Testing

CPSC 1181 – O.O.

Jeremy Hilliker

Summer 2017

Langara.

THE COLLEGE OF HIGHER LEARNING.

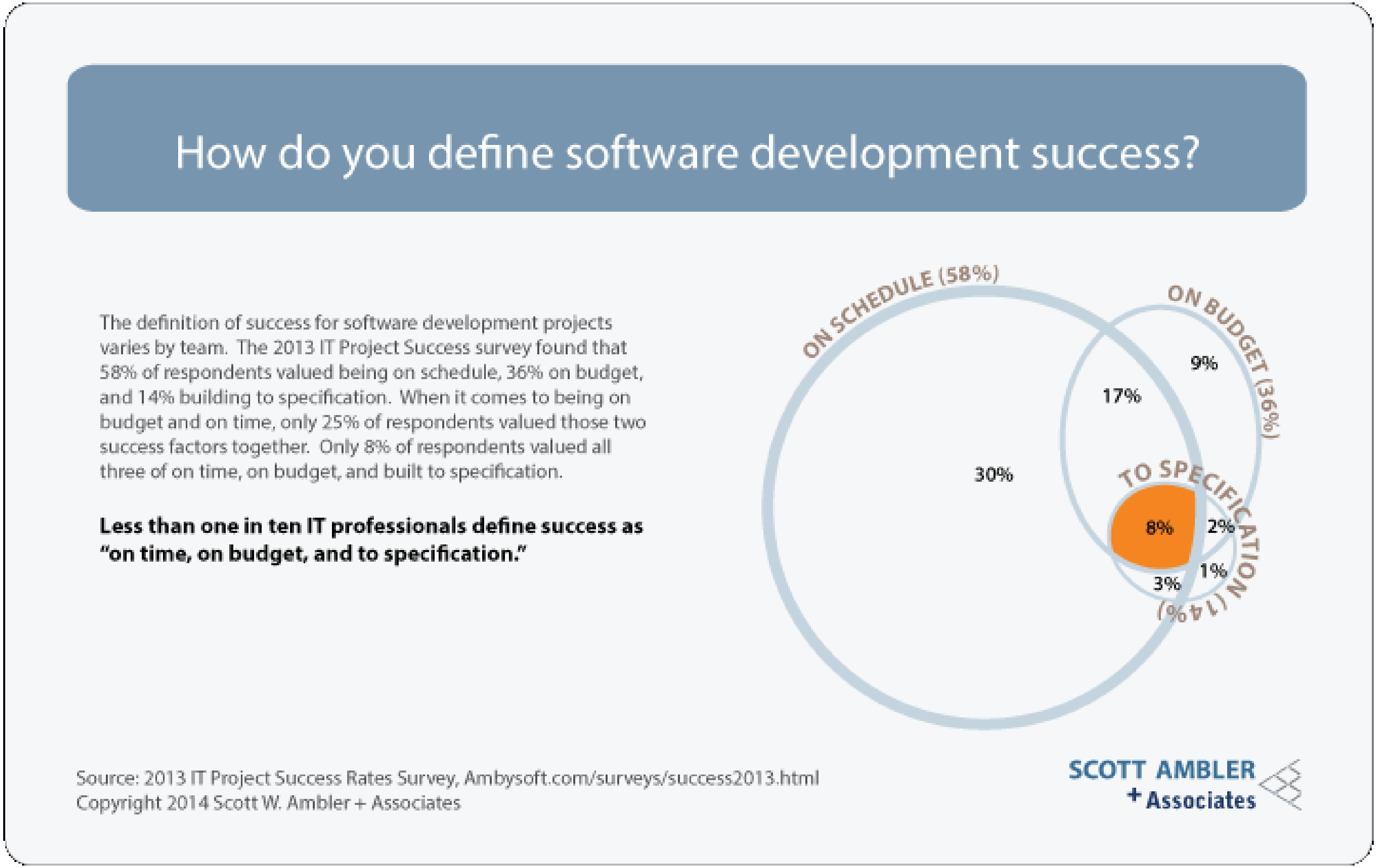
Outline

- SDLC
- Nomenclature
- Coverage
- Automated Testing
- Testing Process
- Regression Tests

Goals of SDLC

- The right product
    - Validated
- Done right
    - Verified
- Managed right

Goals of SDLC



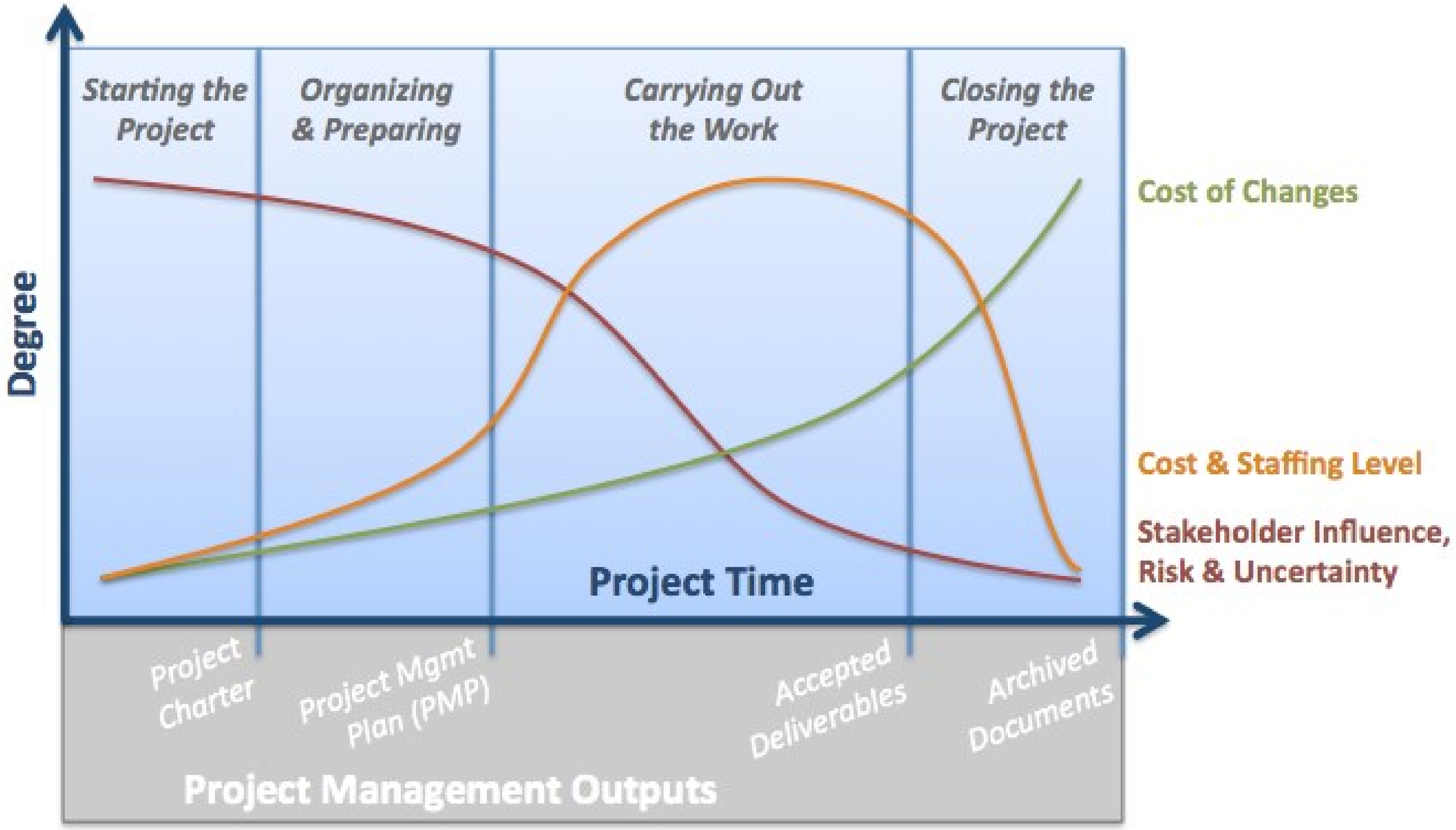
How do you define software development success?

The definition of success for software development projects varies by team. The 2013 IT Project Success survey found that 58% of respondents valued being on schedule, 36% on budget, and 14% building to specification. When it comes to being on budget and on time, only 25% of respondents valued those two success factors together. Only 8% of respondents valued all three of on time, on budget, and built to specification.

Less than one in ten IT professionals define success as "on time, on budget, and to specification."

Source: 2013 IT Project Success Rates Survey, Ambysoft.com/surveys/success2013.html

Cost of Change



https://kevinberardinelli.files.wordpress.com/2011/03/project-life-cycle.png

Defects, Faults, & Failures

- A programmer makes an **error (mistake)**
- Which results in a **defect** in the code
- If the defect is triggered, a **fault** occurs in execution
- If the fault produces an erroneous result given to the user (in any way), then it becomes a **failure**
- Note:
  - Defects that are never triggered don’t become faults
  - Faults that never change output don’t become failures

Sources of Defects

- Not all defects come from coding

- Some are from incomplete (or incorrect) specifications
- Which are the result of inaccurate requirements gathering
  - Often the non-functional requirements

- Some are from a bad design
  - Didn’t account for all cases

- GIGO: garbage in, garbage out
- IKIWISI: I’ll Know It When I See It

What is Testing

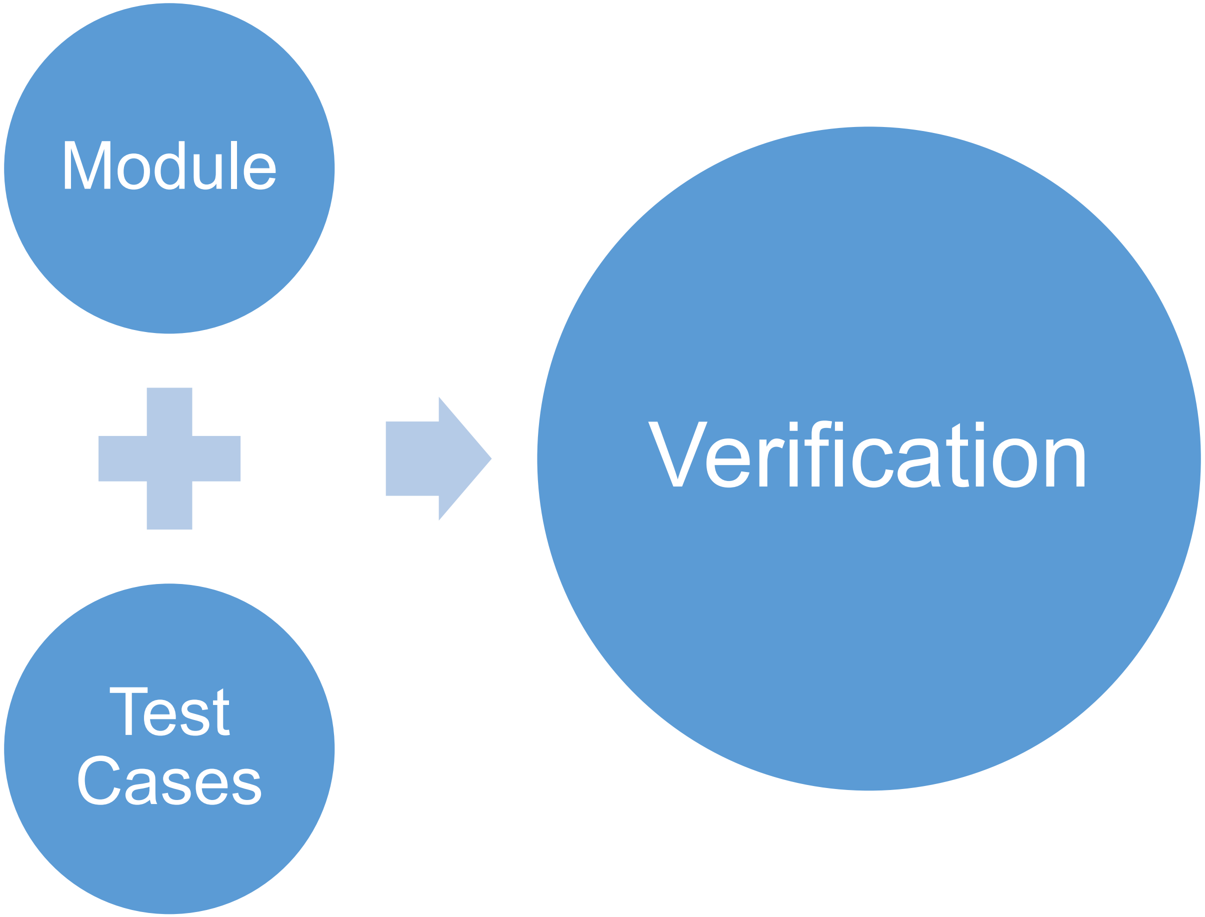
- Testing is the process of exercising a program with the specific intent of finding errors prior to delivery

- Goal: find bugs
  - LOVE FINDING BUGS!
  - Every bug that you find & fix is a bug that doesn’t ship
  - Have pride in shipping low-defect product
  - Easy bugs found in production are embarrassing
  - Hard bugs found in production are interesting

Note:

- Tests aren’t free
- Just like software, they must be maintained
- That has a cost

Diagram of a Unit Test



Types of Testing

- Black Box:
  - No knowledge of implementation
  - Written from specification

- White Box:
  - Full knowledge of the implementation
  - Written from code

- Grey Box:
  - Limited knowledge of implementation

Test Coverage (White Box)

- Minimum: Code Coverage
    - Every line of code is executed by the test suite
- Better: Branch Coverage
    - Every possible branch is executed
        - Both the “true” and the “false” of every if statement

- Equivalency class:
    - $\{x \in S \mid x \sim a\}$
    - a subset where all elements are “equivalent”
    - ie: different inputs, same coverage / conditions

Levels of Testing

- **Unit Testing (small)**
- Integration testing
- Component testing
- System testing
- Acceptance testing (large)

- **Regression testing**
- A/B testing (business outcomes)

Regression Testing

- The re-execution of previous tests
  - To ensure that changes in implementation have not introduced new defects

1. Have a test suite
2. Make a change
3. Rerun old test to verify still working
4. Optional: add new tests for the change
   - White box testing

Automated Testing

- ***All*** of your tests should be ***completely automated***
  - No input. NONE.
  - No evaluation of output. NONE.
  - Test suites are completely *non-interactive*.
  - When you run the tests, they just go
    - They require absolutely no supervision
  - When they finish, they say PASS or FAIL
    - On Pass, they might tell you some stats
      - Number of tests, runtime, etc
    - On fail, they may tell you about the failure
      - Failure message, test name, line number, etc.

  - Should have as few dependencies as possible

What to Test?

- For you:
    - Correctness
        - Does it do what it’s supposed to?
            - Do the pre-conditions produce the post-conditions?
            - Design-by-contract
    - Robustness
        - Range of inputs
        - Error handling
    - At the “Unit” level
        - In OO, a “Unit” is a Class
        - The smallest sub-unit of test on a Class is a method

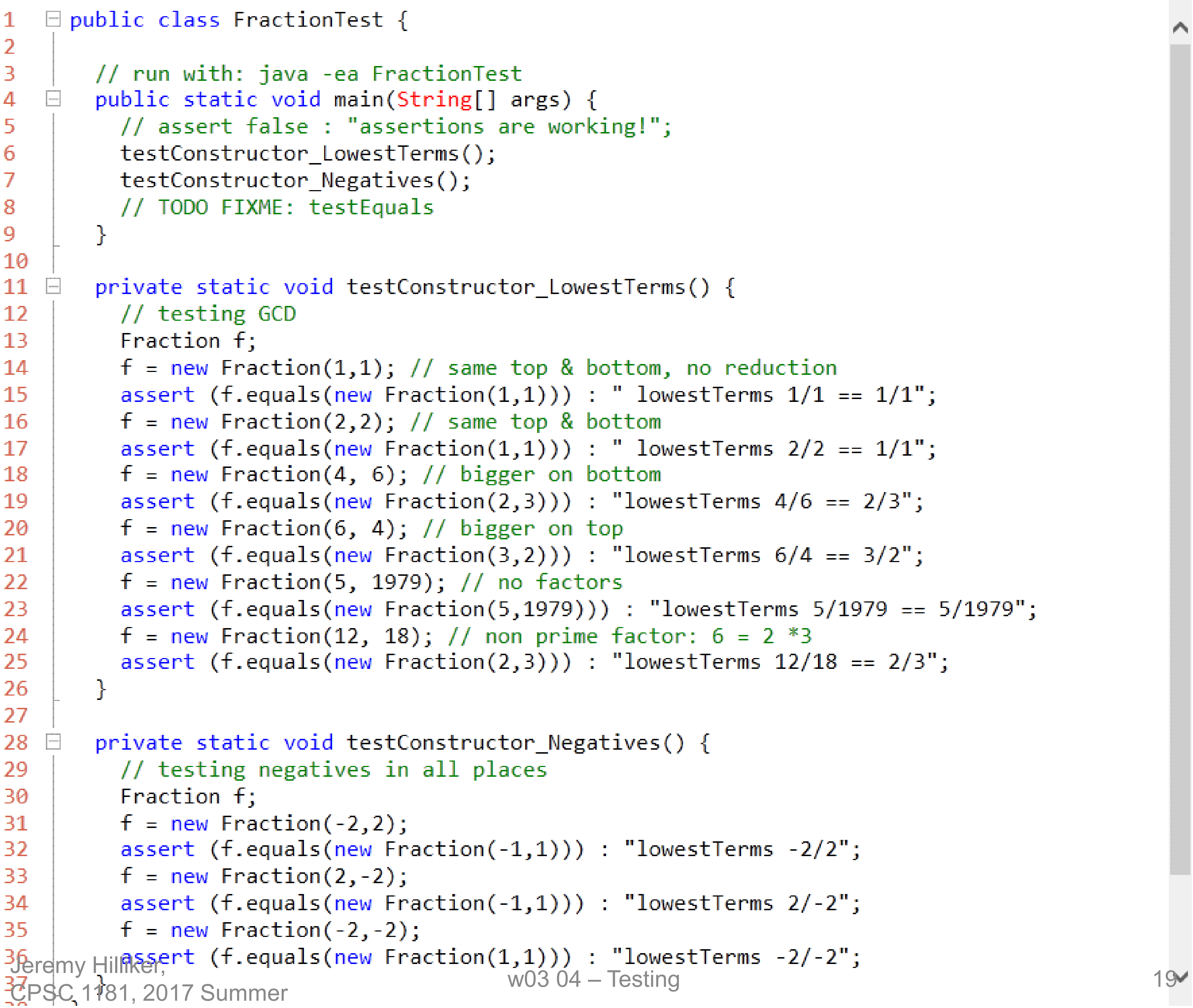
How to Test

1. Exercise a typical case
    1. Then boundaries of that equivalency class
2. Then other typical cases
    - But not in the same equivalency class
    - Then its boundaries
    - (repeat)
3. Then test “special” values
    - 0, -1, 1, null
4. Error handling?
    - Required if part of spec, optional if not

How to Test (Traditional)

- Whenever you have a minimum “working unit”*
  - Test it!
- Write a testMethodName() method:

```
assert (boolean expression to test) : error message;
assert expected == actual : “message”;
assert 29 == primes.get(10) : “10th prime is 29”;
assert new Foo(1).equals(fizz.buzz()) : “fizz buzz”;
```



```java
public class FractionTest {

  // run with: java -ea FractionTest
  public static void main(String[] args) {
    // assert false : "assertions are working!";
    testConstructor_LowestTerms();
    testConstructor_Negatives();
    // TODO FIXME: testEquals
  }

  private static void testConstructor_LowestTerms() {
    // testing GCD
    Fraction f;
    f = new Fraction(1,1); // same top & bottom, no reduction
    assert (f.equals(new Fraction(1,1))) : " lowestTerms 1/1 == 1/1";
    f = new Fraction(2,2); // same top & bottom
    assert (f.equals(new Fraction(1,1))) : " lowestTerms 2/2 == 1/1";
    f = new Fraction(4, 6); // bigger on bottom
    assert (f.equals(new Fraction(2,3))) : "lowestTerms 4/6 == 2/3";
    f = new Fraction(6, 4); // bigger on top
    assert (f.equals(new Fraction(3,2))) : "lowestTerms 6/4 == 3/2";
    f = new Fraction(5, 1979); // no factors
    assert (f.equals(new Fraction(5,1979))) : "lowestTerms 5/1979 == 5/1979";
    f = new Fraction(12, 18); // non prime factor: 6 = 2 *3
    assert (f.equals(new Fraction(2,3))) : "lowestTerms 12/18 == 2/3";
  }

  private static void testConstructor_Negatives() {
    // testing negatives in all places
    Fraction f;
    f = new Fraction(-2,2);
    assert (f.equals(new Fraction(-1,1))) : "lowestTerms -2/2";
    f = new Fraction(2,-2);
    assert (f.equals(new Fraction(-1,1))) : "lowestTerms 2/-2";
    f = new Fraction(-2,-2);
    assert (f.equals(new Fraction(1,1))) : "lowestTerms -2/-2";
  }
```

To Run

- **javac YourTestSuite.java**
- **java –ea YourTestSuite**
  - **ea** = “enable assertions”

- Limitations:
  - Can only see *failures*
    - No progress indicators
  - Entire test suite stops on 1st failure

- A better platform: JUnit
  - Maybe we’ll get to this later (has dependencies)

Output

- On Failure:

```
> javac *.java
> java -ea FractionTest
Exception in thread "main" java.lang.AssertionError:
assertions are working!
        at FractionTest.main(FractionTest.java:5)
```

- On Pass

```
> javac *.java
> java -ea FractionTest
> [no output]
```

Recap

- SDLC
- Nomenclature
  - Error, defect, fault, failure
  - Levels
  - Types
- Test Coverage
- Testing Process
  - What to test
  - How to test
    - Automated Testing
  - Example Test Suite
  - Running Tests
  - Test Output
- Regression Tests