CSCI 362-02
Fall 2016
Final Report

Martus

Team HABA
Hugo Felkel, Arthur Hilgendorf, Brett Perrine, Angel Rodriguez

Table of Contents

Chapter 1: Introduction

What is Martus?

Martus is an open source software that allows Individuals and groups who protect human rights and civil liberties work in environments where resources are scarce and adversaries often have the upper hand. Martus is a free, open source, secure information collection and management tool that empowers these rights activists to be stronger in their fight against injustice and abuse.

Choosing our Project

At first glance, we thought Martus was a good project candidates, since the documentation seemed to be an amazing resource, but we were held back in the testing of the platform due to the restraints of using the Linux operating system. The documentation to get Martus to compile was mostly outdated, but provided a good resource, although everything had to be sticked together in order for it compile to run the designed test cases.

The constants imposed upon us such as only being able to use the Linux operating system created a scenario, in which we had to make several different developing decisions such as running an older version of Martus, since the current version has several different display issues associated with it.We were forced to use Martus 4.4 instead of the current Martus 5.1. We experienced system related issues with our virtual machines such as determining the proper amount of RAM and physical memory in order for our IDE, Eclipse, to run with minimal lag. Another issue arose when we were trying to edit the files within Eclipse, and it would continue to crash, but this issue related back to the aforementioned RAM problems associated with our virtual machines.

Chapter 2: Test Planning

After solidifying Martus as our chosen project, the group could now move forward with planning how we were going to test the functionality. Despite the setback of having to use an older version, Martus is still coded in Java, so the testing format was straightforward, at least in theory. Before setting up the testing system proper though, we had to design test cases. Martus is primarily a forum-based application, so the major functionality was not something we could reasonably test. In order to have a reasonable chance at releasing the system on-time, we needed to find methods that were self-contained, asking for simple inputs and giving a simple value back (such as a boolean).

This proved to be more difficult than initially thought. While documentation of Martus itself was decent, documentation within its source code was negligible or non-existent. The end result was all members of the group scouring through classes one by one to find appropriate methods. After much trial and error, we found our initial candidate in the form of the MartusUserNameAndPassword class, which controlled constraints with regards to a login. With this, we had an important piece of the base functionality ready to test.

The testing process was to be based on a script, which ran a suite of tests on the Martus source code. These test cases would be kept in text files, and read by the script during runtime. This necessitated the script to live in the same directory as the Martus source code. Finally, once all tests had been run, the results would need to be displayed in an HTML file, automatically opened at the conclusion of testing.

Initial Test Cases

Test ID - 1a
Requirement - The method should return true if the input password length is greater than or equal to 15 and it must contain two or more non-alphabetical characters as well as digits.
Component - Module: martus-client Package: org.martus.client.core Class:MartusUserNameandPassword.java
Method - isWeakPassword(char[ ] password)
Input - ("tested".toCharArray())
Expected Output - return true

Test ID - 1b

Requirement - The method should return true if the input password length is greater than or equal to 15 and it must contain two or more non-alphabetical characters as well as digits.
Component - Module: martus-client Package: org.martus.client.core Class:MartusUserNameandPassword.java
Method - isWeakPassword(char[ ] password)
Input - ("Testedpassword$$".toCharArray())
Expected Output - return true

Test ID - 1c
Requirement - The method should return true if the input password length is greater than or equal to 15 and it must contain two or more non-alphabetical characters as well as digits
Component - Module: martus-client Package: org.martus.client.core Class:MartusUserNameandPassword.java
Method - isWeakPassword(char[ ] password) Input - ("TestedPassword21".toCharArray())
Expected Output - return false

Test ID - 1d
Requirement - The method should not throw any exceptions if the username is longer than 0, as a password length greater than 8, and the password and username do not match each other.
Component - Module: martus-client Package: org.martus.client.core Class:MartusUserNameandPassword.java
Method - validateUserNameAndPassword(String username, char[ ] password)
Input - (“TestUserName.toCharArray()”,"TestedPassword#$")
Expected Output - No exceptions thrown

Test ID - 1e
Requirement - The method should not throw any exceptions if the username is longer than 0, as a password length greater than 8, and the password and username do not match each other.
Component - Module: martus-client Package: org.martus.client.core Class:MartusUserNameandPassword.java
Method - validateUserNameAndPassword(String username, char[ ] password)
Input - (“TestUserName.toCharArray()”,"TestedUsername")
Expected Output -Throw PasswordMatchedUserNameException

Testing Schedule

- Week of 9/25: Test plan conceived, five test cases conceived
- Week of 10/09: Driver to run tests should be finished, work on framework structure should have started. At least 15 test cases should be conceived
- Week of 10/16: Testing framework structure should be finished. Should be able to run at least one test case.
- Week 10/23: All 25 test cases should be conceived. Debugging of testing framework for other cases and a full run will be underway.
- Week 10/30: Debugging continues
- Week 11/07: Project is as bug-free as can be reasonably achieved in a period of time. System should be ready to present

Chapter 3: Testing Framework

Initial construction of the system was focused on one thing: making it run as intended with only a single test case. Once a single test case was working as intended (assuming nothing was hardcoded) then adding additional text files to be read shouldn't cause any issues, or so we believed. As it turned out, getting the script to work wasn't as difficult as initially expected, and with a series of 8 test cases, it looked like we were finished ahead of schedule. However, a chat with our customer (professor) revealed a glaring issue in communication between us and our design goals.

On the surface, our code appeared to work as intended. It read the test cases, gave the HTML file, and adding or removing new test cases didn't cause an issue. However, the script and *singular* driver were doing far more work than intended. The script was not supposed to "know" anything about the methods it was calling. It should have read the test cases, and called the driver given in the test case. However, our project had a singular driver, with the script reading the test case and figuring out which method of the driver needed to be called. In the long run, this would have been a problem, as the "super driver" would need to be altered every time a new method needed to be tested.

As a result of the above misunderstanding, we had to all but scrap the initial script and start over. Fortunately, the newer script was actually much simpler than the script that worked with our "super driver," and the individual drivers were simple to code in Java. With that fix, the system then worked as intended. Calling the script "runAllTests" caused the script to read the test cases, take the inputs and the driver that was listed, run that driver with that input, and then print the results. All results were then released in an HTML file, shown in a graph format.

Chapter 4: Test Cases

Once the architecture for testing was complete, with 8 starting test cases, all that was needed was another 17 test cases for a total of 25. Those 8 test cases had worked in the previous deliverable, so there should have been no issues.

As it turns out, this was largely what happened. Our biggest hiccup in development had been solved only a week prior to when the third deliverable was due, meaning the majority work was just adding new test cases. The only issue remaining was a slight miscommunication regarding what an oracle was. The test cases had both the expected output, and an oracle, which lead to a file that held the expected output, a needless step. That did not require much effort to remedy however. The most difficult stretch of this task was we ran out of testable methods within MartusUserNameAndPassword, causing us to once again scour Martus for reasonably testable methods. The Flexidate class turned out to be our candidate, providing methods for keeping track of dates and offering different date formats for different regions.

Final List of Test Cases:

TestID: 01
Requirement: The method should return true if the input password length is less than 15 and less than 2 non-alphabetical characters.
Component: MartusUserNameAndPassword
Method Name: isWeakPassword
Input: password
Expected Outcome: true
Driver Used: testCase1

TestID: 02
Requirement: The method should return true if the input password length is less than 15 and less than 2 non-alphabetical characters.
Component: MartusUserNameAndPassword
Method Name: isWeakPassword
Input: Testedpassword$$
Expected Outcome: false
Driver Used: testCase1

TestID: 03

Requirement: The method should return true if the input password length is less than 15 and less than 2 non-alphabetical characters.
Component: MartusUserNameAndPassword
Method Name: isWeakPassword
Input: TestedPassward21
Expected Outcome: true
Driver Used: testCase1

TestID: 04
Requirement: The method should return NoError if the password was longer than 8 characters and doesn't match the username, or return the exception generated.
Component: MartusUserNameAndPassword
Method Name: validateUserNameAndPassword
Input: TestUserName, TestPassword&&
Expected Outcome: NoError
Driver Used: testCase2

TestID: 05
Requirement: The method should return NoError if the password was longer than 8 characters and doesn't match the username, or return the exception generated.
Component: MartusUserNameAndPassword
Method Name: validateUserNameAndPassword
Input: , TestPassword&&
Expected Outcome: BlankUserNameException
Driver Used: testCase2

TestID: 06
Requirement: The method should return NoError if the password was longer than 8 characters and doesn't match the username, or return the exception generated.
Component: MartusUserNameAndPassword
Method Name: validateUserNameAndPassword
Input: TestUserName_1, TestUserName_1
Expected Outcome:PasswordMatchedUserNameException
Driver Used: testCase2

TestID: 07
Requirement: The method should return NoError if the password was longer than 8 characters and doesn't match the username, or return the exception generated.
Component: MartusUserNameAndPassword

Method Name: validateUserNameAndPassword
Input: "", TestUserName_1
Expected Outcome: BlankUserNameException
Driver Used: testCase2

TestID: 08
Requirement: The method should return NoError if the password was longer than 8 characters and doesn't match the username, or return the exception generated.
Component: MartusUserNameAndPassword
Method Name: validateUserNameAndPassword
Input: testUser, test021
Expected Outcome: PasswordTooShortException
Driver Used: testCase2

TestID: 09
Requirement: The method should return the combined password delimited by a ':' character, or the exception generated
Component: PasswordHelper
Method Name: getCombinedPassPhrase
Input: testUser, test021
Expected Outcome:test021:testUser
Driver Used: testCase3

TestID: 10
Requirement: The method should return the combined password delimited by a ':' character, or the exception generated
Component: PasswordHelper
Method Name: getCombinedPassPhrase
Input: testUser,
Expected Outcome: ArrayIndexOutOfBoundsException
Driver Used: testCase3

TestID: 11
Requirement: The method should return the combined password delimited by a ':' character, or the exception generated
Component: PasswordHelper
Method Name: getCombinedPassPhrase
Input: ,testPassCombo
Expected Outcome: testPassCombo:

Driver Used: testCase3

TestID: 12
Requirement: The method should return the combined password delimited by a ‘:’ character, or the exception generated
Component: PasswordHelper
Method Name: getCombinedPassPhrase
Input: ,,:
Expected Outcome: ::,
Driver Used: testCase3

TestID: 13
Requirement: The method should return the combined password delimited by a ‘:’ character, or the exception generated
Component: PasswordHelper
Method Name: getCombinedPassPhrase
Input: :,:
Expected Outcome: :::
Driver Used: testCase3

TestID: 14
Requirement: The method should return the correct date format delimited by the ‘-’ character, which distinguishes the year from the month and day, or return the exception generated
Component: MartusFlexidate
Method Name: toStoredDateFormat
Input: 0
Expected Outcome: 1970-01-01
Driver Used: testCase4

TestID: 15
Requirement: The method should return the correct date format delimited by the ‘-’ character, which distinguishes the year from the month and day, or return the exception generated
Component: MartusFlexidate
Method Name: toStoredDateFormat
Input: 3,11,29
Expected Outcome: 0003-11-29
Driver Used: testCase4

TestID: 16
Requirement: The method should return the correct date format delimited by the ‘-’ character, which distinguishes the year from the month and day, or return the exception generated
Component: MartusFlexidate
Method Name: toStoredDateFormat
Input: 3,11
Expected Outcome: ArrayIndexOutOfBoundsException
Driver Used: testCase4

TestID: 17
Requirement: The method should return the correct date format delimited by the ‘-’ character, which distinguishes the year from the month and day, or return the exception generated
Component: MartusFlexidate
Method Name: toStoredDateFormat
Input: 9990,18,40
Expected Outcome: 9991-07-10
Driver Used: testCase4

TestID: 18
Requirement: The method should return the correct date format delimited by the ‘-’ character, which distinguishes the year from the month and day, or return the exception generated
Component: MartusFlexidate
Method Name: toStoredDateFormat
Input: -,-,-
Expected Outcome: NumberFormatException
Driver Used: testCase4

TestID: 19
Requirement: The method should return the correct date format delimited by the ‘-’ character, which distinguishes the year from the month and day, or return the exception generated
Component: MartusFlexidate
Method Name: getMartusFlexidateString
Input: 2003-01-05,2
Expected Outcome: 20030105+2

Driver Used: testCase5

TestID: 20
Requirement: The method should return the correct date format delimited by the ‘-’ character, which distinguishes the year from the month and day, or return the exception generated
Component: MartusFlexidate
Method Name: getMartusFlexidateString
Input: 100-05-20,2
Expected Outcome: NumberFormatException
Driver Used: testCase5

TestID: 21
Requirement: The method should return the correct date format delimited by the ‘-’ character, which distinguishes the year from the month and day, or return the exception generated
Component: MartusFlexidate
Method Name: getMartusFlexidateString
Input: 0100-05-20,9999999999
Expected Outcome: NumberFormatException
Driver Used: testCase5

TestID: 22
Requirement: The method should return the correct stored dateformat by converting the input to a string and adding the delimiter ‘-’ to represent the year, month, and day, or return the exception generated.
Component: MartusFlexidate
Method Name: extractIsoDateFromStoredDate
Input: 1989-11-29
Expected Outcome: 1989--1-1-
Driver Used: testCase6

TestID: 23
Requirement: The method should return the correct stored dateformat by converting the input to a string and adding the delimiter ‘-’ to represent the year, month, and day, or return the exception generated.
Component: MartusFlexidate
Method Name: extractIsoDateFromStoredDate
Input: 189-11-29

Expected Outcome: 189--11--2
Driver Used: testCase6

TestID: 24
Requirement: The method should return the correct stored dateformat by converting the input to a string and adding the delimiter ‘-’ to represent the year, month, and day, or return the exception generated.
Component: MartusFlexidate
Method Name: extractIsoDateFromStoredDate
Input: 19891129
Expected Outcome: 1989-11-29
Driver Used: testCase6

TestID: 25
Requirement: The method should return the correct stored dateformat by converting the input to a string and adding the delimiter ‘-’ to represent the year, month, and day, or return the exception generated.
Component: MartusFlexidate
Method Name: extractIsoDateFromStoredDate
Input: 0031129
Expected Outcome: StringIndexOutOfBoundsException
Driver Used: testCase6

Chapter 5: Fault Injection

The final step to ensure that our testing frame was working properly is to ensure that it would recognize changes in the code, and our tests would fail accordingly. Breaking code theoretically should be much simpler than ensuring it runs properly, however our team did run into some issues with this. The initial problem was that no matter how we changed the source code, none of the test results would change. After some investigation, it turned out that the script was using an archived .jar file, rather than the newly compiled one. As a result, it would only test the source code that we had initially downloaded and compiled, and not anything newly changed. A little more investigation showed that the newly compiled files were not being sent where expected, and a slight change to the script fixed this issue. Once that was resolved, breaking the code was relatively easy. We initiated a series of five faults, three involving username/passwords, and another two dealing with the date methods of the system.

The combination of these five faults caused seven of our twenty-five test cases to fail, as shown below:

Fault Injection 1: Removed the catch BlankUserNameException from the method validateUserNameAndPassword inside the Martus file, MartusUserNameAndPassword.

- Classpath: org/martus/client/core/MartusUserNameandPassword.java
- Results: Causes Test Case #5 and #7 to change the error generated from BlankUserNameException to NoError

Fault Injection 2: Removed the catch PasswordMatchedUserNameException from the method validateUserNameAndPassword inside the Martus file, MartusUserNameAndPassword.

- Classpath: org/martus/client/core/MartusUserNameandPassword.java
- Results: Causes Test Case #6 to change the error generated from PasswordMatchedUserNameException to NoError

Fault Injection 3: Add the catch cause InputMismatchException if the String variable was empty from the method getCombinedPasswordPhrase inside the Martus file, PasswordHelper.

- Classpath: org/martus/clientside/PasswordHelper.java
- Results: Causes Test Case #11 and #12 to change the error generated from ArrayIndexOutOfBounds to InputMismatchException

Fault Injection 4: Changed the methods getMartusFlexidateString and toFlexidateFormat to return integers instead of Strings inside the Martus file, MartusFlexidate.

- Classpath: org/martus/common/utilities/MartusFlexidate.java

- Results: Causes Test Case #19 to change the result generated from 20030105+2 to the error NumberFormatException

Fault Injection 5: Add the catch cause NumberFormatException if the String variable was less than 7 characters in length from the method extractIsoDateFromStoredDate inside the Martus file, MartusFlexidate.

- Classpath: org/martus/common/utilities/MartusFlexidate.java
- Results: Causes Test Case #25 to change the error generated from StringIndexOutOfBoundsException to NumberFormatException

With the faults all causing our code to break as expected, we could now conclude our system was about as bug-free as we could reasonably make it during the course of a single semester.

Chapter 6: Experiences and Reflections

Brett Perrine:

I think this project has been a very beneficial one, particularly regarding our experience working in a group environment without a strict “overwatch” so to speak. Working with all of our individual schedules was certainly a challenge as well as working in a semi-remote environment as one of our team members was running their virtualbox from a desktop. Also, myself and another one of my teammates had work schedules to work around which made physical meeting even more complicated, thus is was critical we be adamant about communication. The project also gave valuable insight into the importance of testing and the extent at which even the Martus development team (who were lacking in documentation throughout their repository) still built a multitude of testing modules to ensure their code functioned properly.
I think this project will be a very useful moving forward into our practicum, as it will be similarly styled in that we will be working in small, self-managed teams as well as begin implementing agile / scrum methodologies. Moreover, understanding the differences in experiences between team members is a notable experience as well. Seeing who tends to fall into which roles and which they thrive in is one that could potentially help aid in leadership roles in the workplace.

Hugo Felkel:

The overall experience of this project provided a great learning platform through which we all gained practically working experience by understanding core concepts such as architecture and design, which needed to be understood in order to build our automated testing framework. My overall view on this project has been positive, since I have enjoyed both the experience and the challenges associated with the project. I have learned the value of teamwork, since a programmer will make mistakes that they won’t be able to see or correct, and thus their teammates will have to spot those mistakes in order for the project to succeed. The primary take away from this project should focus on the importance placed on communication.

Arthur:

This project has offered a great deal of insight into working as a team on a singular goal. The most important thing throughout this project has been communication, which should have been obvious. However, the level of communication to maintain production was beyond what I have grown to expect from other courses. This is communication

between members of the group, as well as communication between the group and the customer (professor). One of the biggest hiccups was a direct result of misunderstanding. Our code appeared to work, for all intents and purposes, but it did not work in the manner the professor desired. It was not until we had a direct conversation with him and mentioned a *single* driver that he realized something might be amiss. Beyond that, I have learned to appreciate the value of documentation *significantly* more than before. I always knew it was important, but the complete lack of comments in Martus's code has made locating methods for testing a much larger nightmare than it needed to be. I would honestly prefer more projects of this nature in the future.

Angel:
I have found that this project has been very informative as to how difficult real world experiences in the software engineering field can be. This project has been very eye opening as to what to expect and I have gained valuable insight on working on a project with team members. Although I have had experience working with code that is not my own, I have never worked with someone else's code to this scale. We were exposed to the difficulty of working with code that mostly undocumented, communication with team members and specifying requirements with the consumer. Overall we were able to overcome these challenges and have acquired valuable knowledge which has lead to a positive outcome.

Suggestions:

In the future, clarity on what the project specifications and requirements are would be appreciated. There is the written requirements, but multiple times these written directions were misinterpreted.

During the semester, multiple class periods were spent as “work days” during which students could work on their project if they didn’t have another time to meet. Using this time to go over specifications as to what the customer wants in the testing framework would be beneficial.

An initial list of open-source projects was given at the start of the class, and the entire class was told to pick from this list. Yet a week or two into the project we were told we could work on *any* open-source project; which would have made several options available that are better documented and better supported. Making this option known at the start of the semester would be appreciated