MÄLARDALENS HÖGSKOLA

Project Name: Yoshi
Acceptance Test Plan

Version 1.1

Revision History

Date	Version	Description	Author
24-12-14	0.1	Initial Draft	Rizwan Khalid
27-12-14	0.2	Filled section 1.3-1.5 and 2.2- 2.6	Rizwan Khalid
29-12-14	0.3	Completed writing test cases, ready for approval from the group members	Rizwan Khalid
15-01-15	1.0	Final adjustments and improvements	Rizwan Khalid
16-01-2016	1.1	Edits of the style, reference and table of content. Adding the list of tables and figures. Adding Captions and enumerating the tables.	Martin Anev

Table of Contents

List of Tables

1. Introduction

1.1 Purpose of this document

This document describes the plan for testing the developed Yoshi Vis software against the user requirements as defined in [PRD]. The purpose of this acceptance test is to make sure that the system developed during the Yoshi project complies with the requirements given in [PRD]. These tests should be executed in the Acceptance Test (AT) phase of the Yoshi project.

1.2 Document organization

The document is organized as follows:

- Section 1, *Introduction*
- Section 2, *Test Plan*
- Section 3, Acceptance Tests

1.3 Intended Audience

The intended audience is:

- The customer of the project
- The supervisors of the project
- Yoshi Team
- All related stakeholders
- Any developer with interest to continue or improve the project

1.4 Definitions and acronyms

1.4.1 Definitions

Keyword	Definitions
Community	A group having particular characteristics in common
Acceptance Test	A test conducted to determine if the requirements have met
Community type	Separation of a group according to its characteristics, e.g. in this project's case, community type can be Informal or Formal.
Yoshi	The name of software originally developed
Prototype	First working model of a product (software)

Table 1. Definitions

1.4.2 Acronyms and abbreviations

Acronym or abbreviation	Definitions
PRD	Project Requirements document
AT	Acceptance test
DD	Design Document
PPD	Project Plan document

Table 2. Acronyms and abbreviations

1.5 Documents

1.5.1 Reference Documents

Acronym or abbreviation	Definitions
[LMM]	'*"Let me measure my self!" Said Open-Source'* - D.A. Tamburri, E. di Nitto, P.Lago
[OSS]	'Organizational Social Structures for Software Engineering' - D.A. Tamburri, P. Lago, H.V. Vliet
[ULS]	'Uncovering Latent Social Communities in Software Development' – D.A. Tamburri, P. Lago, H.V. Vliet
[DOT]	'Discovering Open-Source Community Types: An Automated Approach' – A. Leta

Table 3. Acronyms and abbreviations

1.5.2 Applicable Documents

Abbreviations	Document name and version
[PRD]	Project Requirements document, Yoshi team, MDH/Pol. v 3.2.1
[DD]	Design Description Document v3
[PPD]	Project Plan document v1.1

Table 4. Abbreviation of Documents

2. Test Plan

2.1 Test Items

The software to be tested is the Yoshi Vis. Most of the functionality will be tested according to the user's requirements that can be found in [PRD].

2.2 Features to be tested

The Yoshi Vis system will adhere to the required requirements and will test the required features and design of 'Yoshi Vis'. The features of the software can be found in [DD].

2.3 Test deliverables

The following items must be delivered before testing begins:
Acceptance Test Plan.
Acceptance Test input data.
Software to be tested.

The following items must be delivered when the testing is finished:
Acceptance test report.
Acceptance test output data.
Problem reports (if necessary).

2.4 Testing tasks

The following tasks are necessary for preparing and performing the acceptance tests:
• Designing the acceptance tests.
• Ensuring that all environmental needs are satisfied for the acceptance tests.
• Performing the acceptance tests.

2.5 Environmental needs

The only major requirement of the implemented software is that it gets Input data file from already made "Yoshi" prototype, which include all the names of the repositories needed to evaluate.

2.6 Test case pass/fail criteria

Every test should describe the criteria that should be met to pass that specific test. Any discrepancies identified are classified as one of three types defined in table 5:

Severity	Description
Critical	Discrepancies that halt further program execution. Example: run-time errors that cause the system to lock up in an unexplained or unexpected manner.
Major	Discrepancies that cause the application not to perform as functionally required. Example: inability to print a report.
Minor	Discrepancies that are not considered critical or major. Examples: misspellings on a screen, ambiguous Help messages.

Table 5. Severity Rankings for Discrepancies

3. Acceptance Tests

The following are the test cases for all the requirements given in the [PRD].

3.1 Input Error Check

Test ID	T1.1
Test Name	Repository name input
Description	Enter the wrong name of a repository to see if it shows an error message
Pre-requisite(s)	The software is installed and has been executed
Input Specification	• Open the file "main.py" in the main directory. • Click ”Browse” to load the json data file • Type "anriod" in the input field of ”Repository name” • Clock “OK”
Output Specification	“Unable to find community anriod”

Table 6. Test of Wrong Repository Name Input

Test ID	T1.2
Test Name	Repository name input
Description	Enter the correct name of a repository to see if it shows the result
Pre-requisite(s)	The software is installed and has been executed
Input Specification	• Open the file "main.py" in the main directory. • Click ”Browse” to load the json data file • Type "android" in the input field of ”Repository name” • Clock “OK”
Output Specification	Check if the result has been shown or an error message has displayed

Table 7. Test of Correct Repository Name Input

3.2 Output Format

Test ID	T2.1
Test Name	Community type evaluation
Description	Enter the correct name of a repository to see if it shows the required result, i.e. the type of community.
Pre-requisite(s)	The software is installed and has been executed
Input Specification	• Open the file ‘’main.py’’ in the main directory. • Click ”Browse” to load the json data file • Type ‘’android’’ in the input field of ”Repository name” • Clock “OK”
Output Specification	Check if the result has shown the community type in textual form.

Table 8. Test of Community Type Evaluation

Test ID	T2.2
Test Name	Visual evaluation of the community
Description	Enter the correct name of a repository to see if it shows the result in a visual form
Pre-requisite(s)	The software is installed and has been executed
Input Specification	• Open the file "main.py" in the main directory. • Click ”Browse” to load the json data file • Type "android" in the input field of ”Repository name” • Clock “OK”
Output Specification	Check if the result has been shown result in a visual form (tree/graph)

Table 9. Test of Visual Evaluation of Community

3.3 Amount of community types

Test ID	T3.1
Test Name	Number of community type(s)
Description	Enter the correct name of a repository to see if it shows the result where the repository can belong to more than one community type.
Pre-requisite(s)	The software is installed and has been executed
Input Specification	• Open the file "main.py" in the main directory. • Click ”Browse” to load the json data file • Type "android" in the input field of ”Repository name” • Clock “OK”
Output Specification	Check if the result shows that the repository belongs to two types of community

Table 10. Test of Number of Community Type 1

Test ID	T3.2
Test Name	Number of community type(s)
Description	Enter the correct name of a repository to see if it shows the result where the repository can belong to only one type of community.
Pre-requisite(s)	The software is installed and has been executed
Input Specification	• Open the file "main.py" in the main directory. • Click ”Browse” to load the json data file • Type "MonoGame" in the input field of ”Repository name” • Clock “OK”
Output Specification	Check if the result shows that the repository belongs to only one type of community

Table 11. Test of Number of Community Type 2