Software Testing, Quality Assurance and Maintenance Winter 2019

Lecture 5 — January 14/16, 2019

Patrick Lam *version 2*

Last time. We spent most of the time talking about the distinction between static and dynamic analysis of code. Static analysis finds all the things (perhaps too many), while dynamic analysis tells you exactly what happens on a particular execution (but you have to have the right inputs). We also looked at a JUnit test case (setup/teardown/establishing state/checking system under test actions); the test case we looked at was at `https://github.com/google/guava/blob/master/guava-tests/test/com/google/common/io/MoreFilesTest.java`. Finally, we introduced the notion of coverage and test requirements a bit more formally.

Exploratory Testing

- poke around a system and find bugs
- you might do hallway usability testing

Exploratory testing is usually (but not always) carried out by dedicated testers. In that sense, it's somewhat different from the other testing activities in this course, which are more developer-focussed—our usual goal is learning, as developers, how to deploy better automated test suites for our software. Hallway usability testing, though, is an application of exploratory testing. Furthermore, the dedicated QA function is important, and we should learn about how it works.

Resources. James Bach has a shorter and a longer introduction to exploratory testing:

- `http://www.satisfice.com/articles/what_is_et.shtml`
- `http://www.satisfice.com/articles/et-article.pdf`

There is an exhaustive set of notes on exploratory testing by Cem Kaner:

- `http://www.kaner.com/pdfs/QAIExploring.pdf`

> "Exploratory testing is simultaneous learning, test design, and test execution."

Contrast this to scripted testing: test design happens ahead of time and then test execution happens (repeatedly) throughout the product's development cycle. When we think of dedicated QA teams, we think they are manually executing scripted tests. In 2019, that is not an effective use of staff.

There is a continuum between scripted testing and exploratory testing. Good exploratory testing may use prepared scripts for certain tasks.

Scenarios where Exploratory Testing Excels. (from Bach's article)

- providing rapid feedback on new product/feature;
- learning product quickly;
- diversifying testing beyond scripts;
- finding single most important bug in shortest time;
- independent investigation of another tester's work;
- investigating and isolating a particular defect;
- investigate status of a particular risk to evaluate need for scripted tests.

- not yet clear what right answer is
- don't yet understand the system
- evaluate criteria which are hard to quantify
  e.g. usability

When do you want more scripts/automation?
- run the same test over and over and over
- lack of regressions (idk what this means?)
- as you better understand the system

Exploratory Testing Process. Exploratory testing should not be randomly bumbling around (we can call that "ad hoc testing")—the random approach finds bugs but isn't the most efficient at giving you an idea of how well the software works.

- Start with a charter for your testing activity, e.g. "Explore and analyze the product elements of the software." These charters should be somewhat ambiguous.
- Decide what area of the software to test.
- Design a test (informally).
- Execute the test; log bugs.
- Repeat.

Exploratory testing shouldn't produce an exhaustive set of notes. Good testers will be able to reproduce the bugs that they encounter during their testing from brief notes. Taking full notes takes too long.

The output from exploratory testing is at least a set of bug reports. It may also include test notes, which include overall impressions and a summary of the test strategy/thought process. Artifacts such as test data or test materials are also both inputs and outputs from exploratory testing.

Primary vs contributing tasks. One way to classify tasks that software can do (or, in other words, its features) is *primary* vs *contributing*. A *primary* task is core functionality of the system; it's something that you would say "You Had One Job!" about. As examples, text editors must be able to load text files, add text, and save the text files. On the other hand, *contributing* tasks are secondary. A macro system for a text editor would be a contributing task. Being able to read email in your editor is definitely a contributing task. Sometimes it's not black-and-white. Spell-check can go either way.

Example. I recommend reading the example by James Bach in "et-article.pdf" about the photo editing program ("ET in Action"). He describes how he used ET to evaluate software for Windows compatibility and found critical defects.

In-class exercise: Exploratory testing of WaterlooWorks.

We will try out exploratory testing with WaterlooWorks. I believe that everyone should have access to the system, although for some of you there may be no jobs visible right now.

The charter will be "Explore the overall functionality of WaterlooWorks". Summarize in one or two sentences what the purpose of WaterlooWorks is. Identify the tasks that WaterlooWorks should be able to do and classify them as primary or contributing. Identify areas of potential instability. Test each function and record results (bugs).

Of course, don't do things that have actual effects. Usually, testers would have access to a development server and could test those areas more aggressively. But we are working with production systems here.

The purpose of WaterlooWorks is to let employers meet & hire co-op students and to give a platform for students to find co-ops jobs. It also provides an algorithm for matching students to employers based off ranking each other.

Tasks for WaterlooWorks:
- search and filter jobs (primary)
- apply to jobs (primary)
- post jobs (primary)
- match willing co-ops to willing employers (primary)
- have a messaging platform (contributing)