TEST PLAN

project: Diagram Generator

Type: Internal
Date: 5/25/2016

Table of Contents

1. Introduction

The overall purpose of testing is to ensure the project DIAGRAM GENERATOR meets all of its technical, functional, and business requirements. The purpose of this document is to describe the overall test plan and strategy of testing the DIAGRAM GENERATOR. The approach described in this document provides the framework for all testing related to this application.

1.1 Objectives

The quality objectives of testing DIAGRAM GENERATOR are to ensure complete validation of the business and DIAGRAM GENERATOR requirements.

- Verify DIAGRAM GENERATOR requirements are complete and accurate
- Perform detailed test planning
- Identify testing standards and procedures that will be used on the project
- Prepare and document test scenarios and test cases
- Regression testing to validate that unchanged functionality has not been affected by changes
- Manage DIAGRAM GENERATOR's defect tracking
- Provide test metrics/testing summary reports

1.2 Team Members

Name	Role	Email	GitHub
Fedeles Daniel Andrei	Automation Tester/QA MANAGER (Front-end)	daniel.fedeles@info.uaic.ro	qazwer1
Sparhat Cosmin Mihai	Automation Tester (Front-end)	cosmin.sparhat@gmail.com	Sparhat
Carnaru Tudor	Automation Tester	tudor.carnaru@yahoo.com	genius252

	(Back-end)		
Dobos Stefan	Automation Tester (Back-end)	alexandru.dobos@info.uaic.ro	stfn12
Stanciu Emilian	Manual Tester	emi_stanciu2002@yahoo.com	Emilian28
Bordeianu Andreea	Manual Tester	andreea.bordeianu@info.uaic.ro	AndreeaBorde
Gavrin Flaviana	Manual Tester	flaviana.gavril@info.uaic.ro	Flavinia
Palasanu George	Manual Tester	george.palasanu@info.uaic.ro	Palasanu
Ciobanu Bogdan	Manual Tester		bogdan-ciobanu

1.3 Goals

The goals of testing this application include validating the quality, usability, reliability and performance of the application. Testing will be performed from a black-box approach and a part of white-box. Tests will be designed around requirements and functionality.

Another goal is to make the tests repeatable for use in integration and regression testing during the project lifecycle, and for future application upgrades. Repeated tests (Smoke and Regression) should be automated.

Quality

First DIAGRAM GENERATOR's verification will be undertaken involving reviews and meetings to evaluate documents, plans, requirements and specifications to ensure that the end result of the application is testable and the requirements are covered. The overall goal is to ensure that the requirements are clear, complete, detailed, comprehensive, attainable and testable. In addition, this helps to ensure that requirements are agreed by all stakeholders.

Second, actual testing will be performed to ensure that the requirements are met. The standard by which the application meets quality expectation will be based upon the requirements test matrix, use cases and test cases to ensure test case coverage of the requirements. This testing will also help to ensure the utility of the application.

Reliability

Reliability is both the consistency and repeatability of the application. A large part of testing an application involves validating its reliability in its functions, data, and system availability. To ensure reliability, the test approach will include positive and negative (break-it) functional tests. In addition, to ensure reliability throughout the iterative DIAGRAM GENERATOR development cycle, regression tests will be performed on all iterations of the application.

2. Scope

1)Functional tests are fully covered with test cases in GitHub and partially by Automated tests in Java.
2)System testing provided by QA on Formal stages. During all the stages, QA team checks that all the features work correct.

Cross Browser Testing:

- *User interface testing.* QA team will use this testing technique to verify and validate the Graphical User Interface (check if the interface is easy to use, understand and if the GUI requirements are met).
- *Usability-testing.* QA team will use this technique to simulate how the real users use the system. In our case, the real users can also be developers or testers.
- Integration testing is done with manual and automation testing by QA team.
- Performance testing is done by QA team.
- Unit testing is done by QA team.
- Automation Testing is done by QA team
- Manual Tester is done by QA team.

Portability testing:

Opera	Firefox	Chrome	Internet Explorer	Safari	Microsoft EDGE
OK	OK	OK	OK	N/A	OK

Test modules:

- GUI
- Controller
- View1
- View2

Testing types:

- Functional testing: Unit testing, Integration testing, Regression Testing, Acceptance Testing
- Non-Functional testing: Performance testing, Load testing, Stress testing, Portability testing

Stages of testing:

- Exploratory testing
- Test cases based testing

3. Test Strategy

The project is using an agile approach, with 3 weeks iterations. At the end of each 3 week the requirements identified for that iteration should be delivered to the team and all functionality will be tested.

Smoke test will be executed manually after each build sent to QA team. If there is a release, full regression test will be executed by manual and automation.

Black Box Testing: Behavior testing or Partition testing focuses on the functional requirements of the DIAGRAM GENERATOR.

GUI Testing: GUI testing will include testing the UI part. It covers users Report format, look and feel, error messages, spelling mistakes, GUI guideline violations.

Functional Testing: Functional testing is carried out in order to find out unexpected behavior of the report. The characteristics of functional testing are to provide correctness, reliability, testability and accuracy of the report output/data.

Smoke Testing: Smoke test is executed prior to start actual testing to check critical functionalities of the program is working fine. This set of test cases written such a way that all functionality is verified but not in deep

Regression Testing: Regression testing is done before Release and any time when the DIAGRAM GENERATOR is changed, to make sure that the older programming still works with the new changes.

4. Testing Overview

4.1. Preparing test cases

QA will be Preparing Test Cases in GitHub based on the requirement specifications . This will cover all scenarios for requirements. Test cases should be written before development.

4.2. Test Automation

For the test automation the following technologies and tools will be used:

Java with Selenium Webdriver and TestNG for the Front-end
Java and Junit for Unit tests.

4.3. New Feature

The QA team starts when Feature status is “Ready for QA”, the QA tester execute test cases that are related to the feature. If there is no defect, the QA tester change status of task as “Closed”. If the the QA tester finds an issue, he changes the status of the feature as “Reopened” and a comment to the task that includes the details of defect(Steps, expected result and actual result).

4.4. Fixed Bug

The QA team starts when Bug status is “Ready for QA”, QA tester retests the case. If there is no defect, QA tester changes the status of task to “Closed”. If the issue is still present, the QA tester changes the status of bug as “Reopened” and comment to bug. In retest if the bug is closed but the developer causes a new defect, the QA tester opens new bug.

4.5. Types of testing used:

Unit Testing

All code needs to be unit tested to ensure that the individual units (classes, modules or files when valid) perform the required functions and output the proper results and data. Proper results are determined by using the design limits of the calling (client) function as specified in the design specification defining the called (server) function. Outer effects and dependencies may be isolated by performing mock tests.

Unit testing is typically white box testing and may require the use of DIAGRAM GENERATOR’s stubs and symbolic debuggers. This testing helps ensure proper operation of a module because tests are generated with knowledge of the internal workings of the module.

A special form of unit testing is Integration Test, which does not isolate dependencies and external components of the DIAGRAM GENERATOR.

Development/Testing team members are responsible for testing each program produced to demonstrate correct operation of coded modules units. Recommended tools and techniques for unit testing are Mockito, Junit.Eclipse moreUnit.

Load/Performance Testing

Load/performance testing is conducted to demonstrate the correct operation and performance of the DIAGRAM GENERATOR release as well as to identify the operational envelope of the solution, revealing weaknesses and possible bottleneck points. Load and performance tests should be done with Performance testing software. The results will be discussed within team and decided if measures have to be taken in order to improve performance.

Integration Testing – functional testing part

The version of the project must be increased. New version must be deployed to test environment. This testing type is executed manually, based on the test cases created by the QA team, together with the automated tests. If the QA tester find defects, he/she reports the bugs in GitHub Issues.

User Acceptance Test:

The version of the project must be increased. New version must be deployed to test environment. Smoke tests and Exploratory tests are executed when sufficient time is not available for Regression tests. Smoke test cases are executed after code freeze. If the QA tester find defects, he/she reports the bugs in GitHub Issues.

Smoke Test:

The version of the project must be increased. New version must be deployed to test environment. Smoke test is executed exactly when the build is deployed to Test Environment and when sufficient time is not available for Regression testing. Smoke test cases are executed after code freeze. If the QA tester finds defects, he/she reports the bugs in GitHub Issues

Exploratory test:

The version of the project must be increased. New version must be deployed to test environment. Exploratory testing is executed for the tester to familiarize with the application and when sufficient time is not available for Regression test. Every issue found will be reported in GitHub.

Regression Test:

The version of the project must be increased. New version must be deployed to test environment. Regression testing for the whole application is executed before Release. Also, Regression testing for parts of the application is executed when functionality is added or changed. Every issue found will be reported in GitHub.

4.6. Test Report:

After execution of UAT and Smoke/Regression Test, Test report/s is prepared. Test Report includes: project sprint/version, Test Result, Pea chart, Test Result details, Defect details, Date, Environments and versions.

4.7 Defect Tracking:

Defect Details:

Title	Screen name and short description about the defect being reported, usually providing key words with which to identify and/or search for the defect.
Description:	Preconditions Environment / Device/ OS /Browsers. Test Steps Expected Result Actual Result
Priority	This field describes the impact the defect has on the work in progress and the importance and order in which a bug should be fixed
Component	Component in which the defect was found.

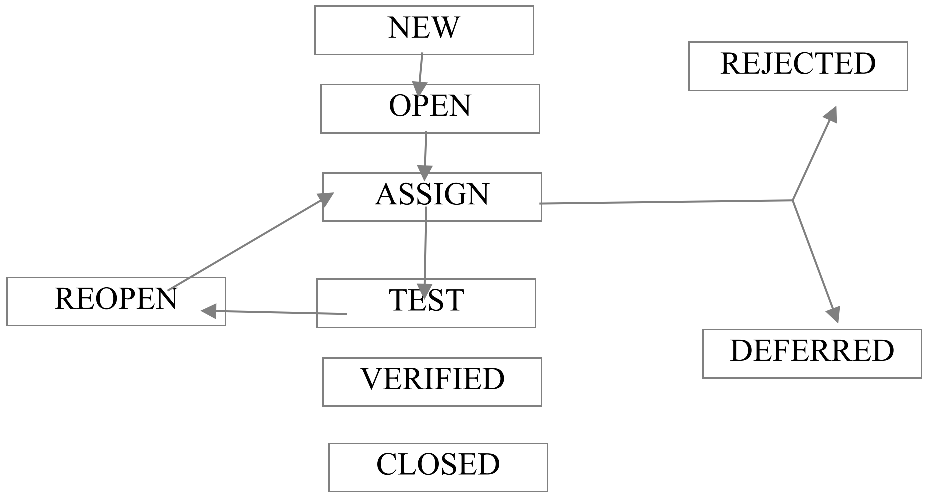
Defect Resolution:
When the defect is opened, it is assigned to the appropriate person, status is changed to "Assigned" .

Once the defect is fixed:

1. The developer to whom the defect is assigned will update the defect comments to document the fix that was made. User ID and Date is automatically added to the defect by clicking on "Add Comment".
2. The developer to whom the defect is assigned will change the status to "Fixed", after the implementation is finished.
3. The tester will retest the submitted defect.
4. If defect passes the retest, the tester will change Status to "Closed".
5. If the defect is not fixed, the tester will change the Status to "Reopened" and comment to defect.

Bug Life cycle:

All the issues found while testing will be logged into GitHub Issues bug tracker. Bug life cycle for this project is as follows:



Definition for Defect Priority:

Priority	Impact	Functionality	Performance	Stability	Graphical/UX
Blocker	Blocks development and/or testing work, production could not run.				
Critical	Significant impact on revenues, customer business operations or the security of the platform where no cost effective work is available.	Development or testing blocked and significantly impacting progress. Any serious regression of previously known/tested functionality Core/critical functionality missing or broken with significant impact on end users	Performance introducing significant overhead to development or test. Critical performance deficiency, system is rendered unusable. Frequent or	Crashes or unrecoverable hangs introducing major overhead to development or test.	

			permanent performance degradation. Easily reproducible.		
Major	Minor impact on revenues, customer satisfaction, customer, business operations or the security of the platform where no cost effective work is available	Critical functionality working but supporting features around it missing or broken with minor impact on end users.	Noticeable decrease in system performance, although system remains usable.	Minor or infrequently problems.	Graphical fault visible to most end users.
Minor	No impact on revenues, customer satisfaction, customer or business operations	Critical functionality working but supporting features around it missing or broken where a cost effective workaround is available with no real impact on end users	Minor or infrequent deterioration in performance.	Stability problem only encountered under conditions not expected to be found during normal use.	Minor graphical fault that would not be noticed by most users.
Trivial					Graphic change or issue where there is no impact to users, but business would prefer to align to existing products/marketing/artefects.

5. Test Environment:

NO	Environment	

1	Number of test machines	9
2	Licenced OS	9
3	Browsers	6

6. Test Tools:

6.1 Test Management Tool:

Notepad and GitHub

6.2. Bug Tracking Tool:

GitHub Issues

6.3 Automated Testing Tools:

Eclipse/Netbeans – Java ID
Java – Technology
Junit – Test Report Tool
TestNG – Test Report Tool
Selenium – Test Framework

6.4 Unit Testing Tools

Junit and Java
QA Team will provide test results for the executed tests.

6.5 Performance Testing Tools:

Performance testing will be performed using free tools and with the support of the dev team.

7. Deliverables:

Deliverable	For	Data
Test Plan	PM, QA M, Test Team	
Test Report	PM, Product Owner/Client	
Test Status Report	PM Product Owner/Client	

8. Entry Criteria

- New features must be functional tested, there is no blocker, critical defect.
- All test hardware and environments must be in place and free for testing.
- Fixed bugs must be retested and there is no open blocker and critical bug.
- Overall unit test coverage must be 40%.
- New test coverage must be 80% (for new development).

Total Estimated Test Cases:

View 1	Controller	View2
30	0	11

9. Exit Criteria:

- Test case execution must be 100% complete.
- Passed case rate should be 80%.
- Release cannot be shipped with any outstanding high business priority (blocker and critical) defects and/or " 9%" of major/medium priority (major, minor and trivial) defects.
- Performance of the system must not slow down with the introduction of new features.
- Test cases are executed considering Browser’s different versions
- User acceptance testing is completed.
- All the planned requirements must be met

10. Risks:

The following risks have been identified and the appropriate action identified to mitigate their impact on the project.
The impact (or severity) of the risk is based on how the project would be affected if the risk was triggered. The trigger is what milestone or event would cause the risk to become an issue to be dealt with.

	Description	Probability	Impact
1	Risk of Attrition	HIGH	VERY HIGH
2	Missing SW Licenses for development	MEDIUM	HIGH
3	Missing dedicated test environment	MEDIUM	HIGH
4	Distributed team cooperation	HIGH	HIGH
5	Team members changing	HIGH	HIGH

Document version:

	Type	*Description*	*Version*
Daniel – Andrei Fedeles	Document creation		1.0
Cosmin-Mihai Sparhat	Review		1.0