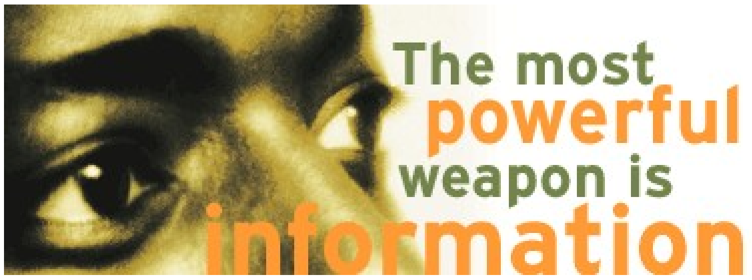
Martus Testing Project
Final Deliverable



November 21, 2013

Project Name: Martus: Global Human Rights Abuse Reporting System.
Call and Contact: charlestonmartustesting@gmail.com
martus website: http://martus.org
Project Duration: 9/3/2013 – 11/21/2013 (2 months and 18 days)
Project Roles:
- Planner: Jason Wilson
- Liaison: Steven Pilkenton
- Researcher: Bobby Jenkins
- Organizer: Tomoko Goddard

Global Human Rights Abuse Reporting System

Table of Contents

1. Introduction

The Martus project was designed by Benetech, whose mission is to bring the immense power of technology to bear on social problems such as human rights, literacy, education and disability access. Human rights and social justice groups throughout the world gather and large amounts of data, yet these organizations often lack the resources to document human rights violations systematically and securely. Much of their information is stored in insecure formats that prevent it from being effectively shared. However, critical documentation is often lost to viruses, computer theft, fire, neglect and staff turnover. Lost information is a problem with far-reaching implications. Social justice organizations know that timely, accurate data distribution is one of their most powerful weapons against human rights violations. When it comes to security, the best way to guarantee data does not end up in the wrong hands is to hold it; literally. With Martus, the number one priority is security. Confidentiality is important in when it comes to handling peoples' private information and personal experiences. Matus, as a whole, provides secure and reliable data storage for information related to human rights, literacy, education and disability access.

1.1 Purpose

The purpose of the deliverable is to document the selected tests we've chosen to implement in our framework. The tests are based on functional and non-functional requirements of the already working open-source software. We simply use preexisting code to ensure parts of it work together accordingly and can handle unexpected situations in a non-fatal way. It is important to know the original project does include a testing suite, however, we have created our own and tested each component/method fully. Test cases are based on a specific state Martus could be in at runtime and dealt with accordingly to avoid software failure.

1.2 Scope

To reach our final goal, we must develop a script (called from the terminal) that calls a series of sub directories to run 25 tests on the existing open-source project. Each contributer to the project must have the framework in the proper format and call the script (bash file) from the command line in order to see each test case's results. The script should call text files (.txt files) to get arguments for each compiling and executing command. These text files contain the location of each executable, its input and its classpath. Once the executable has run, the script should compare each executable's output with the expected output stored in the test's oracle file (.txt files here as well) and add the result to a table of test results displayed via web browser.

1.3 Acronyms

Martus – The title of the project being tested
HFOSS – Humanitarian Free Open Source Software

1.4 Maintenance

This document contains a revision history log. When changes are made, the log will reflect the events that have occurred and reasons for revision.

1.5 High Level Deliverable Description

The methods used in this project will be based from already tested methods and their test cases along with untested test cases that we have chosen to create test cases for. Our task is not to re-engineer the already functioning program, but rather to test functional, and non-functional, requirements of the software. Despite the already working testing framework built-in to the existing project, our mission is to create our own, smaller version.

1.6 Environment

- Ubuntu Desktop 12.04
- Java 1.6
- Eclipse Indigo (3.7.2)
- MercurialEclipse plugin

1.7 Project Building experience

Built using modified version of documentation found at:
https://code.google.com/p/martus/wiki/DeveloperHOWTO

The Mercurial Plugin must be installed in Eclipse in order to function properly.

The instructions were as follows:

1. In Eclipse, create a new workspace.
2. In the Package Explorer view, do a right-click/Import/Mercurial/Clone Existing Mercurial Repository.
3. For the URL, enter:
https://code.google.com/p/martus.martus-amplifier/
4. Hit OK
5. Repeat steps 2-4 for each of the following URLs:
    - https://code.google.com/p/martus.martus-bc-jce/
    - *Appears this package may be skipped when running OpenJDK (causes test errors) – investigate further
    - https://code.google.com/p/martus.martus-client/
    - https://code.google.com/p/martus.martus-clientside/
    - https://code.google.com/p/martus.martus-common/
    - https://code.google.com/p/martus.martus-docs/
    - https://code.google.com/p/martus.martus-hrdag/
    - https://code.google.com/p/martus.martus-jar-verifier/
    - https://code.google.com/p/martus.martus-csv2xml/
    - https://code.google.com/p/martus.martus-logi/
    - https://code.google.com/p/martus.martus-meta/
    - https://code.google.com/p/martus.martus-mspa/
    - https://code.google.com/p/martus.martus-server/
    - https://code.google.com/p/martus.martus-swing/
    - https://code.google.com/p/martus.martus-thirdparty/
    - https://code.google.com/p/martus.martus-utils/
6. Right-click on martus-csv2xml, and to a Refactor/Rename, and change the name to
martus-js-xml-
generator

* Our team chose to omit a step telling the user to push the *bc-jce.jar* above the *jsse.jar* in the jar list due to the *bc-jce.jar* causing errors and having no noticeable impact on the running program or its tests.

Running preexisting tests:
1. Hit Ctrl-Shift-T and enter "TestAllQuick".
2. When it comes up, hit Run/Run As... Junit Test. All tests should pass.

1.8 Experiences Summary

After building the project and running the tests, at this point the team is able to run the completed version of the program and tests. The team has the ability to alter code in Eclipse as desired in an effort to get an idea as to what components and/or functions would be good candidates for testing. However, the we do not intend to contribute to the open-source project from this project.

2. Test Planning

Martus is considered a finished product, and has been released to the public for some years. Based on the status of the project, our test plan will follow a requirements-based release testing format. Our goal this semester is to, at a minimum, thoroughly test what members consider the most fundamental requirements of the system. To ease members into the testing process, we will run Martus as a client to get a better understanding as to what the most important components of the software would be.

2.1 Requirements Traceability

Req ID = requirement ID
Type = type of requirement for the user
testCases = input used by the script
Exe = executable file
input = file used for input and classpaths
gray words = optional

Req ID	Type	Description	Trace from User Requirement to System Requirement	Method Format	Test Cases	Trace Framework
1	Security	A user's saved input must be encrypted properly.	A users saved input into fields must be unreadable by lurking systems and only by other components of the Martus software.	Encrypt(InputStream plainStream, OutputStream cipherStream, Sessionkey sessionkey, String publicKeyString)	• “” • “A String to encrypt” • *The entire Lorem Ipsum doc*	**Exe**: TestEncryptDecrypt.java **Input**: testCase1.txt testCase2.txt testCase3.txt
2	Login Credentials	A user's password must of length 8, different than the username, and username must not be blank. A password must contain at a minimum of 2 non-alphanumeric characters to be considered and be of	To prevent unauthorized users from accessing others' account	• validateUserNameAndPassword(String username, char[] password) • containsEnoughNonAlphanumericCharacters(char[] password)	• **Username**= Password **Password** = Password • **Username** = Username **Password** = 1234567	**Exe**: TestUsernamePassword.java TestStrongPassword.java **Input**: testCase4.txt testCase5.txt testCase6.txt testCase7.txt

Req ID	Type	Description	Trace from User Requirement to System Requirement	Method Format	Test Cases	Trace Framework
		length 15 to be considered Strong.			• **Username** = “” **Password**= Password • **Password**= “Snow{>#12345678” • **Username** = Username **Password** = Password • **Password** = notStrong&^ • **Password** = longButNotStrong& • **Password** = longAndStrong&^	testCase20.txt testCase21.txt testCase22.txt
3	Changing Bulletin's State	A database key is created for each user ID and can be set to “sealed” (the account is stored) or “draft” (the account is in the creation process).	A unique key is given to each User account so Martus knows where to get that user's information. A key is set “sealed” when that is the permanent/final location for a user's information and set “draft” when a User's ID is being altered or is in an unsaved state.	• isSealed() • isDraft() • setDraft() • setSealed()	• Is this **Key** set to Draft? • Is this **Key** set to Sealed? • Set this **Key** to sealed • Set this **Key** to draft	**Exe**: TestDatabaseKeyDrafted.java TestDatabaseKeySealed.java TestDatabaseKeyStatuses.java **Input**: testCase8.txt testCase9.txt testCase10.txt
4	Storing and Deleting	Martus must be able to store and delete user data with no trace of it existing after deltion.	User's information must be saved somewhere when created and deleted from that location when deleted.	writeRecord() discardRecord()	• 5 newly created records with random data thrown into a newly created database • 0 created entries and an attempt to store a String (word) there.	**Exe**: TestDatabaseRecordCreation.java TestDatabaseRecordDelete.java TestDatabaseRecordCreateDelete.java **Input**: testCase11.txt

Req ID	Type	Description	Trace from User Requirement to System Requirement	Method Format	Test Cases	Trace Framework
						testCase12.txt testCase13.txt
5	Privacy/Security	Martus must transfer data only through secure ports. It connects in a greedy manner chosing the first of the scured ports and must find a secured port wherever it may be in the list of available ports. It must not connect at all if all secured ports fail or non are available and throw an exception if all ports fail.	A User must work in a secure environment and not send any information via unsecured connection. This keeps information between the user and Martus only.	• callServer(String serverName, String method, Vector params) • callServer(String serverName, Caller caller)	• **good-port-middle**=7 • **good-port-first**=7 • **fail-all**=true; **good-port-first**=7	**Exe**: testSSLPortSelect.java **Input**: testCase14.txt testCase15.txt testCase16.txt
6	Data Accessing	The software must create a Universal ID from an account ID and its prefix,m account ID and local ID, or a string representation of a combination of account ID and local ID. The universal ID will then be used to map data to its user.	A user is given a universal ID which tells Martus which data belongs to that user. This way Martus can uniquely identifiy users.	• createLocalIdFromByteArray(String prefix, byte[] originalBytes) • createFromAccountAndLocalId(String accountId, String localId) • createFromString(String uidAsString)	• **Account ID** = someAccountID **Local ID** = someLocalID **Prefix** = D • *this same instance above is used to test all 3 methods that create a Universal ID*	**Exe**: TestUniversalID.java **Input**: testCase17.txt testCase18.txt testCase19.txt
7	Non-Functional	There exists a datatype "FieldSpec" which is given a tag and a label. The datatype must pass its parameters to the datatype "MartusField" which is a more global type.	*Is not a user requirement and is only a system requirement.*	• common.fieldspec.MessageField.getFieldSpec()	• "tag" • "label" • compare **these are strings the system sends into FieldSpec when it requests for a MartusField's tag or label.*	**Exe**: TestFieldBasics.java **Input**: testCase23.txt testcase24.txt testCase25.txt

2.2 Tested Items

Listed below are the items to be tested displayed in such a way the user can understand more clearly. We gave each tested function a risk-level based on how crucial each test's success is for the system's functionality and overall purpose.

Function	Items used for Testing	Risk
Encrypting and Decrypting User's Data	• Empty word • Standard string • *Lorm ipsum document*	High
Checking for an acceptable password	• The word "Username" • A short password • *the empty string*	Low
Checking for a strong password	• A short word with 2 non-letter/numbers • a long password with 1 non-letter/number • a long password with 2 non-letter/numbers	Low
Checking for unacceptable username	• Blank username • the word "Username"	High
Setting Bulletin's status to "sealed" and "draft"	• Setting it to "draft" • Setting it to "sealed" • Comparing the two	High
Creating and deleting Martus data	• 5 automatically generated bulletin entries • 0 generated bulletins	High
Only connecting through a secure/trusted port	• List of all secure ports • list of all unsecure ports • list of one secure port	High
Create a user ID	• Account ID, local ID, prefix together	Medium
Inheritance of Data Types	• The word "tag" • The word "label" • The word "compare"	Low

2.3 Testing Schedule

The testing schedule is displayed below indicating start and end dates of specific phases of project production. Phases must be completed by the indicated ending date in order to meet the final deadline.

Phase Number	Starting Date	Ending Date	Duration	Phase	Comments
1	9/3/2013	9/17/2013	14 days	Choosing	Team will chose a

Phase Number	Starting Date	Ending Date	Duration	Phase	Comments
				Project	project from the HFOSS database which must be approved by the professor
2	9/17/2013	9/24/2013	7 days	Building Project	Each team member must complete their local build of the Martus project and be ready to create and run their own tests
3	9/24/2013	10/8/2013	15 days	Identifying 5 tests to be part of final product	Team must have 5 test cases ready to be implemented
4	10/8/2013	10/29/2013	21 days	Implementing these 5 tests	Team must have 5 working test cases in their properly functioning framework
5	10/29/2013	11/14/2013	17 days	Implementing 25 test cases	Team must have 25 working test cases
6	11/14/2013	11/21/2013	7 days	Final framework, documentation, and poster completed	Team must present fully working testing framework, poster (digitally), and final deliverable

***Project completed and presented on November 21,2013**

2.4 The Testing Process's Results

The team completed the project as required with a few speed bumps along the way. Overall, the final deadline was met and project completed.

Phase Number	Met Deadline?	Criteria Met?	Outcome
1	Yes	Yes	Martus Project was successfully approved
2	Yes	Yes	Each Team Member successfully built the project
3	Yes	No	The team came up with 5 test cases, however, did not present the progress appropriately.
4	Yes	Yes	The team implemented more than enough test

Phase Number	Met Deadline?	Criteria Met?	Outcome
			cases and presented the material accordingly.
5	Yes	Yes	The team presented the project with all 25 test cases.
6	Yes	Yes	The team has the poster, powerpoint, and full working framework functioning properly and uploaded on the subversion repository.

2.5 Hardware and Software Requirements

Hardware

- 32-bit or 64-bit system

Software

- Ubuntu 12.04 or similar Linux Distribution
- Java 1.6 or higher
- Eclipse Indigo 3.7.2 or higher
- Mercurial Plugin for Eclipse

2.6 Constraints

Issues holding us back were no longer applicable towards the end of production. Members had issues early on due to meetings in noisy areas and not meeting frequent enough. This was quickly resolved by meeting twice a week at a minimum in reserved library rooms. The team members had never work with each other before and had to adapt to others' schedules. This was resolved by holding meetings at the same time on Mondays, Wednesdays, and/or Fridays.

3 The Testing Process

3.1 Purpose

This chapter of the deliverable is meant to give the stock holders an inside look at how the software is progressing. This will describe the structure of the framework, the how-to document, test cases, and major experiences.

3.2 Experiences

Number	Summary	Date	Affected
1	Martus project files uploaded into repository	10/10/2013	All
2	Testing script uploaded	10/14/2013	Scripts folder
3	First 3 test cases added	10/16/2013	Folders affected

			include Test Case text files, Oracles, and testCase Executalbes
4	Script moved to top level directory	10/16/2013	runAllTests.sh, all testCase text files
5	6 test cases added	10/22/2014	Folders affected include Test Case text files, Oracles, and testCase Executalbes
6	11 test cases added	11/10/2013	Final test cases added and submitted to the repository.

We had met once a week up until our presentation of deliverable 2. Every since then, we've been meeting every Monday and Friday for a few hours being sure all of us are up to date and understand the changes that have been made to the repository. There was a bit of confusion as to how the framework was to interact with the project files and what files were to be generated by tests. Once we arrived a reasonable agreement, the project and script were uploaded and everyone was uploading test cases remotely. Our remaining meetings were then about updates made in the cloud.

3.3 Test Cases to be implemented

Test Case	Requirement	Method Tested
#1	Application must be able to encrypt data	common.crypto.MartusSecurity.encrypt()
#2	Application must be able to decrypt encrypted data	common.crypto.MartusSecurity.decrypt()
#3	Username must not match password	client.core.MartusUserNameAndPassword.validateUsernameAndPassword()
#4	Password must be at least 8 characters	client.core.MartusUserNameAndPassword.validateUsernameAndPassword()
#5	Username must not be blank	client.core.MartusUserNameAndPassword.validateUsernameAndPassword()

#6	Testing if setting a database key to draft works after its been initialized to "sealed" by default	common.database.DatabaseKey.isDraft()
#7	Testing if database keys are set to "sealed" by default	common.database.DatabaseKey.isSealed()
#8	Testing for difference in keys set to sealed vs keys set to drafted	common.database.DatabaseKey.isSealed()
#9	Application must be able to store user data	common.database.FileDatabase.createRecord()
#10	Application must be able to delete stored user data	common.database.FileDatabase.discardRecord()
#11	Application will try allowed secure ports until a connection is established	clientside.ClientSideNetworkHandlerUsingXmlRpc.callServer()
#12	Application will try only enough secure ports to establish a connection	clientside.ClientSideNetworkHandlerUsingXmlRpc.callServer()
#13	Application will try all available ports to establish a connection	clientside.ClientSideNetworkHandlerUsingXmlRpc.callServer()
#14	Application must create a universal ID from account and local IDs	common.packet.UniversalId.createFromAccountAndLocalId()
#15	Application must create a universal ID from account ID and a prefix	common.packet.UniversalId.createFromAccountAndPrefix()
#16	Application must create a universal ID from the string representation of account and local IDs	common.packet.UniversalId.createFromString()
#17	Application must create a universal ID from account and local IDs	common.packet.UniversalId.createFromAccountAndLocalId()

#18	Application must create a universal ID from account ID and a prefix	common.packet.UniversalId.createFromAccountAndPrefix()
#19	Application must create a universal ID from the string representation of account and local IDs	common.packet.UniversalId.createFromString()
#20	Passwords must be at least 15 characters with at least 2 non alphanumeric characters to be considered "strong"	client.core.MartusUserNameAndPassword.isWeakPassword()
#21	Passwords must be at least 15 characters with at least 2 non alphanumeric characters to be considered "strong"	client.core.MartusUserNameAndPassword.isWeakPassword()
#22	Passwords must be at least 15 characters with at least 2 non alphanumeric characters to be considered "strong"	client.core.MartusUserNameAndPassword.isWeakPassword()
#23	once a variable of type FieldSpec is given attributes, MartusField must inherit them (.getTag() field checked here)	common.fieldspec.MessageField.getFieldSpec()
#24	once a variable of type FieldSpec is given attributes, MartusField must inherit them (.getLabel() checked here)	common.fieldspec.MessageField.getFieldSpec()

#25	once a variable of type FieldSpec is given attributes, MartusField must inherit them (FieldSpec and MartusField are compared to each other here)	common.fieldspec.MessageField.getFieldSpec()

3.4 Framework Descriptions

The framework is an extensible testing framework made for Java and run on Unix systems. The framework makes use of test case files to define parameters to run Java test classes. Test case files are named in the format: 'testCaseX.txt' where X is the test number. The framework operates by looping through all valid test case files located in /top_level_directory/testCases, running the Java test classes defined by each test case, creating the oracle and recording the results. Evaluation of test results are made by comparing actual output of the test class to the expected output - any mismatch results in test failure. Results of each test run are generated as testReport_mm-dd-yyyy_hh:mm:ss.html and are stored in /top_level_directory/reports. Each report gives detailed testing parameters for each test case, and, in the event of failure, gives reasons for test failures. The framework may be extended by defining your own test cases that reference your Java test cases. The framework should execute all tests given any input.

3.4.1 Framework Directory Structure

```
/team4
        runAllTests.sh
        /project
                /martus-amplifier
                /martus-client
                /martus-cleintside
                /martus-common
                /martus-hrdag
                /martus-jar-verifier
                /martus-js-xml-generator
                /martus-logi
                /martus-meta
                /martus-mspa
                /martus-server
                /martus-swing
                /martus-thirdparty
                /martus-utils
        /scripts
                runTest.bash
        /testCases
                testCase1.txt
                testCase2.txt
                testCase3.txt
                testCase4.txt
                testCase5.txt
                testCase6.txt
                testCase7.txt
                testCase8.txt
                testCase9.txt
                testCase10.txt
                testCase11.txt
                testCase12.txt
                testCase13.txt
                testCase14.txt
                testCase15.txt
                testCase16.txt
        /testCasesExecutables
                testDatabaseKeyDrafted.java
                testDatabaseKeySealed.java
                testDatabaseKeyStatuses.java
                testDatabaseRecordCreation.java
                testDatabaseRecordDelete.java
                testDecrytp.java
                testEncrypt.java
                testEncryptDecrypt.java
                testSSLPortSelect.java
                testUniversalID.java
                testUsernamePassword.java
        /temp
        /oracles
        /docs
                README.txt
                team4_deliverable1.pdf
                team4_deliverable2.pdf
        /reports
                /img
                        fail.png
                        pass.png
                report.css
```

3.5 "How-To" Be A Martus Tester

3.5.1 Overview

The testing framework is packaged with .java test files for the open source project Martus (www.Martus.org), but can easily be adapted to test any Java project. The framework uses text files to define test cases to run, and creates a detailed report for each test run.

3.5.2 Requirements

The framework should run in any Unix environment and the Java version compatibility is entirely dependent on the methods used in your classes. However, testing of the framework with the pre-packaged class files has been limited to Ubuntu 12.04 and Java 1.6 or any other newer combinations of the two.

3.5.3 How-to use

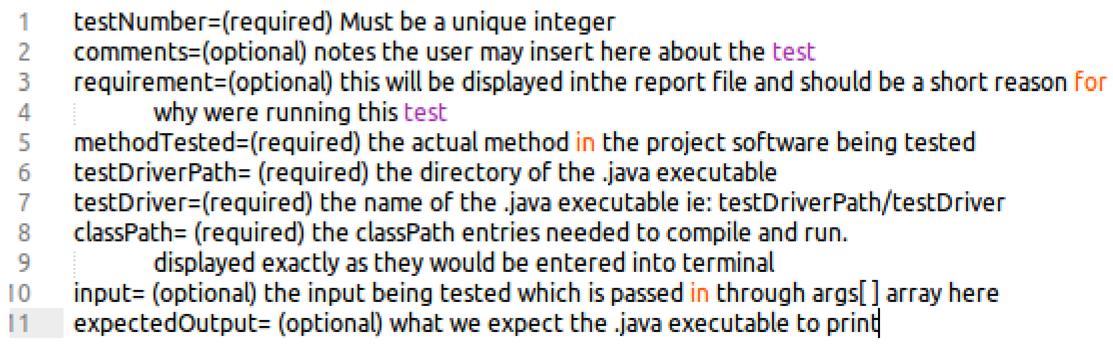
Using terminal, navigate to the top level folder (/team4 by default) and type 'sh runAllTests.sh'. This will execute all available test cases, and, upon completion, will open the report file created in '/team4/reports'.

3.5.4 How-to Clean Build Your Project

If your project under test is an eclipse project, you may use the included build script (./scripts/buildProject.bash) to clean build your project during each test run. Change ./scripts/projectName.txt to contain the name of your project (Any name is valid - this is purely cosmetic) Change ./scripts/build.txt to contain your project package names in the necessary build order seperated by spaces, and finally change lines 3 and 4 of buildProject.bash to contain the classpath entry of any external jars and the folder name of your build output ("bin") folder.

3.5.4 Creating your own Test Cases

The framework is made extensible through the use of test case plain text files. Once you have written a test in Java it is a simple matter of creating a test case with the necessary parameters to add the new test to the suite. Test case files must be named using the convention 'testCaseX.txt' where X is any number. The following parameters are contained in a test case file:



```
testNumber=(required) Must be a unique integer
comments=(optional) notes the user may insert here about the test
requirement=(optional) this will be displayed inthe report file and should be a short reason for
        why were running this test
methodTested=(required) the actual method in the project software being tested
testDriverPath= (required) the directory of the .java executable
testDriver=(required) the name of the .java executable ie: testDriverPath/testDriver
classPath= (required) the classPath entries needed to compile and run.
        displayed exactly as they would be entered into terminal
input= (optional) the input being tested which is passed in through args[] array here
expectedOutput= (optional) what we expect the .java executable to print
```

testNumber=(required) will be incremental and continuous with no gaps ie: 1,2,3 and nor 1,3,4.
requirements(optional) will be displayed in the final report file and should tell the user what we're testing and why.
classPath= (required) The classpath entries needed to run. Entered as above.
input= (optional) The input, if any, used by your Java test class. Input is passed to the class exactly as typed and passed in Java via string array. Input will accept the relative path to a .txt file as input, and retrieve literal input from the file as well. Any spaces in the input parameter will be interpreted by your Java main class as delimiting the string array elements. String building must be handled in your Java class.
expectedOutput= (required) The expected output of your Java class. Determines test pass or fail status.

4. Testing Experience

The Martus project gave the us first hand experience working on a large project. The benefit here was that members were required to code trace others' code and use Java types and classes they were not used to using before Martus.

4.1 Group Experiences

We met twice a week in an effort to finalize the deliverable and implement test cases properly. Meetings were basically progress reports on whats been done by each team member and used for some code consolidating. Meetings started with a quick summary as to what will be completed during that meeting as well as debate time for all members to speak out about ideas and gather one way to implement each test case. Test cases were discussed individually as were sections of this document.

4.2 Test Cases

Test Case	Requirement	Method Tested	Input Used	Output Expected
#1	Application must be able to encrypt data	common.crypto.MartusSecurity.encrypt() common.crypto.MartusSecurity.decrypt()	“”	Derypted String = “”
#2	Application must be able to encrypt data	common.crypto.MartusSecurity.encrypt() common.crypto.MartusSecurity.decrypt()	A string to encrypt	String encrypted = true
#3	Application must be able to decrypt encrypted data	common.crypto.MartusSecurity.encrypt() common.crypto.MartusSecurity.decrypt()	*Lorem Ipsum entire document*	*Lorem Ipsum entire document*
#4	Username must not match password	client.core.MartusUserNameAndPassword.validateUsernameAndPassword()	--username=Password --password=Password	org.martus.common.Exceptions$PasswordMatchedUserNameException
#5	Password must be at least 8 characters	client.core.MartusUserNameAndPassword.validateUsernameAndPassword()	--username=Username --password=1234567	org.martus.common.Exceptions$PasswordTooShortException
#6	Username must not be blank	client.core.MartusUserNameAndPassword.validateUsernameAndPassword()	--username= --password=Password	org.martus.common.Exceptions$BlankUserNameException
#7	Valid usernames and passwords must be deemed as such	client.core.MartusUserNameAndPassword.validateUsernameAndPassword()	--username=Username --password=Password	Username and password are valid.
#8	Testing if setting a database key to draft works after its been initialized to "sealed" by default	common.database.DatabaseKey.isDraft()	true	true
#9	Testing if database keys are set to "sealed" by default	common.database.DatabaseKey.isSealed()	true	true
#10	Testing for difference in keys set to sealed vs keys set to drafted	common.database.DatabaseKey.isSealed()	true	false
#11	Application must be able to store user data	common.database.FileDatabase.createRecord()	--entries-to-create=5 –to-store = A String	Database record count = 0
#12	Application must be able to delete stored user data	common.database.FileDatabase.discardRecord()	--entries-to-create=5 --to-store=	Database record count = 0
#13	Application must be able to store and delete user data	common.database.FileDatabase.createRecord() common.database.FileDatabase.discardRecord()	--entries-to-create=0 --to-store=A string	Database record count = 0
#14	Application will try allowed secure ports until a connection is established	clientside.ClientSideNetworkHandlerUsingXmlRpc.callServer()	--good-port-middle=7	Number of tried ports = 3

#15	Application will try only enough secure ports to establish a connection	clientside.ClientSideNetworkHandlerUsingXmlRpc.callServer()	--good-port-first=7	Number of tried ports = 1
#16	Application will try all available ports to establish a connection	clientside.ClientSideNetworkHandlerUsingXmlRpc.callServer()	--fail-all=true --good-port-first=7	Number of tried ports = 5
#17	Application must create a universal ID from account and local IDs	common.packet.UniversalId.createFromAccountAndLocalId()	--account-id=someAccountID --local-id=someLocalID --prefix= --from-string=false	Account ID = someAccountID & Local ID = someLocalID
#18	Application must create a universal ID from account ID and a prefix	common.packet.UniversalId.createFromAccountAndPrefix()	--account-id=someAccountID --local-id=someLocalID --prefix=D- --from-string=false	Account ID = someAccountID & Local ID length = 26
#19	Application must create a universal ID from the string representation of account and local IDs	common.packet.UniversalId.createFromString()	--account-id=someAccountID --local-id=someLocalID --prefix= --from-string=true	Account ID = someAccountID & Local ID = someLocalID
#20	Passwords must be at least 15 characters with at least 2 non alphanumeric characters to be considered "strong"	client.core.MartusUserNameAndPassword.isWeakPassword()	--password=notStrong&^	Weak password
#21	Passwords must be at least 15 characters with at least 2 non alphanumeric characters to be considered "strong"	client.core.MartusUserNameAndPassword.isWeakPassword()	--password=longButNotStrong&	Weak password
#22	Passwords must be at least 15 characters with at least 2 non alphanumeric characters to be considered "strong"	client.core.MartusUserNameAndPassword.isWeakPassword()	--password=longAndStrong&^	Strong password
#23	once a variable of type FieldSpec is given attributes, MartusField must inherit them (.getTag() field checked here)	common.fieldspec.MessageField.getFieldSpec()	tag	true
#24	once a variable of type FieldSpec is given attributes, MartusField must inherit them (.getLabel() checked here)	common.fieldspec.MessageField.getFieldSpec()	label	true
#25	once a variable of type FieldSpec is given attributes, MartusField must inherit them (FieldSpec and MartusField are compared to each other here)	common.fieldspec.MessageField.getFieldSpec()	compare	true

*this is an exact copy of the report that is displayed via html file once runAllTests.sh has completed. Please resort the higher-level version of this in chapter 3 if you have difficulties understanding this version.

4.3 Overall Testing experience

The outcome of our tests were as we expected while the road along the way to our goal took longer than expected. The test report shown above indicates we have met the 25 test case requirement and displayed all necessary details alongside each test case respectively.

5 Fault Injection Process

We were asked to inject faults in 5 of our test cases to demonstrate the test suite's smooth error handling. Naturally, the best way to handle errors in Java is by implementing try and catch blocks in your code. One could also use finally blocks if desired. The Faults we've injected have been scattered throughout our project to ensure different executables are being stressed.

5.1 Faults

Fault Num	Method	Method(s) Effected	Location	Description of Change	Expected Effects
1	client.core.MartusUserNameAndPassword.containsEnoughNonAlphanumbericCharacters()	client.core.MartusUserNameAndPassword.isWeakPassword()	Line 81	changed to return false instead of true when there are enough non alphanumeric characters	all passwords will be considered weak
2	clientside.ClientSideNetworkHandlerUsingXmlRpc.callServer()	clientside.ClientSideNetworkHandlerUsingXmlRpc.callServer()	Line 267	changed "(portIndexToTryNext+1) % numPorts)" to "portIndexToTryNext+numPorts"	index out of bounds exception except when good port is tried first
3	common.crypto.MartusSecurity.createCipherOutputStream()	• common.crypto.MartusSecurity.encrypt() • common.crypto.MartusSecurity.decrypt()	Line 373	commented out the line: "output.writeInt(encryptedKeyBytes.length);"	attempts to decrypt data will result in decryption exception
4	common.database.FileDatabase.discardRecord()	common.database.FileDatabase.discardRecord()	Line 280	changed "if(file.exists())" to "if(!file.exists())"	IOException when deleting database records
5	common.packet.UniversalId.UniversalId()	• common.packet.UniversalId.createFromAccountAndLocalId() • common.packet.UniversalId.createDummyFromString() • common.packet.UniversalId.createFromString()	Line 64	changed "setAccountId(accountIdToUse);" to "setAccountId(localIdToUse);"	universalIDs will be created with invalid AccountIDs

5.2 Fault Summary

The faults listed above were simply changes of code and/or states on input. As any finalized software should be, our test suite handles errors to the point where it will never trip on errors and halt execution. We had changed data types to cause fault five, and created an conditional branch from executing when a condition is true, to executing when that condition is false to cause fault four. We had removed the output entirely to cause fault 3 and changed the output to cause fault one. And finally, we caused an out of bounds error in to create fault two.

*our current version in the subversion repository executes with these faults visible and in their respected locations.

6 Experiences

We experienced few troubles along the way throughout the development of this project. The main issues we had were schedule collisions. Many of us have scattered schedules and could not meet at the same time and place for very long. For instance, we schedules meetings for 2pm Mondays, Wednesdays and Fridays. One member had a class at 1 O' clock, another had a class at 3 O' clock and the other two were readily available during that meeting time of 2-4. We found that being in a study room in the library was optimal as well. We attempted using the computer science lab early on for meetings, however, that resulted in much distraction and noise. Overall, we tackled our issues early on so there was nothing serious holding back our progress.

6.1 Performance Problems

Our project seemed to build and execute just fine on each of our systems. The only issue here being that a few members had issues adapting to the learning curve of using subversion from the command line interface. Our output presented in class showed a GLIB error in the terminal once the test case suite was finished executing. This was looked into and found as a firefox error and in no was a side-effect from our test suite. In the final stages of the project, more specifically points beyond deliverable 3, our project had no performance issues and only experienced fine tuning of the script and files required for testing.

6.2 Compilation Issues

We had originally used an already compiled version of the Martus program then ran our test suite. Upon request by the professor, we incorporated actually building the entire project as step one of the test suite before actually performing the testing. This was our only compilation issue.

6.3 Testing and Debugging

Once the universal script was created, we simply created test cases that ran using the Martus project's classes and objects that cooperated with our script. The problems that arose here were mostly Java specific. However, with the College of Charleston's extensive Java incorporation in its courses, we were all able to debug our code just fine.

6.4 METRICS

The Martus project contains about 30 packages and about 300 classes total. Of those 300 classes, about 150 were raw Java types. The code was completely undocumented and made use of many Java types we were unfamiliar with before this project.

7 Project Assignment Evaluation

Our group saw the deliverable deadlines as a good pace for the project. The confusion our group had was figuring out exactly what a testing suite was and how we were to incorporate it using existing code. Few of us had ever called classpaths via the terminal which required much self research. Learning scripting was also a bit difficult considering we had never been required to use it extensively in any previous classes. We believe two classes at a minimum of Q and A about how to begin the project would have been convenient. Many students, especially in this field, pride themselves on not asking questions, but we believe having a deliverable or class discussion requiring each group to draw out and talk about how they will go about implementing their project without actually showing any code where the professor would give unlimited feedback until the group had the correct implementation, would benefit each group greatly. This way, no group would be “flying blind” into deliverable 2.