Software Quality & Testing

Thomas Fritz

Many thanks to Reid Holmes and Elisa Baniassad

Agenda

1. Introduction to software quality
2. Testing
3. Stopping Criteria (Test Suite Breadth)
    1. Code Coverage
    2. Equivalence Class Partitioning & Boundary Testing
4. Mutation testing (Test Suite Depth / Effectiveness)
5. Quiz & Midterm

Note:

- *Have an IDE (e.g. VS Code or IntelliJ) installed on your machine for next week*

Examinable skills

By the end of this lecture, you should be able to…

- Describe what software quality is and the benefits of high-quality code
- Describe mechanisms for improving code quality
- Describe and be able to explain the differences between the various ways of testing and testing tactics, and know which one to choose when
- Decide when to stop testing
- Given a method, be able to determine a ‘good’ and minimal test suite (e.g. using equivalence class partitioning or achieving a certain coverage criteria)
- Explain mutation testing and be able to define test cases and calculate kill score

Software Quality

Learning Objectives

Be able to:

- Describe what software quality is
- Describe ways to to improve code quality

Software Failures / Bugs

American Airlines – no pilots for the holidays (Nov 2017)

- Scheduling system to assign pilots to flights, indicated plenty pilots and other system gave people time off
- Would have affected 15,000 flights if not caught;
- AA offered 1.5 pay and time off
- https://www.wired.com/story/american-airlines-computer-glitch-leaves-it-without-pilots-over-christmas/

Nissan– recall of approx. 3.5 Mio cars (2013 – 2016)

- System could not detect whether an adult was sat in the car's passenger seat and as a result, the airbags would not inflate
- Three injuries but no fatalities
- https://www.telegraph.co.uk/news/2016/04/30/nissan-recalls-35-million-cars-over-airbag-issues/

Poor software quality has become one of the most expensive topics in human history > $ 150 billion / year in US and > $ 500 bio / year world wide

What is Software Quality?

According to IEEE

- The degree to which a system, component or process meets the specified requirements / customer or user needs and expectations.

According to Roger Pressman

- Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

Software Quality Attributes

- Functionality, reliability, maintainability, efficiency, usability, portability,…

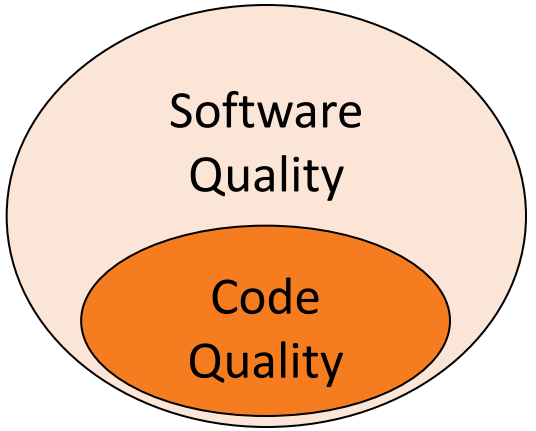
Quality Management

Quality Planning, Quality Assurance, Quality Control, Quality Improvement

- **Quality assurance** activities are focused on the process used to create the deliverables.
- **Quality control** activities are focused on the product itself.

Code is not the only element of software quality

Process quality influences code quality



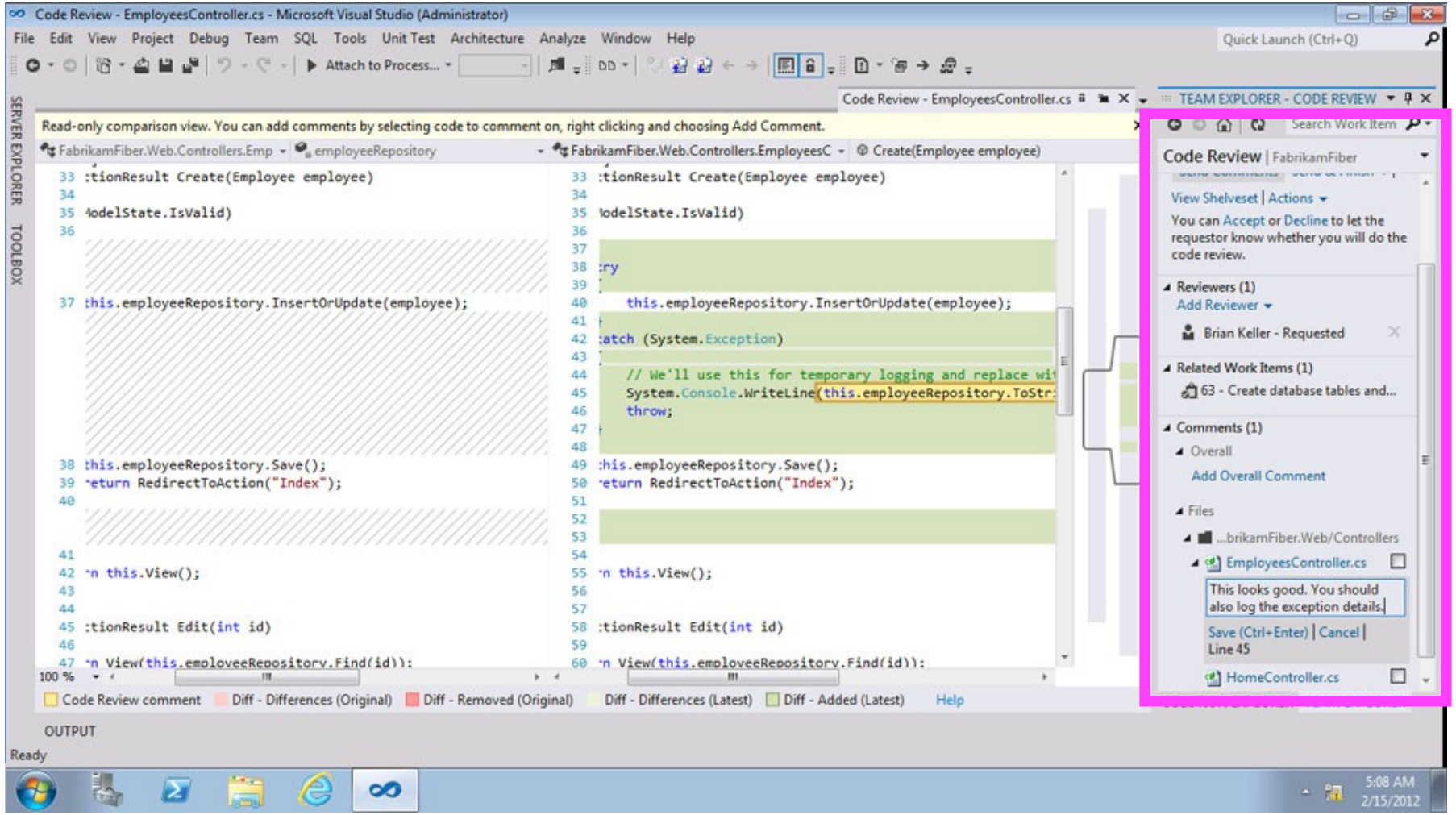
How could you ensure / improve code quality in a software project? [2mins]

- With your neighbour, briefly discuss what you can do.

Examples of mechanisms to improve code quality

- Testing (unit, integration, regression, acceptance, …); automated testing, TDD
- Code reviews (formal, informal)
- Pair programming
- Frequent demonstration of working software to stakeholders
- Code smell detection and refactoring
- Component reuse
- Coding standards
- Automatic tools (static analysis, lint(ers), IDEs, …)
- Team building activities
- *Process standards* (e.g. CMMi: Capability Maturity Model integration)
- …

Code reviews – Visual Studio: benefits???



Similar: reviewing changes in pull requests: https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/reviewing-changes-in-pull-requests

Code Reviews – Guideline Examples

Am I able to **understand** the code easily?

Is the code written following the **coding standards/guidelines**?

Is the same code **duplicated** more than twice?

Can I **unit test / debug** the code easily to find the root cause?

Is this function or class **too big**? If yes, is the function or class having too many responsibilities?

[https://www.evoketechnologies.com/blog/code-review-checklist-perform-effective-code-reviews/]

Does the code completely and correctly implement the design?
Does the code conform to any pertinent **coding standards**?
Is the code well-structured, consistent in **style**, and consistently **formatted**?
Are there **any uncalled or unneeded procedures** or any unreachable code?
Are there any blocks of **repeated code** that could be condensed into a single procedure?
…
Is the code clearly and adequately **documented** with an easy-to-maintain commenting style?
…
Are all **variables** properly defined with meaningful, consistent, and clear names?

https://www.liberty.edu/media/1414/%5B6401%5Dcode_review_checklist.pdf

Joel Test: 12 steps to better code

1. Do you use source control?
2. Can you make a build in one step?
3. Do you make daily builds?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
6. Do you have an up-to-date schedule?
7. Do you have a spec?
8. Do programmers have quiet working conditions?
9. Do you use the best tools money can buy?
10. Do you have testers?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

see http://www.joelonsoftware.com/articles/fog0000000043.html

Testing

Learning Objectives

Be able to:

- Describe differences between various ways of testing

Why Test?

https://hiveminer.com/Tags/error%2Ctram

Everyone (developers, managers, …)

should care about testing!

Industry averages

- 30-85 errors per 1000 lines of code
- 0.5-3 errors per 1000 lines of code in product (i.e. not discovered before the product is delivered)

Testing and goals

> Given a finite amount of time and resources, how can we validate that the system has an acceptable risk of costly or dangerous defects.
>
> [Bob Binder]

Verification: ”Did we build the system right?”

- Discover situations in which the software behavior is incorrect or undesirable

Validation: “Did we build the right system?”

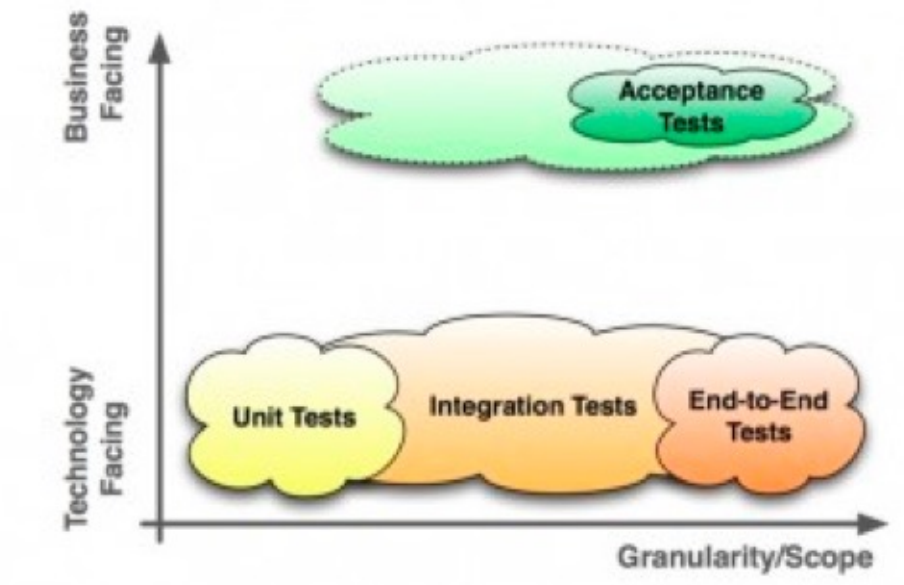
- Demonstrate that the software meets its requirements

Keep in mind…

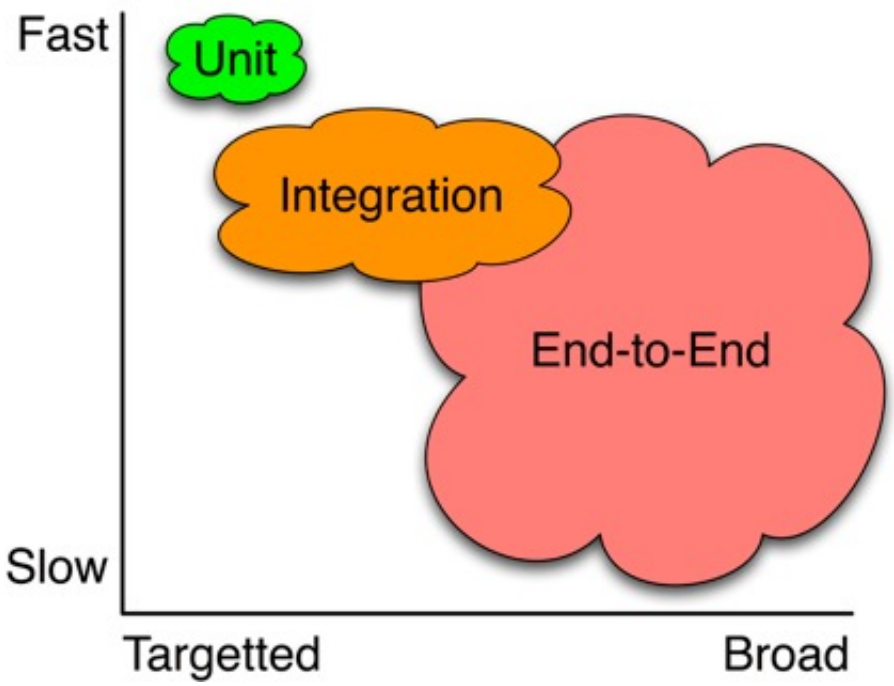
> “Testing can show the presence, but not the absence of errors.”
>
> [Dijkstra]

Types of Testing

- Unit
- Regression
- Integration
- System
- End-to-End / Acceptance



Test value



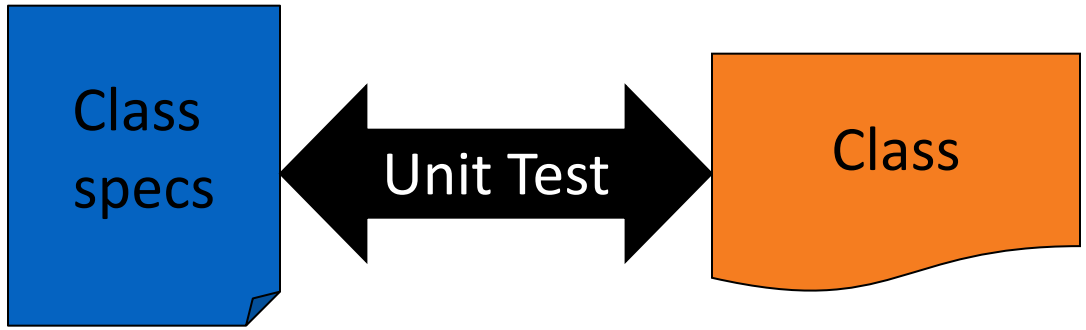
	Unit	End-toEnd
Fast		
Reliable		
Isolates Failures		
Simulates a Real User		

Unit Test

A method of testing that verifies the individual units of source code are working properly. A unit is the smallest testable part of an application.

- Process of testing individual components in isolation
- Units are usually methods or functions in isolation

- Typically written by the developer
- Frequently fully automatic



Class Exercise – Unit Test (3 mins) [https://bit.ly/3wlrnxO]

Identify and specify a good set of unit tests for a command line program that calculates prime factors

Prime factor for integer i, X: {n | i % n = 0, n prime}

i.e. pf(12) = {2,3}

Consider special cases

The anatomy of any test

To reveal a fault, a test must:

- Reach some code
- Trigger a defect
- Propagate an incorrect result
- The result must be observed
- The result must be interpreted as incorrect

Test threats:

- Non-deterministic dependencies
- Threading/Race Conditions/Deadlock
- Shared data

Each Unit Test Implementation

Each test:

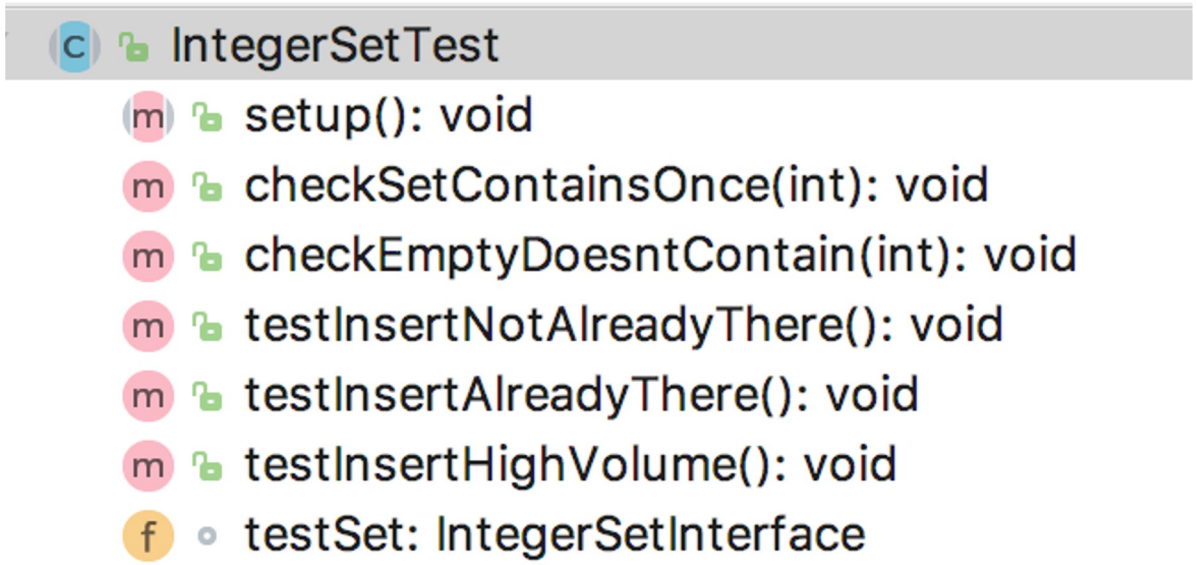
- A meaningful name that indicates the point of the test (just one thing!)
- A setup portion (sometimes this is refactored into the @Before code)
- Invocation of the code to be tested
- Check of the result against expectations, usually through assertions

```
@Test
public void testInsertNotAlreadyThere(){
   assertEquals(testSet.size(), 0);
   assertFalse(testSet.contains(num));

   testSet.insert(3);

   assertTrue(testSet.contains(num));
   assertEquals(testSet.size(),1);
}
```

Unit Test Suite

Test Suite:

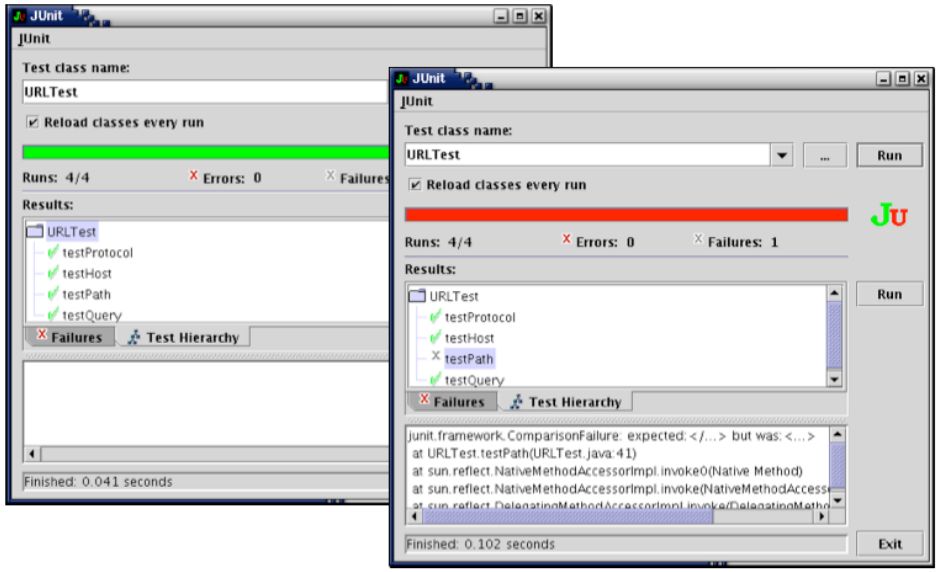
Contains the collection of tests for one project, sometimes broken down by component (like tests for a particular class)



Regression Test

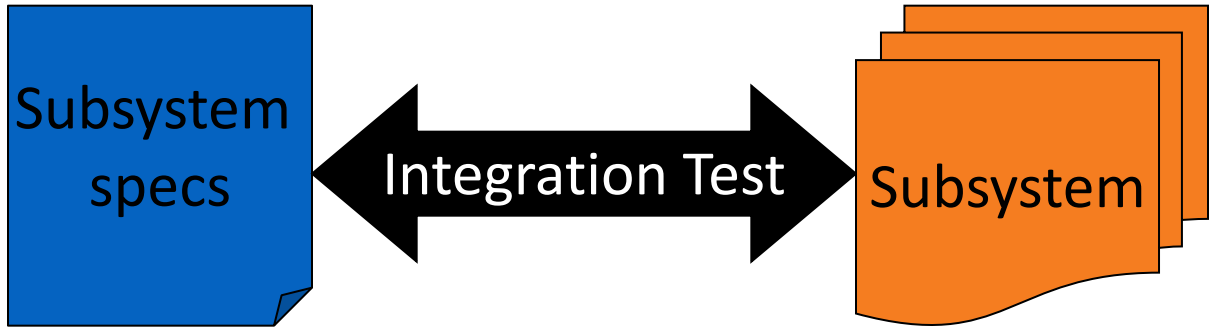
- Testing the system to check that changes have not ‘broken’ previously working code
- Not new tests, but repetition of existing tests
- Automated testing makes this simple, e.g., JUnit makes it easy to re-run all existing tests
- This is why it is so important

  not to write “throwaway tests”



Integration Test

- Exercise groups of components to ensure that their contained units interact correctly together
- Touch much larger parts of the system (major subsystems)
- Since these tests validate many different units at the same time, identifying root-cause of a test failure can be difficult



System Test

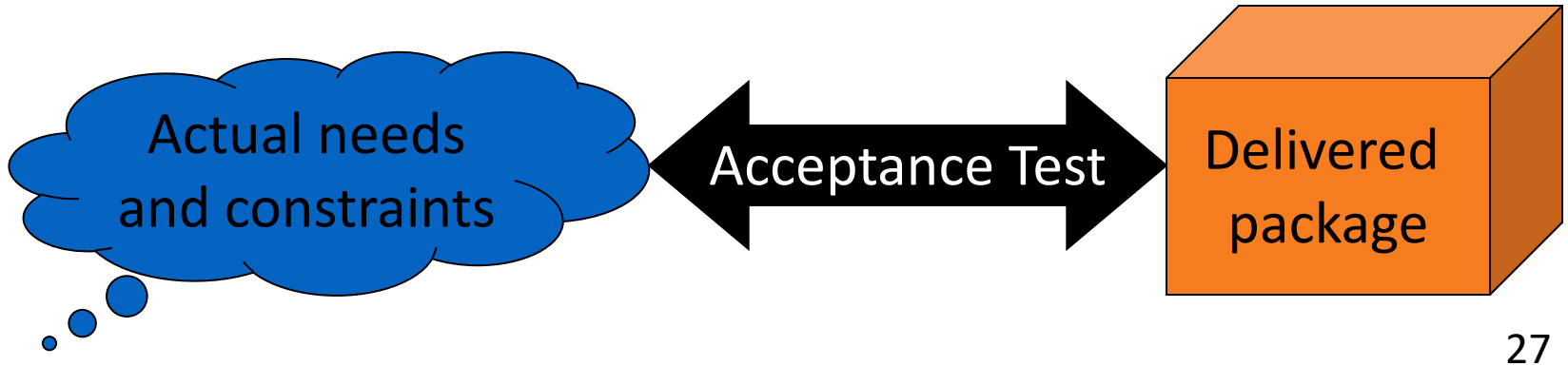
Test large parts of system, or whole system

Involves integrating components to create a version of the system

- Security testing
  - Verifies that protection mechanism will protect software against improper penetration
- Stress testing
  - Executes system in a manner that demands resources in abnormal quantity, frequency or volume
- Performance testing
  - Test run-time performance of the software in context of fully integrated system
- …

Acceptance Test

- Formal testing with respect to user needs and requirements to determine whether the software satisfies the acceptance criteria.
- Does the end product solve the problem it was intended to solve?
- Typically incremental:
  - Alpha test: at production site
  - Beta test: at user's site



Stopping Criteria

Learning Objectives

Be able to:

- Decide when to stop testing
- Given a method, be able to determine a ‘good’ and minimal test suite (e.g. using equivalence class partitioning or achieving a certain coverage criteria)

Testing

- When are you done?
- What constitutes a good test case / test suite?

Key questions for a test suite

Is the test suite:

sufficiently *broad / comprehensive*?

sufficiently *deep / effective*?

Test suite breadth – Code Coverage

- Intuition: the more parts are executed, the higher the chance that defects are uncovered
- Measures the proportion of the system that is executed by the test suite

Different coverage criteria:

- line coverage
- statement coverage
- branch coverage: each condition
- path coverage: each possible path through the code
- multiple condition coverage (MCC)

Statement coverage tools

Control Flow Graph Excursion

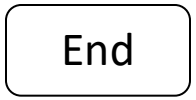
Learning Objectives

Be able to:

- create a control flow graph for a code snippet

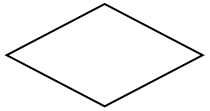
Excursion: Control Flow Graphs

Indicates the start of the control-flow



Indicates the end of the control-flow

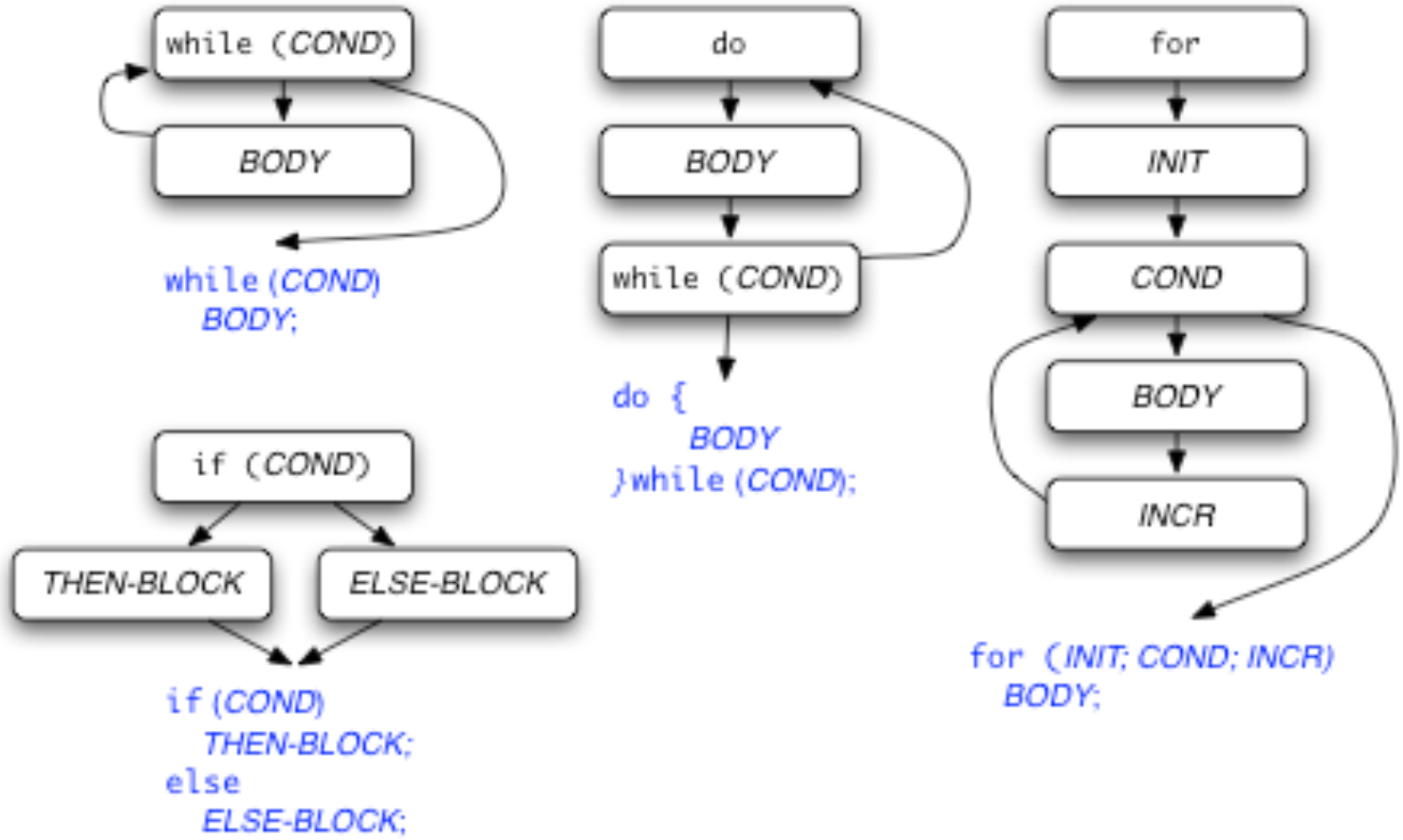
Indicates a processing step, text in the box is the code to execute



Indicates a conditional representing a yes/no question or true/false test. Two arrows emanate (one for yes/true and one for no/false) and must be labeled. The yes/true arrow typically comes out the bottom and the other out one of the sides.

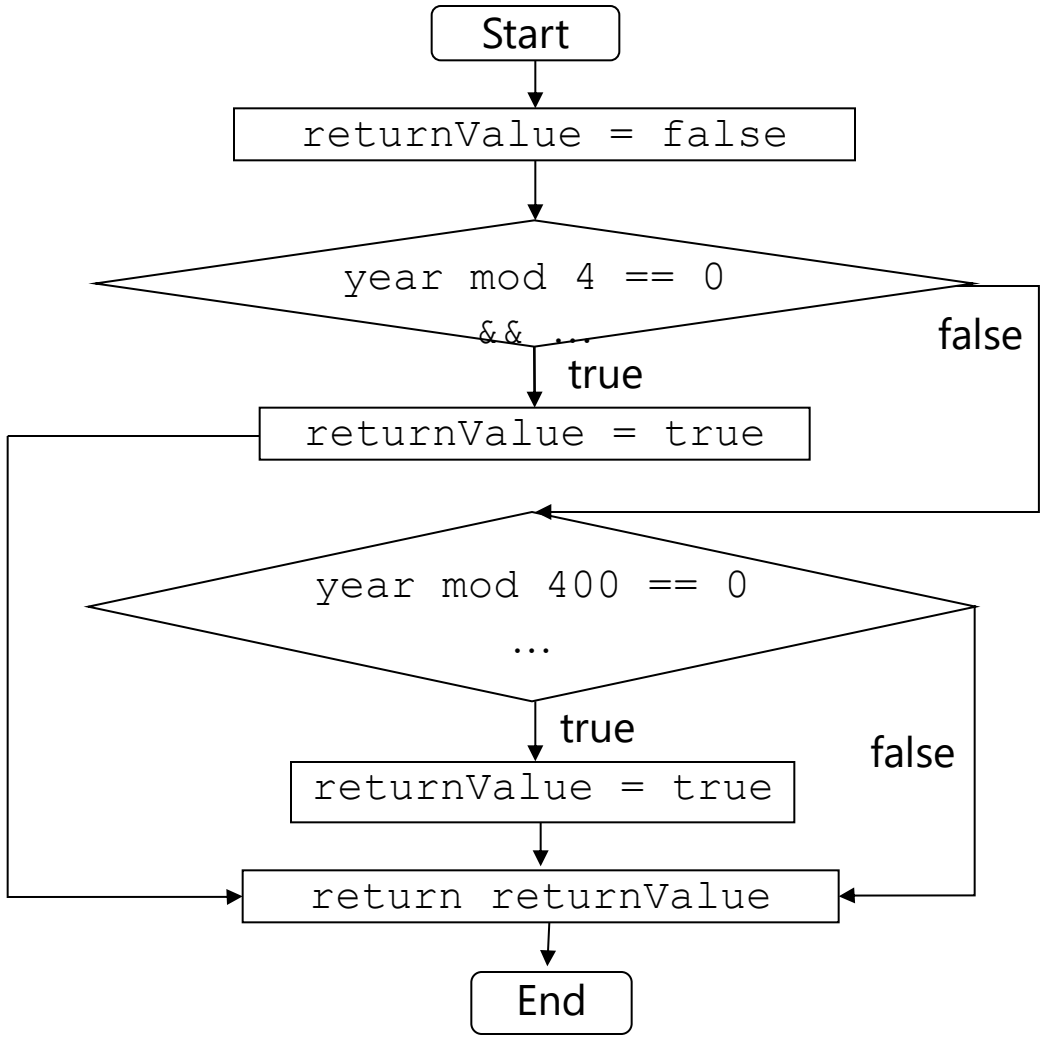
From Gail Murphy's CPSC 210

Excursion: Control Flow Patterns



Excursion: Control Flow Graph Example

```
boolean isALeapYear( int year ) {
   // Declare a variable for the return value
   // of the function
   boolean returnValue = false;

   // If the year is divisible by 4 and
   // not by 100 it is a leap year
   if ( ( year mod 4 == 0 ) &&
           ( year mod 100 != 0 ) )
      returnValue = true;
   else if ( year mod 400 == 0 )
      returnValue = true;
    return returnValue;
}
```



From Gail Murphy's CPSC 210

Stopping Criteria / Coverage continued

Learning Objectives

Be able to:

- Decide when to stop testing
- Given a method, be able to determine a ‘good’ and minimal test suite (e.g. using equivalence class partitioning or achieving a certain coverage criteria)

Class exercise on coverage: eval function [https://bit.ly/44x80yA]

```
int eval(int x, boolean c1, boolean c2){
    if (c1)
        x++;
    if (c2)
        x--;
    return x;
}
```

In small groups or by yourself:

1. Does *assertEquals(eval(0,false,false),0);* achieve 100% statement coverage?
2. How many test cases are needed for statement coverage and which one(s)?
3. Specify a test suite that achieves branch coverage.
4. Specify a test suite that achieves path coverage.

Statement coverage:

```
int eval(int x, boolean c1, boolean c2){
    if (c1)
        x++;
    if (c2)
        x--;
    return x;
}
```

Test Suite

`eval(0, false, false)`

60%

Statement coverage with one test case

```
int eval(int x, boolean c1, boolean c2){
    if (c1)
        x++;
    if (c2)
        x--;
    return x;
}
```

Test Suite

`eval(0, true, true)`

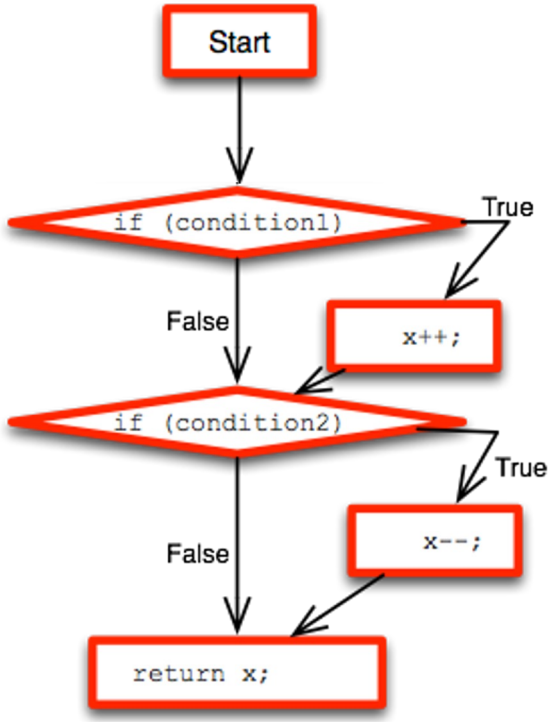
100%

Statement coverage with one test case

```
int eval(int x, boolean c1, boolean c2){
    if (c1)
        x++;
    if (c2)
        x--;
    return x;
}
```

Test Suite

`eval(0, true, true)`



100%

Branch Coverage?

```
int eval(int x, boolean c1, boolean c2){
    if (c1)
        x++;
    if (c2)
        x--;
    return x;
}
```

Test Suite

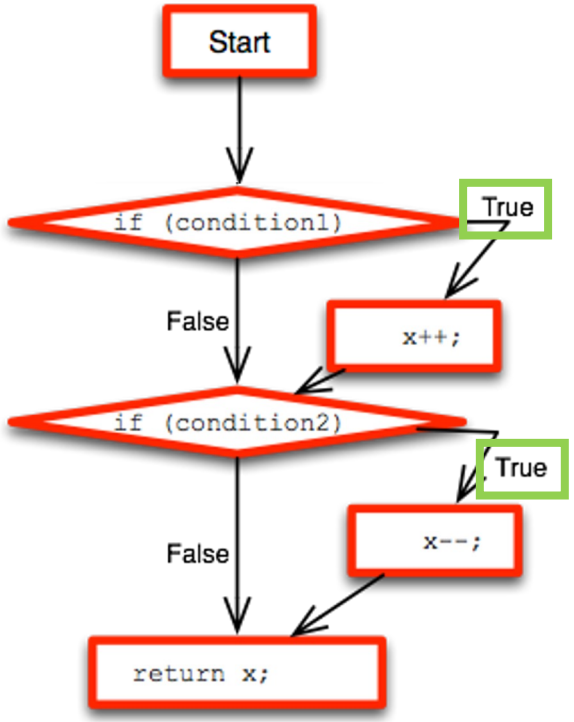
`eval(0, true, true)`

Branch Coverage?

```
int eval(int x, boolean c1, boolean c2){
    if (c1)
        x++;
    if (c2)
        x--;
    return x;
}
```

Test Suite

`eval(0, true, true)`



50%

Branch Coverage

```
int eval(int x, boolean c1, boolean c2){
    if (c1)
        x++;
    if (c2)
        x--;
    return x;
}
```

Test Suite

```
eval(0, true, true)
eval(0, false, false)
```

100%

Path Coverage?

```
int eval(int x, boolean c1, boolean c2){
    if (c1)
        x++;
    if (c2)
        x--;
    return x;
}
```

Test Suite

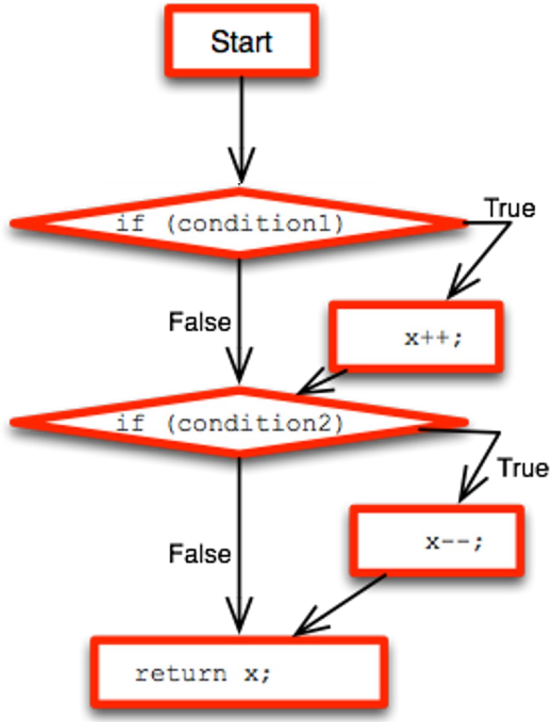
```
eval(0, true, true)
eval(0, false, false)
```

Path Coverage?

```
int eval(int x, boolean c1, boolean c2){
    if (c1)
        x++;
    if (c2)
        x--;
    return x;
}
```

Test Suite

```
eval(0, true, true)
eval(0, false, false)
```



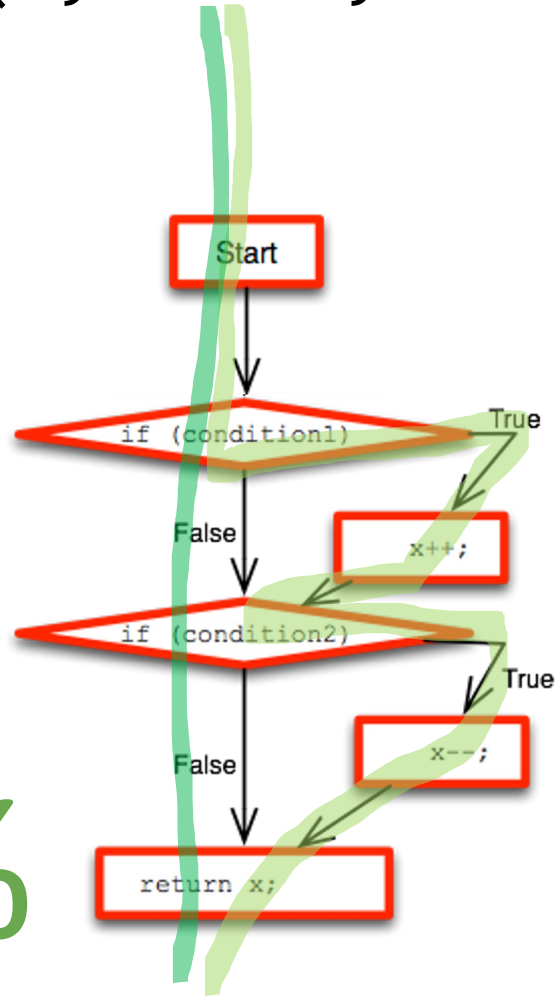
50%

Path Coverage?

```
int eval(int x, boolean c1, boolean c2){
    if (c1)
        x++;
    if (c2)
        x--;
    return x;
}
```

Test Suite

```
eval(0, true, true)
eval(0, false, false)
```



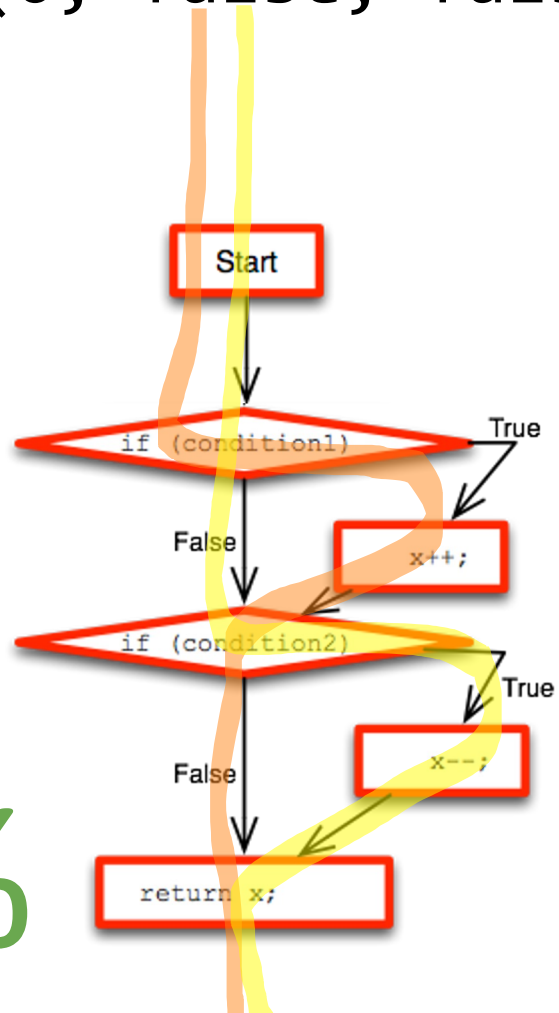
50%

Path Coverage?

```
int eval(int x, boolean c1, boolean c2){
    if (c1)
        x++;
    if (c2)
        x--;
    return x;
}
```

Test Suite

```
eval(0, true, true)
eval(0, false, false)
```



50%

Path Coverage

```
int eval(int x, boolean c1, boolean c2){
    if (c1)
        x++;
    if (c2)
        x--;
    return x;
}
```

Test Suite

```
eval(0, true, true)
eval(0, false, false)
eval(0, true, false)
eval(0, false, true)
```

100%

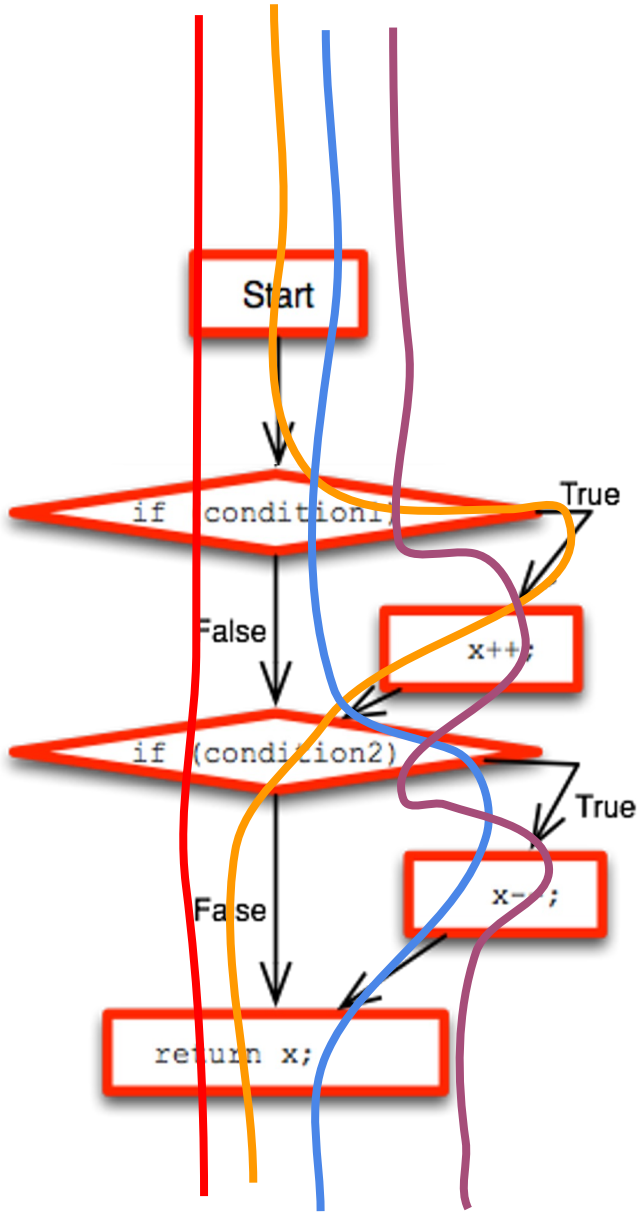
Path Coverage

```
eval(0, false, false)
```

```
eval(0, true, false)
```

```
eval(0, false, true)
```

```
eval(0, true, true)
```



Code Coverage & minimal test suite size

100% Statement

```
eval(0, true, true)
```

100% Branch

```
eval(0, true, true)
eval(0, false, false)
```

100% Path

```
eval(0, true, true)
eval(0, false, false)
eval(0, true, false)
eval(0, false, true)
```

Multiple Condition Coverage (MCC)

All combinations of conditions inside each decision are tested, i.e. all combinations of true and false for the conditions in the decision have to be tested

```
if ((x > 10 || y > 20) && z > 0)
        return true;
    else
        return false;
```

Test all true/false combinations of ‘x > 10’, ‘y > 20’, and ‘z > 0’

→ 2^3 tests

Code Coverage

- Pros
  - Actionable
  - Cheap (except for path coverage that can become infeasible, e.g. while loop)
  - Intuitive
- Correctness?
- Effectiveness?
- Coverage as stopping criteria is good, but ***smart*** testing is always better

What if you don’t know the code?

Black Box – Functional

- Tests based on *spec*
- Treats system as atomic
- Covers as much *specified* behaviour as possible
- Test criteria: cover input space

White Box – Structural

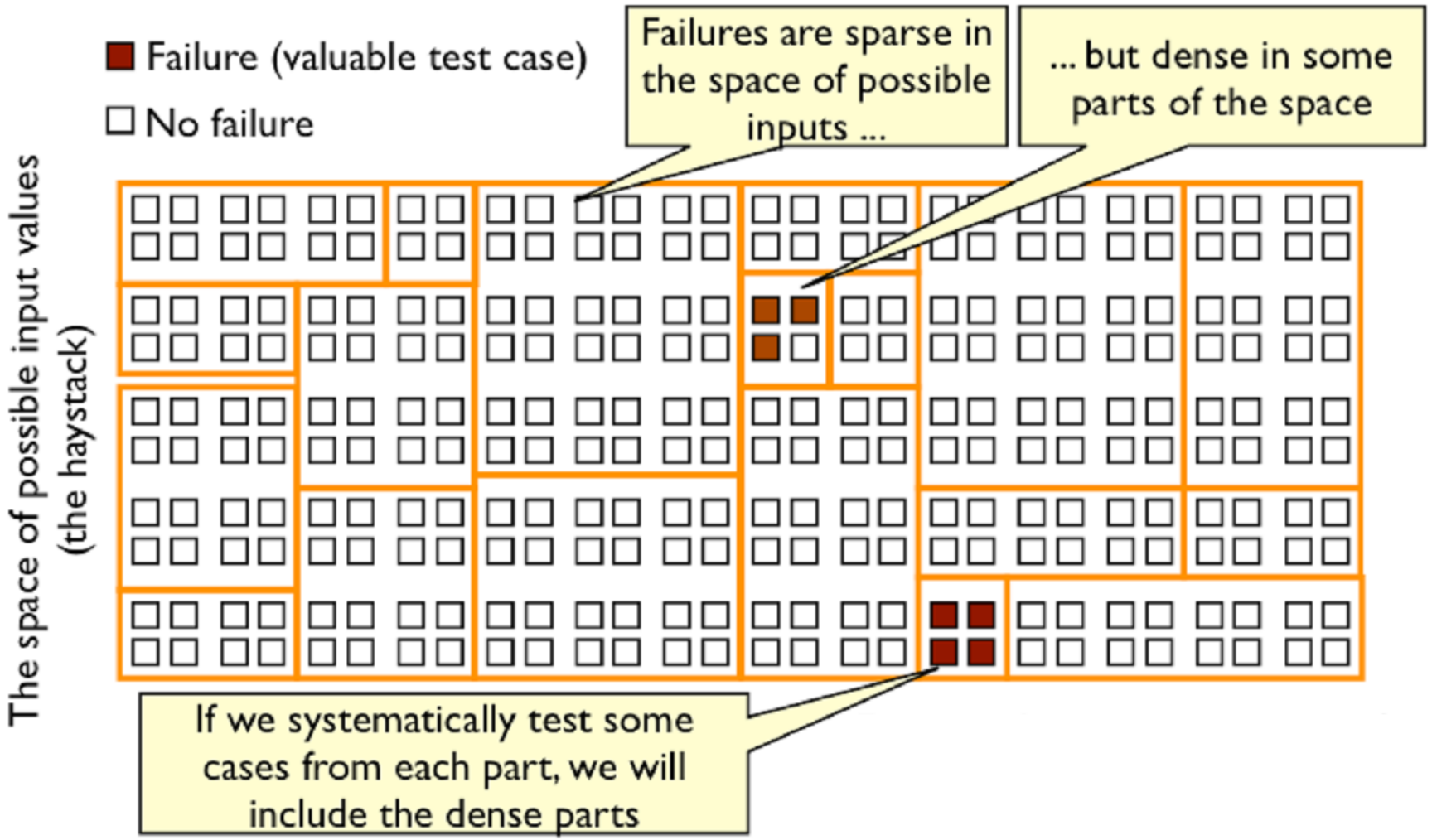
- Tests based on *code*
- Examines system internals
- Covers as much *implemented* behaviour as possible
- Test criteria: cover execution paths

Stopping Criteria (breadth) – black box

- Equivalence class partitioning
  - Convert a continuous input (output) space into something manageable
  - Identify/guess which types of inputs are likely to be processed in a similar way

- Each test should exercise one and only one equivalence partition

Equivalence Class Partitioning



Example of ECP

- System asks for numbers between 100 and 999
- Equivalence partitions:
  - Less than 100
  - Between 100 and 999
  - More than 999
- Three tests:
  - 50, 500, 1500

Example of equivalence classes

If the specification had said:

```
//REQUIRES: positive value as input (meaning we don't need to test negative values)
//EFFECTS:  if value is 0-6 throw ExceptionA
//          if value is 7-20 throw ExceptionB
//          if value is over 20 do the right thing
```

then to hit all the equivalence classes, you'd need to pass in something between 0 and 6, something between 7 and 20, and something over 20, and then test that the right things happened. But you wouldn't have one test where you pass in 4, and another one where you pass in 5. Because clearly that's going to result in the same output, and likely will run the same code and won't help with your coverage.

REQUIRES *means a strong assumption, like "user is logged in". So you don't guarantee any behaviour, and potentially can't even check whether the state is correct -- the method is just hoping it is! A really classic example is Binary Search Tree's search function. It assumes the tree is sorted. But it can't check (too expensive). So there's no point testing whether search works on an unsorted tree, because who knows what the "right" response should be! This would have been just as effective an example if it had no requires clause -- in that case you would need to check -1 (for the boundary) and -10 (for the equivalence class of everything negative).*

Equivalence classes

```
// REQUIRES: non-null inputs
// EFFECTS:  permits passwords that are between 8 and 20 chars,
//           and that contain a special character [“!”, “@”, “#”, “&”]
```

```
private final List<String> SPECIAL_CHARSET = List.of(“!”, “@”, “#”, “&”);

public boolean isValidPassword(String password) {
    if (password == null) {
        return false;
    }
    if (password.length() < 8 || password.length() > 20) {
        // passwords should have a length between 8 and 20 characters
        return false;
    }
    for (String s : SPECIAL_CHARSET) {
        // passwords should contain at least one character from
        // SPECIAL_CHARSET
        if (password.contains(s)) {
            return true;
        }
    }
    return false;
}
```

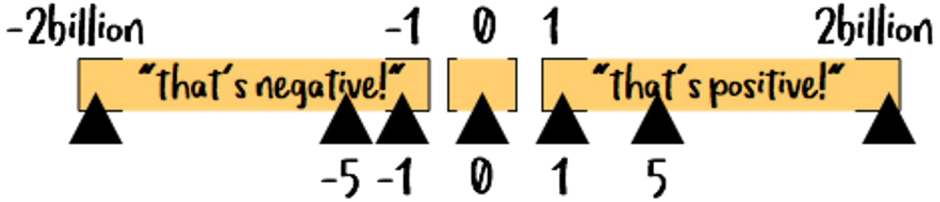
On purpose small code, since in black box testing you don’t have this!

What are good tests using ECP?

Boundary testing

- Test inputs inputs starting from known good values and progressing through reasonable but invalid to known extreme and invalid
  - E.g. max/min, just inside/outside boundaries, typical values and error values
- Like ECP but looking specifically at edge cases

```
//MODIFIES: this
//EFFECTS: inserts the integer into the set, unless it's
//already there, in which case it does nothing.
//if the number is 0 it returns "that's nothing!"
//if the number is <0 it returns "that's negative!"
//if the number is >0 it returns "that's positive!"
public String insert(Integer num){...}
```



Revisiting for boundary checks

If the specification had said:

```
//REQUIRES: positive value as input (meaning we don't need to test negative values)
//EFFECTS:  if value is 0-6 throw ExceptionA
//          if value is 7-20 throw ExceptionB
//          if value is over 20 do the right thing
```

You would want to hit all the boundaries (and all around the boundaries) to make sure you didn't have a typo in your < > ='s, so 0,1,5,6,7,8,19,20,21

Why don’t we test <0?

Mutation Testing (Effectiveness of Tests)

Learning Objectives

Be able to:

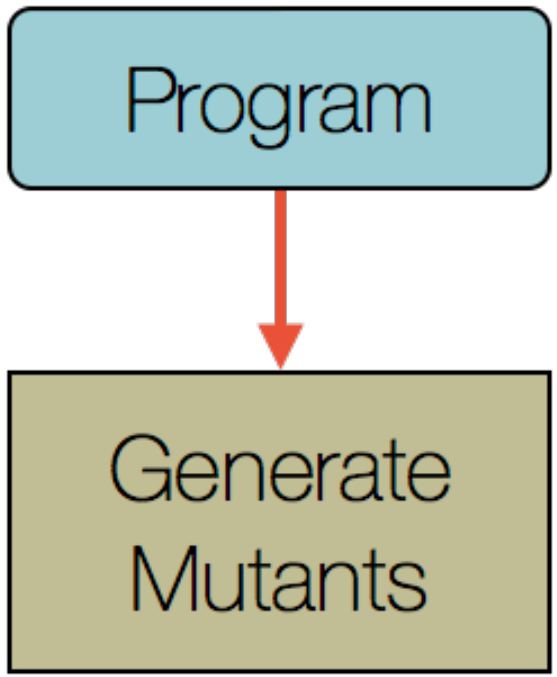
- Explain mutation testing
- define test cases and calculate their kill score

Test suite depth – Mutation Testing

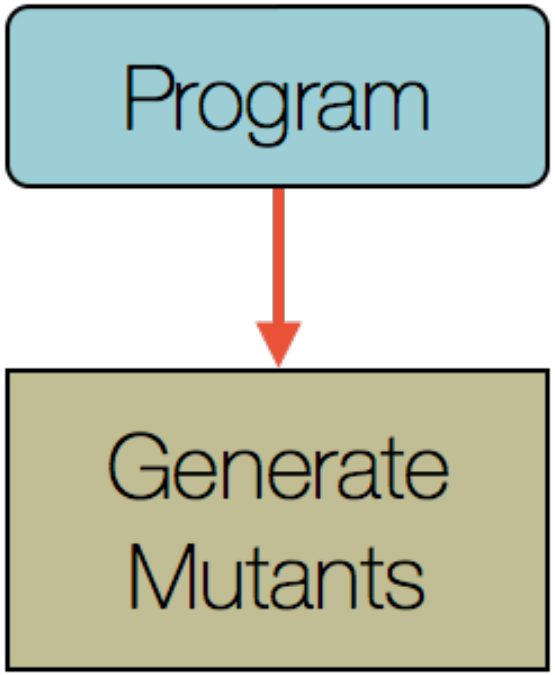
- Goal: ensure that a test suite is able to detect faults (test suite quality)

- Modify program in small ways and examine if test suite is able to find the errors/mutants (killing a mutant)
- Mutation operators mimic typical programming errors
  - Flip boolean, flip boundaries (<,>=, …), remove/negate conditional, flip mathematical operators, etc.

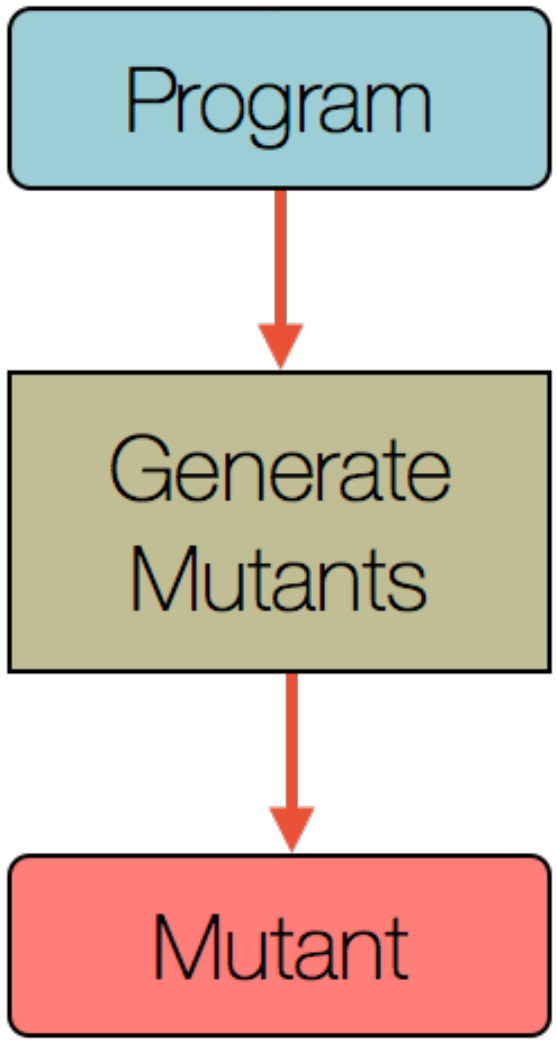
- Effectiveness of your test suite is determined by the percentage of mutants killed




```
public float avg(float[] data){
    float sum = 0;
    for (float num : data){
        sum += num;
    }
    return sum / data.length;
}
```

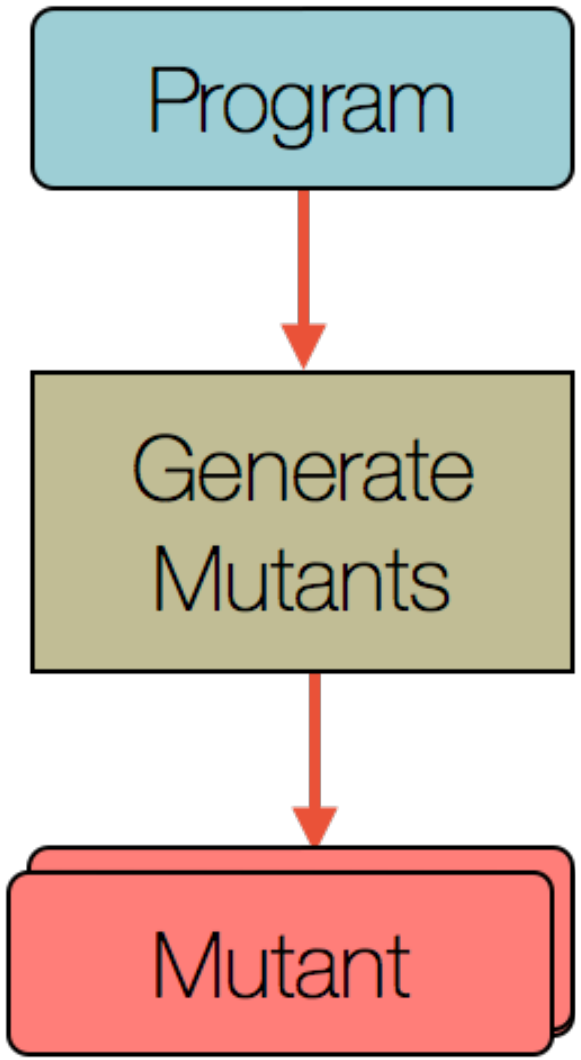



```
public float avg(float[] data){
    float sum = 0;
    for (float num : data){
        sum += num;
    }
    return sum / data.length;
}
```




```
public float avg(float[] data){
    float sum = 0;
    for (float num : data){
        sum += num;
    }
    return sum / data.length;
}
```

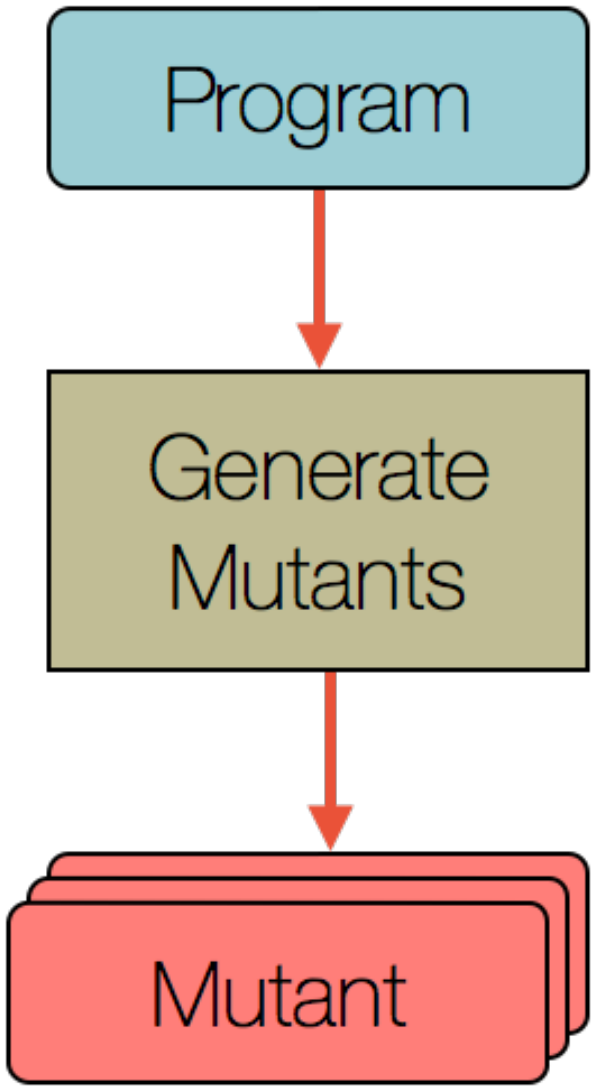
```
public float avg(float[] data){
    float sum = 1;
    for (float num : data){
        sum += num;
    }
    return sum / data.length;
}
```




```
public float avg(float[] data){
    float sum = 0;
    for (float num : data){
        sum += num;
    }
    return sum / data.length;
}
```

```
public float avg(float[] data){
    flo
    for
    }
    ret
}
```

```
public float avg(float[] data){
    float sum = 0;
    for (float num : data){
        sum += num;
    }
    return sum * data.length;
}
```

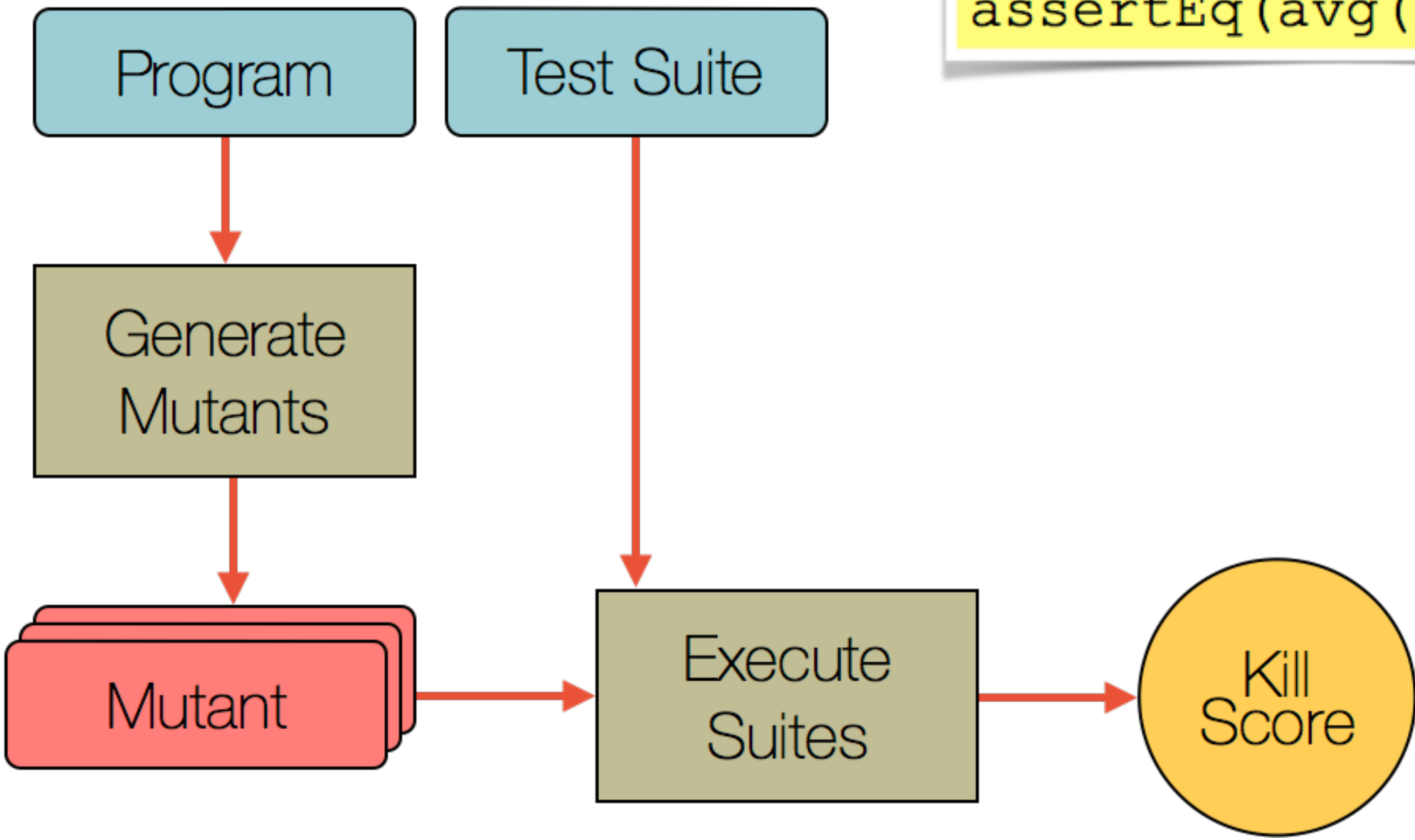



```
public float avg(float[] data){
    float sum = 0;
    for (float num : data){
        sum += num;
    }
    return sum / data.length;
}
```

```
public float avg(float[] data){
    float sum = 0;
    for (float num : data){
        sum -= num;
    }
    return sum / data.length;
}
```

Test suite:

```
assertEq(avg([1]), 1);
```



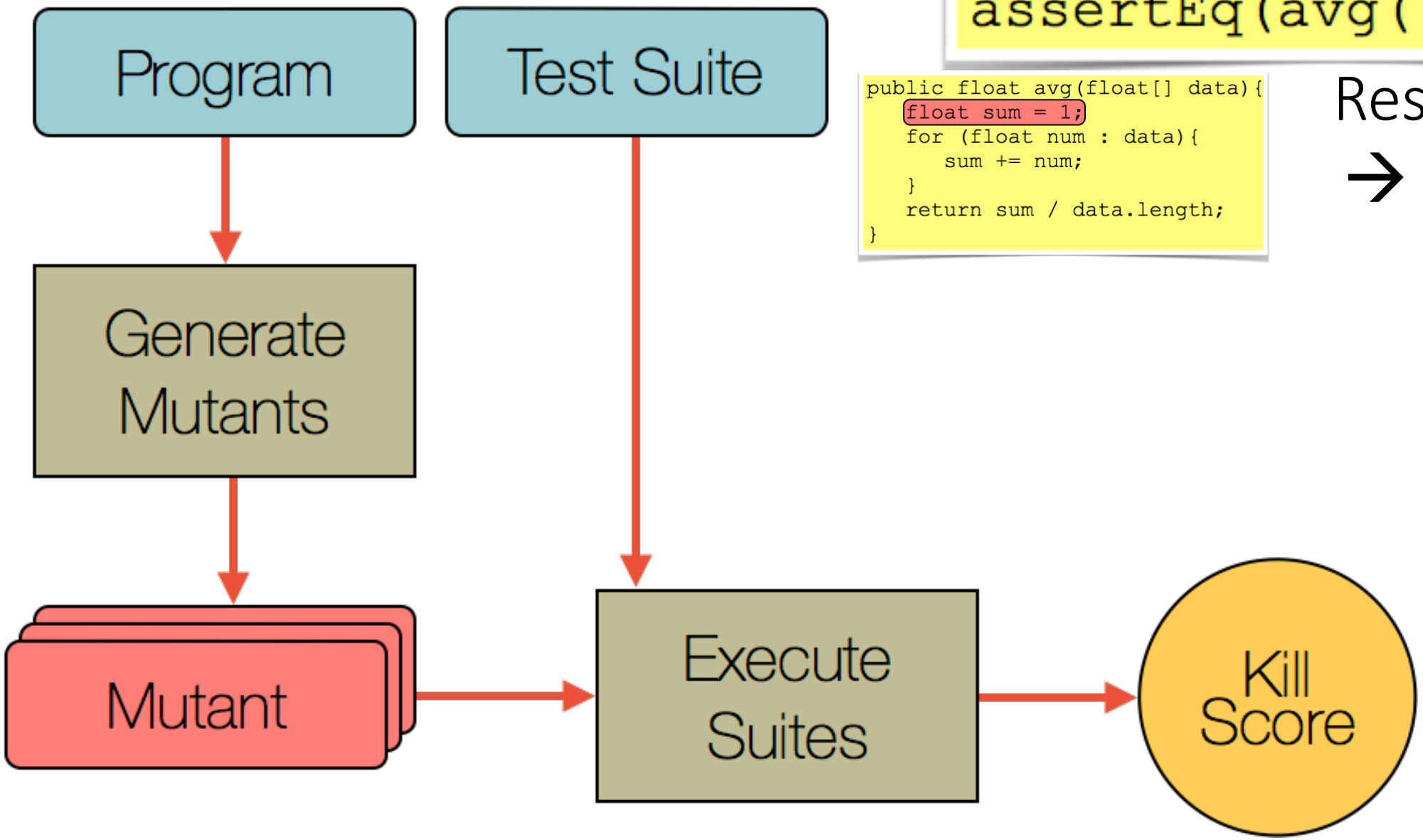
Test suite:

```
assertEq(avg([1]), 1);
```

```
public float avg(float[] data){
    float sum = 1;
    for (float num : data){
        sum += num;
    }
    return sum / data.length;
}
```

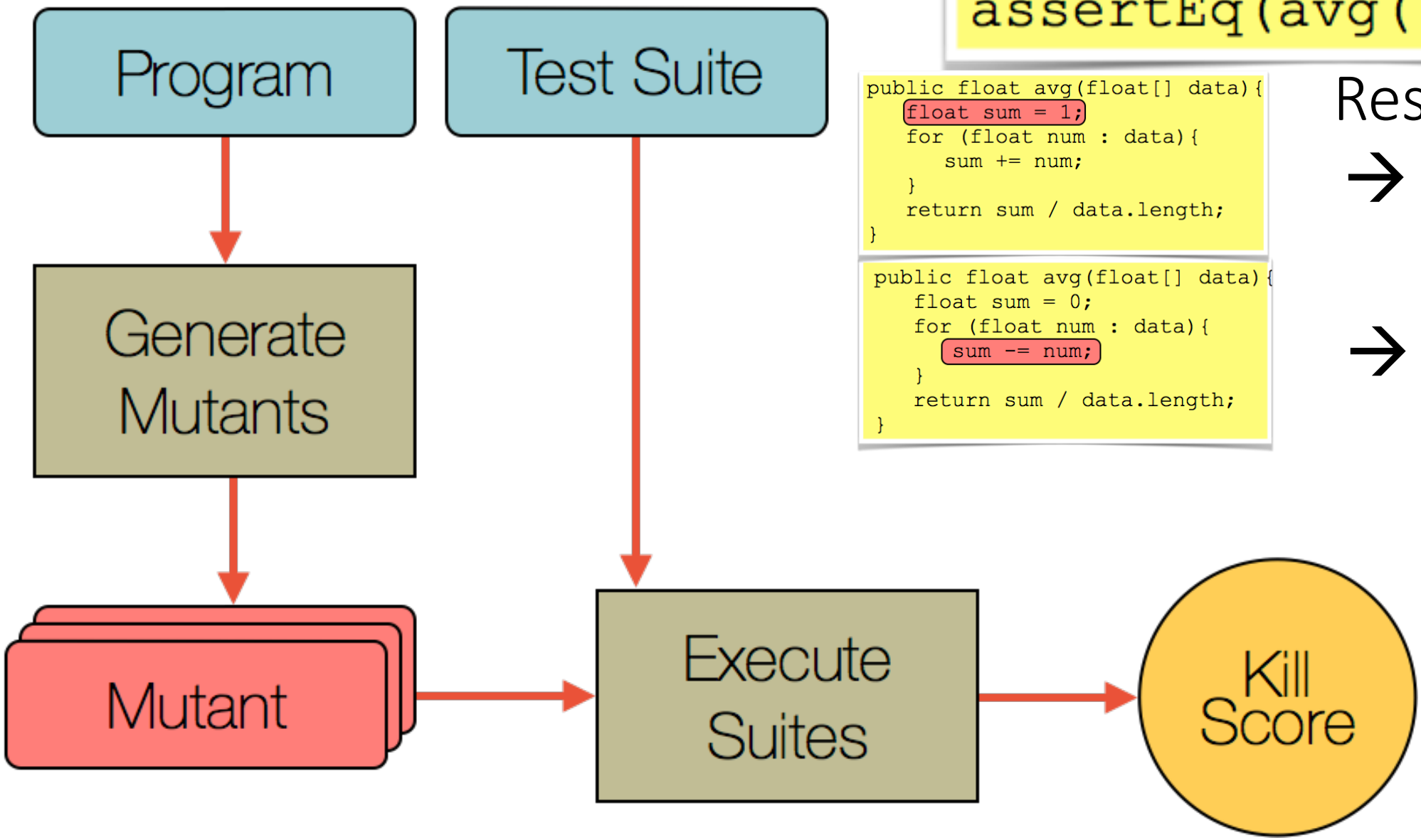
Result

→ 2 X



Test suite:

```
assertEq(avg([1]), 1);
```



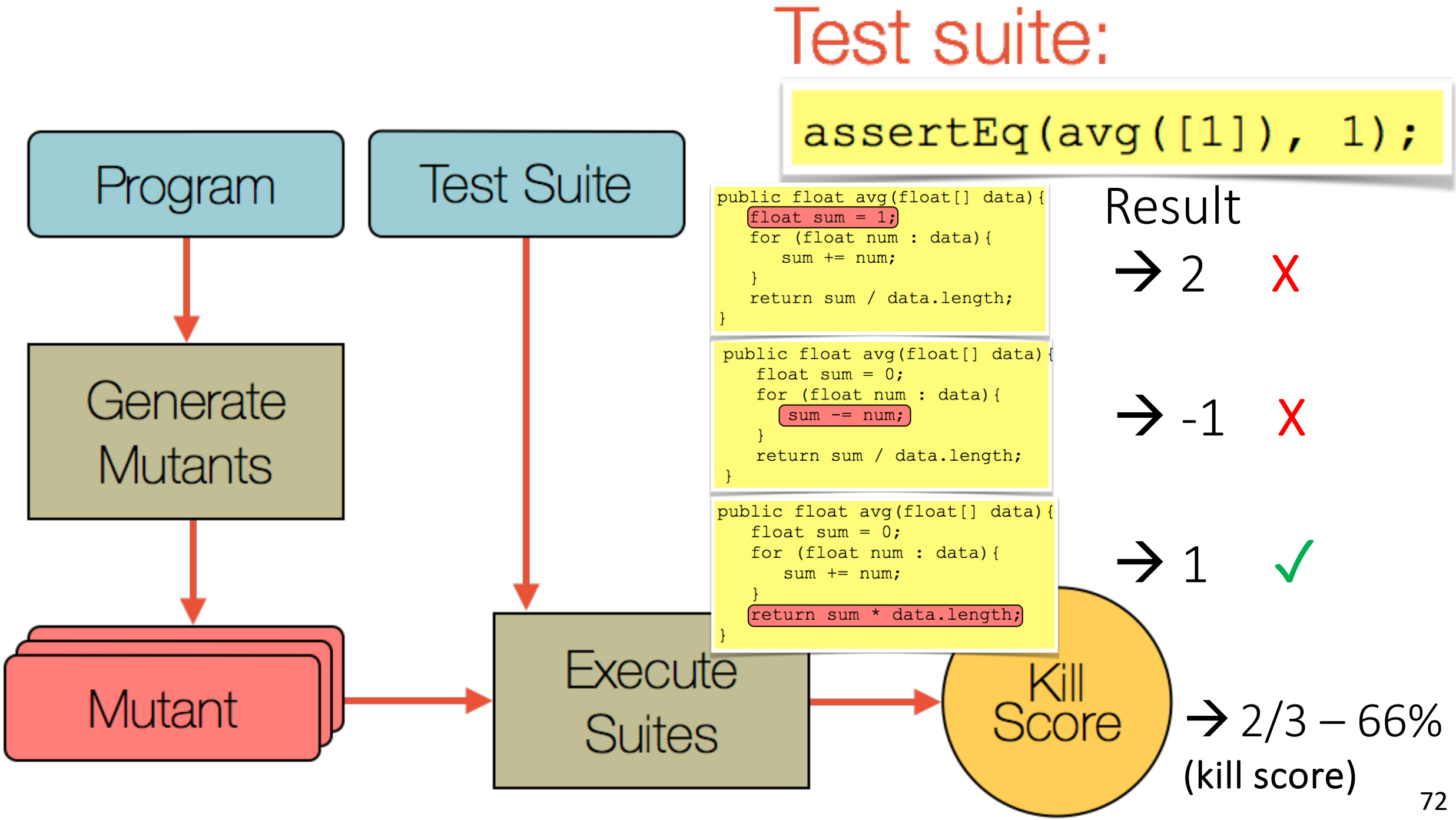

```
public float avg(float[] data){
    float sum = 1;
    for (float num : data){
        sum += num;
    }
    return sum / data.length;
}
```

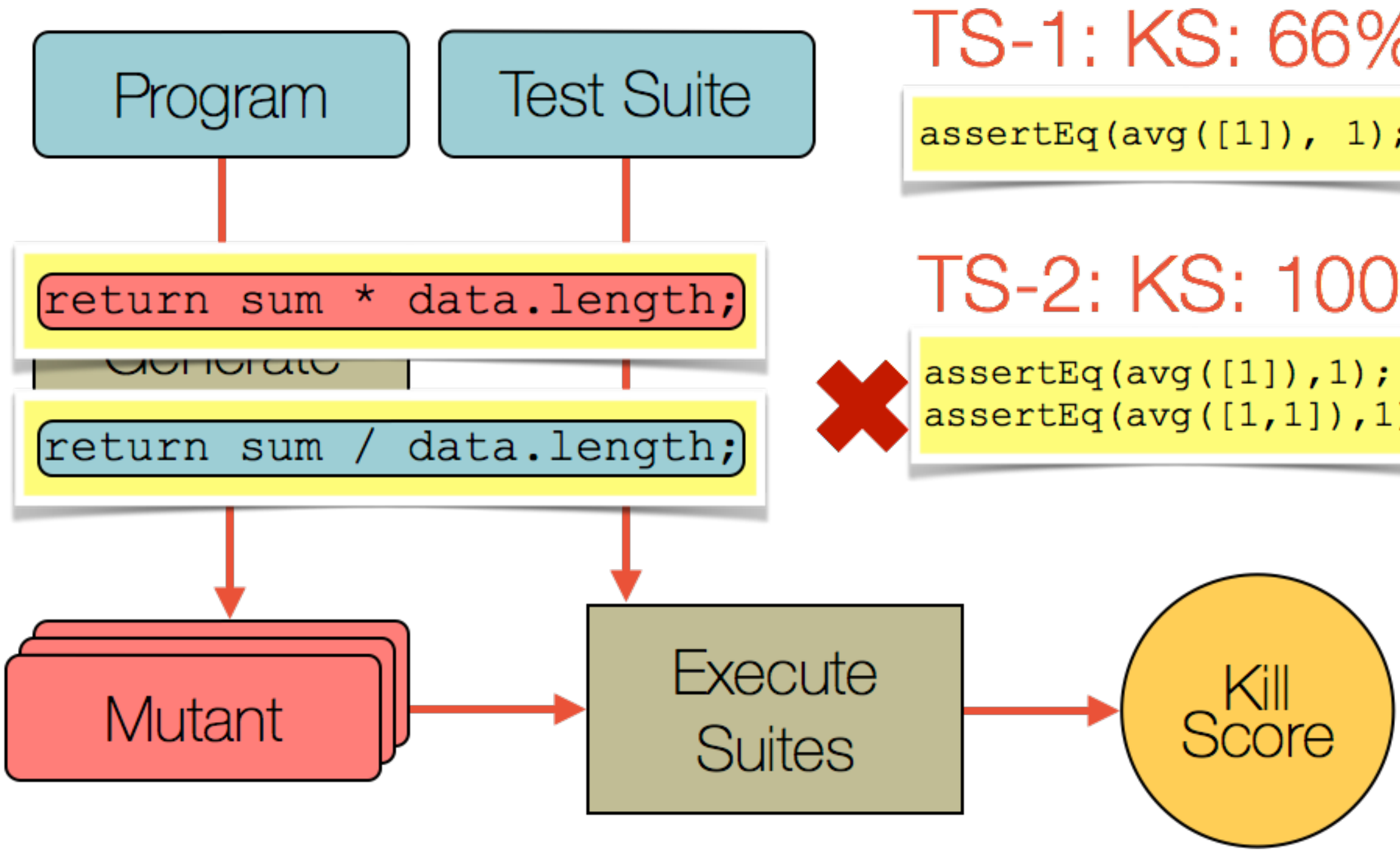
```
public float avg(float[] data){
    float sum = 0;
    for (float num : data){
        sum -= num;
    }
    return sum / data.length;
}
```

Result

→ 2 X

→ -1 X


Test suite:
assertEq(avg([1]), 1);
Program
Test Suite
Generate Mutants
Mutant
Execute Suites
Kill Score
public float avg(float[] data){
float sum = 1;
for (float num : data){
sum += num;
}
return sum / data.length;
}
public float avg(float[] data){
float sum = 0;
for (float num : data){
sum -= num;
}
return sum / data.length;
}
public float avg(float[] data){
float sum = 0;
for (float num : data){
sum += num;
}
return sum * data.length;
}
Result
→ 2 X
→ -1 X
→ 1 ✓
→ 2/3 – 66%
(kill score)


Program
Test Suite
return sum * data.length;
return sum / data.length;
Mutant
Execute Suites
Kill Score
TS-1: KS: 66%
assertEq(avg([1]), 1);
TS-2: KS: 100%
assertEq(avg([1]),1);
assertEq(avg([1,1]),1);

Mutation testing

- Pros
  - Programmatic Oracle
  - Correctness focus
- Cons
  - Synthetic?
  - Time to execute

Additional Coverage Example

Class Exercise (if time available) [https://bit.ly/4a4U0wV]


```
float calcRentalFee(Movie[] movies, Customer c){
      float total = 0;
      for(int i=0; i<movies.length; i++){
            total += movies[i].getRentalPrice();
      }
      if (movies.length > 10){
            total *= .8;
      } else if(movies.length > 5){
            total *= .9;
      }

      if(c.isPremium()){
            total *= .9;
      }
      return total;
}
```

What is a minimal test suite to achieve branch coverage? Only specify movies.length and c.isPremium for the test suite.

from Laurie Williams

Quiz & More

Additional References

Testability Reading and videos by Reid:

- https://github.com/ubccpsc/310/blob/master/resources/readings/TestabilityAssertions.md
- https://github.com/ubccpsc/310/tree/master/resources (section 4)

Mocks aren't stubs (Martin Fowler):
https://martinfowler.com/articles/mocksArentStubs.html

Quiz

1. Given the following code snippet and the 3 assertions/test cases below, decide for each statement whether it is true with respect to the isGreatNumber method.

```
public boolean isGreatNumber(int num) {
    if (num <= 1) {
        return false;
    } else if (num % 100 == 0) {
        return true;
    }
    return false;
}
```

Test cases:

```
// Assertions:
assert isGreatNumber(2) == false;
assert isGreatNumber(7) == false;
assert isGreatNumber(1200) == true;
```

The test cases / assertions achieve:

A) 100% line coverage.

B) More than 70% branch coverage.

C) More than 60% path coverage.

Quiz

2. For each of the following statements, assess whether it is true or false:

   In White Box Testing, testers:

   a) Focus on the functionality of the software without looking at the internal implementation.
   b) Examine the internal code structure and implementation details of the code.
   c) Only perform testing after the software is deployed to production.

3. Choose whether the following statement is true or false: “System tests generally offer good failure isolation so that it is easy to locate where a failure occurs.” [True/False]

Quiz

4. Assess whether the following statement is true or false: “Code reviews are a common way in industry to find defects in the code before it is being deployed.” [True / False]

Midterm

Section – Requirements

A group of innovative minds from UZH, including you, identified a significant challenge in the local restaurant industry: the gap between restaurants wanting to reduce food waste and environmentally conscious consumers eager to support sustainable practices. Your team wants to create an innovative application that connects consumers with restaurants offering surplus food at a discounted price, thus promoting sustainability while offering great deals. For the development of this application, you created and prioritized the following user stories for one of the sprints.

Section – Requirements

U1: As a user, I want to be able to check participating restaurants and the surplus food items each of them offers in order to buy an item and support local businesses that are engaged in reducing food waste. *Definition of Done:*

1. User can access a map or list of participating restaurants by clicking a "Check participating restaurants" button.
2. User can see details such as the names and addresses of participating restaurants.
3. User can select a restaurant and see the available surplus food items, and discounted prices.
4. User can select a surplus food item and purchase it.

U2: As a user, I can purchase an available surplus food item from a restaurant in order to pick it up from the restaurant and eat it.

Definition of Done:

1. User selects an available surplus food item from a restaurant.
2. The item will be put into the shopping cart of the user.
3. User can select "buy" to purchase the item.
4. Restaurant will be notified so that the item will be reserved and prepared for pick up by the user.

For each of the following statements, please state whether it is true or false.

1. The user story U1 addresses a functional requirement.
2. The user story U1 follows the role-goal-benefit format.
3. The user story U1 is independent according to INVEST. [False: yellow part overlaps and U1 depends on U2]
4. The user story U1 is testable according to INVEST.

Section – Requirements

U3: As a user, I want to be able to rate on any of my completed orders using a 5 star rating and a text field labelled "Comment" for a maximum of 200 characters. *Definition of Done:*

1. Once an order is completed, a user is asked to leave a rating via a push notification.
2. User can choose to leave a rating of 1 to 5 stars for the completed order.
3. User has a text field labeled "Comment" and can write a comment with a maximum of 200 characters.
4. User can press "Submit" button to submit the rating and comment.
5. User can see the rating and comment when they select the restaurant the order was from.

U4: As a restaurant owner I want to be able to specify whether users are able to provide comments for their ratings in order to avoid negative comments. *Definition of Done:*

1. Restaurant Owner (RO) can access their account page and select whether comments should be enabled or not for the customer ratings.

For each of the following statements, please state whether it is true or false.

- The user story U3 is negotiable according to INVEST **[False: text field labelled "Comment", maximum of 200 characters; a good story captures essence not details]**
- The user stories for the first sprint (U1-U5) are consistent. **[False: inconsistency between U3 and U4]**
- To validate your requirements, it would be best to transform them into a more formal software requirements specification. **[False: user stories can be validated with users especially since they are already written in the users' language so no need for transformation]**

Section – Decomposing user stories

You are part of a small development team that is asked to design a Pet Care Application designed for pet owners. The application aims at supporting pet owners to keep track of their pet's food intake, sleep schedule and other health related information, such as health records from veterinarians. The app should also support pet owners to schedule appointments with veterinarians and allow the veterinarian to access the information on the pet as well as upload further health records.

Since you want to make sure that you and your team members are on the same page about the direction of the app, you have started to sketch out some of the requirements in the form of user stories. Note that the user stories are not complete yet.

U1. As a pet owner, I want to be able to add a pet with its type, name and birthdate to my pet owner profile in order to keep track of the pets I own.

U2. As a pet owner, I want to be able to add a daily log entry for any of my pets and specify the daily food intake and the sleep schedule for the pet, so I can keep track of relevant information and maintain a consistent routine that supports the pet's overall health and well-being.

U3. As a pet owner, I want to be able to share my pet's information with a veterinarian in order for them to have access to the information if they need to for the medical examination.

U4. As a veterinarian I want to be able to add a medical examination record for a pet based on the medical examination in order for the pet owner to better understand what's going on.

U5. As a pet owner, I want to be able to get a list of the medical examination records that a veterinarian shared with me for each pet associated with my profile, so that I can better understand what's going on with my pet and how to treat it.

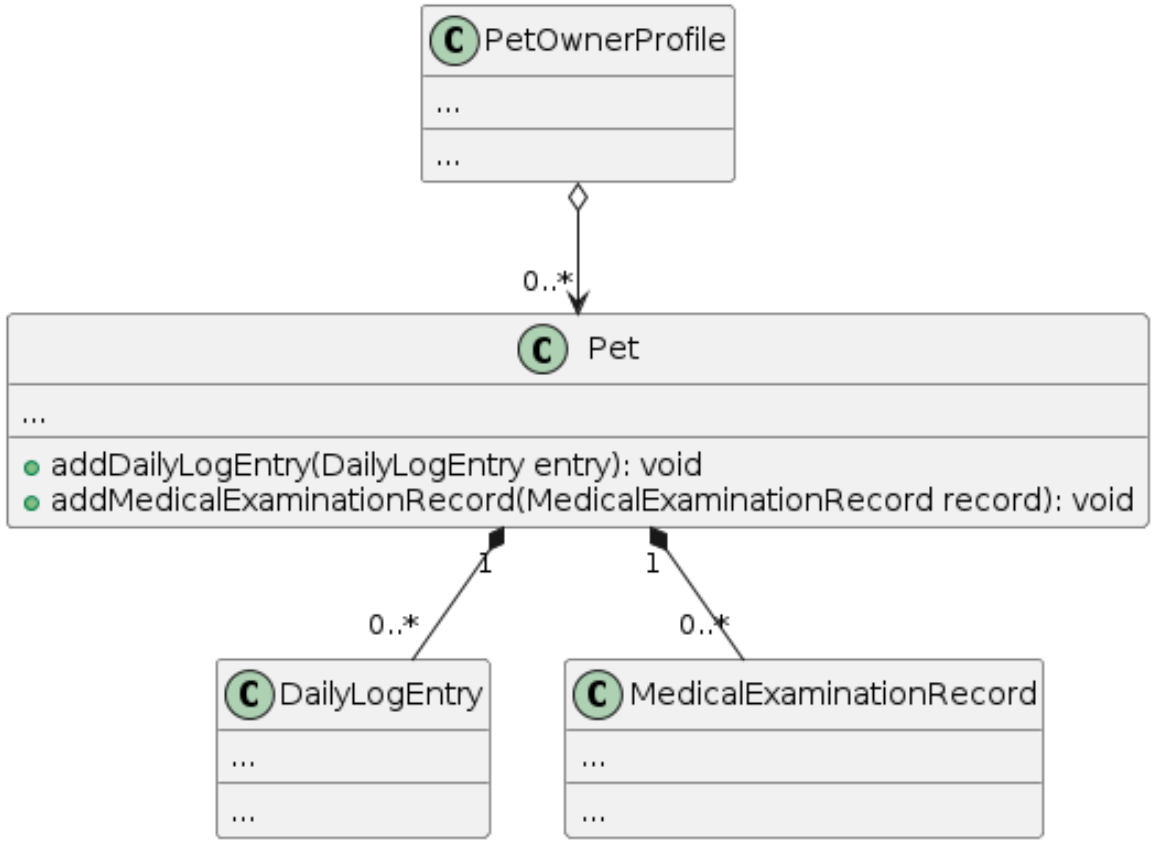
Section – Decomposing user stories

You now want to decompose these user stories achieving good modularity with low coupling and high cohesion. You first want to translate the user stories into a class diagram and then depict just some important parts as a sequence diagram. Based solely on the user stories, the description, and your own class and sequence diagram, assess for each of the following statements whether it is correct (true) or not (false). (Note: It may help you to sketch a class and sequence diagram first to answer the questions.)

1. In a class diagram, there should be a class Pet, a class DailyLogs, and an association between Pet and DailyLogs that denotes which pet the daily logs are related to. **[FALSE: based on the user story, a Pet is associated with daily log entries, so a class DailyLogEntry (not a plural, but an association of one pet to many daily log entries)]**
2. In a class diagram, there should be a class Pet, a class DailyLogEntry, and an aggregation or composition from Pet to DailyLogEntry with multiplicity '1..*' that denotes the daily log entries that are part of a pet. **[FALSE: should be 0..* since it does not have to have a daily log entry]**
3. In a class diagram, there should be a class Veterinarian and a method in this class labelled with a method name (similar to) "addMedicalExaminationRecord" to add a medical examination record to the system. **[FALSE: the method should be in the Pet class, that's where the medical examination record is added]**

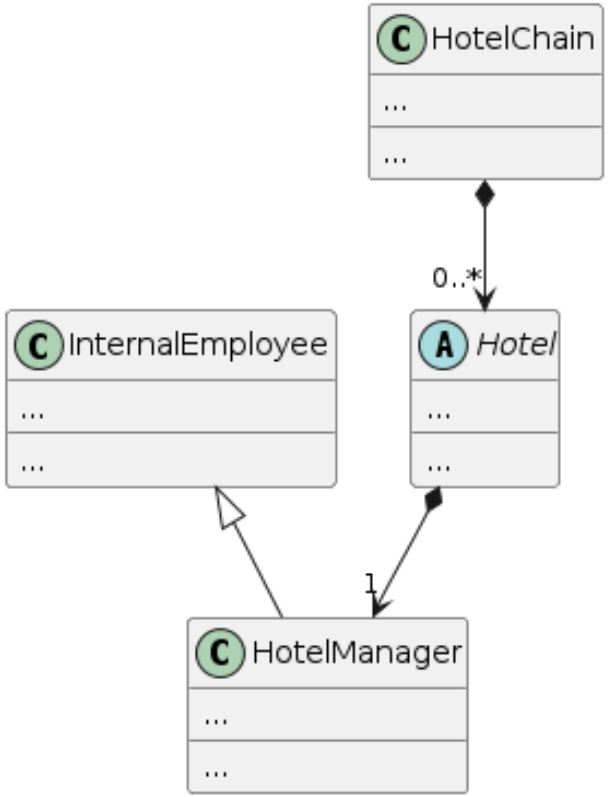
Section – Decomposing user stories

1. In a class diagram, there should be a class Pet, a class DailyLogs, and an association between Pet and DailyLogs that denotes which pet the daily logs are related to. **[FALSE: based on the user story, a Pet is associated with daily log entries, so a class DailyLogEntry (not a plural, but an association of one pet to many daily log entries)]**
2. In a class diagram, there should be a class Pet, a class DailyLogEntry, and an aggregation or composition from Pet to DailyLogEntry with multiplicity '1..*' that denotes the daily log entries that are part of a pet. **[FALSE: should be 0..* since it does not have to have a daily log entry]**
3. In a class diagram, there should be a class Veterinarian and a method in this class labelled with a method name (similar to) "addMedicalExaminationRecord" to add a medical examination record to the system. **[FALSE: the method should be in the Pet class, that's where the medical examination record is added]**



Section – Design, Diagrams and Modularity

You are working as a software developer for a hotel chain company called “ParadiseHotels”. You are currently working on a project where the goal is to develop an application that captures the hotels, the employees and the expenses in a dashboard for all hotels of “ParadiseHotels”. A colleague of yours has already sketched a first draft of the design of the application in the form of a class diagram, which is by no means complete, and has started to implement the application in Java based on the class diagram. Note that the class diagram is an early draft so not everything is captured yet and there might be minor mistakes that should however not affect questions below.



1. Based on the class diagram, the HotelManager class is associated with at least two other classes. **[False: inheritance is not an association]**

…

…you are asked to go through each of the user stories/requirements and decide whether it is consistent with the current application design represented in the class diagram and correctly supported (true) or not (false).

1. As a hotel manager, I want to be able to add a new beach hotel to the hotel chain I am working for to be able to expand the hotel chain. **[False: in the diagram the manager does nothing about hotel chain]**