TESTING AND TEST-DRIVEN DEVELOPMENT

CIS 440 · DR. JOSEPH CLARK

OBJECTIVES

- Answer: what is software quality? Differentiate validation and verification.
- Talk about ways to do validation testing.
- Talk about ways to do verification tests.
- Discuss test automation and traceability.
- Introduce TDD and BDD.
- Discuss James Bach's lecture on testing.
- Define your testing requirement.

WHAT IS QUALITY?

- Quality means "less waste":
  - less need to re-work or fix things because they are better made
  - less effort spent on re-working or fixing bugs because you capture them earlier
  - less need to take back products or refund customers

- Quality means "happy customers":
  - If a customer likes the product, he's less likely to ask for a refund.
  - He's more likely to refer you to other customers.
  - He's more likely to come back for more.

VERIFICATION = "did we build the system right?"

VALIDATION = "did we build the right system?"

TESTS FOR VERIFICATION OF SOFTWARE QUALITY

- **System Tests** (aka integration tests, functional tests, or "black box" tests) test functionality of the system from the user's perspective: does it do what it's supposed to do?
  - Are the "user stories" implemented?
  - Do the inputs result in the expected outputs?
  - Browser automation and other "macro" tools can be used to automate.

- **Unit tests** (aka structural tests or "white box" tests) test individual pieces of code. You might test each function, or object, or SQL query, as you are coding it.
  - Does the function give the expected result?
  - Does the query run in an acceptable amount of time?
  - Can be done as "asserts": see James Coplien's article "Most Unit Testing is Waste" : http://www.rbcs-us.com/documents/Why-Most-Unit-Testing-is-Waste.pdf
  - The video for next Tuesday gives an example in Python
  - Most languages have a unit test library: JUnit for Java, unittest for Python, NUnit for .NET, etc. Also look for Visual Studio functions.

TESTS FOR VALIDATION THAT THE SOFTWARE MEETS USERS' NEEDS

- 29th Drive had a 3-part method, which I failed to write down, but it was something like this:
  - Build empathy for the end user.
  - Refine and explore ideas.
  - Validate and get feedback.
- Big idea: if you want to know if your product serves the users, you need to (a) know who the users are, and (b) get them to see, use, and give feedback on the product or prototype.

- Tools:
  - Pen sketching and paper prototypes – iterate over lots of designs quickly
  - Software versions of this: Balsamiq, Justinmind, etc. But they're slower than pen sketches.
  - **Verifyapp.com** – test screenshots/sketches to see if users understand them; where they click on the screen, etc.
  - **Usertesting.com** – crowdsourced usability testing; get users to play with your app and video themselves trying it out
  - Traditional research methods: surveys, focus groups
  - Observe user behavior: Which pages do they visit? How willing are they to pay, or to register?

TEST AUTOMATION

- Testing needs a mix of automation and human testing.
- Automation serves to reduce manual labor. If a tester has to repeat simple steps many times, he either won't do it, or he won't do it well.
  - Imagine testing a 10-page online questionnaire, where you have to fill out all 10 pages just to test a change to the 10th page.
  - Be wary of metrics like "coverage" that give you false confidence in the results.

- Human testing (also called exploratory testing) should be used where repetition is not needed, or automation is difficult. Humans can test systems in a lot more ways, in a lot less time.
  - You can still write "scripts" to document your tests and make them repeatable.
  - Focus on areas where there are real questions to answer – if you know the test will always pass or always fail, there's no need to do it.

TDD AND BDD

- **TDD = Test Driven Development**
  - This is an "eXtreme Programming" (XP) practice
  - Simply: you write the test first, then write the code to pass it, then refactor if necessary to clean up the code
  - "Red – Green – Refactor"
  - The **tests become the specification** of the software.
  - This approach also prevents you from building a lot of stuff you don't need to build – just the minimum to make tests pass.

- **BDD = Behavior Driven Development**
  - Like TDD, but tests are written in plain language by the client or product owner, rather than by developers.
  - This is a great way to train your product owners to write better specifications! They will quickly learn if they're asking for things they don't really need, or failing to ask for things they ought to have asked for.
  - Cucumber (http://cukes.info) is a leading product in this area. Anyone want to try it on their project?

YOUR THOUGHTS ON THE BACH LECTURE?

- ???

YOUR TESTING ASSIGNMENT

- Beginning with release v0.4 and continuing to the end of the class, your project will include a testing component.

- From this point onward, the project repo should include a folder called `tests` and there should be some documentation in the README file about 'how to test' or **'how to run the tests'.**
  - Tests may be automated scripts or may be written documentation of manual test procedures.
  - These may be "validation" tests, i.e. user tests or A/B tests that test whether the product serves the customer's needs, or "verification" tests that make sure the software is of good quality, doesn't have bugs, etc.
  - Whatever type of testing you do, you must document how to carry out the tests.