MIDDLE EAST TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

CENG492 TEST SPECIFICATION REPORT

KORSAN YAZILIM

Table of Contents

1. Introduction

Test plan and test procedure of our project will be described in this document.

1.1. Goals and Objectives

We will be testing our project for finding errors and bugs most importantly. And by taking lessons these errors, we will try to improve our product. After doing the project as error-free as possible, we will try to solve issues such as: performance, logical correctness, user friendliness and reliability.

1.2. Statement of Scope

What we will try to check during software testing is generally: User should not wait so long for Yaver to do its job; Yaver should not get any wrong information about users' environment and their assignments. Its' interface tools should be easily accessible and usable. Yaver should adapt user interface as we expected in a user friendly way, we will try work out wrong arrangements of user interface. It should guarantee that assignments are coming from command center correctly. All the features of Yaver will be checked for ensuring functionality. We will try to fix all of these by testing.

1.3. Major Constraints

1.3.1. Time

Time is our first major constraint, because there are few weeks for finalizing this project; however all of our four group members have very busy schedules due to other time-consuming courses of our department and their responsibilities such as homework and exams, part-time internships and the other responsibilities. Besides these, we are still implementing our project, so while implementing and improving project, testing it for errors at the same time is very difficult. Whenever we add functionality to application, it requires some fitting in among other functionalities of product.

1.3.2. Data

Due to that we get our map information from Google and it sends over-detailed information to our application, its' size is not very optimum, so it does slow down our application little bit.

1.3.3. Hardware

Although we have started this project by thinking we will develop an application for a mobile device, it seems that we could not. So, we did implement our project on computer, by assumptions on how it will work on a mobile device. And we also should be able to try it on different machines with different system properties to check whether it works or not. Because it may work beautifully in a high property machine while it may not in a low property machine.

1.4. Definitions, Acronyms and Abbreviations

- Java SDK(jdk): Java Software Development Kit
- Debugger: Tools that are used to find bugs.

1.5. References

1. http://netbeans.org/projects/debugger
2. http://www.ej-technologies.com/products/jprofiler/overview.html
3. http://www.pgadmin.org/

2. Test Plan

2.1. Software to be tested

We have three parts of project which are: simulation for command center which we called as YaverListener, the user-application part and application-camera part of the software. We will be testing these parts separately and thoroughly. And we will also test their interactions as a whole. We will also operate on parts of main project which are Messages, Assignments and Filtering sections.

2.2. Testing Strategy

In order to present a robust, bug-fixed product we will make many tests on our application. As a computer science genius Edsger W. DIJKSTRA states “Program testing can be used to show the presence of bugs, but never to show their absence! Keeping this in mind, in order to have a program as free of bugs as possible, it is a must to have a good testing plan. We divided testing procedure into 4 parts.

i. Unit Testing

ii. Integration Testing

iii. Validation Testing

iv. High-Order Testing

We made unit testing while creating modules. After implementing a class, we tested this already implemented unit. If we find any bug then, related developers focused on fixing that bug. We assume that a module takes correct input from another module; that is we do not try to fix inter-module bugs in unit testing. Inter-module bugs will be fixed during integration test. In validation testing, we will check whether our product runs on different platforms or not. During unit testing and integration testing, we are working on Microsoft Windows 7 operating system with already installed Java SDK (jdk) and other test tools described at 2.4.

2.2.1. Unit testing

We choose white box testing mechanism as strategy since it considers the internal details of a module and tries to test every unit inside. We will implement test mechanism to cover all possible situations that will be tried. Each module will be tested separately. At unit testing, we will test modules under excessive (more than expected) stress and sure to be still alive for each module. Moreover, we will test Yaver under invalid inputs to protect system against errors occurred by this kind of inputs.

Also, we found that for decreasing the errors of our codes and making test easier for us, we have to put into our consideration these characteristics of codes that we will test.

- Simplicity
- Stability
- Understandability
- Controllability
- Operability
- Decomposability

2.2.2. Integration testing

After we had tested the modules individually, we are going to test all the units of the project by integrating each other and test for any collision between them. Moreover, we will test all the expected features of the system user interface are prepared for easy use, the module functionalities work properly. Finally, we will look for if all the modules are integrated successfully and the functionalities are interacting properly.

2.2.3. Validation testing

Validation test asks for whether the product is working in the right way or not. Validation testing is the testing procedure that is enabling compatibility between implementation and the other previous reports such as Requirement and Design analysis reports. After integrating all modules, initially we will test our final product is completely consistent to the reports. Then we will test whole system under different environment conditions with lots kind of inputs.

2.2.4. High-order testing

We will do performance testing which means we will check our project with different workloads such as fast camera movements and detection of them; fast user movements and getting location of user in that speed, handling many messages and assignments to see limits of its performance. We will do stress testing to ensure that no crash happens by maximizing interactions between different sections of the product. And we will do security testing for preventing any data loss during getting messages and assignments.

2.3. Test Metrics

We will keep track of number of errors we encounter, number of solutions to problems we get, number of different tests we executed and number of solutions to problems which are image processing related.

2.4. Testing tools and environment

The tool list we will use is below:

NetBeans Debugger: The debugger plug-in of NetBeans includes most of its user interfaces, and ability to delegate to a specific debugger implementation at the user's request. We will use to detect errors of parts before integrating them.

JProfiler: This profiler is an award-winning all-in-one java profiler. JProfiler's intuitive GUI helps you find performance bottlenecks, pin down memory leaks and resolve threading issues. We will use to test performance of our system.

PgAdmin: This tool is designed to answer the needs of all users, from writing simple SQL queries to develoing complex databases. This tool provides managing the database operations. We have been controlling and testing database operations using PgAdmin.

Abbot Java GUI Test Framework: This framework provides automated event generation and validation of Java GUI components, improving upon the very rudimentary functions provided by the java.awt.Robot class. The framework may be involved directly from java code or accessed without programming through the use of scripts. We will use it for GUI parts testing.

We will also test the system different environments. First environment type is operating system. We will test the system on Linux, Windows, 32 bit or 64 bit operating systems.

Second environment type is hardware. We will test the system on computers which have different system properties. Since our program needs to process images taken by camera, there should be at least one camera on the computer which our program runs on.

3. Test Procedure

This section describes as detailed test procedure including test tactics and test cases for the software.

3.1 Unit test cases

We will test below classes and modules sequentially. While testing, if any of them fails related developer will try to fix the failed module or class. For server module, we will test both network connection and database. For user module, each class will be tested to verify whether it is done or not.

- ✓ Server Module
  - Communication Class
- ✓ User Module
  - Toolbar Class
  - Assignment Class
  - Message Class
  - Filtering Class
- ✓ Server Module (Again)
  - Redesign Class

3.2 Integration testing

After testing all units separately, we will integrate all of them. Initially, we will integrate user module's class and test them to verify whether it is done or not. Then we will integrate communication class to previously obtained composite unit. Finally, we will integrate Redesign unit to all previous unit and test again.

3.3 Validation testing

We will validate and check for all requirements are fulfilled or not in the order of integration testing. If necessity arises, we can have additional requirements for parts of project which are going to be tested.

3.4 High-order testing

We will test final product in terms of performance and stress. Performance Testing aims to achieve reasonable response times to the user when s/he is running the Yaver. Those tests

are conducted within many different PCs with different configurations to observe the varying waiting times. Waiting times of modules and functions are calculated precisely with JProfiler. Functions or modules that consumes too many CPU time are controlled and optimized by developers. Stress testing aims to tolerate excessive load and doesn't reflect to the user when s/he is running the Yaver. In Unit testing, we tested module units against excessive loads. Although all units don't fail separately, may complete system crashes. In order to prevent this type of crashes, after integration and validation, whole system is tested again with too many interactions.

4. Testing Resources and Staffing

Testing is mostly done by the developers and each developer tests his part. Then, each part is tested by someone other than its developer. If there is no error than the corresponding part or module can be integrated to the whole system. Eventually, system tests are done by all development team members. Staff responsibilities for testing different modules and classes are seen below:

Ali Karakaya: Server Module - Communication class unit testing

Yunus Emre Işıklar: Server Module - Redesigning class unit testing

Ali Oğuzhan Önkal: User Module – Toolbar class, Assignment class unit testing

Aydın Atay: User Module – Message class, Filtering class unit testing

Our plan is also to test the system with the end-users. We will tell general features of the system to the users and allow them to use the system. User manuals which will be prepared can be used for this purpose.

5. Test Work Products

The result of the testing process is the identification of the bugs. Whenever one of the team members finds a bug, he will assign the correction to the corresponding developer using TRAC. This provides keeping records of the bugs.

6. Test Record Keeping and Test Log

We will use NetBeans debugger for debugging purposes. It is a plug-in for the NetBeans IDE. We will also archive versions of the software after each test.

NetBeans debugger includes necessary user interfaces. It provides record keeping and logging for users. For details, please see section 2.4.

7. Organization and Responsibilities

Organization of Korsan Yazılım's test group is shown below:

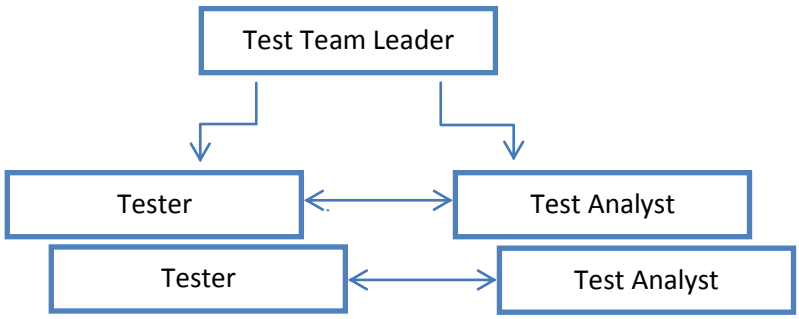

Figure 1: Organization of the Korsan Yazılım Testing Team

Test Team Leader:

Test team leader has several responsibilities such as assigning tasks to Testers and Test Analysts, monitoring their progress according to the testing plan and schedule.

Test Analyst:

Test Analyst is responsible for the design and implementation of testing cases. During the design of the test cases, Test Analyst needs to analyze software requirements specifications to highlight specific requirements that must be tested.

Tester:

Tester's most important responsibility is executing the test cases designed by the Test Analyst for the interpretation and documentation of the results of these test cases. Before

executing the test cases, Tester needs to setup and initialize the test environment, including the test data and test hardware. During the execution of cases, Tester is responsible for keeping forms which reflect the observed results from the test execution.

Test Team Member	Responsibilities
Ali Karakaya	Test Team Leader, Test Analyst
Yunus Emre Işıklar	Test Analyst
Aydın Atay	Tester
Ali Oğuzhan Önkal	Tester

Table 1: Korsan Yazılım Test Team Members and Responsibilities

8. Test Schedule

Testing schedule, that we are planning to stick on it, is below:

Task	Start Date	End Date
Test Specification Report		25.04.2011
Feature and Integration Tests		15.06.2011
Performance and Stress Tests	01.06.2011	08.06.2011
Alpha and Beta Tests	05.06.2011	12.06.2011
Bug Tracking and Fixes		15.06.2011

Table 2: Korsan Yazılım Test Schedule