Test Plan

Version 1.0 – 2008.10.24

Created 2008.10.14

Yahoo! Property View

Rob Shaw	- Team Leader
Jacob McDorman	- Project Leader
Robert Read	- Technologist
Brad Van Dyk	- Editor

Table of Contents

[1] Introduction

[1.1] Document Scope

This document describes the testing plan that will be put into place and followed by Team E working on the Yahoo! Property View widget. This document contains a detailed outline of the guidelines that will be followed as well as testing processes and test cases that will be used in conjunction of developing the Yahoo! Property View widget.

Section two of this document pertains to Software Quality Goals. Described in this section are metrics and tools that will be used by the development team to assess the quality of the software that is produced.

Testing processes (section three) are described in this document with information relating to how the development team will doing version control as well as code reviews on the project as a whole.

The final section of the testing plan document pertains to test cases and how they will be used in conjunction with development of our components.

[1.2] Intended Audience

This document is **not** intended for widget users and will not give any information on how to use the widget. This document is intended for stakeholders (persons affected by the project) and the software development team (Purdue Team E).

[1.3] Project Identification

As stated in the requirements document 1.4.1, this project will provide a solution to Yahoo's difficulty with accessing data collected from the various systems offered by Yahoo. The domain of this widget is statistic analysis, and as such, the motivation of this software is to aid the user with analysis by providing trending data, volume data, and alerting mechanisms.

[2] Software Quality Goals

[2.1] Overview

The Property View widget shall be developed to be used internally by Yahoo!. Therefore, the code should be maintainable, well-documented, and easy to read. Code inspections shall be done to assess whether or not any vulnerabilities exist within the system that could harm the user or the Yahoo! network. The Property View widget shall be developed such that any SQL statements that are sent to the Yahoo! Oracle database will be checked for malicious attempts such as SQL injection.

[2.2] Metrics

Metrics shall be performed on the code at intervals spread out throughout the remainder of the development cycle.

A metric that will be used in the analysis of the Property View widget's code is code coverage.

Code coverage describes the degree to which the source code of a program (Property View widget) has been tested. It is a form of testing that inspects the code directly and is therefore a form of white box testing. The coverage criteria that will be used to measure how well the program is being used are listed as such:

- *Function coverage:* Has each function in the program been executed?
- *Statement coverage:* Has each line of the source code been executed?
- *Decision coverage:* Has each control structure (ex: if statement) evaluated to both true and false?
- *Condition coverage:* Has each Boolean sub-expression evaluated both to true and false?
- *Path coverage:* Has every possible route through a given part of the code been executed?
- *Entry/exit coverage:* Has every possible call and return of the function been executed?

Coverage criteria will be evaluated with the Fortify software mentioned earlier as well as a program called JSCoverage which was developed to test the coverage of Javascript code. There are also some other Javascript Coverage programs which have been researched and could potentially be used.

Another metric that will be used in the analysis of the Property View widget is *source lines of code* (SLOC). SLOC will be used to measure the size of the program by counting the number of lines in the source code. This will be used to give a rough estimation of the amount of time that has already been required to develop the widget as well as estimations as to how much effort shall be needed on the remaining components.

Cohesion and coupling are also metrics which be used to help analyze the Property View widget. This will be analyzed in an effort to determine how much each of the components rely on each other as well as how strongly related the responsibilities of a given class are.

[2.3] Tools

The Fortify Software package will be used as a tool to analyze code written in the development of the Yahoo! Property View widget. The current program under the software package called the Fortify Source Code Analyzer (SCA) is a tool in which the team has access to and shall use when writing code. The Fortify SCA examines every line of code and every program path to identify hundreds of different types of potentially exploitable vulnerabilities.

JSCoverage is a tool that shall be used to help analyze the coverage code of the Property View widget. It is a tool that shall be run periodically to see how well the code is executing. Reports shall be created based on the results of the test.

[3] Testing Processes

[3.1] Overview

As discussed in the Overview of Software Quality Goals (current document, section 2.1), the Property View widget shall be developed to be used internally by Yahoo!. The development team shall adhere to those guidelines (of the SQG section) that were outlined as well as follow the practices described below.

[3.2] Version Control

According to section 3 (Version Control) of the Implementation Plan, a CVS repository shall be set up by the development team to be used in conjunction with construction of the Property View widget. Each team member shall be given a separate branch in which to develop code in. All team members shall have the ability to view the code of another team member. Individual team members will be given a task (that can possibly be shared with other team members depending) and during ongoing development of said task, the code shall be updated within the user's branch.

Versioning and committing of code to the head branch will be incremented upon the completion of a new code base that has passed code reviews, unit testing, and regression testing. For a new version of software to be incremented, it must meet one or more of the criteria discussed in Section 3 of the Implementation Plan which are:

1. Yields better test results.
2. Fixes known bug(s).
3. Provides an improved look and/or feel.
4. Adds new features (not previously implemented).

As stated in the Implementation Plan, Eclipse shall be used by each team member that has access to the team's CVS repository.

[3.3] Code Reviews

All code developed by team members must follow the guidelines outlined in the Implementation Plan describing coding standards (section 5). These guidelines have information pertaining to the styling and commenting of code.

Code reviews shall be a very important task involved with the development of the Property View widget. Following the Gantt Chart provided in the Implementation Document (Section 2), there is time allotted for testing of features that were recently implemented. This time frame will be used to conduct code reviews among other types of testing involved with development of the widget.

As code changes are made by a certain team member, there will be a document in which must be filled out to highlight changes that are made with references to line numbers and how they relate to the requirement and design documents. The team member must do individual testing before submitting the code to be processed in a code review.

[3.4] Other Test Processes

Tests to be run shall be automated. Scripts shall be written to test individual components. However, there shall be some manual testing of the GUI interface that cannot be tested with automated scripts.

[4] Test Cases

[4.1] Unit Test Cases

Will need some separate functions to help with unit testing. One function will be needed to dump all the data pulled from a SQL statement to a file for quick and easy viewing.

[4.1.1] Unit Test Case 1: DatabaseComm

Module or Class: Database Communicator

1. Testing: connect() method
2. Testing: query(String) method
3. Testing: insert() method

[4.1.2] Unit Test Case 2: UserDataClass

Module or Class: User Data

1. Testing: loadUserPreferences() method
2. Testing: SaveUserPreferences() method

[4.1.3] Unit Test Case 3: TabDataManager

Module or Class: Tab Data Manager

1. Automatic Updates
    a. Dump all data from MMT to a text file using SQL statements pulled from the user table.
2. Testing: enableAlerts(boolean) method
3. Testing: stopAlertsUntil(Time) method

[4.1.4] Unit Test Case 4: TabViewer

Module or Class: Tab Viewer

1. Refresh function.

[4.2] Integration Test Cases

Integration testing will be done using Usage Model testing. This strategy relies heavily on the team members to follow the isolated unit testing for individual components in which they work on. The goal of the strategy is to avoid redoing the testing already done, and instead flesh out problems caused by the interaction of the components in the environment.

[4.2.1] Integration Test Case 1: Testing User Data

Preconditions: User is valid in database with some property data.

#	Description	Pass	Fail
1	Call is made to connect to database with user name. Result: Successful connection, no errors logging in.		
2	Call is made to grab data from MMT relating to current user name. Result: User data is downloaded and stored in a temporary text file (this is to check whether or not data was downloaded correctly).		

[4.2.2] Integration Test Case 2: Testing Added/Removed Property

Preconditions: User is valid in database with some property data.
Property has been added to user's table.
New data is available to be added to test new property.
Connection has already been made with current user.

#	Description	Pass	Fail
1	Call is made to refresh user information from user table. Result: Current data for user as well as new SQL statements are downloaded.		
2	Call is made to get information from MMT using user's SQL statements with data filled in pertaining to properties in which they belong. Result: A dump of all data is created in a temporary text file to check for the validity of data. Information from the newly created property shall be downloaded.		

3	Tester removes user from property in users table. Result: User belongs to one less property.		
4	Call is made to refresh user information from user table. Result: Current data for user as well as new SQL statements are downloaded.		
5	Call is made to get information from MMT using user's SQL statements with data filled in pertaining to properties in which they belong. Result: A dump of all data is created in a temporary text file to check for the validity of data. Information from the newly created property shall not be downloaded.		

[4.2.3] Integration Test Case 3: Stop Alerts until Specified Time

Preconditions: User is valid in database with some property data.
Data is added to test new anomaly alert.
Connection has already been made with current user.

#	Description	Pass	Fail
1	Date and time are specified in a variable (make sure it's only a few minutes in the future for easy testing purposes). Call is made to change current user settings. Result: Current users next specified date/time to receive alerts is updated.		
2	Add new anomaly information to the database (must be a part of a property that the current user belongs to). Check the alerts log. Result: No new alerts should be displayed.		
3	Wait until date/time specified in variable. Call checkForAlerts(). Check the alerts log. Result: New alert should be created with information pertaining to data that was added in step 2.		

[4.2.4] Integration Test Case 4: Testing Update of User Settings

Preconditions: User is already logged in.
User has settings already defined.

#	Description	Pass	Fail
1	Set receiveAlerts variable to 'true'. Call updateUserSettings(). Result: Message of successful change returned.		
2	Add new information to anomaly database. Call checkForAlerts(). Result: New alert should be created with information pertaining to property in which user belongs.		
3	Set receiveAlerts variable to 'false'. Call updateUserSettings(). Result: Message of successful change returned.		
4	Add new information to anomaly database. Call checkForAlerts(). Result: No new alerts should have been created..		

[4.2.5] Integration Test Case 5: Test Login and Connection with Read Capabilities

Preconditions: Database login is valid to connect to database.

#	Description	Pass	Fail
1	Set oracleDBLoginName = "----". Set oracleDBPassword = "----". (The two above values are used to make the initial connection to Oracle database. These are different from usernames.) Call openOracleConnection(). Result: Message of Connection Open success.		
2	Create SQL statement to dump first 10 users in database. Call runOracleCommand(sqlSTM).		

	Result: First 10 users in Oracle Database User Table returned.		

[4.3] System and User Interface Test Cases

[4.3.1] UI Test Case 1: Logging in

Preconditions: User has started the program

#	Description	Pass	Fail
1	User tries logging in with no username typed in the login display Result: Error message appears telling user to enter a username		
2	User tries logging in with a username not found in the contacts table of the database Result: Error message appears saying that the username is not valid		
3	User logs in with a valid username Result: Users tabs according to MMT are created and any options for that user saved from a previous login are loaded otherwise if the user has not previously logged in the default options are loaded		

[4.3.2] UI Test Case 2: Manual Refresh

Preconditions: User has logged in

#	Description	Pass	Fail
1	User clicks refresh button for specific tab Result: Tab's data is synchronized with the database data it is displaying and graphing. Any alerts arising from updated data are handled according to user's specifications for alerts regarding the tab		
2	User clicks refresh all button Result: The data for each tab is synchronized with the database. Any alerts arising from the updated data are handled according to the alert settings chosen by the user for the tab(s) receiving alerts		

[4.3.3] UI Test Case 3: User edits data that is writable to the database

Preconditions: User has logged in
User has a tab open where there is an editable field

#	Description	Pass	Fail
1	User inputs data of an invalid data type into editable field		

	Result: Error message appears saying the input value is an invalid data type		
2	User inputs valid data into editable field Result: The table entry in the database that corresponds to the field edited by the user is updated with the user's input		

[4.3.4] UI Test Case 4: Setting Alert Options

Preconditions: User has logged in
User has the settings tab open.

#	Description	Pass	Fail
1	User inputs data of an invalid data type into editable field Result: Error message appears saying the input value is an invalid data type		
2	User inputs valid data into editable field Result: The table entry in the database that corresponds to the field edited by the user is updated with the user's input		

[4.3.5] UI Test Case 5: Receiving Alerts

Preconditions: User has logged in
User has a 'receive alerts' optioned set to true.

#	Description	Pass	Fail
1	Insert new information into database (creating anomaly) for property in which user does not belong. Result: No alert shall be displayed.		
2	Insert new information into database (creating anomaly) for property in which user does belong. Result: Alert shall pop up with information pertaining to the anomaly.		

[4.3.6] UI Test Case 6: Stop Alerts Until Specified Time

Preconditions: User has logged in
User belongs to a property.

#	Description	Pass	Fail
1	User selects time and date that is in the past (relevant to the current date and time).		

	Result: Error message shall be displayed saying a date was specified that is not in the future.		
2	User selects time and date that is in the future. Result: Pop-up will inform user that all alerts will be held until specified date as well as updating label that states when the next alert shall be processed.		

[5] Appendix

Definitions:

Property View widget →The application being developed for Yahoo!.

External Links:

Tools Used by Development Team:

Fortify Software → http://www.fortify.com/

JSCoverage → http://siliconforks.com/jscoverage/