Testing document x.x

SQUID

Helsinki 5th April 2005

Software Engineering Project

UNIVERSITY OF HELSINKI
Department of Computer Science

Course

581260 Software Engineering Project (6 cr)

Project Group

Mikko Jormalainen
Samuli Kaipiainen
Aki Korpua
Esko Luontola
Aki Sysmäläinen

Client

Lauri J. Pesonen
Fabio Donadini
Tomas Kohout

Project Masters

Juha Taina
Jenni Valorinta

Homepage

`http://www.cs.helsinki.fi/group/squid/`

Change Log

Version	Date	Modifications
0.1	31.3.2005	First version (Aki Korpua)
0.2	5.4.2005	Corrections, Use Case sample (Aki Korpua,Mikko Jormalainen)

Contents

1 Introduction

This document describes how this software (Ikayaki) is planned to be tested properly. Mainly this document concentrates on describing methods used for testing and test cases. It is important that all members of team to make tests in same way. This lowers possibility of testing conflicts and helps on integration test phase.

2 Overview of testing the system

Because program will be used to control a magnetometer, testing will be more important than in normal software engineering student projects. We will do unit testing for each class, integrate testing to program and use separate squid emulator to test squid interface system.

In unit testing each class is tested independently. Unit testing will be done by using JUnit. Every programmer will test his own classes. Class should be tested when it is finished and corrected before integration test begins.

Integration testing tests interfaces between classes. It will be done by going through all user interface protos and checking that all sections in requirements document can be done. Some critical sequences which are done many times with program should be done too.

Squid interface integration testing is done simulating real system with emulator. It will be done using Squid-emulator before testing it with real magnetometer. Squid-emulator runs in different machine and is connected by few (2-3) Serial I/O cables. Squid-emulator will be tested with old program (2G) same way before testing Ikayaki-system so that it will have all same tested properties which old program have and both systems have same results with squid emulator.

To verify that old program and new program works same way, we will do critical measurement with old program and emulator, save emulators log file and then use emulator with that log file and do same critical measurement with new program and see that both have same results.

If Rita testing utility is easy enough to use it will be used in testing. Tests will be constructed in such way that every line of code is visited at least once.

3 Testing

3.1 Unit Testing

Unit Testing is done for each class separately. Every class has it's own ClassName-Test.java class in test-directory. They should be done before and during coding class. All test cases must be executable after classes are ready. Every class should be tested succesfully before integration tests.

JUnit is simple java-based framework for testing your java classes. We use it in this project for Unit Testing. For more information visit http://junit.sourceforge.net/.

First you need to download JUnit from http://sourceforge.net/project/showfiles.php?group _id=15278 and extract it to directory (different than java directory) and set classpath for it.

Then you must write test class for every class to be tested. Test classes extend TestCase. They will have test methods, one for each Test Case. Test class also need suite()-method and Main for run. Sample test class:

```
import java.util.*;
import junit.framework.*;

public class SimpleTest extends TestCase {

//Simple Test Case

public void testEmptySimple() {
      Simple simple = new Simple();
      //New Simple must be empty
      assertTrue(simple.isEmpty());

}

public static Test suite() {
      return new TestSuite(SimpleTest.class);
}

public static void main(String args[]) {
      junit.textui.TestRunner.run(suite());
}
}
```

Type "java junit.swingui.TestRunner SimpleTest" to run test cases for Simple.

All test cases are listed in section 3.4.

3.2 Integrity Testing

Integrity testing is done after unit test is passed for all classes. Testing is done using squid emulator at first and finally with real Squid system.

Graphical User Interface is tested using all use cases from Requirements Document. If one fails, it is corrected immediatelly and all use cases must be done again. When all use cases are done without errors this phase is ready.

Interface testing is done with Emulator. Use cases are Automatic Measurement, Thellier Measurement and Manual Measurement with all variations (see Requirements Document). In first phase we use old program with emulator, save its log file and take results. After that we run it with new program and compare results. If they are not same, corrections are made immediatelly to new program and test is runned again. This is done until results are same for all use cases.

3.3 Squid Emulator

Squid emulator is tested with old program. Use cases are Automatic Measurement, Thellier Measurement and Manual Measurement with all variations (see Requirements Document). When all use cases can be done with emulator it is ready enough for integrity testing.

This should be done so that Old Program has same results with SQUID-system and emulator for all use cases. But we don't have resources for this. So this test only tells us that Old and New program works same way with squid emulator.

3.4 Test Cases

Here are listed JUnit test cases for each class.

3.4.1 ConfigurationPanel

1. Data values match Settings after creation

   True:

   - All field values match Settings values
   - At least 2 COM ports are found

   False:

   - a field is empty

2. Wrong values are not accepted

   True:

   - All values are typed and right type.

   False:

   - Wrong values are saved