IS2545: Lecture 24

—

Dustin Iser

Last lecture

Page Object Model

Software is built in an object oriented way, why don’t we build our tests in an object oriented way?

Advantages of POM

- Cleaner easier to understand tests.
- Object repository is independent of test cases.
- Less changes to make when refactoring tests.

Hallmarks of POM

- One page object per page in web application.
- Segmentation.
- Every method returns a page object.

Front end unit testing

In the unit testing lecture and deliverable we only built unit tests for back end applications, but most applications are made up of a back end and a front end.

It may seem like front end automation is only a task for automated UI tests such as the ones we built using Selenium, but this is not the case.

Mutation testing

When we talked about unit testing we briefly talked about unit test code coverage.

Unit test code coverage is a good tool for testers to find code they missed during unit testing, but it's not the best metric to use for measuring quality of unit tests.

- Defects are not distributed uniformly across a program.
- Risk is not distributed uniformly across a program.
- Just because code is executed doesn't mean it's well tested.

**Beautiful Testing*, "Seeding Bugs to Find Bugs: Beautiful Mutation Testing", Andreas Zeller & David Schuler

Pareto effect

In general, 80% of the effects come from 20% of the causes.

What this means for software developers:

- Lowell Arthur: “20% of the code has 80% of the errors. Find them, fix them!”
- Microsoft found that by fixing 20% of their most-reported bugs, 80% of the related errors and crashes in a given system would be eliminated.
- When load testing, it is common practice to assume 80% of the traffic occurs during 20% of the time period.

Retail side note:

80% of your sales will come from 20% of your customers.

Mutation testing

The practice of intentionally inserting defects into our application with the purpose of determining if our test suite finds them.

Mutation testing may be used to test unit, integration, automated UI, or even manual tests.

Steps:

1. Insert mutation.
2. Execute test.
3. Insure test fails.

Why am I just now hearing about this?

It’s hard to do because it takes time. Each mutation requires a build and test execution.

We need automation to perform mutation testing effectively.

- Manipulate binary code directly.
- Ignore non-covered code.

High quality mutations

Some mutations are better at creating defects than others. Studies show that a small set of mutation operators create the same number of defects as a larger set.

- Replace numerical constant
  - Replace a numerical constant x with x + 1, x -1, or 0.
- Negate jump condition
  - Replace a branch condition with its negation.
- Replace arithmetic operator
  - Replace an arithmetic operator with another one, e.g., + with - or * with /.
- Omit method call
  - Replace a method call with a constant.

Static Code Analysis

Some defects may be found without even executing the code. Tools that do this are called static code analyzers.

Examples:

- Compilers
- IDEs
- Linters
- 3rd party tools

https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis

Static Code Analysis

Static code analyzers can help you find issues like:

- Style inconsistencies
- Find duplicate code
- Detect unreachable code (‘dead’ code)
- Make recommendations that will make code simpler or easier to understand.
- Make recommendations that will make code perform better.
- Detect security defects.