Automated Testing Framework for Amara

Josh Jettie and John Maruhn
CSCI 362 Software Engineering
College of Charleston

Introduction

Team Blue's experience while developing an automated testing framework of the open source H/FOSS project Amara.

Amara is an award winning subtitling software that allows anyone to subtitle a YouTube video. Amara's main purpose is to bring accessibility to more people through subtitled videos.

Deliverable #1

Checkout or clone the project from its repository and build it. Run existing tests and collect the results.

Step 1:

Download and Install Docker

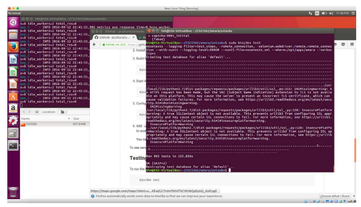
Step 2:

Clone Amara into Git and Docker

Step 3:

Build and Run Amara!

Once Amara is built and running we were able to navigate to their built in test suite and run it.
Their built in test suite has 965 Tests.



After running the test suite we noticed that all the tests passed but there were a couple tests that have problems with libraries.

Deliverable #2

Create 5 of the eventual 25 tests cases to be included in the automated testing framework.

Test One
Test to see if the data we sent to the database was actually stored within the database.

Method being tested:
Check_user_data

Test Two
Instantiate a blank user.

Method being tested:
Test_create_user_blank_data

Test Three
Checks is a non staff user can create comments.

Method being tested:
User_can_edit_subtitles

Test Four
Updates the comments section when given a post request.

Method being tested:
Update_comment

Test Five
Tests to see if we can create a video from a specific URL.

Method being tested:
test_create_video

Deliverable #4

Compete the design of our framework and tests all 25 tests.

Methods Tested:
LoggingTimer()
Log()
Log_nostar()

We were able to write 25 tests that all passed on the three methods above

This was the first Deliverable where we ran into dependency problems. An inability to easily and consistently import modules from within Amara's codebase caused us to locate methods to test that did not involve a database. Because we were not able to use the database we were forced to scratch our original test plan and only test the three methods listed to the left.

Deliverable #3

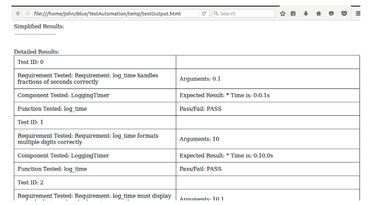
Build an automated testing framework to implement the test plan. Framework was written in Python.
Here are the steps on how to run the framework.

Step 1.
Install Git onto your machine.

Step 2.
Clone our repo from https://github.com/CSCI-362-02-2016/Blue

Step 3.
Navigate to the Scripts directory

Step 4.
Run the file runAllTests.py with Python



Step 5.
An HTML page should automatically open with test results.

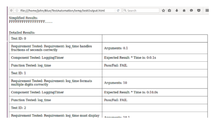
Deliverable #5

Inject faults into our program in order to simulate a third party changing our code. Our tests cases were able to pick up the faults that we injected.

Fault 1.
Deleted a star within the log method causing a formatting error

```
def log_nostar(msg, *args, **kwargs):
    sys.stdout.write(msg.format(*args, **kwargs))
    #sys.stdout.write("\n")
    sys.stdout.write("\n")
    sys.stdout.flush()
```

Results: All the tests that required the start to be within the format failed.



Fault 2.
Added an extra line to the method

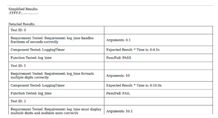
```
def log(msg, *args, **kwargs):
    #log_nostar(" " + msg, *args, **kwargs)
    log_nostar(" " + msg, *args, **kwargs)
```

Results: All the tests that required the start to be within the format failed.

Fault 3.
Changed the method to divide by one instead of 60.

```
def log_time(self, msg, *args, **kwargs):
    total_time = time.time() - self.start_time
    #mins, secs = divmod(total_time, 60)
    mins, secs = divmod(total_time, 1)
    msg = msg.format(*args, **kwargs)
    log("{}: {}:{:0.1f}s", msg, int(mins), secs)
    #log("{}: {}:{:0.1f}", msg, int(mins), secs)
    #log("{}: {}:{:0.1f}s", msg, int(mins), int(secs))
    self.reset()
```

Results: This caused all the tests that relied on the seconds to be correct to fail.



Fault 4.
Altered the format in which Log_time() prints to stdout.

```
def log_time(self, msg, *args, **kwargs):
    total_time = time.time() - self.start_time
    mins, secs = divmod(total_time, 60)
    #mins, secs = divmod(total_time, 1)
    msg = msg.format(*args, **kwargs)
    #log("{}: {}:{:0.1f}", msg, int(mins), secs)
    #log("{}: {}:{:0.1f}s", msg, int(mins), int(secs))
    self.reset()
```

Results: This caused all he tests that used Log_time() to fail.

Fault 5.
Changed the Log_time() method to log seconds as Ints instead of Floats

```
def log_time(self, msg, *args, **kwargs):
    total_time = time.time() - self.start_time
    mins, secs = divmod(total_time, 60)
    #mins, secs = divmod(total_time, 1)
    msg = msg.format(*args, **kwargs)
    #log("{}: {}:{:0.1f}", msg, int(mins), secs)
    #log("{}: {}:{:0.1f}s", msg, int(mins), int(secs))
    log("{}: {}:{:02d}", msg, int(mins), int(secs))
    self.reset()
```

Results: This caused all the tests that had fractions of seconds to fail.