**LAB MANUAL**

**SUBJECT: SOFTWARE ENGINEERING**

**CLASS: THIRD YEAR IT**

**Experiment No:1**

### **Aim:** To finalize the project topic and define the problem statement by analyzing user needs, identifying key challenges, and framing objectives for software development.

### **Theory**

Software engineering emphasizes structured problem-solving to ensure efficient and reliable software development. The initial phase of any software project involves topic finalization and problem definition, which forms the foundation for subsequent phases.

1. **Topic Finalization:**
   * Select a relevant and feasible project topic aligned with industry or societal needs.
   * Ensure the topic integrates software engineering principles and involves a mix of functional and non-functional requirements.
2. **Problem Definition:**
   * Understand the real-world scenario that necessitates the software system.
   * Clearly articulate the problem, considering constraints, stakeholders, and the environment in which the system will operate.

The importance of this phase lies in providing clarity of purpose, ensuring alignment with project goals, and mitigating risks early in the project lifecycle.

### **Learning Objective:**

* To identify and select an appropriate project topic.
* To clearly define the problem and scope for software development.
* To establish a solid foundation for further stages of the software development process.

### **Learning Outcome:**

At the end of this experiment, students will be able to:

1. Finalize a relevant and impactful topic for software development.
2. Define the problem statement in a structured manner.
3. Justify the need for the software system based on real-world challenges.

### **Course Outcomes (COs):**

* CO1: Understand and explain the fundamentals of software engineering, the software process frameworks, and umbrella activities to manage and improve software development.
* CO3: Identify and formulate functional and non-functional requirements for software systems, and perform cost estimation using various models.

### **Cognitive Levels of Attainment as per Bloom’s Taxonomy:**

* **L2 (Understand):** Comprehend and explain the importance of problem definition in software engineering.
* **L3 (Apply):** Apply problem-solving techniques to finalize the project topic and define the problem statement.

### **Programme Outcome (POs):**

* **PO1: Engineering Knowledge:** Apply fundamental concepts of engineering to finalize a project topic and define problems effectively.
* **PO2: Problem Analysis:** Identify and analyze complex engineering problems for defining software requirements.
* **PO3: Design/Development of Solutions:** Design structured problem definitions and solutions aligned with the identified needs.
* **PO9: Communication:** Communicate the problem statement effectively, both in writing and presentations.

### **Programme Specific Outcome (PSOs):**

* **PSO1:** Apply software engineering principles to identify sustainable IT solutions.
* **PSO2:** Integrate technologies to frame scalable solutions addressing real-time problems.
* **PSO3**: Design structured problem definitions to develop web-based solutions for societal challenges.

### **Result & Discussion:**

* **Result:**
  1. Successfully finalized the project topic.
  2. Defined a clear and concise problem statement.
  3. Identified potential functional and non-functional requirements.
* **Discussion:**  
  The experiment highlighted the significance of topic selection and problem definition in ensuring project success. The exercise also emphasized considering stakeholder needs, constraints, and the system environment.

### **Conclusion:**

The process of topic finalization and problem definition forms a critical part of the software development lifecycle. A well-defined problem ensures clarity, reduces risks, and serves as a foundation for developing innovative and effective software solutions.

#### ****Experiment No. 2: Literature Survey on Software Modeling****

### **Aim:**

To conduct a literature survey on various software development models and select the most suitable model for the project based on its requirements, constraints, and desired outcomes.

### **Theory:**

The selection of an appropriate software development model is crucial to the success of a software project. Various models provide structured approaches to plan, execute, and manage projects effectively.

1. **Popular Software Development Models:**
   * **Waterfall Model:** Sequential and phase-based approach.
   * **Iterative Model: Repeated refinement of software prototypes.**
   * **Agile Model: Focus on incremental delivery and adaptability.**
   * **Spiral Model: Emphasis on risk management with iterative cycles.**
   * **V-Model: Testing and development occur simultaneously.**
2. **Criteria for Selecting a Model:**
   * **Project Type and Size: Small vs. large-scale projects.**
   * **Flexibility and Risk: Dynamic requirements and risk mitigation.**
   * **Stakeholder Involvement: Frequency and depth of collaboration.**
   * **Timeline and Budget: Strict deadlines and resource constraints.**

The literature survey allows teams to understand these models and select the one that aligns best with their project’s objectives and constraints.

### **Learning Objective:**

* To understand different software development models and their characteristics.
* To analyze the pros and cons of various models.
* To select the most suitable model for the project based on the findings.

### **Learning Outcome:**

At the end of this experiment, students will be able to:

1. Identify and describe various software development models.
2. Evaluate the suitability of each model for specific project scenarios.
3. Select and justify the choice of a model for their project.

### **Course Outcomes (COs):**

* CO1: Understand and explain the fundamentals of software engineering, the software process frameworks, and umbrella activities to manage and improve software development.
* CO2: Analyze and compare traditional and agile software development models, including their applicability to various project scenarios.

### **Cognitive Levels of Attainment as per Bloom’s Taxonomy:**

* **L1 (Remember):** Identify and list various software development models.
* **L2 (Understand): Explain the characteristics and advantages of different models.**
* **L4 (Analyze): Evaluate the strengths and weaknesses of models to select the most suitable one for the project.**

### **Programme Outcome (PO) Attainment:**

* **PO1: Engineering Knowledge: Apply knowledge of engineering fundamentals to understand software development models.**
* **PO2: Problem Analysis: Analyze the characteristics of various models to solve complex problems.**
* **PO5: Engineering Tool Usage: Utilize research tools and resources to perform a detailed literature survey.**
* **PO11: Life-Long Learning: Develop the ability to explore and evaluate emerging software engineering models.**

### **Programme Specific Outcome (PSO) Attainment:**

* **PSO1: Apply software engineering knowledge to select sustainable and efficient development models for IT projects.**

### **Result & Discussion:**

* **Result**:
  1. Identified and documented the characteristics of various software development models.
  2. Evaluated each model's applicability to the project scenario.
  3. Selected the most suitable model for the project and justified the choice.
* **Discussion:**  
  The literature survey emphasized the importance of understanding different software development models to ensure their proper alignment with project requirements. The selected model balances flexibility, risk, and development speed.

### **Conclusion:**

A systematic literature survey provides a comprehensive understanding of software development models. Selecting the appropriate model ensures that the project adheres to timelines, meets stakeholder expectations, and mitigates potential risks effectively.

#### ****Experiment No. 3: Technical Article on Software Development Methodology****

### **Aim:**

To write a detailed technical article about the selected software development methodology for the project, including its description, applications, pros and cons, and justification for selection.

### **Theory:**

Software development methodologies provide structured frameworks for planning, managing, and executing software projects. Selecting the right methodology ensures alignment with project objectives, resource constraints, and stakeholder expectations. Popular methodologies include Agile, Waterfall, Iterative, Spiral, and Rapid Application Development (RAD). Each methodology is suitable for specific types of projects and offers unique benefits and challenges.

### **Learning Objective:**

* To understand and describe a selected software development methodology in detail.
* To analyze the applicability of the methodology to specific project scenarios.
* To develop critical writing and technical reporting skills.

### **Learning Outcome:**

At the end of this experiment, students will be able to:

1. Describe the selected software development methodology in detail.
2. Evaluate the situations in which the methodology is applicable and justify its selection.
3. Develop technical writing skills to produce professional-level documentation.

### **Course Outcomes (COs):**

* **CO1: Understand and explain the fundamentals of software engineering, the software process frameworks, and umbrella activities to manage and improve software development.**
* **CO2: Analyze and compare traditional and agile software development models, including their applicability to various project scenarios.**
* **CO5: Evaluate and mitigate software project risks and apply configuration management practices to maintain project integrity.**

### **Cognitive Levels of Attainment as per Bloom’s Taxonomy:**

* **L2 (Understand):** Comprehend the chosen methodology and its characteristics.
* **L3 (Apply): Implement critical thinking to analyze the applicability of the methodology.**
* **L6 (Create): Produce a well-structured, plagiarism-free technical article.**

### **Programme Outcome (POs):**

* **PO1: Engineering Knowledge:** Apply knowledge of engineering principles to describe and justify the chosen methodology.
* **PO2: Problem Analysis: Analyze the suitability of the methodology for specific project needs.**
* **PO9: Communication: Effectively communicate the methodology and its applications through a detailed technical article.**
* **PO11: Life-Long Learning: Develop the ability to independently research and analyze new and existing software methodologies.**

### **Programme Specific Outcome (PSOs):**

* **PSO1:** Apply software engineering knowledge to select sustainable IT solutions for various domains, including software programming and development.

### **Result & Discussion:**

* **Result:**
  1. Researched and selected a software development methodology suitable for the project.
  2. Produced a plagiarism-free technical article that provides a comprehensive overview of the methodology, including its applications and evaluation.
* **Discussion:**  
  The detailed article highlights the selected methodology’s characteristics, its applicability to the project, and an evaluation of its pros and cons. The choice of methodology was justified based on its ability to meet project requirements effectively.

### **Conclusion:**

Writing a technical article helps in deepening the understanding of software development methodologies and their practical applications. Selecting the most suitable methodology ensures that the project adheres to quality standards while efficiently utilizing resources and meeting stakeholder expectations.

#### ****Experiment No. 4: Write Software Requirements Specification (SRS) Document for Your Project****

### **Aim:**

To create a detailed Software Requirements Specification (SRS) document for the assigned project, clearly outlining functional and non-functional requirements, constraints, assumptions, and dependencies.

### **Theory:**

A Software Requirements Specification (SRS) is a critical document in the software development lifecycle. It describes the functionalities, performance, and design constraints of the system to be developed. The SRS serves as a reference for developers, testers, and stakeholders to ensure the alignment of expectations and deliverables. The SRS is structured according to IEEE 830 standards, which include the following sections:

1. **Introduction:** Purpose, scope, and definitions.
2. **Overall Description: Product perspective, user characteristics, and operating environment.**
3. **Functional Requirements: Detailed specifications of functionalities.**
4. **Non-functional Requirements: Performance, reliability, and usability constraints.**
5. **Assumptions and Dependencies: External factors that impact the system.**

### **Learning Objective:**

* To understand the importance of an SRS document in the software development process.
* To develop skills in writing clear, concise, and structured software requirements.
* To analyze and document functional and non-functional requirements comprehensively.

### **Learning Outcome:**

At the end of this experiment, students will be able to:

1. Understand the structure and components of an SRS document.
2. Write an SRS document that includes detailed functional and non-functional requirements for their project.
3. Analyze and evaluate constraints, assumptions, and dependencies to ensure project feasibility.

### **Course Outcomes (COs):**

* **CO1:** Understand and explain the fundamentals of software engineering, the software process frameworks, and umbrella activities to manage and improve software development.
* **CO3:** Identify and formulate functional and non-functional requirements for software systems, and perform cost estimation using various models.
* **CO5:** Evaluate and mitigate software project risks and apply configuration management practices to maintain project integrity.

### **Cognitive Levels of Attainment as per Bloom’s Taxonomy:**

* **L3 (Apply):** Apply theoretical knowledge to create a structured SRS document for the project.
* **L4 (Analyze):** Analyze and evaluate the project requirements to ensure completeness and feasibility.

### **Programme Outcome (POs):**

* **PO2: Problem Analysis:** Identify, formulate, and analyze project requirements systematically.
* **PO3: Design/Development of Solutions:** Develop detailed documentation to meet identified needs while considering safety, culture, and environmental constraints.
* **PO5: Engineering Tool Usage:** Utilize appropriate tools and techniques to draft a professional SRS document.
* **PO9: Communication:** Communicate effectively by documenting project requirements in a clear and concise manner.

### **Programme Specific Outcome (PSO):**

* **PSO1:** Apply software engineering principles to develop sustainable IT solutions.
* **PSO2:** Integrate requirements with technologies to provide effective and scalable solutions for real-world problems.

### **Result & Discussion:**

* **Result:** A structured and complete SRS document was created, specifying the functional and non-functional requirements of the project.
* **Discussion:**  
  The SRS document highlights the system requirements and constraints to ensure successful implementation. The document adheres to IEEE 830 standards and serves as a key reference for subsequent phases of software development.

### **Conclusion:**

Developing an SRS document helps in formalizing the understanding of project requirements and serves as a contract between stakeholders and the development team. It provides clarity and minimizes misunderstandings, ensuring that the developed software meets the desired specifications.

### **SRS Template (IEEE Standard)**

**1. Introduction**

* **Purpose**: State the purpose of the project.
* **Scope**: Define the boundaries of the project.
* **Definitions, Acronyms, and Abbreviations**: List all relevant terms and acronyms.
* **References**: Cite any reference documents used.

**2. Overall Description**

* **Product Perspective**: Describe how the product fits into the larger system.
* **User Characteristics**: Specify user roles and characteristics.
* **Operating Environment**: Define the hardware, software, and network environments.
* **Constraints**: Outline project constraints, e.g., standards or hardware limitations.

**3. Functional Requirements**

* List all system functionalities with detailed descriptions.

**4. Non-Functional Requirements**

* **Performance**: Define speed, scalability, and throughput requirements.
* **Reliability**: State uptime, recovery, and failover capabilities.
* **Usability**: Describe user experience requirements.

**5. Assumptions and Dependencies**

* List all external dependencies and assumptions.

This structure ensures that students can create a clear and professional SRS document for their projects, aligned with real-world standards.

**Experiment No. 5: Use Case and Gantt Chart Development**

### **Aim:**

To create use case diagrams and Gantt charts for the project, representing functional requirements and project timelines effectively.

### **Theory:**

**1. Use Case Diagrams**  
A use case diagram is a visual representation of the interactions between the system and its users (actors). It depicts the functional requirements of the system and helps in understanding the user’s perspective.

Key Components:

* **Actors:** External entities interacting with the system (e.g., users, devices).
* **Use Cases:** Specific functions or features of the system.
* **Relationships:** Associations between actors and use cases (e.g., include, extend).

**2. Gantt Charts :**  
A Gantt chart is a project management tool used to represent the project schedule. It shows tasks, their durations, dependencies, and progress over time.

Key Components:

* **Tasks:** Activities or deliverables in the project.
* **Duration:** Time allocated to each task.
* **Dependencies:** Relationships between tasks (e.g., Task A must finish before Task B starts).
* **Timeline:** A visual representation of the schedule.

### **Learning Objective:**

* To understand the role of use case diagrams in capturing functional requirements.
* To gain proficiency in creating Gantt charts for effective project scheduling and tracking.
* To apply analytical and design skills for project management and system design.

### **Learning Outcome:**

At the end of this experiment, students will be able to:

1. Develop use case diagrams to represent functional requirements of their project.
2. Create Gantt charts to manage project timelines and dependencies.
3. Demonstrate the ability to plan, schedule, and track project progress effectively.

### **Course Outcomes (COs):**

* **CO3:** Identify and formulate functional and non-functional requirements for software systems.
* **CO5**: Evaluate and mitigate software project risks and apply configuration management practices to maintain project integrity.

### **Cognitive Levels of Attainment as per Bloom’s Taxonomy:**

* **L3 (Apply):** Apply knowledge to create use case diagrams and Gantt charts for project planning.
* **L5 (Evaluate):** Evaluate project timelines and dependencies using Gantt charts to ensure efficient scheduling.

### **Programme Outcome (POs:**

* **PO2: Problem Analysis:** Analyze project requirements and represent them using use case diagrams.
* **PO3: Design/Development of Solutions:** Design Gantt charts to plan and manage project deliverables.
* **PO9: Communication:** Communicate project requirements and timelines through visual representations.
* **PO10: Project Management and Finance:** Utilize project management tools to organize tasks and resources effectively.

### **Programme Specific Outcome (PSOs):**

* **PSO1:** Apply software engineering principles to create sustainable IT solutions by effectively managing project timelines and requirements.

### **Result & Discussion:**

* **Result**: Developed a comprehensive use case diagram to represent functional requirements and a Gantt chart to manage project tasks and timelines.
* **Discussion:**
  + The use case diagram identified key interactions between actors and the system.
  + The Gantt chart highlighted task durations, dependencies, and the critical path for project completion.

### **Conclusion:**

Use case diagrams and Gantt charts are essential tools for understanding system requirements and managing project timelines. By developing these artifacts, students gained insights into system design and project management, ensuring better planning and execution of their project.

### **Steps for the Experiment**

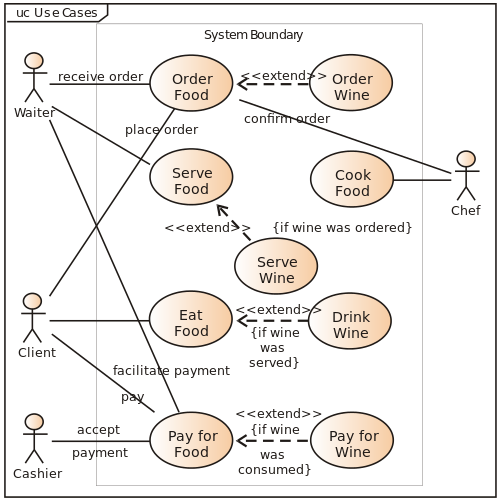
#### ****1. Use Case Diagram Development****

1. Identify all actors interacting with the system.
2. List the primary use cases or functionalities of the system.
3. Establish relationships (e.g., include, extend) between actors and use cases.
4. Use tools like Lucidchart, Microsoft Visio, or draw.io to create the diagram.

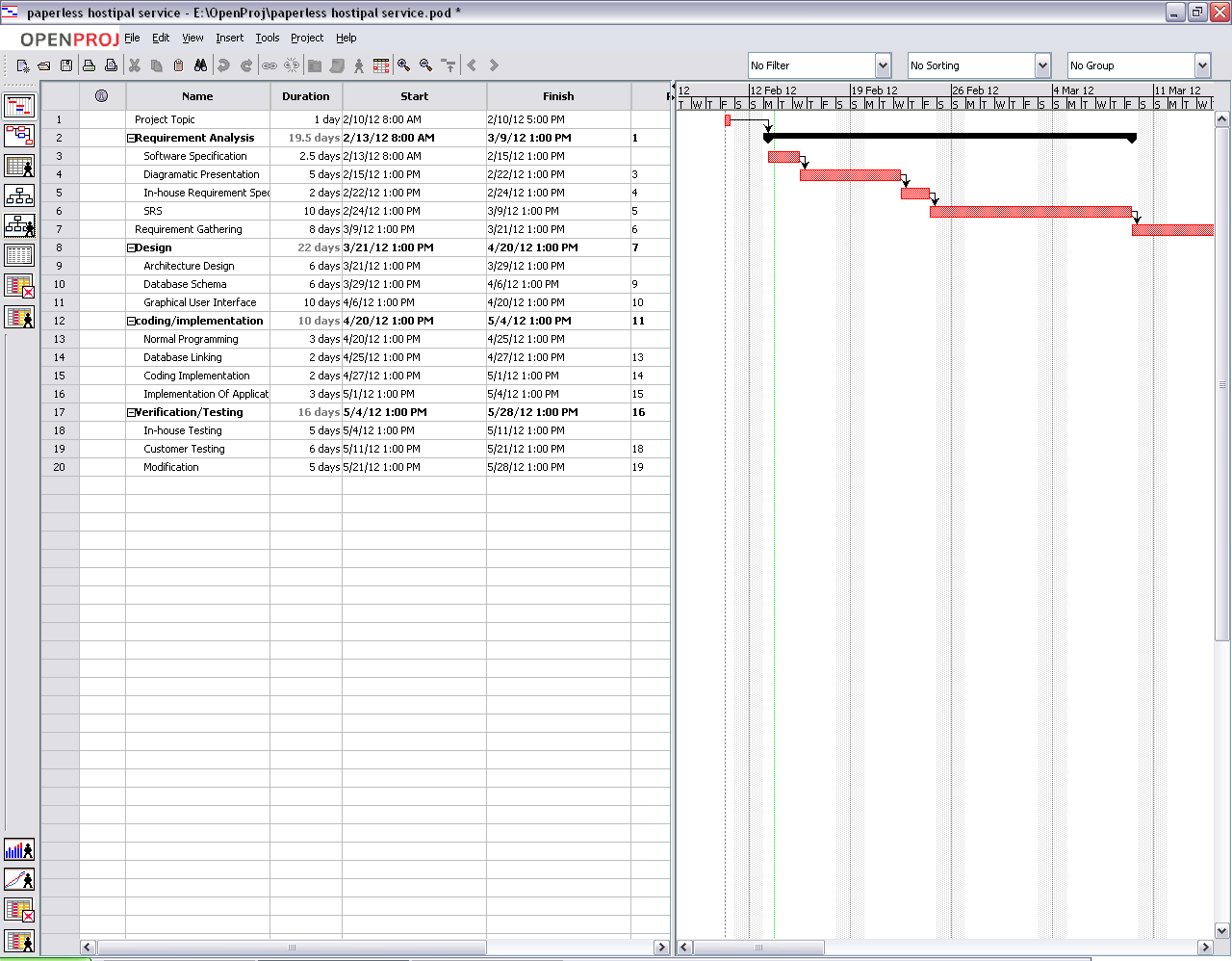
#### ****2. Gantt Chart Development****

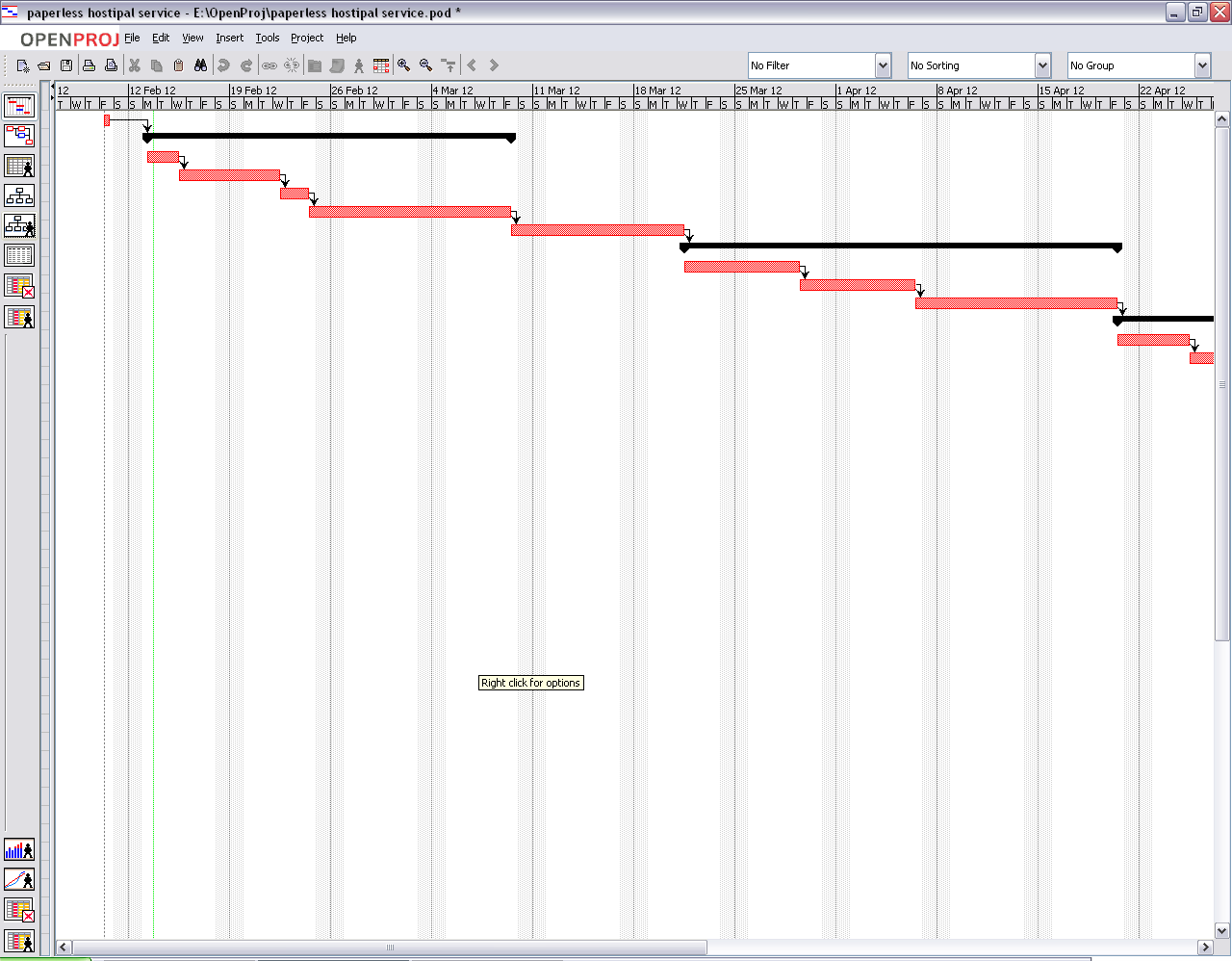
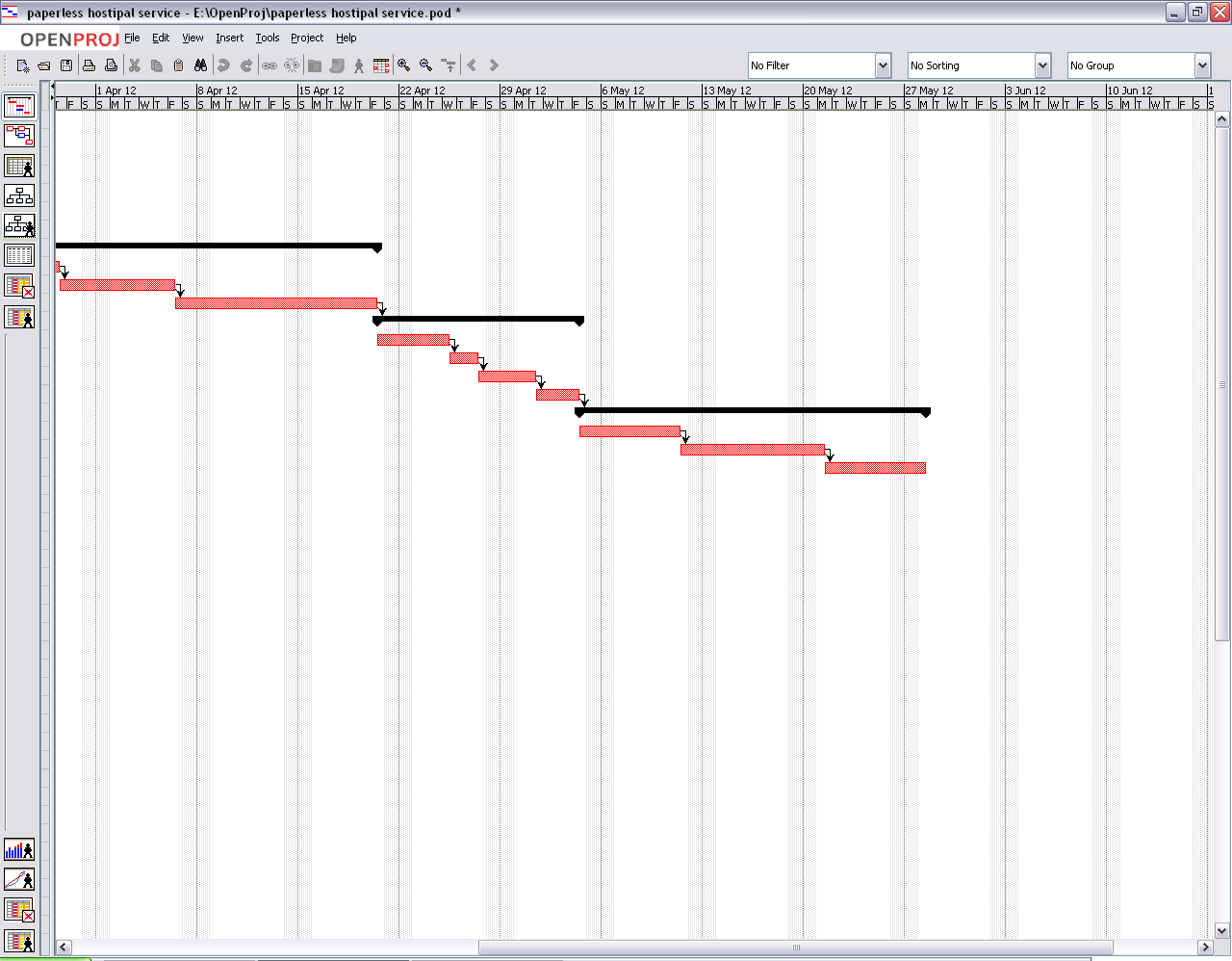
1. Break down the project into tasks or activities.
2. Assign durations and dependencies to each task.
3. Create a Gantt chart using tools like MS Project, Excel, or online Gantt chart tools.
4. Review and refine the chart to ensure it reflects the project schedule accurately.

### **Sample Use Case Diagram for Restaurant Management System:**



### **Sample Gantt Chart:**





**Experiment No. 6: Draw Data Flow Diagram for Your Project**

### **Aim:**

To design a Data Flow Diagram (DFD) for the project, representing the flow of information and processes involved in the system.

### **Theory:**

A Data Flow Diagram (DFD) is a graphical representation of data flow within a system. It illustrates how data moves between processes, data stores, and external entities (sources/sinks) in a system. DFDs help in understanding the system's functionality and identifying bottlenecks or inefficiencies.

**Key Components of DFD:**

1. **Processes:** Represented as circles or ovals, they depict actions performed on data.
2. **Data Stores:** Represented as open rectangles, they show where data is stored within the system.
3. **External Entities:** Represented as squares, they denote sources or sinks that interact with the system.
4. **Data Flows:** Represented as arrows, they show the flow of data between entities, processes, and data stores.

**Levels of DFD:**

* **Level 0 (Context Diagram):** Provides an overview of the system with a single process.
* **Level 1:** Breaks down the main process into sub-processes.
* **Level 2 and Beyond:** Further decomposes sub-processes into detailed steps.

### **Learning Objective:**

* To understand the role of Data Flow Diagrams in system design and analysis.
* To learn to represent processes, data stores, and data flows effectively.
* To develop analytical skills for visualizing the flow of information in the system.

### **Learning Outcome:**

At the end of this experiment, students will be able to:

1. Analyze the data flow in their project.
2. Develop Level 0, Level 1, and, if needed, Level 2 DFDs for the project.
3. Demonstrate the ability to use DFDs for identifying system functionality and data dependencies.

### **Course Outcomes (COs):**

* **CO3:** Identify and formulate functional and non-functional requirements for software systems.
* **CO5:** Evaluate and mitigate software project risks and apply configuration management practices to maintain project integrity.

### **Cognitive Levels of Attainment as per Bloom’s Taxonomy**

* **L3 (Apply):** Apply knowledge to create DFDs for the project.
* **L5 (Evaluate):** Evaluate the data flow and ensure its correctness and completeness.

### **Programme Outcome (POs):**

* **PO2: Problem Analysis:** Analyze complex system requirements to identify data flow and processes.
* **PO3: Design/Development of Solutions:** Design DFDs to represent the system’s processes and data interactions.
* **PO5: Engineering Tool Usage:** Use modern tools to create accurate DFDs.
* **PO10: Project Management and Finance:** Plan and document system processes for better project management.

### **Programme Specific Outcome (PSOs):**

* **PSO1:** Apply software engineering principles to design sustainable IT solutions for system data flow.
* **PSO2:** Integrate data flow representation into effective system modeling and real-time problem-solving.

### **Result & Discussion:**

* **Result:** Designed a Level 0 and Level 1 DFD for the project.
* **Discussion:**
  + The DFD successfully depicted the data flow and system processes.
  + Level 1 DFD provided detailed insights into sub-processes, aiding in system analysis.

### **Conclusion:**

Drawing Data Flow Diagrams is a crucial step in understanding and designing system processes. Through this experiment, students gained the ability to represent system functionality and data flow visually, ensuring clarity in system design and development.

### **Steps for the Experiment**

#### ****1. Level 0: Context Diagram Development****

1. Identify the main system as a single process.
2. List external entities interacting with the system.
3. Identify data flows between the system and external entities.
4. Create a context diagram showing the overall data flow.

#### ****2. Level 1: DFD Development****

1. Decompose the main process into sub-processes.
2. Identify data stores used by the system.
3. Map data flows between sub-processes, data stores, and external entities.
4. Ensure completeness and correctness in data representation.

#### ****3. Optional: Level 2 and Beyond****

1. Further break down sub-processes from Level 1 into smaller steps.
2. Refine data flows and interactions for more granularity.

#### ****4. Tools for DFD Creation****

* Use tools like Lucidchart, Microsoft Visio, draw.io, or SmartDraw for creating DFDs.

### **Sample Data Flow Diagrams:**

**Context diagram**

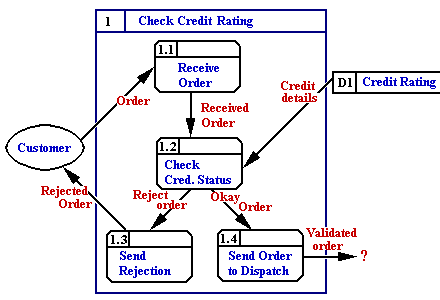
|  |
| --- |
| mhtml:file://F:\UWE-CEMS%20-%20Data%20Flow%20Diagrams.mht!images/dfdcont2.gif |

**The Top (1st level) DFD**

|  |
| --- |
| mhtml:file://F:\UWE-CEMS%20-%20Data%20Flow%20Diagrams.mht!images/dfdcred1.gif |

**The next step - the Next Level(s)**

**Each Process box in the Top Level diagram** will itself be made up of a number of processes, and will need to be decomposed as a second level diagram.



**DATA FLOW DIAGRAM**

Patient

Healthy Person

Gets admitted discharged

**LEVEL – 0 Data Flow Diagram (DFD)**

PATIENT

Discharge

Get admitted

**LEVEL – 1 Data Flow Diagram (DFD)**

Patient

Rooms codification

Patient Registration

Doctors codification

Total bill paid

Discharge

Diagnosis and Treatment

**LEVEL – 2 Data Flow Diagram (DFD)**

**Experiment No. 7: Estimate the Cost of Your Project**

### **Aim:**

To estimate the cost of the software project using appropriate cost estimation techniques and models.

### **Theory:**

Cost estimation in software engineering is the process of predicting the resources, effort, time, and money required to develop a software project. Accurate cost estimation helps in planning, budgeting, and resource allocation, ensuring the success of the project.

**Common Cost Estimation Techniques:**

1. **COCOMO (Constructive Cost Model):** A mathematical model that estimates cost based on the size of the project in terms of lines of code (LOC).
2. **Function Point Analysis (FPA):** Focuses on functional requirements and quantifies them for cost estimation.
3. **Expert Judgment:** Relies on the experience of experts to provide cost predictions.
4. **Analogy-Based Estimation:** Compares the current project with similar past projects to estimate costs.
5. **Parametric Models:** Uses parameters like effort, schedule, and quality to calculate costs.

**Factors Influencing Cost Estimation:**

* Project size
* Team experience and expertise
* Development model chosen
* Project complexity
* Resource availability

The function point method was originally developed by Bij Albrecht. A function point is a rough estimate of a unit of delivered functionality of a software project. Function points (FP) measure size in terms of the amount of functionality in a system. Function points are computed by first calculating an unadjusted function point count (UFC). Counts are made for the following categories

• Number of user inputs

Each user input that provides distinct application oriented data to the software is counted.

• Number of user outputs

Each user output that provides application oriented information to the user is counted. In this context "output" refers to reports, screens, error messages, etc. Individual data items within a report are not counted separately.

• Number of user inquiries

An inquiry is defined as an on-line input that results in the generation of some immediate software response in the form of an on-line output. Each distinct inquiry is counted.

• Number of files

Each logical master file is counted.

• Number of external interfaces

All machine-readable interfaces that are used to transmit information to another system are counted.

Once this data has been collected, a complexity rating is associated with each count according to Table

TABLE 1: Function point complexity weights.

|  |  |  |  |
| --- | --- | --- | --- |
| Measurement parameter | Weighting factor | | |
|  | Simple | Average | Complex |
| Number of user inputs | 3 | 4 | 6 |
| Number of user outputs | 4 | 5 | 7 |
| Number of user inquiries | 3 | 4 | 6 |
| Number of files | 7 | 10 | 15 |
| Number of external interfaces | 5 | 7 | 10 |

Each count is multiplied by its corresponding complexity weight and the results are summed to provide the UFC. The adjusted function point count (FP) is calculated by multiplying the UFC by a technical complexity factor (TCF) also referred to as Value Adjustment Factor (VAF). Components of the TCF are listed in Table 2

Table 2. Components of the technical complexity factor.

|  |  |  |  |
| --- | --- | --- | --- |
| F1 | Reliable back-up and recovery | F2 | Data communications |
| F3 | Distributed functions | F4 | Performance |
| F5 | Heavily used configuration | F6 | Online data entry |
| F7 | Operational ease | F8 | Online update |
| F9 | Complex interface | F10 | Complex processing |
| F11 | Reusability | F12 | Installation ease |
| F13 | Multiple sites | F14 | Facilitate change |

Alternatively, the following questionnaire could be utilized

1. Does the system require reliable backup and recovery?

2. Are data communications required?

3. Are there distributed processing functions?

4. Is performance critical?

5. Will the system run in an existing, heavily utilized operational environment?

6. Does the system require on-line data entry?

7. Does the on-line data entry require the input transaction to be build over multiple screens or operations?

8. Are the master files updated online?

9. Are the input, outputs, files or inquiries complex?

10. Is the internal processing complex?

11. Is the code designed to be reusable?

12. Are conversions and installation included in the design?

13. Is the system designed for multiple installations in different organizations?

14. Is the applications designed to facilitate change and ease of use?

Each component is rated from 0 to 5, where 0 means the component has no influence on the system and 5 means the component is essential (Pressman, 1997). The VAF can then be calculated as:

VAF = 0.65 + (Sum of GSCs x 0.01) Where Sum of GSCs = SUM(Fi)

The factor varies from 0.65 (if each Fi is set to 0) to 1.35 (if each Fi is set to 5) (Fenton, 1997). The final function point calculation is:

Final Adjusted FP = UFC x VAF

Convert AFP into SLOC using appropriate conversion factor.

SLOC = 16 x SLOC/AFP [NOTE: 16 is the conversion factor]

EFFORT = EAF x A x (SLOC)EX

EAF = CPLX x TOOL

A = 3.2= Constant based on the development mode.

EX = 0.38= Constant based on the development mode.

CPLX = 1.3 = Constant based on the development language.

TOOL = 1.1 = Constant based on the development Tool.

TDEV = 2.5 x (EFFORT) EX in months.

**PROGRAM/ CODE:**

//function point size estimation model

import java.util.\*;

class FP

{

int weight[][]={{3,4,5},{4,5,7},{3,4,6},{7,10,15},{5,7,10}};

Scanner sc=new Scanner(System.in);

int menu()

{

System.out.println("1.Low"+"\n"+"2.Average"+"\n"+"3.High");

System.out.println("Enter choice");

int n=sc.nextInt();

return(n);

}

int wfactor(int r,int h)

{

return(weight[r-1][h-1]);

}

void calc()

{

int v=0;

int sum=0;

System.out.println("Enter external input value");

v=sc.nextInt();

sum=sum+v\*wfactor(1,menu());

System.out.println("Enter external input value");

v=sc.nextInt();

sum=sum+v\*wfactor(1,menu());

System.out.println("Enter external output value");

v=sc.nextInt();

sum=sum+v\*wfactor(2,menu());

System.out.println("Enter external inquiries value");

v=sc.nextInt();

sum=sum+v\*wfactor(3,menu());

System.out.println("Enter internal logical file value");

v=sc.nextInt();

sum=sum+v\*wfactor(4,menu());

System.out.println("Enter internal logical file value");

v=sc.nextInt();

sum=sum+v\*wfactor(4,menu());

System.out.println("Enter external interface value");

v=sc.nextInt();

sum=sum+v\*wfactor(5,menu());

System.out.println("Enter high transaction rate");

int ra=sc.nextInt();

System.out.println("Enter multiple comm. protocol");

int cpr=sc.nextInt();

System.out.println("UFP= "+sum);

int TDI=ra+cpr;

double VAF=(TDI\*0.01)+0.65;

double AFP=sum\*VAF;

System.out.println("AFP= "+AFP);

}

public static void main(String sd[])

{

FP f=new FP();

f.calc();

}

}

### **RESULT/ OUTPUT:**

Enter external input value

10

1.Low

2.Average

3.High

Enter choice

1

Enter external input value

10

1.Low

2.Average

3.High

Enter choice

3

Enter external output value

15

1.Low

2.Average

3.High

Enter choice

2

Enter external inquiries value

13

1.Low

2.Average

3.High

Enter choice

2

Enter internal logical file value

2

1.Low

2.Average

3.High

Enter choice

3

Enter internal logical file value

2

1.Low

2.Average

3.High

Enter choice

1

Enter external interface value

7

1.Low

2.Average

3.High

Enter choice

2

Enter high transaction rate

5

Enter multiple comm. protocol

5

UFP= 300

AFP= 225.0

PROGRAM CODE:

//COCOMO MODEL

import java.util.\*;

class COCOMO

{

public static void main(String args[])

{

Scanner s=new Scanner(System.in);

System.out.println("Enter Size");

double si=s.nextDouble();

System.out.println("Enter Software Reliability");

double r=s.nextDouble();

System.out.println("Enter use of software tools");

double st=s.nextDouble();

System.out.println("Enter product complexity");

double c=s.nextDouble();

System.out.println("Enter Execution time complexity");

double ec=s.nextDouble();

double EAF=r\*st\*c\*ec;

double a=3.2;

double b=1.05;

double E=a\*Math.pow(si,b)\*EAF;

System.out.println("Value of organic project="+Math.ceil(E));

double a1=3.0;

double b1=1.12;

double E1=a1\*Math.pow(si,b1)\*EAF;

System.out.println("Value of Semidetach project="+Math.ceil(E1));

double a2=2.8;

double b2=1.2;

double E2=a2\*Math.pow(si,b2)\*EAF;

System.out.println("Value of Embedded project="+Math.ceil(E2));

}

}

### **RESULT /OUTPUT:**

Enter Size

200

Enter Software Reliability

1.15

Enter use of software tools

.91

Enter product complexity

.85

Enter Execution time complexity

1

Value of organic project=742.0

Value of Semidetach project=1008.0

Value of Embedded project=1438.0

### **Learning Objective:**

* To understand various cost estimation techniques and their applications.
* To learn how to estimate project costs systematically.
* To develop skills for budget planning in software projects.

### **Learning Outcome:**

At the end of this experiment, students will be able to:

1. Analyze the project requirements to determine cost estimation inputs.
2. Apply cost estimation techniques to predict project costs accurately.
3. Evaluate the impact of resource allocation and project complexity on cost estimation.

### **Course Outcomes (COs)**

* **CO3:** Identify and formulate functional and non-functional requirements for software systems and perform cost estimation using various models.
* **CO5:** Evaluate and mitigate software project risks and apply configuration management practices to maintain project integrity.

### **Cognitive Levels of Attainment as per Bloom’s Taxonomy**

* **L3 (Apply):** Apply cost estimation techniques to the project.
* **L5 (Evaluate):** Evaluate the accuracy and feasibility of cost estimation.

### **Programme Outcome (POs):**

* **PO2: Problem Analysis:** Analyze project requirements to identify cost-related factors.
* **PO3: Design/Development of Solutions:** Design and calculate cost estimation models for the project.
* **PO9: Communication**: Document and present cost estimation details effectively.
* **PO10: Project Management and Finance:** Manage project costs by applying engineering management principles.

### **Programme Specific Outcome (PSOs):**

* **PSO1:** Apply knowledge of cost estimation models to develop sustainable IT solutions.

### **Result & Discussion:**

* **Result:** Project cost estimation completed using the selected technique/model.
* **Discussion:**
  + The selected cost estimation technique provided detailed cost breakdowns for various project components.
  + The estimate highlighted key cost-driving factors, enabling resource optimization.

### **Conclusion:**

Cost estimation is a vital step in software project planning, ensuring proper allocation of resources and project success. Through this experiment, students gained practical experience in applying cost estimation techniques and understanding the factors affecting project costs.

### **Steps for the Experiment**

#### ****1. Understand Project Requirements****

1. Identify the size and scope of the project.
2. Define the functional and non-functional requirements clearly.

#### ****2. Choose a Cost Estimation Technique****

1. Select a suitable cost estimation technique such as COCOMO, FPA, or Expert Judgment.
2. Justify the choice of technique based on project characteristics.

#### ****3. Perform Cost Estimation****

1. Calculate project size (e.g., LOC, function points).
2. Use estimation formulas or tools based on the chosen technique.
3. Break down costs into subcomponents like development, testing, and deployment.

#### ****4. Document and Evaluate Results****

1. Create a detailed cost estimation report.
2. Verify the estimates against project requirements and constraints.
3. Suggest ways to optimize costs, if applicable.

### **Tools for Cost Estimation**

* COCOMO Calculator
* Function Point Analysis Tools
* Estimation spreadsheets or software (e.g., MS Excel)

**Experiment No. 8: RMMM Plan Development**

### **Aim:**

To develop a Risk Management, Mitigation, and Monitoring (RMMM) plan for your software project.

### **Theory:**

Risk Management in software engineering involves identifying potential risks, assessing their impact, and developing strategies to manage them effectively. A comprehensive **RMMM Plan** ensures that risks are identified early, mitigated proactively, and monitored throughout the software development lifecycle.

**Key Components of RMMM Plan:**

1. **Risk Identification**
   * Identify potential risks that may impact project objectives, such as technical risks, resource risks, and schedule risks.
2. **Risk Assessment**
   * Assess the probability of risk occurrence and its potential impact on the project using qualitative or quantitative methods.
   * Represent risks using a risk matrix (Probability vs. Impact).
3. **Risk Mitigation**
   * Develop strategies to minimize the likelihood of risks or reduce their impact.
   * Examples: Contingency planning, use of proven technologies, or resource reallocation.
4. **Risk Monitoring**
   * Continuously monitor identified risks and detect new risks throughout the project lifecycle.
   * Update the RMMM plan as required based on project progress.

**Benefits of RMMM Plan:**

* Reduces project delays and cost overruns.
* Enhances project stability and success.
* Improves team preparedness for unforeseen challenges.

### **Learning Objective:**

* To understand the principles of risk management in software projects.
* To identify, assess, mitigate, and monitor risks effectively.
* To develop a systematic approach to managing project risks.

### **Learning Outcome:**

At the end of this experiment, students will be able to:

1. Identify potential risks in their software project.
2. Formulate a detailed RMMM plan to manage risks.
3. Monitor risks and update the plan to ensure project stability.

### **Course Outcomes (COs):**

* **CO5:** Evaluate and mitigate software project risks and apply configuration management practices to maintain project integrity.

### **Cognitive Levels of Attainment as per Bloom’s Taxonomy:**

* **L3 (Apply):** Apply risk management principles to identify and mitigate project risks.
* **L5 (Evaluate):** Evaluate the effectiveness of the RMMM plan and suggest improvements.

### **Programme Outcome (POs):**

* **PO2: Problem Analysis:** Identify and analyze risks related to project requirements and constraints.
* **PO3: Design/Development of Solutions:** Design strategies to mitigate risks in software projects.
* **PO5: Engineering Tool Usage:** Utilize tools to monitor and document risks effectively.
* **PO10: Project Management and Finance:** Apply risk management principles to optimize project outcomes.

### **Programme Specific Outcome (PSO):**

* **PSO1:** Apply the culture of augmenting existing technologies to create sustainable IT solutions.
* **PSO2:** Integrate various technologies to address and mitigate risks in real-time project scenarios.

### **Result & Discussion:**

* **Result:** A detailed RMMM plan for the software project has been developed.
* **Discussion:**
  + Identified risks were categorized based on probability and impact.
  + Mitigation strategies were proposed and documented to address the top-priority risks.
  + The monitoring framework was established for continuous risk evaluation.

### **Conclusion:**

Developing an RMMM plan is crucial for ensuring project success. By completing this experiment, students gained hands-on experience in risk management, enhancing their ability to predict and manage challenges effectively in software engineering projects.

### **Steps for the Experiment**

#### ****1. Identify Project Risks****

1. Analyze the project scope, requirements, and resources to identify potential risks.
2. Classify risks into categories such as technical, resource, schedule, and operational risks.

#### ****2. Assess Risks****

1. Assign a probability (Low, Medium, High) and impact (Low, Medium, High) to each risk.
2. Create a risk matrix to prioritize risks based on their probability and impact.

#### ****3. Develop Mitigation Strategies****

1. For high-priority risks, propose actionable strategies to reduce probability or impact.
2. Document fallback plans for risks that cannot be mitigated entirely.

#### ****4. Establish Monitoring Framework****

1. Define metrics and tools for monitoring risks throughout the project lifecycle.
2. Assign responsibilities for risk tracking and reporting.

#### ****5. Prepare the RMMM Plan****

1. Compile all risk-related information into a structured RMMM document.
2. Include risk descriptions, assessment data, mitigation strategies, and monitoring plans.

### **Tools for Risk Management:**

* Risk Matrix Templates
* MS Excel or Google Sheets for tracking risks
* Project Management Tools (e.g., Jira, Trello)

### **Example: Hotel Management System (HMS)**

A Hotel Management System involves various modules like room booking, customer management, billing, and inventory management. Developing an RMMM plan for this system involves considering risks in areas such as:

Technical Risks: Software architecture flaws, database failures, or integration issues.

Operational Risks: System downtime, poor user experience, or inadequate security leading to data breaches.

Management Risks: Delays in project delivery, budget overruns, or miscommunication between stakeholders.

Procedure:

Risk Identification:

Conduct brainstorming sessions with the development team to identify potential risks.

Review project documents such as requirements, design, and use cases to foresee possible risk areas.

Example Risks for HMS:

Inaccurate billing due to wrong data entry.

Database crashes or slowdowns due to poor optimization.

Failure in integrating with third-party systems (e.g., payment gateways).

Risk Analysis:

For each identified risk, assess its likelihood (high, medium, low) and its potential impact (severe, moderate, minor).

Example Analysis for HMS:

Inaccurate billing: Likelihood = High, Impact = Severe.

Database crashes: Likelihood = Medium, Impact = High.

Third-party integration failure: Likelihood = Low, Impact = Moderate.

Risk Mitigation:

For each risk, develop strategies to reduce its impact or likelihood.

Example Mitigation Strategies for HMS:

Inaccurate billing: Implement validation checks for input data and perform testing on the billing module.

Database crashes: Use a robust, scalable database design; implement database backup and recovery strategies.

Third-party integration failure: Use well-documented APIs and perform mock integration tests.

Risk Monitoring:

Set up monitoring to track the identified risks. Use tools like JIRA or Trello to log risks and track their status.

Regularly review the risks during sprint meetings or progress reviews.

Risk Management:

Continuously update the RMMM plan as new risks arise or existing risks evolve. Update the risk register with new information.

Assign owners for each risk who are responsible for ensuring mitigation actions are executed.

Risk Mitigation, Monitoring, and Management Plan for Hotel Management System (HMS)

### **Sample RMMM Plan Template:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Risk ID** | **Risk Description** | **Probability** | **Impact** | **Mitigation Strategy** | **Monitoring Plan** |
| R1 | Resource unavailability | High | High | Hire additional resources proactively. | Weekly team resource review. |
| R2 | Requirement changes | Medium | High | Allocate buffer time for requirement changes. | Regular client meetings. |
| R3 | Technology unfamiliarity | High | Medium | Conduct training sessions for team members. | Monthly skill evaluation. |

#### ****Experiment No. 9: GUI Design and Implementation of Mini Project****

### **Aim:**

To design and implement a Graphical User Interface (GUI) for the mini-project using appropriate tools and techniques, ensuring usability and functionality.

### **Theory:**

Graphical User Interface (GUI) design is a crucial aspect of software engineering that focuses on creating user-friendly and visually appealing interfaces. A well-designed GUI enhances user experience, ensures accessibility, and improves the overall usability of the system.

**Key Concepts in GUI Design:**

1. **User-Centered Design:**
   * Focuses on the users’ needs, goals, and preferences.
   * Involves iterative prototyping and feedback to refine the interface.
2. **Principles of Good GUI Design:**
   * **Consistency:** Uniformity in design elements, such as colors, fonts, and layout.
   * **Feedback:** Providing visual or auditory responses to user actions.
   * **Visibility:** Ensuring critical elements are easy to find and accessible.
   * **Error Prevention:** Designing interfaces to reduce the possibility of user errors.
3. **GUI Development Tools:**
   * Visual Studio
   * JavaFX
   * PyQt
   * Figma, Adobe XD (for prototyping)

**Process of GUI Design and Implementation:**

1. Requirement analysis.
2. Wireframe or mockup creation.
3. GUI design using selected tools.
4. Integration of functionality with back-end logic.

### **Learning Objectives:**

* To understand the principles of GUI design.
* To implement a GUI for a mini-project that is functional and user-friendly.
* To integrate the GUI with the underlying system functionalities.

### **Learning Outcomes:**

At the end of this experiment, students will be able to:

1. Design a functional GUI adhering to usability principles.
2. Implement a GUI using appropriate development tools.
3. Integrate the GUI with the project's back-end.

### **Course Outcomes (COs):**

* **CO6:** Design and develop user-centric software systems with effective GUI.

### **Cognitive Levels of Attainment as per Bloom’s Taxonomy:**

* **L3 (Apply):** Apply GUI design principles to create user-friendly interfaces.
* **L6 (Create):** Design and implement a complete GUI for a mini-project.

### **Programme Outcome (POs):**

* **PO3: Design/Development of Solutions:** Develop systems with well-structured GUI solutions.
* **PO5: Modern Tool Usage:** Utilize advanced tools and technologies for GUI design and development.
* **PO8: Ethics:** Ensure GUI designs are inclusive and accessible to all users.
* **PO10: Project Management and Finance:** Manage GUI design as part of the overall project timeline and resources.

### **Programme Specific Outcome (PSOs):**

* **PSO2:** Integrate various technologies to develop scalable and effective solutions.
* **PSO3:** Design, develop, and test software with a focus on user interaction and usability.

### **Result & Discussion:**

* **Result:** A functional GUI was designed and implemented for the mini-project.
* **Discussion:**
  + The GUI adheres to the principles of usability and accessibility.
  + The interface is consistent with the project requirements and integrates seamlessly with the back-end.

### **Conclusion:**

GUI design is an integral part of software development, directly impacting the usability and success of the application. Through this experiment, students gained practical experience in designing and implementing a GUI for a mini-project using modern tools and adhering to design principles.

### **Sample write up of Implementation of mini project: Hotel management System:**

Hotel Management System aims at automation of processes related to Hotel Management. The project consists of following parameter:

For data security, there are two level of authentications. All the fields can be accessed by high authorities whereas fields containing data related to rooms, reservation, payment and customer can be operated by low level management like operators, staff, manager, attendance, salary can be operated by high level authority.

After login, depending upon the purpose user can access forms like –

Customer form to fill customer details

Room form to fill room detail

Reservation form to fill reservation detail

Payment form to fill payment related details.

Rest forms accessed by fop level management are –

Manager form to add details about new manager and show record.

Attendance form to put daily attendance on the basis of staff\_id

Staff form to add a new staff and display record of all the staff working.

Inventory form to add order of modify any item details.

There is additional form bill to print bill after all the transaction related to room reservation are over.

Report –

Reports provided by this system are as listed –

* 1. Bill
  2. Attendance record monthly basis
  3. Payment record monthly basis
  4. Inventory record monthly basis
  5. Staff and their salary record
  6. reservation details and room availability record
  7. Items left, ordered, etc. sheet.

Bill –

Bill basically consist of customer name, room number, type of service chosen, reservation number. This bill will be given to the customer after payment.

Stocksheet –

This sheet has items code, type, quality present and if ordered during than span quantity. For providing mere flexibility, for few there in three different sheets, one displayed on monthly basis showing all record.

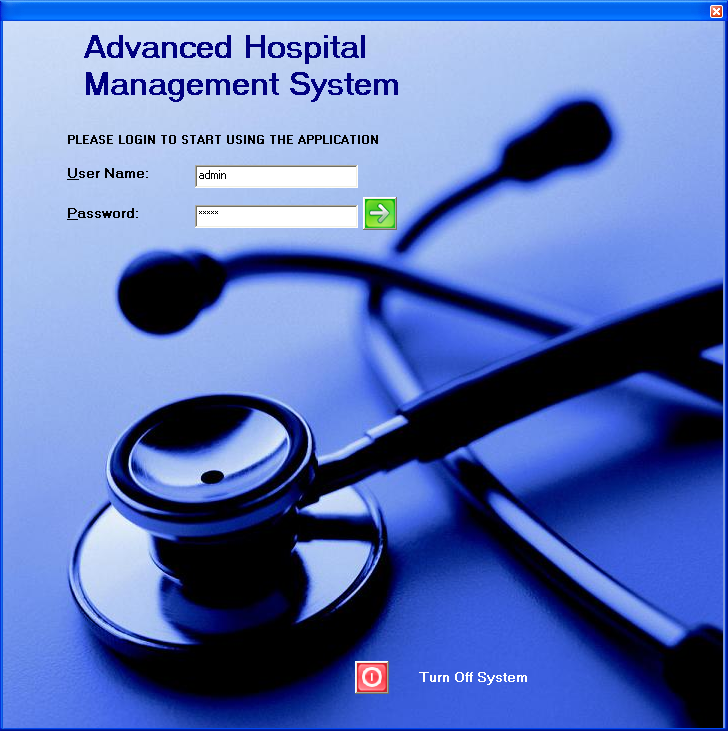
Payment Record –

This shows which all payments have been received this month or week.

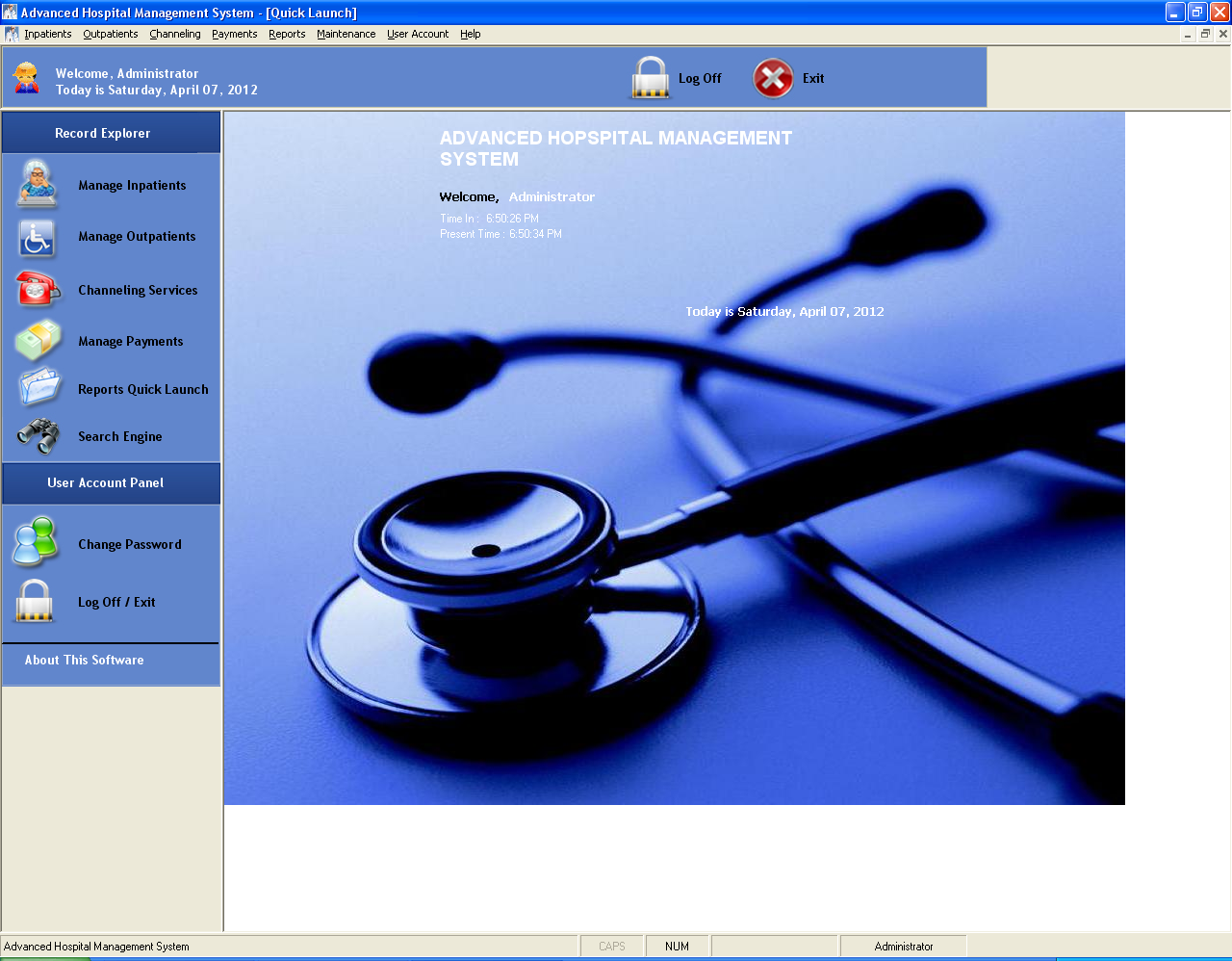
Attendance Sheet –

This sheet can be accessed by manager and he keeps record of attendance as well as salary of staff.

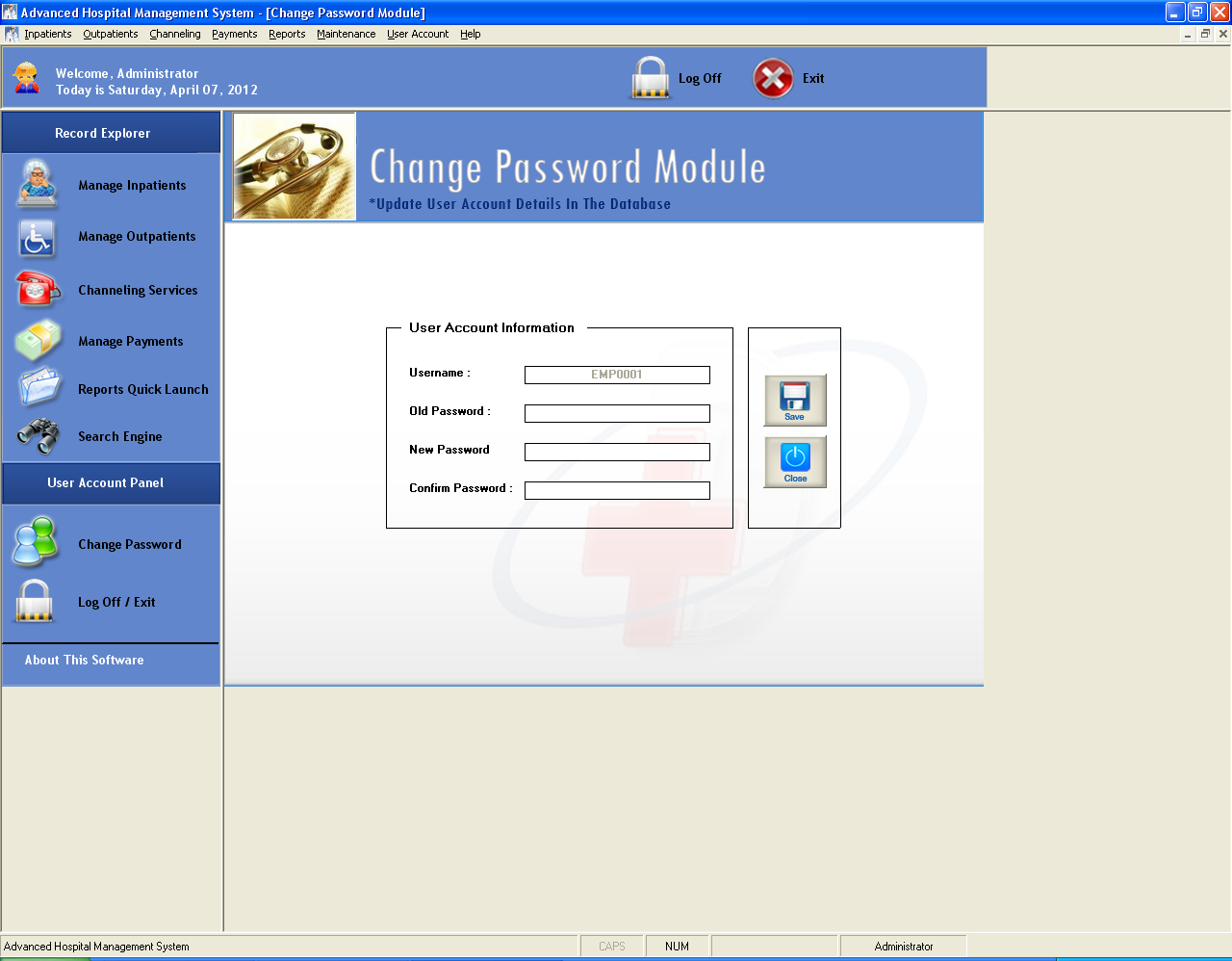
1.Login into Main System



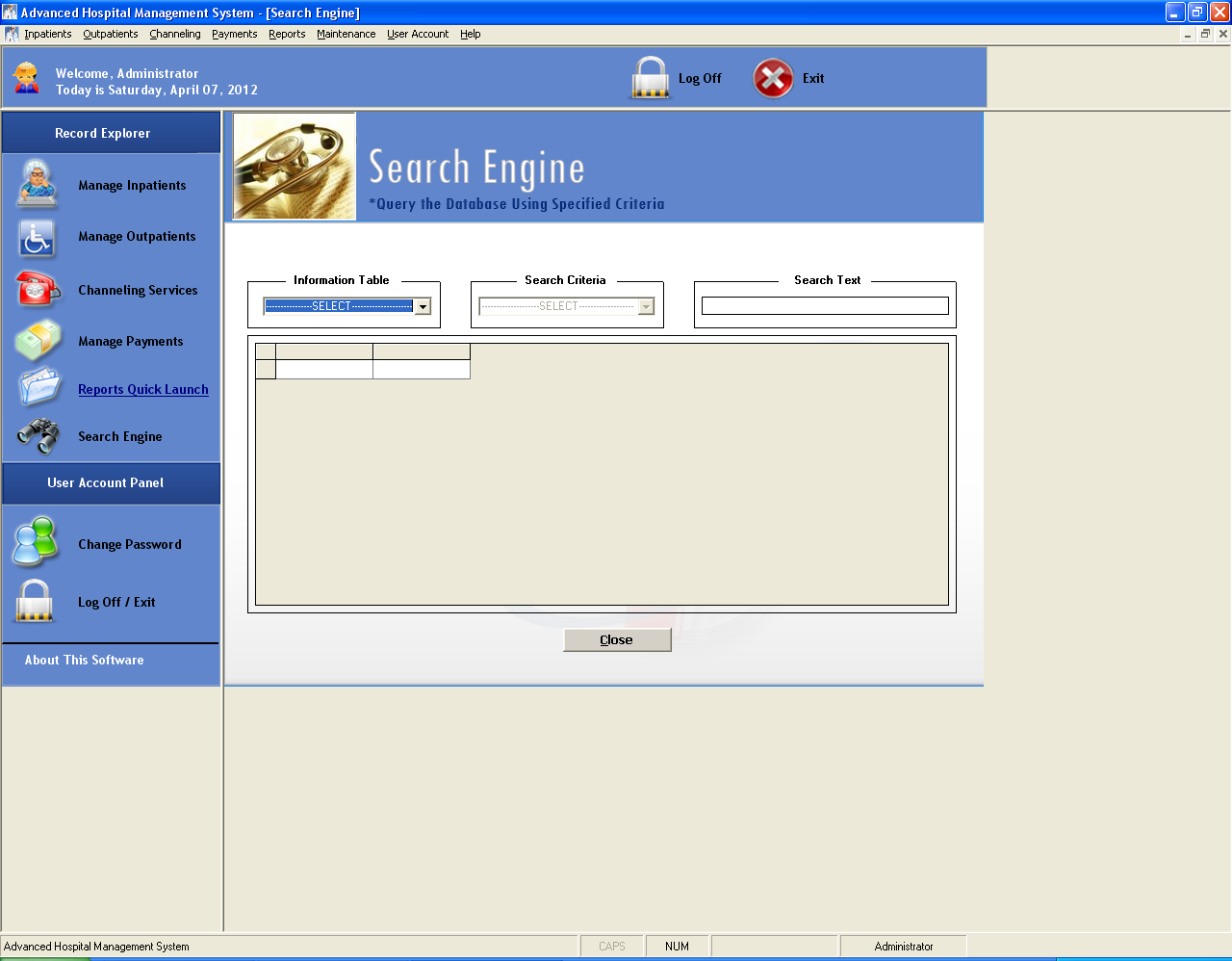
2. Startup after Login



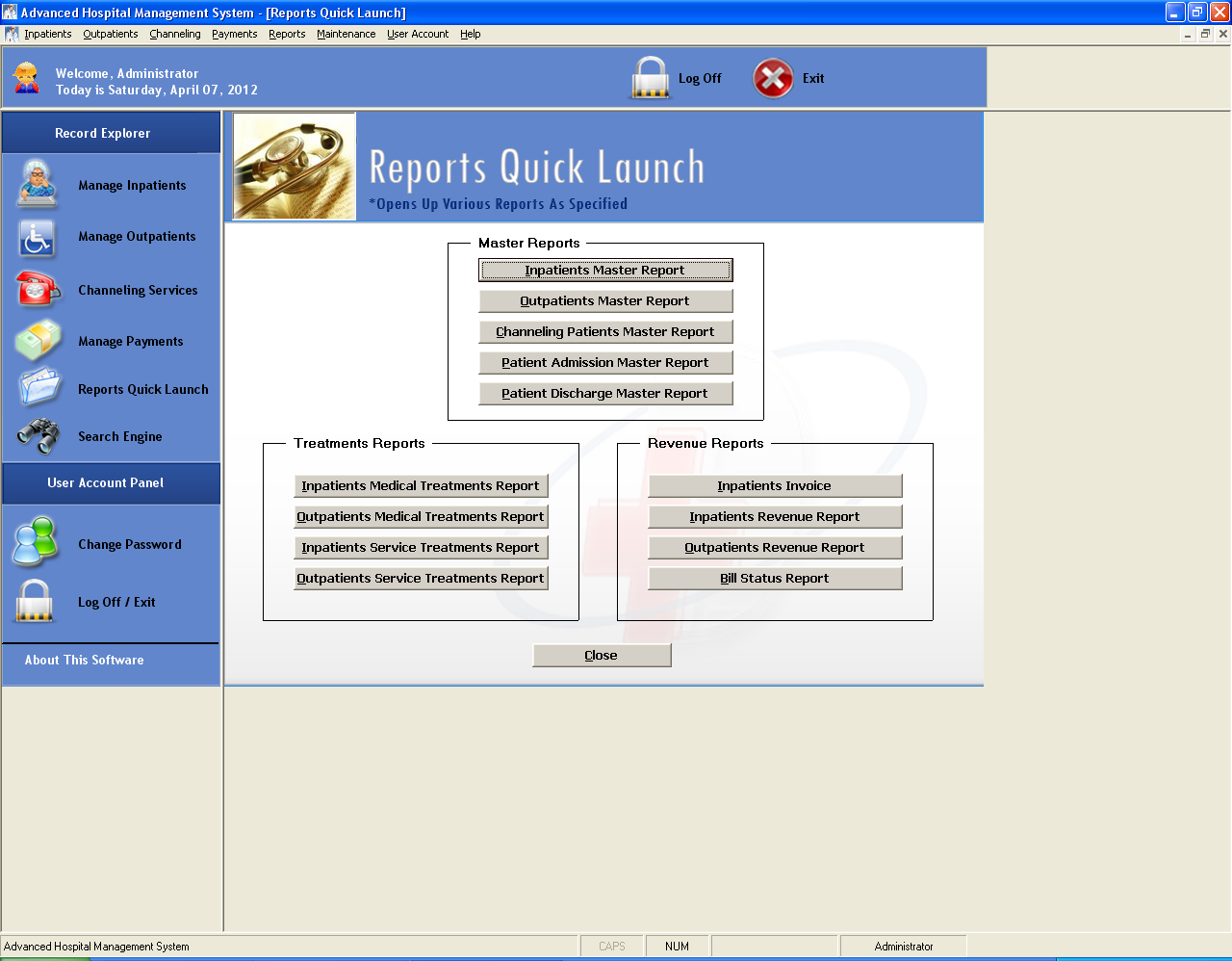
3. Change password



4. Search engine

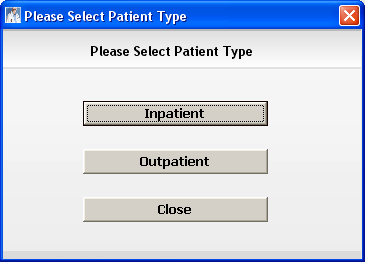


5. Reports quick launch

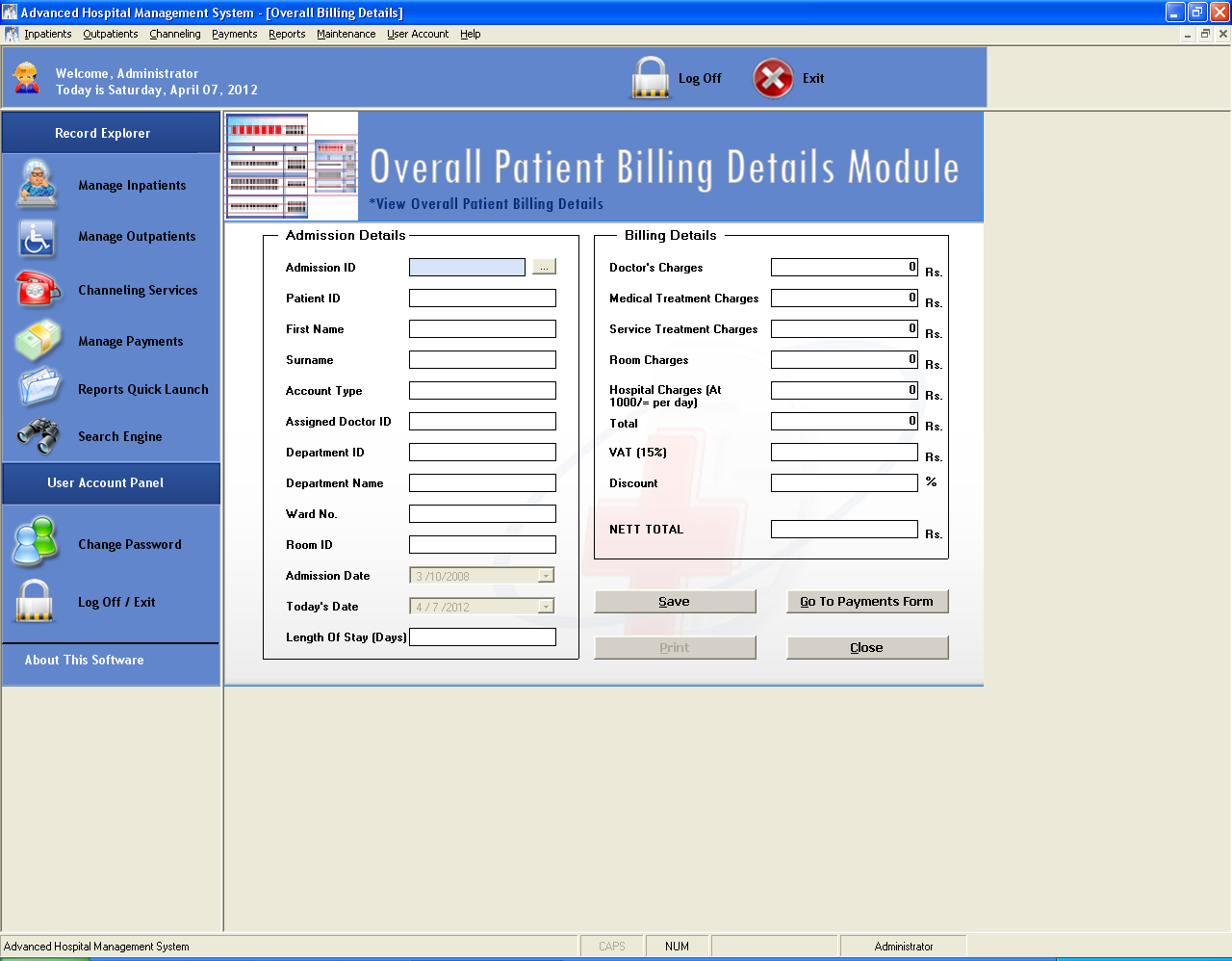


6. Manage payments

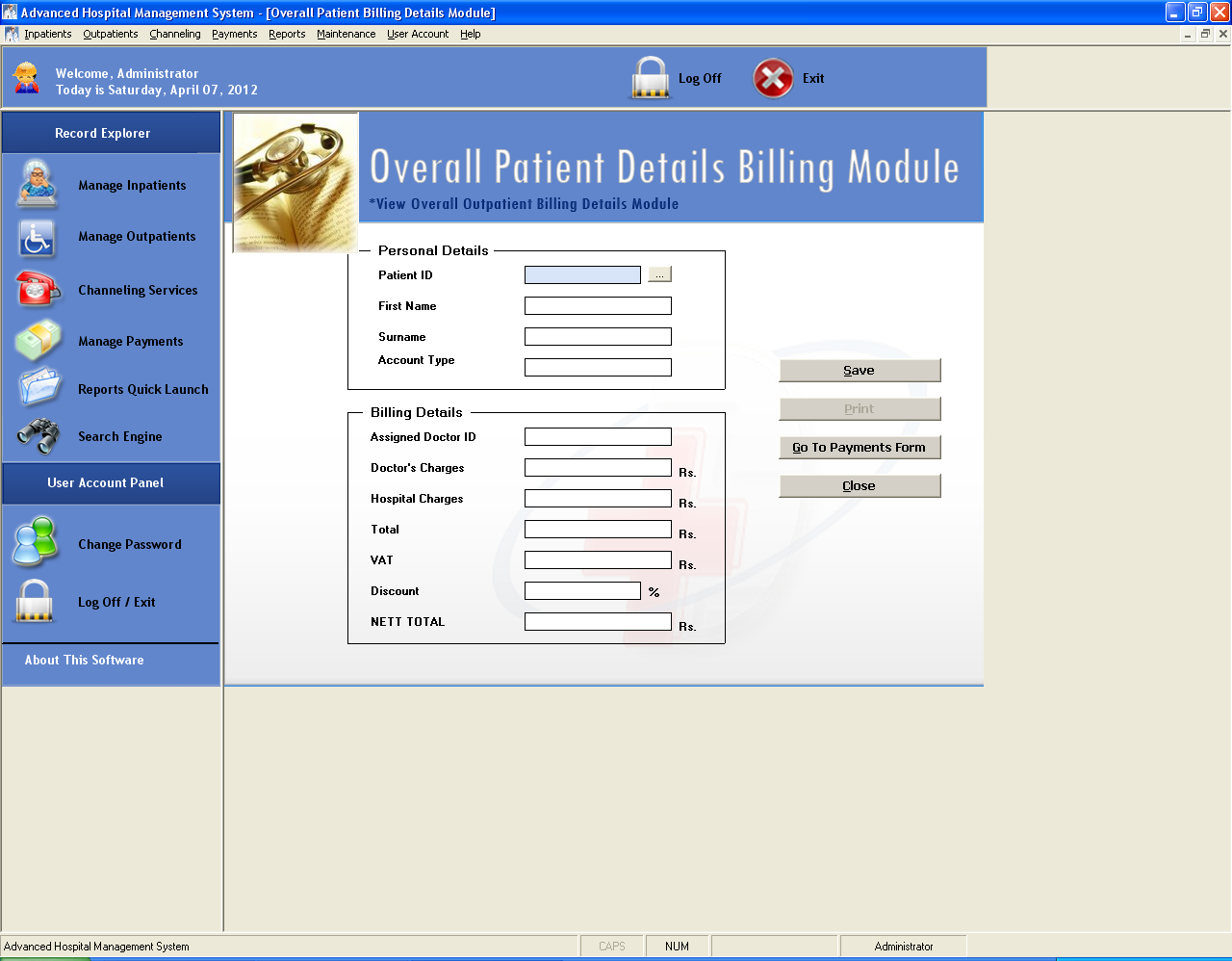
1. Type of patient



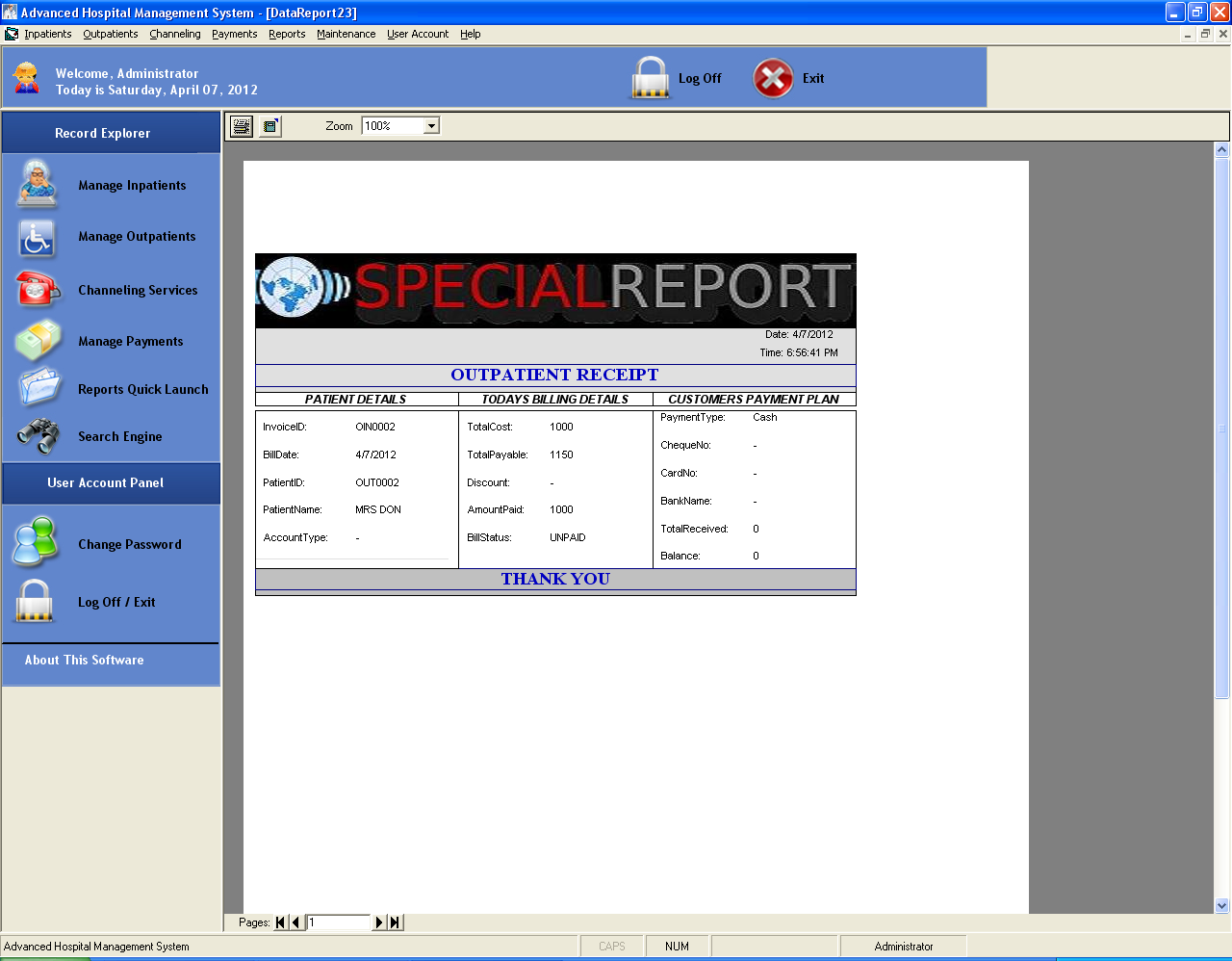
2. Inpatient



3. out Patient



3.1.1 Payment form (print copy)

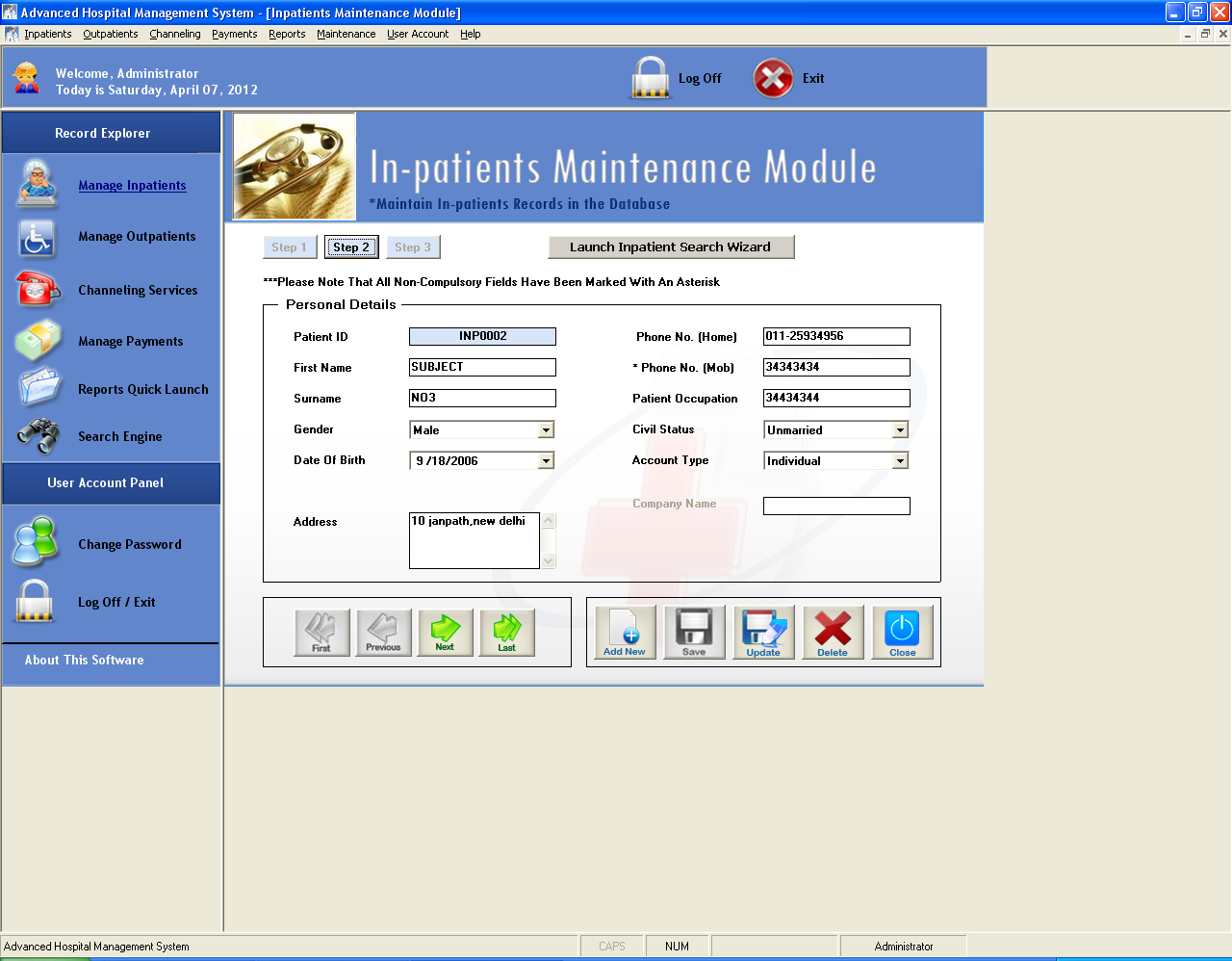


7. Maintaining appointments

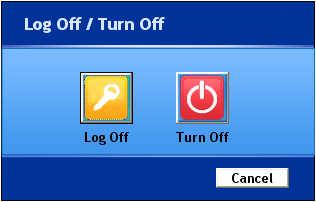
1. Manage outpatients



2. Manage inpatients



8. Log off /exit



**Experiment No. 10: Develop Test Cases for Testing Mini Project**

### **Aim:**

To develop test cases for the mini-project, ensuring comprehensive testing of all functionalities and verifying adherence to project requirements.

### **Theory:**

Testing is a critical phase in software development, aimed at identifying and rectifying errors in the system to ensure it performs as intended. Test cases are documented scenarios that validate the functionality of the software by specifying inputs, execution steps, and expected outcomes.

**Types of Testing:**

1. **Unit Testing:** Testing individual components or modules.
2. **Integration Testing:** Verifying the interactions between different modules.
3. **System Testing:** Ensuring the complete system functions correctly.
4. **Acceptance Testing:** Validating the software against user requirements.

**Key Components of a Test Case:**

1. **Test Case ID:** A unique identifier for the test case.
2. **Test Objective:** The purpose of the test case.
3. **Preconditions:** Conditions that must be met before executing the test.
4. **Test Steps:** Detailed steps to execute the test.
5. **Expected Results:** The desired outcome of the test.
6. **Actual Results:** The outcome observed during execution.
7. **Status:** Indicates whether the test passed or failed.

**Common Testing Tools:**

* Selenium (for automated testing)
* JUnit (for unit testing in Java)
* Postman (for API testing)
* TestRail (for test case management)

### **Learning Objectives:**

* To learn how to write test cases systematically.
* To validate the mini-project's functionality and identify potential issues.
* To document testing results effectively.

### **Learning Outcomes:**

At the end of this experiment, students will be able to:

1. Develop comprehensive test cases for a software project.
2. Perform functional and non-functional testing of their mini-project.
3. Analyze testing results to improve software quality.

### **Course Outcomes (COs):**

* **CO6:** Test and validate software systems to ensure quality and functionality.

### **Cognitive Levels of Attainment as per Bloom’s Taxonomy:**

* **L3 (Apply):** Develop test cases for specific project functionalities.
* **L4 (Analyze):** Evaluate testing results to identify issues and improvements.

### **Programme Outcome (POs):**

* **PO3: Design/Development of Solutions:** Develop robust test cases for project validation.
* **PO5: Modern Tool Usage:** Utilize tools and techniques for testing.
* **PO8: Ethics:** Maintain integrity and accuracy in testing practices.
* **PO10: Project Management and Finance:** Efficiently plan and execute the testing process within project timelines.

### **Programme Specific Outcome (PSO):**

* **PSO1:** Apply testing techniques to verify the correctness of software solutions.
* **PSO3:** Test and validate software solutions to meet societal and user requirements.

### **Result & Discussion:**

* **Result:** Test cases were successfully created and executed, ensuring the mini-project met the desired requirements.
* **Discussion:**
  + Test results highlighted areas for improvement.
  + Issues identified during testing were documented for resolution.

### **Conclusion:**

Through this experiment, students learned how to design, document, and execute test cases systematically. They also gained practical experience in identifying and resolving software issues, which is essential for delivering quality software solutions.

### **Steps for the Experiment**

#### ****1. Analyze the Mini-Project Requirements:****

1. Review the software requirements specification (SRS) to understand functional and non-functional requirements.
2. Identify key modules and features to be tested.

#### ****2. Develop Test Cases:****

1. Write test cases for each module using a standard test case template.
2. Ensure test cases cover all functionalities, including edge cases and error conditions.
3. Categorize test cases into types, such as functional, integration, or system tests.

#### ****3. Execute Test Cases:****

1. Follow the test steps and document the actual results.
2. Compare actual results with expected outcomes to determine the status (Pass/Fail).

#### ****4. Document Test Results:****

1. Record the outcomes of all test cases in a test case report.
2. Highlight failed test cases for further analysis and debugging.

### **Test Case Template:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Objective** | **Preconditions** | **Test Steps** | **Expected Results** | **Actual Results** | **Status** |
| TC001 | Validate login functionality | User credentials exist | Enter valid username and password | Login successful, navigate to dashboard | Same as expected | Pass |
| TC002 | Test invalid login | User credentials exist | Enter invalid username and password | Display error message | Same as expected | Pass |
| TC003 | Test edge case for input fields | N/A | Leave fields blank and submit | Display validation error | Login page does not proceed | Pass |

### **Tools and Technologies:**

* **Testing Tools:** Selenium, JUnit, TestNG, Postman
* **Languages Used for Testing:** Python, Java, or JavaScript

**Experiment No. 11: Case Study on Software Configuration Management (SCM)**

### **Aim:**

To study and analyze the principles, processes, and tools of software configuration management (SCM) by exploring a case study.

### **Theory:**

Software Configuration Management (SCM) is a critical process in software engineering that ensures consistency, traceability, and control over software components and artifacts during development. It includes managing changes, maintaining integrity, and preventing issues such as version conflicts.

**Key Concepts of SCM:**

1. **Configuration Items (CIs):** Files, code modules, documents, or artifacts under version control.
2. **Version Control:** Managing multiple versions of files or code. Examples: Git, SVN.
3. **Change Management:** Recording and controlling modifications in the project.
4. **Build Management:** Automating the creation of executable software from source code.
5. **Release Management:** Delivering the final software to stakeholders.

**Importance of SCM:**

* Maintains consistency and integrity across project components.
* Reduces development time by providing structured processes.
* Facilitates team collaboration by tracking and merging changes.
* Enhances software quality through better traceability and control.

**Popular SCM Tools:**

* **Git:** Distributed version control for tracking source code changes.
* **SVN:** Centralized version control system.
* **Jenkins:** Automation tool for continuous integration and delivery.
* **Docker:** Manages containerized environments.

### **Learning Objectives:**

1. To understand the importance of SCM in software projects.
2. To explore tools and techniques used in SCM.
3. To analyze a real-world case study on SCM and its implementation.

### **Learning Outcomes:**

At the end of this experiment, students will be able to:

1. Describe the processes and components of software configuration management.
2. Analyze the application of SCM in a real-world scenario.
3. Evaluate the effectiveness of SCM tools and techniques.

### **Course Outcomes (COs):**

* **CO5:** Demonstrate an understanding of modern tools and processes in software engineering, including SCM.

### **Cognitive Levels of Attainment as per Bloom’s Taxonomy:**

* **L2 (Understand):** Explain the processes and benefits of SCM.
* **L4 (Analyze):** Evaluate SCM implementation in a real-world scenario.
* **L5 (Evaluate):** Assess the effectiveness of SCM tools and practices.

### **Programme Outcome (POs):**

* **PO1:** Apply knowledge of engineering fundamentals to software management practices.
* **PO5: Use modern tools and techniques for software development and management.**
* **PO6: Analyze societal and environmental impacts of software practices.**
* **PO7: Understand the sustainability of software systems through effective management.**
* **PO11: Apply project management principles to SCM.**

### **Programme Specific Outcome (PSOs):**

* **PSO1:** Utilize configuration management tools to ensure consistency in software solutions.

### **Result & Discussion:**

* **Result:** A case study on SCM was analyzed to identify best practices, challenges, and outcomes in the implementation process.
* **Discussion:**
  + Insights into the importance of version control and change management were gained.
  + The selected SCM tool's impact on software development and team collaboration was discussed.

### **Conclusion:**

This experiment provided an understanding of SCM processes and tools. By analyzing a case study, students learned how SCM ensures project integrity, reduces risks, and improves software quality.

### **Steps for the Experiment**

#### ****1. Select a Case Study on SCM:****

* Choose a real-world example where SCM was implemented effectively.
* Example case studies:
  + SCM practices in an open-source project using Git.
  + SCM in a corporate environment using Jenkins for continuous integration.

#### ****2. Analyze the Case Study:****

* Identify the key components of SCM implemented in the project:
  + Version control, change management, build automation, release management.
* Understand the challenges faced and solutions applied in the project.
* Evaluate the impact of SCM tools on the development process.

#### ****3. Document the Findings:****

* Write a detailed report covering the following sections:
  + **Introduction to the project**
  + **SCM processes implemented**
  + **Tools and technologies used**
  + **Challenges and resolutions**
  + **Outcomes and lessons learned**

### **Example Questions for Case Study Analysis**

1. **Describe the role of version control in the selected case study.**
2. **What SCM tools were used, and why were they chosen?**
3. **What were the key challenges faced during SCM implementation?**
4. **Evaluate the effectiveness of the SCM process in ensuring project success.**

### **Sample SCM Process in Git**

**Steps for SCM using Git:**

1. Initialize a repository: git init
2. Add files to version control: git add
3. Commit changes: git commit -m "Initial commit"
4. Create branches for feature development: git branch feature\_x
5. Merge branches after testing: git merge feature\_x
6. Use tags for releases: git tag v1.0