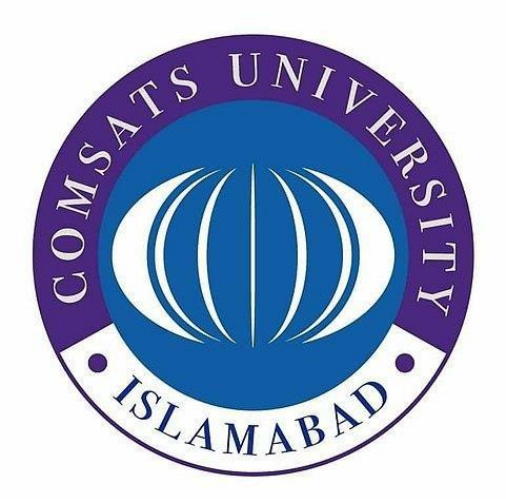


COMSATS UNIVERSITY ISLAMABAD, ABBOTTABAD CAMPUS

Assignment 01

Software Testing

Submitted by:

Hanzla Nouman FA21-BSE-015-6A

Submitted To:

Sir Mukhtiar Zamin

Introduction

Test planning is a critical aspect of software testing. It outlines the strategy, resources, scope, and schedule for testing activities to ensure that a software product meets its requirements and is defect-free. Test plans have significantly evolved from manual documentation to automated and integrated processes influenced by various methodologies, technological advancements, and industry standards. This assignment explores the evolution of test plans, from early manual documentation to modern practices for AI/ML systems.

Initial Test Plans: Manual and Documented (1980s)

In the early days of software development, testing was largely a manual process. Test plans were documented but were simplistic and lacked standardization. They relied heavily on the manual effort and expertise of individual testers.

Example Project:

- **Company:** Large Government Agency
- **Project:** Mainframe Payroll System
- **Details:** Tested payroll processing and report generation.

Manual Test Plan Template (1980s)

- **Title:** Test Plan for [Application Name]
- **Objective:** [Objective of the test plan]
- **Scope:** [Scope of testing]
- **Resources:**
    - **Tester:** [Tester Name]
    - **Environment:** [Testing environment]
    - **Tools:** [Tools used]
- **Schedule:**
    - **Start Date:** [Start Date]
    - **End Date:** [End Date]
- **Test Cases:** [Detailed test cases with TC ID, description, input data, expected result, actual result, status]

Evolution to Formal Test Plans (1990s)

As software development evolved, the need for more structured and documented test plans became evident. Formal test plans provided better organization and traceability, leading to more effective testing processes.

Example Project:

- **Company:** Global Retail Corporation
- **Project:** Inventory Management System
- **Details:** Covered stock tracking, order processing, and reporting features.

Formal Test Plan Template (1990s)

- **Title:** Test Plan for [Application Name]
- **Introduction:** [Introduction to the test plan]
- **Test Items:** [Items to be tested]
- **Features to be Tested:** [Features and functionalities to be tested]
- **Features not to be Tested:** [Features excluded from testing]
- **Approach:** [Testing approach and techniques]
- **Item Pass/Fail Criteria:** [Criteria for passing or failing test items]
- **Test Deliverables:** [Documents and reports to be delivered]
- **Testing Tasks:** [Tasks to be performed]
- **Environmental Needs:** [Hardware, software, and other requirements]
- **Responsibilities:** [Roles and responsibilities of team members]
- **Schedule:** [Testing schedule]
- **Risks and Contingencies:** [Identified risks and contingency plans]
- **Approvals:** [Approvals required for the test plan]

Standardized Test Plans with ISO/IEC/IEEE 29119 (2000s)

The ISO/IEC/IEEE 29119 standard brought a comprehensive framework for software testing, providing detailed guidelines for creating standardized test plans that ensure quality and consistency.

Example Project:

- **Company:** International Banking Institution
- **Project:** Online Banking Platform
- **Details:** Covered user authentication, transactions, and security features.

ISO/IEC/IEEE 29119 Test Plan Template

- **Title:** Test Plan for [Application Name]
- **Test Plan Identifier:** [Unique Identifier]
- **Introduction:** [Introduction to the test plan]
- **Test Items:** [Items to be tested]
- **Features to be Tested:** [Features and functionalities to be tested]
- **Features not to be Tested:** [Features excluded from testing]
- **Approach:** [Testing approach and techniques]
- **Item Pass/Fail Criteria:** [Criteria for passing or failing test items]
- **Suspension Criteria and Resumption Requirements:** [Conditions for suspending and resuming testing]
- **Test Deliverables:** [Documents and reports to be delivered]
- **Testing Tasks:** [Tasks to be performed]
- **Environmental Needs:** [Hardware, software, and other requirements]
- **Responsibilities:** [Roles and responsibilities of team members]
- **Staffing and Training Needs:** [Staffing requirements and training plans]
- **Schedule:** [Testing schedule]
- **Risks and Contingencies:** [Identified risks and contingency plans]
- **Approvals:** [Approvals required for the test plan]

Continuous Integration and Continuous Deployment (CI/CD) Test Plan

Continuous Integration (CI) and Continuous Deployment (CD) are pivotal in modern software development practices, facilitating the automation of the build, test, and deployment processes. These practices ensure that code changes are consistently integrated and deployed, allowing for rapid delivery and real-time feedback. This section outlines the CI/CD test plan template, detailing the essential components to ensure effective and efficient testing within a CI/CD pipeline.

Example Project: CI/CD Test Plan for E-commerce Website

Title

CI/CD Test Plan for E-Shop Application

Pipeline Configuration

- **CI/CD Tool:** GitHub Actions
- **Pipeline Stages:**
    - **Source:** Code commit and repository in GitHub
    - **Build:** Compilation using Node.js
    - **Test:** Automated tests with Jest
    - **Deploy:** Deployment to AWS Elastic Beanstalk
    - **Monitor:** Monitoring with AWS CloudWatch

Automated Tests

- **Unit Tests:** Validate individual components (e.g., product listing, user authentication)
- **Integration Tests:** Verify interactions between modules (e.g., cart functionality, payment processing)
- **System Tests:** Validate end-to-end functionality (e.g., browsing, checkout process)
- **Regression Tests:** Ensure new changes do not disrupt existing features

Manual Tests

- **Exploratory Testing:** Identify unexpected issues through ad-hoc testing
- **User Acceptance Testing (UAT):** Validate the system with end-users before production

Test Data

- **Data Sources:** Sample data including user accounts, product listings, and transactions
- **Data Management:** Regularly refresh and manage test data to ensure relevance

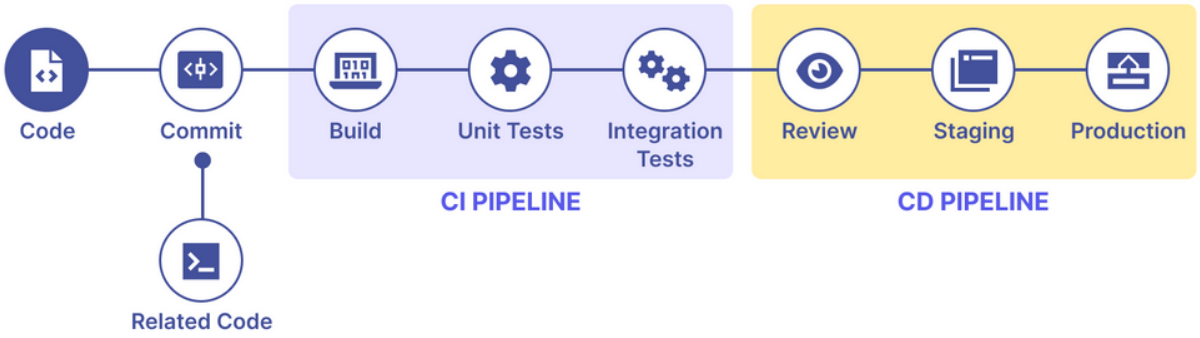
Environment

- **Development Environment:** Local setup with Node.js, React, MongoDB
- **Testing Environment:** Isolated testing environment replicating production settings
- **Staging Environment:** Pre-production environment for final validation
- **Production Environment:** Live environment on AWS Elastic Beanstalk

Build and Deployment Process

- **Build Steps:**
    1. Clone repository
    2. Install dependencies
    3. Run linting and tests
    4. Compile code
- **Deployment Steps:**
    1. Deploy to staging
    2. Run automated tests
    3. Deploy to production
- **Rollback Plan:** Revert to the previous stable release in case of failure, followed by regression testing.

ISO/IEC/IEEE 29119 Test Plan Template



Summary

The CI/CD test plan is designed to integrate seamlessly into the CI/CD pipeline, ensuring continuous and automated testing throughout the software development lifecycle. By detailing each aspect of the testing process, from automated and manual tests to environment configurations and rollback procedures, the CI/CD test plan aims to enhance software quality, accelerate delivery, and minimize risks.

References

1. **Wikipedia - CI/CD:** Overview of Continuous Integration and Continuous Deployment practices. Available at: CI/CD on Wikipedia
2. **GitLab - CI/CD:** Detailed explanation of CI/CD practices and benefits. Available at: CI/CD on GitLab
3. **Medium - Implementing CI/CD:** Article on implementing CI/CD in software projects. Available at: Implementing CI/CD on Medium