

EX.NO: 1

Reg No:

Date:

Name:

DESIGN A BASIC TEST PLAN DOCUMENT

AIM

To prepare a Test Plan Document for an Online Book Store Application by defining its scope, objectives, resources, and test execution schedule.

PROCEDURE

1. Select the project to be tested.
2. Define the scope of testing.
3. Identify the testing objectives.
4. Allocate the required testing resources.
5. Prepare the test execution schedule.
6. Document the complete test plan.

TEST PLAN

1. Project Name

Online Book Store Application

2. Scope of Testing

The testing scope covers the major functionalities of the Online Book Store Application:

- User Registration
- User Login
- Book Search
- Book Details

- Add to Cart
- Cart Management
- Order Placement
- Order Confirmation

3. Testing Objectives

The main objectives of testing are:

- To verify that all application functionalities work as expected.
- To identify and report software defects.
- To ensure that the application meets the specified requirements.
- To verify the correctness of user interactions and transactions.
- To improve the overall quality, reliability, and usability of the application.

4. Testing Resources

Resource	Quantity
Test Manager	1
QA Engineer	2
Developer	2
Test Environment	1

5. Test Environment

The application will be tested using the required hardware, software, database, browser, and network environment. The test environment will be configured to provide conditions similar to the actual application usage environment.

6. Test Schedule

Activity	Duration
Requirement Analysis	2 Days
Test Case Design	3 Days
Test Execution	5 Days
Defect Reporting	2 Days
Retesting	2 Days

7. Expected Deliverables

The following testing deliverables will be prepared:

- Test Plan Document
- Test Cases
- Test Execution Results
- Defect Reports
- Retesting Results
- Final Test Summary Report

RESULT

The Test Plan Document for the Online Book Store Application was successfully designed. The document includes the testing scope, objectives, resources, test environment, deliverables, and test execution schedule.

EX.NO: 2

Reg No:

Date:

Name:

CLASSIFY SOFTWARE REQUIREMENTS AND MAP THEM TO QUALITY FACTORS

AIM

To classify software requirements into Functional and Non-Functional Requirements and map them to the relevant software quality factors.

PROCEDURE

1. Study the Software Requirements Specification (SRS).
2. Identify the Functional Requirements.
3. Identify the Non-Functional Requirements.
4. Classify the requirements based on their nature.
5. Map each requirement to the appropriate software quality factor.
6. Analyze the relationship between the requirements and quality factors.

REQUIREMENT CLASSIFICATION AND QUALITY MAPPING

1. Functional Requirements

Functional requirements describe what the system should do.

Requirement ID	Functional Requirement	Quality Factor
FR1	User should be able to register an account.	Functionality
FR2	User should be able to log in using valid credentials.	Functionality
FR3	User should be able to search for products.	Functionality

Requirement ID	Functional Requirement	Quality Factor
FR4	User should be able to add products to the cart.	Functionality
FR5	User should be able to place orders online.	Functionality

2. Non-Functional Requirements

Non-functional requirements describe how the system should perform.

Requirement ID	Non-Functional Requirement	Quality Factor
NFR1	System response time should be less than 3 seconds.	Performance Efficiency
NFR2	System should be available 99.9% of the time.	Reliability
NFR3	User passwords should be encrypted.	Security
NFR4	Application should support multiple web browsers.	Compatibility
NFR5	User interface should be easy to use.	Usability
NFR6	System should be easy to modify and maintain.	Maintainability
NFR7	Application should work on Windows, Linux, and Mac.	Portability

SOFTWARE QUALITY FACTORS

Quality Factor	Description
Functionality	Ability of the software to perform the required functions correctly.
Reliability	Ability of the system to operate consistently without failure.
Usability	Ease of learning, understanding, and using the software.

Quality Factor	Description
Performance Efficiency	Ability to provide efficient response time and use system resources effectively.
Security	Protection against unauthorized access, attacks, and data breaches.
Maintainability	Ease of modifying, correcting, and enhancing the software.
Compatibility	Ability of the software to work with different systems and environments.
Portability	Ability of the software to operate on different platforms and operating systems.

RELATIONSHIP BETWEEN REQUIREMENTS AND QUALITY FACTORS

Functional requirements mainly focus on the features and operations that the application must provide, while non-functional requirements define the quality and performance expectations of those features. Mapping requirements to quality factors helps the development and testing teams verify not only whether the system performs the required functions, but also whether it performs them efficiently, securely, reliably, and conveniently for users.

RESULT

The software requirements were successfully classified into Functional and Non-Functional Requirements and mapped to their corresponding software quality factors. This classification helps ensure that both business requirements and quality expectations are properly addressed during software development and testing.

EX.NO: 3

Reg No:

Date:

Name:

DRAFT AN SQA POLICY DOCUMENT

AIM

To develop a Software Quality Assurance (SQA) Policy Document that defines quality objectives, standards, responsibilities, and procedures for ensuring software quality.

PROCEDURE

1. Identify the quality objectives of the organization.
2. Define roles and responsibilities for quality assurance.
3. Establish quality standards and procedures.
4. Specify review, testing, and defect management processes.
5. Document the SQA policy.
6. Review and approve the policy document.

SQA POLICY DOCUMENT

1. Purpose

The purpose of this Software Quality Assurance Policy is to ensure that all software products are developed, tested, and maintained according to established quality standards, project requirements, and customer expectations.

2. Scope

This policy applies to all software development, testing, maintenance, deployment, and support activities carried out within the organization.

3. Quality Objectives

The main quality objectives are:

- To deliver high-quality and reliable software products.
- To minimize defects in software releases.
- To ensure customer satisfaction.
- To comply with organizational and applicable industry standards.
- To improve software development and testing processes continuously.
- To ensure that software requirements are properly implemented and verified.

4. Roles and Responsibilities

Role	Responsibility
Project Manager	Ensure compliance with the SQA policy and manage project quality activities.
QA Manager	Monitor quality activities, conduct audits, and ensure adherence to quality standards.
Developers	Follow coding standards, perform unit testing, and fix identified defects.
Test Engineers	Design, execute, and document test cases and report defects.
Customers	Provide requirements, feedback, and acceptance criteria.

5. Quality Standards

The following quality standards shall be followed:

- Software requirements shall be clearly documented and reviewed.
- Approved coding standards shall be followed during development.
- Peer reviews shall be conducted for important software components.
- Appropriate testing shall be completed before software release.
- All identified defects shall be systematically tracked and resolved.

- Test results and quality records shall be properly maintained.
- Changes to software shall follow an approved change management process.

6. Testing Process

The software shall undergo the following testing activities:

1. Unit Testing – Testing individual modules or components.
2. Integration Testing – Testing the interaction between integrated modules.
3. System Testing – Testing the complete software system.
4. Regression Testing – Ensuring that modifications do not introduce new defects.
5. User Acceptance Testing (UAT) – Verifying that the system satisfies user and business requirements.

7. Defect Management

The following defect management procedures shall be followed:

- All identified defects shall be recorded in a defect tracking system.
- Defects shall be prioritized according to their severity and impact.
- Developers shall analyze and fix the reported defects.
- Fixed defects shall be retested by the testing team.
- Verified defects shall be closed after successful retesting.
- Defect reports shall be maintained for future analysis and process improvement.

8. Review and Audit

- Regular quality reviews shall be conducted throughout the software development lifecycle.
- Internal audits shall be performed to verify compliance with established standards.

- Review findings shall be documented and communicated to the responsible team members.
- Corrective and preventive actions shall be taken for identified quality issues.

9. Continuous Improvement

Continuous improvement activities shall include:

- Collecting and monitoring software quality metrics regularly.
- Analyzing defects and identifying their root causes.
- Implementing process improvements based on quality findings.
- Reviewing the effectiveness of existing testing and development processes.
- Conducting training programs to improve the skills of team members.
- Updating quality standards and procedures when required.

RESULT

A Software Quality Assurance (SQA) Policy Document was successfully drafted. The document defines the quality objectives, scope, roles and responsibilities, quality standards, testing procedures, defect management, reviews, audits, and continuous improvement practices required to ensure software quality.

EX.NO: 4

Reg No:

Date:

Name:

IMPLEMENT THE CONTRACT REVIEW PROCESS WITH A CASE STUDY

AIM

To implement the Contract Review Process for an Online Bookstore Management System and ensure that customer requirements are clearly understood, documented, reviewed, and agreed upon before software development begins.

PROCEDURE

1. Receive the customer requirements document.
2. Analyze the project scope and objectives.
3. Review the functional and non-functional requirements.
4. Identify project risks, constraints, and assumptions.
5. Verify technical feasibility and resource availability.
6. Discuss ambiguities and unclear requirements with the customer.
7. Obtain approval from both the customer and development team.
8. Finalize the Contract Review Report.

CASE STUDY: ONLINE BOOKSTORE MANAGEMENT SYSTEM

1. Project Description

The client requires an Online Bookstore Management System that allows users to browse books, search for products, manage shopping carts, place orders, make online payments, and track their orders. The system should provide a secure, reliable, and user-friendly platform for managing online book purchases.

2. Contract Review Checklist

Review Item	Status	Remarks
Project Scope Defined	Approved	Scope clearly defined
Functional Requirements Reviewed	Approved	All major modules identified
Non-Functional Requirements Reviewed	Approved	Performance and security requirements specified
Resource Availability Checked	Approved	Development and testing team assigned
Budget Verification	Approved	Within client's budget
Schedule Verification	Approved	4-month timeline accepted
Risk Assessment Completed	Approved	Major project risks documented
Technical Feasibility Verified	Approved	Technology stack finalized

3. Functional Requirements

Requirement ID	Description
FR1	User Registration
FR2	User Login
FR3	Book Search
FR4	Add Books to Cart
FR5	Place Orders
FR6	Online Payment
FR7	Order Tracking

4. Non-Functional Requirements

Requirement ID Description

NFR1	System response time should be less than 3 seconds.
NFR2	System should provide 99.9% availability.
NFR3	User authentication should be secure.
NFR4	Application should support multiple web browsers.

5. Risk Analysis

Risk	Impact	Mitigation
Requirement Changes	High	Freeze requirements after customer approval
Schedule Delay	Medium	Conduct weekly project monitoring
Resource Unavailability	Medium	Allocate backup resources
Security Issues	High	Perform security testing and regular security audits

6. Contract Review Report

Item	Details
Customer Name	ABC Book Distributors
Project Name	Online Bookstore Management System
Review Date	02-06-2026
Project Duration	4 Months
Estimated Cost	₹5,00,000
Review Status	Approved

7. Review and Approval

The project requirements, scope, resources, budget, schedule, technical feasibility, and risks were reviewed by the customer and development team. Any ambiguities identified during the review were clarified before approval. Both parties agreed to the finalized requirements and project deliverables.

CONCLUSION

After reviewing the project requirements, resources, schedule, budget, technical feasibility, and risks, the Online Bookstore Management System was found to be feasible and acceptable. The customer and development team agreed on the defined project scope, requirements, schedule, budget, and deliverables before starting the development process.

RESULT

The Contract Review Process for the Online Bookstore Management System was successfully implemented. All major requirements, resources, schedules, risks, budget details, and project constraints were reviewed and approved before project initiation.

EX.NO: 5

Reg No:

Date:

Name:

PREPARE SOFTWARE DEVELOPMENT AND QUALITY PLANS FOR A PROJECT

AIM

To prepare a Software Development Plan (SDP) and Software Quality Assurance Plan (SQAP) for an Employee Leave Management System.

PROCEDURE

1. Identify the project requirements.
2. Define the software development activities.
3. Allocate resources and responsibilities.
4. Establish project milestones and schedules.
5. Prepare the Software Quality Assurance Plan.
6. Define testing, review, and quality control activities.
7. Document the plans and obtain approval.

PROJECT: EMPLOYEE LEAVE MANAGEMENT SYSTEM

1. Project Description

The Employee Leave Management System is designed to automate employee leave requests, approval and rejection processes, leave balance tracking, leave history management, and report generation. The system reduces manual work and provides employees and managers with an efficient platform for managing leave-related activities.

SOFTWARE DEVELOPMENT PLAN (SDP)

2. Project Objectives

The main objectives of the project are:

- To automate employee leave management.
- To reduce manual paperwork and administrative effort.
- To improve leave tracking and reporting.
- To provide accurate leave balance information.
- To ensure secure access to employee leave records.

3. Project Scope

The system includes the following major modules:

- Employee Registration and Login
- Leave Application
- Leave Approval/Rejection
- Leave Balance Management
- Leave History Tracking
- Report Generation

4. Project Team

Role	Responsibility
Project Manager	Project planning, coordination, and monitoring
System Analyst	Requirement analysis and documentation
Developers	Coding and implementation
Test Engineers	Testing and quality assurance
Database Administrator	Database design and management

5. Development Schedule

Activity	Duration
Requirement Gathering	1 Week
System Design	1 Week
Coding	3 Weeks
Testing	2 Weeks
Deployment	1 Week
Maintenance	Ongoing

6. Development Methodology

The project follows an Agile Development Model with:

- Iterative development
- Incremental delivery
- Continuous customer feedback
- Regular testing and review
- Continuous improvement

SOFTWARE QUALITY ASSURANCE PLAN (SQAP)

7. Quality Objectives

The main quality objectives are:

- To deliver reliable and high-quality software.
- To meet all specified customer requirements.
- To ensure system reliability and security.
- To minimize software defects.

- To improve user satisfaction.
- To ensure proper documentation and maintainability.

8. Quality Standards

The following quality standards shall be followed:

- Follow approved coding standards.
- Maintain complete and accurate project documentation.
- Conduct peer code reviews.
- Review requirements and system design.
- Execute test cases before software release.
- Track and resolve defects systematically.
- Maintain test results and quality records.

9. Review Activities

Review Type	Conducted By
Requirement Review	Project Manager and System Analyst
Design Review	Development Team
Code Review	Senior Developers
Test Case Review	QA Team

10. Testing Plan

Testing Type	Purpose
Unit Testing	Verify individual modules
Integration Testing	Verify interactions between modules

Testing Type	Purpose
System Testing	Validate the complete system
Regression Testing	Verify that existing functionality is not affected by changes
User Acceptance Testing	Validate the system against customer requirements

11. Defect Management Process

The defect management process follows these stages:

New → Assigned → Open → Fixed → Retest → Verified → Closed

All identified defects shall be recorded, assigned to the responsible developer, fixed, retested, and closed only after successful verification.

12. Quality Metrics

Metric	Target
Defect Density	< 2 Defects/KLOC
Test Case Pass Rate	> 95%
Requirement Coverage	100%
Critical Defects	0 Before Release

13. Risk Management

Risk	Impact	Mitigation
Requirement Changes	High	Conduct regular requirement review meetings
Schedule Delays	Medium	Perform regular project tracking
High Defect Rate	High	Conduct early testing and reviews
Security Issues	High	Perform security testing and audits

14. Quality Control Activities

The following activities will be performed to maintain software quality:

- Requirement verification
- Design reviews
- Code reviews
- Test case reviews
- Functional and non-functional testing
- Defect tracking and retesting
- Quality metric analysis
- Final release verification

RESULT

The Software Development Plan (SDP) and Software Quality Assurance Plan (SQAP) were successfully prepared for the Employee Leave Management System. The plans define the project objectives, scope, team responsibilities, development schedule, quality standards, review activities, testing strategy, defect management process, quality metrics, and risk management activities required for successful software development and quality assurance.

EX.NO: 6

Reg No:

Date:

Name:

PREPARE V&V PLAN WITH DEFECT REMOVAL FOR LIBRARY MANAGEMENT SYSTEM

AIM

To prepare a Verification and Validation (V&V) Plan and establish a defect removal process for the Library Management System to ensure software quality, reliability, and compliance with user requirements.

PROCEDURE

1. Gather and analyze the system requirements.
2. Define verification activities for each development phase.
3. Define validation activities for testing the system.
4. Establish a defect identification and removal process.
5. Prepare the V&V Plan document.
6. Review and finalize the plan.

LIBRARY MANAGEMENT SYSTEM

1. Functional Requirements

Requirement ID	Requirement Description
FR1	User should be able to log in using valid credentials.
FR2	Librarian should be able to add new books.
FR3	Librarian should be able to update book details.
FR4	Librarian should be able to delete book records.

Requirement ID Requirement Description

FR5	Members should be able to search for books.
FR6	Librarian should be able to issue books to members.
FR7	Librarian should be able to return books from members.
FR8	System should calculate overdue fines automatically.
FR9	System should maintain member records.
FR10	System should generate library reports.

2. Non-Functional Requirements

Requirement ID Requirement Description

NFR1	System response time should be less than 3 seconds.
NFR2	System should be available 24×7.
NFR3	User data should be secure and protected.
NFR4	System should support multiple web browsers.
NFR5	System should be easy to maintain and update.

VERIFICATION PLAN

Verification ensures that the software work products are developed correctly according to the specified requirements and standards.

Phase	Verification Activity
Requirements	Requirement Review
Design	Design Review

Phase	Verification Activity
-------	-----------------------

Development	Code Review
-------------	-------------

Test Planning	Test Case Review
---------------	------------------

Documentation	Documentation Review
---------------	----------------------

Verification Activities

- Requirement Review: Check whether requirements are complete, consistent, and unambiguous.
- Design Review: Verify that the system design satisfies the approved requirements.
- Code Review: Examine source code for correctness, coding standards, and defects.
- Test Case Review: Verify that test cases provide adequate requirement coverage.
- Documentation Review: Ensure that project and technical documents are accurate and complete.

VALIDATION PLAN

Validation ensures that the developed software satisfies the actual needs and requirements of the users.

Testing Type	Purpose
Unit Testing	Validate individual modules
Integration Testing	Validate interactions between modules
System Testing	Validate the complete system
Regression Testing	Verify that existing functionality remains unaffected after changes
User Acceptance Testing	Validate the system against customer and user requirements

DEFECT REMOVAL PROCESS

The defect removal process consists of the following steps:

Step	Activity	Description
1	Defect Identification	Identify defects during reviews or testing.
2	Defect Logging	Record the defect in the defect tracking system.
3	Defect Analysis	Analyze the cause, impact, and severity of the defect.
4	Defect Assignment	Assign the defect to the responsible developer.
5	Defect Fixing	Correct the identified defect.
6	Retesting	Test the corrected functionality.
7	Verification	Verify that the defect has been properly resolved.
8	Defect Closure	Close the defect after successful verification.

Defect Removal Flow

Defect Identification → Defect Logging → Defect Analysis → Defect Assignment → Defect Fixing → Retesting → Verification → Defect Closure

DEFECT SEVERITY CLASSIFICATION

Severity Description

Critical System crash, data loss, or complete failure of a major function

High Major functionality failure that significantly affects system operation

Medium Functional issue that has an available workaround

Low Minor cosmetic, usability, or UI issue

QUALITY METRICS

Metric	Target
Requirement Coverage	100%
Test Case Pass Rate	>95%
Defect Removal Efficiency	>95%
Critical Defects Before Release	0

CONCLUSION

The Verification and Validation Plan ensures that the Library Management System is systematically reviewed, tested, and validated throughout the software development lifecycle. Verification activities focus on checking requirements, design, code, test cases, and documentation, while validation activities confirm that the final system meets user requirements. The defect removal process provides a structured approach for identifying, analyzing, fixing, retesting, and closing defects.

RESULT

The requirements for the Library Management System were successfully identified, and a Verification and Validation (V&V) Plan with a defect removal process was prepared. The plan helps ensure software quality, reliability, requirement compliance, and effective defect management before the system is released.

EX.NO: 7

Reg No:

Date:

Name:

STUDY OF ANY TESTING TOOL (WINRUNNER)

AIM

To study the features, architecture, and working methodology of WinRunner, an automated software testing tool used for functional and regression testing.

PROCEDURE

1. Study the purpose and features of WinRunner.
2. Understand the architecture and components of the tool.
3. Learn the Test Script Language (TSL) used by WinRunner.
4. Explore the recording and playback process.
5. Analyze the testing workflow using WinRunner.
6. Document the observations.

INTRODUCTION

WinRunner is an automated functional testing tool originally developed by Mercury Interactive and later acquired by Hewlett-Packard (HP). It was designed to automate functional and regression testing of software applications. WinRunner enables testers to record user actions, generate test scripts, execute automated tests, and analyze test results.

FEATURES OF WINRUNNER

The major features of WinRunner include:

- Automated Functional Testing
- Regression Testing
- Record and Playback Functionality

- Test Script Language (TSL) Support
- Data-Driven Testing
- GUI Object Recognition
- Integration with Test Management Tools
- Test Result Reporting and Analysis

WINRUNNER ARCHITECTURE

The major components of WinRunner are:

1. GUI Map

- Stores information about application GUI objects.
- Helps WinRunner identify and interact with GUI elements during test execution.

2. Test Script

- Contains automated test commands written using Test Script Language (TSL).
- Defines the actions that should be performed during testing.

3. Test Runner

- Executes the recorded or manually created test scripts.
- Performs the defined actions on the target application.

4. Results Window

- Displays the test execution status.
- Shows passed and failed test results and related information.

WORKING OF WINRUNNER

Step 1: Recording

- Launch WinRunner.

- Open the application to be tested.
- Start the recording process.
- Perform the required user actions.
- WinRunner captures the actions performed by the tester.

Step 2: Script Generation

WinRunner converts the recorded user actions into TSL automation scripts. The generated scripts can be reviewed and modified when required.

Step 3: Playback

- Execute the generated test script.
- WinRunner performs the recorded actions automatically.
- Actual results are compared with the expected results.

Step 4: Result Analysis

- Review the execution results.
- Identify successful and failed test steps.
- Analyze failures and report defects when necessary.

SAMPLE WINRUNNER TSL SCRIPT

```
set_window("Login", 10);

edit_set("Username", "admin");

edit_set("Password", "admin123");

button_press("Login");

obj_check_info("HomePage", "enabled", 1);
```

EXPLANATION

- set_window() selects the Login window.

- `edit_set()` enters the username and password.
- `button_press()` clicks the Login button.
- `obj_check_info()` verifies whether the Home Page object is enabled.

ADVANTAGES OF WINRUNNER

- Reduces manual testing effort.
- Increases test execution speed.
- Supports repetitive testing.
- Improves consistency and repeatability of test execution.
- Supports data-driven testing.
- Provides test execution reports.
- Helps automate functional and regression testing.

LIMITATIONS OF WINRUNNER

- Primarily designed for older Windows-based application environments.
- Requires knowledge of the TSL scripting language for advanced automation.
- Test scripts may require maintenance when the application's GUI changes frequently.
- It is a legacy testing tool and has been superseded in many environments by newer automation solutions such as UFT and Selenium.

APPLICATIONS OF WINRUNNER

WinRunner can be used for:

- Functional Testing
- Regression Testing
- GUI Testing
- Data-Driven Testing

- Database Validation
- Enterprise Application Testing

RESULT

The WinRunner automated testing tool was studied successfully. Its features, architecture, working methodology, Test Script Language (TSL), recording and playback process, advantages, limitations, and applications were analyzed for automated software testing.

EX.NO: 8

Reg No:

Date:

Name:

CASE STUDY FOR IMPLEMENTING WHITE BOX AND BLACK BOX TESTING

AIM

To study and implement White Box Testing and Black Box Testing techniques and understand their applications in software testing.

PROCEDURE

1. Study the concepts of White Box Testing and Black Box Testing.
2. Select a sample application for testing.
3. Design test cases based on software requirements.
4. Execute the test cases and record the results.
5. Analyze the internal program logic using White Box Testing.
6. Document the observations and findings.

1. BLACK BOX TESTING

Definition

Black Box Testing is a software testing technique in which the functionality of an application is tested without examining its internal source code or implementation. The tester validates the system based on requirements, inputs, outputs, and expected behavior.

Case Study: Login Module Testing

Requirements

The Login module follows these requirements:

- Username field is mandatory.

- Password field is mandatory.
- Username must be registered in the system.
- Password must match the registered username.
- Successful login should redirect the user to the Dashboard page.
- Invalid credentials should display an appropriate error message.

Black Box Test Cases

TC ID	Test Scenario	Test Data	Expected Result
TC_01	Login with valid credentials	Username: admin Password: admin123	User should be redirected to the Dashboard page.
TC_02	Login with invalid password	Username: admin Password: test123	Error message "Invalid Username or Password" should be displayed.
TC_03	Login with invalid username	Username: user123 Password: admin123	Appropriate error message should be displayed.
TC_04	Login without username	Username: Blank Password: admin123	"Username is required" validation message should be displayed.
TC_05	Login without password	Username: admin Password: Blank	"Password is required" validation message should be displayed.
TC_06	Login without username and password	Username: Blank Password: Blank	Required field validation messages should be displayed.
TC_07	Login with special characters	Username: @#\$% Password: admin123	Appropriate error message should be displayed.

TC ID	Test Scenario	Test Data	Expected Result
TC_08	Login with password case sensitivity	Username: admin Password: ADMIN123	Login should fail and an error message should be displayed.

Test Steps

TC_01 – Valid Login

1. Open the Login page.
2. Enter the valid username.
3. Enter the valid password.
4. Click the Login button.
5. Verify that the Dashboard page is displayed.

TC_02 – Invalid Password

1. Open the Login page.
2. Enter a valid username.
3. Enter an invalid password.
4. Click the Login button.
5. Verify that an appropriate error message is displayed.

TC_03 – Invalid Username

1. Open the Login page.
2. Enter an invalid username.
3. Enter a valid password.
4. Click the Login button.
5. Verify that an appropriate error message is displayed.

TC_04 – Blank Username

1. Open the Login page.
2. Leave the username field blank.
3. Enter a valid password.
4. Click the Login button.
5. Verify the username validation message.

TC_05 – Blank Password

1. Open the Login page.
2. Enter a valid username.
3. Leave the password field blank.
4. Click the Login button.
5. Verify the password validation message.

2. BOUNDARY VALUE ANALYSIS

Boundary Value Analysis tests values at and around the boundaries of an input range.

Example Requirement

Password length should be between 8 and 15 characters.

TC ID	Boundary Condition	Test Data	Expected Result
BVA_01	Below Minimum	abc1234 – 7 characters	Password should be rejected.
BVA_02	Minimum Boundary	abc12345 – 8 characters	Password should be accepted.

TC ID	Boundary Condition	Test Data	Expected Result
BVA_03	Maximum Boundary	abc123456789012 – 15 characters	Password should be accepted.
BVA_04	Above Maximum	abc1234567890123 – 16 characters	Password should be rejected.

3. EQUIVALENCE PARTITIONING

Equivalence Partitioning divides input data into valid and invalid groups, where each group is expected to behave similarly.

Example Requirement

User age must be between 18 and 60 years.

TC ID	Partition	Input Age	Expected Result
EP_01	Valid Partition	25	Accepted
EP_02	Invalid Partition (<18)	16	Rejected
EP_03	Invalid Partition (>60)	65	Rejected

4. WHITE BOX TESTING

Definition

White Box Testing is a testing technique in which the internal structure, source code, logic, conditions, loops, and execution paths of a program are examined and tested.

Sample Program

```
public class Result {

    public static void main(String[] args) {

        int marks = 65;
```

```

    if (marks >= 50)

        System.out.println("Pass");

    else

        System.out.println("Fail");

}

}

```

White Box Test Cases

TC ID	Condition	Input	Expected Output	Path Covered
WB_01	marks >= 50	65	Pass	True Path
WB_02	marks >= 50	50	Pass	True Path
WB_03	marks < 50	35	Fail	False Path

5. COVERAGE ANALYSIS

Statement Coverage

All executable statements in the program are executed at least once during testing.

Branch Coverage

Both branches of the if-else statement are covered:

- True Branch: marks >= 50
- False Branch: marks < 50

Condition Coverage

The condition marks >= 50 is evaluated with both possible outcomes:

- True – when marks are 50 or above.

- False – when marks are below 50.

6. OBSERVATION

- Black Box Testing focuses on validating software functionality against user requirements without examining the internal code.
- White Box Testing focuses on verifying the internal program logic, conditions, statements, and execution paths.
- Boundary Value Analysis helps identify defects at input boundaries.
- Equivalence Partitioning helps reduce the number of test cases by grouping similar inputs.
- Using both Black Box and White Box Testing provides broader test coverage and helps identify different types of software defects.

RESULT

The implementation of Black Box Testing and White Box Testing was successfully studied. Test cases were designed for the Login module using Black Box Testing techniques, including Boundary Value Analysis and Equivalence Partitioning. The internal program logic, branches, conditions, and execution paths were verified using White Box Testing. Both techniques helped in improving software quality, reliability, and test coverage.

EX.NO: 9

Reg No:

Date:

Name:

DESIGN AND EXECUTE TEST CASES FOR A LOGIN MODULE

AIM

To design and execute comprehensive positive and negative test cases for the Login Module and verify its functionality, validation, security, usability, account lockout, and compatibility.

PROCEDURE

1. Study the Login Module requirements.
2. Identify positive, negative, boundary, validation, security, and usability scenarios.
3. Design test cases covering different user interactions.
4. Execute the test cases.
5. Compare actual results with expected results.
6. Record the execution status.

1. LOGIN MODULE REQUIREMENTS

The Login Module contains:

- Username Field
- Password Field
- Login Button
- Forgot Password Link

Business Rules

- Username is mandatory.

- Password is mandatory.
- Username and password are case-sensitive.
- Only registered users can access the application.
- Successful login redirects the user to the Dashboard.
- Invalid credentials should display an appropriate error message.
- Account should be locked after multiple failed login attempts, if the feature is implemented.

2. POSITIVE TEST CASES

TC ID	Test Scenario	Test Data	Expected Result
TC_001	Login with valid username and password	admin / admin123	Login successful and redirected to Dashboard
TC_002	Login with username at minimum allowed length	Valid data	Login successful
TC_003	Login with username at maximum allowed length	Valid data	Login successful
TC_004	Login with password at minimum allowed length	Valid data	Login successful
TC_005	Login with password at maximum allowed length	Valid data	Login successful
TC_006	Verify Enter key functionality after entering credentials	Valid data	User logged in successfully
TC_007	Verify Login button is clickable	Valid data	Login successful

TC ID	Test Scenario	Test Data	Expected Result
TC_008	Verify successful login after password reset	New password	Login successful

3. NEGATIVE TEST CASES

TC ID	Test Scenario	Test Data	Expected Result
TC_009	Valid username with invalid password	admin / wrong123	Error message displayed
TC_010	Invalid username with valid password	wronguser / admin123	Error message displayed
TC_011	Invalid username and invalid password	wrong / wrong	Error message displayed
TC_012	Login with blank username	Blank / admin123	"Username required" message displayed
TC_013	Login with blank password	admin / Blank	"Password required" message displayed
TC_014	Login with both fields blank	Blank / Blank	Required field validations displayed
TC_015	Login using deleted user account	Deleted account	Login denied
TC_016	Login using inactive account	Inactive user	Login denied
TC_017	Login using locked account	Locked user	Account locked message displayed

TC ID	Test Scenario	Test Data	Expected Result
TC_018	Login with expired password	Expired password	Password expired message displayed

4. VALIDATION TEST CASES

TC ID	Test Scenario	Test Data	Expected Result
TC_019	Username containing special characters	@#\$\$%	Validation/error displayed
TC_020	Password containing only spaces	" "	Validation message displayed
TC_021	Username containing only spaces	" "	Validation message displayed
TC_022	Username with leading spaces	" admin"	Spaces trimmed or validation displayed
TC_023	Username with trailing spaces	"admin "	Spaces trimmed or validation displayed
TC_024	Password with leading spaces	" admin123"	Handled according to requirement
TC_025	Password with trailing spaces	"admin123 "	Handled according to requirement
TC_026	Username exceeding maximum length	100+ characters	Validation message displayed
TC_027	Password exceeding maximum length	100+ characters	Validation message displayed

TC ID	Test Scenario	Test Data	Expected Result
TC_028	Username containing SQL keywords	' OR 1=1 --	Login denied
TC_029	Password containing SQL keywords	' OR 1=1 --	Login denied

5. SECURITY TEST CASES

TC ID	Test Scenario	Test Data	Expected Result
TC_030	Verify password is masked	Password entry	Password displayed as dots/asterisks
TC_031	Verify copy-paste restriction in password field, if applicable	Password	Restriction enforced according to requirement
TC_032	Verify SQL Injection attack prevention	SQL payload	Login denied and input handled securely
TC_033	Verify Cross-Site Scripting (XSS) prevention	<script>alert()</script>	Script should not execute
TC_034	Verify session timeout after inactivity	Valid login	Session expires after configured time
TC_035	Verify direct URL access without login	Dashboard URL	Redirected to Login page
TC_036	Verify browser Back button after logout	Logout → Back Button	User should not access secured pages

6. USABILITY TEST CASES

TC ID	Test Scenario	Expected Result
TC_037	Verify Username field label visibility	Clearly visible
TC_038	Verify Password field label visibility	Clearly visible
TC_039	Verify Login button alignment	Properly aligned
TC_040	Verify error message readability	User-friendly message displayed
TC_041	Verify Forgot Password link functionality	Redirects to password reset page

7. ACCOUNT LOCKOUT TEST CASES

TC ID	Test Scenario	Expected Result
TC_042	Enter wrong password 1 time	Error displayed
TC_043	Enter wrong password 2 times	Error displayed
TC_044	Enter wrong password 3 times	Account locked, if implemented
TC_045	Attempt login after account lock	Account locked message displayed

8. COMPATIBILITY TEST CASES

TC ID	Test Scenario	Expected Result
TC_046	Login using Chrome browser	Login successful
TC_047	Login using Firefox browser	Login successful
TC_048	Login using Edge browser	Login successful
TC_049	Login using Mobile browser	Responsive and functional

TC ID	Test Scenario	Expected Result
TC_050	Login using Tablet device	Responsive and functional

9. TEST EXECUTION SUMMARY

Category	Number of Test Cases
-----------------	-----------------------------

Total Test Cases	50
------------------	----

Positive	8
----------	---

Negative	10
----------	----

Validation	11
------------	----

Security	7
----------	---

Usability	5
-----------	---

Account Lockout	4
-----------------	---

Compatibility	5
---------------	---

Total: 50 Test Cases

10. OBSERVATION

- Positive test cases verify successful authentication using valid credentials.
- Negative test cases verify error handling and validation for invalid inputs.
- Validation test cases verify boundary conditions, spaces, special characters, and invalid input formats.
- Security test cases verify protection against common attacks and unauthorized access.
- Usability test cases verify that the Login Module is clear and easy to use.
- Account lockout test cases verify the behavior after repeated failed login attempts.

- Compatibility test cases verify consistent functionality across different browsers and devices.

RESULT

A comprehensive set of 50 test cases covering positive, negative, validation, security, usability, account lockout, and compatibility scenarios was successfully designed and executed for the Login Module. The testing provides broad coverage of login functionality and verifies the expected behavior of the module under various valid and invalid conditions.

EX.NO: 10

Reg No:

Date:

Name:

IDENTIFICATION AND DOCUMENTATION OF QUALITY COMPONENTS, SQA TOOLS AND EXTERNAL PARTICIPANTS

AIM

To study and document the quality components applied during software development, the Software Quality Assurance (SQA) tools used during the maintenance phase, and the external participants involved in ensuring software quality throughout the Software Development Life Cycle (SDLC).

PROCEDURE

1. Study the Software Quality Assurance process.
2. Identify the quality components applied during software development.
3. Analyze the SQA tools used during software maintenance.
4. Identify external stakeholders involved in QA activities.
5. Document their roles and responsibilities.
6. Evaluate their contribution to software quality.

INTRODUCTION

Software Quality Assurance (SQA) is a systematic process used to ensure that software products meet specified requirements, applicable standards, and customer expectations. SQA activities are performed throughout the Software Development Life Cycle to prevent defects, improve reliability, maintain security, and enhance customer satisfaction.

1. QUALITY COMPONENTS APPLIED DURING SOFTWARE DEVELOPMENT

Quality components are characteristics and practices that contribute to the overall quality of a software product.

1.1 Correctness

Description:

Correctness ensures that the software performs its required functions according to the specified requirements.

Activities:

- Requirement Analysis
- Requirement Reviews
- Requirement Traceability Matrix (RTM)
- Functional Testing

Example:

A banking application should correctly calculate account balances after every transaction.

1.2 Reliability

Description:

Reliability refers to the ability of software to operate consistently without failure for a specified period.

Activities:

- Error Handling
- Recovery Testing
- Stability Testing
- Stress Testing

Example:

An e-commerce website should continue functioning properly during peak traffic periods.

1.3 Efficiency

Description:

Efficiency measures how effectively software utilizes resources such as CPU, memory, storage, and network bandwidth.

Activities:

- Performance Testing
- Load Testing
- Resource Monitoring

Example:

A web page should load within an acceptable response time under normal operating conditions.

1.4 Integrity and Security**Description:**

Integrity ensures that software protects data from unauthorized access, modification, or loss.

Activities:

- Security Testing
- Vulnerability Assessment
- Penetration Testing
- Authentication and Authorization Validation

Example:

Only authorized users should be able to access confidential employee records.

1.5 Usability**Description:**

Usability determines how easily users can learn, understand, and operate the software.

Activities:

- User Interface Reviews
- User Acceptance Testing
- Accessibility Testing

Example:

A user should be able to complete a purchase with minimal effort.

1.6 Maintainability**Description:**

Maintainability measures how easily software can be modified, corrected, or enhanced.

Activities:

- Code Reviews
- Coding Standards
- Documentation Maintenance
- Refactoring

Example:

Developers should be able to add new features without affecting existing functionality.

1.7 Flexibility**Description:**

Flexibility refers to the ability of software to adapt to changing requirements.

Activities:

- Modular Design
- Configurable Architecture
- Change Management

Example:

Adding a new payment gateway should require minimal changes to the existing application.

1.8 Testability**Description:**

Testability refers to how easily software can be tested and verified.

Activities:

- Unit Testing
- Integration Testing
- Automated Testing
- Test Data Preparation

Example:

Each software module should be independently testable.

1.9 Portability**Description:**

Portability measures the ability of software to operate on different platforms and environments.

Activities:

- Cross-Browser Testing
- Cross-Platform Testing
- Environment Validation

Example:

An application should work correctly on Windows, Linux, and macOS.

1.10 Reusability**Description:**

Reusability refers to the ability to use existing software components in multiple applications.

Activities:

- Component-Based Development
- Library Management
- Framework Development

Example:

A common authentication module can be reused across multiple applications.

2. SQA TOOLS USED DURING THE MAINTENANCE PHASE

The maintenance phase begins after software deployment and involves correcting defects, improving performance, managing changes, and implementing enhancements.

Category	Tool	Major Uses
Defect Tracking	JIRA	Bug reporting, defect tracking, sprint and workflow management
Defect Tracking	Bugzilla	Issue tracking and defect lifecycle management
Test Automation	Selenium	Regression, smoke, and functional testing
Test Automation	UFT	Automated functional and GUI testing
API Testing	Postman	API validation and request-response verification
Version Control	Git	Source code management, branching, and change tracking
Repository Management	GitHub / GitLab	Code repository management and collaboration
Code Quality	SonarQube	Static analysis, code smells, and security issues

Category	Tool	Major Uses
Code Quality	Checkstyle	Coding standard enforcement
Performance Testing	JMeter	Load testing, stress testing, and performance analysis
Performance Testing	LoadRunner	Scalability and performance testing
Continuous Integration	Jenkins	Automated builds, continuous testing, and deployment automation

3. EXTERNAL PARTICIPANTS INVOLVED IN QA PROCESSES

Software quality is not maintained only by developers and testers. External participants and stakeholders also contribute to quality assurance.

3.1 Customers

Responsibilities:

- Define and clarify requirements.
- Review project deliverables.
- Provide feedback.
- Approve the final product.

QA Contribution:

- Requirement Validation
- Acceptance Testing

3.2 End Users

Responsibilities:

- Use the software in real-world situations.

- Provide feedback about usability and functionality.
- Report issues encountered during actual usage.

QA Contribution:

- User Acceptance Testing (UAT)
- Usability Evaluation

3.3 External Auditors

Responsibilities:

- Verify compliance with organizational processes.
- Conduct independent quality audits.
- Review quality records and procedures.

QA Contribution:

- Process Assessment
- Compliance Verification

3.4 Certification Bodies

Responsibilities:

- Evaluate compliance with applicable quality standards.
- Assess organizational processes.

Examples:

- ISO 9001 Certification Auditors
- CMMI Assessors

QA Contribution:

- Quality Certification
- Process Improvement Recommendations

3.5 Third-Party Vendors

Responsibilities:

- Provide integrated software services.
- Support external APIs and system interfaces.
- Maintain third-party components.

QA Contribution:

- Integration Testing
- Compatibility Validation

3.6 Regulatory Authorities

Responsibilities:

- Ensure compliance with applicable industry regulations.
- Define regulatory and security requirements.

Examples:

- Banking Regulatory Authorities
- Healthcare Regulatory Bodies

QA Contribution:

- Compliance Verification
- Security and Privacy Audits

3.7 Independent Testing Organizations

Responsibilities:

- Conduct independent software testing.

- Perform specialized testing activities.

QA Contribution:

- Independent Quality Assessment
- Security Evaluation
- Performance Evaluation

4. BENEFITS OF QUALITY ASSURANCE ACTIVITIES

The major benefits include:

- Improved software reliability
- Reduced defect leakage
- Increased customer satisfaction
- Better software maintainability
- Enhanced security
- Reduced maintenance costs
- Improved compliance with standards
- Better software quality
- Improved development and testing processes

5. SUMMARY

Area	Major Components
Development Quality	Correctness, Reliability, Efficiency, Security, Usability, Maintainability, Flexibility, Testability, Portability, Reusability
Maintenance SQA Tools	JIRA, Bugzilla, Selenium, UFT, Postman, Git, GitHub/GitLab, SonarQube, Checkstyle, JMeter, LoadRunner, Jenkins

Area	Major Components
External Participants	Customers, End Users, External Auditors, Certification Bodies, Third-Party Vendors, Regulatory Authorities, Independent Testing Organizations

RESULT

The quality components applied during software development, the SQA tools used during the maintenance phase, and the external participants involved in quality assurance processes were successfully studied and documented. The study demonstrates how these components and participants contribute to defect prevention, software reliability, security, maintainability, compliance, and continuous improvement throughout the Software Development Life Cycle.

EX.NO: 11

Reg No:

Date:

Name:

STUDY OF ANY TEST MANAGEMENT TOOL (QACOMPLETE)

AIM

To study the features, architecture, functionalities, modules, and usage of QAComplete, a Test Management Tool used for managing software testing and Software Quality Assurance activities.

PROCEDURE

1. Study the purpose of Test Management Tools.
2. Understand the features of QAComplete.
3. Analyze the workflow of test management using QAComplete.
4. Study the modules available in the tool.
5. Understand defect tracking and reporting capabilities.
6. Document the observations and findings.

INTRODUCTION

QAComplete is a web-based Test Management Tool developed by SmartBear. It is used by Quality Assurance teams to manage requirements, test cases, test execution, defects, and reporting activities throughout the Software Testing Life Cycle (STLC).

QAComplete provides a centralized platform for organizing testing activities, improving collaboration between team members, tracking defects, maintaining traceability, and generating reports for project stakeholders.

OBJECTIVES OF QACOMPLETE

The main objectives are:

- To manage software testing activities efficiently.

- To maintain centralized test repositories.
- To track defects and issues.
- To improve collaboration among team members.
- To generate testing reports and metrics.
- To ensure complete test coverage.
- To maintain traceability between requirements and test cases.

FEATURES OF QACOMPLETE

1. Requirement Management

- Stores and manages project requirements.
- Maintains requirement traceability.
- Links requirements with corresponding test cases.

2. Test Case Management

- Creates and organizes test cases.
- Maintains test steps and expected results.
- Supports test case reuse across projects.

3. Test Execution Management

- Supports manual and automated test management.
- Records test execution results.
- Tracks pass and fail status.

4. Defect Management

- Allows testers to log software defects.
- Assigns defects to responsible team members.

- Tracks defect status and resolution.

5. Traceability

- Maps requirements to test cases.
- Helps ensure complete requirement coverage.
- Reduces the possibility of missing requirements during testing.

6. Reporting and Dashboard

QAComplete can provide testing information such as:

- Test Execution Reports
- Defect Reports
- Project Progress Reports
- Requirement Coverage Reports
- Testing Metrics and Dashboards

7. Integration Support

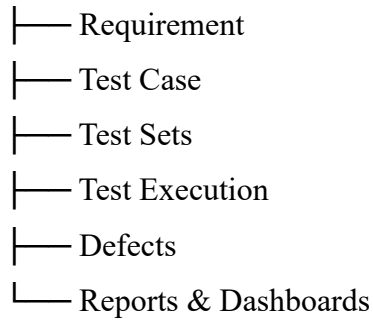
QAComplete can be integrated with various development and testing tools and environments, including:

- JIRA
- Selenium
- Jenkins
- Bug Tracking Systems
- Automation Frameworks

QACOMPLETE ARCHITECTURE

The basic structure of test management in QAComplete can be represented as:

Project



The modules are connected to provide traceability from requirements through test cases and execution to defects and final reports.

MODULES IN QACOMPLETE

1. Requirements Module

Purpose

Stores and manages project requirements.

Activities

- Requirement Creation
- Requirement Review
- Requirement Traceability
- Requirement Status Tracking

Benefits

- Ensures requirement coverage.
- Reduces missed requirements.
- Provides traceability throughout testing.

2. Test Library Module

Purpose

Stores test cases and test suites in a centralized repository.

Activities

- Create Test Cases
- Update Test Cases
- Organize Test Suites
- Maintain Test Steps

Benefits

- Easy test case maintenance.
- Supports test case reuse.
- Provides centralized test management.

3. Test Execution Module

Purpose

Manages the execution of planned test cases.

Activities

- Run Test Cases
- Record Test Results
- Track Pass/Fail Status
- Capture execution evidence where required

Benefits

- Easy tracking of testing progress.
- Better execution management.
- Provides visibility into test results.

4. Defect Management Module

Purpose

Manages software defects throughout their lifecycle.

Defect Lifecycle

New → Assigned → Open → Fixed → Retest → Verified → Closed

Benefits

- Provides systematic defect tracking.
- Improves communication between QA and development teams.
- Helps monitor defect resolution.

5. Reports and Metrics Module

Purpose

Generates reports and metrics related to project quality and testing progress.

Report Type	Description
Test Execution Report	Displays test pass/fail statistics
Defect Summary Report	Displays defect distribution and status
Requirement Coverage Report	Shows requirement testing status
Project Status Report	Shows overall testing progress

WORKFLOW OF QACOMPLETE

Step 1: Requirement Creation

Project requirements are entered and maintained in the system.

Step 2: Test Case Design

Test cases are created based on the requirements and linked to the corresponding requirements.

Step 3: Test Planning

Test sets, test suites, and execution schedules are prepared.

Step 4: Test Execution

Test cases are executed and the results are recorded.

Step 5: Defect Reporting

Defects identified during testing are logged and assigned to the responsible team members.

Step 6: Retesting

Fixed defects are retested by the QA team to verify that the reported issues have been resolved.

Step 7: Reporting

Testing reports, defect reports, coverage reports, and dashboards are generated for project stakeholders.

ADVANTAGES OF QACOMPLETE

- Provides centralized test management.
- Supports requirement traceability.
- Simplifies defect tracking.
- Improves team collaboration.
- Provides testing reports and metrics.
- Supports integration with automation and development tools.
- Provides better visibility into project quality.
- Supports management of both manual and automated testing activities.

LIMITATIONS OF QACOMPLETE

- Requires proper configuration and maintenance.
- New users may require training to use the tool effectively.

- Licensing costs may be a consideration for small organizations.
- Advanced customization may require additional configuration.
- Effective use depends on proper test management practices.

APPLICATIONS OF QACOMPLETE

QAComplete can be used for:

- Functional Testing Projects
- Regression Testing Management
- Agile Testing Projects
- Enterprise Software Testing
- Manual Test Management
- Automated Test Management
- Requirement and Test Traceability
- Defect Management and Reporting

OBSERVATION

QAComplete provides a centralized platform for managing the software testing lifecycle, including requirements, test cases, test execution, defect tracking, traceability, and reporting. It helps testing teams organize their activities, collaborate effectively, monitor testing progress, and provide better visibility into software quality.

RESULT

The QAComplete Test Management Tool was successfully studied. Its features, architecture, modules, workflow, defect management process, reporting capabilities, advantages, limitations, and applications were analyzed. The study demonstrates how QAComplete can support systematic test management and Software Quality Assurance activities throughout the software testing lifecycle.

EX.NO: 12

Reg No:

Date:

Name:

CASE STUDY TO DEMONSTRATE CASE TOOL USAGE TECHNIQUES

AIM

To study and demonstrate the usage of CASE (Computer-Aided Software Engineering) Tools and understand their role in supporting software development and Software Quality Assurance activities.

PROCEDURE

1. Study the concept of CASE Tools.
2. Identify different types of CASE Tools.
3. Analyze the software development activities supported by CASE Tools.
4. Demonstrate the usage of CASE Tool techniques through a case study.
5. Document the observations and findings.

INTRODUCTION

CASE (Computer-Aided Software Engineering) Tools are software applications that assist software developers, analysts, testers, and project managers throughout the Software Development Life Cycle (SDLC). These tools help automate development activities, improve productivity, maintain documentation, reduce errors, and enhance software quality.

CASE Tools can support different phases of software development, including:

- Requirement Analysis
- System Design
- Coding
- Testing

- Maintenance
- Project Management

OBJECTIVES OF CASE TOOLS

The main objectives of CASE Tools are:

- To improve software quality.
- To reduce development effort.
- To automate repetitive tasks.
- To improve software documentation.
- To increase development productivity.
- To support project management activities.
- To facilitate software maintenance.
- To reduce errors during software development.

TYPES OF CASE TOOLS

1. Upper CASE Tools

Upper CASE Tools are mainly used during the initial stages of the SDLC.

Activities Supported

- Requirement Analysis
- Feasibility Study
- System Modeling
- Design Documentation

Examples

- Rational Rose

- StarUML
- Microsoft Visio

2. Lower CASE Tools

Lower CASE Tools are mainly used during the implementation, testing, and maintenance stages.

Activities Supported

- Coding
- Testing
- Debugging
- Software Maintenance

Examples

- Selenium
- JUnit
- Visual Studio

3. Integrated CASE Tools

Integrated CASE Tools support multiple or almost all phases of the Software Development Life Cycle.

Examples

- IBM Rational Suite
- Enterprise Architect

CASE STUDY: ONLINE SHOPPING SYSTEM

Project Description

An Online Shopping System is a software application that allows customers to purchase products through an online platform.

The system provides the following major functions:

- Customer Registration and Login
- Product Search
- Adding Products to Cart
- Placing Orders
- Making Payments
- Tracking Orders

To develop and maintain the system efficiently, different CASE Tools can be used throughout the project lifecycle.

CASE TOOL USAGE TECHNIQUES

1. Requirement analysis technique

CASE Tool Used

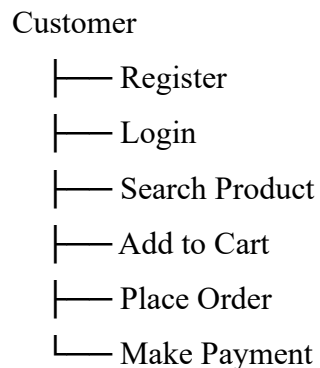
Microsoft Visio / StarUML

Purpose

Used to represent system requirements through diagrams and models.

Activity

A Use Case Diagram can be created to represent the major functions of the Online Shopping System.



Benefit

- Provides clear visualization of system requirements.
- Helps identify system actors and functions.
- Improves communication between stakeholders and development teams.

2. System design technique**CASE Tool Used**

StarUML

Purpose

Used to create software design models before implementation.

Diagrams Created

- Use Case Diagram
- Class Diagram
- Sequence Diagram
- Activity Diagram

Benefit

- Improves understanding of the system.
- Provides a structured design before coding.
- Helps identify relationships between system components.
- Reduces design errors.

3. Database design technique**CASE Tool Used**

MySQL Workbench

Purpose

Used to design and manage the database structure of the application.

Sample Tables

Table Name Purpose

Customer	Stores customer information
Product	Stores product details
Cart	Stores shopping cart data
Order	Stores order information
Payment	Stores payment-related information

Benefit

- Provides organized database design.
- Helps identify relationships between tables.
- Reduces database design errors.
- Supports better database maintenance.

4. Coding support technique

CASE Tool Used

Visual Studio Code / Eclipse

Purpose

Used to assist developers during software implementation.

Features

- Code Suggestions
- Syntax Highlighting

- Code Navigation
- Debugging Support
- Error Detection
- Source Code Management Support

Benefit

- Improves coding productivity.
- Reduces programming errors.
- Makes source code easier to understand and maintain.
- Simplifies debugging.

5. Testing technique

CASE Tool Used

Selenium

Purpose

Used to automate functional and regression testing of web applications.

Example Test Scenarios

Test Case	Input	Expected Result
Valid Login	Valid username and password	Login successful
Invalid Login	Invalid username/password	Error message displayed
Product Search	Valid product name	Matching products displayed
Add to Cart	Select product and click Add to Cart	Product added to cart
Order Placement	Valid order details	Order placed successfully

Benefit

- Reduces manual testing effort.
- Supports repetitive test execution.
- Improves testing efficiency.
- Helps identify defects early.

6. Defect tracking technique**CASE Tool Used**

JIRA

Purpose

Used to record, assign, monitor, and manage software defects.

Defect Lifecycle

New



Assigned



Open



Fixed



Retest



Verified



Closed

Benefit

- Improves defect tracking.
- Provides better communication between QA and development teams.
- Helps monitor defect resolution.
- Maintains a history of reported issues.

7. Project management technique

CASE Tool Used

Microsoft Project

Purpose

Used to plan, schedule, and monitor software development activities.

Activities Managed

- Resource Allocation
- Schedule Planning
- Task Tracking
- Milestone Management
- Risk Management

Benefit

- Helps monitor project progress.
- Supports effective resource utilization.
- Helps identify schedule delays.
- Improves project control.

ADVANTAGES OF CASE TOOLS

- Increased software development productivity.
- Better documentation.

- Improved software quality.
- Reduced development time.
- Efficient defect management.
- Better project planning and control.
- Enhanced team collaboration.
- Reduced development and maintenance errors.
- Improved software maintainability.

LIMITATIONS OF CASE TOOLS

- High initial cost for some tools.
- Training may be required for new users.
- Compatibility issues may occur between different tools.
- Tool configuration and maintenance may require additional effort.
- Some advanced features may require additional licenses or setup.

OBSERVATION

CASE Tools support different phases of the Software Development Life Cycle by providing automation, modeling, documentation, testing, defect tracking, and project management capabilities. Their proper use helps reduce development errors, improve collaboration, maintain project documentation, and enhance overall software quality and maintainability.

RESULT

The usage techniques of CASE Tools were successfully demonstrated through the Online Shopping System case study. Various CASE Tools supporting requirement analysis, system design, database design, coding, testing, defect tracking, and project management were studied. The case study demonstrated how CASE Tools contribute to improving software development efficiency,

quality assurance, documentation, and project management throughout the Software Development Life Cycle.

EX.NO: 13

Reg No:

Date:

Name:

TESTING PROTOCOL TEMPLATE AND PRE-DEPLOYMENT DEVICE INSPECTION CHECKLIST WITH MAINTENANCE

AIM

To design a standard Testing Protocol Template and a Pre-Deployment Device Inspection Checklist with maintenance activities to ensure software quality, system reliability, and device readiness before production deployment.

PROCEDURE

1. Study the software deployment process.
2. Identify the testing activities required before deployment.
3. Prepare a standard Testing Protocol Template.
4. Create a Pre-Deployment Device Inspection Checklist.
5. Define maintenance activities required after deployment.
6. Review and document the findings.

INTRODUCTION

A Testing Protocol is a structured document that defines the procedures, test activities, test results, and acceptance criteria required before software deployment. It provides a systematic approach for verifying that the application meets its functional and non-functional requirements.

A Pre-Deployment Device Inspection Checklist ensures that hardware, operating system, application configuration, network connectivity, security settings, and backup mechanisms are properly verified before releasing an application to the production environment.

These activities help reduce deployment failures, improve software reliability and security, ensure device readiness, and support smooth system operation after deployment.

1. TESTING PROTOCOL TEMPLATE

A. Project Information

Field	Details
Project Name	_____
Version	_____
Test Environment	_____
Tester Name	_____
Test Start Date	_____
Test End Date	_____
Testing Scope	_____

B. Testing Scope

Test Type	Description
Functional Testing	Verifies application functionality
Integration Testing	Verifies interactions between modules
System Testing	Verifies complete system behavior
Regression Testing	Verifies that existing functionality remains unaffected
Performance Testing	Verifies system performance and response time
Security Testing	Verifies security controls and protection mechanisms
User Acceptance Testing	Verifies that the system meets business requirements

C. Test Execution Summary

Test Type	Total Cases	Passed	Failed
-----------	-------------	--------	--------

Functional Testing			
--------------------	--	--	--

Integration Testing			
---------------------	--	--	--

System Testing			
----------------	--	--	--

Regression Testing			
--------------------	--	--	--

Security Testing			
------------------	--	--	--

User Acceptance Testing			
-------------------------	--	--	--

Total			
-------	--	--	--

D. Defect Summary

Severity	Count
----------	-------

Critical	
----------	--

High	
------	--

Medium	
--------	--

Low	
-----	--

Total	
-------	--

E. Exit Criteria

The software can proceed towards deployment when:

- All critical defects are closed.
- High-priority defects are resolved or formally accepted.
- Planned test execution is completed.
- Test results are reviewed and documented.

- Test summary report is approved.
- Required customer or stakeholder sign-off is obtained.

2. PRE-DEPLOYMENT DEVICE INSPECTION CHECKLIST

The following checklist should be completed before installing and deploying the application on a production device.

A. Hardware Verification

Check Item	Status (✓/X)	Remarks
System power supply working		
CPU functioning properly		
RAM installed and functioning correctly		
Hard disk/SSD health verified		
Network adapter working		
Peripheral devices connected and functioning		

B. Operating System Verification

Check Item	Status (✓/X)	Remarks
Operating system installed		
Latest required updates applied		
System date and time configured		
Required drivers installed		
User permissions configured		

C. Application Verification

Check Item	Status (✓/✗)	Remarks
-------------------	---------------------	----------------

Application installed successfully		
------------------------------------	--	--

Configuration files updated		
-----------------------------	--	--

Database connectivity verified		
--------------------------------	--	--

Application launches successfully		
-----------------------------------	--	--

All required modules are accessible		
-------------------------------------	--	--

D. Security Verification

Check Item	Status (✓/✗)	Remarks
-------------------	---------------------	----------------

Antivirus/security software updated		
-------------------------------------	--	--

Firewall enabled and configured		
---------------------------------	--	--

User authentication configured		
--------------------------------	--	--

SSL/TLS certificates installed where required		
---	--	--

Password policies applied		
---------------------------	--	--

E. Network Verification

Check Item	Status (✓/✗)	Remarks
-------------------	---------------------	----------------

Internet/network connectivity available		
---	--	--

Server connectivity verified		
------------------------------	--	--

DNS configuration verified		
----------------------------	--	--

Check Item	Status (✓/X)	Remarks
------------	--------------	---------

Required API endpoints reachable

Network latency acceptable

F. Backup Verification

Check Item	Status (✓/X)	Remarks
------------	--------------	---------

Database backup completed

Application backup available

Recovery plan prepared

Rollback plan documented

3. MAINTENANCE CHECKLIST

Software maintenance is performed after deployment to correct defects, adapt the application to changing environments, improve performance, and prevent future problems.

A. Corrective Maintenance

Corrective maintenance is performed to fix problems identified after deployment.

Activity	Purpose
----------	---------

Bug Fixing	Resolve reported software defects
------------	-----------------------------------

Error Correction	Improve system stability
------------------	--------------------------

Patch Installation	Resolve known issues and address vulnerabilities
--------------------	--

B. Adaptive Maintenance

Adaptive maintenance modifies the software to support changes in its operating environment.

Activity	Purpose
Operating System Compatibility Updates	Support new or updated platforms
Browser Compatibility Updates	Support newer browser versions
API Updates	Maintain compatibility with external services

C. Perfective Maintenance

Perfective maintenance improves existing features and system performance.

Activity	Purpose
Performance Optimization	Improve response time and resource usage
UI Enhancements	Improve usability and interface quality
Feature Improvements	Meet changing business and user requirements

D. Preventive Maintenance

Preventive maintenance is performed to identify and prevent potential problems before they affect the system.

Activity	Purpose
Log Monitoring	Detect problems at an early stage
Database Optimization	Improve database efficiency
Security Audits	Identify and prevent security weaknesses
Backup Verification	Ensure data recovery readiness

4. POST-DEPLOYMENT MONITORING CHECKLIST

Check Item	Status (✓/X)	Remarks
Application running successfully		
Error logs reviewed		
Performance metrics monitored		
User access verified		
Database operations verified		
Backup jobs running successfully		
Network connectivity monitored		
Critical application services verified		

BENEFITS

The proposed Testing Protocol and Pre-Deployment Device Inspection Checklist provide the following benefits:

- Reduces deployment risks.
- Ensures device and system readiness.
- Improves software reliability.
- Enhances system security.
- Helps identify configuration problems before deployment.
- Supports systematic software maintenance.
- Minimizes production failures and downtime.
- Improves recovery and rollback readiness.
- Supports continuous monitoring after deployment.

OBSERVATION

The Testing Protocol provides a structured method for verifying software functionality, performance, security, integration, and user acceptance before deployment. The Pre-Deployment Device Inspection Checklist ensures that hardware, operating system, application, security, network, and backup configurations are properly verified.

Maintenance activities such as corrective, adaptive, perfective, and preventive maintenance help maintain software quality throughout its operational life. Post-deployment monitoring further helps identify and resolve issues at an early stage.

RESULT

The Testing Protocol Template and Pre-Deployment Device Inspection Checklist with Maintenance activities were successfully designed. The protocol provides a systematic approach for software testing, deployment readiness verification, device inspection, post-deployment monitoring, and ongoing maintenance, thereby supporting reliable, secure, and high-quality software delivery.

EX.NO: 14

Reg No:

Date:

Name:

CREATE A TEST PLAN DOCUMENT FOR A LIBRARY MANAGEMENT SYSTEM

AIM

To prepare an industry-standard Test Plan Document for the Library Management System by defining the testing scope, objectives, test strategy, resources, schedule, risks, and deliverables to ensure software quality.

PROCEDURE

1. Study the Software Requirement Specification (SRS) of the Library Management System.
2. Identify the modules to be tested.
3. Define the testing objectives and scope.
4. Select appropriate testing strategies.
5. Prepare the test environment and resource plan.
6. Define entry and exit criteria.
7. Create the defect management process.
8. Prepare the test schedule and risk management plan.
9. Review and approve the Test Plan Document.

TEST PLAN DOCUMENT

1. DOCUMENT INFORMATION

Item	Details
-------------	----------------

Project Name	Library Management System
--------------	---------------------------

Item	Details
Test Plan ID	LMS-TP-001
Version	1.0
Prepared By	QA Team
Reviewed By	Test Lead
Approved By	Project Manager
Date	_____

2. INTRODUCTION

The Library Management System (LMS) is developed to automate library operations such as maintaining book records, managing members, issuing and returning books, calculating fines, and generating reports.

This Test Plan describes the overall testing strategy and activities required to validate that the Library Management System satisfies its functional and non-functional requirements before deployment. It provides a systematic approach for test planning, execution, defect management, reporting, and test closure.

3. TESTING OBJECTIVES

The primary objectives of testing are:

- To verify all functional requirements.
- To validate important business workflows.
- To ensure data integrity and accuracy.
- To identify, document, and track software defects.
- To verify system security and reliability.
- To ensure proper interaction between system modules.

- To validate system performance under expected conditions.
- To ensure user satisfaction through User Acceptance Testing (UAT).

4. TEST SCOPE

4.1 In-Scope Modules

Module	Description
Login Module	User authentication
Book Management	Add, edit, and delete books
Member Management	Add, update, and delete members
Search Module	Search books and related records
Book Issue Module	Issue books to members
Book Return Module	Return books and update availability
Fine Calculation Module	Calculate overdue fines
Reports Module	Generate library reports

4.2 Out-of-Scope Modules

The following areas are outside the scope of this testing plan:

- Mobile Application Testing
- Third-Party Payment Gateway Integration
- Hardware Device Testing
- SMS and Email Notification Services

5. TEST STRATEGY

5.1 Functional Testing

Functional testing will verify that all system functions operate according to the specified requirements.

5.2 Integration Testing

Integration testing will verify interactions between modules, such as Book Issue, Book Return, and Book Inventory Management.

5.3 System Testing

System testing will validate complete end-to-end business processes of the Library Management System.

5.4 Regression Testing

Regression testing will ensure that existing functionality is not affected by modifications, bug fixes, or new features.

5.5 Security Testing

Security testing will verify:

- User authentication
- Authorization
- Password protection
- Data protection
- Session security

5.6 User Acceptance Testing

UAT will validate whether the system satisfies business requirements and user expectations before final release.

6. TEST ENVIRONMENT

6.1 Hardware Requirements

Component Specification

Processor Intel Core i5 or higher

RAM 8 GB minimum

Storage 256 GB SSD or higher

6.2 Software Requirements

Component	Version/Specification
------------------	------------------------------

Operating System	Windows 11
------------------	------------

Browsers	Chrome, Firefox, Edge
----------	-----------------------

Database	MySQL 8.0
----------	-----------

Application Server	Apache Tomcat
--------------------	---------------

Defect Tracking Tool	JIRA
----------------------	------

Test Management Tool	QAComplete
----------------------	------------

7. TEST DELIVERABLES

7.1 Pre-Testing Deliverables

- Test Plan Document
- Software Requirement Specification
- Test Cases
- Test Data
- Test Environment Details

7.2 During Testing

- Test Execution Reports

- Defect Reports
- Daily Status Reports
- Updated Test Results

7.3 Post-Testing Deliverables

- Test Summary Report
- Defect Closure Report
- Test Metrics Report
- Sign-Off Document

8. ROLES AND RESPONSIBILITIES

Role	Responsibility
Project Manager	Project monitoring and approval
Business Analyst	Requirement clarification
Developer	Defect fixing and technical support
QA Lead	Test planning, review, and monitoring
Test Engineer	Test design and execution
Customer	User Acceptance Testing

9. ENTRY CRITERIA

Testing will begin only when the following conditions are satisfied:

- Requirements are approved.
- Test environment is available and configured.
- Application build is successfully deployed.
- Test cases are reviewed and approved.

- Required test data is prepared.
- Required testing resources are available.

10. EXIT CRITERIA

Testing will be completed when:

- 100% of planned test cases have been executed.
- All Critical and High severity defects are closed.
- Required test coverage has been achieved.
- Test Summary Report is completed and approved.
- Customer or authorized stakeholder sign-off is received.

11. DEFECT MANAGEMENT PROCESS

11.1 Defect Lifecycle

New



Assigned



Open



In Progress



Fixed



Retest



Verified



Closed

11.2 Defect Severity Levels

Severity Description

Critical System crash, major data loss, or complete system failure

High Major functionality failure affecting important business operations

Medium Partial functionality failure with an available workaround

Low Minor functional, cosmetic, or UI issue

12. TEST SCHEDULE

Activity	Duration
Requirement Analysis	2 Days
Test Planning	2 Days
Test Case Preparation	4 Days
Test Environment Setup	2 Days
Functional Testing	5 Days
Integration Testing	3 Days
Regression Testing	3 Days
UAT Support	2 Days
Test Closure	1 Day

13. RISKS AND MITIGATION

Risk	Mitigation
Requirement Changes	Conduct requirement reviews and obtain approval before testing

Risk	Mitigation
Build Delays	Maintain buffer time in the testing schedule
Environment Issues	Perform early environment setup and validation
Resource Unavailability	Assign backup resources
High Defect Count	Conduct daily defect review meetings
Test Data Issues	Prepare and validate test data before execution

14. TEST METRICS

Metric	Target
Test Case Execution	100%
Requirement Coverage	100%
Test Pass Percentage	>95%
Defect Removal Efficiency	>95%
Critical Open Defects	0

15. QUALITY MONITORING

The following activities will be performed throughout the testing process:

- Regular test execution monitoring.
- Daily defect review.
- Requirement-to-test-case traceability.
- Test coverage monitoring.
- Review of critical and high-severity defects.
- Periodic test status reporting.

- Final review of test results before release.

OBSERVATION

The Test Plan provides a structured approach for testing the Library Management System. It defines the testing objectives, scope, strategy, environment, resources, deliverables, schedule, defect management process, risks, and quality metrics required for effective Software Quality Assurance.

The plan also establishes clear entry and exit criteria, ensuring that testing is performed systematically and that the system meets the required quality standards before deployment.

RESULT

An industry-standard Test Plan Document for the Library Management System was successfully prepared. The document includes testing objectives, scope, strategy, test environment, deliverables, roles and responsibilities, entry and exit criteria, defect management, test schedule, risk management, and quality metrics, providing a comprehensive framework for effective software testing and reliable system delivery.

EX.NO: 15

Reg No:

Date:

Name:

STUDY AND IMPLEMENTATION OF SOFTWARE QUALITY METRICS

AIM

To study Software Quality Metrics and implement various metrics to measure the quality, performance, reliability, and effectiveness of a software product and its testing process.

PROCEDURE

1. Study the concept of Software Quality Metrics.
2. Identify product, process, and project metrics.
3. Select suitable quality metrics for evaluation.
4. Collect the required project data.
5. Calculate the metrics using standard formulas.
6. Analyze the calculated results.
7. Document the observations and findings.

INTRODUCTION

Software Quality Metrics are quantitative measures used to evaluate the quality of software products, processes, and projects. These metrics provide measurable information about software quality, testing effectiveness, reliability, project progress, and development processes.

Quality metrics help organizations identify areas for improvement, monitor testing activities, reduce defects, evaluate project performance, and support management decision-making. They also provide objective information for continuous process improvement and quality assurance.

OBJECTIVES OF SOFTWARE QUALITY METRICS

The main objectives are:

- To measure software quality.
- To evaluate testing effectiveness.
- To monitor project progress.
- To improve software reliability.
- To reduce defects in production.
- To support management decision-making.
- To improve software development and testing processes.
- To enhance customer satisfaction.

CLASSIFICATION OF SOFTWARE QUALITY METRICS

1. Product Metrics

Product metrics measure the characteristics and quality of the software product.

Examples

- Defect Density
- Reliability
- Code Complexity
- Maintainability
- Performance

2. Process Metrics

Process metrics measure the effectiveness and efficiency of software development and testing processes.

Examples

- Defect Removal Efficiency
- Test Effectiveness

- Review Effectiveness
- Defect Leakage

3. Project Metrics

Project metrics measure project execution, planning, and management.

Examples

- Schedule Variance
- Effort Variance
- Resource Utilization
- Cost Variance

SOFTWARE QUALITY METRICS AND CALCULATIONS

1. DEFECT DENSITY

Definition

Defect Density measures the number of defects found per unit size of software.

Formula

Defect Density = Total Number of Defects / Software Size (KLOC)

Example

- Total Defects = 25
- Software Size = 5 KLOC

Calculation:

Defect Density = $25 / 5$

Defect Density = 5 Defects/KLOC

Result

Defect Density = 5 Defects/KLOC

2. DEFECT REMOVAL EFFICIENCY (DRE)

Definition

Defect Removal Efficiency measures the effectiveness of the development and testing process in removing defects before software release.

Formula

$$\text{DRE} = (\text{Defects Removed Before Release} / \text{Total Defects}) \times 100$$

Example

- Defects Removed Before Release = 90
- Defects Found After Release = 10
- Total Defects = 90 + 10 = 100

Calculation

$$\text{DRE} = (90 / 100) \times 100$$

$$\text{DRE} = 90\%$$

Result

Defect Removal Efficiency = 90%

3. TEST CASE EXECUTION PERCENTAGE

Definition

Test Case Execution Percentage measures the percentage of planned test cases that have been executed.

Formula

$$\text{Test Execution \%} = (\text{Executed Test Cases} / \text{Planned Test Cases}) \times 100$$

Example

- Planned Test Cases = 500

- Executed Test Cases = 480

Calculation

$$\text{Test Execution \%} = (480 / 500) \times 100$$

$$\text{Test Execution \%} = 96\%$$

Result

$$\text{Test Execution Percentage} = 96\%$$

4. TEST PASS PERCENTAGE

Definition

Test Pass Percentage measures the percentage of executed test cases that passed successfully.

Formula

$$\text{Pass Rate} = (\text{Passed Test Cases} / \text{Executed Test Cases}) \times 100$$

Example

- Passed Test Cases = 450
- Executed Test Cases = 480

Calculation

$$\text{Pass Rate} = (450 / 480) \times 100$$

$$\text{Pass Rate} = 93.75\%$$

Result

$$\text{Test Pass Percentage} = 93.75\%$$

5. DEFECT LEAKAGE

Definition

Defect Leakage measures the percentage of defects discovered by customers or users after the software has been released.

Formula

$$\text{Defect Leakage} = (\text{Production Defects} / \text{Total Defects}) \times 100$$

Example

- Production Defects = 5
- Total Defects = 100

Calculation

$$\text{Defect Leakage} = (5 / 100) \times 100$$

$$\text{Defect Leakage} = 5\%$$

Result

$$\text{Defect Leakage} = 5\%$$

6. REQUIREMENT COVERAGE**Definition**

Requirement Coverage measures the percentage of requirements that are covered and tested by test cases.

Formula

$$\text{Requirement Coverage} = (\text{Requirements Tested} / \text{Total Requirements}) \times 100$$

Example

- Total Requirements = 120
- Requirements Tested = 120

Calculation

$$\text{Requirement Coverage} = (120 / 120) \times 100$$

$$\text{Requirement Coverage} = 100\%$$

Result

Requirement Coverage = 100%

7. MEAN TIME BETWEEN FAILURES (MTBF)

Definition

MTBF measures system reliability by calculating the average operating time between two consecutive failures.

Formula

$MTBF = \text{Total Operating Time} / \text{Number of Failures}$

Example

- Total Operating Time = 200 Hours
- Number of Failures = 4

Calculation

$MTBF = 200 / 4$

MTBF = 50 Hours

Result

Mean Time Between Failures = 50 Hours

8. REVIEW EFFECTIVENESS

Definition

Review Effectiveness measures the effectiveness of software reviews in identifying defects before the testing or release stage.

Formula

$\text{Review Effectiveness} = (\text{Defects Found During Reviews} / \text{Total Defects}) \times 100$

Example

- Defects Found During Reviews = 40

- Total Defects = 50

Calculation

Review Effectiveness = $(40 / 50) \times 100$

Review Effectiveness = 80%

Result

Review Effectiveness = 80%

SUMMARY OF METRICS

Metric	Calculated Value
Defect Density	5 Defects/KLOC
Defect Removal Efficiency	90%
Test Execution Percentage	96%
Test Pass Percentage	93.75%
Defect Leakage	5%
Requirement Coverage	100%
MTBF	50 Hours
Review Effectiveness	80%

INTERPRETATION OF RESULTS

The calculated metrics provide measurable information about the software testing process:

- Defect Density: 5 defects were identified per KLOC of software.
- DRE: 90% of the total identified defects were removed before release.
- Test Execution: 96% of the planned test cases were executed.

- Test Pass Rate: 93.75% of the executed test cases passed.
- Defect Leakage: 5% of the total defects were identified after release.
- Requirement Coverage: All identified requirements were covered by testing.
- MTBF: The system operated for an average of 50 hours between failures.
- Review Effectiveness: 80% of the identified defects were detected during reviews.

BENEFITS OF SOFTWARE QUALITY METRICS

- Provides objective measurement of software quality.
- Helps identify weaknesses in development and testing processes.
- Improves testing efficiency.
- Supports project monitoring and control.
- Helps reduce production defects.
- Improves software reliability.
- Supports management decision-making.
- Enhances customer satisfaction.
- Facilitates continuous process improvement.

OBSERVATION

Software Quality Metrics provide measurable indicators for evaluating software quality, testing effectiveness, reliability, and project performance. Metrics such as Defect Density, Defect Removal Efficiency, Test Execution Percentage, Test Pass Percentage, Defect Leakage, Requirement Coverage, MTBF, and Review Effectiveness help organizations identify quality issues and monitor the effectiveness of their software development and testing activities.

RESULT

The Software Quality Metrics were successfully studied and implemented. Various product, process, and project-related metrics were calculated using sample project data. The results

demonstrated how quantitative measurements can be used to evaluate software quality, testing effectiveness, reliability, defect management, requirement coverage, and overall project performance.