SIT725

Testing Strategy

Version 1.0 [Draft]

John Collins
223617689

Contents

.....1
Version History.....4
Documentation Github Links.....4
Key Objectives of Software Testing.....5
Relationship to Quality Assurance.....5
Developer/ Data Science Software Test Responsibilities for QA.....5
Key Components of QA Testing.....6
Developing the Project Test Strategy.....6
Developing a Project Test Plan.....6
Test Results Reporting & QA Metrics.....7
Core Concepts of Software Testing Lifecycle.....7
Relationship Between Software Testing and Various Software Development Concepts.....9
Requirements and Design.....9
Software Architecture and Design.....9
Software Quality Attributes.....9
Appendix.....10
Types of Functional Testing.....10
Manual Testing.....10
Unit Testing:.....10
Integration Testing.....10
System Testing.....11
Acceptance Testing.....11
Regression Testing.....11
Types of Non-Functional Testing.....11

Version History

`	Changes by	Update Notes	Comment
1.0	John Collins	Initial Draft	Links need to be added.
1.1			

Documentation Github Links

TBA – link to Document in Github

Key Objectives of Software Testing

Software testing plays a pivotal role in ensuring the quality and reliability of software applications. Its primary objectives are:

Defect Identification and Resolution: Identifying and rectifying defects to ensure the software functions correctly and meets specifications.

Quality Assurance: Verifying that the software adheres to industry standards, best practices, and user expectations.

Proactive Problem Solving: Addressing potential issues early in the development process to prevent future failures and reduce costs.

Risk Mitigation: Minimizing the risk of software failures that could impact user experience or business operations.

Relationship to Quality Assurance

Software testing is an integral part of quality assurance. It provides a systematic approach to evaluating the software's quality attributes, including:

Functionality: Ensuring the software performs as intended and meets user requirements.

Reliability: Assessing the software's stability and dependability.

Performance: Evaluating the software's speed, responsiveness, and resource usage.

Usability: Ensuring the software is easy to use and understand.

Security: Verifying that the software is protected against vulnerabilities and unauthorized access.

Developer/ Data Science Software Test Responsibilities for QA

Aligned with the overall goal of re-usability and knowledge retention within the project, in particular for programmed activities required for the project highlighted priority use cases.

The following is an expectation regarding the items that need to be submitted work of each develop in a github folder, regardless of team level:

- **Manual Tests** for each function: Description, and screen shot sequence /video of operation (demo).
- **Unit Tests with Test Data Model** (Data fields/variables) exercising each individual programmed activity. (E,g Standalone Jupyter Notebook with Data Model CSV or Mocha/Chai (or equivalent test) with MongoDB CSV
- **Integration Tests** with test data exercising the interaction of programmed activity with (e.g code that exercises a GET/POST to googlemaps or CRUD for databases or runs a python script etc)
- Documentation of any contribution relevant to **System tests** e.g test environments local/cloud, api key permissions, links to technology stacks etc)
- Basic Documentation of the **Data Model (Data fields/variables) & Data Flow** with a simple explanation on how to run any relevant test.

Key Components of QA Testing

Effective QA testing relies on a structured approach that incorporates several essential components. These components work together to ensure high-quality and functional software applications.

- **Test Strategy:** This overarching plan outlines the principles and guidelines for the entire testing process. It provides a clear direction for the QA team to achieve testing goals efficiently.
- **Test Plan:** A detailed document that specifies the objectives, timeline, resources, and approach for a specific testing project. It serves as the blueprint for the testing effort.
- **Test Cases:** A set of specific inputs designed to verify software requirements. These cases provide step-by-step instructions for testers to follow.
- **Test Data:** The input provided to the software during testing. This data helps evaluate the software's behavior under various conditions, including positive and negative scenarios.
- **Test Scripts:** Automated sequences of instructions used to execute tests. They are essential for automating repetitive testing tasks and improving efficiency.

By effectively utilizing these components, QA teams can ensure that software applications meet quality standards, are reliable, and deliver a positive user experience.

Developing the Project Test Strategy

The strategy defines the testing types and approach of testing to be adopted across the project. The key components are as follows:

- **Define Overall Testing Approach:** Outline the overall testing approach, objectives, including types of tests, priorities, and resources.
- **Guide Testing Efforts:** The test strategy provides direction for the testing team.
- **Focus on Priorities:** Prioritize features (use cases) based on their importance to the application.
- **Test-Level Risks & Impacts:** It identifies the risks and potential impacts associated with testing at the end-user and organisational levels.
- **Assist in defining Testing and Developer Roles and Responsibilities:** Who does what? The test strategy clarifies the responsibilities of individual testers.
- **Standardisation of Test Execution Tools:** Which tools will we use for test execution? Automation frameworks, test management tools, etc.

Developing a Project Test Plan

A software test plan is a detailed document that outlines how testing activities will unfold during a project. It covers objectives, scope, resources, schedules, and the approach for validating the quality of a software product, and addresses the following elements:

- **Define Objectives:** Determine the goals of the testing effort.
- **Define Scope:** Clearly define what will be tested, by whom, and when.
- **Assess Application & Test Environment:** Define the hardware and software needed for testing, which the objective of mimicking the real-world application environment.
- **Select Testing Tools:** Choose appropriate tools for testing, automation or management.

- **Analyze Automation Feasibility:** Determine if certain tests can be automated.
- **Estimate Efforts:** Determine the time and resources required for testing.
- **Address Training Needs:** Identify training requirements for the testing team.
- **Document Test Plan:** Outline the testing objectives, approach, resources, dependencies, and timeline.
- **Update Regularly:** Regularly review and update the test plan as testing progresses.

Test Results Reporting & QA Metrics

QA metrics can be used throughout the testing process to guide decision-making, measure progress, and evaluate the effectiveness of testing activities. By analyzing and reporting on these metrics, teams can gain valuable insights and make data-driven improvements to the software development process.

- **Derivative Test Metrics:** These metrics (across manual, unit, integration, systems etc) help pinpoint specific areas within the testing process that require attention, allowing teams to focus their efforts on improving accuracy and overall effectiveness.
- **Percentage of Passed Test Cases:** This metric measures the success rate of test cases, indicating the effectiveness of the testing process in identifying defects.
- **Defect Category:** This metric analyzes the distribution of defects across different quality criteria, such as usability, performance, functionality, stability, and dependability.
- **Defect Severity Index (DSI):** The DSI measures the impact of defects on the software or specific components, helping to assess the quality of the software and the efficiency of the testing team.
- **Percentage of Critical Defects:** This metric calculates the proportion of defects classified as critical, highlighting the severity of issues that could impact the software's functionality, usability, or safety.
- **Test Coverage:** This metric evaluates the extent to which the software's functionality is tested, indicating the completeness of the testing process.
- **Defect Density:** This metric measures the number of defects per unit of measurement (e.g., lines of code, function points), providing insights into the overall quality of the software.

Core Concepts of Software Testing Lifecycle

The Software testing lifecycle is a structured process that outlines the key phases involved in software testing. It ensures that the software meets quality standards and requirements before release.

1. **Requirement Analysis:**
    - **Understand Requirements:** Familiarize yourself with user stories, requirements and design documents.

- **Identify Test Items:** Determine what needs to be tested based on requirements.
- **Define Test Requirements:** Specify functional and non-functional requirements for testing.
- **Link requirements to Test Cases to ensure traceability.**

2. Test Case Development:

- **Prioritize Critical Features:** Focus on testing the most critical aspects of the application.
- **Verify Test Case meets Requirements:** Ensure that test cases cover all functional and non-functional requirements.
- **Design Test Cases:** Document step-by-step instructions/actions to be performed on the software to detail test case that specify steps, expected outcomes, and prerequisites.
- **Create Test Cases:** Develop detailed test cases based on requirements.
- **Review Test Cases:** Ensure test cases are comprehensive and accurate.

3. Create Test Data:

- **Create Test Data for Test Cases:** Use a small simple sample of data that reflects real-world scenarios. Prepare data sets for testing.
- **Parallel Development:** Develop test data concurrently with test cases.
- **Avoid Modifying Real Data:** Protect sensitive user data by using test data.

4. Test Environment Setup:

- **Configure Hardware and Software:** Set up the necessary environment for testing that mimic real-world usage. E.g Local, Cloud. Test the software on various devices, browsers, and operating systems.
- **Prepare Test Data and Environments:** Configure the environment and load test data.
- **Install Applications:** Install required software components.
- **Conduct Smoke Tests:** Perform basic tests to verify the build's stability.

5. Test Execution:

- **Execute Test Cases:** Run test cases to exercise with the software to verify its behaviour against expected outcomes.
- **Document Test Outcomes & Issues:** Record test results, identify defects, or unexpected behaviour.
- **Communicate Issues:** Report defects to developers.
- **Analyze Defects:** Developers analyze and prioritize defects for resolution/fixes.
- **Retest Failed Cases:** Re-run failed tests to verify fixes.

6. Test Closure:

- **Generate Test Reports:** Create comprehensive reports summarizing test results.
- **Evaluate Test Results:** Evaluate the effectiveness of testing and identify areas for improvement. Assess test coverage, defect count, and overall objectives.
- **Close Defects:** Verify that resolved defects are fixed.
- **Archive Test Artifacts:** Store test plans, cases, and results for future reference.

By understanding and applying these core concepts, testers can conduct effective and comprehensive software testing to ensure high-quality products.

Relationship Between Software Testing and Various Software Development Concepts

Software testing plays a crucial role in ensuring the quality and functionality of a software application. It is closely intertwined with various aspects of the software development process, including:

Requirements and Design

- **Hi-Level Requirements:** Testing helps validate if the software meets the high-level requirements defined in the project's scope.
- **Product Vision:** Testing ensures that the software aligns with the overall product vision and goals.
- **User Stories:** Testing verifies that the software fulfills the requirements specified in user stories, ensuring it meets the needs of end-users.
- **Use Cases:** Testing validates that the software can handle the scenarios described in use cases, demonstrating its functionality and usability.

Software Architecture and Design

- **Information Architecture:** Testing helps evaluate how well the information is organized and presented to users, ensuring a seamless user experience.
- **Domain Class Models:** Testing verifies that the software's implementation aligns with the defined domain class models, ensuring correct data handling and relationships.
- **Data Flow:** Testing validates the flow of data through the system, ensuring it is processed and transformed correctly.
- **Data Model:** Testing ensures that the software interacts with the data model as expected, handling data storage, retrieval, and manipulation appropriately.
- **Software Architecture:** Testing verifies that the software's architecture is robust, scalable, and meets performance requirements.

Software Quality Attributes

- **Modularity:** Testing can assess how well the software is divided into manageable components, making it easier to test, maintain, and update.
- **Efficiency:** Testing can measure the software's performance and resource consumption to ensure it meets efficiency requirements.
- **Design:** Testing helps evaluate the user interface design, ensuring it is intuitive, visually appealing, and easy to navigate.
- **Accessibility:** Testing verifies that the software complies with accessibility standards, making it usable by people with disabilities.
- **GUI:** Testing evaluates the GUI's clarity, consistency, and responsiveness to user interactions.

By thoroughly testing the software against requirements, design specifications, and quality attributes, organizations can ensure that the final product meets user expectations, is reliable, and delivers value.

Appendix

Types of Functional Testing

Functional testing verifies that a software application operates as expected and meets its specified requirements. It focuses on ensuring that all features work correctly from the user's perspective, based on the defined functional specifications.

Manual Testing

Manual software testing is a quality assurance process that involves manually testing whereby testers interacting with the software directly, simulating user actions and observing the results.

Objectives:

- **Goal:** To identify defects, ensure functionality, and verify compliance with requirements.
- **Process:** Testers manually interact with the software, simulating user actions and observing the results.
- **Focus:** Thoroughly evaluating the software's behaviour under various conditions.

Unit Testing:

Unit testing is a granular level of software testing that focuses on verifying the correctness and functionality of individual units or components within the software architecture. These units typically consist of small code blocks like functions, methods, or procedures. Developers typically conduct unit testing during development.

Key Objectives:

- **Test Script Focus:** Unit tests are designed to exercise specific code blocks, isolating them from the rest of the application.
- **Function and Method Validation:** Ensure that each function or method produces the expected output for various input combinations.
- **Code Correctness Verification:** Verify that the code adheres to the specified requirements and performs its intended tasks accurately.

Integration Testing

Integration testing involves combining and testing multiple units or modules of a software application to ensure they interact correctly and work together as expected.

Key Objectives:

- **Identify Interface Issues:** Detect problems in how modules communicate and exchange data.
- **Validate Data Flow:** Verify that data flows correctly between modules.
- **Ensure Proper Interactions:** Ensure that modules interact as expected under various scenarios.

System Testing

System testing involves evaluating the entire integrated software system to ensure it meets all specified requirements correctly in a complete environment.
It focuses on verifying the system's functionality, performance, and user experience. System testing plays a critical role in ensuring that the software application functions correctly and meets the needs of its users.

Key Objectives:

- **Validate Requirements:** Confirm that the system meets its defined requirements.
- **Test User Experience:** Evaluate the system's usability and performance from a user's perspective.
- **Assess Quality:** Ensure the system meets technical and functional specifications.

Acceptance Testing

A comprehensive evaluation of the software from a user's perspective to ensure it meets their needs and expectations.

Regression Testing

Verifies that changes to the software have not introduced new bugs or impacted existing functionality, by re-running existing test.

By conducting these types of functional testing, teams can ensure that their software applications are reliable, meet user needs, and are ready for release.

Types of Non-Functional Testing

Non-functional testing evaluates aspects of software quality that are not directly related to specific features or functionality. It focuses on attributes such as performance, usability, reliability, and security.

- **Performance Testing:** Ensures that the software meets specified performance criteria, such as response time and throughput. It evaluates factors like network latency, database performance, and server load.
- **Usability Testing:** Evaluates the ease of use and user experience of the software. It involves observing real users interacting with the application to identify usability issues.
- **Security Testing:** Identifies vulnerabilities in the software to protect against security threats and ensure data confidentiality, integrity, and availability.
- **Load Testing:** Evaluates the software's ability to handle expected or peak loads, ensuring it remains stable and performs as intended under various conditions.

By conducting non-functional testing, teams can ensure that their software applications are not only functional but also perform well, are easy to use, secure, and reliable.