Soft Boys: Deliverable #2

Joe Ayers, Maz Little, Jasmine Mai, Joe Spencer

Chapter 2: Test Plan

Introduction

The four of us approached the testing plan by dividing the work between ourselves, and discussing our results to ensure we ended up on the same page. In order to implement an organized testing framework, a templated must be implemented. To show off our work, a design will be created and displayed at the end of our project. The project structure has now been divided up so each team member is responsible for one deliverable component of the final project (detailed below). The testing schedule has also been outlined with sub-deliverables due for each class for the rest of the semester (the more granular schedules are up to the lead for each sub-deliverable). These sub-deliverables were selected from the project specifications.

Team Structure:

All team responsibilities	Be able to run environment on their machine Clerical/formatting/documentation for assigned deliverables Contribute to completion of all deliverables
Lead responsibilities	Plans deliverable component schedule, including sessions to work on code Assigns team members tasks and follows up Has final say on decisions made about deliverable
Joe Ayers - Scripter	Lead on runAllTests script

	Compiles Final Report PDF
Maz Little - Admin	Lead on Testing Report Creates final presentation
Jasmine Mai - Framer	Lead on Test Case Template Designs final poster
Joe Spencer - Tester	Lead on Test Cases Compiles final GitHub/Wiki/Blog downloading and formatting

The testing process

We will start by testing top tier components such as the activities ability to start and stop and work our way down to more basic components such as the system's ability to store the users age. Individual components will be removed to test and tests will be automated using individual .py files.

Requirements traceability

In order to provide thorough and consistent testing, documentation will be provided through each process. Documenting and independently testing whether or not the requirements are met. In order to due so, a testing schedule is being implemented.

Tested items

Components within sugarlabs, including those in the jarabe folder.

Testing schedule

An overall testing schedule and resource allocation. This schedule should be linked to the more general project development schedule.

- **10/3 - finished test plan with five general test cases**
- **10/31 - first test framework**
    - rough ideas for 25 test cases
    - test case template (Jasmine)
    - testing report (Maz)
    - runAllTests script (Joe A)
    - 25 test cases (draft code) (Joe S)
    - 10/12 - 10 test cases
    - 10/19 - 15 test cases
    - 10/26 - 20 test cases
- 11/7 - refine test framework
- 11/9 - compile test reports
- **11/14 - finished and implemented test framework with 25 test cases**
- 11/16 - final pass at test framework
- **11/21 - fault testing**
- 11/21 - presentation finished; final report first draft
- 11/21 - poster draft finished
- 11/27 - have printed poster
- 11/27 - review final report draft
- **11/28 - final report**

Test recording procedures

All test cases will be logged and named by the date and time that the test case code executed (e.g. tclog_2017.10.02_13.42.45).

Hardware and software requirements

Sugar Labs outlines their system requirements rather explicitly. On their website they state that the computer on which the software is run needs to have “A 400 MHz or faster processor, At least 512 MB memory (RAM), 1 GB recommended for best performance.” Dedicated graphics cards are not required, but the ability to handle limited graphics is a necessity. Graphics are handled by a window manager tool called X that needs to already

be installed on the system. Additionally, Sugar is intended to be distributed to and used on linux-based operating systems. Their recommendations are Ubuntu 16.04 or Fedora 19. Sugar can be installed on either 32-bit or 64-bit Operating systems.

Constraints

Constraints involved during testing consists of limited time and team members. The hardware and software requirements could also be considered as a constraint.

Test Cases

1. **Case 1:**
    a. Test ID: 001
    b. Component being tested: Application launch
    c. Method being tested: launch_and_stop_activity
    d. Requirement being tested: Operating system must be able to access Sugar activities installed on the operating system.
    e. Test input(s): ["Browse"]
    f. Expected outcome: The application will launch, then close, outputting a print statement for each action.
2. **Case 2:**
    a. Test ID: 002
    b. Component being tested: Application launch
    c. Method being tested: launch_and_stop_activity
    d. Requirement being tested: Operating system must be able to handle attempts to access Sugar activities not installed on the operating system.
    e. Test input(s): [“Bingo”]
    f. Expected outcome: The system will print error messages for the first input.
3. **Case 3:**
    a. Test ID: 003

b. Component being tested: Application launch
c. Method being tested: launch_and_stop_activity
d. Requirement being tested: Operating system must be able to handle attempts to access Sugar activities not installed on the operating system.
e. Test input(s): []
f. Expected outcome: The system will print an error message for a lack of input.

4. **Case 4:**
   a. Test ID: 004
   b. Component being tested: agepicker.py
   c. Method being tested: calculate_birth_timestamp
   d. Requirement being tested: given an age input the system creates a birth_timestamp
   e. Test input(s): 17
   f. Expected outcome: 978283893.3076539 (depending on current time)

5. **Case 5:**
   a. Test ID: 005
   b. Component being tested: Web service
   c. Method being tested: test_webservice
   d. Requirement being tested: Given a username and password, validation of correct inputs
   e. Test input(s): [“ , ”, “admin, ”, “ , password”, “admin, password1”, “admim, password”, “admin, password”]
   f. Expected outcome: Only the correct username and password will give validation