Testing document x.x

SQUID

Helsinki 12th 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)
0.3	12.4.2005	Use case split and now look (Aki Korpua)

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. Classes tested with JUnit have it's own ClassNameTest.java class in test-directory and gui-components are tested manually. 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.

All use cases are listed in section 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 first and after that Squid Interface is tested.

Graphical User Interface is tested using all use cases from 4.3. 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.

Squid Interface testing is done with Emulator. Use cases are Automatic Measurement, Thellier Measurement and Manual Measurement with all variations (see 4.3). 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.

4 Test cases

4.1 JUnit test cases

Here are listed JUnit test cases for <insert class names>. JUnit is introduced in section 5.

4.1.1 SampleClass

1. Test1

   Conditions:

   - Cond1
   - Cond2

2. Test2

   Conditions:

   - Cond1

4.2 GUI-component test cases

Here are listed test cases for gui-classes, these are done manually just clicking and changing values.

4.2.1 SettingsPanel

1. Magnetometer,Handler and Degausser COM-port combobox

   Conditions:

   - Shows all COM-ports on system in alphabetical order
   - Loads correct COM port from Settings
   - Doesn't allow same value as in Handler and Magnetometer/Degausser COM-port comboboxes

2. Save button

   Conditions:

   - Only available if changes are made and values are permissible.
   - Unavailable on start
   - On click closes window and saved data correctly

3. Cancel button

   Conditions:

   - On click closes window and doesnt save changes
   - Always available

4. Calibration constants

   - Loads data correctly from Settings
   - Accepts positive and negative decimal numbers

5. Degausser ramp

   - Loads data correctly from Settings
   - Has only elements: 3,5,7,9

6. Degausser delay

- Loads data correctly from Settings
- Has only elements: 1..9

7. Acceleration and deceleration fields
   - Loads data correctly from Settings
   - Accepts only values in range of Integers 0..127
8. Velocity and velocity in measurement fields
   - Loads data correctly from Settings
   - Accepts only values in range of Integers 50..20000
9. Translation positions fields
   - Loads data correctly from Settings
   - Accepts only values in range of Integers 0..16777215
10. Right limit
    - Loads data correctly from Settings
    - Has only elements: plus, minus
11. Rotation field
    - Loads data correctly from Settings
    - Accepts positive and negative Integer numbers

4.2.2 className

1. Use case1
   - Cond1
   - Cond2

4.3 Intergrity test case

Here is test case which is done in integration phase to all components at same time. This is done in following order and must be repeated until done succesfully

4.3.1 GUI test case

1. Open software

   Conditions:
   - Last project is opened, if first open then no project.
   - All components are in right place

4.3.2 Automatic Measurement

4.3.3 Thellier Measurement

4.3.4 Manual Measurement

5 JUnit

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.