CMPT 275 – Quality Assurance Plan
iRemember

Instructor: Dr. Herbert Tsang
TA: Jordon Phillips
Group Number: 11
Group Name: Double One
Due: October 12, 2012

Group Members:

Nicholas Pan
Charles Shin
Matt Numsen
Jake Nagazine
Steven Tjendana

Table of Contents

Table of Revisions

Revision	Status	Revision Date	Revised by
1.0	Created quality assurance document.	October 7, 2012	Matt Numsen Jake Nagazine Nicholas Pan Charles Shin Steven Tjendana
2.0	Added all sections.	October 8, 2012	Charles Shin
3.0	Revised sections, added table of revisions, abstract and table of contents.	October 10, 2012	Nicholas Pan
4.0	Revised document and addressed grammatical issues.	October 12, 2012	Matt Numsen Jake Nagazine Nicholas Pan Charles Shin Steven Tjendana

Figure 1 shows the Table of Revisions, which contains a revision number, status (description), date, and author for each revision made to the document.

Abstract

This quality assurance document will address the details in ensuring that our team develops a quality product. Ensuring that there is a plan to ensure quality important in software development and is part of our team's the guidelines. This project, especially on the quality side, has been influenced by a document on code ethics written by Gotterbam, Miller, and Rogenon [1]. The following document will cover software tools used in testing, internal deadlines, user acceptance testing, integration testing, software metrics, and additional testing resources.

Software Testing

Unit Testing

Unit testing of our project will ensure that methods within our application will be fit for use and also lessen the likelihood of errors. We, as developers will create unit tests, which internally will be used for white-box testing during our development process. In the case for our project, we will rely on XCode's built-in unit-testing environment. This built-in unit-testing feature is based on the SenTestingKit framework.

The following is components which we will examine:

- Logic tests: To test correct functionality of tested units. These tests will be focused on retrieving the intended result of tested units with given test cases.
- Application tests: To test the correct mapping between user-interface and intended functionalities.

To ensure quality of our software, we will try to maximize code coverage of unit tests.

Software/Application Testing

The XCode integrated development environment is bundled with an iPhone/iPad simulator which emulates the actual hardware device. Using it, we will run our application through this simulator for black-box testing. In order for real performance testing, we will be requiring in the later parts of the semester, a physical Apple iPhone device to for our deployment in the Apple AppStore.

Our team hopes that the utilization of white-box and black-box testing throughout development, we will be able to produce a polished quality application.

Additional Testing Tools

In addition to using the built-in tools from XCode, we will attempt to utilize the most testing tools available. The following will list some potential tools that we may use depending on our future implementation:

- Selenium (www.seleniumhq.org), web-browser automation for testing purposes.
- iPhoney (www.marketcircle.com/iphoney/), pixel accurate mobile web-browser environment by Safari.
- Apphance (www.apphance.com), mobile application quality tool.

Internal Deadlines

The internal deadlines are based off of the Gantt chart found in Appendix A. These dates are flexible and may change based on the priorities at any given time.

Task	**Start implementation**	**Due date**	**Number of days**	**Scheduled Unit test date**	**Scheduled System test date**
Assignment 3					
• Version1	2012-10-13	2012-10-22	9	2012-10-18	2012-10-20
Assignment 4					
• Version2	2012-11-01	2012-11-12	11	2012-11-08	2012-11-10
Assignment 5					
• Version3	2012-11-13	2012-11-26	13	2012-11-21	2012-11-23

Figure 2 shows the internal deadlines table, which contains a task, start date, due date, number of days, unit test due date and scheduled system test date.

User Acceptance Testing

Date, Time and location: 2012-11-24, 2:00 PM at Simon Fraser University, Burnaby Campus

Target Testers:

- Person(s) suffering memory loss
- Person(s) with no memory impairment

During the test, the tester will only be provided with an Apple iPhone and our memory-aid application. The following is a descriptive process for users during the user acceptance testing:

1. At the beginning of the test, there will not be any instruction manual on how to operate the application. However, within the application, there will be tips and basic instructions steps provided for first use, and/or inexperienced users. The "tutorial" phase will target how intuitive and easy the application is to use. This step will be applied to both testers of memory-loss and with no memory impairment. During the test, we will collect data on exact time to start and navigate through the application, understand the main functionality of the application and create an image with tags.

2. The next test will be held after memory loss symptom appears to tester with memory loss. Repeat step 1 only with tester with memory problem.

3. A brief tutorial on the overall functionality of the application to testers. This will test how easily they can follow the instruction and do what they are asked to do. We will collect data such as time spent, and number of questions asked to complete the task.

4. After memory loss symptom appears again, ask both testers to perform what we have thought during the tutorial. Once again, we will record time spent, and number of questions asked.

At the end of each step, the testers will be asked to fill out the closed question survey.

Integration Testing

Integrating testing will be performed from a bottom-up approach. All units of codes that passes unit tests will be integrated into larger unit which shares similar functionalities. Test cases to test larger unit will be every test cases used to perform unit testing of units belong to larger unit. By comparing result data with each individual unit test result, success of integration testing can be determined. The following is the proposed flow to test our system:

1. User class and Authorization class will be integrated.
   A. Create few random user data.
   B. Test if authorization granted using data created during step A.
   C. Test if authorization denied using random data excluding data from step A
2. Tag class integrated with Note and Draw classes.
   A. Create few mock-up Tag data with note and/or draw.
   B. Call each Tags to test correct note and/or draw can be retrieved
3. Integrate step 2 with TagFactory and TagManager class
   A. Using TagManager, create Tag with note and/or draw.
   B. Repeat step 2 to confirm correct functionality.
4. Integrate Picture class with PictureManager class
   A. Create a picture using PictureManager.
   B. Test if all functions such as geo-tag is set on picture.
   C. Load picture using PictureManager
5. Integrate step 3 and step 4
   A. Create a picture with tag(s).
   B. Search picture with a given tag(s).
6. Integrate everything together to perform system test
   A. For a given user, test to search for picture with tag(s).

Software Metrics

The XCode IDE and iOS SDK has feature called XCode assistant which measure project size and complexity. Using this tool, number of lines, number of classes, and number of files will be automatically reported. Using number of classes and methods, we can estimate complexity of software.

To help assist us in software metrics, we will use tools that are readily available to use such as the following:

- ProjectCodeMeter (www.projectcodemeter.com), estimates development cost & time, as well as measuring code quality, and team productivity.
- GitHub Code Analytics provides commit data and net added/removed code statistics.
- CLOC (Count Lines of Code, http://cloc.sourceforge.net/), counts physical lines of code in many different languages.
- OCLint (www.oclint.org), static code analysis tool for improving quality and reducing defects.

We have figured that size and complexity of software will grow with each iterative version of our program. The following table is our estimation of code growth:

Version	**Number of Lines**	**Number of Classes**	**Number of Files**
Version 1	4780	8	4
Version 2	8043	11	5
Version 3	10523	14	6

Figure 3.a shows the version, number of lines, number of classes, and number of files.

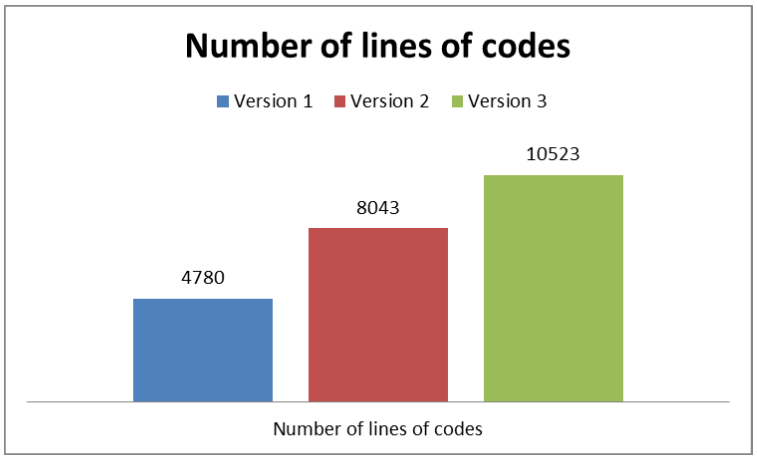


Figure 3.b shows a comparison bar-chart of number of lines between different versions.

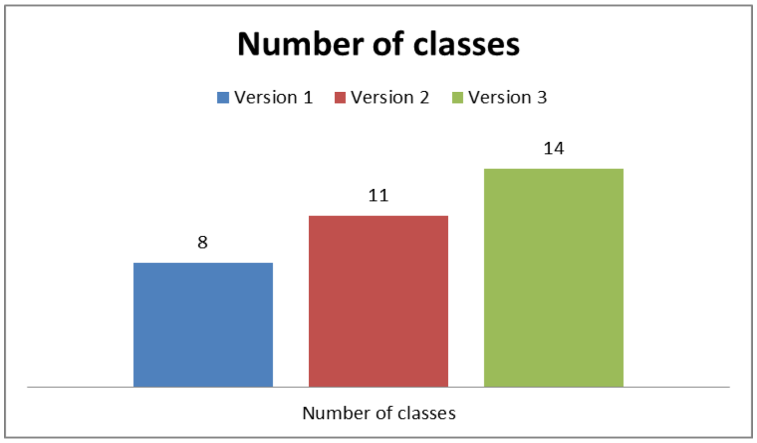


Figure 3.c shows a comparison bar-chart of number of classes between different versions.

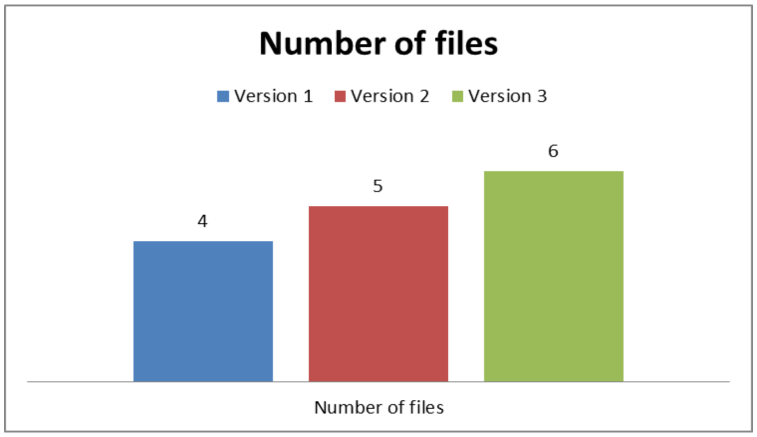


Figure 3.d shows a comparison bar-chart of number of files between different versions.

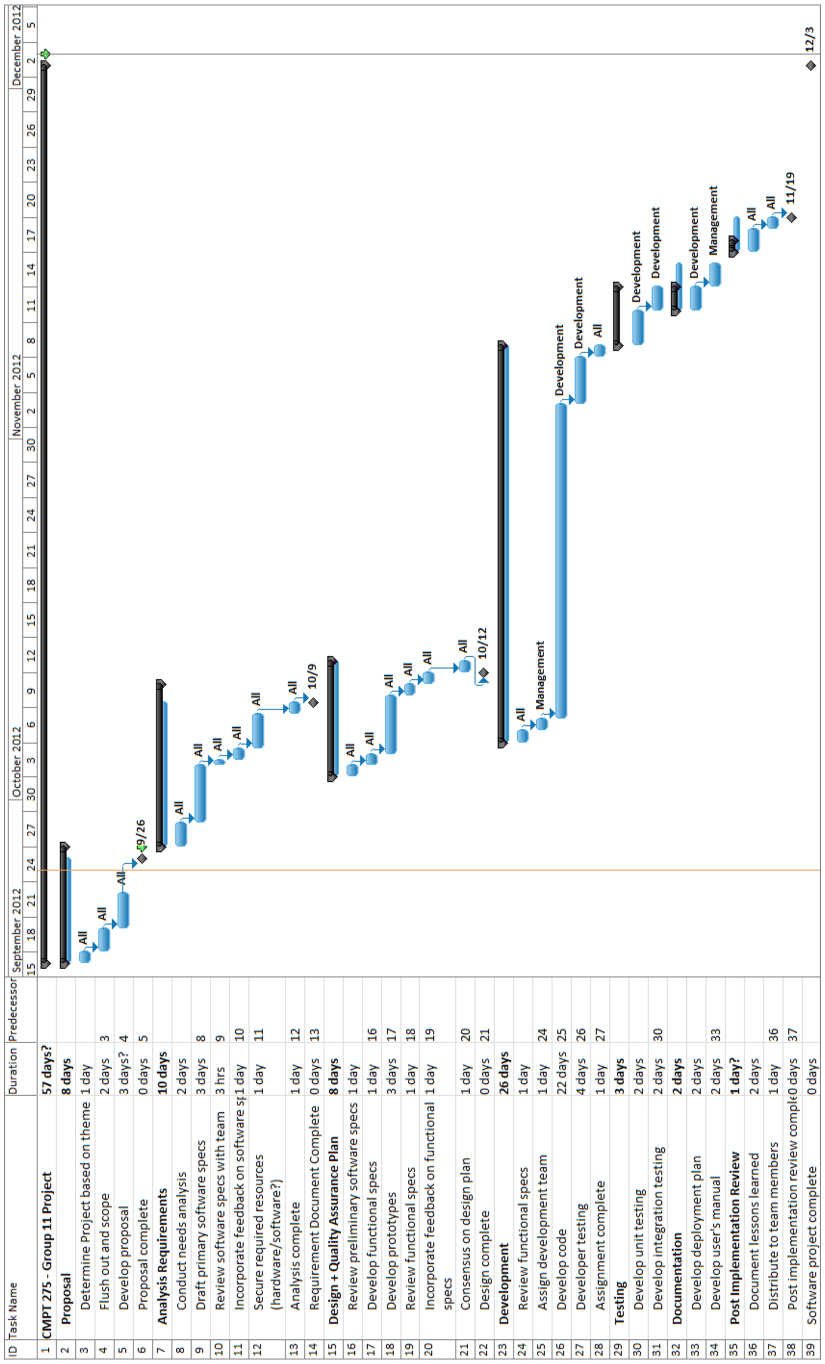
Additional Quality Assurance

To ensure complete quality assurance, we will use utilize many tools to provide quality code, which is as follows:

- Utilize GitHub’s pull request system for peer-code review before merging to main repository.
- Heuristic Evaluation – Inspection Form to evaluate the user interface of the application.
- Utilize a black-box testing system with external testers.

Appendices

Appendix A – Gantt Chart



ID	Task Name	Duration	Predecessors
1	**CMPT 275 - Group 11 Project**	**57 days?**	
2	**Proposal**	**8 days**	
3	Determine Project based on theme	1 day	
4	Flush out and scope	2 days	3
5	Develop proposal	3 days?	4
6	Proposal complete	0 days	5
7	**Analysis Requirements**	**10 days**	
8	Conduct needs analysis	2 days	
9	Draft primary software specs	3 days	8
10	Review software specs with team	3 hrs	9
11	Incorporate feedback on software sp	1 day	10
12	Secure required resources (hardware/software?)	1 day	11
13	Analysis complete	1 day	12
14	Requirement Document Complete	0 days	13
15	**Design + Quality Assurance Plan**	**8 days**	
16	Review preliminary software specs	1 day	
17	Develop functional specs	1 day	16
18	Develop prototypes	3 days	17
19	Review functional specs	1 day	18
20	Incorporate feedback on functional specs	1 day	19
21	Consensus on design plan	1 day	20
22	Design complete	0 days	21
23	**Development**	**26 days**	
24	Review functional specs	1 day	
25	Assign development team	1 day	24
26	Develop code	22 days	25
27	Developer testing	4 days	26
28	Assignment complete	1 day	27
29	**Testing**	**3 days**	
30	Develop unit testing	2 days	
31	Develop integration testing	2 days	30
32	**Documentation**	**2 days**	
33	Develop deployment plan	2 days	
34	Develop user's manual	2 days	33
35	**Post Implementation Review**	**1 day?**	
36	Document lessons learned	2 days	
37	Distribute to team members	1 day	36
38	Post implementation review comple	0 days	37
39	Software project complete	0 days	

Figure X shows the Gantt Chart representation of the schedule for the project.

References

[1] Freedom Software Foundation. (29 June 2007) *GNU General Public Licens*e, Version 3 Internet. Available: http://www.gnu.org/licenses/gpl.html

[2] E. Berkeley, "*The Social Responsibilities of Computer People*", Garden City, N.Y., pp. 461-471

[3] D. Gotterbam, K. Miller, and S. Rogenon, *"Software Engineering Code of Ethics, Version 3.0"*, Journal, Oct. 1997, vol.30, no.10, pp. 88-92