LOGO FIRMY

|  |
| --- |
| Test PLAN for XXX project |
|  |

Contents

[1 Introduction 3](#_Toc447791061)

[1.1 Definitions and Acronyms 3](#_Toc447791062)

[2 Testing approach 3](#_Toc447791063)

[2.1 Test types 3](#_Toc447791064)

[2.2 Tools 4](#_Toc447791065)

[3 Metrics 5](#_Toc447791066)

[4 Testing process in scrum teams 6](#_Toc447791067)

[4.1 Diagrams 6](#_Toc447791068)

[4.2 Test stages 6](#_Toc447791069)

[5 testing roles 8](#_Toc447791070)

# Introduction

This is the Test Strategy Document for XXX projects. This document shall be completed and used by the project test team to guide how testing will be managed for this project. The test effort will be prioritized and executed based on the project priorities as defined in the Project Plan and Requirements Specification. This is a living document that may be refined as the project progresses.

## Definitions and Acronyms

**Project name XXX Web, XXX API, Projects related to legal.**

**Ad Hoc Testing**

Testing contrived for only the specific purpose or problem at hand; testing not carefully planned in advance.

**Scenario**

Detailed description (specific instance) of a use case, including rules, exceptions, boundaries, limits, etc.

**Test Case**

A specific set of test data along with expected results for a particular test objective.

**Test Coverage**

Describes how much of a system has been tested.

**Test Design**

Describes how a feature or function shall be tested.

**Test Procedure**

Describes the steps for executing a set of test cases and analyzing their results.

**Test Script**

Step by step description for specific tests.

**Test Strategy**

Describes the scope, approach, resources and schedule for the testing activities of the project. This includes defining what will be tested, who will perform testing, how testing will be managed, and the associated risks and contingencies. Also referred to as a Test Plan.

**Use Case**

Describes a sequence of interactions between a system and an external actor that results in the actor accomplishing a task that provides benefit to someone. An actor is a person or other entity external to the software system being specified who interacts with the system to accomplish tasks. Different actors often correspond to different user classes, or roles, identified from the customer community that will use the product.

# Testing approach

Testing will be designed to encompass the following:

Testing will cover manual functionality testing for Online XXX changes through the use of the test interface. This will validate base functions of the new code. There will also be prepared automation tests for regression checking for both Online XXX and APIs.

## Test types

**Unit tests**

Unit testing is testing performed to determine that individual program modules perform per the design specifications. They are created by developers.

**Manual functional tests**

Check functionality for the developed features. Maintained by manual tester - mainly Jan Nowak.

**Manual Ad-hoc tests**

Check if feature has been fixed after bug fixing - retesting bugs. Other not planned manual tests. Maintained by manual tester - mainly Jan Nowak.

**Automation smoke tests**

A smoke test is basically just a sanity check to see if the software functions on the most basic level. If your smoke test fails, it means there is no point in running your other tests because the main applications functionality does not work. Maintained by automation testers. Specific roles are described in chapter "Testing Roles".

**Automation regression tests**

The purpose of regression testing is to ensure that changes have not introduced new faults. One of the main reasons for regression testing is to determine whether a change in one part of the software affects other parts of the software. If comes to API testing integration tests are regression tests as well. Maintained by automation testers. Specific roles are described in chapter "Testing Roles".

**Performance tests**

Performance testing is in general, a testing practice performed to determine how a system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage. Maintained by automation testers. Specific roles are described in chapter "Testing Roles".

## Tools

**Selenium WebDriver**

Selenium is a browser automation library. Most often used for testing web-applications, Selenium may be used for any task that requires automating interaction with the browser.

**AssertJ Swing**

AssertJ Swing is a Java library that provides a fluent interface for functional Swing UI testing.

**Rest Assured**

Library for testing and validation of REST services in Java is harder than in dynamic languages such as Ruby and Groovy. REST Assured brings the simplicity of using these languages into the Java domain.

**EasyTest Framework**

EasyTest is a framework providing its users a convenient mechanism to perform Data Driven testing and at the same time keeping it intuitive.

**Hamcrest**

Hamcrest is a library of matchers, which can be combined in to create flexible expressions of intent in tests. They've also been used for other purposes.

**TestRail**

Modern Test Case Management Software for QA and Development Teams.

**Jira**

Project tracking tool.

**Confluence**

Documentation storing tool.

**Stash**

Tool for code reviewing.

**Jenkins**

Jenkins is the leading open source automation server. Built with Java, it provides hundreds of plugins to support building, testing, deploying and automation for virtually any project.

**Git**

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

**JMeter + plugins**

Performance testing and measuring tool. One of the leading open source tools with big amount of plugins and possibility to manage with CI tools like Jenkins, test could be drive from Maven as a part of project technology stack.

**Kibana**

Visualisation tool which is helpful in time of analyzing results of response times, tracking logs on server side.

# Metrics

Each field contain percentage coverage for functional tests, automations tests, positive and negative paths. Percentage is calculated based on acceptance criteria and description for each user story. For calculate whole coverage for functional tests has been used following equation - sum of all coverage in functional column divided by amount of rows. For rest columns have been used similar equations, but not each user story needs to have automation tests or negative paths. If they are not needed field contains information NO NEED. In case of that this filed will be excluded from calculations.

This is example table for the first milestone. It state from 7th of January 2016. Positive paths and Negative paths fields are not updated yet. If you are more interested full up to date document can be found here:

[<Link](https://conf.booxdev.com/display/PR/QA+Metrics) to the doc online>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| MILESTONE 1 | | [ % ] | | | |
|  | **User Story** | **Functional TCs** | **Automation TCs** | **Positive Paths** | **Negative Paths** |
| 1 | TIPDEV-8651 [Header] Component - Logged Out | 100 | 40 | 0 | 0 |
| 2 | TIPDEV-8669 [Header] Component - Login Layer | 100 | 0 | 0 | 0 |
| 3 | TIPDEV-8650 [Header] Component - Logged In | 100 | 0 | 0 | 0 |
| 4 | TIPDEV-8998 / DEV-445 [Games Manager] Permissions | 100 | 0 | 0 | 0 |
| 5 | TIPDEV-8999 [Games Manager] Game categories | 100 | 0 | 0 | 0 |
| 6 | TIPDEV-8989 [Multisite] AT domain binding | 100 | 0 | 0 | 0 |
| 7 | TIPDEV-7245 [Multisite] Use Magnolia Multisite feature to allow for customisation depending on domains and region | 100 | 0 | 0 | 0 |
| 8 | TIPDEV-8990 / DEV-442 [Multisite] DE domain binding | 100 | 0 | 0 | 0 |
| 9 | TIPDEV-6972 Left Casino Navigation Menu | 100 | 70 | 0 | 0 |
| 10 | TIPDEV-9304 Casino Game List Component | 0 | 0 | 0 | 0 |
| 11 | TIPDEV-8996 [Multisite] Provide ability to add and remove new languages for a particular domain | 100 | 0 | 0 | 0 |
| 12 | TIPDEV-7211 Casino Content Page Template | 100 | 0 | 0 | 0 |
| 13 | TIPDEV-9000 [Games Manager] Game Filter Attributes | 100 | 0 | 0 | 0 |
| 14 | TIPDEV-7203 Casino Game Category Template | 100 | 0 | 0 | 0 |
| 15 | TIPDEV-7201 Casino Lobby Template | 100 | 0 | 0 | 0 |
| 16 | TIPDEV-7200 Casino Game Search | 100 | 0 | 0 | 0 |
| 17 | TIPDEV-9042 [Games Manager] List of games for game manager | 100 | 0 | 0 | 0 |
| 18 | TIPDEV-7110 [Games Manager] Provide the ability to manage and add games through the CMS | 100 | 50 | 0 | 0 |
| 19 | TIPDEV-8997 [Games Manager] Game publishing process | 100 | 0 | 0 | 0 |
| 20 | TIPDEV-9001 [Games Manager] Game Artwork Labels | 100 | 0 | 0 | 0 |
| 21 | TIPDEV-8995 [Multisite] Different set of games on each domain | 100 | 0 | 0 | 0 |
| 22 | TIPDEV-7046 Casino Page Structure Static HTML | 100 | 0 | 0 | 0 |
| **Coverage [%]** | | **95,65** | **6,67** | **0,00** | **0,00** |

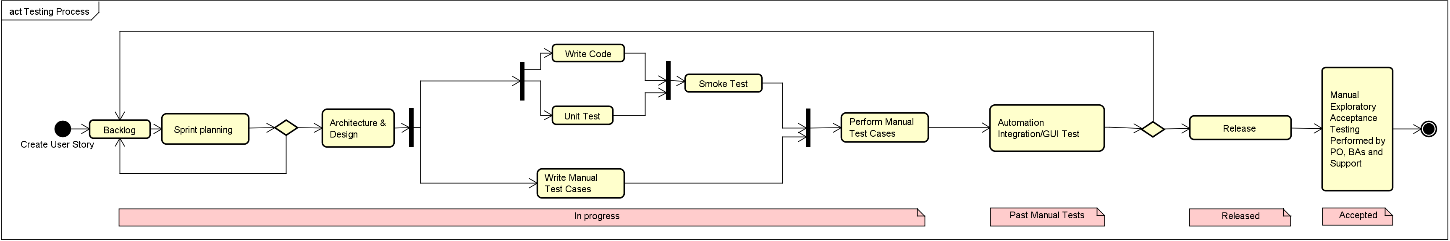
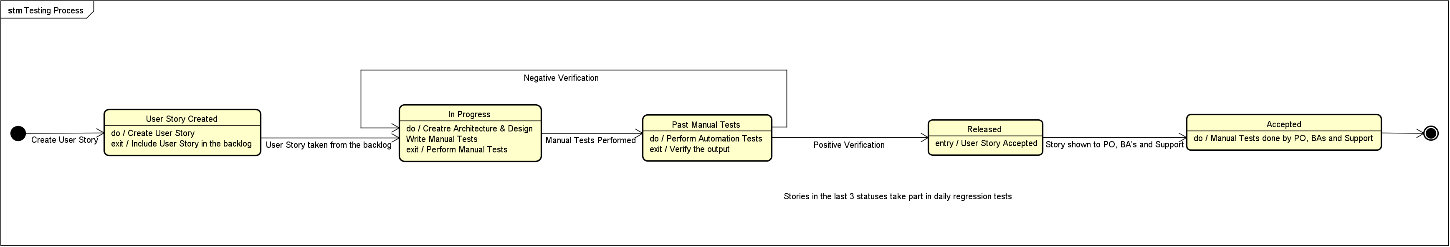
For non-functional Performance tests Metrics will be meeting requirements stored there:

[<Link](https://conf.booxdev.com/display/PR/QA+Metrics) to the doc online>

# Testing process in scrum teams

## Diagrams

This is proposition of testing process that can be implanted in our scrum teams.



## Test stages

**Feature Tests**

Each feature should be tested after deploy. It has to be covered by manual tests - firstly test cases should be written and stored in TestRail then performed by manual tester. Automation tests can be created in parallel or later - the best option in this same sprint. Any defect found has to be reported and fixed, based on its priority.

**Unit Tests**

Unit tests - should be run after each build.

**Smoke Tests**

Smoke GUI Tests - The ability to give early feedback during testing is a cornerstone to providing value on a software project. A well designed smoke test is one part of a strategy that can enable that feedback.

Those tests should be run after each build, to make sure that we did not broke anything down. Any defect found has to be reported and fixed, based on its priority.

How we can determine what to put in our smoke test? First, we need to ask what the critical behaviors that need to be working are. Then we have to consider the types of changes that have occurred since the last test cycle and what you need to check to gain necessary confidence that these things and what they impact are in “working order”. Finally, we need to prioritize which other areas of the system we need to review in more detail than others (eg: what would we do next if we had some extra time).

**Regression Tests**

Automation is a critical component of Agile testing. It would be impossible to keep pace with the Agile development schedule otherwise. Automation is used to run regression testing. The combined team (Developers, Product Owners and Testers) usually predetermine, at the start of the project, which parts of the software will be tested using automation.

In our case, by regression tests, I understand

- Regression GUI tests part of TA-WEB project - XXX package that contains all tests can be found here - https://git.XXX.com/projects/EBPL/repos/ta-web/browse,

- Integration API Tests - Tests checking real API without mocking data - Those tests can be found here - https://git.XXX.com/projects/EBPL/repos/ebet-api/browse/ebet-api-services/src/test/java/integration.

Should be run daily. Any defect found has to be reported and fixed, based on its priority. Any defect found has to be reported and fixed, based on its priority.

**Manual exploratory tests**

Should be performed before ends date of each sprint - 2 or 3 days before its end. There is nothing that compares with the tester's creativity, and we should apply this kind of testing once every sprint. Those tests should be performed by product owner, business analysts. Testes should support them.

**Performance Tests**

Performance tests will be focus only on Casino (Magnolia Application) and XXX REST API.

*Performance tests should be conduct with 2 main scenarios (with combinations of other types).*

* Performance tests as it is + Load testing
  + Measure standard response times (without generating big load)
  + Find out proper behavior of application
  + Check proper configuration of environment
  + Check proper configuration of performance tools (like Jmeter) to make sure that we are generating proper load
* Stress tests + Spike tests + Endurance (according to requirements)
  + Find out upper capacity of application servers - stress
  + Find out how system respond under heavy load in long time - stress
  + Find out how system restore after fail – spike, endurance
  + Check session management (memory leaks) – endurance, spike
  + Increase load over the maximum to find out how is the maximum load (when system fails)

*Number of Application Tiers (Nodes)*

* For proper testing we need to firstly find out capacity of particular node of application and then extend this to testing server with all nodes (using Load Balancer).

*Test environment structure*

* Servers dedicated for Casino and Casino-API will be used at the end as a part of production services (testing environment = final production environment) – so there is no need to check differences between production and test environment.

*Mocks & Stubs*

* All external services (XXX production) need to be mocked before testing XXX and XXX-API – it need to be done to prevent crash of production.

*Physical isolation of Test Environment*

* Test environment should be separated from production environments

*Quality indicators*

* Quality of performance should be no worse that SLA of 3rd party products (f.eg. NETENT, NYX, etc.)
* Quality of performance should be no worse than other (already production) parts of the system.

*Elements of the System Under Test (SUT)*

* Application server (with application itself)
* Database server (cache of DB as integration part of the system)
* Static server files (sometimes it could be a part of the application) **–** **to be check**
* Network connection between server environment and testing tools environment

*Elements for analyzing*

* Response times in testing tool (JMeter)
* Execution times on server side
* Execution of queries in SQL DB (if required for very slow endpoints)
* Network connection should not affect response times (f.eg response times on performance tool should not be much bigger than on server side)

*Database preparation*

* Database for testing production should not be empty and be ready to get big amount of newly created data (for measuring increasing response times in fullfilling DB – valid only for endurance tests)

*IP Spoofing Implementation*

* Find out preventing of multiple sessions from one IP
* If multiple sessions from one IP is blocked, then – change configuration of server or prepare IP Spoofing methods for performance **– to be check**

*Performance Tests Environment Checklist*

1. Number of Servers: Number of physical and virtual servers
2. Load Balancing Strategy: The type of load balancing mechanism is in use
3. Hardware Resources: CPUs count and type, RAM capacity, Number and type of NICs
4. Software Resources: Standard application build apart from components of the AUT
5. Application Components: Application components description which needs to be deployed on the server
6. External Links: Links to third party application and other internal system components

# testing roles

**Jan Nowak**

Jan is manual tester who is responsible for:

- performing manual tests cases,

- creating manual test case scenarios in TestRail,

- retesting bugs,

- updating test coverage documents,

- supporting team in ad-hoc testing for Online XXX in Rainbow Six team.

**Franek Kimono**

Franek is automation tester who is responsible for:

- performing and creating manual test cases for legal area in Oceans 11 team,

- creating automation tests for Online XXX for Rainbow Six team,

- creating automation tests for Oceans11 legal stories,

- setting up testing process inside Oceans 11 team,

- helping setting up testing environments,

- Franek is a key person if comes to performance testing across teams.

**John Snow**

John is automation tester who is responsible for:

- Creating test scenarios for Ghost Busters team for XXX,

- Creating automation tests for Ghost Busters team for XXX,

- Maintaining TA-WEB framework dedicated to end to end tests across systems (XXX, YYY ZZZ, AAA),

- Developing AssertJ tests for Swing applications,

From 25th of January John is joining Avangers team where he will be responsible for:

- creating automation tests for APIs and other features developed by Avangers team,

- helping with manual and automation tests for Online XXX for Rainbow Six team.

**John Doe**

John is automation tester who is responsible for:

- Creating test scenarios for Ghost Busters team for XXX,

- Creating automation tests for Ghost Busters team for XXX,

- Maintaining TA-WEB framework dedicated to end to end tests across systems (YYY, Online XXX, Iovation),

- Developing AssertJ tests for Swing applications,

From 25th of January John is joining Argonauts team where he will be responsible for:

- creating automation tests for APIs and other features developed by Argonauts team,

- helping with manual and automation tests for Online XXX for Rainbow Six team.

**Robert Burneika**

Robert is automation tester who is responsible for:

- coordinating testing processes across the teams,

- supporting testing teams,

- helping with manual and automation tests for Online XXX for Rainbow Six team,

- helping with automation tests for APIs for Avangers and Argonauts team,

- coaching and mentoring testers.

| REVISION HISTORY | | | | | |
| --- | --- | --- | --- | --- | --- |
| Rev. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| 0.1 | Document created. | Robert Burneika | 06-Jul-2015 | Super Manager | 7-Jul-2015 |
| 0.2 | QA review provided | Anna Jakastam | 1-Apr-2016 |  |  |
| 1.0 | Changed according to review notes | Manager Name | 7-Apr-2016 | Super Manager | 7-Apr-2016 |