Members: Vu Ngoc Quang, Choo Yi Ming, Khaled Al Sabbagh

# Assignment 3

## Basic Information

In our testing process, we used *JUnit 4.12* together with *Selenium 2.45* to test our SUT *Fossil 1.32.* To ensure that all of us have the same testing environment, the test is done on a new and empty repo that is created from scratch via the following command line commands,

- *fossil init lab*
- Description: This commands creates a new and empty repository into the current working directory and names it ‘lab’

- *fossil open lab*
- Description: This commands opens the repository ‘lab’

- *fossil ui*
- Description: This commands starts the UI server which runs on localhost (127.0.0.1:8080) and allows us to start using JUnit and Selenium for testing

## Scope

We are focusing on the key features of Fossil's UI, which is the Tickets and Wiki sections of the website.

The Fossil UI allows users to add tickets into the system, indicating whether there is a defect in the code, build problem, new feature request, etc. Users can further describe what issues it is they are experiencing and enter it into the system.

On the other hand, users can also view the tickets submitted by everyone else by generating a Ticket Report. When generating a report, one can specify what kind of tickets that he wants to see by using filter in the SQL query string.

Another key feature that we will be testing is the Wiki section of Fossil's UI. It allows users to create their own Wiki pages, to see a list of Wiki pages and to append comments to them. In our testing process, we will be testing these three simple test cases to ensure that all input is correctly saved and output is shown accurately.

## JUnit Test Cases

### 1. Adding new ticket

This test case aims to test the ticket adding function of Fossil. The script will first search for the HTML element named “Title” and input the the string “testTicket” as the ticket’s name using sendKeys(). Similarly, some comment is added. Then the script will call click() on the Preview and Submit buttons. Finally, the test will verify the result by checking if the title element with className “tktDspValue” has the ticket name: “testTicket”. It will also check the whether it has the comment.

### 2. Adding and Editing a ticket

The test will check if Fossil can perform editing an existing ticket successfullly. The script will first create one ticket named “Ticket to be edited” with some description in the comment. From the resulting screen, the script will call click() on the HTML element with linkText “Edit”. Then, it will change the name of the ticket to “Ticket has been edited” by calling sendKeys(). Next, it will call click() on the “View” and then the “Submit” buttons. Finally, the script will check the title of the edited ticket to see if it is “Ticket has been edited”.

### 3. Creating ticket report format

The test case will check whether a ticket report is created successfully. First, the script will call the url “http://127.0.0.1:8080/rptnew” to the new page to create report. Then, it will input the report name via sendKeys(). After submitting by calling submit() on the “Apply Change” button, the script will verify the current URL by checking to see if a part of the URL contains the string “rptview”, which is the resulting URL after creating a report.

### 4. Adding a duplicate ticket report name

The test will check what happens if user adds another report of the same name as one of the existing reports. To prepare for this test, we create one report called “All tickets report” and then we let the script create another report of the same name and verify the feedback provided by Fossil. After attempting to create this new report, the script will obtain the text contained in the HTML element of class “reportError” and verify that the text is "There is already another report named "All tickets report”".

### 5. Adding and Deleting a ticket report format

This test case verifies the report-deleting function of Fossil. The script will first create a report named “Report to be deleted”. After creating, it will obtain the report ID from the resulting URL. The ID is then used in the next URL call “http://127.0.0.1:8080/rptedit?rn=XX” where XX is the ID. Then the script will call click() on the “Delete this report” button and the on the same name “Delete this report” button on the next page. After that, the website is expected to go back to homepage. Thus, the script will verify the final URL to see if it contains the substring “reportlist” which is expected to appear in the URL of the homepage.

### 6. Adding new Wiki

This test case is intended to test the creation of wiki, whether it  is implemented correctly. It compares both the title and the content of the created wiki with some static input strings. The name of this Test script  in JUnit is CreateWiki, it instantiates an object of a Class named “Tracer” and uses a method that sets the web driver attributes.

### 7. List of All Wiki Pages

This test case compares a set of predefined strings, which represent the names of all the previously created wikis in the repository with the ones displayed on the wiki page. If a match between both arrays is identified then the test passes. The name of this test script in JUnit is ListAllWiki().

### 8. Append To Wiki

This test case will execute an append operation performed to a wiki. A wiki was created and then a comment was appended. This test case checks on on whether the appended comment has been bound correctly to the designated wiki. The name of this test script in JUnit is “AppendToWiki”.