# Team Term Project

## Syntax-Error

*(Daniel Feliciano, Brenard Casey, Carson Smith )*

## Software Engineering CSCI-362

## December 1, 2016

## TABLE OF CONTENTS

## Introduction

The purpose of this project was to find an open source project and build an automated testing framework (on a linux operating system, specifically Ubuntu) that would test methods of the chosen project. Throughout this process we documented obstacles and other related experiences that we encountered.

## Chapter I: Building the Project

*PokemonGo-Map*

We chose a live third-party visualization of the PokemonGo-Map. It is an interactive graphical map application based on the popular television show and now game, Pokemon. The PokemonGo-Map application is used to find and search for Pokemon given latitude/longitude coordinates and range values. Range values determine how far the application will search from the given latitude and longitude coordinates. Other features include the ability to retrieve spawn time frequency of Pokemon.

Compiling the source code initially was challenging. The required dependencies needed to run the application were large in size and quantity. Outside of installing all components listed in the required.txt file, there were numerous additional packages that had to be installed. Some of these packages were over 200MB. Because we are using Virtual Machines to run linux (specifically Ubuntu), disk space became a problem. On initial setup, we allocated approximately 8GB of disk memory. The 8GB of memory became depleted once all required and additional packages were installed. As result the system would not allow us to run any execute any commands . To correct this issue we freed disk space through system recovery options.

Once the system memory issue was resolved, signing remotely into our Pokemon account became a hassle. When signing in remotely from the terminal, we would receive messages that our Pokemon account "may have been banned". In order to get the graphical map to populate with Pokemon at the location where they are located, we had to sign into our Pokemon Trainer Account. After extensive research of this issue we found that this was a common issue. Some reported success after brute force approach… making a multitude of Pokemon accounts and attempting to login with different ones until success was reached. This approach was not successful for us.

After extensive research, we able to successfully login and run the application by changing a few credentials in our Pokemon account. Unfortunately, a few days later the PokemonGo-Map application was forced to shut down because Niantic, the Developers of Pokemon-Go, closed public access to their API.

*RiotWatcher*

As a result of the PokemonGo-Map ban, we found a project that was less likely to be banned... RiotWatcher. RiotWatcher is wrapper for the League of Legends API. We choose this project because of the readability of the source code, and the mass amount of methods that we could test. The dependencies needed to execute the code was provided in a setup.py file. This made installing the necessary dependencies to run the source code straightforward. Also, the provided ReadMe file stated all command line parameters needed to run the code. The only parameter that was not provided was the an account api-key. We retrieved this through our personal account. Once we obtained our api-key, we were successfully able to execute the code.

## Chapter II: Test Plan

We developed our testing framework in Python largely because our project, the Riot Games API Python Wrapper (formerly the Pokemon GO Pokemon Finder), is also written in Python. We believed it would be easiest to associate our project and framework with each other if we developed using the same language and also we all had a familiarity with Python.
The way we intend for our test driver to work is by reading the directory of test cases and grabbing the required information from each test case. This template includes information such as the test id, the requirement being tested, the component being tested, the input of the test case, the testable method, and the expected output. This information will be read from each .txt file and loaded into an array of test case objects. Once the test cases are finished loading, the methods that we are testing will be run and the result of the method calls will be compared to the expected output of each test case.

Finally, we plan to print the results to the browser in an easy to read, digestible format.

Originally we designed test cases for the PokemonGO-Map application, but since it has been taken down we had to work to create 'dummy' test cases to work with our test driver to demonstrate in class. We wrote a simple add method in Python, that took two integers and added them together, in order to demonstrate our testing framework's ability to complete tests.

As far as our team goes, we are working very well together. We each have different skills that we bring to the table and each have a unique way of approaching our problems. We think that we will be able to get a lot done and be very efficient and flow well throughout the project.

## Chapter III: Testing Continued

Throughout working on our project, we came up with how our testing framework would work. We were able to pretty much fully implement it as we thought, but we ran into a few minor errors that had to be corrected. The most time consuming portion of this process was getting our testCase.txt files to be read. For a long span of time (over many hours) our cases continually failed. We then thoroughly combed through all our files and we discovered that the contents of the text file had been erased. After repopulating the file we got our expected results.

After that, we ran into the issue of the program not really wanting to work on Carson's windows laptop. We were able to get it to run on Casey's and Daniel's VM, Carson's desktop (which is windows), as well as a VM on Carson's machine, but not the Windows laptop. It was not that big of an issue, but Carson preferred to work on Windows with some of the code editors he already had rather than the ones he had installed on his VM. We never really figured out why it didn't want to work on the Windows laptop, although it worked on a different windows machine.

Past that, we developed our first couple of test cases using the static_get_champion method and everything went well.

## Chapter IV: Completing the Test Framework

We have 25 test cases derived from the following 5 methods:

*static_get_item*
*static_get_summoner_spell*
*static_get_mastery*
*static_get_champion*
*static_get_rune*

The testing framework is fully functional and complete. We have designed our twenty-five test cases around the above methods. We choose these because we thought they were easiest, and we were familiar with some of the subject material of what the methods deal with (what a champion is, what an item is, etc.) so that we would be able to easily determine if we were getting erroneous or odd results. Along with this, we developed a few extra test cases to ensure that the information was correct, and in the event that some of the data changes in the coming days.

While we do not believe the data will change and don't think our driver will be affected at all by it, a new season of League of Legends has just begun, and therefore Riot is in the process of evaluating items, runes, and masteries and evaluating whether or not they should stay in the game. This information has been made pretty open to what they intend on changing, and therefore we're not very worried about it, but we developed the extra test cases just in case.

## Chapter V: Breaking the Code

Injecting faults into the code proved to be harder than anticipated. This was due to well written code. The methods all make a call to a base *get_request* method, which determines what type of information needs to be grabbed and pushed back based on the information supplied by the "parent" function. For example, a call using *static_get_champion* will call *static_base_request* and *base_request* will determine that a champion needs to be grabbed and returned. Then, *static_get_champion* formats the data and sends it out as a result. These succession of events made it difficult to break one method without breaking others.

However, we decided to break the following methods: *static_get_item* and *static_get_mastery*. This was accomplished by setting the ID of the input to 2009 and 2133, respectively. This forces the output to be the results as certain items, which makes our test cases fail (as they should). Only one of our mastery test cases will still pass as we used a mastery that our test case was already searching for. This was the most challenging portion of the project for us. It involved installing a new virtual machine… as we couldn't figure out why our project was not producing failing results. After countless hours we realized that our project needed to be recompiled after making changes. We were not accustomed to a python project needing to be "compiled". After re-compiling, are fault injections successfully worked.

## Chapter VI: Our Experience

This project showed us how important planning, scheduling, and research is when it comes to project design. We initially chose our project (PokemonGo-Map) out of thrill and excitement for the game itself. We neglected to deeply research potential problems that could occur, and failed to accurately assess the complexity of the source code. If we had better researched issues regarding the PokemonGo-Map we would discovered the possibility of the app being banned, as similar apps if not identical were already banned. Because of our oversights and poor research, we spent a lot of time getting the source code to compile only to ultimately switch projects.

Having a detailed schedule for working on different portions of the project proved beneficial. This aided us in ensuring that requirements were met, and kept us from falling behind on deliverables. It also allowed for use to divvy tasks so that everyone was responsible for a certain portion of the project.

All-in-all, our grouped meshed well together. We had not worked together prior to this project, but meeting regularly broke the ice for us.

## Chapter VII: Evaluation

One thing we noticed when choosing our project was that a lot of the other groups sole choose projects from the HFOSS list. It seemed that they developed an understanding that they could not choose a project that was not from the list. A lot of our group excitement came from searching and finding projects that were all interested in, but still feel within the realm of the given requirements. We believe that if it were more transparent that projects could be chosen outside of the HFOSS list, it would add an extra level of excitement and motivation to the class.