

Test Strategy By:

Bugvengers

* **Problem:**
* **Date**: 18 May 2022
* **Document Status**: Done
* **Author**: Prashant, Jaspreet, Rahul and Azim
* **Document Number**: Bugvengers v1.0

[CONFIDENTIAL]

Table of Contents

[1. Introduction 3](#_Toc103785423)

[1.1. Purpose 4](#_Toc103785424)

[2. Testing Targets 4](#_Toc103785425)

[2.1. In Scope 4](#_Toc103785426)

[2.2. Out of Scope 4](#_Toc103785427)

[3. Testing Approach 5](#_Toc103785428)

[3.1. Testing Phases and Activities 5](#_Toc103785429)

[3.1.1. Overview 5](#_Toc103785430)

[3.1.2. Test Initiation 5](#_Toc103785431)

[3.1.3. Requirements Analysis 6](#_Toc103785432)

[3.1.4. Test Planning 7](#_Toc103785433)

[3.1.5. Test Analysis and Design 7](#_Toc103785434)

[3.1.6. Test Execution 8](#_Toc103785435)

[3.1.7. Test Reporting 8](#_Toc103785436)

[3.1.8. Test Closure 9](#_Toc103785437)

[3.2. Testing Levels 9](#_Toc103785438)

[3.2.1. Unit Testing 9](#_Toc103785439)

[3.2.2. Component Testing 10](#_Toc103785440)

[3.2.3. Component Integration Testing 10](#_Toc103785441)

[3.2.4. System Testing 10](#_Toc103785442)

[3.2.5. System Integration Testing 11](#_Toc103785443)

[3.2.6. User Acceptance Testing 11](#_Toc103785444)

[3.3. Testing Types 12](#_Toc103785445)

[3.3.1. API Testing 12](#_Toc103785446)

[3.3.2. Risk Based Testing 13](#_Toc103785447)

[3.3.3. Exploratory Testing 13](#_Toc103785448)

[3.3.4. Regression Testing 13](#_Toc103785449)

[3.3.5. Smoke / Sanity Testing 14](#_Toc103785450)

[3.3.6. GUI Testing 14](#_Toc103785451)

[3.3.7. Performance Testing 14](#_Toc103785452)

[3.3.8. Compatibility Testing 15](#_Toc103785453)

[3.3.9. Usability Testing 15](#_Toc103785454)

[3.3.10. Accessibility Testing 16](#_Toc103785455)

[3.3.11. Internationalization and Localization Testing 16](#_Toc103785456)

[3.3.12. Static Testing 16](#_Toc103785457)

[3.3.13. Installation/Operation/Performance qualifications 17](#_Toc103785458)

[4. Test Schedule and Milestones 18](#_Toc103785459)

[5. Test environments 18](#_Toc103785460)

[6. Defect Management 20](#_Toc103785461)

[6.1. Defect life cycle 20](#_Toc103785462)

[6.1.1. Defect reporting 21](#_Toc103785463)

[6.1.2. Defect evaluation 23](#_Toc103785464)

[**6.1.3.** Defect fixing 24](#_Toc103785465)

[6.1.4. Defect validation 24](#_Toc103785466)

[**6.1.5.** Defect closing 25](#_Toc103785467)

[6.2. Defects Monitoring and Controlling 26](#_Toc103785468)

[7. Change Management 26](#_Toc103785469)

[7.1. Definitions 26](#_Toc103785470)

[7.2. Change Request lifecycle 26](#_Toc103785471)

[8. Testing deliverables 26](#_Toc103785472)

[8.1. Test Strategy 26](#_Toc103785473)

[8.2. Test Plan 26](#_Toc103785474)

[8.3. Test Cases 27](#_Toc103785475)

[8.4. Traceability Matrix 27](#_Toc103785476)

[8.5. Test Report 27](#_Toc103785477)

[9. Testing roles and responsibilities 27](#_Toc103785478)

[10. Testing assumptions and risks 28](#_Toc103785479)

[11. Tools set 31](#_Toc103785480)

# 

# Introduction

The approach of creating this document for ABC Bank is to write high level testing approach

Different types of testing to cater the problem statement

Our testing KPI’s if we are using scrum model of delivery

## Purpose

This document is produced for use by ABC Banks IT team and assumes proper understanding of the project scope and deliverables.

List of reviewers and personnel to sign-off:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Role | | Signature |
| Prashant Chauhan | Team Member |  |  |
| Jaspreet Kaur | Team Member |  |  |
| Rahul Mathur | Team Member |  |  |
| Mohd. Azim | Team Member |  |  |
|  |  |  |  |

**Customer Company**

|  |  |  |
| --- | --- | --- |
| Name | Role | Signature |
| ABC Bank | Customer |  |
| ABC Bank IT Team | Customer |  |

# Testing Targets

Provide a high-level overview of the main functionalities and business requirements that will have to be covered during testing. Keep the focus on both functional, and non-functional requirements.

## In Scope

Functional and Non-Functional Testing

## Out of Scope

No explicit request or approval from the client side

Features in scope only for future versions

Commercial off-the-shelf components or products

Platforms limitation

# Testing Approach

[The Test Approach presents the recommended strategy for designing and implementing the required tests, describing how the tests will be realized.

An important aspect to consider for the test approach is the techniques to be used. This should include an outline of how each technique can be implemented, both from a manual and/or an automated perspective, and the criterion for knowing that the technique is useful and successful. For each technique, provide a description of the technique and define why it is an important part of the test approach by briefly outlining how it helps achieve the Evaluation Mission or addresses the Test Motivators.

The strategy is based on requirements and risk-based testing in order to minimize the effort and increase the efficiency of the testing process.]

## Testing Phases and Activities

[The section describes the tailored Testing Process to be applied on the project. Highlight the customizations of the process phases derived from the applied software development methodology.

Define the inputs and outputs of the phase, where appropriate.]

### Overview

[Provide the list of main phases and activities of the tailored Testing Process. In the following paragraphs, detail each phase, using either the table format, or the free text one. In case certain phases are not applicable provide a justification of it.]

The testing process applied on project will consist of the following main phases and activities:

* Test Initiation
* Requirements Analysis
* Test Planning
* Test Analysis and Design
* Test Execution
* Test Reporting
* Test Closure

### Test Initiation

|  |  |
| --- | --- |
| Test Initiation | |
| Purpose | This phase is oriented to achieve an agreement between the Customer, on one side, and the Project Manager and the Test Manager on the other side, on the provided Requirements, the expectation for the testing scope, and the procedural approach that is to be taken to implement the testing of the project. |
| Activities | * Check the SOW / Contract / Agreements * Cover project expectations and needs, communication channels, client expectations, deadlines, agreed testing deliverables, etc. * Discuss deadlines and priorities * Set the required skills * Set the deliverables/output * Read lessons learnt from other projects; check what can be applied * Do the risks assessment, and raise concerns * Create the Test Strategy: set the testing scope and objectives, set the testing types and levels, define roles and responsibilities, complete chapters where information is available |
| Entry Criteria | * SOW / Contract is signed * Requirements available |
| Exit Criteria | Test Strategy – first draft |

### Requirements Analysis

|  |  |
| --- | --- |
| Requirements Analysis | |
| Purpose | * During this phase, the Requirements are carefully read and analyzed. * This phase helps to refine the scope of the testing.   [If any feature does not appear as testable, it should be communicated in a timely manner, so that the mitigation strategy can be planned]   * The Input consists in the set of provided Requirements, and User Acceptance Criteria, if available, and the very first draft of the Test Strategy. * The Output is represented by the updated draft version of the Test Strategy, containing among other relevant sections, the Acceptance Criteria and the Risks List. In case of complex and/or critical acceptance criteria one can dedicate them a whole sub-section. |
| Activities | * Review and clarify all Functional and Non-functional Requirements * Identify and analyze User Acceptance Criteria for deliverables verification   [If not clearly stated in the initial Requirements, define a reasonable set of Acceptance Criteria based on those Requirements, and agree them with the client.]   * Define additional Acceptance Criteria   [Cover topics like proper event sequence, data and control flow, planned resource allocation, fault handling, initialization of variables, self-diagnostics, memory management, overflows, boundary conditions, where appropriate.]   * Perform the Testing feasibility analysis   [Find out whether the requirements are testable or not; have brainstorming sessions with other teams, if needed]   * If project requires automation, perform the Automation feasibility analysis * Provide rough estimations for the testing effort * Update the draft version of the Test Strategy |
| Entry Criteria | * All agreed Requirements documents are available * The first draft of the Test Strategy available |
| Exit Criteria | * Results of the static analysis on Requirements, including Q&A * Risks identified and addressed * Testing feasibility analysis done * Updated Test Strategy |

### Test Planning

|  |  |
| --- | --- |
| Test Planning | |
| Purpose | During the planning phase there are identified:   * the test procedures based on standards and guideline; * test data requirements; * hardware, software, and network required to support test environment; * non-functional measurements; * a preliminary test schedule; * defect-tracking procedures and associated tracking tools; * a procedure to control test configuration and environment; |
| Activities | * Define the scope of testing in accordance to project phase * Plan the testing activities / testing types, according to the release process based on the schedules and milestones * Do the risk analysis and prepare the risk mitigation plan * Identify dependencies for testing * Identify test environments |
| Entry Criteria | * All agreed Requirements documents are available * Architecture is defined * Test Strategy is available |
| Exit Criteria | * Risk analysis and mitigation done * Test Plan ready and approved * Updated Test Strategy |

### Test Analysis and Design

|  |  |
| --- | --- |
| Test Analysis and Design | |
| Purpose | Test analysis and design transforms the general testing objectives into tangible test conditions and test cases, clarifying *what* must be tested, and *how* exactly. |
| Activities | * Analyze in detail the Requirements, including Q&A, for the features in scope * Generate Test Ideas * Gather, generate, update the Test Data set: * analyze the system inputs and outputs stated in the Requirements * identify data characteristics (ranges, limits, default values), database requirements * establish the Test Data needs * mention and motivate the Test Data selection techniques used in the process * Create, review, update the Test Cases, and appropriate Test Scenarios * Create and update the Traceability Matrix * Identify the candidate Test Cases for regression and automation * Create, review and update the automation scripts * Take sign-off of the Test Cases and Test Scripts * Identify the test environments needs and address them |
| Entry Criteria | * All agreed Requirements documents are available * Test Strategy / Plan documents |
| Exit Criteria | * Test Cases and Test Scenarios finalized and approved * Test Data definition and mapping done * Test Scripts (Automation) ready * Traceability Matrix is created |

### Test Execution

|  |  |
| --- | --- |
| Test Execution | |
| Purpose | Test execution is the activity of running the application and comparing the expected results with actual results using various test methods and test data, deviations being documented in defects. |
| Activities | * Prioritize the test execution * Execute Test Cases as planned * Compare the expected results with actual results * Defects are reported * Fixed defects are retested * Record and track the test execution results and coverage * Make sure the name of the tester and the date of testing are recorded * Check the planned testing scope vs. execution * Test environments are maintained (cleanup, restore, backup configurations) |
| Entry Criteria | * Test Cases and Test Scripts available * Test data identified and mapped to the test cases * Test environments have been setup as per the application requirements * All required access is provided for the team members |
| Exit Criteria | * All in scope test scenarios / test cases / test scripts are executed * All identified defects are reported * Test execution coverage is documented |

### Test Reporting

|  |  |
| --- | --- |
| Test reporting | |
| Purpose | * Test reporting is the activity where test execution is assessed against the defined objectives from Test Strategy. * Software quality and readiness are evaluated, documented and communicated based on the agreed quality measures (outcome of the test results, number and severity of known defects, criticality of residual risks, etc.). |
| Activities | * Assess the Test Results. These must clearly identify the Tester who has performed the tests and the Date of testing. * In case the project imposes it, the Test Evidences will be stored on Project Repository. * Review existing opened defects. Create the List of Known Issues by exporting from the Defect Management tool the list of open defects and include it in the deliverables, either in the Test Report or in the Release Notes. * Provide release recommendations. * Create Test Report for stakeholders as agreed. If no specific requests from the client, the Test Report Template is going to be used. * Good practices to be documented. |
| Entry Criteria | The reporting level of detail, frequency and the format are agreed with the stakeholders. |
| Exit Criteria | All agreed Test Reports are created and delivered, including the release recommendation statement. |

### Test Closure

|  |  |
| --- | --- |
| Test closure | |
| Purpose | * Test closure activities collect data from completed test activities to consolidate experience. * Check for the completion of all the deliverables. |
| Activities | * Review existing opened defects and provide a resolution. * Archive repository with project artifacts (documents, emails, tools, etc.) and share it with relevant people. * Keep lessons learnt meeting, and create lessons learnt document.   [Include what went well, where are the scope of improvements and what can be improved]   * Update Good Practices Repository. |
| Entry Criteria | * Testing scope is completed, or * The project is ended/canceled. |
| Exit Criteria | The archive repository is available and contains all agreed project artifacts. |

## Testing Levels

[Outline the different levels of the software development lifecycle where testing is conducted.

For each level of testing, define the Roles that carry the responsibility, the Environment in which the tests will be performed, and their Scheduling. Set for each level the appropriate Entry Criteria and Exit Criteria.

Justification to be provided for not applicable levels.]

### Unit Testing

|  |  |
| --- | --- |
| Unit testing | |
| Description | * Unit testing is a white box testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use. * The Unit Testing is not in-scope for the Testing Team as the Unit tests are written and executed by software developers, to make sure that code meets its design and requirements and behaves as expected. |
| Dependencies | None |
| Approach | * Each module needs to be tested individually to verify proper operation, so that any failed test to be evaluated and fixed. * Special techniques, like Test Driven Development, can be used for increased efficiency. |
| Environment | Development |
| Entry Criteria | * Detailed Design available * Unit/Module Code developed |
| Exit Criteria | No failed unit tests |
| Roles | Software Engineer |

### Component Testing

|  |  |
| --- | --- |
| Component testing | |
| Description | Also called as module testing, component testing searches for defects in, and verifies the functionality of software modules or components, that are separately testable, as early as possible, before their integration. |
| Dependencies | As it is usually done in isolation from the rest of the system, stubs, mocks, drivers and simulators might be needed. |
| Approach | Each component is tested in isolation from the rest of the system. |
| Environment | QA |
| Entry Criteria | * Component Requirements available * Detailed Design available * Component Code developed |
| Exit Criteria | No open *Blocker* or *Critical* defects |
| Roles | Test Engineer |

### Component Integration Testing

|  |  |
| --- | --- |
| Component integration testing | |
| Description | Testing the interactions between several components, before having in place the entire functionality planed for the system.  Usually, the integrated components come from the same system, but there are cases when components from different systems are involved. |
| Dependencies | Components to be integrated are functional as standalone units. |
| Approach | The testing scenarios should capture the proper interaction between the tested components. |
| Environment | QA |
| Entry Criteria | * Functional Requirements available * System Design, covering the analyzed components, available * Each integrated Component passed Component Testing level |
| Exit Criteria | No open *Blocker* or *Critical* defects |
| Roles | Test Engineer |

### System Testing

|  |  |
| --- | --- |
| System testing | |
| Description | Testing all components of a single system, working integrated and according to the requirements. Its purpose is to detect potential inconsistencies in both design and behavior. |
| Dependencies | Integrated components that have passed the Component Integration Tests |
| Approach | * It is performed over all available parts of the system * The focus is on both functional, and non-functional aspects * The end-to-end flows of the system are covered as much as possible |
| Environment | * QA / Staging |
| Entry Criteria | * System Requirements are available – both Functional and Non-functional * System Design is available * Test Cases are available, reviewed and approved * Test Data is available in the defined range * Test Environment is available * Mocks are implemented, where needed |
| Exit Criteria | * No open Blocker or Critical defects * Plan for handling open defects is documented * All planned Test Cases are performed, justification is provided for the unexecuted ones * Test Report is created |
| Roles | * Test Engineer * Test Manager |

### System Integration Testing

|  |  |
| --- | --- |
| System integration testing | |
| Description | Testing the interactions between different systems of the implemented solution. The aim is to cover the end-to-end flows for all integrated systems, by maintaining data integrity and all dependencies being functionally correct. |
| Dependencies | Systems to be integrated are functional as stand alone. |
| Approach | * Big-bang method integrates all the components simultaneously, everything being tested as a whole. This can save time but when a defect is found it might require more time to investigate its cause. * Top-down is when the top integrated components are tested first, and then following the architectural structure it is tested step by step until the last component * Bottom*-up* approach starts with the lowest components, and then these are used to facilitate the testing of higher-level components |
| Environment | Staging |
| Entry Criteria | * Requirements documentation is available * Solution readiness for system integration testing * System testing level is passed for all involved systems * Test cases and test scripts are available * Staging environment with all external systems integrated is available * Production-like Test Data is available, in the defined range |
| Exit Criteria | * No open Blocker or Critical defects * Plan for handling open defects is documented * All planned Test Cases are performed, justification is provided for the unexecuted ones * Test Report is created |
| Roles | * Test Engineer * Test Manager * Vendor * Solution Architect |

### User Acceptance Testing

|  |  |
| --- | --- |
| User acceptance testing | |
| Description | This is the final testing phase that verifies, using real life scenarios, if the business requirements were properly implemented. It focuses on the usability of the product. |
| Dependencies | Environment shall be reserved for this activity only to ensure that the outcome is not impacted by other parallel activities. |
| Approach | * Vendor responsible person(s) or selected end-user(s) support the definition of the test scenarios from the business perspective, and these are run with support from the Test Engineers and Business Analysts * User acceptance criteria are defined and verified during this level * The UAT scenarios are run on a production-like environment * UAT final report is created/updated with recommendation for go live/no go decision * All defined parties agree and sign-off the UAT completion |
| Environment | Staging / Production |
| Entry Criteria | * Solution is ready for UAT, meaning all planned features are implemented and tested * System Integration Testing is finalized with no Blocker or Critical defects * List of known defects is available * Production-like environment with all external systems integrated is available * Production-like data is available, in the defined range |
| Exit Criteria | * No open Blocker, Critical or Major defects (according to product go-live agreement) * List of known defects is available * UAT document is signed-off |
| Roles | * End User * Business Analyst * Test Engineer |

## Testing Types

[Describe the types of testing planned to be applied on the product under evaluation, in order to evaluate its quality and its behavior.

The below described test types can be used in one or both functional and non-functional testing. Functional testing is conducted to evaluate the compliance of a system or component with specified functional requirements. Non-functional testing is conducted to evaluate the compliance with the non-functional requirements of the system such as performance, usability, security, reliability, etc.

Note: the list of the testing types is an exemplification, not complete nor limited, shall be updated based on project specifics.]

### API Testing

|  |  |
| --- | --- |
| API testing | |
| Description | Tests the [application programming interfaces](https://en.wikipedia.org/wiki/Application_programming_interface) directly and as part of [integration testing](https://en.wikipedia.org/wiki/Integration_testing), to determine if they meet expectations for functionality, reliability, performance and / or security.  Can be employed in both functional and non-functional testing |
| Dependencies | API specification is available |
| Approach | * Identify parameters and their combinations, boundaries, value types, etc. * Proper call sequencing is required, as this may lead to inadequate coverage in testing * Verify and validate the response, analyzing data accuracy, HTTP status code, response time, error codes, authorization checks * It can be partially or completely automated * Can be performed by executing scripted tests or by performing exploratory testing using ad-hoc parameter values |
| Roles | Test Engineer |

### Risk Based Testing

|  |  |
| --- | --- |
| Risk Based Testing | |
| Description | Type of software testing based on tests prioritization, derived mainly from the requirements risk assessment, considering the risk of failure, importance, likelihood and impact. |
| Dependencies | Requirements risk assessment is required |
| Approach | * Identify, analyze and evaluate risks related to the software requirements under test (requirements risk assessment) * All software requirements are sorted based on the identified levels of risks * Testing scope might be limited to include only requirements with a certain level of risk * Test design and execution for requirements under test is prioritized, starting the test coverage from the highest risk software requirements |
| Roles | Test Engineer |

### Exploratory Testing

|  |  |
| --- | --- |
| Exploratory testing | |
| Description | Exploratory testing implies simultaneous learning, test design and test execution. |
| Dependencies | None |
| Approach | * Interact with the application in whatever way the tester wants * Different test design tools are used (e.g. mind maps, workflows, screenshot tools) * Use the information the application provides to react, change course, and generally explore the application’s functionality without restraint * Should be performed as soon as functionality is available * Should generate a high-level description of the workflows / functionalities / scope covered |
| Roles | * BA – provides functional and context knowledge to the tester * Test Engineer |

### Regression Testing

|  |  |
| --- | --- |
| Regression testing | |
| Description | Regression testing verifies that a code change in the software does not impact the existing functionality of the product. Previously executed test cases are partially or entirely re-executed in order to verify the impact of change. |
| Dependencies | * History of the product and its functionality is available * Historical test cases / scenarios are available |
| Approach | The regression test scenarios are created from existing functional/non-functional tests that were agreed and already have been executed prior to this period |
| Roles | Test Engineer |

### Smoke / Sanity Testing

|  |  |
| --- | --- |
| Sanity testing | |
| Description | Smoke / sanity testing is the subset of Regression testing to evaluate overall testability of the product. |
| Dependencies | New build is available |
| Approach | * It can be performed at any project phase * Tests are usually scripted * Should be relevant, avoiding thoroughly or in-depth analysis * Test execution timeframe should be short, time-boxed |
| Roles | Test Engineer |

### GUI Testing

|  |  |
| --- | --- |
| GUI testing | |
| Description | Graphical User Interface testing is the process of testing a product's graphical user interface to ensure it meets its specifications. It is also to determine that the software meets the UI/UX requirements/design and ensures that all the flows, interactions, content, etc. behave as expected |
| Dependencies | UI/UX specifications are available |
| Approach | * Design elements like links, colors, fonts, font sizes, fields etc. are displayed as specified * GUI user actions are performed against specifications |
| Roles | * Test Engineer * UI Designer * UX Specialist |

### Performance Testing

|  |  |
| --- | --- |
| Performance testing | |
| Description | Performance testing is a non-functional testing technique performed in order to determine the system behavior regarding time, throughput or volume under a specific workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.  Below are the most common performance testing types:   * Load test – verify application behavior under normal and anticipated peak load conditions * Stress test – is about overloading things until they break, applying unlikely load scenarios * Volume test – refers to testing a software application with a certain amount of data * Scalability test – tests the ability of the system to cope with volume or load increase during operation |
| Dependencies | * Performance requirements are available * Features in scope are stable * A dedicated performance production-like environment is up and running * Production-like test data is available |
| Approach | * A Performance Testing Strategy/Plan document can be created separately and could contain the following test activities: run tests, monitor the test environment, analyses the results, tune the tests, retest, KPIs, etc. * Analysis and design for performance testing is done * Dedicated performance testing tools shall be employed – see * Tools set * It is recommended to be performed as soon as the dependencies are met |
| Roles | * Test Engineer * System Administrator (for test environments support) |

### Compatibility Testing

|  |  |
| --- | --- |
| Compatibility testing | |
| Description | Compatibility testing is a non-functional testing conducted on the application to evaluate the application's capability of running on different environments/ hardware/ devices/ operating systems/ web browsers etc. |
| Dependencies | None |
| Approach | * Define scope and limit number of OS/ Browsers/ devices combinations to a realistic timeframe * The selection of OS/Browser platforms can be done based on google metrics when client’s request is not specific * The BA has the role of prioritizing the found defects between the environment/browser, as he is the person having the most contact with the Product Owner |
| Roles | * Test Engineer * Business Analyst |

### Usability Testing

|  |  |
| --- | --- |
| Usability testing | |
| Description | The Usability Testing is a technique used in user-centered interaction design to evaluate a product by testing it on users. It gives direct input on how real users use the system. Usability experts review brings valuable input in product quality assessment. |
| Dependencies | None |
| Approach | Perform a heuristic evaluation of the interface based on usability principles, like:   * Visibility of system status * User control and freedom * Consistency and standards * Error prevention * Recognition rather than recall * Flexibility and efficiency of use * Aesthetic and minimalist design * Help and documentation |
| Roles | * Test Engineer |

### Accessibility Testing

|  |  |
| --- | --- |
| Accessibility testing | |
| Description | The Accessibility testing is a subset of Usability Testing performed to ensure that the application is usable by people with disabilities or with special needs, or by other disadvantaged groups by using appropriate technology assistance. |
| Dependencies | Accessibility standard WCAG 2.1 or US Section 508 knowledge required |
| Approach | * Identify main target users for accessibility testing * Define accessibility requirements and context * Recommended checklist: screen readers testing, sounds, navigation, colors, readability of the text, zooming the pages * Use dedicated tools where possible |
| Roles | * Test Engineer |

### Internationalization and Localization Testing

|  |  |
| --- | --- |
| Internationalization and Localization testing | |
| Description | * Internationalization and Localization are means of adapting computer software to different languages, regional peculiarities and technical requirements of a target locale. * Internationalization is the process of designing a software application so that it can be adapted to various languages and regions without engineering changes. * Localization is the process of adapting a given software for a specific region or language, by translating text and adding locale-specific components. * Pseudo localization testing checks the internationalization aspects of a software using only altered versions of the original textual elements instead of full translations in foreign languages. |
| Dependencies | * The target regions/countries/languages are defined * The translations are available; if missing, pseudo localization is recommended |
| Approach | * The specific characters for each language are used for testing * All the differences are known and checked with the culture and language expert * Checking localization resource files could be involved * Location specific things can be checked: install/uninstall process, UI, functionality related to currency, time and date format, legal aspects requirements, documentation, multimedia, support fonts and chars, text rendering |
| Roles | * Test Engineer * Language/Culture Expert * Client |

### Static Testing

|  |  |
| --- | --- |
| Static testing | |
| Description | The primary goal of static testing is to identify as early as possible issues in the software development process by documentation and software code reviews in the following two major categories:   * Review – Typically used to find and eliminate errors or ambiguities in documents such as Requirements, Design, Test Cases, User Manuals, etc. Performed by Business Analysts and Test Engineers. * Static analysis or Code review – The code written by Developers is analyzed for structural defects that may lead to functional defects. Performed by Developers, manually or with specialized tools. |
| Dependencies | None |
| Approach | * The review approach is defined. * Documents are reviewed in order to identify possible defects. * Found defects are addressed to be fixed, documents being modified accordingly. * Final review is required before approval. |
| Roles | * Test Engineer * Business Analyst * Developer |

### Installation/Operation/Performance qualifications

|  |  |
| --- | --- |
| Installation/Operation/Performance qualifications | |
| Description | * The *Installation Qualification* represents a security measure to ensure and document that the system is installed as specified * The *Operational Qualification* represents a security measure to ensure and document that the system is operational. It documents that a system meets the defined functional requirements, or that the system does what it’s supposed to do. * The *Performance Qualification* represents a security measure to ensure and document specified operational performance of the system. |
| Dependencies | * The system under test is changed and it is brought into operational mode. * The software full functionality was previously tested. |
| Approach | A limited number of predefined and approved Test Cases are executed due to various changes of the system like (but not limiting to):   * Security patching * Software configuration changes * Service patching * Hardware relocation * Hardware update/change * Software re-deployment * Software/Hardware parameter tuning |
| Roles | * System Administrator * Test Engineer * Managed Service team |

# Test Schedule and Milestones

[Define the testing schedules and milestones – if different of the project schedule/milestones, depending on the project model – iterative, agile, waterfall. These are meant to help proper guiding and directing of the testing effort.

In case the timelines are defined in the project documentation/ project plan, the corresponding Chapter can be referenced, or data can be replicated. If the project milestones are not fit to be used, feel free to use any scheduling tool/format.]

# Test environments

[The testing environment setup is closely related to the project needs, timeline, testing levels to be covered and client commitment. It's up to the project team to decide the best approach on which environments need to be used. Detail the specific environments needed to be deployed in order to make sure the testing will react in a flexible manner and will provide reliable results.]

The environments strategy identified four areas of testing environments.

* **Development environment** – represent the environment on which development teams are working and testing the developed code. In many situations this environment is also linked with the Continuous Integration solution where applications are deployed automatically and on which continuous builds are done.
* **Test environment –** represents the environment where Test Teams are running different testing levels and types depending on what is agreed at the project level. This environment is a more stable environment and where releases are deployed based on an agreed release management procedure. Not all the Production systems or configurations are available on this environment, but it should have more systems/modules installed versus the Development environment.
* **Staging environment –** represents the pre-production environment and this is the most stable environment from all the environment's used within the service delivered. This needs to be a copy of the production environment and is the environment where UAT, for example, can take place. The staging data should mirror latest production data available. The testing can be conducted directly in staging or production if the complexity of the project is high. It is a good practice to request the staging environment as quickly as possible.
* **Production environment –** represents the live environment. The installation/operation/performance qualifications are conducted here after the deployment or update of the product.



|  |  |  |
| --- | --- | --- |
| Environment | Responsible | Access |
| Development | Development team:  Technical lead is the main contact, if no other designated person is responsible for this environment  Decides what needs to be deployed, what unit tests are run, how continuous integration is performed | **Unrestricted**  Should be easy to grant access by development team to the involved team members to hardware resources or credentials to tools/services. |
| Test | Testing team:  Test manager/Test lead is the main owner  Builds are delivered as agreed  Can install/uninstall systems, deploy builds, do configurations  Manages local testing setup per project needs: hardware, devices and software  Test execution is performed | **Unrestricted**  Should be easy to grant access by test team to the involved team members to hardware resources or credentials to tools/services.  No restriction should exist for testers on the environments. |
| Staging | Testing team/Managed Services/Client  Usually, it’s a shared environment and responsibility between test team, client and other parties  Schedule builds delivery – can impact others work or overlap with other project activities | **Have some restriction**:  Access might be reduced – not all machines or tools have full access rights to system/ logs/ configurations.  Resources access should be granted for 3rd parties if needed. |
| Production | Client/ Client IT  Client decides when to deploy based on a schedule | **Restricted**  Any authorization requires request to the client’s IT department in charge.  Testing needs to determine the level of access they need. |

# Defect Management

[Describe the tailored process by which the discovered defects will be managed throughout the entire project. Once agreed, the process should be loosely followed by all team members, PM, developers, testers, and customers and end users. To be detailed here the defects life cycle, the defect management tool to be used, the defects fields and their list of values, where appropriate.]

The defect management process includes defect discovery and resolution, defect causal analysis, and process improvement.

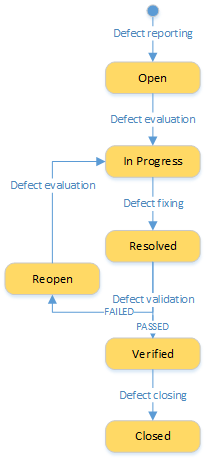
All defects will be opened accordingly based on the defect management tool selected in the organization.

All relevant information about the defects during its life cycle is tracked and stored using the defect management tool.

## Defect life cycle

At a high level look the defect life cycle will include the following phases:

* Defect reporting
* Defect evaluation
* Defect fixing
* Defect validation
* Defect closing



### Defect reporting

It is essential for the *Reporter* to be thorough in filling out the ticket and to accurately describe how to replicate the defect so that further communication between tester and developer is not required.

Thorough reporting responsibility is not restricted to Test Engineers. Anyone filing a defect must follow the rules described in this section.

Here are the fields that require attention whenever a new defect is submitted:

|  |  |
| --- | --- |
| Field | Description |
| Issue Type | Describes the type of entry which can be tracked:   * Bug * Improvement * Change Request |
| Summary | Describes shortly and clearly the problem raised by the defect. The summary should follow the agreed rules on the project. |
| Severity | Describes the impact of the defect on the application behavior.  The severity is set based on application business, taking into consideration how the problem affects the usage of the application.  Possible values:   * *Blocker*: errors which have a severe impact; an error that causes the application to crash or end, so that the testing cannot go further * *Critical*: crashes, loss of data, severe memory leak of the app; an entire functionality does not work, or a defect with a critical business impact with no effective workaround available * *Major*: an area of functionality which does not work, or a defect with a major business impact; effective workaround might be available * *Normal*: loss of functionality in secondary cases, with limited business impact * *Minor*: UI minor loss of functionality, or other problem where easy workaround is present |
| Priority | Describes the urgency to fix the defect.  Additional to severity, a priority value can be set for a defect in order to speed up certain required fixes, even if they have not high severities.  Possible values:   * *Urgent:* it is extremely important, and require immediate action, being business critical/urgent * *High*: it is important, but may not require immediate action, as it is not business critical/urgent * *Medium*: it is important, but do not require action to be taken on high priority * *Low*:it has little business impact, and requires attention when there are no other higher priorities   The default value is Medium. |
| Components | Describes the area/component for which the defect is found. |
| Affect Versions | Describes the version in which the defect was found.  It must be completed by the person who reports the defect.  This information is needed in order to have a history related to on which build the defect was raised.  In case of defect’s reopening, the “affect version” should be updated. |
| Assignee | The person who is responsible for doing the defect specific activities. |
| Reporter | The person who raised the defect. This is automatically set. |
| Environment | Describes the environment on which the defect was found. Details like operating system, browser, and versions are provided.  This information can be also provided on multiple fields if the project requires these fields to be separated. |
| Description | Contains information clearly described to be able to reproduce the defect. It should have the following format:   * *Steps to reproduce* describe the succession of actions to be executed to reproduce the defect * *Expected results* describe the desired behavior of the system while performing the above steps in accordance with the system requirements * *Actual results* describe the existing behavior of the system while performing the above steps * *Notes* are optional, containing any additional relevant information |
| Status | The phase the defect is currently at, in its lifecycle:  Possible values:   * *Open* – is set automatically at creation and the defect is ready to be evaluated (Resolution – *Unresolved*) * *Blocked* – the defect processing cannot move forward due to missing vital details (Resolution – *Incomplete*) * *In progress* – the defect is actively worked on by the assignee (Resolution – *Unresolved*) * *Resolved –* a resolution has been identified and implemented (possible Resolutions – *Fixed*, *Won’t Fix*, *Duplicate*, *Incomplete*, *Invalid*, *Cannot Reproduce*) * *Reopened* – the defect is reproducible again and needs re-evaluation and fixing (Resolution – *Unresolved*) * *Verified –* the defect was validated by the testing team, but it might require further input from other parties (e.g. client, business users, stakeholders) (possible Resolutions – *Fixed*, *Won’t Fix*, *Duplicate*, *Incomplete*, *Invalid*, *Cannot Reproduce*) * *Closed –* The defect does no longer require attention (possible Resolutions – *Fixed*, *Won’t Fix*, *Duplicate*, *Incomplete*, *Invalid*, *Cannot Reproduce*) |
| Resolution | The way in which the defect is resolved:   * *Unresolved* – the defect is not fixed * *Fixed* – a fix for this defect has been implemented * *Won’t Fix* – the defect is valid, but not going to be fixed * *Duplicate* – the defect is a duplicate of an existing defect (an appropriate Link should be added) * *Incomplete* – not enough information to work on this defect * *Invalid* (Issue) – the defect is not valid * *Cannot Reproduce* – the defect cannot be reproduced |
| Attachments | Contains the attachments to the defect that bring additional information for helping the developer in the investigation and fixing process (e.g. warning messages, screenshots, error logs) |

### Defect evaluation

All new submitted defects are evaluated if they need to be fixed within the current release or scheduled for future releases. This evaluation is done during a screening meeting as agreed on the project level. The defects needed to be fixed within a release are prioritized, the *Fix Version/s* is set, and the defect is assigned to a developer who can start working on it.

During the evaluation phase a defect can also be set directly to *Resolved* status. If the defect is not detailed enough, it should be reassigned to the owner to add all the missing information. The resolution in this case is *Incomplete*.

If the defect is not going to be fixed the resolution is set to *Won’t fix*. If the defect cannot be reproduced the resolution is set to *Cannot Reproduce*. These resolutions can be discussed, decided and documented for further steps during the screening meetings.

When evaluating a defect, the following fields are updated:

|  |  |
| --- | --- |
| Field | Change |
| Status | The status can be changed to *Resolved*, if the defect is not going to be fixed.  The status is not affected by the defect assigning to a Developer/ BA/ User. |
| Resolution | Resolution is changed if the defect is not going to be fixed. Possible values: *Won’t Fix, Duplicate, Incomplete, Invalid, Cannot Reproduce* |
| Assignee | The defect is assigned to the Developer who can start working on the defect |
| Fix Version | The field is updated to reflect the version / build in which the defect is going to be fixed and can be validated |
| Comment | Information relevant for the developer should be added here |

### Defect fixing

The new assignee of the defect investigates the defect, and if it is still reproducible and valid, will fix it.

When fixing a defect, the following fields are updated:

|  |  |
| --- | --- |
| Field | Change |
| Status | While working on the defect the Status is set to *In progress*.  After the defect is fixed, the status is changed in *Resolved.* |
| Resolution | Resolution is set according to the provided solution. Possible values: *Fixed* (most of the cases), *Won’t Fix, Duplicate, Incomplete, Invalid, Cannot Reproduce* |
| Assignee | The defect is assigned to the agreed responsible (e.g. Reporter, Test Lead) |
| Fix Version | The field is updated to reflect the version/build in which the defect was fixed |
| Comment | The information about how the defect was fixed is provided. Any information relevant to the tester for better testing coverage is added here. |

After fixing, the resolution of a defect must be very well documented in order to transfer the knowledge back to the testing team who will review and retest the defect.

### Defect validation

When defect Resolution is marked as *Fixed*, the Test Lead assigns it to a member of the test team to validate it. The assigned Tester, besides the steps described by the defect, does a selection of flows that cover what was fixed and a selection of additional tests that aims to expose side effects.

Based on the defect Resolution, the Tester takes specific actions:

|  |  |
| --- | --- |
| Resolution | Actions |
| Fixed | Whenever the fix is accepted and no side effects are identified, the Tester closes the defect with comments regarding the tests performed, on which version and environment. The defect *Status* is changed to *Verified* with *Fixed* resolution.  In cases when the delivered fix does not cover all defects raised or does not solve all the problems:   * either the defect is Reopened and assigned back to the previous assignee who fixed the defect, * or the current defect is closed with relevant comment, and a new one is raised for the remaining aspects.   The tester should provide additional details/scenarios that highlight how the fix does not solve raised problems. The added details should be well documented in the ticket. The *Fix Version* field value should be reset.  Whenever side effects are identified during a defect validation, depending on the size of the problem, the problem can be raised as a separate defect, linked and assigned to the owner of the fix under test.  If the side effect has low severity/ impact and the developer agrees it is related to the fix, the defect will be treated as fix not accepted, and the original one is reopened.  It is important to highlight side effects separately, as after investigation they may expose other problems that may or may not be related to the initial fix, and therefore they should be treated separately. |
| Won’t Fix | The defect marked as *Won’t Fix* with a clear comment and resolution is checked again by the assignee and update the test cases accordingly. If indeed the defect should be resolved regardless of the reason being marked as *Won’t Fix*, the defect can be *Reopened*. Then the status changes to *Open* with *Unresolved* resolution.  In case the arguments not to fix the defect are understood and agreed, the defect is *Closed*.  The Status changes to *Closed* with *Won’t Fix* Resolution, accordingly. |
| Duplicate | A defect marked as *Duplicate* is checked again if indeed it’s a duplicate. If yes, the defect is closed. The status changes to *Closed* with *Duplicate* Resolution.  If it turns out that the defect is not a duplicate, it will be *Reopened* and reassigned back to the previous assignee. The Status changes to *Reopened* and the Resolution to *Unresolved*. |
| Incomplete | The defect is reopened with all missing details and assigned back to the previous assignee.  The Status changes to *Reopened* with *Unresolved* Resolution. |
| Invalid | The defect is checked again. If indeed the defect is not according with requirements, the defect is closed.  The Status changes to *Closed* with *Invalid* Resolution. |
| Cannot Reproduce | The defect marked as *Cannot Reproduce* is revalidated by the reporter of it or by another Test Engineer. If the defect indeed cannot be reproduced, it is closed.  The Status changes to *Closed* with the Resolution *Cannot Reproduce*.  If the defect is still reproducible, it is reopened, and reassigned to the previous assignee with additional/clearer details provided.  The Status changes to *Reopened* with Resolution *Unresolved*. |

After validation the following fields are updated:

|  |  |
| --- | --- |
| Field | Change |
| Status | Status is changed depending on the validation output, either to *Verified*, or to  *Reopened* |
| Comment | The information about how the defect was validated is provided. Also, it is mandatory to provide the version and environment for which the defect was validated. |
| Assignee | If the defect is reopened, the defect is assigned back to the one who fixed it. |

### Defect closing

When all activities related to that defect are completed, the defect can be closed without changing the resolution.

While closing the following fields are updated:

|  |  |
| --- | --- |
| Field | Change |
| Status | Status is changed in *Closed*. |
| Comment | The information related to other extra activities that were performed after validation is provided. If there is none, the defect can be closed directly. |

## Defects Monitoring and Controlling

Project custom metrics can be created using the project’s test management tool. These metrics are good indicators of the quality of the product under test. Based on these, the testing priorities can change, new testing activities can be planned.

# Change Management

[Change Management is the discipline that allows planning, tracking, approval, scheduling, implementing and reviewing of the changes in the best viable way, in order to provide the best outcome for the business and all concerned.

At the heart of change management is the Change Request (CR) also known as Request for Change (RFC). This is a formal statement of "something needs to change". A CR contains all the details of the change. Different types of changes may require different fields, or information to be recorded.]

## Definitions

The Change Requests aim to make the software application more efficient, faster, more usable, more useful, and/or more desirable, by adding new functionality or changing the existing ones.

There are two major categories of Change Requests, depending on their goal:

* *Enhancement* or *Feature Request* – meant to add new features or functionalities
* *Improvement* or *Perfective Change –* meant to improve the functionality or the design

## Change Request lifecycle

Unlike the Defects, the Change Requests are usually altering the agreed scope of the project. Consequently, even if the lifecycle of the Change Requests is very similar to the Defects’ one, additional Approval/Rejection and Risk Analysis steps make the difference. No implementation effort, either development or testing related, will be invested before reaching the appropriate approval from the stakeholders.

# Testing deliverables

## Test Strategy

The Test Strategy document is used by the test team to guide how the testing will be managed for the project.

## Test Plan

The Test Plan document is used by the test team to guide a testing cycle execution. It describes the timeline, requirements or test cases in scope to be executed, test environment related activities, roles and responsibilities.

The Test Plan can be created in digital format, using an appropriate Test Case Management tool. In this case, all relevant information that might not be included – like test environments, roles and responsibilities – will be stated in the Test Strategy, accordingly.

## Test Cases

The test case is a set of preconditions, steps, test data and expected results, developed for a given functionality in order to verify its compliance with a requirement or to mitigate a risk.

They are defined using the appropriate methods, reviewed by a different person than the author, maintained during project life cycle, run and re-run if needed with results recorded, and stored.

## Traceability Matrix

The Traceability Matrix is used to trace and correlate the business or other requirements to their implementation, testing or completion. Usually it is a two-dimensional table, which correlates two entities (e.g., requirements and test cases).

The table allows tracing back and forth the links of one entity to the other, thus enabling the determination of coverage achieved and the assessment of impact of proposed changes. At any time, this should provide the project requirements status in terms of their level of completion.

## Test Report

The Test Report summarizes, for a specific time-period or a given testing cycle, the results of testing, including testing team related details, the defect status, a list of deviations from the test plan, test metrics, quality recommendations. All Test Report details are agreed with the stakeholders at the beginning of the project.

The Test Report can be created in digital format, by using a Test Case Management tool, and exported if needed in the agreed format. All relevant information that might not be included – like defect status, quality recommendation – will be stated in other deliverables, accordingly.

# Testing roles and responsibilities

[The roles and responsibilities shall be updated for each product based on agreements with the internal and the external teams regarding project and RACI roles, deliverables, activities, etc.

Try to avoid defining or assigning multiple RACI roles to the same project role for a given deliverable.]

|  |
| --- |
| RACI     Responsibility |
| Responsible – main responsibility in delivering  Those who do the work to complete the task. There is at least one role with a participation type of responsible, although others can be delegated to assist in the work required. |
| Accountable – management of delivering  The one ultimately answerable for the correct and thorough completion of the deliverable or task, the one who ensures the prerequisites of the task are met and who delegates the work to those responsible. In other words, an accountable must sign off (approve) work that responsible provides. There must be only one accountable specified for each task or deliverable. |
| Consulted – assistance (both active and advisory).  Those whose opinions are sought, typically subject matter experts; and with whom there is two-way communication. |
| Informed – informed or report to  Those who are kept up to date on progress, often only on completion of the task or deliverable; and with whom there is just one-way communication. |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Test Engineer | Test Lead | Project Manager | Client Responsible | Developer | BA |
| Deliverables / Activities (e.g. Test Strategy) | R | R, A | I | I | I | I |
| Test Plan | C | R, A | C | I | I | I |
| Test Cases | R, A | C | I |  | C | C |
| Test Scripts / Automation | R | I | I |  | C | C |
| Traceability Matrix | R | A | I | I | C | C |
| Test Report | R, C | R, A, C | C | I | I | I |
| Unit Testing | I | I | I |  | R, A |  |

# Testing assumptions and risks

[Risks below should be used as guidelines for commons risks that might be identified during a project.

Impact and Probability are given as examples; these are project related and should be modified accordingly. For guidance or tailoring refer to [TM-RSK Risk and Opportunity Management Process](https://confluence.iquestgroup.com/display/IMSSP/TM-RSK+Risk+and+Opportunity+Management+Process)

Some aspects to consider (additional to the below examples in the table):

Cultural differences

part of the client team might not be able to communicate in English

a different time zone will delay communication

Dependency to external service providers. Release dates may be impacted by acceptance from them (App Store, Google Play Store)

Forward compatibility

New desktop/ mobile OS versions, new browser versions released during development

Unexpected project scope extension

Failure to identify complex functionalities and time required to cover the testing for those functionalities

Delays in testing due to difficult project modules integration]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Risk description | Impact | Probability | Mitigation | Contingency |
| Test environments are not ready in time for any tests | High | Medium | Ensure overall test planning is known by all involved people  Clarify who is responsible for each environment as designated person | Re-planning |
| Not all defects are fixed before UAT phase | High | Low | Clarify conditions of entry criteria in the UAT phase based on previous test results | Re-planning/re-schedule UAT phase |
| The software versions on testing environments are not the same with the production used software systems or hardware versions | Medium | High | Clarify during planning sessions software used by the client in production versus the needed and required software in testing/staging environments. | Test in production what can’t be tested in test environments due to lack of needed software and hardware versions |
| Missing requirements or not definitive requirement documentation | High | High | Review Sessions |  |
| Lack of testing resources | High | High | Identify skills and needed resources.  Set-up trainings. | Reduce scope of testing  Re-estimate  Obtain new resource |
| Lack of testing or other specific skills in test team | Medium | Medium | Test team training is well prepared and performed.  Create a list of skills required for the project. | Re-allocate test resource to less difficult areas.  Ask for a different test resource.  Allocate time for training/coaching for test team. |
| Test design phase delayed | High | High | Status update on test case writing.  Clarify in the test strategy the scope and the timeline for test design.  Clarify exit criteria for test design. | Re-adjust test design estimated.  Re-planning of the test design phase.  Drop or reduce scope of test design. |
| Lack of communication | High | High | Set-up recurring meetings.  Establish who should participate in each meeting and the scope of each meeting.  Determine a way to monitor the meetings duration and quality of discussion (rating, feedback). | Identify issues from lack of communication and setup new meetings with the relevant people.  Escalate to project/test manager. |
| Absence in test team | Medium | High | Shared knowledge inside the team. | Re-schedule and re-prioritization. |
| The test environment becomes unavailable during testing | Medium | Medium | Planning takes environment IT service windows in consideration.  Establish if IT needs to be notified | Any incident to be raised immediately |
| Third party services fail. Some features will not work; testing of certain features is blocked | Medium | Medium | Determine contact persons for 3rd party services  Notify 3rd parties before testing starts | Reprioritize testing tasks  Determine a new testing slot when 3rd party service will be available |
| Requirements are not clarified in time | High | Medium | Start analyzing and design as soon as possible for all clarified requirements. | Establish next steps if requirements are not ready.  Switch testing to clear areas. |
| Features are released with delay, available testing time is reduced, some features might not get the expected coverage | High | Medium | Start testing as soon as possible for available features.  Feature testing prioritization. | Perform only exploratory testing. |
| Testing only on local development and testing environments will miss a lot of functionality (external interfaces, OS and hardware compatibility, network particularities, firewalls, etc.) | Medium | Medium | Staging environment to be ready before testing phase ends. Production environment to be ready before the system integration testing phase ends.  Move testing activities to staging environment as soon as possible. | Do testing activities in staging environment. |
| Scheduled deliveries are not respected | High | Normal | Have a constant overview on the current product status and remaining work.  Make sure that the scope from each delivery is according to the Test Plan. | Raise the problems in the Product Status Meeting.  Check impact on the Test Plan and redefine resource allocation. |
| Shift in project major milestones | Low | High | Regular checks with the product owner in terms of product major milestones. | Test Plan update.  Calculate the estimated impact.  Add priorities for features and start scope release negotiation. |
| Testing estimates are inaccurate | High | Medium | Discuss and elaborate assumptions made for the estimates.  Add testing estimates buffer. | Escalate as soon as possible.  Re-estimate.  Re-schedule.  Add more resources if needed. |
| Communication overhead | Medium | Medium | Determine what meetings are relevant.  Establish clear communication channels. | Escalate to your superior if meetings are not appropriate or useful. |

# Tools set

[Describe the list of tools planned to be used in the project. Only approved tools shall be selected. In particular cases, other tools can be selected and used with proper approval.

For each tool, mention the category it belongs to, and provide a brief description of the intended use. Also based on the project a responsible person can be delegated for each tool.]

|  |  |  |
| --- | --- | --- |
| Tool | Category | Description |
|  |  |  |
|  |  |  |

[For guidance, the tools used primarily in iQuest are listed below:]

|  |  |
| --- | --- |
| Tools | License Availability |
| Test Case Management Tools | |
| Spira | Yes |
| HP ALM | No |
| Microsoft Test Manager | Yes |
| Defect Tracking and Management Tools | |
| JIRA | Yes |
| HP ALM | No |
| Microsoft Team Foundation Server TFS | Yes |
| Redmine | No |
| Automated Test Tools / Frameworks | |
| Selenium WebDriver | Yes |
| TestNG | Yes |
| Test Complete | Yes |
| SOAP UI | Yes |
| Katalon | Yes |
| Performance Testing Tools | |
| Apache JMeter | Yes |
| SOAP UI | Yes |
| LoadRunner | No |
| Webload | Yes |
| Source Control Tools | |
| Stash | Yes |
| Tortoise SVN | Yes |
| Microsoft Team Foundation Server | Yes |
| Accessibility Testing Tools | |
| W3C validator | Yes |
| Wave web accessibility tool | Yes |
| WCAG color contrast checker (Firefox addon) | Yes |
| JAWS, sound spelling | No |
| Other Tools | |
| Notepad++ (enriched text editor with multiple add-ons) | - |
| Firebug (HTML inspection, analyze network usage) | Yes |
| Confluence (web platform for information sharing inside teams: test reports, environments details etc.) | Yes |
| Microsoft Office | Yes |
| Jenkins (automation server used for continuous integration jobs) | Yes |
| Apache Maven (software project management and comprehension tool, mainly used in Selenium WebDriver test automation projects) | Yes |
| Eclipse (IDE mainly used in Selenium WebDriver test automation projects) | Yes |
| Intelij IDEA (IDE mainly used in Selenium WebDriver test automation projects) | Yes |
| SQL Developer (IDE for ORACLE databases) | Yes |
| Toad for Oracle (IDE for ORACLE databases) | Yes |
| Microsoft SQL Server Management Studio (IDE for Microsoft SQL Server databases) | Yes |
| Fiddler (proxy for logging HTTP/HTTPS traffic) | Yes |
| Putty (SSH and telnet client) | Yes |
| WinSCP (sFTP client, used for file transfer between a local and a remote computer) | Yes |
| VMware (virtualization software) | Yes |