COMS W4156 Advanced Software Engineering (ASE)

November 16, 2021

shared google doc for discussion during class

First Assessment

Average: 84.1, Median: 88, Stdev: 13.4

You were graded by your team's IA mentor, contact them about grading concerns. Do not contact me about regrading unless your IA mentor tells you to do so - and then use piazza, not email!

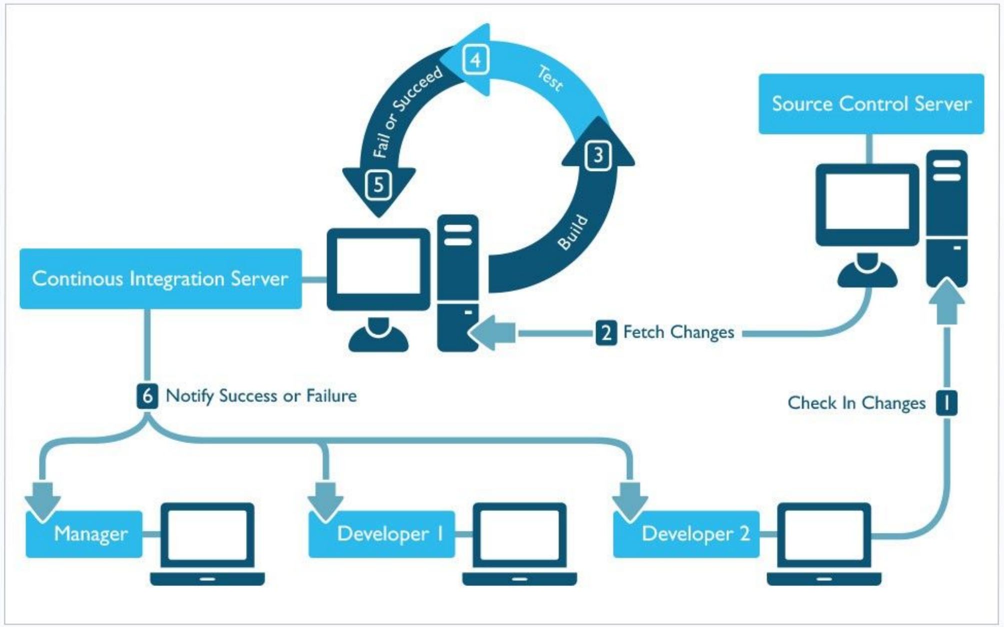
A few students did not follow instructions for part 0A, so their grades were set to 0 until they submit a proper response to part 0A - please attach to a comment in courseworks. The same kind of question will be asked in the second assessment, and you will <u>not</u> get a chance then to add a new answer, you need to do it right the first time. (The stats above use those students' rubric scores, not 0, everyone fixed it.)

Continuous Integration (CI)

Tools like Travis CI and Circle CI hook into a version control system and re-build after every commit to the shared repository

Can also run pre-commit (partial build) and/or on a timer (e.g., nightly full build)

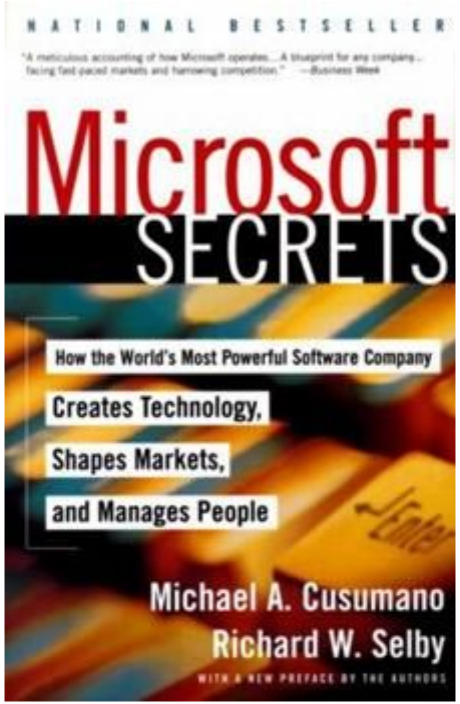
Goal is to detect errors as soon as they are introduced



Don’t Break The Build

Errors detected during continuous integration were traced to *your* commit

If all developers had fully tested their changes locally (on their own machine) prior to commit, and fixed any errors prior to commit, in theory CI should not find any errors


NATIONAL BESTSELLER

"A meticulous accounting of how Microsoft operates....A blueprint for any company facing fast-paced markets and harrowing competition." —*Business Week*

Microsoft SECRETS

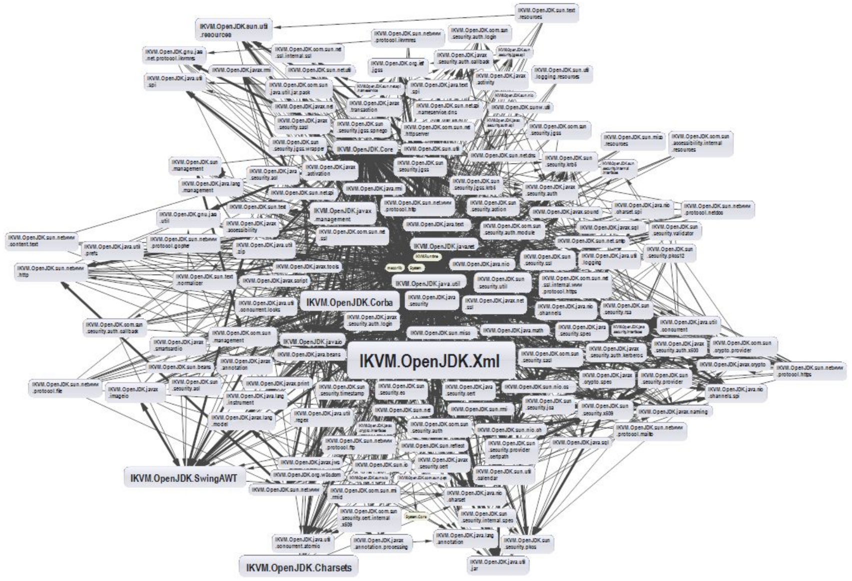
How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People

Michael A. Cusumano
Richard W. Selby

WITH A NEW PREFACE BY THE AUTHORS

I Thought Build Meant Compile

Dependencies:



Build often refers to gather all dependencies, then compile and link (compiled languages) or just gather all dependencies (non-compiled languages)

In CI context, “build” also runs testing and analysis tools

Which test cases actually run on commit may vary - in some organizations CI runs only selected tests

Test Selection

A common choice for commit testing is to use a “test minimization” strategy that selects only new/modified test cases plus existing test cases whose results (pass or fail) could have been affected by the committed code changes (change analysis)

This may be a relatively small subset of all the tests, but there’s a cost to accurately computing the affected tests without actually running any tests

Change Analysis

Track all code c exercised by each test t during *previous* testing

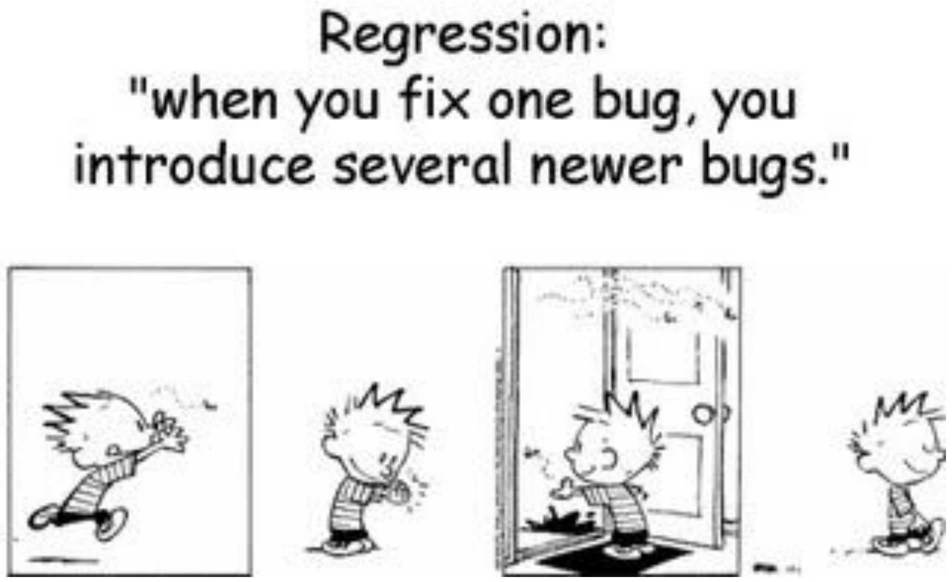
If the commit changes c, or changes any other code that c depends on directly or indirectly (determined by static analysis), then re-run t

It may be less expensive to run full regression testing than to run full static analysis

Regression Testing

Regression testing is the typical choice for nightly builds

A *regression* is when fixing a bug or adding a new feature causes previously working code to break, or “regress”, sometimes code that seems completely unrelated in another part of the codebase



Detecting Regressions

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

Of course, if the code change fixed a bug, you would expect the corresponding test case(s) that detected the bug to now pass. The developers should test this before commiting

But other test cases that do not seem relevant may also now pass due to dependencies in the application (incomplete or incorrect change analysis) or in the test cases themselves

Some Tests Are "Flaky"

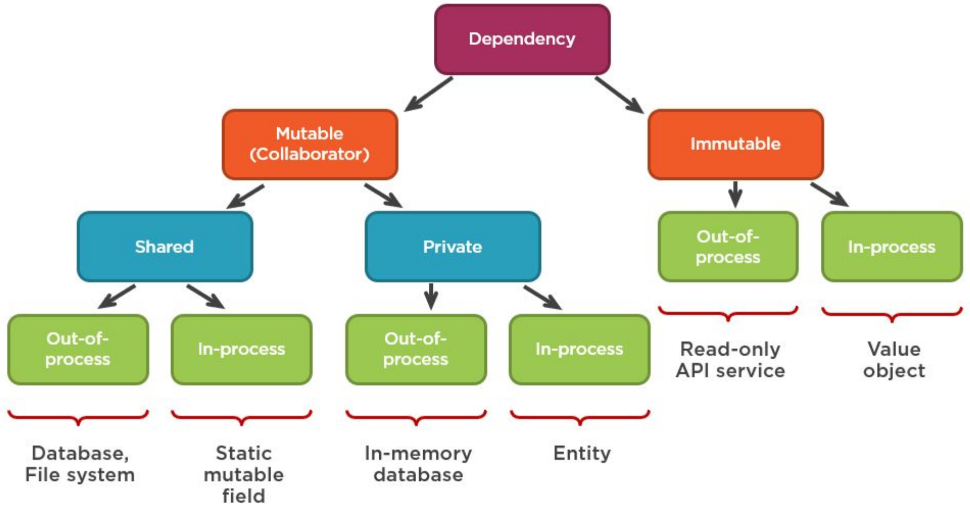
A flaky test is a test case that produces different results (pass or fail) on different test suite executions *even when no application code has changed*

This can happen due to hidden dependencies among test cases, reliance on external resources (e.g., timeouts, non-determinism), or a problem in the test's own code

Dependencies May Be Among the the Test Cases

Application code necessarily has dependencies on other application code and often on third-party code, and test cases are necessarily dependent on the application code they test, but test cases are supposed to be *independent* from each other

Test minimization and parallelization can change test results due to dependencies among test classes



Example Re-Order

set up global state then run tests in sequence

do lots of tests;

test1 () { do something that changes global state; };

do lots of other tests;

test2 () { do something that uses global state; }

set up global state then run tests in sequence

do lots of tests;

test2 () { do something that uses global state; }

do lots of other tests;

test1 () { do something that changes global state; }

Example Parallel

set up global state then run tests in parallel;

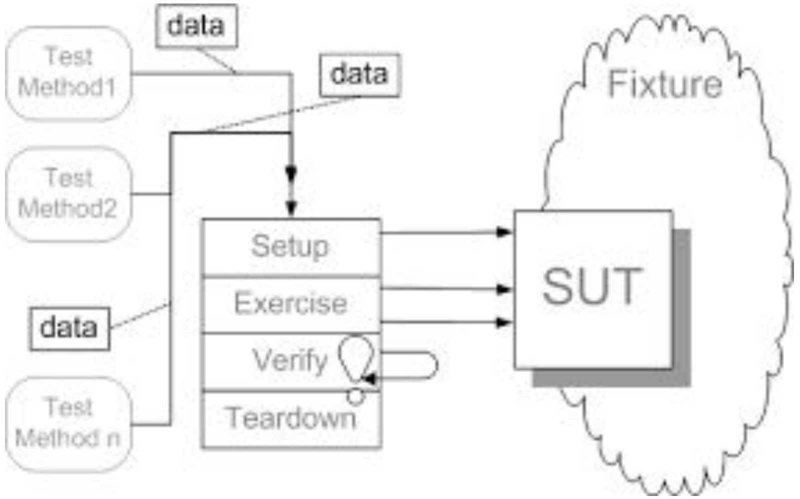
do lots of tests;

test1 () { do something that changes global state; }

do lots of other tests;

test2 () { do something that uses global state; }

Test Classes



Any tests *intended* to share application state and run in a specific order, e.g., one test constructs application state that is used by the next test, should be grouped together in the same “test class” - which testing frameworks treat as an atomic unit

The shared state might include class or global variables, mocks and/or real resources

Setup/teardown fixtures associated with test classes construct and clean up the shared state

Hidden Dependencies Among Test Classes

Finding all dependencies among test classes is expensive - for n test classes, in worst case O(2^n), but only has to be done once (until some test class changes, assumes no flaky tests)

Rather than try to find all dependencies, some testing tools offer an option to *prevent* dependencies from affecting test results by restarting/rebooting the runtime environment in between test classes - just in case

Extremely expensive every time the test suite runs: It might take 3-5ms to run each JUnit test vs. 1.4s to restart JVM between each pair of tests. It takes much longer to reboot Linux. But, technically, only O(n) cost for n test classes...

Simba, this is our code base now you'll maintain it
What's the shadowy section over there
That's legacy code and we don't touch it
meme-arsenal.ru

SolarWinds

I asked you to read two reports about the SolarWinds supply chain attack

The IT management company was infiltrated by a “foreign threat actor” who maintained persistence in its network for months

The threat actor left the network only after it had compromised the company’s build servers (where it ran CI), used the build process to introduce malware into a software product with tens of thousands of corporate and government customers, and used the company’s update process to infiltrate customer networks

What do you think the SolarWinds company could/should have done to secure its build process and/or before releasing CI-produced updates to customers?

Software Supply Chain Attacks

A software supply chain attack occurs when a cyber threat actor infiltrates a software vendor’s network and employs malicious code to compromise the software before the vendor sends it to their customers

The compromised software then compromises the customer’s data or system

Newly acquired software may be compromised from the outset, or a compromise may occur through other means like a patch or hotfix (in these cases, the compromise still occurs prior to the patch or hotfix entering the customer’s network)

These types of attacks affect all users of the compromised software and can have widespread consequences for government, critical infrastructure, and private sector software customers

COMS E6156 Topics in Software Engineering

Software supply chain attacks are one of the kinds of software engineering and security concerns discussed in 6156

The two reports are the kinds of papers students are expected to read before attending the in-class discussion, there will also be academic research papers

The course is not about software supply chain security, that is only one of many topics that may be discussed - last year about half the readings were about AI solutions to SE and/or security problems and the other half included security and bug-fixing mistakes made by developers, flaky tests, clone detection, fuzz testing, automatic program repair, symbolic execution, static analysis, and software fairness

Last year's reading list is here, this year's list will be different, students can suggest papers

6156 is a Seminar, not a Lecture Course

4156 and/or 4181 are recommended prerequisites but not required

Students are **required to attend class**, take turns presenting papers, and discuss the papers (there may or may not be a CVN option, TBD)

The students do most of the talking and talk to each other not to me (but if the university still requires masks in the spring - right, this will be difficult)

6156 Workload

Some number of presentations per student - probably two, depends on enrollment

Midterm literature review paper on student’s choice of topics

Each student writes their own individual paper, 10-15 pages

Final research project on student’s choice of topics

Usually involves software development and/or evaluation

Students may do projects alone or form teams of any reasonable size

Sample papers and project final reports from last year here

Project Student Advertisement

3998, 4901, 6901 Projects in Computer Science

I’m looking for students to develop the 4156 mini-project for fall 2022

Possibly also looking for students to develop a refactoring assignment for fall 2022

Potential research projects to integrate metamorphic testing with fuzzing for finding security vulnerabilities, reverse change analysis to detect hacked build processes, extended ABI for ELF files for splitting and merging Linux binaries, port an existing JVM taint tracker to Android, others...

Team Project Reminder: First Iteration due!

Assignment T3: First Iteration due yesterday

Assignment T4: First Iteration Demo due Friday

Looking for one or two volunteer teams to demo *in class* on Thursday - must do your IA mentor demo before that for sanity check, then ask your mentor to contact me (quickly!) to “recommend” your team for an in-class demo

Assignment T5: Second Iteration due December 11

Assignment T6: Second Iteration Demo due December 15

Assignment T7: Demo Day on December 20