CSE4940 CSE Design Laboratory II

Prototyping and Testing

DUE DATES: Week 14

CSE4940 will have different types of possible testing depending on your project and the presentation of your project on SDP day on the last day of the semester. Throughout your prototyping during the semester, each increment must be accompanied by a focused testing regime that occurs at varying levels of granularity, spanning white box testing of the code (method level testing) to general performance testing that varies based on the type of application and its specific needs to evaluating software qualities. There will be five testing exercises this semester, corresponding to each prototype increment, with the intent that each team member takes the lead in learning about one of the testing techniques so that the entire team is able to conduct sufficient testing using that technique for each increment. There is a PPT on the web page from the software engineering course on testing, if you need to review the various concepts (http://www.engr.uconn.edu/~steve/Cse230/finchapter6.ppt).

Unit and white-box testing; testing in the small

Module testing or testing in the small using white-box testing is the first approach to take when focused on the functional and correctness of the code, typically emphasizing the testing of public, protected, and private methods of classes. In the Java world, unit testing is supported by many products, such as junit (http://junit.sourceforge.net/) and there are many good tutorials available for its usage: http://www.vogella.com/articles/JUnit/article.html and also http://www.tutorialspoint.com/junit/index.htm. For objective-C, there are many tools such as OCUnit (http://cocoadev.com/wiki/OCUnit) and GHUnit (http://gabriel.github.com/gh-unit/), and there are many tutorials on line for you to review. For C, there is Check for C (http://check.sourceforge.net/), C/C++test http://www.parasoft.com/jsp/products/cpptest.jsp, and googletest for C++ applications. In the Microsoft world, MSTest can be utilized with Visual Studio (http://msdn.microsoft.com/en-us/library/ms182489%28v=vs.80%29.aspx). For mobile applications in general, the appcelerator (Titanium) recommends jsunity (https://github.com/atesgoral/jsunity), s Universal framework for Javascript. Qunit is another Javascript unit testing framework (http://qunitjs.com/). If you are using a gaming framework, it may have its own unit testing support. For example, http://www.prosoxi.com/2011/11/26/unit-testing-with-cocos2d-and-xcode-4/ can be used for unit testing with Cocos2d and Xcode (https://developer.apple.com/xcode/). For Unity3D, one testing source is (http://blogs.unity3d.com/2012/05/08/testing-unity/). For Senchta touch, see (http://www.sencha.com/blog/ui-testing-a-sencha-app/). For Django, see: https://docs.djangoproject.com/en/dev/topics/testing/?from=olddocs, and html5 see: the jasmine product (https://github.com/pivotal/jasmine). In summary, each team must choose the testing approach most appropriate for their project.
**User Interface Testing**

By this point in the semester, there is an expectation that your graphical user interfaces (GUI's) for your project should be near completion, and you should be able to plan and execute extensive tests of the business or program function and logic. For example, only allowing correct inputs into fields, checking formats of dates or phone numbers, making sure minimums are entered for a particular screen or stage of the software before allowed to go forward, etc. Or for games, checking that all of the buttons (on touch screen) and/or options work correctly; you can even check to be sure there is consistency in placement across multiple screens. For web or mobile apps, checking individual screens, order of screens, minimums need to be able to complete a screen to submit or go to the next screen, are all reasonable approaches. Each team needs to make a decision in regards to the focus of GUI testing that is most relevant for the project. You can even use a scenario based approach that enumerates possibilities that can occur on a given screen that would cause some action at the server or database. But the key focus is on insuring correct input. As examples, the course web page has three files: TestScenario.pdf (Scenario link) that contains a GUI screen and an elaborate set of tests to be conducted; WebTesting.pdf (Web App link) that contains testing results for web app; and, LicenseTesting.pdf (Java App link) that contains testing results for Java desktop application. The first file eliminates a scenario to test for in a Java desktop application; you can see that the testers (domain users) are attempting to enumerate all of the possible ways that the screen can be used for processing (business logic) and trying cases that should succeed, as well as one that should fail. These results as well as the information on who tested what is carefully tracked. In the other two files, you can see some of the results of the testing, where the end users (tester) takes screen shots and annotates each screen with the appropriate changes and/or corrections. To collect, track, and annotate your testing, you can use a screen capture application (e.g., [http://camstudio.org/](http://camstudio.org/) or [http://www.wisdom-soft.com/products/screenhunter_free.htm](http://www.wisdom-soft.com/products/screenhunter_free.htm)).

**General Performance Testing**

General performance testing has a wide range of possibilities that are chosen based on your application. For example, web and mobile applications often to what is called stress testing where a person is testing a particular application to see at what points the application breaks, to determine the amount of simultaneous users, to assess performance as users increase, etc. One such tool to utilizes is jmeter ([http://jmeter.apache.org](http://jmeter.apache.org)) which can send multiple http requests to test traffic capacity on a server. Stress testing can also be used to try to inundate the web server, application server, or database server. Another product has different levels of we testing for loads, and can record and replay tests ([http://www.webperformance.com/](http://www.webperformance.com/)). For teams doing interactive multi-player applications games, testing on different loading in terms of user, size of maps, etc., are all relevant to understand essentially under what conditions a system may fail or performance starts to degrade. A third type of testing may be if there is some aspect of your system that requires a high amount of CPU cycles and/or memory to operate – perhaps there is a complex algorithm. In this case, you can put checkpoints (calls to the system clock) before and after a piece of code to record timing based on different loads, and record these time and events in the actual code. Another type of testing can be database related, and there is analog to JUing called DBUnit ([http://www.dbunit.org/](http://www.dbunit.org/)) to search large repositories. Also – for databases, you have to worry about preventing (at the GUI or web service or API level) users
that input SQL commands that could be inadvertently applied to the database to return information – this is called SQL injection testing. There is SQL injection testing for web apps operating in a browser (like Firefox – see https://addons.mozilla.org/en-us/firefox/addon/sql-inject-me/). The basic idea is to ensure that potential malicious users are not attempting to input SQL commands into a data entry field (e.g., last name, address, etc.) in order to attempt to access the database directly.

**Software Quality Attainment**

Recall that from CSE4939W, there was a software quality assurance project involving the qualities that are non-function in nature such as: understandability, maintainability, evolvability, and user friendliness. What is interesting about this type of testing is for you and your team to try to ascertain the way that these capabilities can be measured in some way. Is your system easy to understand from both the developer and end user perspectives? How can you demonstrate its maintainability? Was there a recent problem that you identified and fixed, and if so, was it easy or hard? Can you make changes easily? Was there an evolution you had to do in the last prototype that wasn’t anticipated, and if so, was it difficulty or easy? How user friendly is your system? Did you get any feedback from the black box testing? There are a number of sources on-line that you can consider: http://www.softwaregatest.com/index.html and http://www.aptest.com/resources.html. It is your responsibility to explore SW quality attainment and the degree that your final project is able to do so.