**Fat Mike’s Pizza Joint**

*Chandler DeLoach, Ethan Hendrix, Jayse White*

*Dr. James Bowring*

*College of Charleston, Department of Computer Science*

**Chapter 0**

Our top 3 Candidates:

Cadasta: "Technology to help communities document their land rights around the world."

Main website: http://cadasta.org/

GitHub: https://github.com/Cadasta/cadasta-platform

Martus: "Martus, the Greek word for "witness," is a software tool that allows users to document incidents of abuse by creating bulletins, and storing them on redundant servers located around the world. Using Martus helps countries torn apart by civil conflicts come to a consensus and rational understanding of their histories, leading to reconciliation and reform processes."

Main website: https://www.martus.org/

GitHub: https://github.com/benetech/Martus-Project

Insight Segmentation and Registration Toolkit (ITK): "National Library of Medicine: ITK is an open-source, cross-platform system that provides developers with an extensive suite of software tools for image analysis. Developed through extreme programming methodologies, ITK employs leading-edge algorithms for registering and segmenting multidimensional data."

Main website: https://itk.org/

GitHub: https://github.com/InsightSoftwareConsortium/ITK

Reasoning:

Cadasta caught our attention, because it is an interactive map to document land rights around the world. This is particularly interesting because of the political implications of land rights, especially in regions of the world such as the Middle East. Cadasta is primarily implemented in Python, and its GitHub repo has just under 1,000 commits.

Martus caught our attention, because it is software aimed at aiding less-developed countries in documenting and understanding reports of abuse. Additionally, it was interesting that Martus emphasizes security and redundant servers. Martus is implemented in Java, and is hosted on a much smaller repository than Cadasta or ITK.

ITK caught our attention, because it's aimed at aiding devlopers with image analysis. Likewise, it is not enduser software, but a library given to developers that are building tools for endusers in the medical field. It's a huge project with over 45,000 commits, so this would provide invaluable experience for our team with working with large projects. Plus, it's an excuse to learn C++.

**Chapter 1**

Originally, we went with Martus for our project and were aiming to develop a testing framework for this. However, after some digging, we discovered that the project was dead. Many of the dependencies that were required for compiling and running this code were out-of-date and many of them were not hosted for download any longer. Given that we could not compile the code independently and that the project was dead, we decided to consider changing projects.

We went back to the drawing board, and revisited our second choices for projects. Cadasta, a land-rights documentation program, caught out attention. There were over 1,000 commits and they were as recent as the previous week. Additionally, the dependencies were up-to-date and their github wiki had an easily understood installation instructions for development. Essentially, to install Cadasta for development, the user creates a Vagrant VM to run as their Cadasta server, and then they can SSH into the server to run commands.

What we found inside was that Cadasta had a fairly comprehensive testing framework already implemented. Runtests.py runs all of their unit tests and reports the results. Additionally, they provide the option to export these test results to html files. After the tests are finished, it reports a quick overall result: "1998 passed, 501 warnings in 972.86 seconds."

When examining the test result's directory, what you find is a collection of many html files that correspond to each unit of the software. At the bottom, is an html file labeled "index" that provides a quick overview of all test results. If you wish to view a specific test, however, you can browse to that html file specifically. What this does is it allows a developer to get a quick, comprehensive overview of all the unit tests, but still allows them to delve deeper and get a granular view of the unit tests.

Altogether, Cadasta appears to be a much more approachable project that is aimed at OSS development. The easy installation instructions combined with a well-maintained code base give much more confidence than Martus. Additionally, the current testing framework inspires ideas for furthering the testing database.

**Chapter 2**

Fat Mike’s Pizza Joint - Testing Plan

The Testing Process

Write Tests

Automate running tests

Collect Data

Log testing process

Compare to oracles

Prepare results for overview

Requirements Traceability

Accounts -Manage multiple project and organization accounts for one user

Projects

Manage projects, the relationship between it, its users, and its parent organization

Organizations

Manage relationships between an organization and its project(s)

Pages

Manage display of data in a graphical user interface

Users

Manage accounts for one user and the relationship between it and all of its organizations and projects

Test Items

Managing Cadasta Accounts

Managing Cadasta Organizations

Test Recording Procedures

Run automated tests

Compare oracles

Compare report

Log results

Hardware and Software Requirements

Hardware requirements:

x86 or x64 system

Available 2 GB for dynamically allocated memory

Software Requirements:

Linux

VirtualBox 5.0.x

Vagrant 1.8.1

Ansible 2.1.3.0

Constraints

Time:

This testing process will occur over a four month period

Version control:

Keeping team’s code base up-to-date through Git

Organization

Manage team organization through project management software(Teamwork)

Initial Test Cases:

Test Case: CSVImporter.get\_headers() from vagrant/cadasta/organization/importers/csv.py

Input: .CSV file

Output: file pointer

Tests:

Valid CSV file

Corrupt CSV file

Improperly delaminated CSV file

Invalid file type

File not found

Test Case: EmailSimilarityValidatorvalidate() from vagrant/cadasta/accounts/validators.py

Input: User & Password

Output: None object or error

Tests:

Valid User

Valid password

Invalid user

Invalid password

Password containing email

Test Case: XLSImporter.get\_header\_map() from vagrant/cadasta/organization/importers/xls.py

Input: XLS

Output: Headers

Tests:

File not found

Corrupted XLS file

Small XLS file

Large XLS file

Non-XLS file

Standard XLS file

Test Case: validate\_row() from /vagrant/cadasta/organization/importers/validators.py

Input: rows from CSV file

Output: tuple of field values

Tests:

Correct row format

NULL row

NULL headers

Unexpected dimensions length

Unexpected row format -Test Case: CharacterTypePasswordValidator.validate() from vagrant/cadasta/accounts/validators.py

Input: Password, User

Output: void or error message

Tests:

Password contains enough specified unique characters type

Password does not contain enough specified unique character type

Password contains invalid character type

valid User

invalid user

**Chapter 3**

For the third deliverable, we worked on implementing our testing driver. The driver will read from testCase#.yml files.

Our scripts and driver came along nicely, but we ran into some roadblocks, the biggest of which was getting the driver to run on our code base. The driver and scripts currently are proof-of-concept, and, in the future, we will be looking to make our driver run over our code base.

Below is an example of one of our five test cases:

name: Password containing email requirement: Email Similarity component: class: EmailSimilarityValidator provider: vagrant/cadasta/accounts/validators.py method: validate() input: constructor: Object method: "ValidEmail@website.com","ValidEmail@website.com" oracle: Error

In this test case, we were entering a password that contains the email. The EmailSimilarityValidator should return an Error, which is reflected in the Oracle. The other fields of this test case will be used to import necessary files from the code base when we finish implementing this in the future. All five test cases are listed below.

test case 1: name: Valid Username requirement: Email Similarity component: class: EmailSimilarityValidator provider: vagrant/cadasta/accounts/validators.py method: validate() input: constructor: Object method: "ValidPassword","ValidEmail@website.com" oracle: None

test case 2: name: Vallid Password requirement: Email Similarity component: class: EmailSimilarityValidator provider: vagrant/cadasta/accounts/validators.py method: validate() input: constructor: Object method: "ValidPassword","ValidEmail@website.com" oracle: None

test case 3: name: Invalid Username requirement: Email Similarity component: class: EmailSimilarityValidator provider: vagrant/cadasta/accounts/validators.py method: validate() input: constructor: Object method: "ValidPassword", "invalidUserEmail" oracle: Error

test case 4: name: Invalid Password requirement: Email Similarity component: class: EmailSimilarityValidator provider: vagrant/cadasta/accounts/validators.py method: validate() input: constructor: Object method: "Invalid Password?!@#($(","ValidEmail@website.com" oracle: Error

test case 5: name: Password containing email requirement: Email Similarity component: class: EmailSimilarityValidator provider: vagrant/cadasta/accounts/validators.py method: validate() input: constructor: Object method: "ValidEmail@website.com","ValidEmail@website.com" oracle: Error

While these test cases above were our original idea for testing the Cadasta project, this did not come to fruition. After struggling with Django imports, we revised our project. At this point in time, we are developing a testing framework, but, instead of running this testing framework over the Cadasta project, we will run the testing framework over itself, using test cases that are written for the framework's individual pieces.

For the third deliverable, we worked on implementing our testing driver. The driver will read from testCase#.yml files.

Our scripts and driver came along nicely, but we ran into some roadblocks, the biggest of which was getting the driver to run on our code base. The driver and scripts currently are proof-of-concept, and, in the future, we will be looking to make our driver run over our code base.

Below is an example of one of our five test cases:

name: Password containing email requirement: Email Similarity component: class: EmailSimilarityValidator provider: vagrant/cadasta/accounts/validators.py method: validate() input: constructor: Object method: "ValidEmail@website.com","ValidEmail@website.com" oracle: Error

In this test case, we were entering a password that contains the email. The EmailSimilarityValidator should return an Error, which is reflected in the Oracle. The other fields of this test case will be used to import necessary files from the code base when we finish implementing this in the future. All five test cases are listed below.

test case 1: name: Valid Username requirement: Email Similarity component: class: EmailSimilarityValidator provider: vagrant/cadasta/accounts/validators.py method: validate() input: constructor: Object method: "ValidPassword","ValidEmail@website.com" oracle: None

test case 2: name: Vallid Password requirement: Email Similarity component: class: EmailSimilarityValidator provider: vagrant/cadasta/accounts/validators.py method: validate() input: constructor: Object method: "ValidPassword","ValidEmail@website.com" oracle: None

test case 3: name: Invalid Username requirement: Email Similarity component: class: EmailSimilarityValidator provider: vagrant/cadasta/accounts/validators.py method: validate() input: constructor: Object method: "ValidPassword", "invalidUserEmail" oracle: Error

test case 4: name: Invalid Password requirement: Email Similarity component: class: EmailSimilarityValidator provider: vagrant/cadasta/accounts/validators.py method: validate() input: constructor: Object method: "Invalid Password?!@#($(","ValidEmail@website.com" oracle: Error

test case 5: name: Password containing email requirement: Email Similarity component: class: EmailSimilarityValidator provider: vagrant/cadasta/accounts/validators.py method: validate() input: constructor: Object method: "ValidEmail@website.com","ValidEmail@website.com" oracle: Error

While these test cases above were our original idea for testing the Cadasta project, this did not come to fruition. After struggling with Django imports, we revised our project. At this point in time, we are developing a testing framework, but, instead of running this testing framework over the Cadasta project, we will run the testing framework over itself, using test cases that are written for the framework's individual pieces.

**Chapter 4**

At this point during the development of our testing framework, we are currently working on an implementation that will test itself.

After some difficulty managing Django and its imports within the Cadasta project, we once again changed the goal of our project to accommodate this unintended complexity. The project is now to create a testing framework, and use test cases that will test the framework's individual pieces. We were able to do this, because, up to this point, we were developing with the goal in mind that this testing framework would be generic enough that anyone could run this framework over their own project so long as they correctly wrote a test case according to our specified syntax.

We worked on implementing at our full test cases: 8 test cases for parsing files, 5 test cases for parsing singles lines, 2 test cases for a store\_value function, and then 12 test cases on our Driver class. In total, this resulted in 27 test cases. While this is more than the specified 25 objective, we felt that this adequately tested our framework in its current state.

**Chapter 5**

For this deliverable, we were focusing on fault injection. Since we were testing our own code base, however, we had to once again work around the problem of injecting faults into our code base while still being able to test it using our code base.

We made copies of necessary files in our src directory into a directory called "brokenPythonFiles," then we pointed our Driver at these files instead of itself moving forward. The main files we focused on were TestCase.py, and we sought to inject faults into the parser and other TestCase methods.

We were successful in doing so. We were able to produce seven faults in total that would cause our tests to fail, but, thankfully, not all of the tests failed.