Deliverable #5

Task: Design and inject 5 faults into the code you are testing that will cause at least 5 tests to fail, but hopefully not all tests to fail. Exercise your framework and analyze the results.

Experience

Our experience with fault injection proved to be easier than expected compared to the last deliverables. I think it is because it is a lot easier to make code not work than to make code work. We began with a very small bank of methods so it was really just figuring out tiny bugs we could insert that would not make all 25 tests fail. We were able to inject minor faults such as changing formatting or changing the division of the seconds in order to produce the incorrect amount of seconds on a log timer.

Fault Injection #1

Method being tested: log(msg, args, kwargs)
The first fault we injected we in the Log method and we deleted a * with in the . The log method before the injection prints a message beginning with a * to the stdout.

Results: Previous to the fault being injected all 25 tests pass. After the fault was injected 16 tests failed because because the methods required a * to be present.

Fault Injection #2

Method being tested: log_nostar(msg, args, kwargs)
The second fault we induced was to add an extra line to the stdout in the log_nostar method. This should cause all of our tests to fail because all of our tests rely on log_nostar to print to stdout with no star.

Results: Before the fault was injected all 25 tests pass and after the fault was injected all 25 tests failed because all the methods being tested used log_nostar in some capacity.

Fault Injection #3

Method being tested: log_time(self, msg, args, kwargs)
This method previous to the fault injection keeps track of the time and prints the time to the stdout. The method in order to keep track of seconds in a minute divides by 60. Our fault will cause the method to divide by 1 in order to give us the incorrect amount of seconds.

Results: Before the fault was injected all 25 tests passed. After we induced the fault 5 out of 25 tests failed because the seconds on the log timer were incorrect.

Fault Injection #4

Method being tested: log_time(self, msg, args, kwargs) Our fault we will be injecting is altering the format in which the log timer prints to stdout. Before the fault the format looks like this. Our fault will remove the “s” at the end of the stdout message causing some tests that require the correct format to fail.

Results: Previous to our fault all 25 tests passed. After we induced the fault 7 out of 25 tests failed because they required an “s” indicating seconds at the end of the stdout message.

Fault Injection #5

Method being tested: log_time(self, msg, args, kwargs)
Previous to the fault log_time logs logs seconds as floats. The fault we will be injecting will cause the log_time method to log seconds as ints thus causing any tests that require]ing fractions of seconds to fail.

Results: Before our fault 25 tests passed and after we induced the fault 4 out of 25 tests failed. The reason they failed was that we rounded all the decimals causing any of our tests that required fractions of seconds to fail.

We were able to successfully inject five different faults in order to simulate a real life scenario in which we run our tests frequently and another engineer were to come along and change our code and our tests began to fail. This demonstrates the necessity for running tests constantly because it allows us to catch any minor bugs that may have been injected by other engineers.