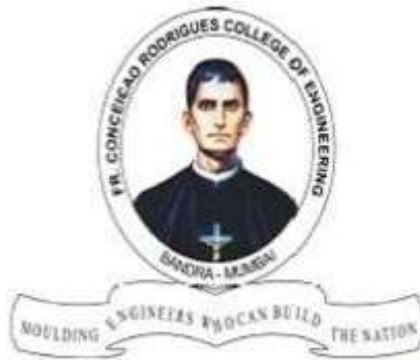# Fr. Conceicao Rodrigues College of Engineering

Fr. Agnel Ashram, Bandstand, Bandra (W) Mumbai 400 050.

Software Engineering Lab

(CSL501)

Department of Computer Engineering

University of Mumbai

B.E. (Computer, Sem - V) - AY 2023-24

by

Dr. B. S. Daga

LAB OBJECTIVES

Welcome to the Software Engineering Lab! This lab serves as a practical and hands-on platform for students to delve into the fascinating world of software engineering. The primary focus of this lab is to equip students with essential skills and knowledge required to develop high-quality and efficient software solutions that meet real-world demands and challenges.

The objectives of this lab are multifaceted, aiming to cultivate a comprehensive understanding of various aspects of software engineering. Throughout this course, students will embark on a journey to achieve the following outcomes:

1. **Identify Requirements and Apply Software Process Models**: Understanding the requirements of a software project is the cornerstone of building successful software solutions. In this lab, students will learn various techniques to elicit, analyze, and document software requirements. Furthermore, they will gain hands-on experience in applying different software process models, such as the Waterfall, Agile, and Spiral models, to selected case studies. By doing so, students will appreciate the importance of selecting the right process model based on the project's characteristics and goals.
2. **Develop Architectural Models for the Selected Case Study:** Creating a robust and scalable software architecture is pivotal in ensuring the long-term success of any software project. In this lab, students will explore architectural design principles and methodologies to develop comprehensive architectural models for the chosen case study. Through practical exercises, they will gain insights into how to make informed architectural decisions and strike the right balance between performance, maintainability, and scalability.
3. **Use Computer-Aided Software Engineering (CASE) Tools:** Modern software engineering heavily relies on cutting-edge tools and technologies to streamline development processes and enhance productivity. In this lab, students will be introduced to various Computer-Aided Software Engineering (CASE) tools. They will learn how to leverage these tools effectively to automate tasks like code generation, debugging, testing, and documentation. Hands-on experience with CASE tools will empower students to optimize their development workflow and create high-quality software in a more efficient manner.

This lab is structured to foster a dynamic and interactive learning environment. Students will engage in practical exercises, individual and group projects, case studies, and discussions to reinforce their understanding of theoretical concepts.

As technology continues to evolve, the significance of software engineering in shaping the future becomes even more profound. I believe that through this lab, students will not only gain the technical skills needed for their careers but also develop problem-solving abilities and a mindset that embraces innovation and continuous learning.

I hope that this software engineering lab will ignite your passion for software development and equip you with the expertise to contribute meaningfully to the ever-changing landscape of the software industry.

INDEX

Experiment No.	Experiment Name
1	Software Requirement Specification
2	Implementing Project using SCRUM method on JIRA Tool
3	Implementing Project using KANBAN method on JIRA Tool
4	Calculating function points of the Project
5	Estimating project cost using COCOMO Model
6	Data flow analysis of the Project
7	Design using Object Oriented approach with emphasis on Cohesion and Coupling
8	Design test cases for performing black box testing
9	Design test cases for performing white box testing
10	Version controlling & Risk Analysis of the project

Lab Experiment 01

Experiment Name: Software Requirement Specification (SRS) as per IEEE Format

Objective: The objective of this lab experiment is to guide students in creating a Software Requirement Specification (SRS) document following the IEEE (Institute of Electrical and Electronics Engineers) standard format. The IEEE format ensures a structured and consistent approach to capturing software requirements, facilitating effective communication among stakeholders and streamlining the software development process.

Introduction: Software Requirement Specification (SRS) is a formal document that precisely defines the functional and non-functional requirements of a software project. The IEEE standard format provides a systematic framework for organizing the SRS, making it comprehensive, clear, and easily understandable by all parties involved in the project.

Lab Experiment Overview:

1. Introduction to IEEE Standard: The lab session begins with an overview of the IEEE standard format for SRS. Students are introduced to the various sections and components of the SRS as per the standard.
2. Selecting a Sample Project: Students are provided with a sample software project or case study for which they will create the SRS. The project should be of moderate complexity to cover essential elements of the IEEE format.
3. Requirement Elicitation and Analysis: Students conduct requirement elicitation sessions with the project stakeholders to gather relevant information. They analyze the collected requirements to ensure they are complete, unambiguous, and feasible.
4. Structuring the SRS: Using the IEEE standard guidelines, students organize the SRS document into sections such as Introduction, Overall Description, Specific Requirements, Appendices, and other relevant subsections.
5. Writing the SRS Document: In this phase, students write the SRS document, ensuring it is well-structured, coherent, and adheres to the IEEE format. They include necessary diagrams, use cases, and requirements descriptions.
6. Peer Review and Feedback: Students exchange their SRS documents with their peers for review and feedback. This review session allows them to receive constructive criticism and suggestions for improvement.
7. Finalization and Submission: After incorporating the feedback received during the review session, students finalize the SRS document and submit it for assessment.

Learning Outcomes: By the end of this lab experiment, students are expected to:

- Understand the IEEE standard format for creating an SRS document.
- Develop proficiency in requirement elicitation, analysis, and documentation techniques.
- Acquire the skills to structure an SRS document following the IEEE guidelines.

- Demonstrate the ability to use diagrams, use cases, and textual descriptions to define software requirements.
- Enhance communication and collaboration skills through peer reviews and feedback sessions.

Pre-Lab Preparations: Before the lab session, students should review the IEEE standard for SRS documentation, familiarize themselves with the various sections and guidelines, and understand the importance of clear and unambiguous requirements.

Materials and Resources:

- IEEE standard for SRS documentation
- Sample software project or case study for creating the SRS
- Computers with word processing software for document preparation
- Review feedback forms for peer assessment

Conclusion: The Software Requirement Specification (SRS) lab experiment in accordance with the IEEE standard format equips students with essential skills in documenting software requirements systematically. Following the IEEE guidelines ensures that the SRS document is well-organized, comprehensive, and aligned with industry standards, facilitating seamless communication between stakeholders and software developers. Through practical hands-on experience in creating an SRS as per the IEEE format, students gain a deeper understanding of the significance of precise requirement definition in the success of software projects. Mastering the IEEE standard for SRS documents prepares students to be effective software engineers, capable of delivering high-quality software solutions that meet client expectations and industry best practices.

Lab Experiment 02

Experiment Name: Implementing Project Using Scrum Method on JIRA Tool in Software Engineering

Objective: The objective of this lab experiment is to introduce students to the Scrum framework and its implementation using the JIRA tool. Students will gain practical experience in managing a software project using Scrum principles and learn how to utilize JIRA as a project management tool to track and organize tasks, sprints, and team collaboration.

Introduction: Scrum is an agile project management methodology that promotes iterative development, collaboration, and continuous improvement. JIRA is a widely used tool that supports Scrum practices, providing teams with features to plan, track, and manage software projects effectively.

Lab Experiment Overview:

1. Introduction to Scrum: The lab session begins with an overview of the Scrum framework, including its roles (Product Owner, Scrum Master, and Development Team), events (Sprint Planning, Daily Standup, Sprint Review, and Sprint Retrospective), and artifacts (Product Backlog, Sprint Backlog, and Increment).
2. JIRA Tool Introduction: Students are introduced to the JIRA tool and its capabilities in supporting Scrum project management. They learn to create projects, epics, user stories, tasks, and sub-tasks in JIRA.
3. Defining the Project: Students are assigned a sample software project and create a Product Backlog, listing all the required features, user stories, and tasks for the project.
4. Sprint Planning: Students organize the Product Backlog into Sprints, selecting user stories and tasks for the first Sprint. They estimate the effort required for each task using story points.
5. Implementation in JIRA: Students use the JIRA tool to create a Sprint Backlog, add the selected user stories and tasks, and assign them to team members.
6. Daily Standup: Students conduct a simulated Daily Standup meeting, where they update the progress of their tasks and discuss any impediments they are facing.
7. Sprint Review and Retrospective: At the end of the Sprint, students review the completed tasks, demonstrate the implemented features, and gather feedback from their peers. They also conduct a Sprint Retrospective to identify areas of improvement for the next Sprint.
8. Continuous Iteration: Students continue implementing subsequent Sprints, repeating the Sprint Planning, Daily Standup, and Sprint Review & Retrospective events.
9. Conclusion and Reflection: At the end of the lab experiment, students reflect on their experience with Scrum and JIRA, discussing the advantages and challenges they encountered during the project.

Learning Outcomes: By the end of this lab experiment, students are expected to:

- Understand the Scrum framework and its principles in agile project management.

- Gain practical experience in using the JIRA tool for project management in a Scrum environment.
- Learn to create and manage Product Backlogs, Sprint Backlogs, and track progress using JIRA.
- Develop collaborative skills through Daily Standup meetings and Sprint Reviews.
- Gain insights into the iterative nature of software development and the importance of continuous improvement.

Pre-Lab Preparations: Before the lab session, students should familiarize themselves with the Scrum framework and the basics of the JIRA tool. They should review Scrum roles, events, and artifacts, as well as the features of JIRA relevant to Scrum implementation.

Materials and Resources:

- Computers with internet access for accessing the JIRA tool
- Project brief and details for the sample software project
- Whiteboard or projector for explaining Scrum concepts

Conclusion: The lab experiment on implementing a project using Scrum on the JIRA tool offers students a hands-on experience in agile project management. By utilizing Scrum principles and JIRA's capabilities, students learn to collaborate effectively, manage tasks efficiently, and adapt to changing requirements. The practical exposure to Scrum and JIRA enhances their understanding of agile methodologies, equipping them with valuable skills for real-world software development projects. The lab experiment encourages students to embrace the agile mindset, promoting continuous improvement and customer-centric software development practices.

Lab Experiment 03

Experiment Name: Implementing Project Using Kanban Method on JIRA Tool in Software Engineering

Objective: The objective of this lab experiment is to introduce students to the Kanban method and its implementation using the JIRA tool. Students will gain practical experience in managing a software project using Kanban principles and learn how to utilize JIRA as a project management tool to visualize workflow, manage work items, and improve team productivity.

Introduction: Kanban is an agile project management method that emphasizes visualizing work, limiting work in progress, and continuously improving the workflow. JIRA is a popular tool that supports Kanban practices, allowing teams to manage their tasks and activities effectively.

Lab Experiment Overview:

1. Introduction to Kanban: The lab session begins with an overview of the Kanban method, including the principles of visualizing work, managing flow, and making incremental improvements.
2. JIRA Tool Introduction: Students are introduced to the JIRA tool and its features for implementing Kanban. They learn to create boards, swimlanes, columns, and customize workflows.
3. Defining the Project: Students are assigned a sample software project and create a Kanban board in JIRA to visualize their workflow. They set up columns to represent different stages of their development process.
4. Creating Work Items: Students create work items (tasks, user stories, or issues) on the Kanban board, representing the work that needs to be done.
5. Managing Workflow: Students move work items through the columns on the Kanban board as they progress through their development process. They monitor work in progress limits to maintain an efficient workflow.
6. Continuous Improvement: Students conduct regular team meetings to discuss the workflow, identify bottlenecks, and make improvements to enhance their efficiency.
7. Completion and Review: At the end of the lab experiment, students review their project progress on the Kanban board. They discuss their experiences with implementing the Kanban method on JIRA and share insights on its effectiveness.
8. Conclusion and Reflection: Students reflect on their experience with Kanban and JIRA, discussing the benefits and challenges they encountered during the project. They also consider how Kanban principles can be applied to future software development projects.

Learning Outcomes: By the end of this lab experiment, students are expected to:

- Understand the Kanban method and its application in agile project management.
- Gain practical experience in using the JIRA tool to implement Kanban boards and workflows.
- Learn to visualize work, manage flow, and limit work in progress using Kanban principles.

- Develop team collaboration skills by continuously improving the workflow through regular team meetings.
- Appreciate the importance of visualizing and managing work items for better project management.

Pre-Lab Preparations: Before the lab session, students should familiarize themselves with the Kanban method and the basics of the JIRA tool. They should review Kanban principles, visualizing workflows, and the features of JIRA relevant to Kanban implementation.

Materials and Resources:

- Computers with internet access for accessing the JIRA tool
- Project brief and details for the sample software project
- Whiteboard or projector for explaining Kanban concepts

Conclusion: The lab experiment on implementing a project using the Kanban method on the JIRA tool provides students with practical insights into agile project management. By applying Kanban principles and utilizing JIRA's capabilities, students learn to visualize their work, manage flow efficiently, and continuously improve their development process. The hands-on experience with Kanban and JIRA fosters teamwork, collaboration, and adaptability, enabling students to effectively manage software projects with a focus on efficiency and quality. The lab experiment encourages students to adopt Kanban's lean principles, promoting a culture of continuous improvement and optimizing their workflow to deliver valuable software products.

Lab Experiment 04

Experiment Name: Calculating Function Points of the Project in Software Engineering

Objective: The objective of this lab experiment is to introduce students to the concept of Function Points and the Function Point Analysis (FPA) technique for measuring software size and complexity. Students will gain practical experience in calculating Function Points for a sample software project, enabling them to estimate development effort and assess project scope accurately.

Introduction: Function Points are a unit of measurement used in software engineering to quantify the functionality delivered by a software application. Function Point Analysis (FPA) is a widely used technique to assess the functional size of a software project based on its user requirements.

Lab Experiment Overview:

1. Introduction to Function Points: The lab session begins with an overview of Function Points, explaining the concept and their significance in software size measurement.
2. Defining the Sample Project: Students are provided with a sample software project along with its user requirements. The project may involve modules, functionalities, and user interactions.
3. Identifying Functionalities: Students identify and categorize the functionalities in the sample project, such as data inputs, data outputs, inquiries, external interfaces, and internal logical files.
4. Assigning Complexity Weights: For each identified functionality, students assign complexity weights based on specific criteria provided in the FPA guidelines.
5. Calculating Unadjusted Function Points: Students calculate the Unadjusted Function Points (UFP) by summing up the weighted functionalities.
6. Adjusting Function Points: Students apply adjustment factors (e.g., complexity, performance, and conversion) to the UFP to calculate the Adjusted Function Points (AFP).
7. Estimating Development Effort: Using historical data or industry benchmarks, students estimate the development effort required for the project based on the calculated AFP.
8. Conclusion and Reflection: Students discuss the significance of Function Points in software estimation and reflect on their experience in calculating Function Points for the sample project.

Learning Outcomes: By the end of this lab experiment, students are expected to:

- Understand the concept of Function Points and their role in software size measurement.
- Gain practical experience in applying the Function Point Analysis (FPA) technique to assess software functionality.
- Learn to categorize functionalities and assign complexity weights in Function Point calculations.
- Develop estimation skills to assess development effort based on calculated Function Points.
- Appreciate the importance of accurate software size measurement in project planning and resource allocation.

Pre-Lab Preparations: Before the lab session, students should familiarize themselves with the concept of Function Points and the guidelines for their calculation. They should review the

categorization of functionalities and the complexity weighting factors used in Function Point Analysis.

Materials and Resources:

- Project brief and details for the sample software project
- Function Point Analysis guidelines and complexity weighting criteria
- Calculators or spreadsheet software for performing calculations

Conclusion: The lab experiment on calculating Function Points for a software project provides students with a practical approach to estimating software size and complexity. By applying the Function Point Analysis (FPA) technique, students gain insights into the importance of objective software measurement for project planning and resource allocation. The hands-on experience in identifying functionalities and assigning complexity weights enhances their estimation skills and equips them with valuable techniques for effective project management. The lab experiment encourages students to apply Function Point Analysis in real-world scenarios, promoting accuracy and efficiency in software size measurement for successful software engineering projects.

Lab Experiment 05

Experiment Name: Estimating Project Cost Using COCOMO Model in Software Engineering

Objective: The objective of this lab experiment is to introduce students to the COCOMO (Constructive Cost Model) estimation technique for estimating software project cost and effort. Students will gain practical experience in using the COCOMO model to estimate the development effort, duration, and resources required for a sample software project.

Introduction: COCOMO is a widely used algorithmic cost estimation model in software engineering. It helps in quantifying the effort and resources needed for software development based on project size, complexity, and other factors.

Lab Experiment Overview:

1. Introduction to COCOMO Model: The lab session begins with an introduction to the COCOMO model, explaining the different versions (Basic, Intermediate, and Advanced) and their application in software cost estimation.
2. Defining the Sample Project: Students are provided with a sample software project along with its functional and non-functional requirements, complexity, and size metrics.
3. COCOMO Parameters: Students learn about the COCOMO model parameters, such as Effort Adjustment Factor (EAF), Scale Factors, and Cost Drivers, and how they influence the project's effort estimation.
4. Effort and Duration Estimation: Using the COCOMO model formula, students estimate the effort and duration required to complete the sample project based on the provided size and complexity metrics.
5. Resource Allocation: Students estimate the number of required resources, such as developers, testers, and project managers, based on the calculated effort and project duration.
6. Sensitivity Analysis: Students perform sensitivity analysis by varying the COCOMO parameters to observe their impact on the project cost estimation.
7. Conclusion and Reflection: Students discuss the significance of COCOMO in software project estimation and reflect on their experience in estimating project cost using the COCOMO model.

Learning Outcomes: By the end of this lab experiment, students are expected to:

- Understand the COCOMO model and its application in software cost estimation.
- Gain practical experience in using the COCOMO model to estimate effort, duration, and resources for a software project.
- Learn to consider various project factors and adjust COCOMO parameters for accurate cost estimation.
- Develop estimation skills for resource allocation and project planning.
- Appreciate the importance of data accuracy and project size metrics in project cost estimation.

Pre-Lab Preparations: Before the lab session, students should familiarize themselves with the COCOMO model, its parameters, and the cost estimation formula. They should also review the factors that influence the project's size and complexity.

Materials and Resources:

- Project brief and details for the sample software project
- COCOMO model guidelines and cost estimation formula
- Calculators or spreadsheet software for performing calculations

Conclusion: The lab experiment on estimating project cost using the COCOMO model provides students with practical insights into software cost estimation techniques. By applying the COCOMO model to a sample software project, students gain hands-on experience in assessing effort, duration, and resource requirements. The sensitivity analysis allows them to understand the impact of various factors on cost estimation. The lab experiment encourages students to use COCOMO in real-world scenarios, promoting informed decision-making in software project planning and resource allocation. Accurate cost estimation using COCOMO enhances project management and contributes to the successful execution of software engineering projects.

Lab Experiment 06

Experiment Name: Data Flow Analysis of the Project in Software Engineering

Objective: The objective of this lab experiment is to introduce students to Data Flow Analysis, a technique used in software engineering to understand the flow of data within a software system. Students will gain practical experience in analyzing the data flow of a sample software project, identifying data dependencies, and modeling data flow diagrams.

Introduction: Data Flow Analysis is a vital activity in software development, helping engineers comprehend how data moves through a system, aiding in identifying potential vulnerabilities and ensuring data integrity.

Lab Experiment Overview:

1. Introduction to Data Flow Analysis: The lab session begins with an overview of Data Flow Analysis, its importance in software engineering, and its applications in ensuring data security and accuracy.
2. Defining the Sample Project: Students are provided with a sample software project, which includes the data elements, data stores, processes, and data flows.
3. Data Flow Diagrams: Students learn how to construct Data Flow Diagrams (DFDs) to visualize the data flow in the software system. They understand the symbols used in DFDs, such as circles for processes, arrows for data flows, and rectangles for data stores.
4. Identifying Data Dependencies: Students analyze the sample project and identify the data dependencies between various components. They determine how data is generated, processed, and stored in the system.
5. Constructing Data Flow Diagrams: Using the information gathered, students create Data Flow Diagrams that represent the data flow within the software system. They include both high-level context diagrams and detailed level-0 and level-1 diagrams.
6. Data Flow Analysis: Students analyze the constructed DFDs to identify potential bottlenecks, inefficiencies, and security vulnerabilities related to data flow.
7. Conclusion and Reflection: Students discuss the significance of Data Flow Analysis in software development and reflect on their experience in constructing and analyzing Data Flow Diagrams.

Learning Outcomes: By the end of this lab experiment, students are expected to:

- Understand the concept of Data Flow Analysis and its importance in software engineering.
- Gain practical experience in constructing Data Flow Diagrams to represent data flow in a software system.
- Learn to identify data dependencies and relationships within the software components.
- Develop analytical skills to analyze Data Flow Diagrams for potential issues and vulnerabilities.
- Appreciate the role of Data Flow Analysis in ensuring data integrity, security, and efficiency.

Pre-Lab Preparations: Before the lab session, students should familiarize themselves with Data Flow Analysis concepts and the symbols used in Data Flow Diagrams. They should review data dependencies and data flow modeling in software systems.

Materials and Resources:

- Project brief and details for the sample software project
- Whiteboard or projector for constructing Data Flow Diagrams
- Drawing tools or software for creating the diagrams

Conclusion: The lab experiment on Data Flow Analysis of a software project equips students with essential skills in understanding data flow within a system. By constructing and analyzing Data Flow Diagrams, students gain insights into how data is processed, stored, and exchanged, enabling them to identify potential issues and security concerns. The practical experience in Data Flow Analysis enhances their ability to design efficient and secure software systems that ensure data integrity and meet user requirements. The lab experiment encourages students to apply Data Flow Analysis in real-world software development projects, promoting better data management and system design practices.

Lab Experiment 07

Experiment Name: Design Using Object-Oriented Approach with Emphasis on Cohesion and Coupling in Software Engineering

Objective: The objective of this lab experiment is to introduce students to the Object-Oriented (OO) approach in software design, focusing on the principles of cohesion and coupling. Students will gain practical experience in designing a sample software project using OO principles to achieve high cohesion and low coupling, promoting maintainable and flexible software.

Introduction: The Object-Oriented approach is a powerful paradigm in software design, emphasizing the organization of code into objects, classes, and interactions. Cohesion and Coupling are essential design principles that guide the creation of well-structured and modular software.

Lab Experiment Overview:

1. Introduction to Object-Oriented Design: The lab session begins with an introduction to the Object-Oriented approach, explaining the concepts of classes, objects, inheritance, polymorphism, and encapsulation.
2. Defining the Sample Project: Students are provided with a sample software project that requires design and implementation. The project may involve multiple modules or functionalities.
3. Cohesion in Design: Students learn about Cohesion, the degree to which elements within a module or class belong together. They understand the different types of cohesion, such as functional, sequential, communicational, and temporal, and how to achieve high cohesion in their design.
4. Coupling in Design: Students explore Coupling, the degree of interdependence between modules or classes. They understand the types of coupling, such as content, common, control, and stamp coupling, and strive for low coupling in their design.
5. Applying OO Principles: Using the Object-Oriented approach, students design classes and identify their attributes, methods, and interactions. They ensure that classes have high cohesion and are loosely coupled.
6. Class Diagrams: Students create Class Diagrams to visually represent their design, illustrating the relationships between classes and their attributes and methods.
7. Design Review: Students conduct a design review session, where they present their Class Diagrams and receive feedback from their peers.
8. Conclusion and Reflection: Students discuss the significance of Object-Oriented Design principles, Cohesion, and Coupling in creating maintainable and flexible software. They reflect on their experience in applying these principles during the design process.

Learning Outcomes: By the end of this lab experiment, students are expected to:

- Understand the Object-Oriented approach and its core principles, such as encapsulation, inheritance, and polymorphism.
- Gain practical experience in designing software using OO principles with an emphasis on Cohesion and Coupling.

- Learn to identify and implement high cohesion and low coupling in their design, promoting modular and maintainable code.
- Develop skills in creating Class Diagrams to visualize the relationships between classes.
- Appreciate the importance of design principles in creating robust and adaptable software.

Pre-Lab Preparations: Before the lab session, students should review Object-Oriented concepts, such as classes, objects, inheritance, and polymorphism. They should also familiarize themselves with the principles of Cohesion and Coupling in software design.

Materials and Resources:

- Project brief and details for the sample software project
- Whiteboard or projector for creating Class Diagrams
- Drawing tools or software for visualizing the design

Conclusion: The lab experiment on designing software using the Object-Oriented approach with a focus on Cohesion and Coupling provides students with essential skills in creating well-structured and maintainable software. By applying OO principles and ensuring high cohesion and low coupling, students design flexible and reusable code, facilitating future changes and enhancements. The experience in creating Class Diagrams enhances their ability to visualize and communicate their design effectively. The lab experiment encourages students to adopt design best practices, promoting modular and efficient software development in their future projects. Emphasizing Cohesion and Coupling in the Object-Oriented approach empowers students to create high-quality software that meets user requirements and adapts to evolving needs with ease.

Lab Experiment 08

Experiment Name: Designing Test Cases for Performing Black Box Testing in Software Engineering

Objective: The objective of this lab experiment is to introduce students to the concept of Black Box Testing, a testing technique that focuses on the functional aspects of a software system without examining its internal code. Students will gain practical experience in designing test cases for Black Box Testing to ensure the software meets specified requirements and functions correctly.

Introduction: Black Box Testing is a critical software testing approach that verifies the functionality of a system from an external perspective, without knowledge of its internal structure. It is based on the software's specifications and requirements, making it an essential part of software quality assurance.

Lab Experiment Overview:

1. Introduction to Black Box Testing: The lab session begins with an introduction to Black Box Testing, explaining its purpose, advantages, and the types of tests performed, such as equivalence partitioning, boundary value analysis, decision table testing, and state transition testing.
2. Defining the Sample Project: Students are provided with a sample software project along with its functional requirements, use cases, and specifications.
3. Identifying Test Scenarios: Students analyze the sample project and identify test scenarios based on its requirements and use cases. They determine the input values, expected outputs, and test conditions for each scenario.
4. Equivalence Partitioning: Students apply Equivalence Partitioning to divide the input values into groups that are likely to produce similar results. They design test cases based on each equivalence class.
5. Boundary Value Analysis: Students perform Boundary Value Analysis to determine test cases that focus on the boundaries of input ranges. They identify test cases near the minimum and maximum values of each equivalence class.
6. Decision Table Testing: Students use Decision Table Testing to handle complex logical conditions in the software's requirements. They construct decision tables and derive test cases from different combinations of conditions.
7. State Transition Testing: If applicable, students apply State Transition Testing to validate the software's behavior as it moves through various states. They design test cases to cover state transitions.
8. Test Case Documentation: Students document the designed test cases, including the test scenario, input values, expected outputs, and any preconditions or postconditions.
9. Test Execution: In a simulated test environment, students execute the designed test cases and record the results.
10. Conclusion and Reflection: Students discuss the importance of Black Box Testing in software quality assurance and reflect on their experience in designing test cases for Black Box Testing.

Learning Outcomes: By the end of this lab experiment, students are expected to:

- Understand the concept and significance of Black Box Testing in software testing.
- Gain practical experience in designing test cases for Black Box Testing based on functional requirements.
- Learn to apply techniques such as Equivalence Partitioning, Boundary Value Analysis, Decision Table Testing, and State Transition Testing in test case design.
- Develop documentation skills for recording and organizing test cases effectively.
- Appreciate the role of Black Box Testing in identifying defects and ensuring software functionality.

Pre-Lab Preparations: Before the lab session, students should familiarize themselves with Black Box Testing concepts, Equivalence Partitioning, Boundary Value Analysis, Decision Table Testing, and State Transition Testing techniques.

Materials and Resources:

- Project brief and details for the sample software project
- Whiteboard or projector for explaining Black Box Testing techniques
- Test case templates for documentation

Conclusion: The lab experiment on designing test cases for Black Box Testing provides students with essential skills in verifying software functionality from an external perspective. By applying various Black Box Testing techniques, students ensure comprehensive test coverage and identify potential defects in the software. The experience in designing and executing test cases enhances their ability to validate software behavior and fulfill functional requirements. The lab experiment encourages students to incorporate Black Box Testing into their software testing strategies, promoting robust and high-quality software development. Emphasizing test case design in Black Box Testing empowers students to contribute to software quality assurance and deliver reliable and customer-oriented software solutions.

Lab Experiment 09

Experiment Name: Designing Test Cases for Performing White Box Testing in Software Engineering

Objective: The objective of this lab experiment is to introduce students to the concept of White Box Testing, a testing technique that examines the internal code and structure of a software system. Students will gain practical experience in designing test cases for White Box Testing to verify the correctness of the software's logic and ensure code coverage.

Introduction: White Box Testing, also known as Structural Testing or Code-Based Testing, involves assessing the internal workings of a software system. It aims to validate the correctness of the code, identify logic errors, and achieve maximum code coverage.

Lab Experiment Overview:

1. Introduction to White Box Testing: The lab session begins with an introduction to White Box Testing, explaining its purpose, advantages, and the techniques used, such as statement coverage, branch coverage, and path coverage.
2. Defining the Sample Project: Students are provided with a sample software project along with its source code and design documentation.
3. Identifying Test Scenarios: Students analyze the sample project and identify critical code segments, including functions, loops, and conditional statements. They determine the test scenarios based on these code segments.
4. Statement Coverage: Students apply Statement Coverage to ensure that each statement in the code is executed at least once. They design test cases to cover all the statements.
5. Branch Coverage: Students perform Branch Coverage to validate that every branch in the code, including both true and false branches in conditional statements, is executed at least once. They design test cases to cover all branches.
6. Path Coverage: Students aim for Path Coverage by ensuring that all possible execution paths through the code are tested. They design test cases to cover different paths, including loop iterations and condition combinations.
7. Test Case Documentation: Students document the designed test cases, including the test scenario, input values, expected outputs, and any assumptions made.
8. Test Execution: In a test environment, students execute the designed test cases and record the results, analyzing the code coverage achieved.
9. Conclusion and Reflection: Students discuss the significance of White Box Testing in software quality assurance and reflect on their experience in designing test cases for White Box Testing.

Learning Outcomes: By the end of this lab experiment, students are expected to:

- Understand the concept and importance of White Box Testing in software testing.
- Gain practical experience in designing test cases for White Box Testing to achieve code coverage.

- Learn to apply techniques such as Statement Coverage, Branch Coverage, and Path Coverage in test case design.
- Develop documentation skills for recording and organizing test cases effectively.
- Appreciate the role of White Box Testing in validating code logic and identifying errors.

Pre-Lab Preparations: Before the lab session, students should familiarize themselves with White Box Testing concepts, Statement Coverage, Branch Coverage, and Path Coverage techniques.

Materials and Resources:

- Project brief and details for the sample software project
- Whiteboard or projector for explaining White Box Testing techniques
- Test case templates for documentation

Conclusion: The lab experiment on designing test cases for White Box Testing equips students with essential skills in assessing the internal code of a software system. By applying various White Box Testing techniques, students ensure comprehensive code coverage and identify logic errors in the software. The experience in designing and executing test cases enhances their ability to validate code behavior and ensure code quality. The lab experiment encourages students to incorporate White Box Testing into their software testing strategies, promoting robust and high-quality software development. Emphasizing test case design in White Box Testing empowers students to contribute to software quality assurance and deliver reliable and efficient software solutions.

Lab Experiment 10

Experiment Name: Version Controlling & Risk Analysis of the Project in Software Engineering

Objective: The objective of this lab experiment is to introduce students to version controlling and risk analysis in software engineering. Students will gain practical experience in using version control tools to manage project code and collaborating with team members effectively. Additionally, they will learn how to identify and analyze potential risks associated with software projects.

Introduction: Version controlling is a crucial practice in software development, enabling teams to track changes, collaborate efficiently, and manage codebase effectively. Risk analysis involves identifying potential project risks and implementing mitigation strategies to ensure project success.

Lab Experiment Overview:

1. Introduction to Version Controlling: The lab session begins with an introduction to version controlling, explaining its significance in software development, and the principles of version control systems like Git.
2. Version Control Setup: Students set up a version control repository using a version control system (e.g., Git) and create a project structure to store code and other project artifacts.
3. Collaboration and Version Control: Students learn to collaborate with team members using version control, understanding branching, merging, and resolving conflicts.
4. Version Control Operations: Students perform version control operations, such as committing changes, creating branches, merging branches, and tagging releases.
5. Risk Analysis: The lab introduces students to risk analysis, its purpose, and the types of risks that can occur during a software project's lifecycle.
6. Identifying Project Risks: Students analyze the sample software project and identify potential risks related to technology, scope, resources, scheduling, and external factors.
7. Risk Analysis and Mitigation: Students assess the impact and probability of each identified risk and develop mitigation strategies to address and minimize their potential effects.
8. Risk Management Plan: Students create a risk management plan, documenting the identified risks, their assessment, and the corresponding mitigation strategies.
9. Version Controlling and Risk Analysis in Practice: Students apply version controlling practices to manage project code, collaborate with team members, and incorporate risk management strategies in their project.
10. Conclusion and Reflection: Students discuss the importance of version controlling and risk analysis in software development projects and reflect on their experience in implementing version control and risk management techniques.

Learning Outcomes: By the end of this lab experiment, students are expected to:

- Understand the importance and benefits of version controlling in software development.
- Gain practical experience in using version control tools (e.g., Git) for managing project code and collaboration.

- Learn the concepts of branching, merging, and conflict resolution in version control.
- Identify potential risks in software projects and develop mitigation strategies for risk management.
- Appreciate the significance of risk analysis in ensuring project success and effective project planning.

Pre-Lab Preparations: Before the lab session, students should familiarize themselves with version control concepts, version control tools (e.g., Git), and the fundamentals of risk analysis in software projects.

Materials and Resources:

- Version control system (e.g., Git) and project repositories
- Sample software project details
- Risk analysis templates and risk management plan formats

Conclusion: The lab experiment on version controlling and risk analysis in software engineering equips students with crucial skills for effective project management and collaboration. By using version control tools, students learn how to manage code changes and coordinate their work with team members efficiently. The risk analysis component enhances their ability to identify and mitigate potential project risks, ensuring a more successful software development process. The lab experiment encourages students to incorporate version controlling and risk analysis in their future software projects, promoting better project organization, and decision-making. Emphasizing version controlling and risk analysis empowers students to contribute to high-quality software development and successful project delivery.

APPENDIX

The IEEE standard format for Software Requirements Specification (SRS)

The IEEE standard format for Software Requirements Specification (SRS) is a guideline provided by the Institute of Electrical and Electronics Engineers (IEEE) to document software requirements in a consistent and structured manner. The standard is known as IEEE 830-1998, titled "IEEE Recommended Practice for Software Requirements Specifications." Below is an outline of the typical sections found in an SRS document following the IEEE 830-1998 standard:

1. Introduction 1.1 Purpose 1.2 Document Conventions 1.3 Intended Audience 1.4 Scope 1.5 Definitions, Acronyms, and Abbreviations 1.6 References 1.7 Overview of the SRS

2. Overall Description 2.1 Product Perspective 2.2 Product Functions 2.3 User Classes and Characteristics 2.4 Operating Environment 2.5 Design and Implementation Constraints 2.6 User Documentation 2.7 Assumptions and Dependencies

3. Specific Requirements 3.1 External Interfaces 3.1.1 User Interfaces 3.1.2 Hardware Interfaces 3.1.3 Software Interfaces 3.1.4 Communication Interfaces 3.2 Functional Requirements 3.2.1 [Requirement ID] [Requirement Description] ... 3.3 Performance Requirements 3.4 Design Constraints 3.5 Software System Attributes 3.5.1 Reliability 3.5.2 Availability 3.5.3 Security 3.5.4 Maintainability 3.5.5 Portability

4. Supporting Information 4.1 Appendix A: Glossary 4.2 Appendix B: Analysis Models 4.3 Appendix C: To Be Determined 4.4 Appendix D: List of Figures 4.5 Appendix E: List of Tables

Please note that the actual content and level of detail may vary depending on the complexity of the software project and specific organizational requirements. The SRS document should be a comprehensive and unambiguous specification that serves as a basis for further development and validation of the software system.

Function Point Analysis (FPA) is a technique used to measure the functional size of a software application based on the user requirements. It helps in estimating effort, cost, and resources required for software development. Below is a basic template for calculating function points:

Function Point Calculation Template

1. External Inputs (EI):

- Identify the number of unique user inputs that affect the application's processing logic.
- For each input, consider the complexity (low, average, high) based on the number of data elements involved.

No.	Name of External Input	Complexity (Low/Average/High)	Weight
1			
2			
...			
N			
Total			

2. External Outputs (EO):

- Identify the number of unique user outputs generated by the application.
- For each output, consider the complexity (low, average, high) based on the number of data elements presented.

No.	Name of External Output	Complexity (Low/Average/High)	Weight
1			
2			
...			
N			
Total			

3. External Inquiries (EQ):

- Identify the number of unique user inquiries that require information retrieval from the application.
- For each inquiry, consider the complexity (low, average, high) based on the number of data elements involved.

No.	Name of External Inquiry	Complexity (Low/Average/High)	Weight
1			
2			
...			
N			
Total			

4. Internal Logical Files (ILF):

- Identify the number of unique data storage areas maintained by the application.
- For each logical file, consider the complexity (low, average, high) based on the number of data elements.

No.	Name of Internal Logical File	Complexity (Low/Average/High)	Weight
1			
2			
...			
N			
Total			

5. External Interface Files (EIF):

- Identify the number of unique data storage areas used by the application and maintained by external applications.
- For each external interface file, consider the complexity (low, average, high) based on the number of data elements.

No.	Name of External Interface File	Complexity (Low/Average/High)	Weight
1			
2			
...			
N			
Total			

6. Function Point Count:

- Sum up the weights of all the identified function points to get the total function point count.

Calculation Formula: Total Function Point Count = EI Weight + EO Weight + EQ Weight + ILF Weight + EIF Weight

Conclusion: Function Point Analysis helps in quantifying the functional size of a software application, which can be used for estimating development effort, project cost, and resource allocation. By accurately calculating function points, software development teams can make informed decisions and effectively manage software projects.