Master Test Automation Strategy

Contents

[Summary 1](#_Toc435551586)

[Purpose 2](#_Toc435551587)

[Guiding Principles 2](#_Toc435551588)

[Quality and Test Objectives 2](#_Toc435551589)

[Test Approach 4](#_Toc435551590)

[Test Types 4](#_Toc435551591)

[Test Lifecycle 5](#_Toc435551592)

[User Stories 6](#_Toc435551593)

[Acceptance Criteria 6](#_Toc435551594)

[Environments 7](#_Toc435551595)

[Delivery Pipelines 8](#_Toc435551596)

[How To 8](#_Toc435551597)

[How to build 8](#_Toc435551598)

[How to execute Release Test 8](#_Toc435551599)

[How to execute Regression Test 8](#_Toc435551600)

[How to seed data locally 8](#_Toc435551601)

[How to seed data in Test environment 8](#_Toc435551602)

[How to view pipeline metrics 8](#_Toc435551603)

[How to view pipeline metrics 8](#_Toc435551604)

[How to view test metrics 8](#_Toc435551605)

[Test Data Management 8](#_Toc435551606)

[Defect Management 8](#_Toc435551607)

[Classifying Defects 8](#_Toc435551608)

[New Release Defects 9](#_Toc435551609)

[Regression and Non-release Defects 9](#_Toc435551610)

[Defect Life Cycle 9](#_Toc435551611)

# Summary

This document is the Master Test Automation Strategy. The objective is to define the overall testing effort to provide fast feedback to help prevent defects, security flaws and health issues from entering production in order to maximize value and minimize risks for customers and the business.

# Purpose

The purpose of this document is to create a shared understanding of the overall approach, tools, targets and timing of automated test activities.

# Guiding Principles

|  |  |
| --- | --- |
| **Principle** | **Description** |
| **Share Responsibility** | Everyone is responsible for testing, quality, security, and health of the system. |
| **Automate Tests** | All types of tests (unit, integration, acceptance, regression, performance) should be automated. Manual testing will only be used for exploratory type testing and to cover regression tests not covered by automation. |
| **Protect Privacy** | Production data must be obfuscated before being used for testing |
| **Test Management** | Test assets: feature files, code, documents and data will be treated with the same importance as production code. Test assets are stored in the same source code repository as the production code. |

# Quality and Test Objectives

The following quality attributes have been identified as relevant and are used as a basis for the test approach in terms of priority and test targets.

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Measure and Target** | **Priority** |
| **Correctness** | Features and functions work according to acceptance criteria and defined specifications. | * 100% completion of agreed features * Severity 1 defects = 0 * Severity 2 defects = 0 * Severity 3 defects < 2 * Severity 4 defects < 5 * Severity 5 defects < 10 | Must Have |
| **Security** | Code and public interface tested for security flaws. | * Static analysis before promotion to P1 * Dynamic analysis of Vendor and Customer site before promotion to Prod * 0 very high flaws * 0 high flaws * 0 medium flaws older than 30 days * 0 low flaws older than 60 days * 0 very low flaws older than 90 days | Must Have |
| **Integrity** | Ability to prevent unauthorized access, prevent information loss, protect from viruses infection, protect privacy of data entered | * All access will be via HTTPS over a secured connection. * User passwords and session tokens are encrypted. | Must Have |
| **Maintainability** | How easy it is to add features, correct defects or release changes to the system. | *Still evaluating collection and metrics*   * Code Duplication < 5% * Code Complexity < 8 * Unit Test Coverage > 80% * Method Length < 20 Lines | Nice to Have |
| **Availability** | Percentage of planned up-time that the system is required to be operational. | * System is available for 99.99% of the scheduled up-time measured through system logs. | Should Have |
| **Interoperability** | Ease with which the system can exchange information with other systems. | User interface renders and functions properly on the following (and later) browsers versions:   * IE version => 10.0 | Must Have |
| **Performance** | Measures the responsiveness of the system under a given load and the ability to scale to meet growing demand. | * Apdex Score > 0.9 * Response Time < 200ms * Throughput > 100 rpm | Nice to Have |

# Test Approach

## Test Types

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Type** | **Definition** | **Scope** | **Test Tools** | **Speed** | **Execution / Timing** |
| **Unit-Small** | Fast tests that verifies the implementation of software changes work correctly in isolation. | All public classes and methods that can be tested in isolation. | NUnit, xUnit MSTest | <1 sec | Executed locally, on commit |
| **API-Medium** | Testing in which API endpoints are tested. | All public facing APIs. Private APIs are in scope but prioritized after public. | NUnit, xUnit, MSTest, Mocha | <1 min | Executed locally, on deploy |
| **Smoke-Medium** | Functional testing based on known specifications to verify critical and minimum application functionality. | All applications and scenarios that are testable through a web based UI.  Scenarios that involve verification of file or email content are not in scope. | Tarsvin, NUnit, TestPipe, WebDriver | <15 min for execution of all tests. | Executed on deploy |
| **Release-Large** | Functional testing based on specifications defined in a release. | All applications and scenarios that are testable through a web based UI.  Scenarios that involve verification of file or email content are not in scope. | Tarsvin, NUnit, TestPipe, WebDriver | >1 min | Executed on deploy |
| **Regression-Large** | Functional testing based on known specifications to verify that the system still works as specified. | All applications and scenarios that are testable through a web based UI.  Scenarios that involve verification of file or email content are not in scope. | Tarsvin, NUnit, TestPipe, WebDriver | >1 min | Executed on deploy |
| **Performance** | Testing confirm that the system meets required performance. | Common and most used functionality and workflows. | WCAT, OpenSTA |  | Executed monthly |
| **Security** | Testing to identify potential security flaws. | System code and public facing user interfaces and APIs. | Veracode |  | Executed before release to production |
| **Exploratory** | Manual testing that provides simultaneous learning, test design and test execution. | All functionality. | Manual |  | Executed before release to production |
| **Production Validation** | Manual testing that provides validation of changes deployed to production. | Minimum functionality to validate that the release was successful. | Manual |  | Executed after production release |

## Test Lifecycle

|  |  |  |
| --- | --- | --- |
| **Step** | **Description** | **Role Participation** |
| **Discuss** | Discuss or workshop to create a shared understanding of the tickets. Define the initial acceptance criteria necessary for each ticket to be considered complete. Identify real-life scenarios and examples that have realistic context. Define the test types that are in scope for each ticket. | Product Owner  Business Analyst  Developer  Automation Engineer  QA Analyst |
| **Estimate** | Agree on the relative effort to satisfy the acceptance criteria. | Developer  Automation Engineer |
| **Distill** | Distill the ticket into an executable specification based upon the user stories, examples scenarios, and initial acceptance criteria. Specifications are kept in human readable form. Specifications should address the happy path, negative paths, and edge cases (even if they aren’t automated) | Business Analyst  Developer  Automation Engineer  QA Analyst |
| **Develop** | When feasible develop the required tickets using automated test-first practices (using the Red, Green, Refactor pattern). Automated unit and API tests are used to support the tickets.  Automated release tests are developed in parallel with development. | Developer  Automation Engineer |
| **Validate** | Automated tests are ran. | Automation Engineer |
| **Demonstrate** | Demonstrate the acceptance criteria is met by performing manual exploratory tests and running the automated release tests. | Business Managers  Product Owner  Business Analyst  Developer  Automation Engineer  QA Analyst |

### User Stories

A good user story should be:

* “I” ndependent (of all others)
* “N” egotiable (not a specific contract for features)
* “V” aluable (or [vertical](http://guide.agilealliance.org/guide/incremental.html))
* “E” stimable (to a good approximation)
* “S” mall (so as to fit within an iteration)
* “T” estable (in principle, even if there isn’t a test for it yet)

In addition user stories should include acceptance criteria.

### Acceptance Criteria

Gherkin is a common human readable format that can be used to define acceptance criteria, but is not required.

Feature/Story: Title

In order to [value]

As a [role]

I want [feature]

Scenario Outline: Title

Given [context]

And [more context]

When [event]

Then [outcome]

And [another outcome]

# Environments

|  |  |  |  |
| --- | --- | --- | --- |
| **Environment** | **Description** | **Data Setup** | **Test Usage** |
| **Local** | This environment is local and specific to each developer/tester machine. It is based on the version/branch of source code being developed. Integration points are typically impersonated and may use other environments. | Data and configuration is populated through setup scripts. | Unit, API, release and exploratory testing  Static code analysis |
| **Dev** | This environment is used primarily by developers and supports nightly and on demand integration of code changes. | Data and configuration is populated through setup scripts. | API, release, security, and exploratory testing |
| **Test** | This environment supports continuous integration of code changes and execution of automated tests on each check-in and deployment. Additionally, static code analysis and security testing is completed in this environment. | Data and configuration is populated through setup scripts. | Unit, API, release, security testing  Static code analysis |
| **PR** | This environment support exploratory testing | Populated with obfuscated production data. | Exploratory testing |
| **Production** | Live environment | New instances will contain standard project reference data. Existing instances will have current data migrated into the environment. | Production verification testing |

# Delivery Pipelines

Changes flow through multiple pipelines to get from source code check-in to production:

|  |  |  |
| --- | --- | --- |
| **Pipeline** | **Description** | **Trigger** |
| **Build** | This pipeline builds, tests, packages, and publishes packages | Every commit |
| **Test Deploy** | This pipeline gets the latest package and deploys it to the Test environment | On build success when no deploy or test currently running in Test environment |
| **Dev Deploy** | This pipeline gets the latest package and deploys it to a Dev environment | Nightly for latest successful build |
| **Automated Test** | This pipeline run automated tests against the Test environment | On deploy success to Test environment when no deploy or test currently running |
| **PR Deploy** | This pipeline gets the latest package and deploys it to the PR environment | Manual trigger with latest package passing regression test |
| **Manual Test** | This is not an automated pipeline, but a step where manual tests are executed against PR | Manual after successful deploy of PR |
| **Production Deploy** | This pipeline gets the configured build package and deploys it to the Prod environment | Manual trigger with latest package after approval for Prod deployment |

# How To

## How to build

## How to execute Release Test

## How to execute Regression Test

## How to seed data locally

## How to seed data in Test environment

## How to view pipeline metrics

## How to view pipeline metrics

## How to view test metrics

## Test Data Management

Test data is managed through data seed scripts. If you need to edit test data you have to update the seed script and check it into the source repository.

# Defect Management

Defects are considered any functionality that does not function as expected.

## Classifying Defects

|  |  |  |
| --- | --- | --- |
| **Severity** | **Customer Impact** | **Development Impact** |
| **Sev1 Critic al** | Complete outage. | Testers can't test any functionality. Developers can't continue development. |
| **Sev2 Severe** | Major functionality unusable. | Results in major testing/development delays. |
| **Sev3 Major** | Major functionality broken, but usable with. | Results in major testing/development delays, but testing/development can continue in other areas. |
| **Sev4 Minor** | Minor functionality broken, but usable with workaround. | Results in minor testing/development delay or irritation. |
| **Sev5 Trivial** | False alarm or immediate fix available. | Minor testing/development interruption. |

## New Release Defects

If a defect is found that is caused by work in an ongoing release, it should be raised with the development team. The defect should be fixed immediately. If it cannot be fixed immediately a new ticket should be added to the sprint’s defect backlog.

## Regression and Non-release Defects

If a defect is found that is not caused by ongoing work in a release, it should be entered into the defect tracking system.

## Defect Life Cycle

|  |  |
| --- | --- |
| **Step** | **Description** |
| **Identify Defect** | Ensure defect can be reproduced. Identify root cause and dependencies affected by failure. Define or identify failing automated tests. |
| **Triage Defect** | Team reviews defect to determine defect severity and if it should be fixed in current sprint or a defect ticket created and prioritize in the backlog. |
| **Resolve Defect** | Follow test lifecycle |
| **Close Defect** | Execute tests to verify defect is resolved and no regression. If a defect ticket, close the defect ticket. |