A Laboratory Project Report

On

Test Case Management

Submitted

to

CMR Technical Campus,
Hyderabad

In Partial fulfilment for the requirement of the Award of the Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

By

SHIVA KUMAR

(227R1A05J3)

Under the esteemed

guidance of

Ms. G.Lavanya

(Assistant Professor)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CMR TECHNICAL CAMPUS

An UGC Autonomous Institute

Accredited by NBA & NAAC with A Grade

Approved by AICTE, New Delhi and JNTUH Hyderabad

Kandlakoya(V), Medchal Road, Hyderabad – 501401

CERTIFICATE

This to certify that, the Presentation entitled "**TEST CASE MANAGEMENT**" is submitted by **"SHIVA KUMAR"** bearing the Roll Number **227R1A05J3** of B.Tech Computer Science and Engineering. In Partial fulfilment for the requirement of the Presentation and for the award of the Degree of Bachelor of Technology during the academic year 2024-25.

Subject Faculty

Ms. G. Lavanya

(Assistant Professor)

CMR TECHNICAL CAMPUS

UGC AUTONOMOUS

Accredited by NBA & NAAC with 'A' Grade

Approved by AICTE, New Delhi and JNTU Hyderabad

Academic Year	: 2024-2025
Name of the Student	: Shiva Kumar
Roll No	:227R1A05J3
Year	: B. Tech III
Semester	: II
Section	: C
Branch	: COMPUTER SCIENCE AND ENGINEERING
Name of the Laboratory	: SOFTWARE TESTING METHODOLOGIES
Batch No.	: 16
Title of the Lab Report/Project	:Test Case Management
Date	:
Signature of the Student	

LABORATORY REPORT/PROJECT & PRESENTATION				
Problem Statement & Objectives	Design & Methodology	Implementation & Results	Total Marks	Final Marks
10	15	15	40	10

Remarks/Comments by the Faculty:

Name of the Faculty : Ms. G. Lavanya

Signature of the Faculty :

Department of CSE

Institute Vision:

To Impart quality education in serene atmosphere thus strive for excellence in Technology and Research.

Institute Mission:

1. To Create state of art facilities for effective Teaching- Learning Process.
2. Pursue and Disseminate Knowledge based research to meet the needs of Industry &society
3. Infuse Professional, Ethical and Societal values among Learning Community.

Department Vision:

To Provide quality education and a conducive learning environment in computer engineering that foster critical thinking, creativity, and practical problem-solving skills.

Department Mission:

1. To educate the students in fundamental principles of computing and induce the skills needed to solve practical problems.
2. To provide State-of-the-art computing laboratory facilities to promote industry institute interaction to enhance student's practical knowledge.
3. To Inculcate self-learning abilities, team spirit, and professional ethics among the students to serve society

Table of Contents :

1. Abstract :

In the domain of software development, ensuring code quality through systematic testing is a critical phase of the Software Development Life Cycle (SDLC). This project presents a comprehensive **Test Case Management System**, designed to streamline the creation, execution, and analysis of test cases. The system integrates a code editor, a dynamic test case execution engine, and real-time feedback mechanisms to enhance testing accuracy and developer productivity.

Our tool facilitates both manual and automated test case handling, covering a wide range of scenarios such as input validation, boundary testing, and error handling. By incorporating visual indicators and execution metrics, the platform provides clear insights into test coverage and performance. It also supports modification and organization of test cases, promoting flexibility and maintainability.

The solution is particularly tailored for educational and development environments where iterative testing, immediate feedback, and code refinement are essential. The project emphasizes quality metrics such as pass/fail ratios, execution time, and error detection rates to assess testing effectiveness. Through this system, we aim to foster a structured approach to testing that ensures reliable, robust, and efficient software delivery.

Moreover, the system encourages collaboration by allowing multiple users to contribute test cases and review results in real time. It promotes best practices in software testing, offering developers a consistent and repeatable framework. The ability to adapt to various coding problems makes the tool suitable for both beginners and experienced developers. Enhanced visualization of test outcomes aids in debugging and accelerates the development cycle. Overall, this project demonstrates the value of integrating testing tightly within the software development process.

2. Introduction:

Software testing plays a vital role in ensuring the reliability, functionality, and overall quality of software applications. As modern software becomes increasingly complex, the need for efficient, systematic, and scalable testing methodologies has become more important than ever. To address these challenges, this project introduces a **Test Case Management System**a tool specifically designed to support the entire testing lifecycle from test case creation to result analysis.

The core objective of the system is to provide a platform where developers can easily write, manage, and execute test cases for various coding problems or software modules. It enables users to define clear input-output expectations, run code against test cases, and immediately view results along with visual feedback. This not only speeds up the development process but also ensures that the code meets defined functional requirements.

The system is particularly useful in academic environments, coding platforms, and development teams where test-driven development (TDD) and continuous feedback loops are encouraged. With built-in features such as manual test case entry, categorized test scenarios (normal, edge, boundary), and real-time status updates, the tool helps users detect bugs early and refine their code iteratively.

Furthermore, by incorporating essential quality metrics such as test coverage, pass/fail ratio, and error detection rates, the system provides valuable insights into software performance. These metrics aid in decision-making, highlight areas of improvement, and contribute to overall code quality and project success.

In essence, this project aims to empower developers and learners with a practical solution that integrates core software testing principles into an interactive and user-friendly platform.

3. Literature Survey:

A literature survey is essential to understand the current advancements, practices, and challenges in the domain of software testing and test case management. It provides insight into existing tools, methodologies, and research, thereby laying the groundwork for designing an improved solution. Below is a summary of relevant studies and systems reviewed during the planning of this project:

3.1 Traditional Testing Approaches

Manual testing has long been a standard in software validation. While it provides flexibility, it is often time-consuming and error-prone. Traditional methods rely heavily on testers' intuition and documentation, making the process inconsistent across teams. These limitations have led to the development of automated and semi-automated solutions that standardize and speed up testing.

3.2 Automated Testing Tools

Modern tools like **JUnit**, **Selenium**, and **TestNG** have made automation more accessible. These tools support the creation of repeatable test scripts and execution of test suites. However, they often require considerable setup and may lack a centralized interface for managing custom test cases and viewing detailed feedback, especially for learners or developers working on problem-solving tasks rather than large applications.

3.3 Test Case Management Systems

Platforms like **TestRail**, **Zephyr**, and **qTest** offer extensive test management features, including test planning, case organization, execution tracking, and reporting. While powerful, these systems are primarily enterprise-focused, with complex interfaces and limited support for small-scale educational or coding-focused environments.

4. Analysis and Design:

4.1 Problem Analysis

Software testing is often under-emphasized in early-stage development. Manual testing processes can be repetitive, lack structure, and result in overlooked edge cases. Moreover, existing platforms provide limited flexibility in customizing test cases or lack real-time feedback features, which are essential for learning and agile development.

4.2 Proposed Solution

Our system addresses these issues by offering:

A built-in **code editor** for implementing problem solutions.

A **test case interface** for manually adding, modifying, or deleting test cases.

An **execution engine** that runs user code against the defined test cases.

Real-time visual feedback showing the status (Passed/Failed/Pending) of each test.

4.3 System Architecture

Key Components:

1.Frontend Interface:

- Code Editor
- Test Case Manager
- Results Display Panel

2.Backend Engine:

- Code Execution Module
- Test Case Validator
- Metrics Calculator

4.4 Use Case Diagram

Use Cases:

- Create/Edit/Delete Test Cases
- Implement Solution Code
- Run Test Cases
- View Execution Result

5. Implementation

The implementation phase focuses on transforming the system design into a fully functional application. It involves developing key modules, integrating them seamlessly, and ensuring the system performs as expected. Below is a detailed explanation of how the **Test Case Management System** was implemented.

5.1 Technology Stack

- Frontend: HTML, CSS, JavaScript,React
- Backend: Node.js or equivalent server-side environment
- Database: JSON files or a lightweight NoSQL database for test case storage
- Code Execution: In-browser JavaScript execution or a secure backend interpreter
- Tools Used: Visual Studio Code, Git, Postman (for testing APIs)

5.2 Functional Modules

- **Code Editor Module**
  A user interface where users can write and submit code. It supports basic syntax formatting and helps test the written code with inputs.
- **Test Case Management Interface**
  Allows users to add, remove, or edit test cases. Each test case includes an ID, input, expected output, and status (pending, passed, failed).
- **Execution Engine**
  This module takes the written code and runs it against all defined test cases. It then compares the actual and expected outputs to determine the test result.
- **Results Visualization**
  Displays clear feedback on test execution with indicators such as Passed, Failed, or Pending. It also shows summary metrics like total tests passed, failed, and overall performance.
- **Real-time Feedback System**
  Provides immediate feedback after test execution, helping users identify errors and refine their code more efficiently.

5.3 Sample Integration

- A basic problem like "Sum Two Numbers" is used for testing.
- The system allows defining inputs and expected outputs manually.
- When code is run, each test case is evaluated and the results are shown instantly.

5.4 User Interaction Flow

- The user selects or creates a coding problem.
- The user defines one or more test cases.

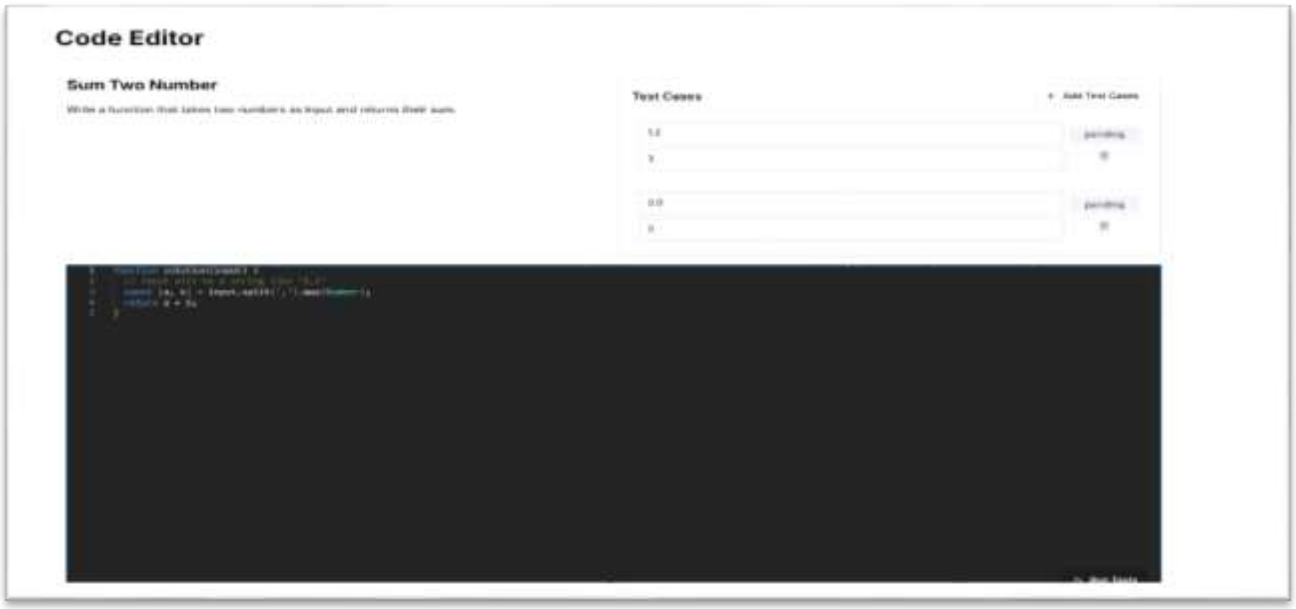
- The user writes their solution in the editor provided.
- On clicking "Run Tests", the system evaluates the solution.
- Feedback and performance metrics are displayed immediately.
- The user can refine their solution based on results and rerun the tests.

SOURCE CODE

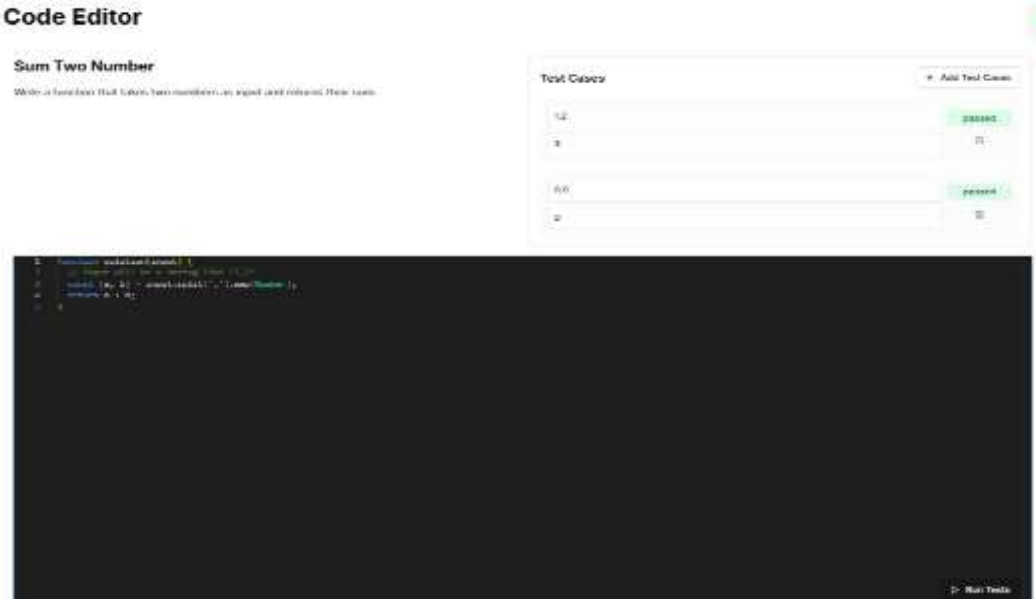
```
import CodeEditor from './components/CodeEditor';

const sampleProblem = {
  id: '1',
  title: 'Sum Two Number',
  description: 'Write a function that takes two numbers as input and returns their sum.',
  defaultCode: `function solution(input) {
  // input will be a string like "1,2"
  const [a, b] = input.split(',').map(Number);
  return a + b;
}`,
  testCases: [
    {
      id: 'tc1',
      input: '1,2',
      expectedOutput: '3',
      status: 'pending',
    },
    {
      id: 'tc2',
      input: '0,0',
      expectedOutput: '0',
      status: 'pending',
    },
  ],
};

export default function Home() {
  return (
    <main className="min-h-screen bg-background">
      <div className="container mx-auto py-8">
        <h1 className="text-4xl font-bold mb-8">Code Editor</h1>
        <CodeEditor problem={sampleProblem} />
      </div>
    </main>
  );
}
```

```tsx
'use client';
import { useState } from 'react';
import { Button } from '@/components/ui/button';
import { Card } from '@/components/ui/card';
import { Play, Plus, Trash2 } from 'lucide-react';
import { toast } from 'sonner';
import Editor from '@monaco-editor/react';
import { runTestCases } from '../utils/evaluator';

export default function CodeEditor({ problem }: { problem: { title: string; description: string; defaultCode: string; testCases: any[] } }) {
  const [code, setCode] = useState(problem.defaultCode);
  const [testCases, setTestCases] = useState(problem.testCases);
  const [isRunning, setIsRunning] = useState(false);

  const handleRunTests = async () => {
    setIsRunning(true);
    try {
      const results = await runTestCases(code, testCases.map(({ input, expectedOutput }) => ({ input, expectedOutput })));
      setTestCases(prev => prev.map((tc, i) => ({ ...tc, status: results[i] })));
      toast.success(`${results.filter(r => r === 'passed').length}/${results.length} tests passed`);
    } catch (error) {
      toast.error('Failed to run tests: ' + error.message);
    } finally {
      setIsRunning(false);
    }
  };

  return (
    <div className="flex flex-col gap-4 p-4">
      <div className="grid grid-cols-2 gap-4">
        <div className="space-y-4">
          <h2 className="text-2xl font-bold">{problem.title}</h2>
          <p>{problem.description}</p>
        </div>
        <Card className="p-4">
          <Button onClick={() => setTestCases(prev => [...prev, { id: Date.now().toString(), input: '', expectedOutput: '', status: 'pending' }])} size="sm" variant="outline">
            <Plus className="w-4 h-4 mr-2" /> Add Test Case
          </Button>
          {testCases.map(({ id, input, expectedOutput, status }) => (
            <div key={id} className="bg-muted/30 p-3 rounded-lg space-y-2">
              <input value={input} onChange={e => setTestCases(prev => prev.map(tc => tc.id === id ? { ...tc, input: e.target.value } : tc))} />
              <input value={expectedOutput} onChange={e => setTestCases(prev => prev.map(tc => tc.id === id ? { ...tc, expectedOutput: e.target.value } : tc))} />
              <div className={`w-24 text-center p-1 text-sm ${status === 'passed' ? 'bg-green-100' : status === 'failed' ? 'bg-red-100' : 'bg-gray-100'}`}>{status}</div>
              <Button variant="ghost" size="icon" onClick={() => setTestCases(prev => prev.filter(tc => tc.id !== id))}>
                <Trash2 className="w-4 h-4 text-muted-foreground" />
              </Button>
            </div>
          ))}
        </Card>
      </div>
      <div className="h-[600px] relative">
        <Editor height="100%" value={code} onChange={setCode} options={{ minimap: { enabled: false }, fontSize: 14, lineNumbers: 'on' }} />
        <Button className="absolute bottom-4 right-4" onClick={handleRunTests} disabled={isRunning}>
          <Play className="w-4 h-4 mr-2" /> {isRunning ? 'Running...' : 'Run Tests'}
        </Button>
      </div>
    </div>
  );
}
```

Results


Code Editor
Sum Two Number
Test Cases


Code Editor
Sum Two Number
Test Cases


Code Editor
Sum Two Number
Test Cases

6. Testing and Debugging/Results

Testing Process

The testing phase follows a defined workflow:

- **Test Case Definition**: Each test case includes an ID, input data, expected output, and a status (pending, passed, or failed).
- **Test Execution Engine**: This component runs each test case against the user-written code, compares the actual and expected outputs, and updates the test status accordingly.

Test Categories:

- **Input Validation**: Normal inputs, edge cases, boundary values, and invalid data.
- **Output Verification**: Checking for correct results, proper error handling, and adherence to performance expectations.

Debugging and Refinement

After executing the test cases, results are analyzed:

Status Indicators:

- ✓Passed
- ✗Failed
- □ Pending

Metrics Used:

- Total number of test cases
- Pass/fail ratio
- Execution time per test
- Error detection rate
- Effectiveness of individual test cases

Failed test cases trigger a debugging phase where the code is reviewed, errors are identified, and the logic is refined. The updated code is then re-tested to ensure all issues are resolved.

Results Visualization

A visual feedback system is incorporated to make analysis easier:

Real-time updates on pass/fail status

Highlighted errors and mismatches

Graphical representation of performance metrics

7. Conclusion

The implementation of a Test Case Management system within this project has demonstrated the importance of structured and automated software testing. By incorporating components such as a test execution engine, real-time feedback, and results visualization, the project ensures a reliable framework for identifying and correcting errors in code efficiently.

This approach not only improves code quality but also enhances the overall development workflow by promoting test-driven development practices. The use of metrics such as test coverage, pass/fail ratio, and execution time provides valuable insights into system performance and test case effectiveness.

Through systematic testing and debugging, this project emphasizes how a well-managed testing strategy contributes significantly to delivering robust and error-free software solutions.

8. References:

- **Pressman, R. S.** (2014). *Software Engineering: A Practitioner's Approach* (8th Edition). McGraw-Hill Education.
- **Myers, G. J., Sandler, C., & Badgett, T.** (2011). *The Art of Software Testing* (3rd Edition). Wiley.
- **Jorgensen, P. C.** (2013). *Software Testing: A Craftman's Approach* (4th Edition). CRC Press.
- Mozilla Developer Network (MDN). "JavaScript Guide – Functions and Objects." https://developer.mozilla.org
- W3Schools. "JavaScript Testing Basics." https://www.w3schools.com
- Official Node.js Documentation. "Unit Testing in JavaScript." https://nodejs.org
- GitHub – Open Source Projects related to Test Case Management Tools https://github.com