. 2. Ask students to identify a refactoring opportunity in a sample code.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What is the purpose of refactoring? 2. Name one technique used in refactoring and explain its benefit. 3. Why is readability more important than shorter code? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 4.7	Course Name: Software Engineering Topic: Software Integration Strategies and Internal Documentation	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Identify different software integration strategies. b. Understand the importance of internal documentation. c. Apply best practices for code documentation.
Teaching Aids (if any)	a. Presentation slides b. Whiteboard c. Example API documentation
Teaching Development	1. **Introduction** (10 minutes) - Ask: "Why do developers need to integrate different software modules?" - Discuss how poor integration leads to software failures. - Explain why documentation is necessary for long-term maintenance. 2. **Development** (30 minutes) a. **Integration Strategies** - Big Bang Integration vs. Incremental Integration. - Continuous Integration (CI) in DevOps. - Case Study: How companies use CI/CD to improve software quality. b. **Internal Documentation** - Types of documentation: Code comments, API docs, UML diagrams. - Best Practices: Keep it simple, avoid over-commenting, use meaningful function names. - Example: Well-documented vs. poorly documented code comparison.
Closure	1. Summarize the role of integration and documentation in software projects. 2. Ask students to document a sample function using best practices.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. Why is continuous integration important? 2. How does good internal documentation help future developers? 3. What are the characteristics of good API documentation? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 4.8	Course Name: Software Engineering Topic: Data Flow Diagrams, Transform & Transaction Mapping	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Define Data Flow Diagrams (DFDs) and their levels. b. Differentiate between Transform Analysis and Transaction Analysis. c. Apply Transform and Transaction Mapping techniques to system design.
Teaching Aids (if any)	a. Presentation slides b. Whiteboard c. Case study diagrams
Teaching Development	1. **Introduction** (10 minutes) - Ask: "How do we visualize how data moves in a system?" - Explain how DFDs help in system analysis and design. - Show a simple DFD for an online shopping system. 2. **Development** (30 minutes) a. **Data Flow Diagrams** - Levels of DFDs (0, 1, 2, etc.) - Example: Banking System DFDs from Level 0 to Level 2. b. **Transform vs. Transaction Analysis** - Transform Analysis: Data-driven systems (e.g., Payroll processing). - Transaction Analysis: Event-driven systems (e.g., ATM operations). **c. Transform & Transaction Mapping** - Steps to map a DFD into software components. - Case Study: Mapping a DFD of an inventory management system to software components.
Closure	1. Summarize how DFDs help in system design. 2. Ask students to create a Level 0 DFD for a library management system.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What is the difference between Transform Analysis and Transaction Analysis? 2. Why are DFDs useful in system design? 3. How would you apply transaction mapping to an e-commerce system? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 5.1	Course Name: Software Quality Assurance, Testing and Maintenance Topic: Software Quality and Software Quality Assurance	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Define software quality and explain its importance. b. Understand the need and role of software quality assurance (SQA). c. Describe key activities in an SQA plan.
Teaching Aids (if any)	a. Presentation slides b. Whiteboard c. Case study diagrams
Teaching Development	1. **Introduction** (10 minutes) ○ Ask: "Why do we care about software quality?" ○ Brief discussion on software success/failure cases due to quality. ○ Define software quality and introduce SQA. 2. **Development** (30 minutes) **a. Definition and attributes (e.g., reliability, maintainability).** b. **Software Quality Assurance** **c. Objectives and scope** d. **SQA Plan: standards, procedures, reviews, audits, and tools.**
Closure	1. Recap the role and scope of SQA. 2. Ask students to list 3 ways quality affects end-users.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What is software quality? 2. Why do we need SQA? 3. Who is responsible for quality in a software project? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 5.2	Course Name: Software Engineering Topic: Formal Technical Reviews	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Define formal technical review (FTR). b. Explain different types of reviews. c. Describe the FTR process.
Teaching Aids (if any)	a. Presentation slides b. Whiteboard c. Case study diagrams
Teaching Development	1. **Introduction** (10 minutes) ◦ Ask: "How do we find defects before testing starts?" ◦ Briefly introduce the review process. 2. **Development** (30 minutes) **a. Formal Technical Reviews** ◦ Types: walkthroughs, inspections, audits - Participants and roles - Steps of an FTR - Benefits: early detection, cost savings **b. Objectives and scope**
Closure	1. Summarize the importance of early reviews. 2. Students create a sample review checklist.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What is an FTR? 2. Why are FTRs essential for quality? 3. Who participates in an FTR? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 5.3	Course Name: Software Engineering Topic: Software Quality Metrics: McCall's Quality Factors	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Explain the purpose of software metrics. b. Describe McCall's Quality Factors. c. Apply the model to assess software quality.
Teaching Aids (if any)	a. Presentation slides b. Whiteboard c. Case study diagrams
Teaching Development	1. **Introduction** (10 minutes) ◦ Ask: "Can we measure software quality like temperature or speed?" ◦ Introduce software metrics. 2. **Development** (30 minutes) **a. Need for Software Metrics** **b. McCall's Model** - Three categories: Product operation, transition, revision - Quality Factors: Correctness, reliability, efficiency, etc.
Closure	1. Summarize McCall's factors. 2. Ask students to evaluate any application using McCall's factors.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What is a software metric? 2. Name and explain any two McCall quality factors. 3. Why is measurement important in SQA? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 5.4	Course Name: Software Engineering Topic: Software Reliability	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Define software reliability. b. Explain factors affecting software reliability. c. Describe techniques to measure and improve reliability.
Teaching Aids (if any)	a. Presentation slides b. Whiteboard c. Case study diagrams
Teaching Development	1. **Introduction** (10 minutes) ○ Ask: "What happens when software fails unexpectedly?" ○ Introduce the concept of reliability. 2. **Development** (30 minutes) **a. Definition and importance** **b. Factors: complexity, defect rate, testing** **c. Reliability metrics: MTBF, availability** - Three categories: Product operation, transition, revision - Quality Factors: Correctness, reliability, efficiency, etc.
Closure	1. Recap definitions and metrics. 2. Ask students to relate software reliability with system downtime.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. Define software reliability. 2. How do we measure reliability? 3. How can we improve reliability? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 5.5	Course Name: Software Engineering Topic: Software Testing Fundamentals	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Understand the objectives and principles of software testing. b. Identify the test levels and test types. c. Describe testing lifecycle phases.
Teaching Aids (if any)	a. Presentation slides b. Whiteboard c. Case study diagrams
Teaching Development	1. **Introduction** (10 minutes) ○ Ask: "Can we prove a program is error-free?" ○ Importance of testing in development. 2. **Development** (30 minutes) a. Definitions and objectives b. Principles of testing c. Test levels and types
Closure	1. Recap key objectives. 2. Students list two reasons testing is critical.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What are the objectives of software testing? 2. List the main types of testing. Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 5.6	Course Name: Software Engineering Topic: Software Testing Fundamentals	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Understand the objectives and principles of software testing. b. Identify the test levels and test types. c. Describe testing lifecycle phases.
Teaching Aids (if any)	a. Presentation slides b. Whiteboard c. Case study diagrams
Teaching Development	1. **Introduction** (10 minutes) ◦ Ask: "Can we prove a program is error-free?" ◦ Importance of testing in development. 2. **Development** (30 minutes) a. Definitions and objectives b. Principles of testing c. Test levels and types
Closure	1. Recap key objectives. 2. Students list two reasons testing is critical.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What are the objectives of software testing? 2. List the main types of testing. Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 5.7	Course Name: Software Engineering Topic: White Box Testing, Basic Path Testing	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Define white box testing. b. Explain basic path testing with flow graphs. c. Design test cases using path testing.
Teaching Aids (if any)	a. Presentation slides b. Whiteboard c. Case study diagrams
Teaching Development	1. **Introduction** (10 minutes) ○ Ask: "How can we test from the inside?" ○ Introduce white box testing. 2. **Development** (30 minutes) a. White Box Testing b. Basic Path Testing ○ Flow graphs, cyclomatic complexity - Example walk-through
Closure	1. Recap steps in path testing. 2. Students draw flow graph for a simple function.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What is white box testing? 2. How do we calculate cyclomatic complexity? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 5.8	**Course Name: Software Engineering** **Topic: White Box Testing, Basic Path Testing**	**Course No.: COM-603**

Objectives	At the end of the lesson the student shall be able to: a. Define white box testing. b. Explain basic path testing with flow graphs. c. Design test cases using path testing.
Teaching Aids (if any)	a. Presentation slides b. Whiteboard c. Case study diagrams
Teaching Development	1. **Introduction** (10 minutes) ◌ Ask: "How can we test from the inside?" ◌ Introduce white box testing. 2. **Development** (30 minutes) a. White Box Testing b. Basic Path Testing ◌ Flow graphs, cyclomatic complexity - Example walk-through
Closure	1. Recap steps in path testing. 2. Students draw flow graph for a simple function.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What is white box testing? 2. How do we calculate cyclomatic complexity? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 5.9	Course Name: Software Engineering Topic: Control Structure Testing	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Define control structure testing. b. Describe different techniques (condition, loop, and data flow testing). c. Apply them to test case design.
Teaching Aids (if any)	a. Flowcharts b. Sample programs
Teaching Development	1. **Introduction** (10 minutes) ○ Ask: "How can we test from the inside?" ○ Introduce white box testing. 2. **Development** (30 minutes) a. White Box Testing b. Basic Path Testing ○ Flow graphs, cyclomatic complexity - Example walk-through
Closure	1. Recap steps in path testing. 2. Students draw flow graph for a simple function.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What is white box testing? 2. How do we calculate cyclomatic complexity? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 5.9	Course Name: Software Engineering Topic: Software Testing Strategies	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Define different software testing strategies. b. Differentiate between top-down and bottom-up approaches. c. Explain test documentation and planning.
Teaching Aids (if any)	a. Strategy diagrams b. Planning templates
Teaching Development	1. **Introduction** (10 minutes) ○ Ask: "How can we test from the inside?" ○ Introduce white box testing. 2. **Development** (30 minutes) a. Testing Strategies b. Top-down, bottom-up, big bang, sandwich c. Test Planning d. Flow graphs, cyclomatic complexity
Closure	1. Recap steps in path testing. 2. Students draw flow graph for a simple function.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What is white box testing? 2. How do we calculate cyclomatic complexity? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 5.11	**Course Name: Software Engineering** **Topic: Unit, Integration, Validation, and System Testing**	**Course No.: COM-603**

Objectives	At the end of the lesson the student shall be able to: a. Define and differentiate testing levels. b. Explain goals and methods for each level. c. Relate testing levels to SDLC phases.
Teaching Aids (if any)	a. Diagrams b. Life cycle models
Teaching Development	1. **Introduction** (10 minutes) ○ Ask: "How can we test from the inside?" ○ Introduce white box testing. 2. **Development** (30 minutes) a. Testing Strategies b. Top-down, bottom-up, big bang, sandwich c. Test Planning d. Flow graphs, cyclomatic complexity
Closure	1. Recap steps in path testing. 2. Students draw flow graph for a simple function.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What is white box testing? 2. How do we calculate cyclomatic complexity? Spend 5 minutes to wrap up and consolidate the learnings

Lesson Plan No. 5.12	Course Name: Software Engineering Topic: Maintenance Characteristics, Reverse Engineering, Re-engineering Course	Course No.: COM-603

Objectives	At the end of the lesson the student shall be able to: a. Explain types and characteristics of software maintenance. b. Define reverse and re-engineering. c. Describe how maintenance supports software quality.
Teaching Aids (if any)	a. Diagrams b. Life cycle models
Teaching Development	1. **Introduction** (10 minutes) ○ Ask: "How can we test from the inside?" ○ Introduce white box testing. 2. **Development** (30 minutes) a. Maintenance Types: Corrective, Adaptive, Perfective, Preventive b. Characteristics and challenges c. Reverse Engineering and Re-engineering
Closure	1. Recap steps in path testing. 2. Students draw flow graph for a simple function.
Evaluation	Reflective Questions (What Why Who?). Allow students to answer and discuss. 1. What is white box testing? 2. How do we calculate cyclomatic complexity? Spend 5 minutes to wrap up and consolidate the learnings