Lecture Notes
September 14, 2017

Static analysis individual/pair assignment due tonight

First team assignment (team formation) due next Tuesday September 19

continuation of Bug Finders

Bug finders (or bug checkers) are sophisticated tools that look for generic bugs as well as code smells

- They do not need to know anything about the intended functionality (features) and in most cases do not require any “specification” for the software

- Sometimes encode domain-specific rules, e.g., web client/server messaging formats and protocols, or knowledge of error specifications, e.g., library functions return 0 on success and a negative integer on failure

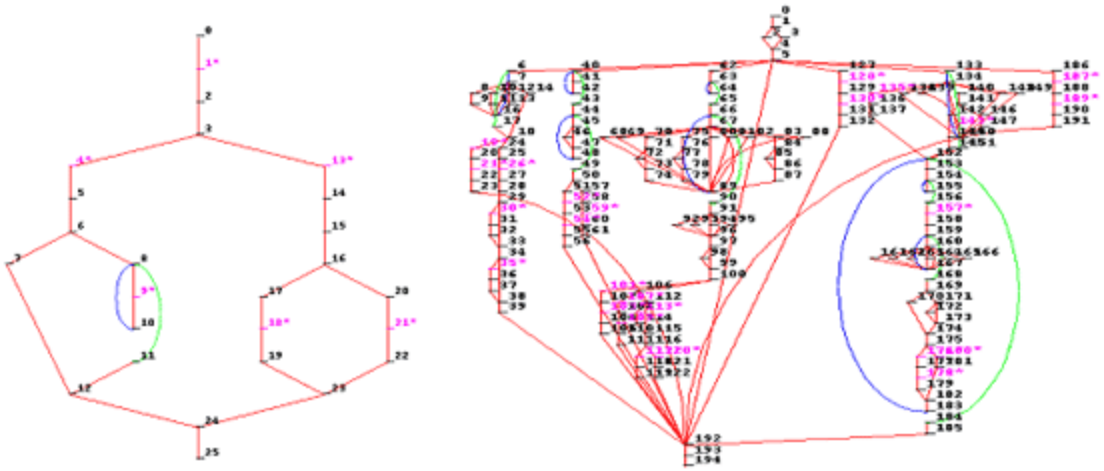
FindBugs for Java

Examples of what bug checkers can detect:

Secret keys, tokens or passwords embedded in code

Using third-party libraries with known security vulnerabilities

Too many paths in a given function or method (cyclomatic complexity)



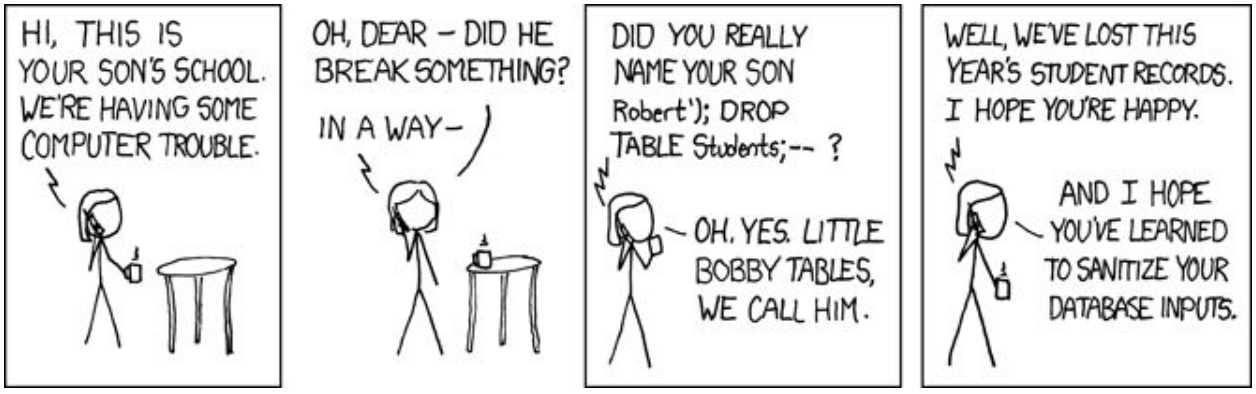
Simple Code vs. Complex Code

Code does not handle every error code that can be returned from an API function or system call

System Error Codes: 1 to 15841

Dereferencing a null pointer: **You're dereferencing a null pointer!**

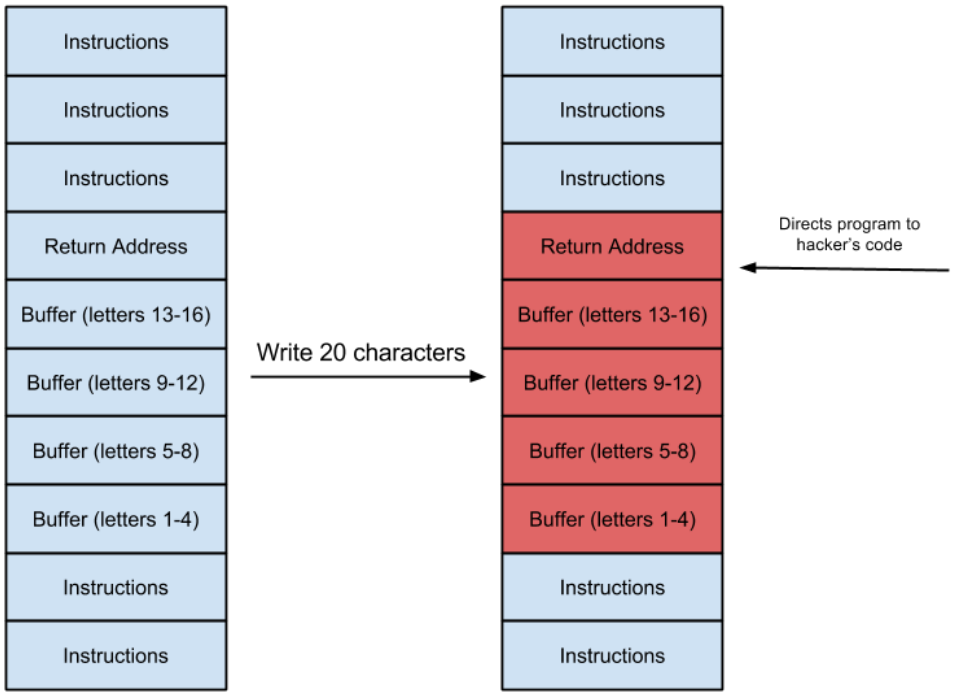
Input from user passed unsanitized to framework/database code - sanitize

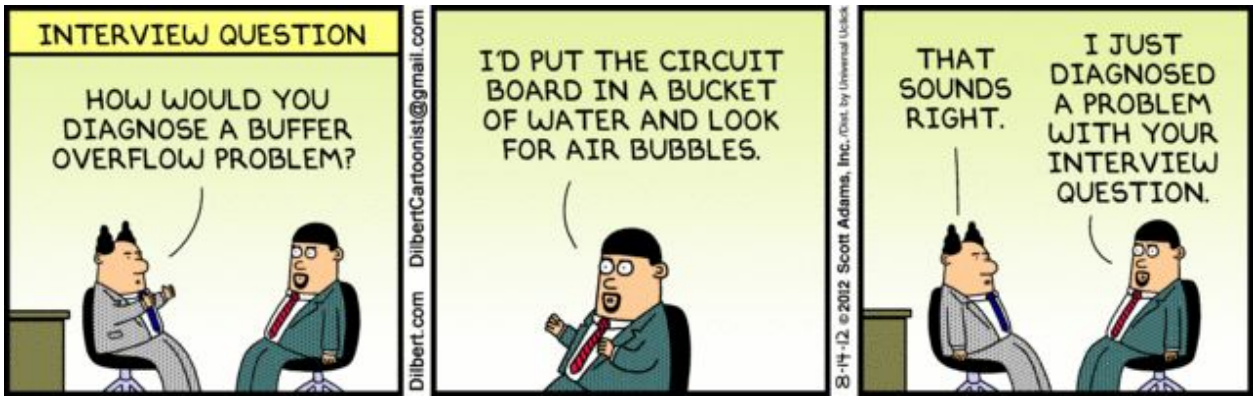


In a non-memory managed language, checks every path from memory allocation to make sure there's corresponding memory deallocation - and that there are no further uses of the memory after it has been deallocated ("use after free")

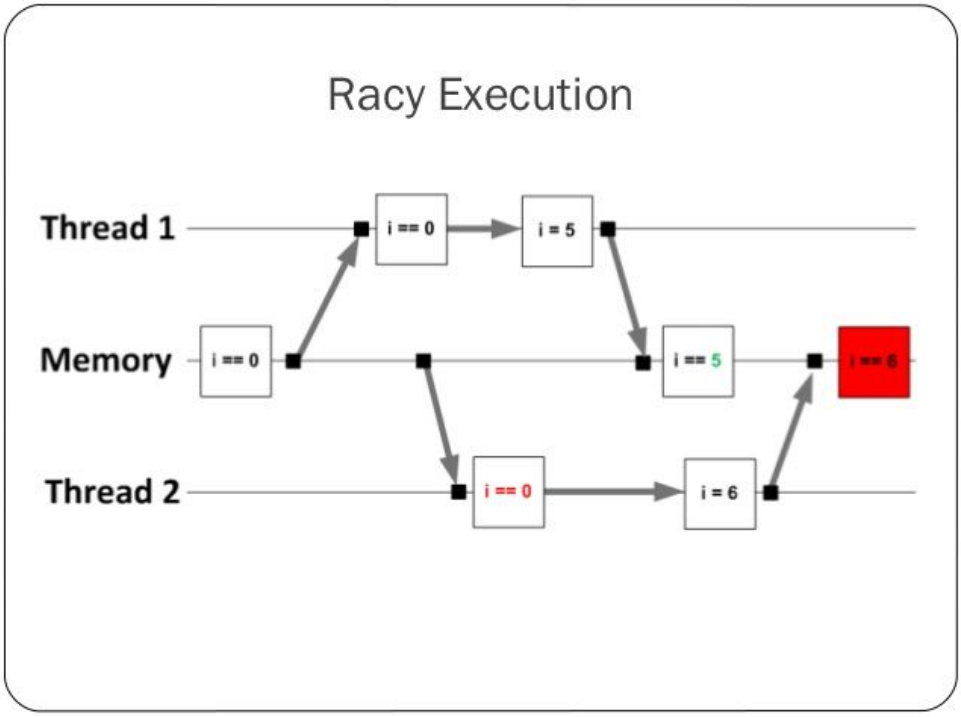
CWE-416: Use After Free

Also looks for checks/limits on array bounds, to prevent buffer overflow





In multi-threaded code, check that data dependencies cannot race and locks are matched with unlocks along every execution path from the lock

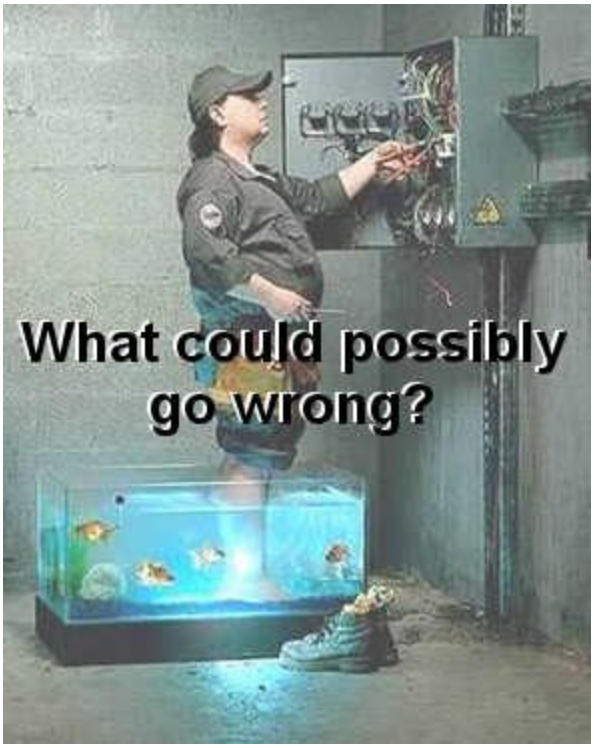


Static analysis bug checkers typically produce many *false positives*, e.g., a missing free or unlock on an infeasible execution path, so various kinds of error/warning messages can be suppressed

Static analysis bug checkers cannot avoid *false negatives*, they cannot find “all” bugs and cannot “prove” that no bug exists

Static analyzers do not replace testing - static analysis finds bugs that *might* happen with some input, dynamic analysis finds bugs that *did* happen with specific input but cannot consider all possible inputs

You do not need to understand how they work internally in order to use static analysis tools



Package Managers and Dependencies

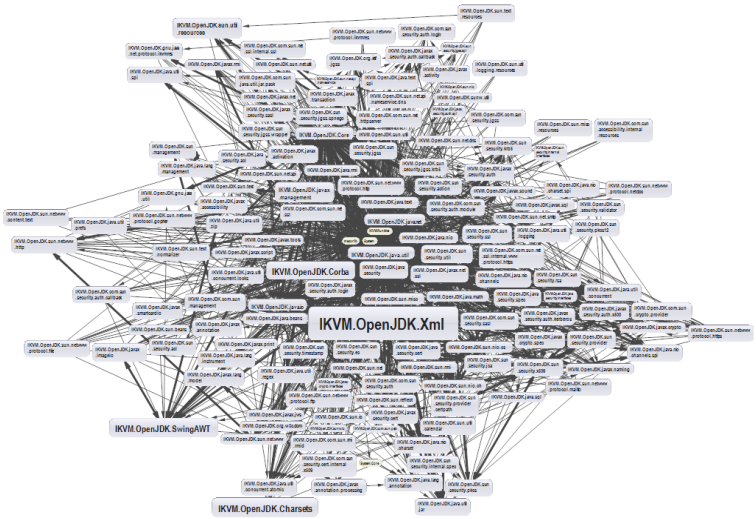
When you organize your own codebase into multiple code units - classes, modules, packages, libraries, etc. - then some of your code units are dependent on your other code units

When you reuse code units or other resources written by others, your code is “dependent” on those resources

If you write an application that requires authentication and authorization (e.g., login userid and password, maybe 2-factor), instead of implementing this capability yourself, you should use a package that some expert has written. Then your program depends on that package.

Dependencies may be *transitive*, A depends on B depends on C depends on D.

Highly interconnected dependencies and/or reliance on specific versions (“dependency hell”, or “DLL hell” on Windows) is **bad**. Why?



Sometimes packages consist of data resources rather than code

In order for your software to run, the compiler or runtime environment needs to know what all the dependencies are and where to find them on the computer

Package managers keep track of the third-party code units (typically in the form of libraries) that your application uses, beyond those that “come with” the language platform

They utilize the “metadata” associated with each package - the software's name, description of its purpose, version number, vendor, checksum, and a list of dependencies

Most of them work like a mobile app store:

- Download third-party libraries and keep up to date
- Based on ecosystem with central repository
- Ensures available in right spot on your file system
- Can also remove and clean up

Hopefully also ensure integrity and authenticity of by verifying digital certificates and checksums

Eliminate the need for manual installs and updates

Examples: npm, pip, also see Which programming language has the best package manager?

From sample project
https://github.com/COMSW4156Fall2017/coms4156_jumpstart/wiki/Installing-ImHere

The term “dependency” is also used in software engineering to refer to *coupling* - the degree to which your code unit depends on the “innards” of another code unit, so that if the other code unit changes you also need to change your own code. This is always **bad**. Why?

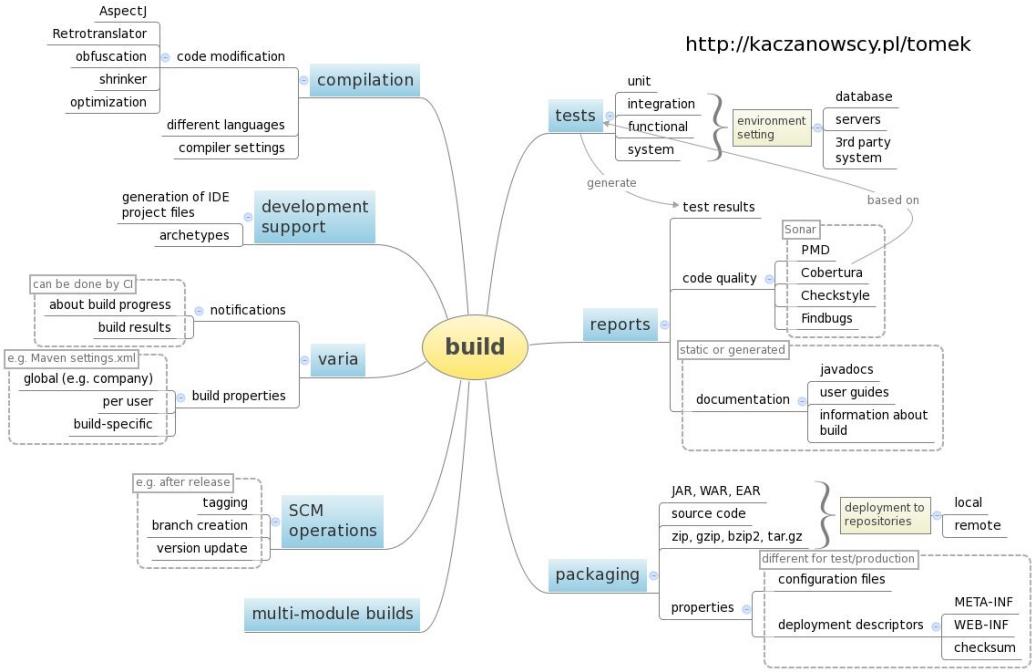
“Dependency” may also mean a relationship among work tasks, such that you cannot do one task until another task has been completed. Ideally, dependencies among tasks by different software developers should be minimized. This will be considered later in the context of iterations and iteration planning.

There are also control flow and data flow “dependencies” between program elements (e.g., statements, expressions, variables) within a single code unit, this will be discussed later in the context of testing and testing coverage.

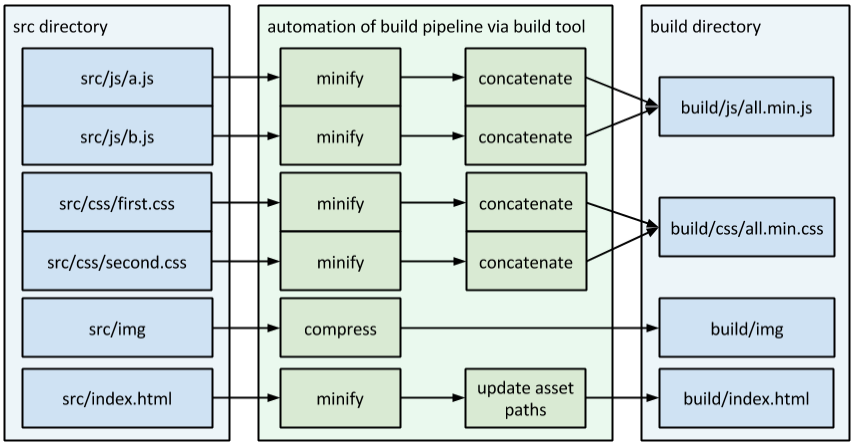
Build Tools

Initially the “make” tool for compile and link of C and other 1970s Unix languages

But now build tools automate lots of things even for non-compiled languages, e.g., runs the package manager to update all external dependencies and/or packages up your program to be installable from a package manager (some build tools also operate as package managers)



Repeatable, reproducible, standard sequence of steps so developers do not have to remember the steps and everyone uses exactly the same steps



Build tools provide some notation for writing down the steps

Example Maven configuration:
https://github.com/Programming-Systems-Lab/phosphor/blob/master/Phosphor/pom.xml
(notice this uses make internally)

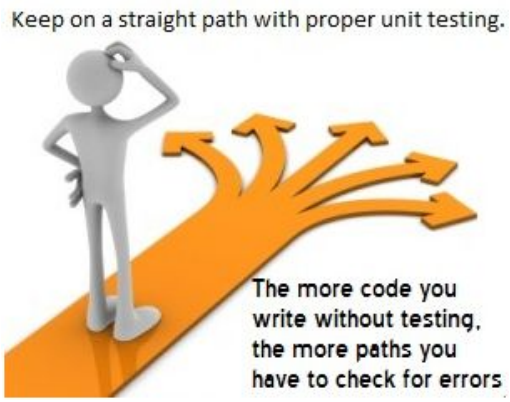
When I talk about “build”, I usually mean press one button / type one command to do whatever is necessary to make your program executable, then run your full set of test cases and other bug finders

Examples: WebPack, PyBuilder, Gradle, Maven
see also Java Build Tools: Ant vs Maven vs Gradle

Unit Testing Tools

Unit testing tools *execute* the test cases you write, they do not write the test cases for you (although some may generate test case skeletons or templates to fill in)

See Top 10 Excuses Programmers Gives to Avoid Unit Testing



What is a unit? Method (or function, procedure, subroutine) or Class (or module, file)

What is a test case?

- Identified by naming convention or annotation
- Usually maintained in a parallel directory/folder hierarchy to corresponding source code
- Constructs inputs for a method
- Invokes that method with the input parameters
- Supplies non-parameter inputs, e.g., inputs from *simulated* human user, file, network, database, device
- Collects the results of that method
- A set of assertions check that the results are as expected, there may be a special notation for writing assertions or this may be regular code (which could itself have bugs)
- Assumes “test oracle” that can always tell whether or not the actual output is correct for the given input

Unit tests aim to test a method or class *in isolation from* the rest of the program

- Called by a test runner or test harness rather than by code in the rest of the program
- Calls stubs (state verification) or mocks (behavior verification) rather than calling code from the rest of the program

See Difference between Stub and Mock object in Java Unit testing

Unit tests, or a unit test class (set of unit tests), associated with “setup” and “teardown” code

- Setup invoked before unit test(s) to set up state of the world that simulates the state the program would be in when the method(s) invoked during full system execution
- Teardown cleans up the state so that there are no externally visible side-effects
- Some resources are very expensive to setup/teardown, so should organize test cases that need the same resource state into the same test class

Test classes should be *independent*, so can run in any order to get same results - one set of tests should not rely on some other set of tests running beforehand (setup should assume pristine state and teardown should return to pristine state)

A “flaky test” is a test case that produces different results on different test suite executions *even when nothing changed*. What could cause this?

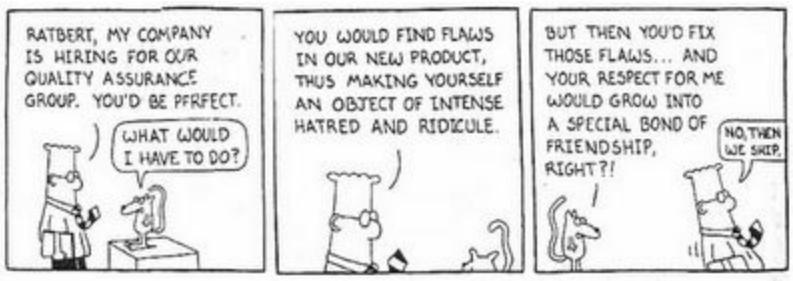
https://uploads.toptal.io/blog/image/91302/toptal-blog-image-1434578005589-4e6897ec04cc0b3c7075b9b011ee915c.gif

Examples: Mocha, pytest, JUnit

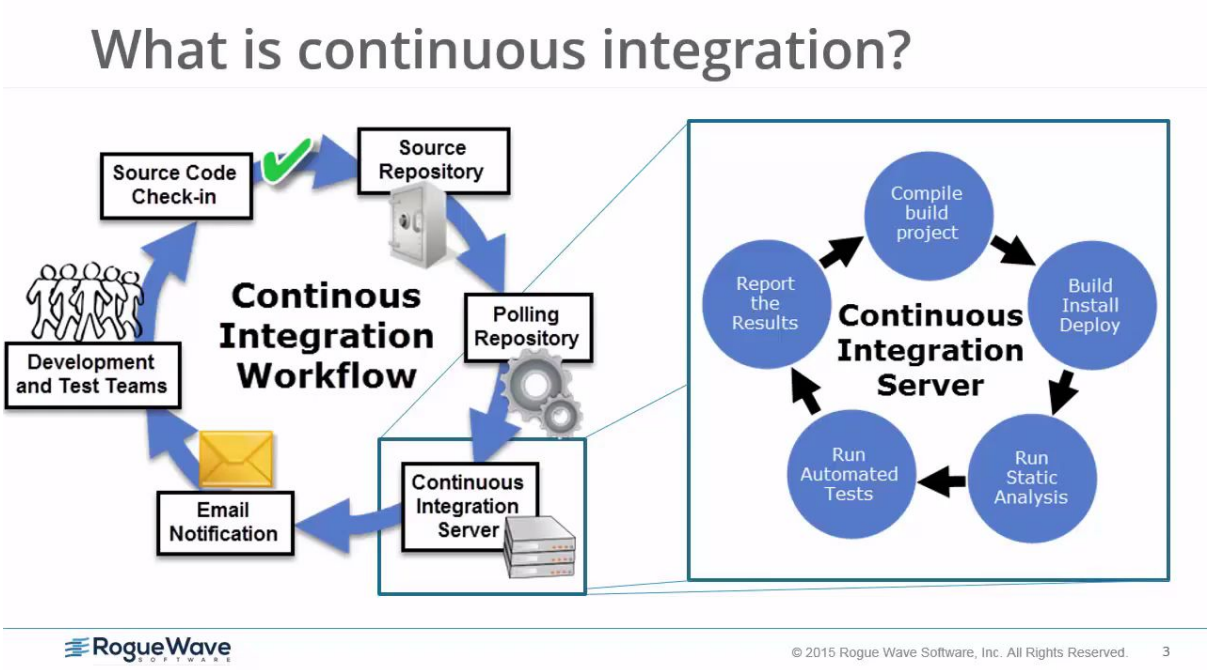
Most unit testing tools work for testing full applications that can run from the command line, not just “units”, but for GUI applications you need something that simulates human user interactions

Examples: Selenium, Appium

How to design your test suite, how to know when you’re done testing, what to do if you do not have a good test oracle, etc. will be covered in depth later in course



Continuous Integration and Continuous Deployment



Integrates version control with “build” (may also integrate with IDE)

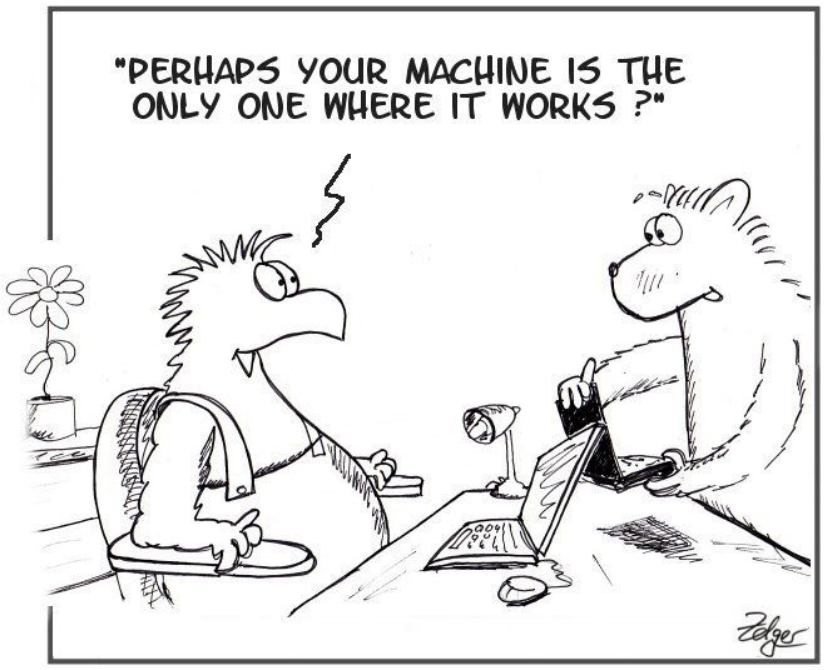
Does a full build every time anything is checked into the version control system (or at pre-designated times, such as 12midnight every day)

Pre-commit does a local build when the commit command is run, and doesn't actually do the commit if the build doesn't pass

From sample project:
https://github.com/COMSW4156Fall2017/coms4156_jumpstart/blob/master/bin/git-hooks/pre-commit

Post-commit does a remote build on the repository server (or some other designated server) after the commit finishes

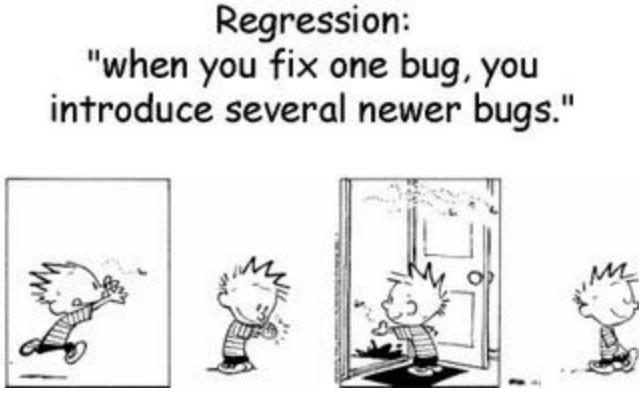
"Who broke the build?" - If the post-commit doesn't "pass", in most cases someone didn't do a pre-commit build OR there is some mismatch between the environments for two or more developers



It works on my machine

For builds before/after every commit, typically only a subset of the full regression test suite is run because many full tests suites take hours to run - thus nightly builds

What is a "regression"? This is when fixing a bug or adding a new feature causes previously working code to break (or regress), sometimes code that seems completely unrelated



To detect regression, need to re-run *all* test cases that previously passed (and any tests that previously failed, in case they now fail in different ways or unexpectedly pass)

Continuous Deployment adds automatic deployment to continuous integration

Installs on the production server(s) and/or produces a patch to push to customers

Examples: Jenkins, Travis CI