**[GuiMi]  
*Testing Documentation***

|  |
| --- |
| **Final Report** |

**Document Information and Approvals**

**version history tab**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version #** | **Date** | **Revised By** | **Reason for change** |
| **1.0** | **15/June/2014** | **Ren Jiayue(1152759)**  **Zhao Qing(1152683)**  **Sheng Wen(1152729)**  **Hu Mian(1152715)** | **Final Report for Software Testing** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

1. Introduction 1

1.0 Scope 1

1.1 Test Objectives 1

1.2 Testing Goals 1

1.2.1 Quality 1

1.2.2 Reliability 2

2. Test Methodology 2

2.1 Entrance Criteria 2

2.2 Exit Criteria 2

2.3 Test Execution 2

2.4 Test Case/Script Development 3

2.5 Defect Reporting 3

3 Unit Testing 4

3.1 Unit Testing for “Login” 4

3.1.1 Test Case Design 4

3.2.2 Test Codes Implementation 5

3.2 3 Results & Analysis 6

4. Functional Testing 7

4.1 System Overall Architecture and Functional Modules Introduction 7

4.2 Functional Testing Test-case Design 8

4.2.1 Login Module 8

4.2.2 “Fashion Trend Circle” 9

4.2.3 “My Closet” 10

4.2.4 “My Space” 10

4.5 Functional Testing Result 11

5. Regression Testing 12

5.1 Regression Testing Goal 12

5.2 Regression Testing Method 12

5.3 Regression Testing Procedures 12

5.4 Regression Testing Result 13

6. Integration Testing 14

6.1 Integration Testing Goal 14

6.2 Integration Testing Method 14

6.3 Integration Testing Result 14

7. Interface Testing 14

7.1 Interface Testing Goal 14

7.2 Interface Testing Procedure 15

7.2.1 Interface Demo 15

16

7.2.2 Measurements and Assessment 16

7.3 Interface Testing Result 17

9. Performance testing 17

9.1 Performance Metrics 17

9.2 Performance Testing Tools 18

9.3 Performance Testing 18

9.3.1 Result 18

20

21

9.3.2 Analysis 21

10. User acceptance testing 21

11. Go/No-go Meeting 22

12. Test Environment 22

12.1 Software Requirements 22

12.2 Testing Platform 22

13. Assumptions and Risks 22

13.1 Assumptions 22

13.2 Risks 23

Additional Project Documents 23

Sign-off and Acknowledgement 24

Test Director – Defect Tracking Process 24

1. Introduction

## 1.0 Scope

The overall purpose of testing is to ensure the “GuiMi” application meets all of its technical, functional and business requirements. The purpose of this document is to describe the overall test plan, strategy, and results for testing the “GuiMi” application. The approach described in this document provides the framework for all testing related to this application. Individual test cases will be written for each version of the application that is released in this document. This document will also be updated as required for each release.

## 1.1 Test Objectives

The quality objectives of testing the “GuiMi” application are to ensure complete validation of the business and software requirements:

* Verify software 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 (currently we only have a single version of [Jspxcms](http://bbs.jspxcms.com/index.php) project, thus specific regression test is not included in this document, but will be included once the later versions come out)
* Manage defect tracking process
* Provide test metrics/testing summary reports
* Ensure the application is certified for release into real environment
* Schedule Go/No Go meeting
* Require sign-offs from all stakeholders

## 1.2 Testing Goals

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

Another goal is to make the tests repeatable for use in regression testing during the project lifecycle, and for future application upgrades. A part of the approach in testing will be to initially perform a ‘Smoke Test’ upon delivery of the application for testing. Smoke Testing is typically an initial testing effort to determine if a new software version is performing well enough to accept it for a major testing effort. For example, if the new software is crashing frequently, or corrupting databases, the software is not in a stable enough condition to warrant further testing in its current state. This testing will be performed first. After acceptance of the build delivered for system testing, functions will be tested based upon the designated priority (critical, high, medium, low).

### 1.2.1 Quality

Quality software is reasonably bug-free, meets requirements and/or expectations, and is maintainable. Testing the quality of the application will be a two-step process of independent verification and validation. First, a verification process 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 that requirements are covered. The overall goal is to ensure that the requirements are clear, complete, detailed, cohesive, attainable, and testable. In addition, this helps to ensure that requirements are agreed to by all stakeholders.

Second, actual testing will be performed to ensure that the requirements are met. The standard by which the application meets quality expectations will be based upon the requirements test matrix, use cases and test cases to ensure test case coverage of the requirements. This testing process will also help to ensure the utility of the application – i.e., the design’s functionality and “does the application do what the users need?”

### 1.2.2 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 software development cycle, regression tests will be performed on all iterations of the application (Still not included owing to the single version available).

2. Test Methodology

## 2.1 Entrance Criteria

* All business requirements are documented and approved by the business users.
* All design specifications have been reviewed and approved.
* Unit testing has been completed by the development team, including vendors.
* All hardware needed for the test environment is available.
* The application delivered to the test environment is of reliable quality.
* Initial smoke test of the delivered functionality is approved by the testing team.
* Code changes made to the test site will go through a change control process.

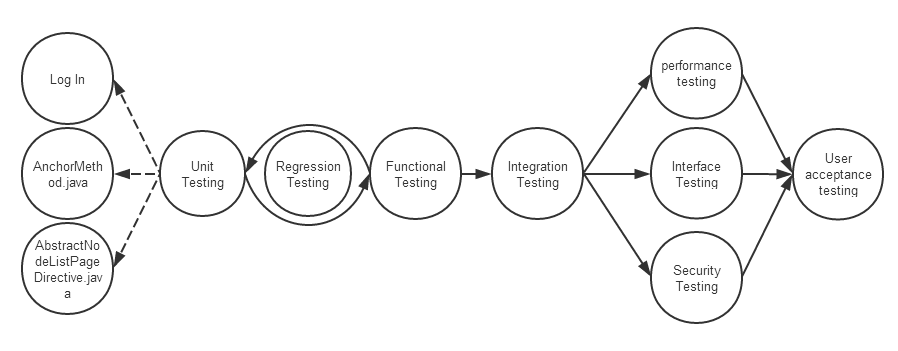
## 2.2 Exit Criteria

* All test scenarios have been completed successfully.
* All issues prioritized and priority 1 issues resolved.
* All outstanding defects are documented in a test summary with a priority and severity status.
* Go/No-go meeting is held to determine acceptability of product.

## 2.3 Test Execution

The test execution phase is the process of running test cases against the software build to verify that the actual results meet the expected results. Defects discovered during the testing cycle shall be entered into Mercury Test Director for Quality Center. Once a defect is fixed by a developer, the fixed code shall be incorporated into the application and regression tested.

In this document, the test execution is divided into several phases. The following graph aptly describes our testing plan:



For more specific test execution information, see next sections from **Unit Testing** to **User Acceptance Testing.**

## 2.4 Test Case/Script Development

Test case/script design is the central focus of a software quality assurance process. A test case or script is defined as a written specification describing how a single or group of business or system requirement(s) will be tested. The test case or script consists of a set of actions to be performed, data to be used, and the expected results of the test. The actual results of the test are recorded during test execution. Test cases or scripts will also be updated as testing proceeds.

Test Cases/Scripts written for this project include the following:

* Software requirement ID
* Requirement description
* Any dependencies and/or special set-up instructions required for performing the test
* Test description
* Expected results

## 2.5 Defect Reporting

Issues/defects are tracked for resolution with the following guidelines:

* Issues will be reported based upon documented requirements.
* Issues will be tracked by the testing team, reported to the test lead and entered into Mercury Test Director for Quality Center.
* Issues will be fixed by the development team based on the priority/severity assigned by the test lead.
* All critical/priority 1 defects will be fixed before release to production.

See the Defect Tracking Process at the end of this document for detailed instructions on how to log and track defects in Test Director.

Note: Since the test document is based on the completion of the Jspcxms system and is only served to assess its overall performance, we simplify this process to the summary of all bugs we find during this testing cycle. For detailed defect report of each testing aspect, please refer to the end of each section from “3. Unit Testing” to “10. Performance Testing”

3 Unit Testing

Unit testing is performed by the application developers testing in the development environment. This testing phase will have a **“white box”** perspective, which means the application developers know, and will be testing the internal logical structure of each software component.

During our project’s entire developing lifecycle, we insist on implementing unit tests on each modules we finish, either by writing simple codes inside the program to test the output, or by using tools like JUnit to conduct the unit tests more formally and reliably, and the choice depends on the module’s importance and complexity.

For example, from various perspective, “login” is definitely a function that is of high importance and to some extent, high complexity, which is why we choose to use JUnit for unit testing and the following are the specific process.

## 3.1 Unit Testing for “Login”

### 3.1.1 Test Case Design

#### 3.1.1.1 Coverage Type

Boundary Coverage

#### 3.1.1.2 Test Case Specification

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tested Methods** | **Methods Description** | **Test Case No.** | **Parameters** | **Test Case Description** |
| **executeLogin(String username, String password)** | This method is used in “com.guimi.LoginActivity.java” for tourists to login the system by providing username and password | 1.1 | username=null  password=null | Null value for both username and password |
| 1.2 | username=”admin”  password=null | No password provided |
| 1.3 | username=”admin”  password=”22222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222” | Long characters |
| 1.4 | username=“admin”  password= “•＆λσ△Δ@” | Special characters |
| 1.5 | username=“admin”  password=  “admin/\"'” | Password contains escape characters |

Table. Test Case Design for "Login"

### 3.2.2 Test Codes Implementation

Fig.6 is the codes for unit testing of “Login”. It contains all the test cases listed above in table.1.

For complete codes, please refer to **Appendix A**.

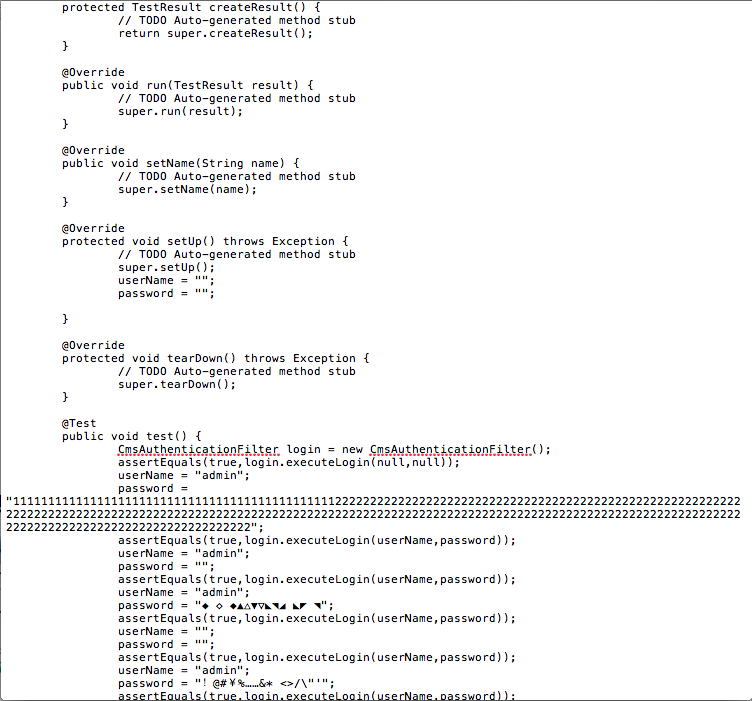


Fig. Codes Implementation for "Login"

### 3.2 3 Results & Analysis

#### 3.2.3.1 JUnit Execution Snapshot

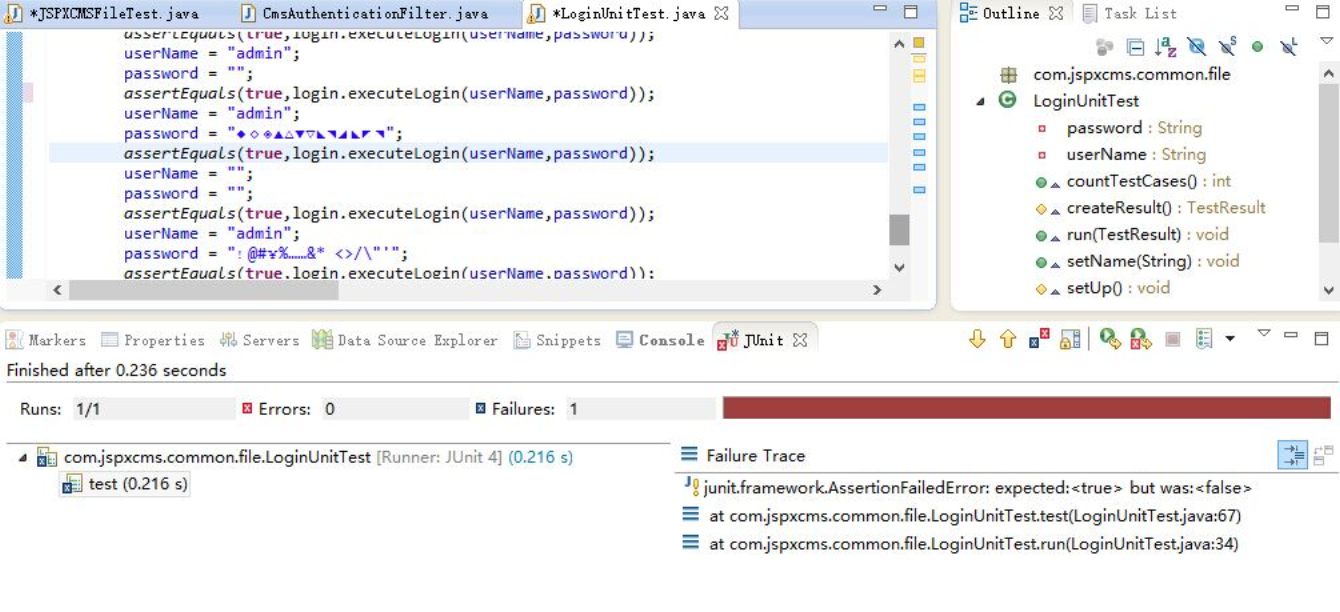


Fig. Junit Test Report for Login

From Fig. 7 we can see that there’s one failure, and failure trace reports “AssertionFailedError”, that the assertion was expected to return true but instead it returns false. This is caused by test case 1.4 where special characters are not correctly recognized.

#### 3.2.3.2 Detailed Results Report.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Tested Methods** | **Test Case No.** | **Parameters** | **Expected Return** | **Actual Return** | **Test Result** |
| **attemptLogin()** | 1.1 | username=null  password=null | Login Fail | Login Fail | Pass |
| 1.2 | username=”admin”  password=null | Login Fail | Login Succeed | Fail |
| 1.3 | username=”admin”  password=”22222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222” | Login Succeed | Login Succeed | Pass |
| 1.4 | username=“admin”  password= “•＆λσ△Δ@” | Login Succeed | Login Fail | Fail |
| 1.5 | username=“admin”  password=  “admin/\"'” | Login Succeed | Login Succeed | Pass |

Table. Test Result Report for Login

#### 3.2.3.3 Suggestions：

Restrict the length and character type of password for more robust performance.

4. Functional Testing

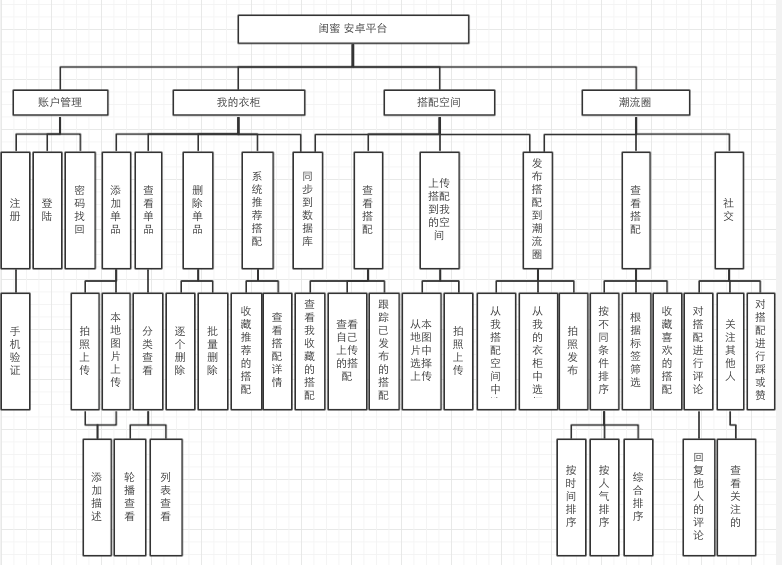
Functional testing, or “black box” testing, focuses on the functional requirements of the software. Functional testing is performed to confirm that the application operates accurately according to the documented specifications and requirements, and to ensure that interfaces to external systems are properly working.

## 4.1 System Overall Architecture and Functional Modules Introduction

We choose an open-source website: http://demo.jspxcms.com/, and deploy it in the local environment.

The overall functional modules architecture is as following picture:

(Unfortunately it is in Chinese ☹, sorry…)



## 4.2 Functional Testing Test-case Design

We design test-case as follows:

### 4.2.1 Login Module

* Equivalence Partitioning

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test-case Design | Test-case No | Username | Password | Expected Output |
| Correct username and password | 1.1 | admin | 123456 | Log in successful |
| password is empty | 1.2 | admin |  | Log in failed |
| Username is empty | 1.3 |  | 123456 | Log in failed |
| Incorrect username | 1.4 | aa | 123456 | Log in failed |
| Incorrect password | 1.5 | admin | 12345 | Log in failed |
| Username input is illegal | 1.6 | ?admin | 123456 | Log in failed |
| Password input is illegal | 1.7 | admin | \_123456 | Log in failed |

### 4.2.2 “Fashion Trend Circle”

|  |  |  |  |
| --- | --- | --- | --- |
| Test-case Description | Test-case No. | Action | Expected Output |
| Looking for more outfits | 2.1 | Scroll down | New outfits would be loaded continuously |
| Looking back at published outfits | 2.2 | Scroll up | Reload outfits pictures from cache |
| Checking the outfits details | 2.3 | Click on the outfit’s picture | Jump to another activity that displays the details of the outfits |
| “like” the outfits | 2.4 | Click on the outfit’s “like” button | Out of the window to save |
| “dislike” the outfits | 2.5 | Click on the outfit’s “dislike” button | Prompts the user is not logged |
| Add the “outfits to collection” | 2.6 | Click on the outfit’s “favorite” button | Successful registration |
| Publish outfit to fashion trend circle from local album | 2.7 | Click the publish button on the up-right of the screen and select “publish from local”, select tags and click “finish” | A toast pops out saying “publish succeed”, and the local outfit can be found in the “Fashion Trend Circle” and “My Publish” |
| Publish outfit to fashion trend circle from “My Outfit” | 2.8 | Click the publish button on the up-right of the screen and select “publish from my outfit”, select tags and click “finish” | A toast pops out saying “publish succeed”, and the outfit in “My Outfit” can be found in the “Fashion Trend Circle” and “My Publish” |
| Publish outfit to fashion trend circle by camera | 2.9 | Click the publish button on the up-right of the screen and select “publish from camera”, select tags and click “finish” | A toast pops out saying “publish succeed”, and the outfit in “My Outfit” can be found in the “Fashion Trend Circle” and “My Publish” |
| All outfits viewed | 2.10 | Scroll down to the bottom of the list | Prompts a toast saying “all outfits in the fashion trend circle has been viewed!” |

### 4.2.3 “My Closet”

|  |  |  |  |
| --- | --- | --- | --- |
| Test-case Description | Test-case No. | Action | Expected Output |
| Jump to “My Closet” | 3.1 | Click “My Closet”, the second tab of the main activity | “My Closet” interface is displayed and the default mode is view by list |
| Change viewing mode | 3.2 | Click the “Gallery Mode” in the drawer | The layout changed to the gallery mode that displays all the clothes one by one |
| View clothes by type | 3.3 | Select which type of clothes the user wants to view in the drawer | The selected type of clothes are displayed |
| Add clothes from album | 3.4 | Click the “add” button on the up-right of the screen and select “add clothes from album”, select tags and click “finish” | The newly added item has been shown on the “My Closet” panel |
| Add clothes from camera | 3.5 | Click the “add” button on the up-right of the screen and select “add clothes from camera”, select tags and click “finish” | The newly added item has been shown on the “My Closet” panel |
| View clothes’ detail | 3.6 | In the “List Mode”, click on the picture of the clothes | The interface jump to the clothes detail page and all the details of the selected clothes are displayed |

### 4.2.4 “My Space”

|  |  |  |  |
| --- | --- | --- | --- |
| Test-case Description | Test-case No. | Action | Expected Output |
| Jump to “My Space” | 4.1 | Click “My Closet”, the third tab of the main activity | “My Space” interface is displayed and the default mode is view by list |
| View all “My Publishes” | 4.2 | Click “My Publishes” | The interface show all the outfits that the user once published |
| View all “My Collection” | 4.3 | Click “My Collection” | The interface show all the outfits that the user added to the collection |
| View all “My Outfits” | 4.4 | Click “My Outfits” | The interface show all the outfits that the user once created |
| Delete “My Publishes” | 4.5 | The user get into the detail page of “my publishes ” and click “delete” | A toast pops out informing “publish deleted” and the item is removed from “Fashion Trend Circle” |
| Delete “My Collection” | 4.6 | The user get into the detail page of “my collections ” and click “delete” | A toast pops out informing “collection removed” and the item is removed from “My collection” |
| Delete “My Outfits” | 4.7 | The user get into the detail page of “my outfits ” and click “delete” | A toast pops out informing “collection removed” and the item is removed from “My outfits” |
| Publish “my outfits” to “Fashion Trend Circle” | 4.8 | The user get into the detail page of “my outfits ” and click “publish” | A toast pops out informing “publishing succeed” and the item is added to “My Publishes” can be seen in “Fashion Trend Circle”. |

## 4.5 Functional Testing Result

Most of the actual test outputs are consistent with our expected outputs. However, the following ones appears to be quite different from the expected results and are determined to be system defects.

|  |  |  |  |
| --- | --- | --- | --- |
| Use-case Description | No. | Expected Output | Actual Output |
| Empty in search box | 2.5 | Search result will be all the news | Search result is 0 entry |
| Click popular tag in headline pages | 5.2.1，5.3.1，5.4.1，5.5.1，5.6.1…. | Jump to tag-matched news page | The results are all the news of the label corresponding |
| Right username and password | 1.1 | Successful landing | Login failed (500 reported wrong) |
| Click download directly | 5.14.1 | Out of the window to save | Download failed (error 404 reported) |
| Username has not logged in and publsh comment | 4.5 | Prompts the user is not logged | Your comment has been |
| Register and input | 6.1 | Successful registration | Always displayed "page loading" |

5. Regression Testing

## 5.1 Regression Testing Goal

Regression testing is a type of software testing that seeks to uncover new software bugs, or regressions, in existing functional and non-functional areas of a system after changes such as enhancements, patches or configuration changes, have been made to them.

The intent of regression testing is to ensure that changes such as those mentioned above have not introduced new faults. One of the main reasons for regression testing is to determine whether a change in one part of the software affects other parts of the software.

## 5.2 Regression Testing Method

* Rerunning previously completed tests and checking
* Whether program behavior has changed
* Whether previously fixed faults have re-emerged
* Testing Tools: Junit
* Testing Tool Introduction:

JUnit has all of the features needed to make regression testing fully automated and thus help produce high-quality products. JUnit performs regression testing by re-running automated functional tests of web applications as developers add new functionality to the application. Regression tests can be composed of any successful keyword, scripted or low-level functional tests that have earlier verified the desired application functionality and behavior.

* The Advantage of Using Junit for Automatically Recursive Testing
* Easily repetitive test those that run for multiple builds
* Convenient to test those that tend to cause human error
* Good for those tests that require multiple data sets
* Used for those frequently-used functionality that introduces high risk conditions
* Help test those are impossible to perform manually
* Make tests that run on several different hardware or software platforms and configurations
* Automatically tests those take a lot of effort and time in manual testing

## 5.3 Regression Testing Procedures

* Choosing test-case to automate regression testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Tested Methods** | **Test Case No.** | **Parameters** | **Expected Return** | **Actual Return** | **Test Result** |
| **attemptLogin()** | 1.1 | username=null  password=null | Login Fail | Login Fail | Pass |
| 1.2 | username=”admin”  password=null | Login Fail | Login Succeed | Fail |
| 1.3 | username=”admin”  password=”22222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222” | Login Succeed | Login Succeed | Pass |
| 1.4 | username=“admin”  password= “•＆λσ△Δ@” | Login Succeed | Login Fail | Fail |
| 1.5 | username=“admin”  password=  “admin/\"'” | Login Succeed | Login Succeed | Pass |

* Using JUnit to do recursive testing

## 

## 5.4 Regression Testing Result

In summary, the result of the regression testing shows that the program behavior has not changed and remains the normal and as what is designed in the requirement analysis specification documents during the developing time. As more modules integrated and new functions added in, there is little (but still happened) defect evolved, and fortunately fixed after they are found.

6. Integration Testing

## Note: since most of the integration works are done prior to writing this document and are unfortunately not recorded therefore can't be displayed here. However, we want to maintain the integrity of the whole testing process and this document as well, and this is the reason we still keep this part. All the info below are purely simulation and hypothesis.

## 6.1 Integration Testing Goal

The idea is to test combinations of pieces and eventually expand the process to test your modules with those of other groups. Eventually all the modules making up a process are tested together. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

* Testing whether interface between modules are matched
* Testing whether code is in consistence with design
* Testing whether the user-interface are using the same standard

## 6.2 Integration Testing Method

* Method: Modified Sandwich Integration
* Advantage of using this method
* Using statics of both Top-down integration and Bottom-up integration
* Make sure every module for our website is tested independently and thoroughly

## 6.3 Integration Testing Result

Every modules in website has been thoroughly tested and incrementally integrate into our system. Bugs we find that not-matched interface has been fixed by our project developers. Also what need to be noticed is that, some modules work successfully independently, while integrated not works well. These happens due to following reasons:

* Inconsistent interface between modules such as data types
* Resource occupy and locked problem

7. Interface Testing

This testing follows a transaction through all of the product processes that interact with it and tests the product in its entirety. Interface testing shall be performed to ensure that the product actually works in the way a typical user would interact with it.

## 7.1 Interface Testing Goal

Interface testing is the process of ensuring proper functionality of the graphical user interface (GUI ) for our website and making sure it conforms to its written specifications.

In addition to functionality, GUI testing evaluates design elements such as layout, colors, fonts, font sizes, labels, text boxes, text formatting, captions, buttons, lists, icons, links and content. GUI testing processes can be either manual or automatic, and are often performed by third -party companies, rather than developers or end users.

## 7.2 Interface Testing Procedure

### 7.2.1 Interface Demo

“GuiMi” is a fairly complicated application with dozens of pages, which has given rise to a great challenge of user interface design and maintaining. Below are some of the key features’ user interface, based on which we are able to conduct the interface testing.

### Macintosh HD:Users:Elena:Desktop:未命名文件夹:Screenshot_2014-06-19-00-31-09.pngMacintosh HD:Users:Elena:Desktop:未命名文件夹:Screenshot_2014-06-19-00-32-08.png Macintosh HD:Users:Elena:Desktop:未命名文件夹:Screenshot_2014-06-19-00-32-18.png

### 7.2.2 Measurements and Assessment

We measure our project in such 7 aspects:

1. Whether consistent with the regulations

The application should conform to the standard android application form. For example, in each page, we designed specific menu buttons on the tool bar and make sure all the layouts are consistent with the android development regulations.

1. Whether direct and salient

As can be seen from the above 6 pictures, all keywords (such as the number of outfis) and key operations (such as “finish” or “submit”) are marked with different colors so as to make sure they can be seen in the first sight and never loose attention.

1. Coherence

We check such following aspects in every page:

* Fonts
* Elements style of layout
* Other website-user practice

As showed below:

We double checked all pages in the application on the above three criteria and clearly, the are coherent in fonts and layout style. When user gets used to one of them, he can easily guess another page’s function without actually using it.

1. Flexibility

Users can flexibly choosing their own way to complete one function. We think it’s the disadvantage part in our website. We should add more fore flexible part for our user.

1. Whether comfortable to use

As whether website is comfortable to use is a very subjective thing, however, we put as much people as possible into our alpha (A) testing, and collect their opinion.

1. Accuracy

Compare with the requirement specification, it’s no doubt that all the modules and designed function for our website is accurately implemented. Also, during testing we found some spelling mistakes and typos in the website, and these small errors are fixed.

1. Practicability

During interface testing, we find bellowing two modules are little redundant as they both show documents downloading functions. But we think they cannot be merged as it’s more direct for user to down things that are not documents, or those they only want to download documents.

## 7.3 Interface Testing Result

Small typos and spelling mistakes has be found in testing and fixed. In summary, as we show the whole procedure in interface testing, we think in 6 aspects we do great well. However, the interface lacks flexibility in some sense. User should be given rights to choose the parts they want to read, or user should be given the choice to customize more personal features, especially in “My Closet”.

This is the part we shall pay more attention during next iteration.

9. Performance testing

In software engineering, performance testing is in general testing performed to determine how a system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

Here we only focus on Stress Testing.

## 9.1 Performance Metrics

In this testing document, we select the following 4 main metrics to measure the app’s performance:

* Memory (percentage)
* CPU
* Network flow
* Battery

## 9.2 Performance Testing Tools

* Emmagee

Emmagee is a practical, handy performance test tool for specified Android App, which can monitor CPU, memory, network traffic, battery current and status([Some devices are not supported](https://github.com/NetEase/Emmagee/wiki/Some-devices-are-not-supported)). Additionally, it also provides several cool features such as customizing interval of collecting data, rendering real-time process status in a floating window, and much more.

* Homepage: <https://github.com/NetEase/Emmagee>
* Wiki: <https://github.com/NetEase/Emmagee/wiki>
* Issues: <https://github.com/NetEase/Emmagee/issues>
* Tags: Android, Java
* Basic Steps

A VERY brief summary of using Emmagee is:

First of all ,you should have Emmagee.apk,download from github or build the apk file youself then :

* Start Emmagee App
* Configure interval
* Select a target process
* Click Start button

## 9.3 Performance Testing

### 9.3.1 Result

By using the automatic testing tool Emmagee, we are able to measure all the performance-related metrics. The tables below aptly shows what happens inside the android phone while running our app:

Table. Testing Environment Parameters

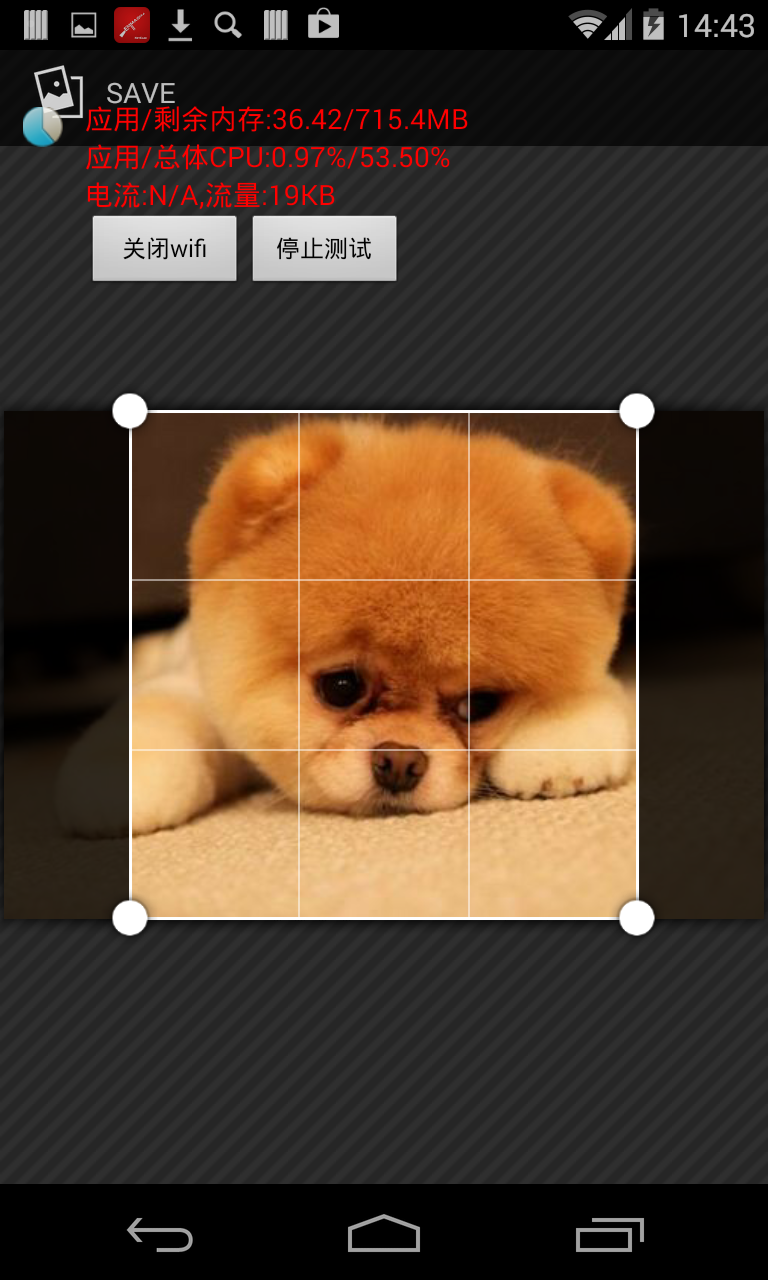
|  |  |
| --- | --- |
| Application Package Name | com.guimi |
| Application Name | GuiMi |
| Application PID: | 3244 |
| Phone Memory(MB)： | 1834.75MB |
| CPU： | ARMv7 Processor rev 2 (v7l) |
| Android system version： | 4.4.3 |
| UID | 10209 |
| Phone version： | Nexus 4 |

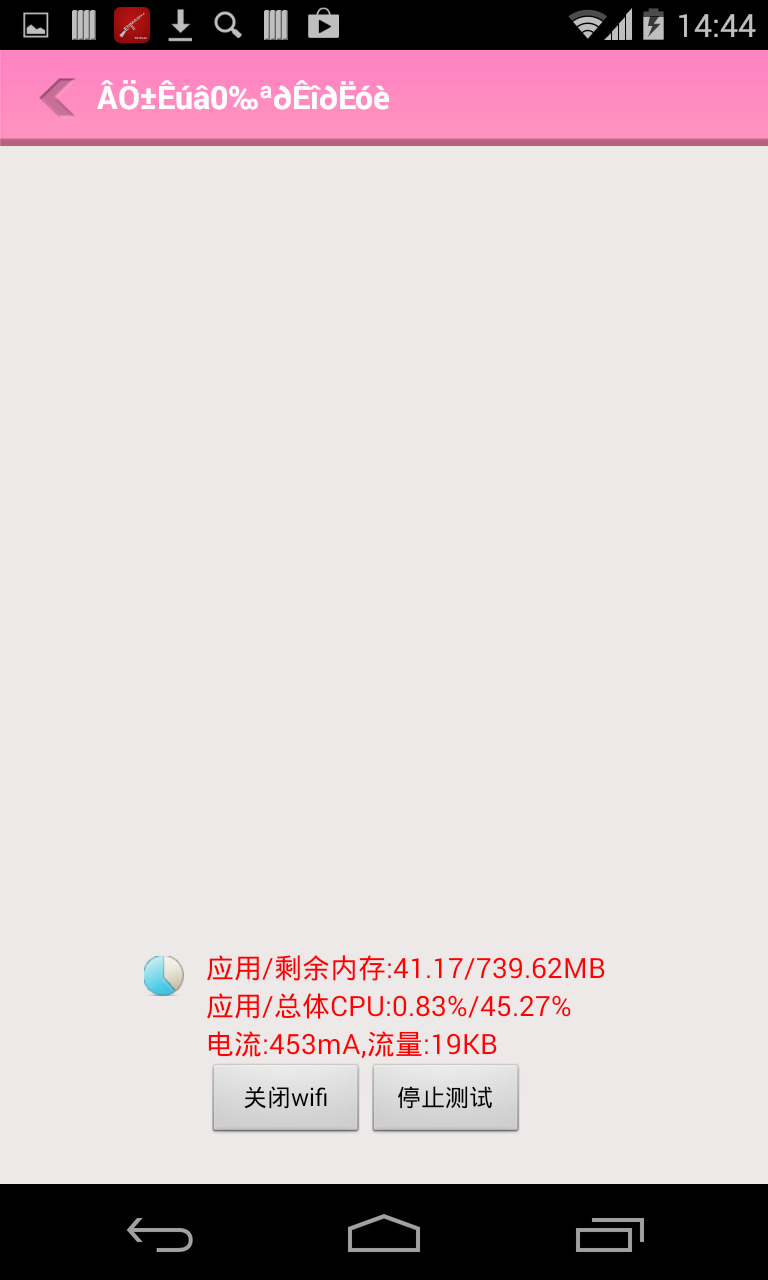
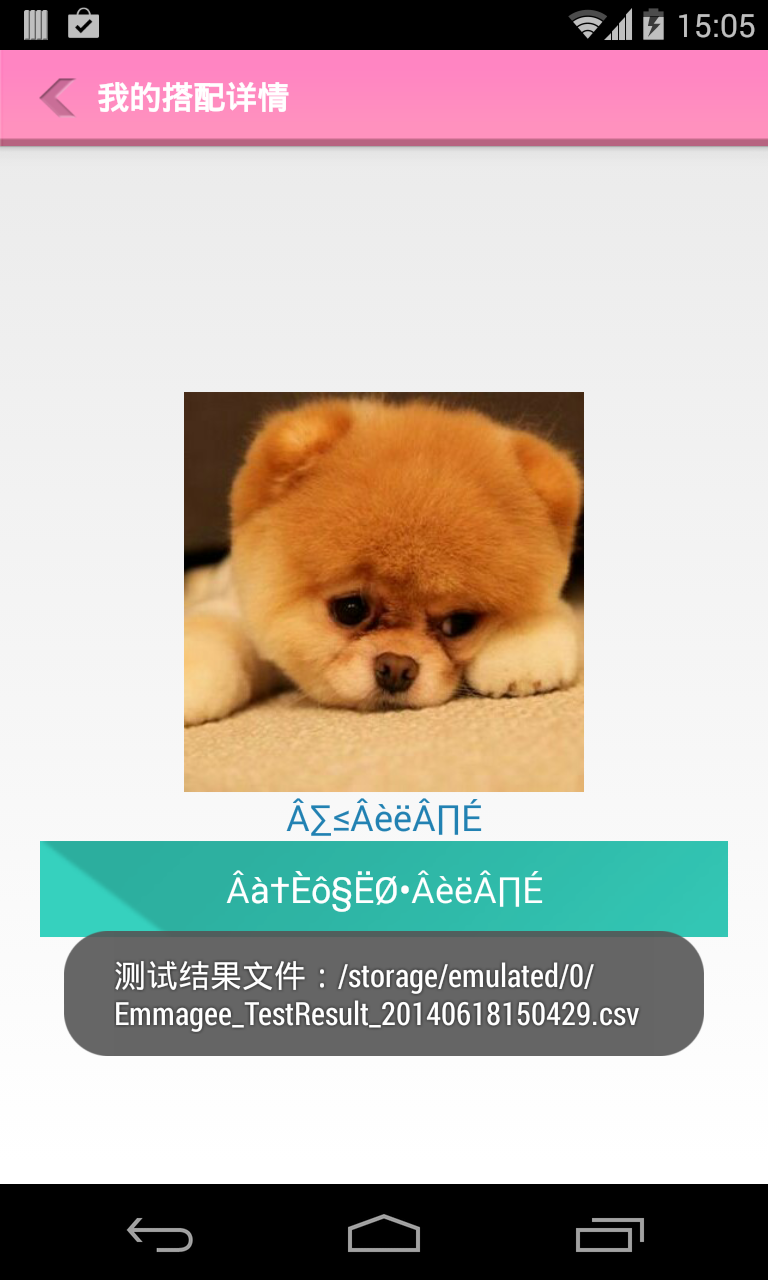
Table. Performance Metrics 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time | Memory PSS(MB) | Memory Percentage(%) | Memory Left(MB) | App CPU Usage (%) | CPU Usage Total(%) |
| 14:41:22 | 44.85 | 2.44 | 745.61 | 0 | 20.16 |
| 14:41:27 | 47.45 | 2.59 | 741.34 | 0 | 0.02 |
| 14:42:04 | 47.45 | 2.59 | 747.8 | 0 | 40.99 |
| 14:42:25 | 46.84 | 2.55 | 730.04 | 0.97 | 48.47 |
| 14:43:19 | 36.42 | 1.99 | 715.4 | 0.97 | 53.5 |
| 14:43:45 | 40.57 | 2.21 | 710.41 | 0.39 | 49.31 |
| 14:44:21 | 41.17 | 2.24 | 739.5 | 0.83 | 45.27 |

Table. Performance Metrics 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time | Network Flow(KB) | Battery(%) | Current(mA) | Temperature(C) | Voltage(V) |
| 14:41:22 | 4 | 80.0% | 499mA | 31.6 | 4 |
| 14:41:27 | 5 | 80.0% | 449mA | 31.6 | 4.026 |
| 14:42:04 | 5 | 80.0% | 482mA | 30.8 | 4.043 |
| 14:42:25 | 9 | 80.0% | 471mA | 30.9 | 4.037 |
| 14:43:19 | 19 | 79.00% | N/A | 31.5 | 4.084 |
| 14:43:45 | 19 | 79.00% | N/A | 31.7 | 4.028 |
| 14:44:21 | 19 | 79.00% | 453mA | 32 | 4.017 |

### 9.3.2 Analysis

From results displayed above we are are happy to say that our app performs pretty well, with CPU, memory, network flow under tight control and haven’t caused much pressure to the phone, thus are fully applicable I real practice！

10. User acceptance testing

User acceptance testing activities will be performed by the business users. The purpose of this testing will be to ensure the application meets the users’ expectations.

User acceptance testing (UAT) consists of a process of verifying that a solution works for the user. It is not system testing (ensuring software does not crash and meets documented requirements), but rather is there to ensure that the solution will work for the user i.e. test the user accepts the solution (software vendors often refer to as Beta testing).

This testing should be undertaken by a subject-matter expert (SME), preferably the owner or client of the solution under test, and provide a summary of the findings for confirmation to proceed after trial or review. In software development, UAT as one of the final stages of a project often occurs before a client or customer accepts the new system. Users of the system perform tests in line with what would occur in real life scenarios.

The UAT acts as a final verification of the required business functionality and proper functioning of the system, emulating real-world usage conditions on behalf of the paying client or a specific large customer. If the software works as required and without issues during normal use, one can reasonably extrapolate the same level of stability in production.

11. Go/No-go Meeting

Once the test team has completed the test cycle, a Go/ No-go meeting is scheduled as part of the implementation planning under launch readiness. This meeting is attended by the project manager, business team, test lead, technical lead, and any other stakeholders.

The test lead will provide a testing summary and list all outstanding unresolved defects and any associated risks with releasing the product to production. All outstanding issues are discussed at that time before a decision is made to push to production. A written sign-off form is signed by all team members as listed above. The list of outstanding issues is also attached to the sign-off form.

12. Test Environment

## 12.1 Software Requirements

* Eclipse+sql server+jdk
* SourceMoniter
* JUnit
* Emmagee

## 12.2 Testing Platform

* Desktop PC – the application supports Windows XP and Windows 2000 with the following browsers:
  + Internet Explorer 6.0 and higher
  + Netscape 7.0 and higher
  + Firefox 1.5 and higher
  + Mac OS 10.3.9 and higher
  + Apple Safari 1.3.1 and higher
* Additional tests will be performed on the Virtual Private Network and through the Internet via a low bandwidth.

13. Assumptions and Risks

## 13.1 Assumptions

* The Business team has reviewed and accepted functionality identified in the business requirements and software requirements documents.
* Project change control process in place to manage requirements.
* Code walkthroughs/reviews will be completed by the development team.
* Unit testing will be completed by the development team prior to release to the test team.
* Testers will test what is documented in the requirements.
* The test team will have a separate test environment to perform testing.
* All changes to requirements will be communicated to the test team.
* Resources identified in this plan are available to test the application and resolve defects and address issues as they are raised by the test team.
* That the delivery of the product to production contains all setup, etc., that is necessary for optimum performance in the production site.
* Project sponsors, business and technical, will provide actionable guidance on defect prioritization and resolution.

## 13.2 Risks

* Scope creep (last minute addition of new requirements) impacts deadlines for development team and test team.
* Aggressive target date increases the risk of defects being migrated to production. If development timelines are not met, this will directly impact the testing timelines.
* Key resources have completing priorities making availability less than scheduled.
* Any downtime of the test system will significantly impact the testing cycle.
* Load testing is not being completed on a consistent basis; true performance of the application may not be known until release to production.

Additional Project Documents

Please see all the related materials (included but are not restricted to: .java documents for unit, regression, and function testing, report for safety testing, etc.) in the attached folder.

# Sign-off and Acknowledgement

I understand that by agreeing to participate in this testing through the execution of the testing plan, I approve of the activities defined and authorize my department to participate as documented for the successful implementation of this application in our department.

\_\_ Sheng Wen \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: 15/June/ 2014

Team Leader

\_\_ Zhao Qing \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: 15/June/ 2014

Team Member

\_\_Ren Jiayue \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: 15/June/ 2014

Team Member

\_\_Hu Mian \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: 15/June/ 2014

Team Member

Test Director – Defect Tracking Process

|  |  |
| --- | --- |
| **Summary:** | Screen name and short description about the defect being reported, usually providing key words with which to identify and/or search for the defect. |
| **Detected By:** | Auto populates with the User ID of person logged in. |
| **Detected on Date:** | Auto populates with current date. |
| **Severity:** | Describes the degree of impact that a defect has on the operation of the application. |
| **Assigned To:** | Individual being assigned the defect for fixing. |
| **Detected in Build:** | Build ID in which the defect was found. Build ID is an identifier for the code release, assigned by Web Development. |
| **Fixed in Build:** | Build ID in which the defect is fixed. Build ID is an identifier for the code release, assigned by Web Development. |
| **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. |
| **Status:** | Indicates the existing state of a defect, auto populates with a default of “New” |
| **Description:** | Enter description of defect  Add individual steps to reproduce. Include all steps and screens that were accessed.  Enter exact words of the error message. |
| **Email Defect:** | After entering defect, right-click on it and select email to send to assigned developer. |
|  | |
| **Defect resolution process:** | When the developer begins working on the defect, s/he changes status 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”, and will change the “Assigned To” field to the tester or defect manager. 3. The tester will retest the submitted defect. 4. If defect passes the retest, the tester or defect manager will change Status to “Verified”. 5. If the defect is not fixed, the tester will change the Status to “Reopen” and enter the UserID of the developer in the Assigned To field. 6. Once the defect has been “Verified”, the project manager (or defect manager) will update the status to “Closed”. |
|  | |
| **DEFINITIONS FOR DEFECT PRIORITY AND SEVERITY** | |
| **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. This field is utilized by the developers and test engineers to prioritize work effort on the defect resolution. | |
| **1 – Urgent Blocks Work** | Further development and/or testing cannot occur until the defect has been resolved. |
| **2 – Resolve ASAP** | The defect must be resolved as soon as possible because it is impairing development and/or testing activities. |
| **3 – Normal Queue** | The defect should be resolved in the normal prioritization and completion of defect resolution. |
| **4 – Low Priority** | The defect is an annoyance and should be resolved, but it can wait until after more serious defects have been fixed. |
| **5 – Trivial** | The defect has little or no impact to development and/or testing work. |
|  | |
| **SEVERITY:** This field describes the degree of impact that a defect has on the operation of the application. | |
| **1 – Critical** | Critical loss of function. The defect results in system crashes, the failure of a key subsystem or module, a corruption or loss of data, or a severe memory leak. |
| **2 – Major** | Major loss of function. The defect results in a failure of the system, subsystem, or module, but the defect does not result in the corruption or loss of significant data. |
| **3 – Moderate** | Moderate loss of function. The defect does not result in a failure of the system, subsystem, or module, but the defect may cause the system to display data incorrectly, incompletely, or inconsistently. |
| **4 – Minor** | Minor loss of function, or another problem where a workaround is present. There are no data integrity issues. |
| **5 – Usability** | The defect is related to the system usability, is the result of non-conformance to a standard, or is related to the aesthetics of the system. There is no loss of system function. |
| **6 – Enhancement** | The defect is a request for an enhancement, i.e. it is not within the scope of the current project effort. |