2016

# QM-Lab

## TESTING DOCUMENT

TEAM 3 – TESTING TEAM


| | |
|---|---|
| Mahmoud, Mahmoud | mtm464 - Lead |
| Wong, Jordan | jtw289 |
| Lee, Bengin | bel529 |
| Stankowski, Angela | ans723 |


| Version | Date Approved | Modified by | Brief Description |
|---|---|---|---|
| **1.0** | 06/02/2016 | Mahmoud, Jordan, Bengin, Angela | Rough Draft of Testing plan[1] |
| **1.1** | 22/02/2016 | Mahmoud, Bengin, Jordan | Official testing plan for ID-2 |
| **1.2** | 04/03/2016 | Jordan | Added Matrix and fixes |
| **1.3** | 20/03/2016 | Bengin | Created path coverage diagram (in different file). |
| **1.3.5** | 20/03/2016 | Jordan | Modified path coverage diagram and changed responsibilities |
| | | | |

[1] Draft was made by pen and paper, and it doesn't have a digital copy

Table of content: Page

# 1. Introduction

## 1.1 Purpose

The purpose of this document is to outline and detail the specifications of the testing phase involved in the progressing life cycle of the application. The document will contain a discussion of test hooks, levels of testing, testing scenarios as well as test cases. The document will also contain a traceability matrix.

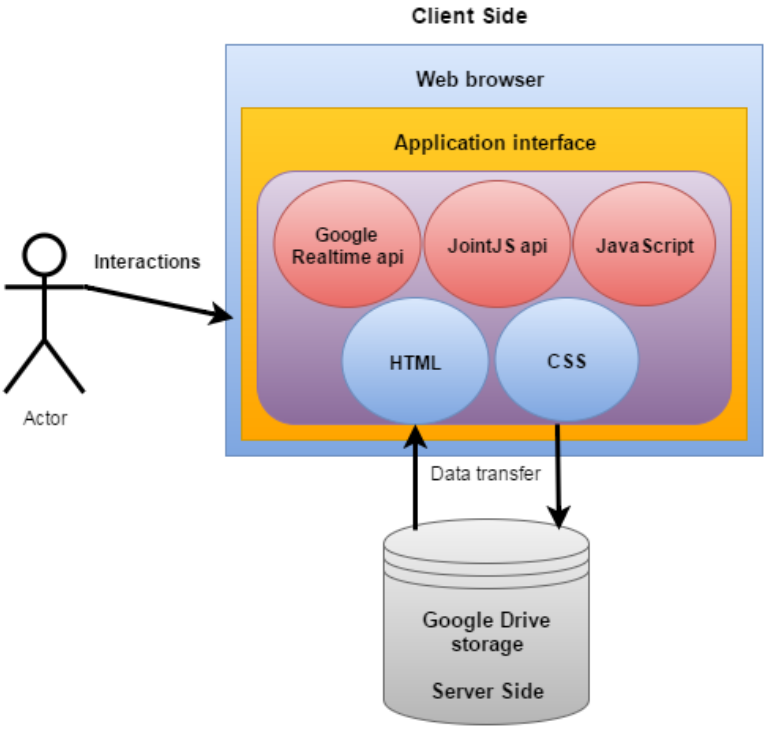
## 1.2 Scope of project

The intent of the software developed throughout this project is to serve clients who are interested in qualitative modeling. The product will also promote productivity by allowing real-time collaboration between clients who are geographically distant. The product will allow the user to share and store models through the Google drive platform.

## 1.3 Product breakdown

Based on requirements elicitation, the product will consist of two main components;  a client side and a server side. Many of the web applications use well-documented API's and frameworks that facilitate the development of sophisticated features. During the design phase of QM-Lab, the team decided on using many technologies that will eliminate risks involved and provide clear guidance on the plausibility of the requirements.

The client side will consist of the following technologies, HTML and CSS for the user interface, JavaScript and JointJS for interaction and behaviour response to/with user. Also, Google Realtime API, for interactions with the server and storing the documents created by the user to Google Drive. QM-Lab will require minimal server side setup because of the features that Google Realtime API provides that simplify the requests to the server.




## 1.4 References

Collaborative Qualitative Modeling Web Application – Nathaniel Osgood[2]
QM-Lab Software requirements specifications document[3]
QM-Lab Software design document[4]

[2] https://drive.google.com/drive/folders/0BxYBewaR90TAUGw2RXZsTUt0bjA
[3] https://docs.google.com/document/d/1iSLs_coktTi3wGdtz0-dOlY0SIl4WDATHY7qJFWMvQM/edit
[4] https://docs.google.com/document/d/1xUiSj1xfvJeiUvymp5dEMmiiBnbqZiR6RPn63iczwEY/edit

# 2. Testing plan

## 2.1 Objective

Testing and validation is an important phase in the cycle of software development. To ensure correctness, reliability and durability of an application, tests with specific scopes must be placed in order to test the various aspects and component of the application. The testing plan outlined below will allow for the software to be tested extensively and guarantee a fully functioning system without any defects.

## 2.2 Goals

The main goal of this testing plan is to test the functionality of each component and feature of the system in order to verify their correctness. A secondary goal will be testing the system to identify and report defects associated with any components of the system or the environment. Other goals will include, verifying software portability and ease of use, persistence and fault tolerance across multiple platforms.

## 2.3 Strategy

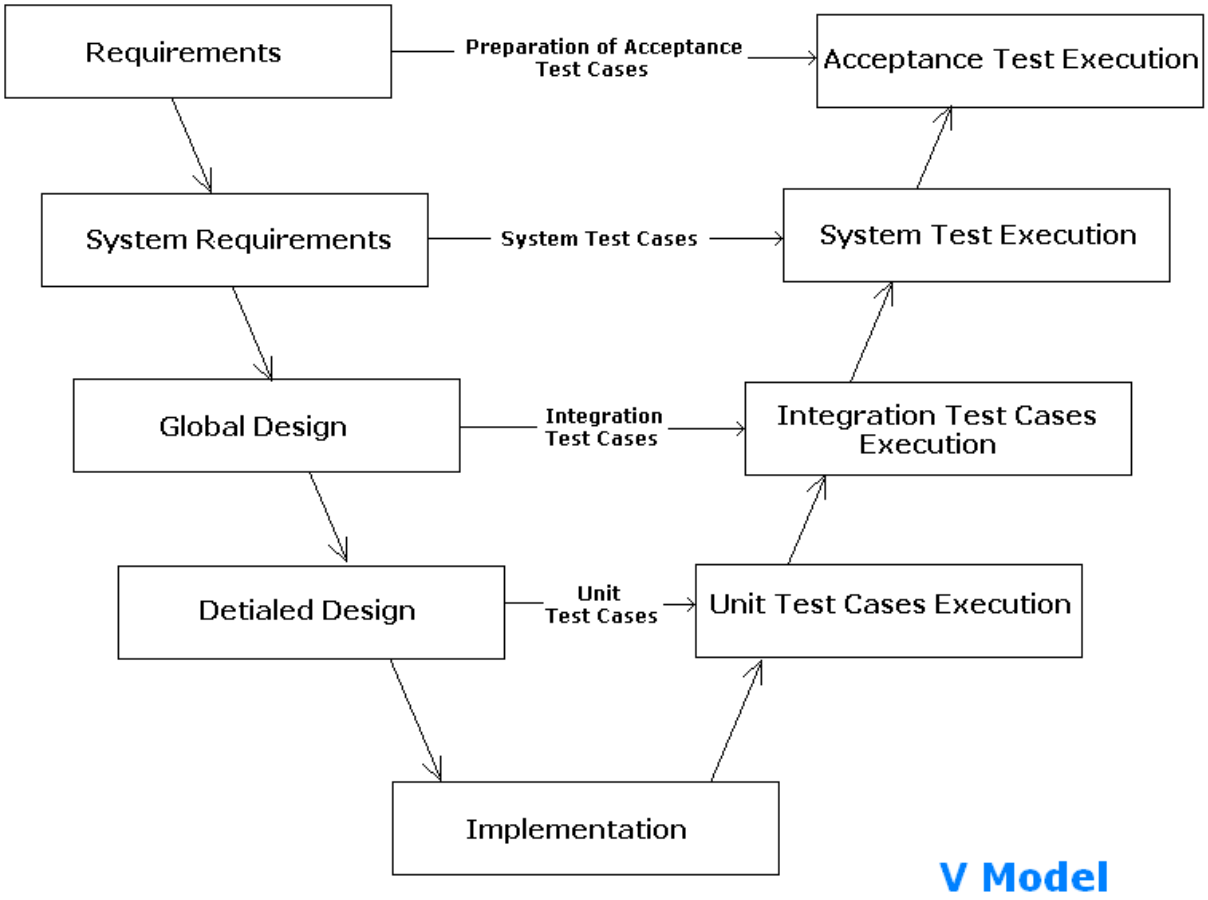
### 2.3.1 Test Harness

The testing team along with the build master will configure git hooks and Travis CI scripts to automate the execution of test scripts. The test harness will encapsulate executing the test scripts within the perspective test libraries.
The test libraries will contain test cases as well as test date or fixtures. The test harness will be responsible for generating reports that will indicate to the tester the state of the tests and the components/defects being tested. The majority of the test cases will be automated which will eliminate the need for having dedicated human testing. Human or manual testing will be employed to test specific components that cannot be automated using the testing tools.

### 2.3.2 Test levels

Due to the nature of the project, it will need to be subjected to various aspect-specific tests, such as integration and regression tests. The levels outlined below will ensure that the software is tested during the implementation phase and after it as well. The manner in which the test levels are structured will allow the testing the team to work more efficiently and focus on the main tasks assigned for the mini or final milestone.

## 2.4 Testing levels




Throughout the life cycle of this project the testing structure and plan will adhere to the steps outlined by the V-Model. The benefits gained by following the V-Model far outweigh its disadvantages. The V-Model is well established and rigid enough to allow structured progress and meet expectations of the development and the testing phase. The Testing levels are structured in such a way that each component is tested first by the developers using unit tests and then the full test suite deployed by the testers.

All of the tests will be done on the deployment machine in order to eliminate 'but it builds on my machine' syndrome.

### 2.4.1 Alpha testing

The developers will perform alpha testing in order to ensure the correct interactions and behaviour specified by the design team. Unit tests will need to be written by the developers with accordance to the detailed design plans. TDD is highly encouraged because it will help to clarify any problems associated with the design plans early on. Using TDD will also guide the testing team to focus more effort on the high-risk components and less on the smaller parts done by the developers. The implementation team is encouraged to use tools such as QUnit framework. QUnit will facilitate faster production of reliable unit tests.

During the alpha testing phase, the testing team will setup and prepare test scripts and data fixtures as well as mocking frameworks.

### 2.4.2 System and Integration testing

Prior to any commit, push and migrate operations to the project repository, the code will be subject to several smoke tests that will guarantee the integrity and the sanity of the newly applied code. Smoke tests will be outlined in test cases and test scenarios below. Smoke tests will be prioritized by functionality importance as well as execution time of such test.

At this level, feature tests will be also deployed, in accordance with the progress timeline set by the development time. The feature tests will be done after the success of all of the smoke tests. Scripts executed by the test harness and Travis CI will automate feature testing.

After the success of current and new features tests, the system will be subjected to end-to-end (E2E) cases, which will test the system as a whole. Many of those cases will be grouped under certain test scenarios that achieve the same result but via different paths.

### 2.4.3 Regression testing

Defects will be detected by reports generated by the test hooks. The testers will then report that defect on github issue tracking and bring it to the attention of the development team. Reports generated by the testing framework will be used in order to replicate the same conditions and then develop a regression test against the defect to verify if it has been fixed or not. Other defects will be detected by various other tests that test different components.

### 2.4.4 Performance and stress testing

Due to the nature of the application, the need to test the application concurrently on different machines using different test scenarios will prove to be a very important procedure in determining the functionality of the system. Such tests will rely on test scenarios associated with a single user and then using Selenium Grid to execute variants of the test cases on different browsers at the same time. To detect system load and performance, test scenarios will be executed through Selenium Grid and JMeter, which will simulate multiple concurrent requests to the server and report results of said tests.

### 2.4.5 User acceptance testing

During this phase of the testing, the software will be run against numerous use cases elicited from the stakeholder as well as implicit and explicit use cases that are deemed necessary. During this phase, each of the requirements specified in the software requirements document (SRD) will be tested in order of risk and importance. The Selenium framework provides various means of accomplishing such tasks. However, the use of an external framework that supports keyword testing such as Robot framework may prove necessary.

## 2.5 Assumptions

This plan was prepared with the following assumptions in mind:

a. The scope of the project is fixed
b. The technologies used to implement the system are final
c. The design plan put in place is final
d. The development timeline imposed by the implementation team is static and final
e. The limitations imposed by the design and implementation teams are final
f. System deployment environment suggested by the stakeholder is final

## 2.6 Features to be tested

### 2.6.1 Implicit Features

a. Internet connectivity
b. Google drive connectivity
c. Google drive storage limitations
d. JavaScript support
e. Ability to Drag and Drop
f. Error tolerance
g. Persistence and connection loss tolerance

h. Browser and machine limitations

### 2.6.2 Explicit Features

a. Creating a new diagram
b. Adding text objects to diagram
c. Adding various shape objects (agent, state, stock, flow) to diagram
d. Adding connections between various objects
e. Allowing for self-looping links as well polarity indication
f. Ability to group objects together as one single object
g. Ability to include/ upload pictures
h. Ability to load/save diagram
i. Ability to control file/document permissions
j. Ability to place comments on various components of the diagram
k. Undo functionality
l. Maintaining version history for later access
m. Allowing for different visibility levels of objects
n. Print layout

## 2.7 Priority rank of feature testing

The ranking of features to test will change with every milestone. The criteria on which features should be tested will based on the priority that the development team places on the feature. However, the testing team due to difficulty or complexity of feature may override the implementation team's ranking. Below is the preliminary ranking of feature testing.

| ***Feature - Implicit*** | **Rank level – 1 is easy, 10 is difficult** |
|---|---|
| Internet connectivity | Level – 3 |
| Google drive connectivity | Level – 4 |
| Google drive storage limitations | Level – 7 |
| JavaScript support | Level – 3 |
| Ability to Drag and Drop | Level – 3 |
| Error Tolerance | Level – 6 |
| Persistence and connection loss tolerance | Level – 7 |
| Browser and machine limitations | Level – 7 |

| *Feature - Explicit* | **Rank level – 1 is easy, 10 is difficult** |
|---|---|
| Creating a new diagram | Level – 3 |
| Adding text objects to diagram | Level – 3 |
| Adding various shape objects (agent, state, stock, flow) to diagram | Level – 3 |
| Adding connections between various objects | Level – 5 |
| Allowing for self-looping links as well polarity indication | Level – 5 |
| Ability to group objects together as one single object | Level – 7 |
| Ability to include/ upload pictures | Level – 7 |
| Ability to load/save diagram | Level – 6 |
| Ability to control file/document permissions | Level – 8 |
| Ability to place comments on various components of the diagram | Level – 5 |
| Undo functionality | Level – 7 |
| Maintaining version history for later access | Level – 8 |
| Allowing for different visibility levels of objects | Level – 6 |

## 2.8 Pass/Fail Criteria

The pass/fail criteria will differ based on the functionality being tested. However, the general consensus should be used:

a. High risk features must always pass the test
b. 90% of all tests on average should pass
c. Defects with medium, high, severe priority must be fixed and fully pass regression testing
d. Test coverage must be at least 90% of all proposed functionality

All test cases will contain more detailed pass/fail criteria that will help in generating concise reports.

## 2.9 Resources Estimates

| Test level | Hours spent |
|---|---|
| **Alpha testing** | 15 hrs |
| **Mocking** | 10 hrs |
| **Smoke tests** | 10 hrs |
| **Feature tests** | 20 hrs |
| **System tests** | 20 hrs |
| **E2E test** | 20 hrs |
| **Regression tests** | 15 hrs |
| **Performance and load tests** | 10 hrs |
| **User acceptance tests** | 15 hrs |
| **Total** | 135 hrs |

There are other resources that will be consumed during the life cycle of the project. However, they are negligible by contrast to the time estimates. The time estimates are calculated with the most plausible risks in mind.

## 2.10 Risks

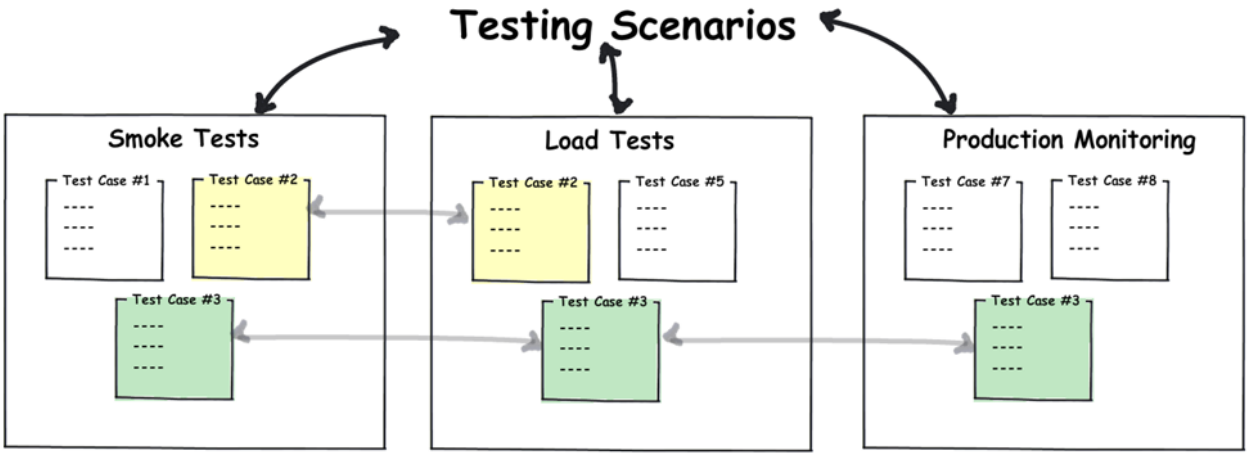
Risk assessment form – 1 is unlikely, 10 very probable:

| Risk | Likelihood | Impact | Detection Difficulty |
|---|---|---|---|
| **Implementation team plan change** | 6 | 8 | 8 |
| **Permanent Loss of personnel** | 5 | 8 | 6 |
| **Incompatibility of tools** | 4 | 8 | 4 |
| **Radical change of priorities and components' ranks** | 4 | 7 | 4 |
| **Misc. risk – hardware and illness** | 3 | 4 | 6 |

Risk response matrix:

| Risk | Response | Contingency plan | Trigger | Personnel responsible |
|---|---|---|---|---|
| **Implementation team plan change** | Mitigate – maintain constant communication with team leads | Work on parts not affected | Not resolved within 24hrs | Jordan |
| **Permanent Loss of personnel** | Mitigate – maintain inter team communications | Have backup personnel in other teams | Not resolved within 72hrs | Jordan |
| **Incompatibility of tools** | Reduce – have many prototypes | Try different tools and ask for help | Not resolved within 24hrs | Angela |
| **Radical change of priorities and components' ranks** | Reduce – evaluate SRD and SDD | Workaround till it is resolved | Not resolved within 24hrs | Bengin |
| **Misc. risk – hardware and illness** | Reduce – keep an eye out for such events | Workaround till resolved | Various – system freezes for more than 1 hrs | James |

# 3. Testing Scenarios




The testing scenarios that the system will be subjected to will be goal oriented, meaning that for a certain goal or action that the user can apply; a test scenario will be put in place such that it contains many variants of how that action is performed - test cases -. Our test cases can be found with the included Excel sheet documentation.[5]

The Excel spreadsheet will contain different sheets the first of which contains the master test scenarios sheet. The third sheet is the requirements organized by a REQ_ID, which will be used to uniquely identify requirements when referenced in test scenarios. The Document also contains a template for test cases.

# 4. Traceability Matrix

The traceability matrix is a system that designed to be progress orientated, meaning that it shows to the implementers and testers what their tests are currently covering and how many test cases are for said requirement. To view the traceability matrix it can be found in the Excel sheet documentation under the sheet[5] "Matrix."

# 5. Path Coverage Diagram

The path coverage diagram is done to demonstrate the basic flow of the decisions that can be made on the application. The path coverage diagram can help a tester understand what to test when tracing the flow on the path coverage diagram. To view the path coverage diagram you can view the photo file[6] located in the documents folder.

[5] Test_Scenario_doc.xlsx
[6] PathCoverage.png

## 6. Definitions, Acronyms, and Abbreviations

| | |
|---|---|
| SRD | Software Requirements Document |
| SDD | Software Design Document |
| TDD | Test Driven Development |
| E2E | End To End |