CS 4530
Software Engineering

Lecture 8 - Testing

Jonathan Bell, John Boyland, Mitch Wand
Khoury College of Computer Sciences

Zoom Mechanics

- Recording: This meeting is being recorded
- If you feel comfortable having your camera on, please do so! If not: a photo?
- I can see the zoom chat while lecturing, slack while you're in breakout rooms
- If you have a question or comment, please either:
  - "Raise hand" - I will call on you
  - Write "Q: <my question>" in chat - I will answer your question, and might mention your name and ask you a follow-up to make sure your question is addressed
  - Write "SQ: <my question>" in chat - I will answer your question, and not mention your name or expect you to respond verbally

Today’s Agenda

Administrative:

- HW2 due tomorrow
- HW3, Project pitch posted tomorrow

Today’s session:

- Review: Testing
- Activity: Testing the Transcript Server

Dijkstra's Law

Pioneer of Software Engineering as a discipline

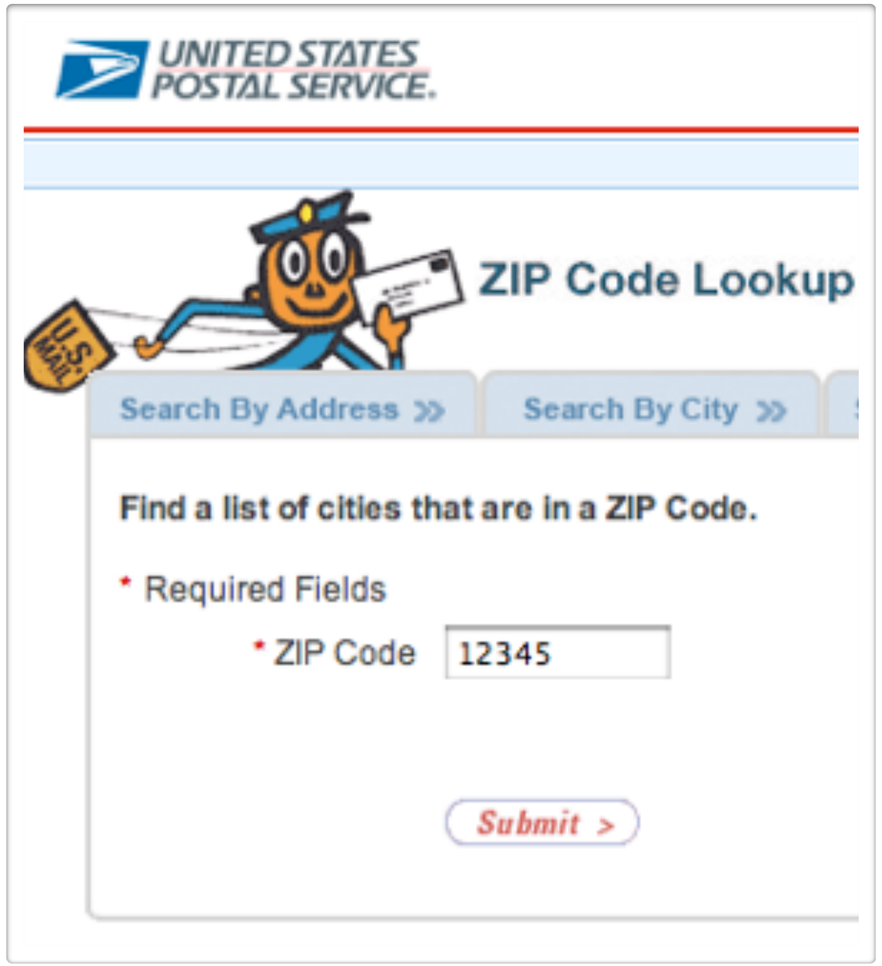
"Program testing can be used to show the presence of bugs, but never to show their absence"

Testing: Two Key Challenges

1. What inputs should I test?
2. For those scenarios: what outputs should I check?

Example: ZIP Code

What inputs should I test?

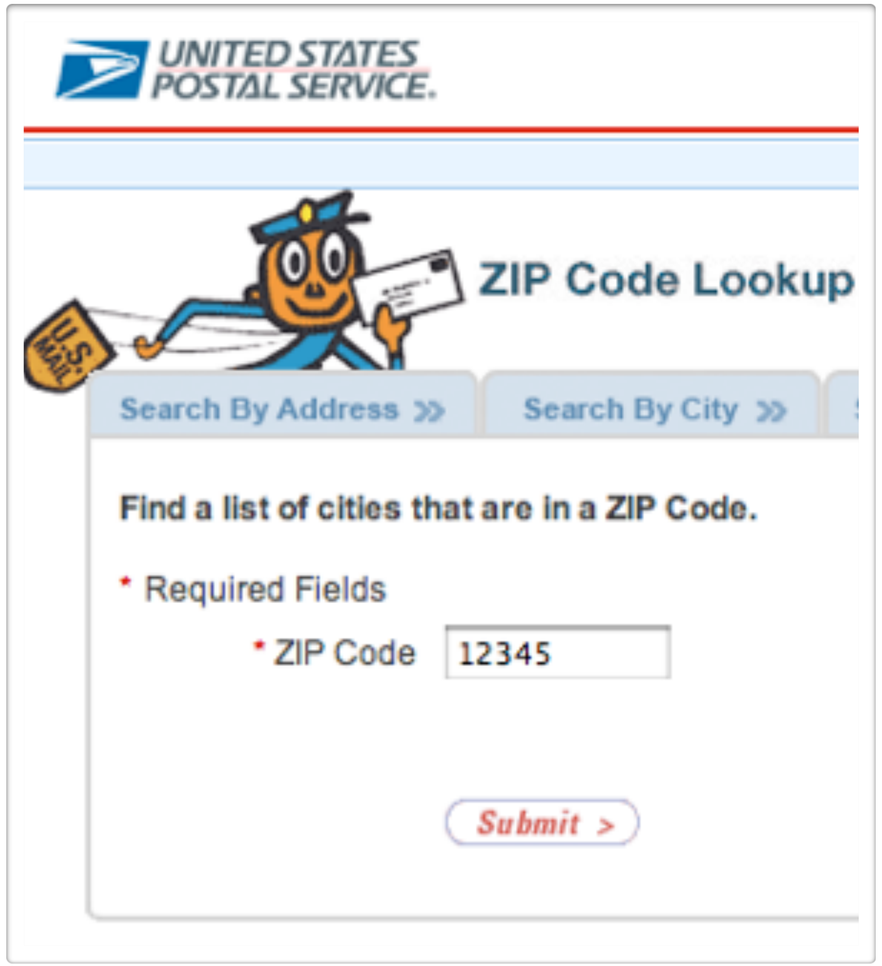


- Input:
  5-digit ZIP code
- Output:
  list of cities
- What are representative values to test?

(from Pezze + Young, “Software Testing and Analysis”, Chapter 10)

Valid ZIP Codes

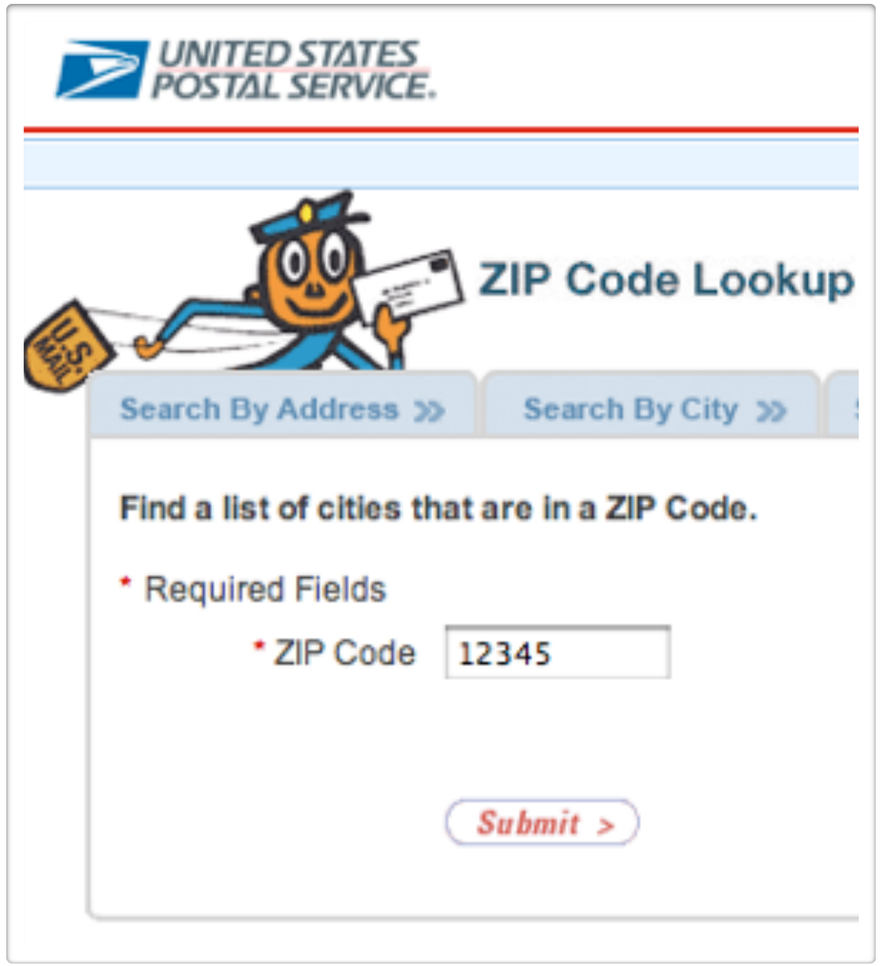
What inputs should I test?



- with 0 cities as output
  (0 is boundary value)
- with 1 city as output
- with many cities as output

Invalid ZIP Codes

What inputs should I test?



- empty input
- 1–4 characters
  (4 is boundary value)
- 6 characters
  (6 is boundary value)
- very long input
- no digits
- non-character data

What inputs should I test?

Two high level answers

- “Black box” input generation: consider specification, conduct boundary value analysis
- “White box” input generation: look at code, figure out input values that will exercise all branches in code

Automated Tests

Is this an effective test?

```
describe('Create student', () => {
  it('should return an ID', async () => {
    const createdStudent = await client.addStudent('Avery');
    expect(createdStudent.studentID).toBeGreaterThan(4);
  });
})
```

Automated Tests

Tests are only as good as their inputs *and* their assertions!

```
describe('Create student', () => {
  it('should return an ID', async () => {
    const createdStudent = await client.addStudent('Avery');
    expect(createdStudent.studentID).toBeGreaterThan(4);
  });
})
```

Test Oracle

Possible Test Oracles

What output should we expect for a given input?

- Human tester infers the right answer
- Simply not crashing is “right”
- Formal specification prescribes the right answer

Pseudo-Oracles

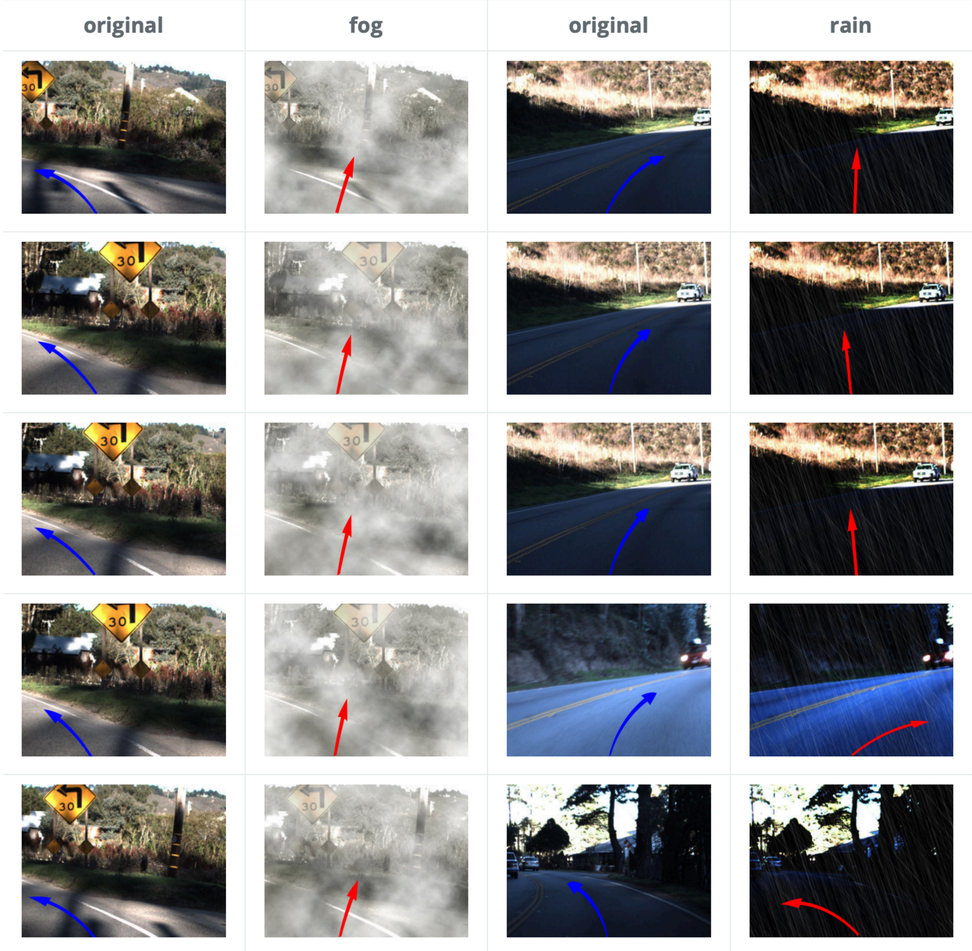
What if we *don't know* what the output should be?

- Regression testing: expect same results on new versions of code
- Differential testing: compare multiple implementations

Pseudo-Oracles and Machine Learning

Testing self-driving cars

- Problem: ML application learns from traffic images, determines how to steer car safely
- How do we exhaustively generate inputs?
- Approach: apply image transformations to known cases



"DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars," Tian et al, ICSE 2018

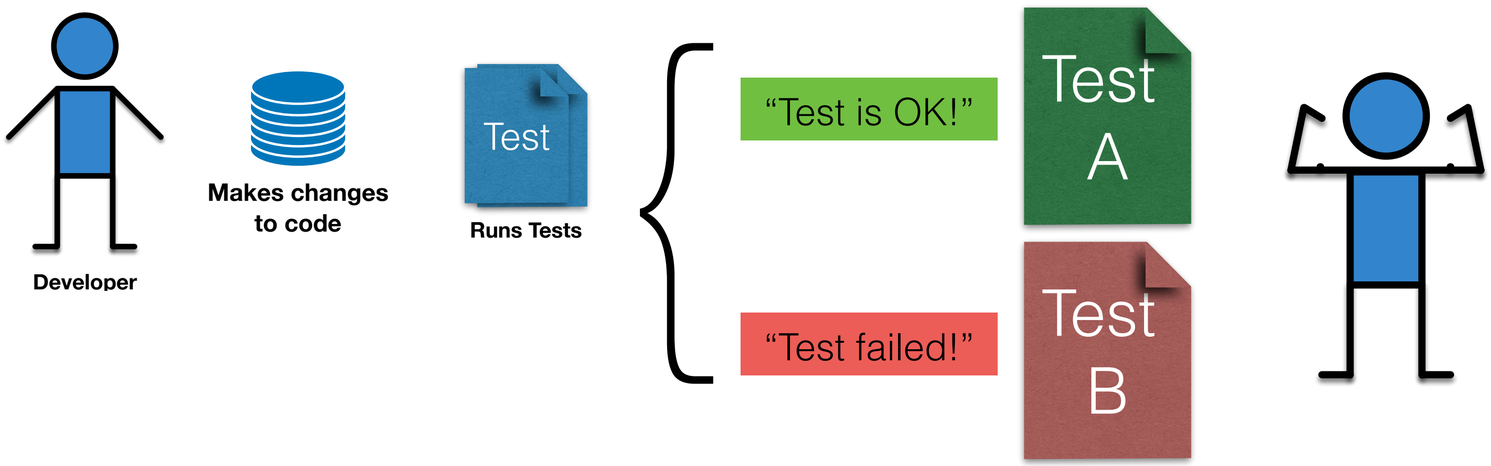
What makes a good test?

The Beyoncé Rule

"Software Engineering at Google: Lessons Learned from Programming Over Time," Wright, Winters and Manshreck, 2020 (O'Reilly)

What makes a good test?

The Beyoncé Rule, applied



What makes a good test?

More than just coverage and oracles

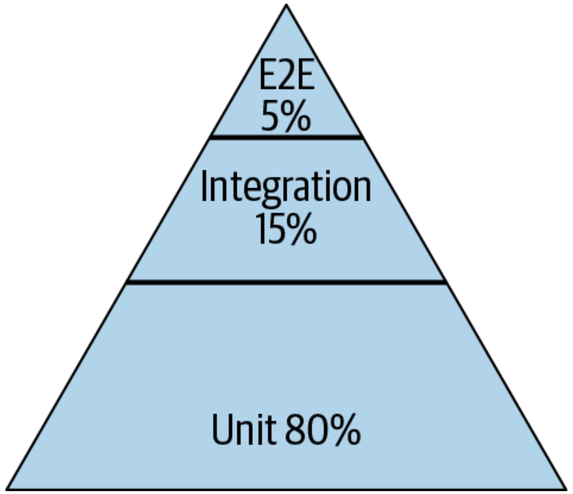
- Tests should be hermetic: reduce flakiness
- Tests should be clear: improves debugging later on
- Tests should be scoped as small as possible: faster and more reliable
- Tests should make calls against public APIs

“Software Engineering at Google: Lessons Learned from Programming Over Time,” Wright, Winters and Manshreck, 2020 (O’Reilly)

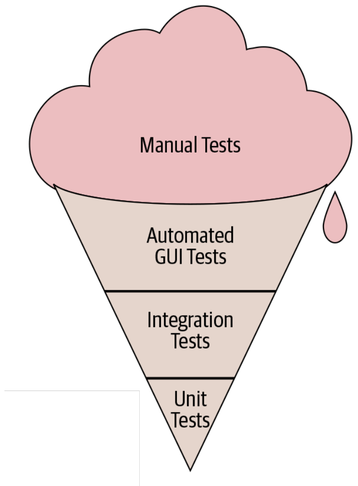
Integration Tests

Integration Tests

Individual unit correctness does not imply full system correctness



Google's Ideal Software Testing Pyramid

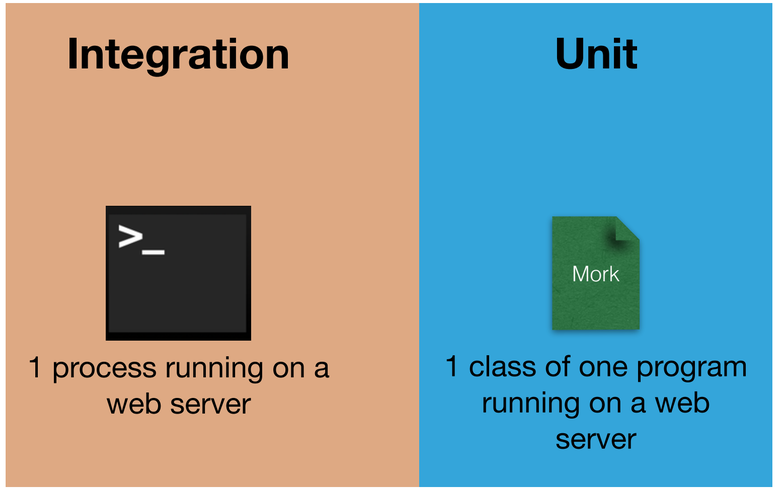


Software Testing Anti-Pattern: Ice Cream Cone Testing

Figures: "Software Engineering at Google: Lessons Learned from Programming Over Time," Wright, Winters and Manshreck, 2020 (O'Reilly)

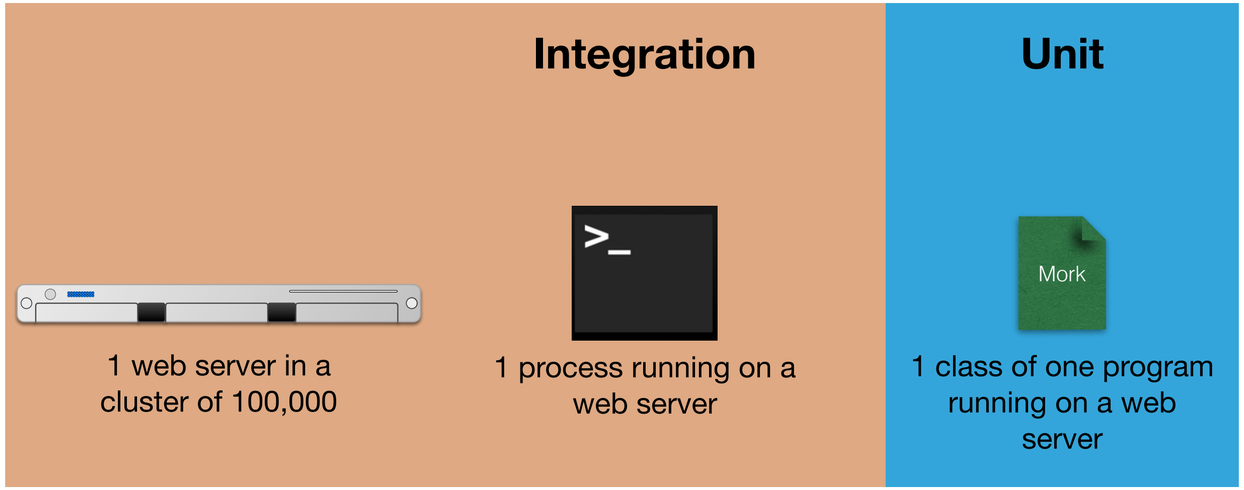
Integration vs Unit Testing

Well, how do you define “unit”?



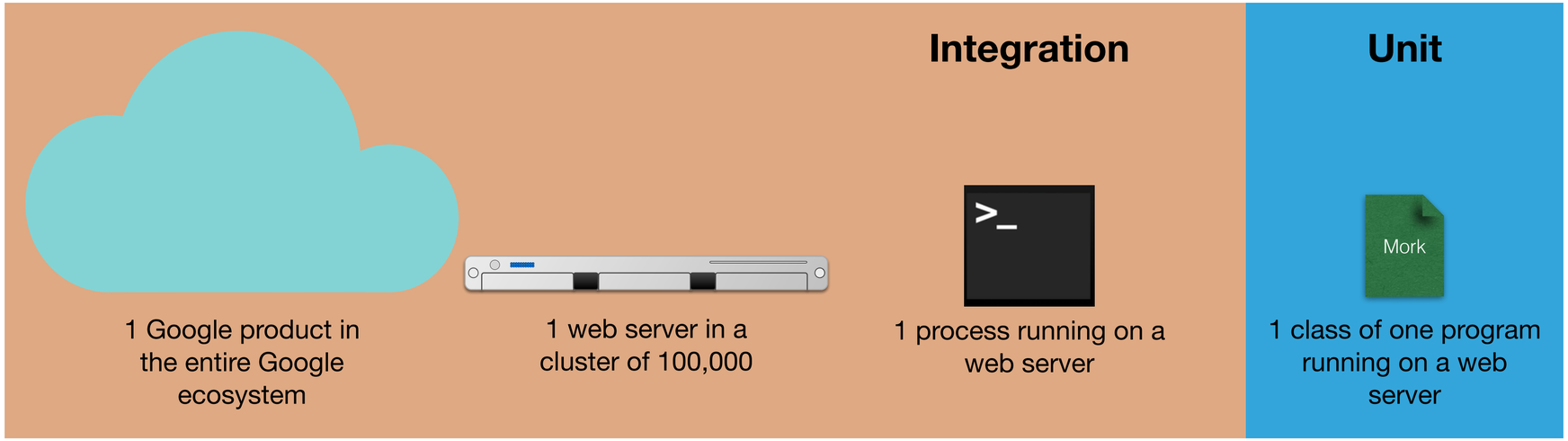
Integration vs Unit Testing

Well, how do you define “unit”?



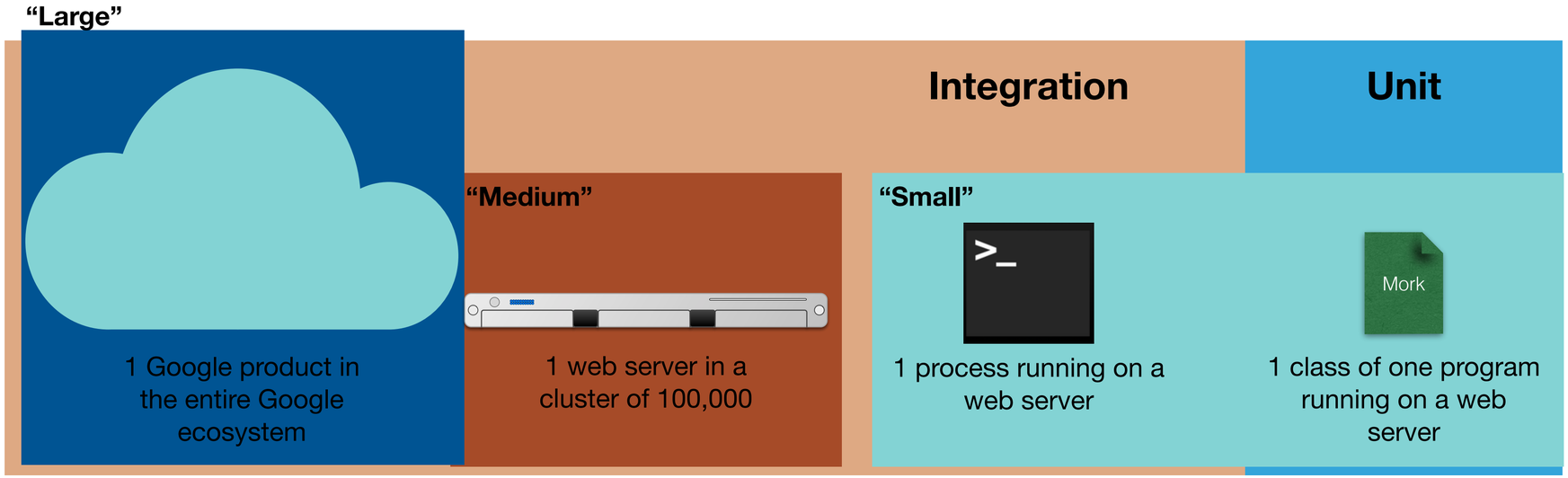
Integration vs Unit Testing

Well, how do you define “unit”?



Integration vs Unit Testing

Consider not just scope, but *size*



"Software Engineering at Google: Lessons Learned from Programming Over Time," Wright, Winters and Manshreck, 2020 (O'Reilly)

How big is my test?

Considerations for test code at Google

- Small: run in a single thread, can’t sleep, perform I/O or making blocking calls
- Medium: run on single computer, can use processes/threads, perform I/O, but only contact localhost
- Large: Everything else

“Software Engineering at Google: Lessons Learned from Programming Over Time,” Wright, Winters and Manshreck, 2020 (O’Reilly)

What makes a good test?

More than just coverage and oracles

- Tests should be hermetic: reduce flakiness
- Tests should be clear: improves debugging later on
- Tests should be scoped as small as possible: faster and more reliable
- Tests should make calls against public APIs

“Software Engineering at Google: Lessons Learned from Programming Over Time,” Wright, Winters and Manshreck, 2020 (O’Reilly)

Is this a good test?

Is it self-contained?

```
describe('Create student', () => {
  it('should return an ID', async () => {
    const createdStudent = await client.addStudent('Avery');
    expect(createdStudent.studentID).toBeGreaterThan(4);
  });
})
```

What makes a bad test?

Test smell: Test Code Duplication

```
describe('hasMork', function () {
    it('Returns true if Mork is in crew', () => {
        let crew = [martianFactory("Mork"), martianFactory("Mal"), martianFactory("Zoe"), martianFactory("Jayne")];
        let ship = mothershipFactory("shipName", crew);
        assert.equal(hasMork(ship), true, "Ship with mork has mork");
    })
    it("Returns false if Mork is not in the crew", () => {
       let crew = [martianFactory("Mal"), martianFactory("Zoe"), martianFactory("Jayne")];
        let ship = mothershipFactory("shipName", crew);
        assert.equal(hasMork(ship), false);
    })
    it("Returns false if Mork is in a daughter ship", () => {
        let mork = martianFactory("Mork");
        let crew = [martianFactory("Mal"), martianFactory("Zoe"), martianFactory("Jayne")];
        let ship = mothershipFactory("shipName", crew, [mothershipFactory("shipName", [mork])]);
        assert.equal(hasMork(ship), false);
    })
})
```

Multiple test methods share the same code

“Refactoring Test Code,” van Deursen et al, XP2001

What makes a bad test: Flaky Tests

Why do Google's testing infrastructure team hate "Large" tests?

- How do we (reliably, repeatedly, cheaply) execute a test that:
  - Changes some global variables?
  - Changes the state of a database?
  - Executes stock trades?
  - Connects to remote servers?

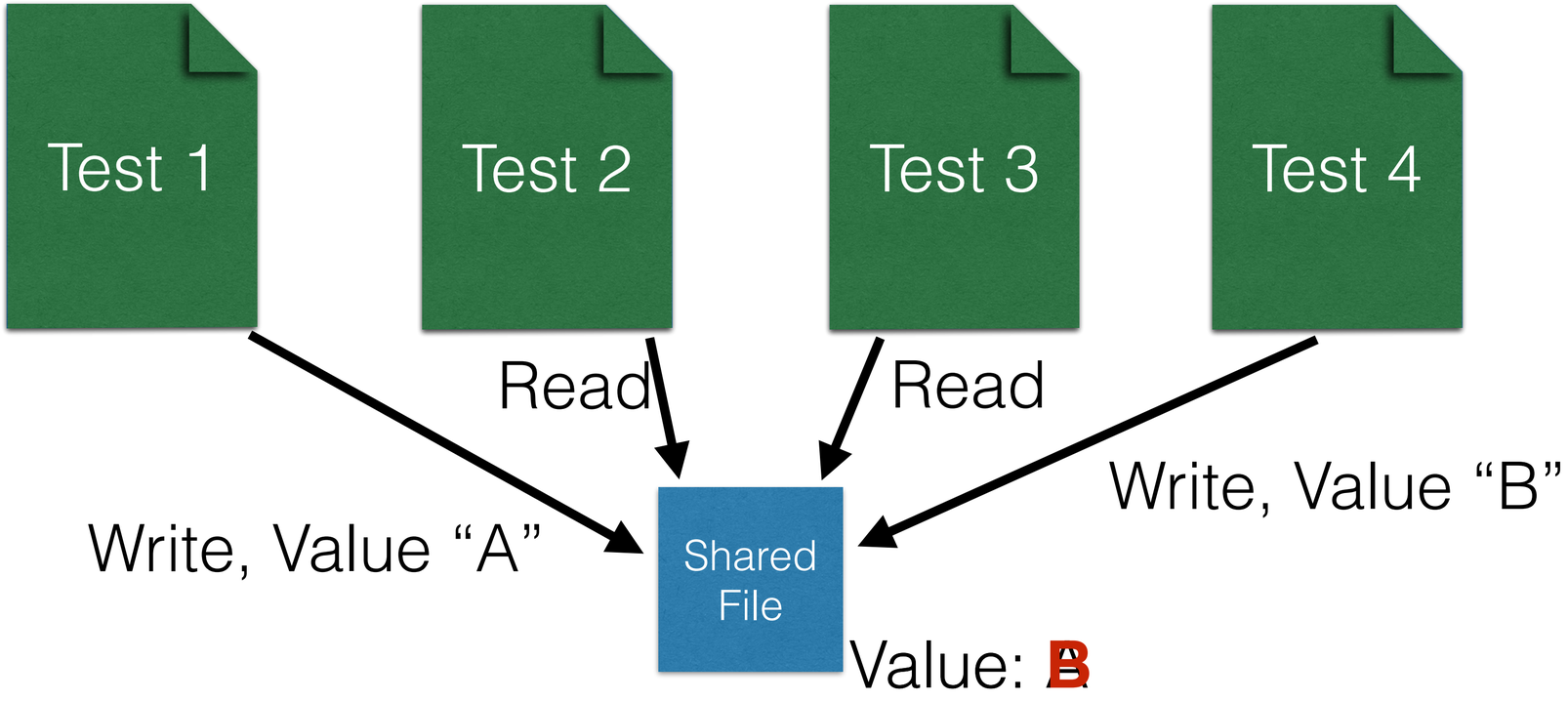
Flaky Tests

An anti-pattern in testing

- Google: 16% of all automated tests are flaky
- Microsoft: 5% of Windows & Dynamics CRM tests are flaky
- Facebook: “Assume all tests are flaky”
- Most developers: flaky tests are a nuisance!

Flaky Tests

Test Order Dependencies

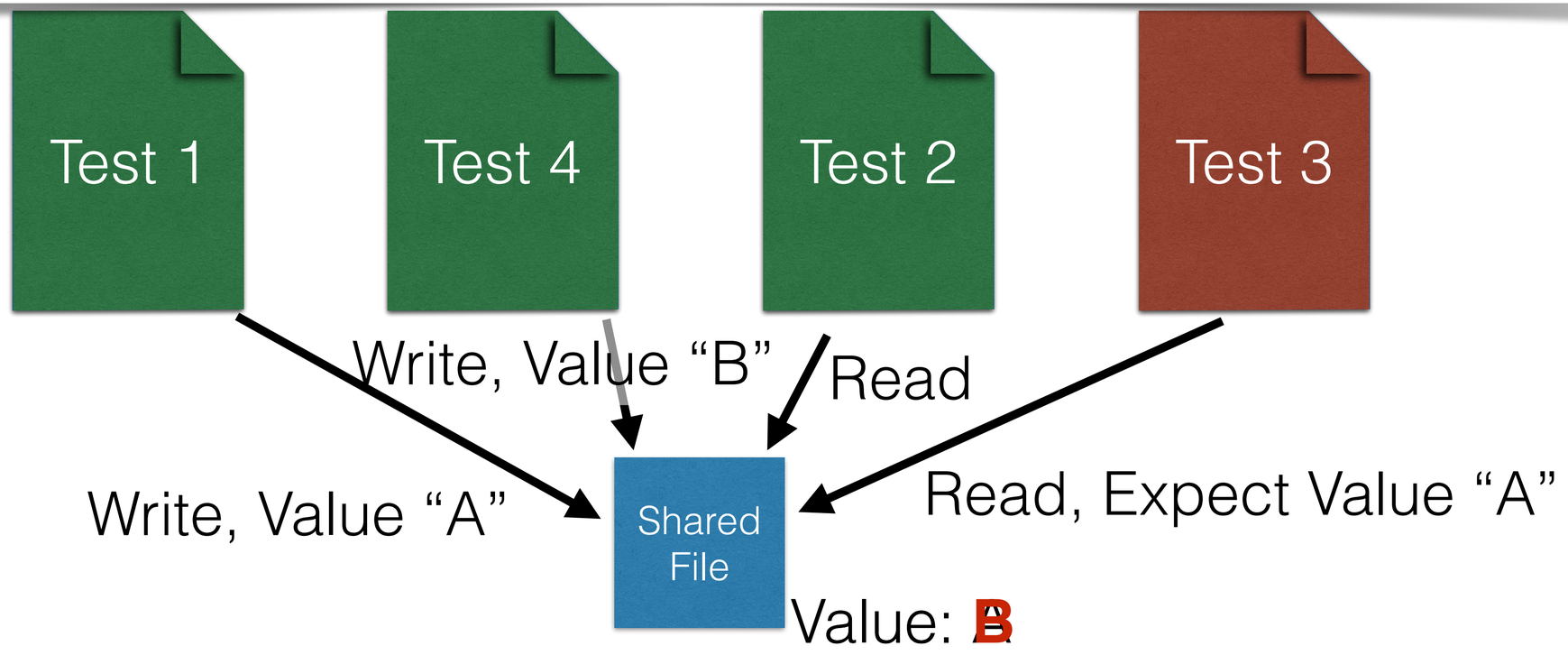


“Efficient dependency detection for safe Java test acceleration”, Bell et al, FSE 2015

Flaky Tests

Test Order Dependencies

A flaky test: outcome of Test 3 changed, but the code hasn't changed!



"Efficient dependency detection for safe Java test acceleration", Bell et al, FSE 2015

Flaky Tests & Test Order Dependencies

Touch global variables or database?

Option 1

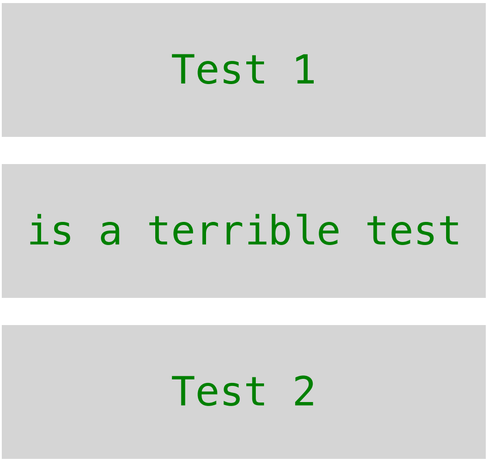
```
let myVar = 5;
describe('test with dependency', function() {
    before( () => {
        // runs once before the first test in this block
        myVar = 10;
    });

    it("is a terrible test", ()=>{
       //do lots of stuff
       myVar = 5;
       //do lots of stuff
       expect(myVar).to.be(5);
    });
    after(() => {
        // runs once after the last test in this block
        myVar = 10;
    });
});
```

Setup, teardown methods

Fast, but “compliance appliance”

Option 2



Isolate each test in a new process (or container)

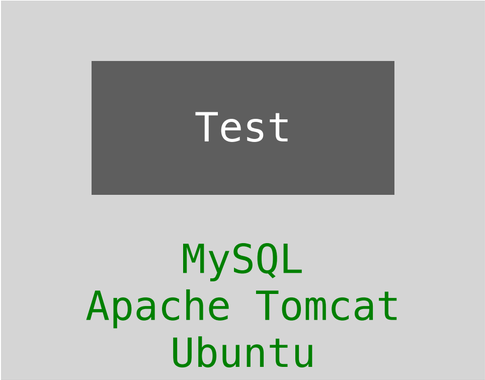
Slow, but “non-compliance appliance”

“Unit Test Virtualization with VMVM,” Bell and Kaiser, ICSE 2014

Flaky Tests & Test Order Dependencies

System tests at scale

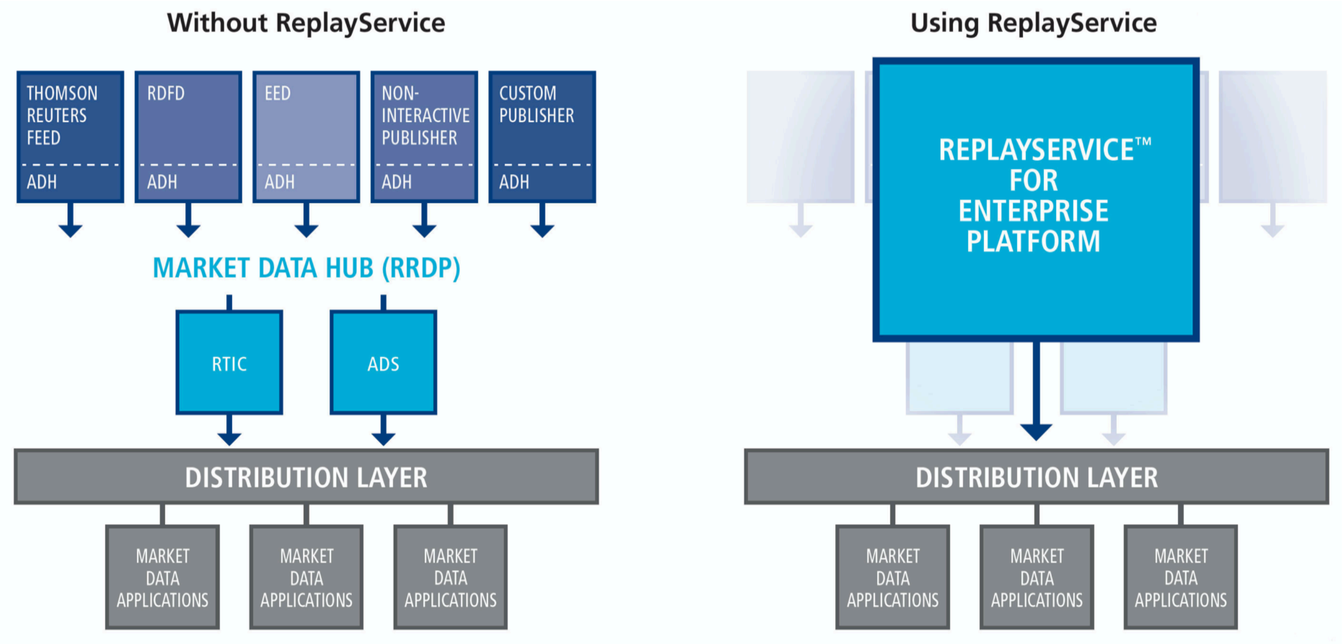
- Relying on engineers to develop and maintain reliable setup/teardown results in unreliable tests
- Without isolation, can’t run multiple tests concurrently
- Common solution: system tests run in entirely isolated environments



Test (running in a newly provisioned VM)

Flaky Tests & External Services

Specialized products replace external components with mocks



Example: TradeWeb ReplayService™: a testing platform for financial market data applications
Originally a product of Thomson Reuters (data provider), then spun off to CodeStreet, then acquired by TradeWeb

https://www.tradeweb.com/our-markets/data--reporting/replay-service/

Flaky Tests Overall

A problem we're stuck with?

- Reduce the scope of a test: small tests aren't flaky
- Remove timed waits, increase timeouts: reduce flaky failures?
- Make tests more understandable: can you tell if a failure is flaky or not?
- Mitigate with reruns, but this increases test cost

"Software Engineering at Google: Lessons Learned from Programming Over Time," Wright, Winters and Manshreck, 2020 (O'Reilly)

Demo: Writing Tests

Activity: Testing the Transcript Server

https://neu-se.github.io/CS4530-CS5500-Spring-2021/Activities/week5-prof-bell-transcript-server.zip

This work is licensed under a Creative Commons Attribution-ShareAlike license