Our testing framework is now fully complete, with all of the changes that we discussed needing to be implemented as of Deliverable #4 fully implemented. We have all of our test cases written, scanned, and evaluated, and the results are being printed to a formatted html page.

The last thing we have had to work on is Fault Injection into our project's code. Unfortunately for us, we have ran into some issues determining where to inject problems into the code. Because the methods are all written to be, look, and operate in the same way it's difficult to change some of the methods to intentionally make some of our tests fail. 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.

Due to this, we struggled finding a way to mess up one of the methods that didn't involve breaking the base_request method.

However, our team decided to break the following methods: static_get_item and static_get_mastery. The way we did this was by setting the ID of the input to be 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 definitely the hardest step of the entire project for us. It involved us installing a fresh new virtual machine as we couldn't figure out why our project was refusing to fail. We finally realize after countless hours that our project was not changing because we needed to compile the project again. We were not used to a python project needing to be "compiled", so this threw us for a loop for sure! After we finally realized this, it was a breeze to get our test cases to fail.

Another important thing we realized was that our Github was missing our project's setup.py file. Without this, the project wouldn't run on a fresh machine, so it's a good thing we caught it!

Ultimately, it's a good thing that it was so difficult for our project to be broken. That means that the python wrapper project that we chose was written effectively…… maybe we found the first person that writes perfect code.