Testing, CI/CD, and Coverage

44-382 Secure Programming

Testing Your Code

- It should go without saying that testing your code is important
  - How do we know it works?
- You should test for both known good behavior and known failure modes
  - Knowing how your code can break is just as important as knowing it behaves correctly
  - Identify your error states!
- You have previously been introduced to unit testing (if not, those slides are on the course website)

Kinds of Testing

- Unit testing
  - “Do my different parts work?”
- Integration testing
  - “Do I play well with others?”
- Performance testing
  - Let’s not worry about optimization, focus on meeting performance requirements not maximizing performance
  - Optimized code often means clever code; clever code often means broken code
- Smoke Testing
  - Basic functionality tests
  - “If I use the most basic functionality, do I let the magic smoke out?”

https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

Automate, Automate, Automate

- Frequently testing and QA is not given the attention it deserves
  - Discussion topic: Broken/incomplete software on release
  - Discussion topic: Why does this seem to be more prevalent now?
- “Testing is boring” ~me (probably), 2022
- Manual testing is time consuming
  - Consider: writing code, running it, providing input, debug, loop
- Once a part works, we may not revisit it
  - Regressions are a thing!
- One (partial) solution: automate tests to run as part of your development cycle

Continuous Integration/Continuous Delivery

- CI: Automating changes from multiple developers
  - Or, just you from multiple computers!
- CD: Automated deployment of live systems
  - Think things like GitHub; web apps, etc.
- These systems can be more than just automation of testing
  - An important part of their functionality

https://www.atlassian.com/continuous-delivery/continuous-integration

Some Automated Testing/CI/CD Tools

- GitHub Actions
  - Run predefined actions/test when you push to GitHub
- Jenkins
- Azure DevOps
- GitLab Ci/CD

Note that these all integrate with your version control system

Demo: Configuring GitHub Actions to Run Unit Tests (Gradle)

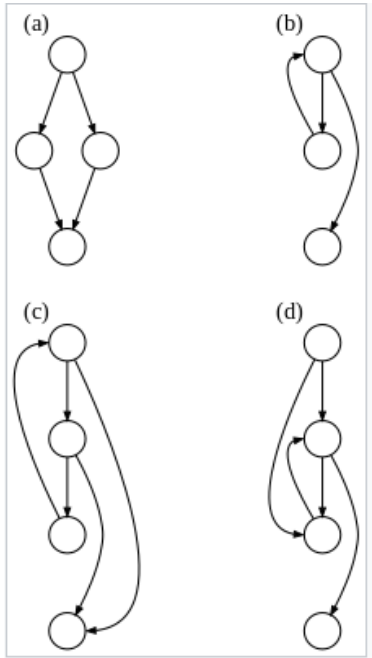
https://github.com/neloe/cicdcov-demo

Code Coverage

- We already know it’s infeasible to test all possible inputs and outputs
- Ideally we’d like our tests to cover all possibilities
  - We call this Code Coverage
- One of the easiest ways to calculate coverage is *Line Coverage*

Demo: Code Coverage (Netbeans, Probably using JCov?)

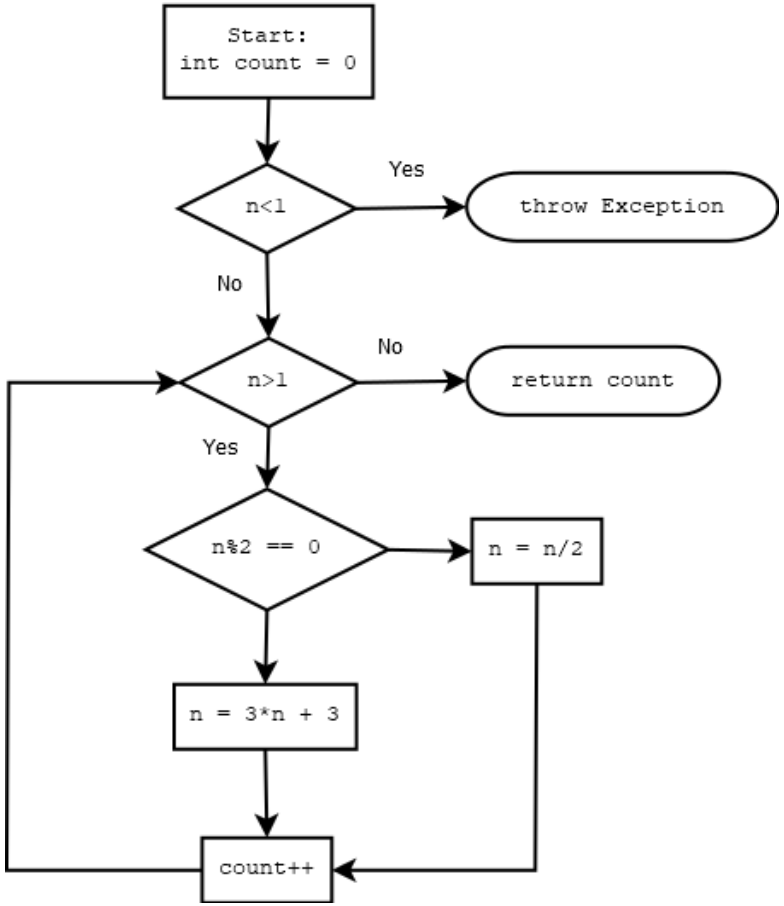
Is Line Coverage Good Enough?



Some CFG examples:
(a) an if-then-else
(b) a while loop
(c) a natural loop with two exits, e.g. while with an if...break in the middle; non-structured but reducible
(d) an irreducible CFG: a loop with two entry points, e.g. goto into a while or for loop

- Consider a *Control Flow Graph*
  - Graph notation representation of paths that control can take as your code runs
- Consider the graphs to the left; how many paths are possible to get from the top to the bottom?
- Determining the number of paths from the start to the end is… nontrivial
  - Easier for smaller graphs; small simple functions are easier to test!
  - Do we consider simple paths? Paths with loops? Reuse edges if we get to a node?
  - What if every path isn't possible?

How Many Paths Through The Graph?



JCov

- Part of OpenJDK
- Types of coverage:
  - Block
  - Line
  - Branch
  - Method
- Works with JUnit, etc

Coverage.py

- https://coverage.readthedocs.io/en/6.4.4/
- Handles
  - Line
  - Branch
- Python has a built in unittest module; plays well with that

GCov

- Coverage tool for use with the GNU Compiler Collection
- Profiles your executable, so use whatever unit test framework you want
- Analyzes line coverage
- Can tell you the number of times a line of code was executed
  - Use with gprof to fine tune performance if you really want to.
- ONLY works with code compiled with a GNU Compiler
  - No clang, sadly