Test Code Coverage

EECS 2311 - Software Development Project

Wednesday, February 5, 202020

YORK UNIVERSITÉ UNIVERSITY

When is testing done?

- Short answer: Never!
- A bit longer answer: When all features of the system have been tested with all possible inputs that could make a difference
- In practice, this is hard to determine
- Metrics such as code coverage can be used to give an idea of how sufficient the testing is

Statement Code Coverage

- Observe the system as it is running
- Keep track of how many of the statements in the code were executed at least once
- Divide by the total number of statements in the system
- A comprehensive test suite is important
- Typically, it is hard to get high coverage. Anything above 70% is pretty good for a large system

Problems with statement coverage

- A statement must be executed with different values for the relevant variables to be fully tested
- Loop bodies may need to be iterated many times to reveal issues
- Not all statements are equally important
- Only the true branch of an if statement may be executed but coverage may be 90% for the statement if the false branch is one tenth of the size

Other kinds of coverage

- Segment coverage
- Branch coverage
- Multi-condition coverage
- Dataflow coverage
- More in EECS 4313

Software engineering guideline

- Low code coverage indicates that more testing must be done
- High code coverage gives little information about the quality of the testing

Let's see a demo (EclEmma)…

GUI Testing

- In an interactive application, coverage can be low because the GUI code is not executed through the test cases
- Testing the GUI can be done by accessing the GUI components in your test cases and programmatically using them
- Ideally, one should search the component hierarchy for the appropriate component
  - Advanced. Will cover in EECS 4313
  - In this course, we will use a simpler method
    - See the example in the course github repository

Software deployment

- The system may be running fine under Eclipse, but the customer needs a standalone system
- The code must be delivered to the customer and assembled and configured at their site
- Any dependencies to other libraries must be transparent to the customer
- Deployment also includes maintenance, updating, and uninstalling

Build and Deployment Tools

- Current practice is to automate the process of building and deploying the system
- Tools such as Ant, Maven, or Gradle can be used to build a release
- Tools such as Jenkins, Travis CI, or Circle CI can be used to ensure continuous integration of all system resources
- After Reading Week, we will use Gradle and Circle CI to manage this aspect of our project.

Prototype submission

- Choose File → Export… → Runnable JAR File
- Click Next, and select a Run configuration that launches successfully
- Leave the option to extract libraries checked
- **Test** that the generated jar file can be launched in all platforms
- Upload the runnable jar file online and paste the URL in your PeerScholar submission
- **Test** that the URL works!

Lab Task

- Install EclEmma and calculate the coverage of your testing for the app you are developing
- Demonstrate your ability to calculate coverage in the lab
- Your midterm and final submission must discuss test code coverage (Testing document)

- In next week's lecture, each team will have a 5-minute slot to give a demo to the "customer"
  - Schedule will be announced on Moodle