

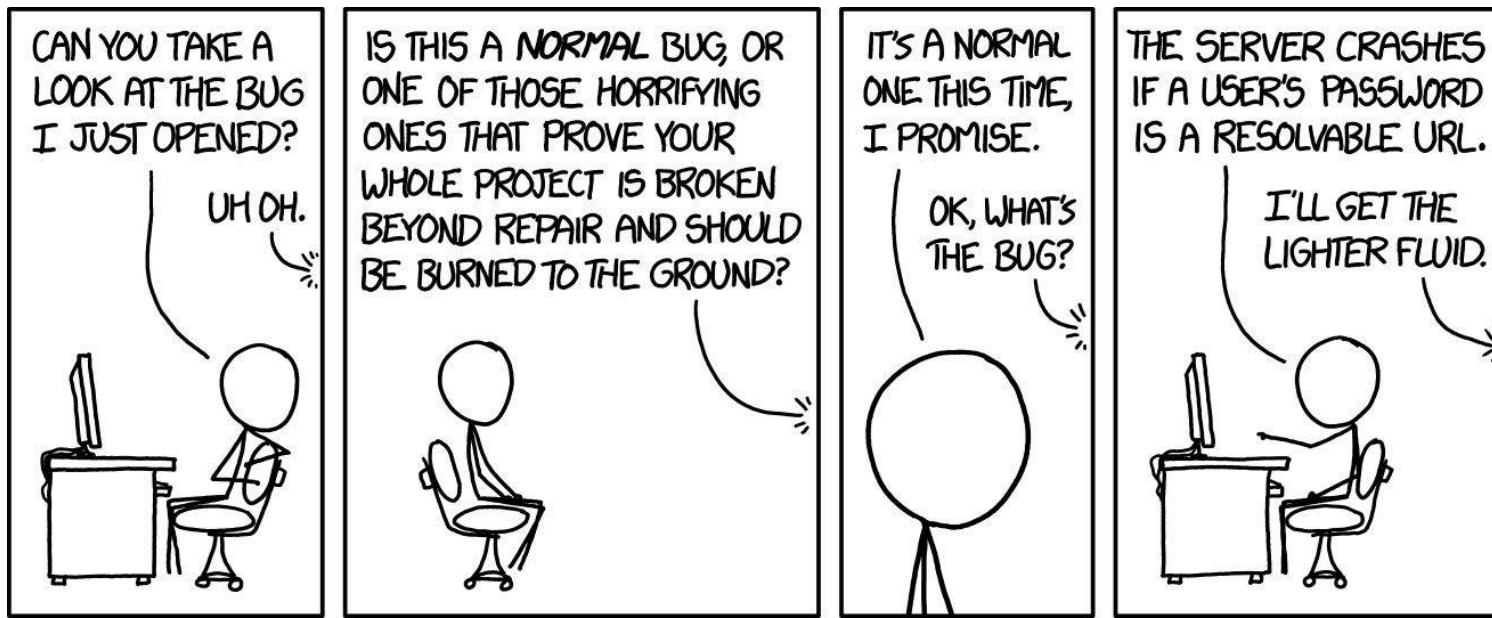
# CSci 3081W: Program Design and Development

## Lecture 7 – Testing

# Testing

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

– Dijkstra, 1969



# Testing

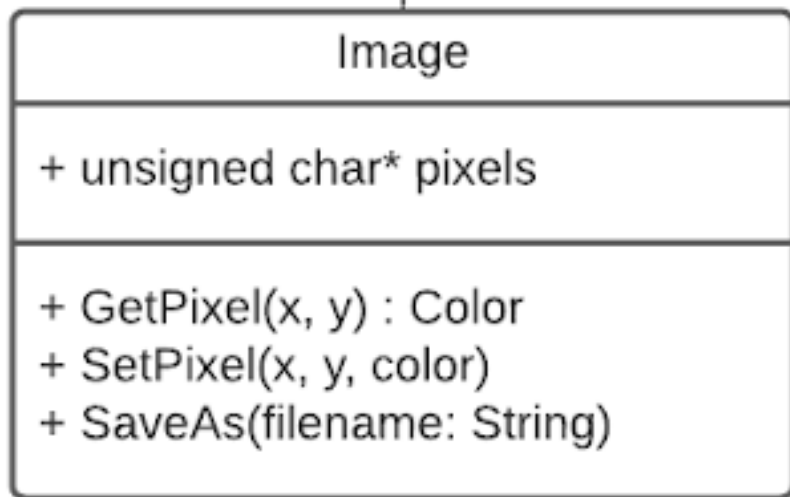
- As the number of detected defects in a piece of software increases, the probability of the existence of more undetected defects also increases
- Assign your best programmers to testing
- Exhaustive testing is impossible
- You cannot test a program completely
- Even if you do find the last bug, you'll never know it
- It takes more time than you have to test less than you'd like
- You will run out of time before you run out of test cases

# Types of Testing

- Different standards on types of testing
- According to the optional textbook, there are five types of tests:
  - Unit Testing – testing a single class
  - Component Testing – testing a subsystem
  - Integration Testing – testing multiple classes
  - Regression Testing – testing that future changes don't break the system
  - System Testing – testing the entire system including hardware performance, security, etc.

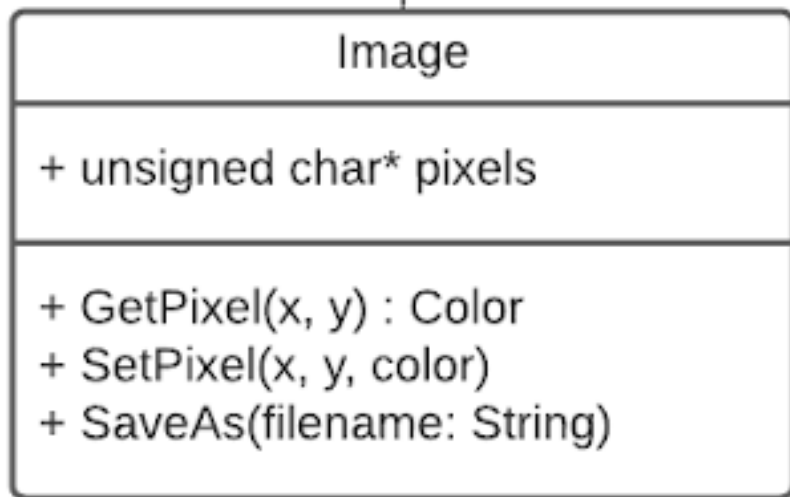
# Unit Test

```
TEST_F(ImageTest, SavesImage) {  
    Image image("input.png");  
    image.SaveAs("output.png");  
    ASSERT_TRUE(fileExists("output.png"));  
}
```



# Integration Test

```
TEST_F(ImageTest, ColorCorrect) {  
    Image image("red.png");  
    Color c = image.GetPixel(0,0);  
    EXPECT_FLOAT_EQ(c.Red(), 1.0)  
}
```



# Testing

Testing can be done at multiple levels of knowledge



# Testing



Figure: Developer

Understands the system  
Will test gently  
Driven by deadlines



Figure: Independent Tester

Must learn the system  
Will attempt to break it  
Driven by "quality"



# Quality Tests



Test Driven Development



Statistical & Boundary Analysis

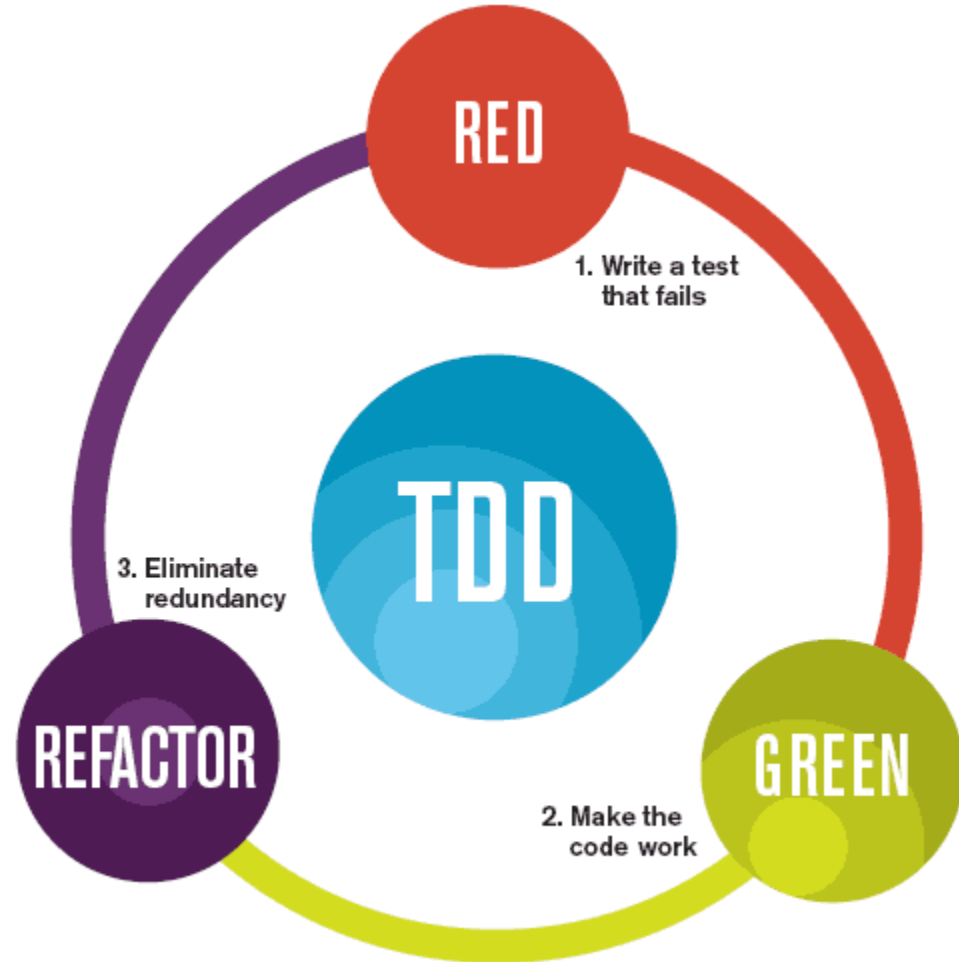


Structured Basis Testing



Mutant Analysis

# Test Driven Development



The mantra of Test-Driven Development (TDD) is “red, green, refactor.”

# Test Driven Development

## THE RULES OF TDD



### RULE 1

Don't write production code unless it's to help a failing unit test pass.



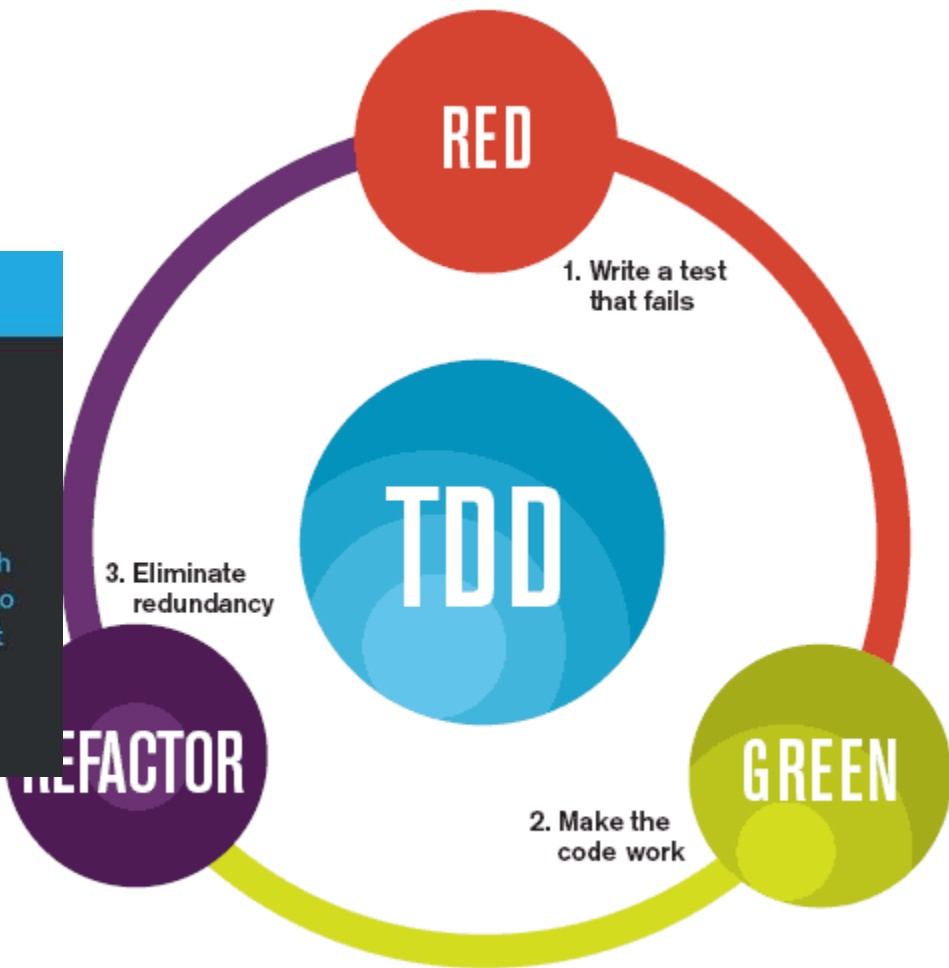
### RULE 2

Only write enough of a unit test to fail.



### RULE 3

Only write enough production code to help a failing unit test pass.



The mantra of Test-Driven Development (TDD) is “red, green, refactor.”

# Test Driven Development (TDD)

Beginner:

- able to write a unit test prior to writing the corresponding code
- able to write code sufficient to make a failing test pass

# Test Driven Development (TDD)

## Intermediate

- practices “test driven bug fixing”: when a defect is found, writes a test exposing the defect before correction
- able to decompose a compound program feature into a sequence of several unit tests to be written
- knows and can name a number of tactics to guide the writing of tests (for instance “when testing a recursive algorithm, first write a test for the recursion terminating case”)
- able to factor out reusable elements from existing unit tests, yielding situation-specific testing tools

# Test Driven Development (TDD)

## Advanced

- able to formulate a “roadmap” of planned unit tests for a macroscopic features (and to revise it as necessary)
- able to “test drive” a variety of design paradigms: object-oriented, functional, event-drive
- able to “test drive” a variety of technical domains: computation, user interfaces, persistent data access...

# Test Driven Development (TDD)

More info / further reading:

## **Test Driven Development: By Example**

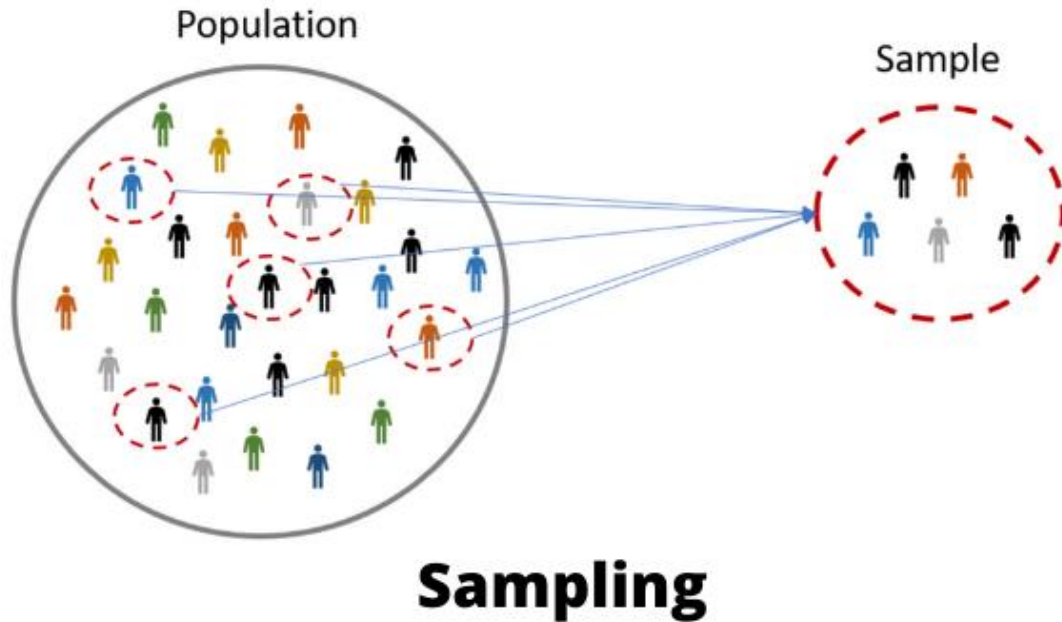
By Kent Beck

[https://www.amazon.com/gp/product/0321146530/ref=as\\_li\\_ss\\_tl?ie=UTF8&linkCode=sl1&tag=&linkId=6d76ed79b7f498d7930b370ad4c059a](https://www.amazon.com/gp/product/0321146530/ref=as_li_ss_tl?ie=UTF8&linkCode=sl1&tag=&linkId=6d76ed79b7f498d7930b370ad4c059a)

C

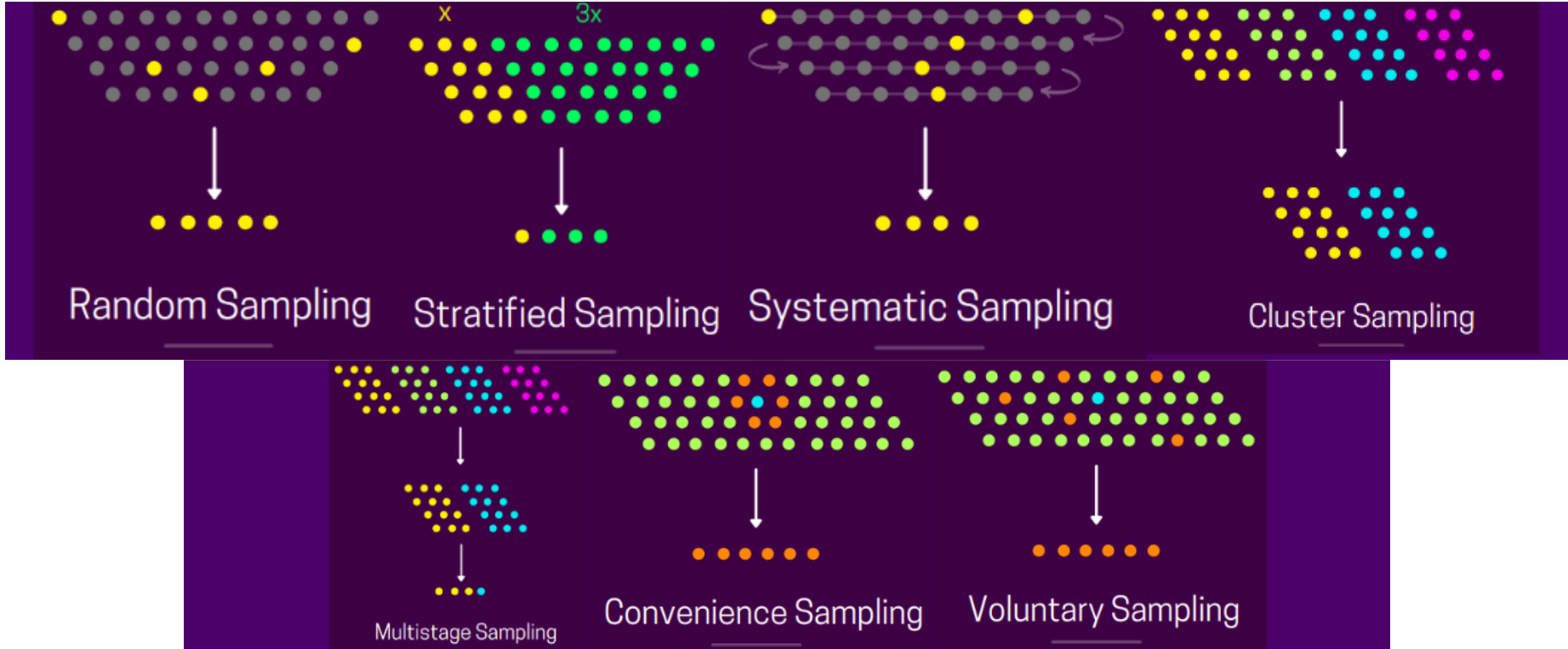
# Statistical and Boundary Analysis

Test with good data and the test should pass





# Sampling Methods



# Boundary Analysis

Test edge cases and bad data

Edge Test Cases:

1. Value just above the maximum
2. Value equal to the maximum
3. Value just below the maximum

Bad Data Cases:

- Incorrect format
- Invalid pointer
- Wrong type
- Too little / too much data

# Structured Basis Testing

1. Is every statement containing a ***condition*** executed by at least one test case?
2. Is every ***statement*** in the code executed by at least one test case?
3. Is every ***branch*** in the code covered in the following sense: is every condition evaluated to true by at least one test case, and to false by at least one test case?
4. Is each part of a ***compound*** condition evaluated to true by at least one test case, and to false by at least one test case?

# Structured Basis Testing

Mentioned in McConnell, Ch. 22.3 (pp. 505 - 509). To find a minimal number of test cases to cover all possible values of each part of each condition in a of a piece of code:

1. Count 1 for the code itself.
2. Count 1 for every loop.
3. Count 1 for every if, else, etc.
4. If any condition (including loop termination) is compound, add one for each additional part of the condition.

# Structured Basis Testing

Start with 1 for  
the routine itself

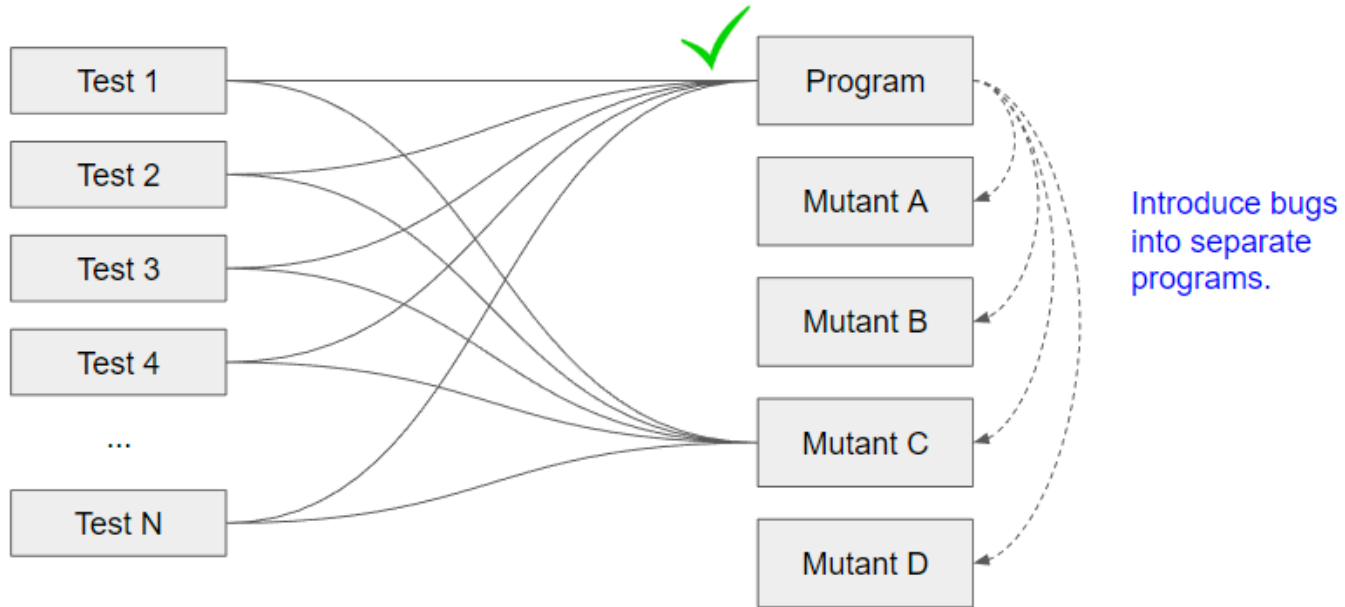
```
characterCount = 0;
for (int i = 0; i < N; i++)
    for (int j = 0; j < lines[i]; j++)
        characterCount += (word[i][j]).size();
cout << "Number of lines: " << N << endl;
cout << "Number of characters: "
    << characterCount << endl;
if (N != 0 && printAverage == true) {
    cout << "Average characters per line: "
        << characterCount/N << endl;
}
```

Add 1 more  
since the  
if has one AND  
in it

This gives a count of 5.

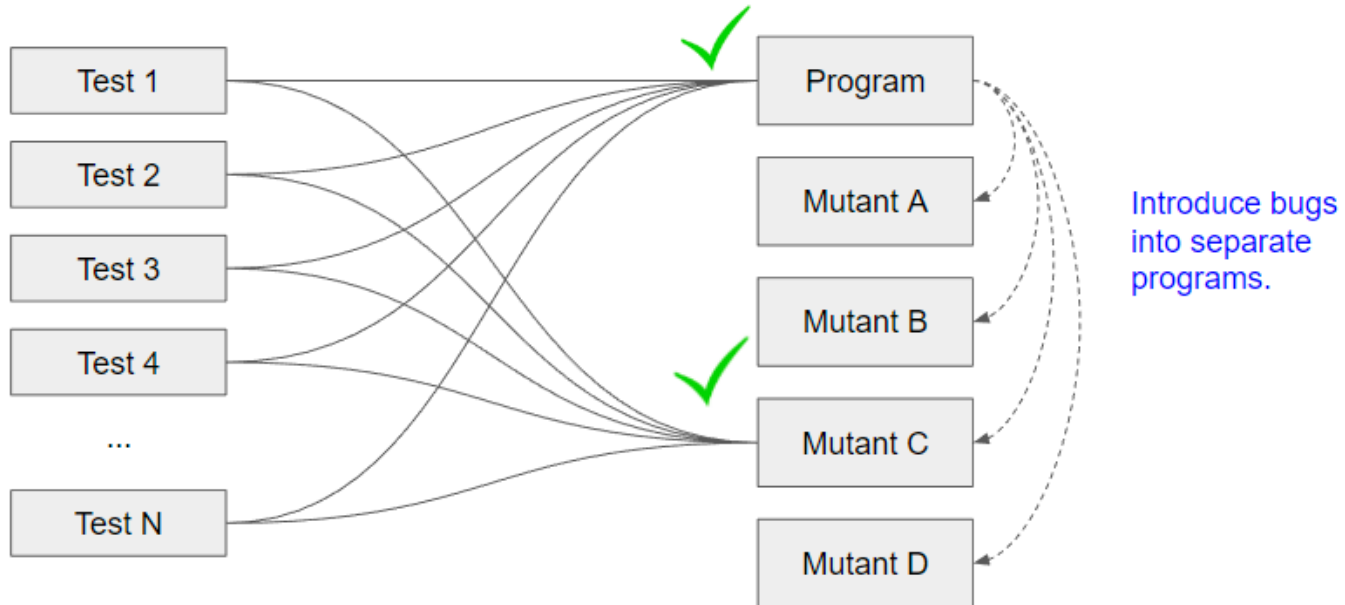
# Mutant Analysis Testing (homework 3)

Process of creating many programs with small bugs



# Mutant Analysis Testing (homework 3)

Process of creating many programs with small bugs



If all the tests pass, that is a bad thing

# Other types of testing

- Functional testing
- End-to-end testing
- Acceptance testing
- Smoke testing

Atlassian's CI/CD guide for software testing:

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



# Google C++ Testing Framework

Skim this: <https://developer.ibm.com/articles/au-googletestingframework/>

Additional reading on Mock Testing:

[http://google.github.io/googletest/gmock\\_for\\_dummies.html](http://google.github.io/googletest/gmock_for_dummies.html)