Systematic debugging

Tools and Techniques for Software Testing - Barbara Russo

SwSE - Software and Systems Engineering group

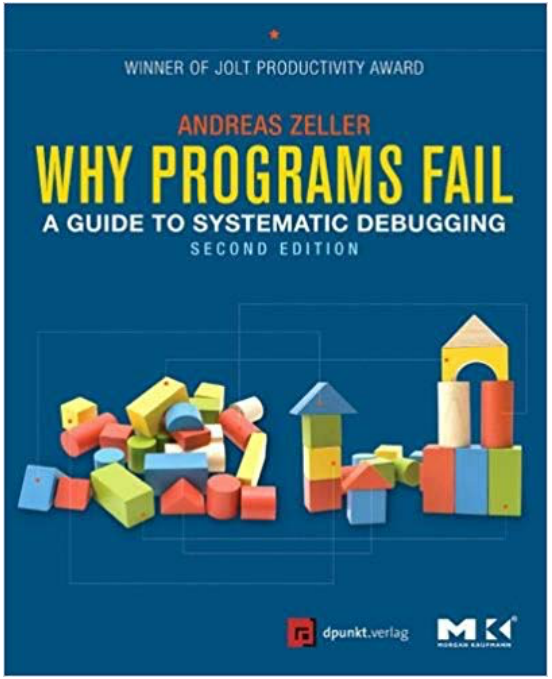
unibz
Freie Universität Bozen
Libera Università di Bolzano
Università Liedia de Bulsan

Overview

- Reproduce the bug as a test case, and put it in your regression suite
- Find the bug using the scientific method
- Fix the bug thoughtfully, no work around
- Use regression testing to keep it from coming back

A reference book

Andreas Zeller, Why Programs Fail



The pain of traditional debugging

- A user passes the whole text of Shakespeare's plays into a method

```
mostCommonWord(allShakespearesPlaysConcatenated)
```

- Discovers that instead of returning a predictably common English word like "the" or "a", the method returns something unexpected, perhaps "e"

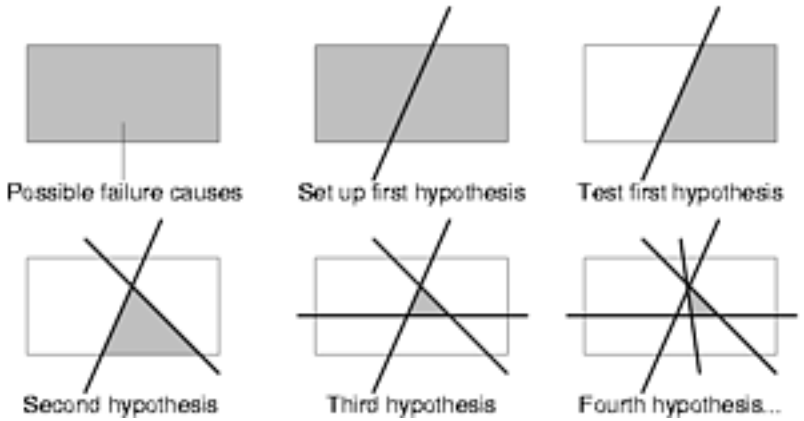
The pain of traditional debugging

- Shakespeare's plays have 100,000 lines containing over 800,000 words
- This input would be very painful to debug by normal methods, like *print-debugging and breakpoint-debugging*

Reducing the size of input

- Debugging will be easier if

*First work on **reducing the size of the buggy input** to something manageable that still exhibits the same (or very similar) bug*



Reducing the size of input

- Examples:

- Does the first half of Shakespeare's plays show the same bug? (Binary search)
- Does a single play have the same bug?
- Does a single speech have the same bug?

Watch out!

- Reducing the size of an input is not partition testing.
- Can I use partition testing for the Shakespeare's plays?

Find and fix with smaller test

- Find and fix the bug using a smaller test case
- Then go back to the original buggy input and confirm that you fixed the same bug by running the test case on it.

- 10 min Exercise: how would you reduce the size of input in your lab project?

Reproduce the failure

- Find a small, repeatable test case that produces the failure
  - If the bug was found by regression testing, then a failing test case is already in the test suite
  - If the bug was reported by a user, it would be harder to reproduce the bug
  - For graphical user interfaces and multithreaded programs, a bug may be even harder to reproduce consistently when it depends on timing of events or thread execution

Modify your regression test suite

- After a bug is fixed add the test case to your regression test suite

Documenting a Testing Strategy

- **For the function on the left, on the right is how to document a partition testing strategy**

```
/**
 * Reverses the end of a string.
 *
 * For example:
 *   reverseEnd("Hello, world", 5)
 *   returns "Hellodlrow ,"
 *
 * With start == 0, reverses the entire text.
 * With start == text.length(), reverses nothing.
 *
 * @param text    non-null String that will have
 *                its end reversed
 * @param start   the index at which the
 *                remainder of the input is
 *                reversed, requires 0 <=
 *                start <= text.length()
 * @return input text with the substring from
 *               start to the end of the string
 *               reversed
 */
static String reverseEnd(String text, int start)
```

```
/*
 * Testing strategy
 *
 * Partition the inputs as follows:
 * text.length(): 0, 1, > 1
 * start:         0, 1, 1 < start < text.length(),
 *                text.length() - 1, text.length()
 * text.length()-start: 0, 1, even > 1, odd > 1
 *
 * Include even- and odd-length reversals because
 * only odd has a middle element that doesn't move.
 *
 * Exhaustive Cartesian coverage of partitions.
 */
```

Each test method should have a comment above it saying how its test case was chosen, i.e. which parts of the partitions it covers:

```
// covers test.length() = 0,
//        start = 0 = text.length(),
//        text.length()-start = 0
@Test public void testEmpty() {
    assertEquals("", reverseEnd("", 0));
}
```

Exercise

```
/**
 * Find the most common word in a string.
 * @param text string containing zero or more words, where a word
 * is a string of alphanumeric characters bounded by non-alphanumerics.
 * @return the most frequent word of the text, ignoring alphabetic case.
 */


public static String mostCommonWord(String text) {
    ...
}
```

Exercise

- A user reports that

```
mostCommonWord("chicken chicken chicken beef")
```

- returns “**beef**” instead of “**chicken**”

Find a small, repeatable test case that produces the failure

Exercise mentimeter.com

- Reduce the input: **which of the following inputs are worth trying?**
  - *mostCommonWord("chicken chicken beef")*
  - *mostCommonWord("Chicken Chicken Chicken beef")*
  - *mostCommonWord("chicken beef")*
  - *mostCommonWord("a b c")*
  - *mostCommonWord("c c b")*

Explanation of answer

- First and last answers are the best choice
  - **"chicken chicken beef": One fewer chicken is shorter.**
  - "Chicken Chicken Chicken beef": This adds capitalization, so it's not simplifying the current bad input. It's making it more complex and trying to reveal a different bug.
    - Good idea to have capitalization in your test cases for this spec, but focus on one bug at a time.
  - "chicken beef": This is too short, because now chicken and beef are tied as they occur the same number of times
  - "a b c": Too different; introduces a three-way tie
  - **"c c b": Shortens the words themselves**

Regression

- Suppose you reduce the "chicken chicken chicken beef" input down to "c c b", *which also shows the problem*
  - both “c c b” and “chicken chicken chicken beef” return the same answer of the test (fail)
  - Then the problem is fixed for both inputs

Exercise mentimeter.com

- **Which test cases should you now add to your test suite?**
- *assertEquals("chicken", mostCommonWord("chicken chicken chicken beef"))*
- *assertEquals("c", mostCommonWord("c c b"))*
- *assertEquals("c", mostCommonWord("c b"))*
- *the test suite should not be changed, because the spec has not been changed*

Explanation

- Answer 2 is correct as it is the simplest
- Both inputs have been fixed with the same change, it is better to add the smaller, as it is simpler
- You want to add a test case as a regression test against the risk of this bug appearing again

Understand the Location and Cause of the Bug - **the scientific method**

Scientific method

- Study the data
- Hypothesize
- Experiment
- Repeat

The scientific method

- **Study the data.** Data: *test input* that causes the bug and the incorrect results, *failed assertions*, and *stack traces* that result from it.
- **Hypothesize.** Propose a hypothesis, consistent with all the data, *about where the bug might be*, or *where it cannot be*.

The scientific method

- **Experiment.** Devise an experiment that tests your hypothesis. Start with a *simple observation that collects information* but disturbs the system as little as possible

The scientific method

- **Repeat.** Add the data from the experiment and make a fresh hypothesis
  - Have ruled out some possibilities and narrowed the set of possible locations and reasons for the bug

When to apply the scientific method?

- A good rule of thumb is the **10-minute rule**.
  - when you've spent 10 minutes hunting for a bug using ad-hoc, unsystematic inspection

Example

- **Study of the data.** A user reports that mostCommonWord(“chicken chicken chicken beef”) returns "beef" instead of “chicken”.
  - *Data on bug report*
  - The particular words “chicken” and “beef” don’t cause the failure: *what matters is the number of times they occur*
- **Hypothesize** Designing a simpler test case.
  - A test case `mostCommonWord("a a a b")`
- **Experiment** Run the test case by replacing “chicken” and “beef” with simpler words

Repeat

- Two related test cases for which one succeeds and one fails.
- For example,
  - mostCommonWords(“c c, b”) fails and
  - mostCommonWords(“c c b”) passes

Another example: using stack traces

Study the data

- Stack traces from exceptions give you useful information about where and what the bug might be

Exercise

- What line actually threw the exception?
- When the exception was thrown, what was the last line in code being executed
- What method of the code was called first in the stack trace?

```
java.lang.NullPointerException
  at java.util.Objects.requireNonNull(Objects.java:203)
  at java.util.AbstractSet.removeAll(AbstractSet.java:169)
  at turtle.TurtleSoup.drawPersonalArt(TurtleSoup.java:29)
  at turtle.TurtleSoupTest.testPersonalArt(TurtleSoupTest.java:39)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
  at java.lang.reflect.Method.invoke(Method.java:498)
  at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
  at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
  at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:47)
  at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
  at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57)
  at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
  at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
  at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
  at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
  at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
  at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
  at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUnit4TestReference.java:86)
  at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:459)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:678)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:382)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:192)
```

Example

```
java.lang.NullPointerException
  at java.util.Objects.requireNonNull(Objects.java:203)
  at java.util.AbstractSet.removeAll(AbstractSet.java:169)
  at turtle.TurtleSoup.drawPersonalArt(TurtleSoup.java:29)
  at turtle.TurtleSoupTest.testPersonalArt(TurtleSoupTest.java:39)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
  at java.lang.reflect.Method.invoke(Method.java:498)
  at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
  at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
  at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:47)
  at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
  at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57)
  at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
  at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
  at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
  at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
  at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
  at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
  at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUnit4TestReference.java:86)
  at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:459)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:678)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:382)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:192)
```

What line actually threw the exception

```
java.lang.NullPointerException
  at java.util.Objects.requireNonNull(Objects.java:203)
  at java.util.AbstractSet.removeAll(AbstractSet.java:169)
  at turtle.TurtleSoup.drawPersonalArt(TurtleSoup.java:29)
  at turtle.TurtleSoupTest.testPersonalArt(TurtleSoupTest.java:39)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
  at java.lang.reflect.Method.invoke(Method.java:498)
  at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
  at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
  at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:47)
  at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
  at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57)
  at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
  at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
  at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
  at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
  at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
  at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
  at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUnit4TestReference.java:86)
  at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:459)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:678)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:382)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:192)
```

When the exception was thrown, what was the last line in code being executed?

```
java.lang.NullPointerException
  at java.util.Objects.requireNonNull(Objects.java:203)
  at java.util.AbstractSet.removeAll(AbstractSet.java:169)
  at turtle.TurtleSoup.drawPersonalArt(TurtleSoup.java:29)
  at turtle.TurtleSoupTest.testPersonalArt(TurtleSoupTest.java:39)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
  at java.lang.reflect.Method.invoke(Method.java:498)
  at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
  at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
  at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:47)
  at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
  at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57)
  at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
  at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
  at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
  at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
  at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
  at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
  at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUnit4TestReference.java:86)
  at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:459)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:678)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:382)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:192)
```

What method was called first in the stack trace?

```
java.lang.NullPointerException
  at java.util.Objects.requireNonNull(Objects.java:203)
  at java.util.AbstractSet.removeAll(AbstractSet.java:169)
  at turtle.TurtleSoup.drawPersonalArt(TurtleSoup.java:29)
  at turtle.TurtleSoupTest.testPersonalArt(TurtleSoupTest.java:39)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
  at java.lang.reflect.Method.invoke(Method.java:498)
  at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
  at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
  at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:47)
  at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
  at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57)
  at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
  at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
  at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
  at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
  at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
  at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
  at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUnit4TestReference.java:86)
  at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:459)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:678)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:382)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:192)
```

Hypothesize

bug localization

- The point where the program actually failed, by throwing an exception or failing, **is not necessarily where the bug is**
- The buggy code may have **propagated some bad values through good parts** of the program before it eventually failed

bug propagation

Example

```
java.lang.NullPointerException
  at java.util.Objects.requireNonNull(Objects.java:203)
  at java.util.AbstractSet.removeAll(AbstractSet.java:169)
  at turtle.TurtleSoup.drawPersonalArt(TurtleSoup.java:29)
  at turtle.TurtleSoupTest.testPersonalArt(TurtleSoupTest.java:39)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
  at java.lang.reflect.Method.invoke(Method.java:498)
  at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
  at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
  at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:47)
  at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
  at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57)
  at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
  at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
  at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
  at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
  at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
  at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
  at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUnit4TestReference.java:86)
  at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:459)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:678)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:382)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:192)
```

Localize the bug

```
java.lang.NullPointerException
  at java.util.Objects.requireNonNull(Objects.java:203)
  at java.util.AbstractSet.removeAll(AbstractSet.java:169)
  at turtle.TurtleSoup.drawPersonalArt(TurtleSoup.java:29)
  at turtle.TurtleSoupTest.testPersonalArt(TurtleSoupTest.java:39)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
  at java.lang.reflect.Method.invoke(Method.java:498)
  at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
  at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
  at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:47)
  at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
  at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57)
  at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
  at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
  at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
  at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
  at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
  at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
  at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUnit4TestReference.java:86)
  at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:459)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:678)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:382)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:192)
```

How the bug propagate

```
java.lang.NullPointerException
  at java.util.Objects.requireNonNull(Objects.java:203)
  at java.util.AbstractSet.removeAll(AbstractSet.java:169)
  at turtle.TurtleSoup.drawPersonalArt(TurtleSoup.java:29)
  at turtle.TurtleSoupTest.testPersonalArt(TurtleSoupTest.java:39)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
  at java.lang.reflect.Method.invoke(Method.java:498)
  at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
  at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
  at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:47)
  at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
  at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57)
  at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
  at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
  at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
  at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
  at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
  at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
  at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUnit4TestReference.java:86)
  at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:459)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:678)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:382)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:192)
```

The bug can be caused elsewhere

```
java.lang.NullPointerException
  at java.util.Objects.requireNonNull(Objects.java:203)
  at java.util.AbstractSet.removeAll(AbstractSet.java:169)
  at turtle.TurtleSoup.drawPersonalArt(TurtleSoup.java:29)
  at turtle.TurtleSoupTest.testPersonalArt(TurtleSoupTest.java:39)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
  at java.lang.reflect.Method.invoke(Method.java:498)
  at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
  at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
  at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:47)
  at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
  at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57)
  at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
  at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
  at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
  at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
  at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
  at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
  at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUnit4TestReference.java:86)
  at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:459)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:678)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:382)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:192)
```

Hypothesize

- So the hypotheses should be about **where** the bug actually is (or is not), and **what** might have **caused** it

Example

```
java.lang.NullPointerException
  at java.util.Objects.requireNonNull(Objects.java:203)
  at java.util.AbstractSet.removeAll(AbstractSet.java:169)
  at turtle.TurtleSoup.drawPersonalArt(TurtleSoup.java:29)
  at turtle.TurtleSoupTest.testPersonalArt(TurtleSoupTest.java:39)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
  at java.lang.reflect.Method.invoke(Method.java:498)
  at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
  at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
  at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:47)
  at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
  at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57)
  at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
  at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
  at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
  at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
  at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
  at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
  at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUnit4TestReference.java:86)
  at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:459)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:678)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:382)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:192)
```

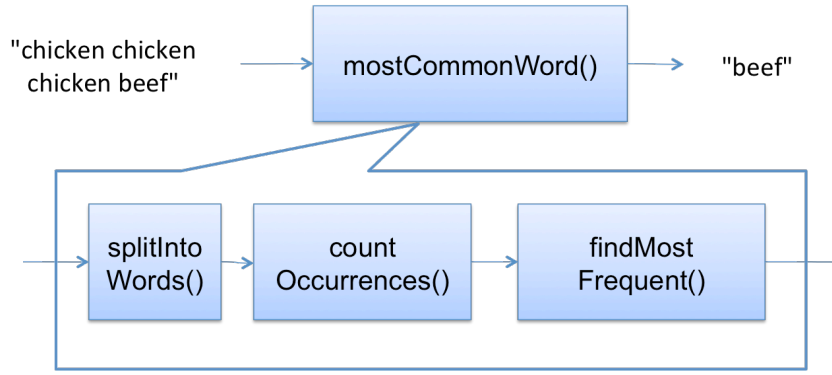
Rule out parts of the code: external libraries

```
java.lang.NullPointerException
  at java.util.Objects.requireNonNull(Objects.java:203)
  at java.util.AbstractSet.removeAll(AbstractSet.java:169)
  at turtle.TurtleSoup.drawPersonalArt(TurtleSoup.java:29)
  at turtle.TurtleSoupTest.testPersonalArt(TurtleSoupTest.java:39)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
  at java.lang.reflect.Method.invoke(Method.java:498)
  at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
  at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
  at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:47)
  at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
  at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
  at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57)
  at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
  at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
  at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
  at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
  at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
  at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
  at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUnit4TestReference.java:86)
  at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:459)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:678)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:382)
  at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:192)
```

Rule out part of the code: use execution flow

- Look at the flow of data in `mostCommonWord()`
- If the **symptom** of the bug *is an exception in `countOccurrences()`*, then you can rule out everything downstream:
  - for example `findMostFrequent()`

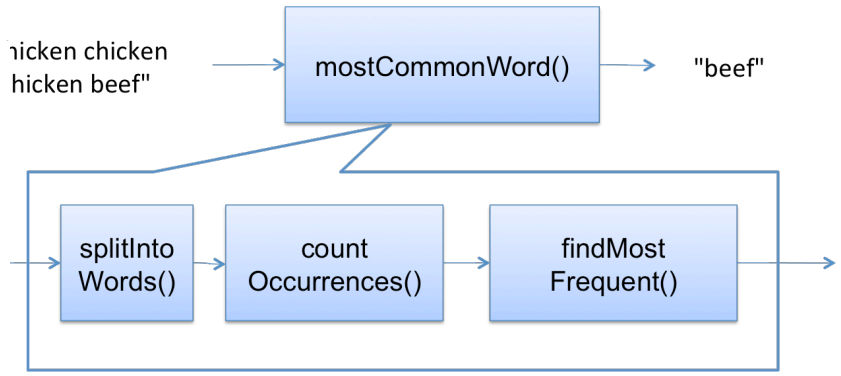
```
public static String mostCommonWord(String text) {
    List<String> words = splitIntoWords(text);
    Map<String,Integer> frequencies = countOccurrences(words);
    String winner = findMostFrequent(frequencies);
    return winner;
}
```



Rule out part of the code: use execution flow

- Choose a hypothesis that tries to localize the bug even further
- For instance, *hypothesize that the bug is in* `splitIntoWords()`, corrupting its results, which then cause the exception in `countOccurrences()`

```
public static String mostCommonWord(String text) {
    List<String> words = splitIntoWords(text);
    Map<String,Integer> frequencies = countOccurrences(words);
    String winner = findMostFrequent(frequencies);
    return winner;
}
```



Experiment

- Third, run test cases:
  - If the hypothesis is true, then `countOccurrences()` is not the source of the problem
  - If it is false, then `splitIntoWords()` is not the source of the problem

Hypothesize: Delta debugging

Rationale

- Two closely-related test cases for which one succeeds and one fails
- For example, maybe
- `mostCommonWords("c c, b")` fails, but `mostCommonWords("c c b")` passes

Rationale

- Examine the difference between the execution of these two test cases to help form a hypothesis
  - *Which code is executed for the passing test case, but skipped for the failing test case, and vice versa?*

Hypothesis: the delta

- One hypothesis is that the **bug lies in those lines of code, the delta, between the passing run and the failing run**

Andreas Zeller, Why Programs Fail

Delta debugging

- Delta debugging defines the cause of a bug as a difference between successful execution and failing execution, and then searches for a minimal cause in order to find the problem
- Delta debugging tools can automate this process, but they are not widely used yet. Only one old implementation in Python (2005). Does it work on new applications? No plug-in for Eclipse!

Perfect thesis work!!

Exercise

- A user reports that when there is only one auction s/he is not able to visualize the next price of the auction.

- Use delta debugging to discover to which part of the code is used in the two test cases
- Used code in Teams:
- Auction.java and AuctionUnitTest.java

Hypothesize: Slicing

unibz
Freie Universität Bozen
Libera Università di Bolzano
Università Liedia de Bulsan

Slicing

- Finding the **parts of a program** that contributed to computing a particular value
- When a failure occurs, the slice for that value consists of the **lines of the program that helped compute that bad value**
- The bug that caused the bad value lies **somewhere in that slice**, so that's your search space

Example

```
int x = 0; // must be >= 0
...
System.out.println("x=" + x);  // prints a negative number
```

Lines that directly change the value of X

```
int x = 0; // must be >= 0
```

Lines that directly change the value of x...

```
    x += bonus;
...
System.out.println("x=" + x);  // prints a negative number
```

Lines that helped compute “bonus” at that point

```
int x = 0; // must be >= 0
final int bonus = getBonus();
...
    x += bonus;
...
System.out.println("x=" + x);  // prints a negative number
```

Control statements that affected the execution of statements already in the slice

```
int x = 0;
final int bonus = getBonus();
...
for (final Sale s : salesList) {
  ...
  if (isWorthABonus(s)) {
    x += bonus;
  }
  ...
}
...
System.out.println("x=" + x);  // prints a negative number
```

Control statements that affected the execution of statements already in the slice

```
int calculateTotalBonus(final List<Sale> salesList) {
  ...
  int x = 0;
  final int bonus = getBonus();
  ...
  for (final Sale s : salesList) {
    ...
    if (isWorthABonus(s)) {
      x += bonus;
    }
    ...
  }
  ...
  System.out.println("x=" + x);  // prints a negative number
  ...
}
```

Exercise

- Find the slice for the buggy lines found for the above exercise on the Auction

Experiment

Experiment

- A good experiment is an observation of the system without disturbing it much. For example:
  - **Run a different test case** with reduced input
  - **Insert a print statement or assertion** in the running program, to check its internal state
  - **Set a breakpoint using a debugger**, then inspect the code with single-step and look at variables and object values

Watch out!

- *Do not work around your code before understanding your code* :
  - Risk to mask the real bug
  - Increase the technical debt
- For example, when getting an `ArrayOutOfBoundsException`, don't just add code that avoids or catches the exception, without fixing the real problem!

Other common experiments

- **Swap components.** Swapping an alternative component implementing the interface.
  - If you suspect java.util.ArrayList, you could swap in java.util.LinkedList instead.
  - If you suspect the Java runtime, run with a different version of Java.

- Watch out!: have good reason to suspect a component as it takes time!

Other common experiments

- **Make sure source code and object code are up to date:**
  - Not running the right code: pull the latest version from the repository, and delete all your binary files and recompile everything (in Eclipse, this is done by Project → Clean)

Exercise mentimeter.com

The quadraticRoots function produces wrong answers sometimes

```
/**
 * Solves quadratic equation ax^2 + bx + c = 0.
 *
 * @param a quadratic coefficient, requires a != 0
 * @param b linear coefficient
 * @param c constant term
 * @return a list of the real roots of the equation
 */
public static List<Double> quadraticRoots(int a, int b, int c) { ... }
```

Put the following items in the order that you should try them: 1, 2, 3

- Change the code from using ArrayList to using LinkedList
- Insert println() statements throughout the method to display the intermediate values of the calculation
- Write a test case that causes the bug to happen

Explain

- **3** Change the code from using ArrayList to using LinkedList
- **2** Put println() statements throughout the method to display the intermediate values of the calculation
- **1** Write a test case that causes the bug to happen

Explanation

- Having a failing test case makes it much easier to debug, and is always the best thing to start with
- Inserting print statements to collect information takes more effort but the effort pays off
- Changing from ArrayList to LinkedList is a good example of the “swap components” technique, but these are trusted components that are supposed to behave alike, so they would be the last things to try

Fix the Bug and bug propagation

- **Classify the bug** as a *coding error*, like a misspelled variable or interchanged method parameters, or a ***design error***, like an underspecified or insufficient interface
- Verify whether **the bug has any relatives**. If I just found a divide-by-zero error here, did I do that anywhere else in the code? Also consider what effects your fix will have. Will it break any other code?

Exercise

- How many errors in the modified Auction.java class that I gave you?

Run the test suite

- After you have applied your fix, add the bug's test case to your regression test suite, and run all the tests to assure yourself that
  - The bug is fixed, and
  - No new bugs have been introduced

Coverage Testing

Tools and Techniques for Software Testing - Barbara Russo

SwSE - Software and Systems Engineering group

unibz
Freie Universität Bozen
Libera Università di Bolzano
Università Liedia de Bulsan

Coverage

- One way to judge a test suite is to ask how thoroughly it exercises the program
- This is called coverage

Example text

```
1  int foo (int a, int b, int c, int d, float e) {
2      if (a == 0) {
3          return 0;
4      }
5      int x = 0;
6      if ((a==b) || ((c == d) && bug(a) )) {
7           x=1;
8      }
9      e = 1/x;
10    return e;
11 }
```

bug(): if a=1 it returns true

and false if a!=1.

Coverage

- Coverage is a measure of the completeness of the set of test cases
  - Method coverage
  - Statement coverage
  - Branch coverage
  - Condition coverage

Method coverage

- Measure: percentage of methods that have been executed at least once by test cases
- Tests should call 100% of the methods
- It seems irresponsible to deliver methods in the product when testing never used these methods
  - you need to ensure you have 100% method coverage

Test Case 1 - $T_0(0,0,0,0,0)$

- There is only one method
- int foo (int a, int b, int c, int d, float e)
- for a=0 foo returns 0 no matter the values of the other parameters
- calling foo with input (0,0,0,0,0) we attain 100% method coverage in our example

Statement coverage

- Measure: percentage of statements that have been executed by test cases
  - Achieve 100% statement coverage. Count the number of statements and cover all of them with a test

Example

- With Test Case 1, we executed the program statements on lines 1-4 out of 11 lines of code
- As a result, we had 45% (5/11) statement coverage from Test Case 1

Example - $T_1(1, 1, 1, 1,1)$

- We can attain **100%** statement coverage by one additional test case,
- Test Case 2: foo(1, 1, 1, 1,1), expected return value of 1
- We have now executed the program statements on lines 5-11

Branch Coverage

- Measure: **percentage of the decision points** have been evaluated as both true and false in test cases.
- Two decision points – one on line 2 and the other on line 6
- 2 if (a == 0) {}
- 6 if ((a==b) OR ((c == d) AND bug(a))) {}

True/False

- For decision/branch coverage, we evaluate an entire Boolean expression as one true-or-false predicate
- We need to ensure that each of these predicates (compound or single) is tested as **both true and false**

Line #	Predicate	True	False
3	(a == 0)	T_0(0, 0, 0, 0, 0)	T_1(1, 1, 1, 1, 1)
7	((a==b) OR ((c == d) AND bug(a)))	T_1(1, 1, 1, 1, 1)	

TestCase3

- With Test Case 1 and Test Case 2, we have executed three of the four necessary conditions
  - we have achieved 75% branch coverage so far
- TestCase3 foo(1, 2, 1, 2, 1) return ??
  - In calculating the output, we discover a division by 0 that can cause future failures!
  - That was due to a local variable that we could not control before!

Line #	Predicate	True	False
3	(a == 0)	$T_0(0, 0, 0, 0, 0)$	$T_1(1, 1, 1, 1, 1)$
7	((a==b) OR ((c == d) AND bug(a)))	$T_1(1, 1, 1, 1, 1)$	$T_{\text{division by zero}}(1, 2, 1, 2, 1)$

Condition Coverage

- Measure: **percentage of Boolean sub-expressions** of the program that have been evaluated as both true or false outcome in test cases
  - applies to compounded predicates
- Condition coverage measures the outcome of each of these sub-expressions independently of each other

Predicate	True	False
(a==b)	$T_1(1, 1, 1, 1, 1)$	$T_{\text{division by zero}}(1, 2, 1, 2, 1)$
(c==d)		$T_{\text{division by zero}}(1, 2, 1, 2, 1)$
bug(a)		

Only the 50% coverage!

Test Case 4 - $T_1(1, 2, 1, 1, 1)$

- We examine our available information on the bug method and determine that is should return true when a=1
- foo(1, 2, 1, 1, 1), expected return value 1

Predicate	True	False
(a==b)	$T_1(1, 1, 1, 1, 1)$	$T_{\text{division by zero}}(1, 2, 1, 2, 1)$
(c==d)	$T_1(1, 2, 1, 1, 1)$	$T_{\text{division by zero}}(1, 2, 1, 2, 1)$
bug(a)	$T_1(1, 2, 1, 1, 1)$	

Test Case 5 - $T_{\text{division by zero}}(3, 2, 1, 1, 1)$

- To finalize our condition coverage, we must force bug(a) to be false
- We again examine our bug() method, which informs us that it should return a false value if fed any integer a different from 1
- So we create Test Case 5, foo(3, 2, 1, 1, 1), expected return value “division by zero”.

Note

- We could have (2,2,1,1,1). The input would have been fine but we would never reach the AND condition. Thus, we must make the (a==b) false to be sure to test the AND condition for FALSE.
- The same applies for (c==d): we need to have it TRUE to be sure that FALSE is due to the bug(a) condition.

Traceability matrix

Predicate	True	False
(a==b)	$T_1(1, 1, 1, 1, 1)$	$T_{\text{division by zero}}(1, 2, 1, 2, 1)$
(c==d)	$T_1(1, 2, 1, 1, 1)$	$T_{\text{division by zero}}(1, 2, 1, 2, 1)$
bug(a)	$T_1(1, 2, 1, 1, 1)$	$T_{\text{division by zero}}(3, 2, 1, 1, 1)$

Coverage goals

- 100% statement coverage is a common goal
  - not achievable in general bc. of defensive code (like “should never get here” assertions)
- 100% branch coverage is highly desirable, but
  - 100% Branch coverage requires more test cases than 100% statement coverage
-

Path coverage

- Path coverage is every possible combination of branches — every path through the program — taken by some test case
- McCabe complexity is used to determine how many complete execution paths (i.e. test built on them) a tester need to consider
- As with code coverage this is a measure that approximates exhaustiveness

- Path coverage is stronger than branch coverage, but 100% path coverage may be unfeasible as it in general may require exponential test suite time

Feasible coverage goals

- Feasible statement coverage:
  - Percentage for which every reachable statement is executed by one test case: there are tools to count number of times each statement is covered by the test suite: jacoco
- With such tools white box testing is easy: you can sees the code covered and create new tests for the missing statements
- Then you just need to design new test cases for the black box testing strategy

JaCoCo

- It is an Eclipse plug-in
- In the node build and sub-node plugins of the POM file include

```
<plugin>
   <groupId>org.jacoco</groupId>
   <artifactId>jacoco-maven-plugin</artifactId>
   <version>0.8.2</version>
   <executions>
      <execution>
         <goals>
            <goal>prepare-agent</goal>
         </goals>
      </execution>
<!-- attached to Maven test phase -->
      <execution>
            <id>report</id>
            <phase>test</phase>
            <goals>
                  <goal>report</goal>
            </goals>
      </execution>
   </executions>
</plugin>
```

Jacoco output

- Line colors:
  - Green – fully covered lines
  - Yellow – party covered lines (some instructions or branches missed)
  - Red – lines that have not been executed at all
- Diamonds:
  - Green for fully covered branches
  - Yellow for partly covered branches
  - Red when no branches in the particular line have been executed

Example

```
17.     public static void main(String[] args) {
18.         System.out.println( "Hello World!" );
19.     }
20.     public static void tt(String args) {
21.         count ++;
22.             if(args.equals("0"))
23.             System.out.printf("Got 0 %d\n", count);
24.         else if(args.equals("1"))
25.             System.out.printf("Got 1 %d\n", count);
26.         else if(args.equals("2"))
27.             System.out.printf("Got 2 %d\n", count);
28.         else if(args.equals("3"))
29.             System.out.printf("Got 3 %d\n", count);
30.     }
```

Example

- To turn green the condition should have been performed twice, one with false and one with true result.
- Until args.equals("0") is always true the line stays yellow

Exercise

```
public class Hailstone {
  public static void main(String[] args) {
    int n = 3;
    while (n != 1) {
        if (n % 2 == 0) {
            n = n / 2;
        } else {
            n = 3 * n + 1;
        }
    }
  }
}
```

- Run this class with JaCoCo code coverage highlighting turned on, by choosing Run → Coverage As → Java Application.
- By changing the initial value of n, you can observe how JaCoCo highlights different lines of code differently.

When `n=3` initially, what color is the line `n = n/2` after execution?

When `n=16` initially, what color is the line `n = 3 * n + 1` after execution?

What initial value of `n` would make the line `while (n != 1)` yellow after execution?