Mutation Testing

Sean Olszewski

bit.ly/muter-swift

@__chefski__

Why should we care about mutation testing?

3

Reasons to Care About Mutation Testing

Identify improvements you can make to an existing test suite

A metric for quality of test suite

Pinpoint the time and cause of a regression of your test suite

Who are you?

Engineering Practice Lead @ Pivotal

Co-organizer of Learn Swift Boston

Author of Muter - Swift Mutation Tester

bit.ly/muter-swift

What is mutation testing?

A mutation test is a test for your tests

It helps you make sure your tests fail when you would expect them to fail

And shows you the instances when they won’t fail

It does this by introducing small changes into your source code

The changed programs are called mutants

Mutants mimic realistic program errors*

* actual scientific statement

A mutant is introduced by a mutation operator

There are many kinds of mutation operators

One kind will change equality operators

```
if myValue == 50 {
    // something
}
```

```
if myValue != 50 {
    // something
}
```

Another kind will
remove side effects

```swift
func update(email: String, for userId: String) {
    var userRecord = getRecord(for: userId)
    userRecord.email = email
    database.persist(userRecord)
}
```

```swift
func update(email: String, for userId: String) {
    var userRecord = getRecord(for: userId)
    userRecord.email = email
}
```

Cool. But what about that science you mentioned?

Competent Programmer Hypothesis

Coupling Effect

Neat. So that’s all?

After applying a mutation operator, your tests are run

The result of running your tests gets recorded

A test suite which failed in response to a mutant killed the mutant

You want your test suite to kill the mutant

THE ONLY GOOD BUG IS A DEAD BUG.

Your app code is then restored for the next mutant

Once all the mutants have been introduced, mutation testing is finished

A mutation score then gets generated for your test suite and source files

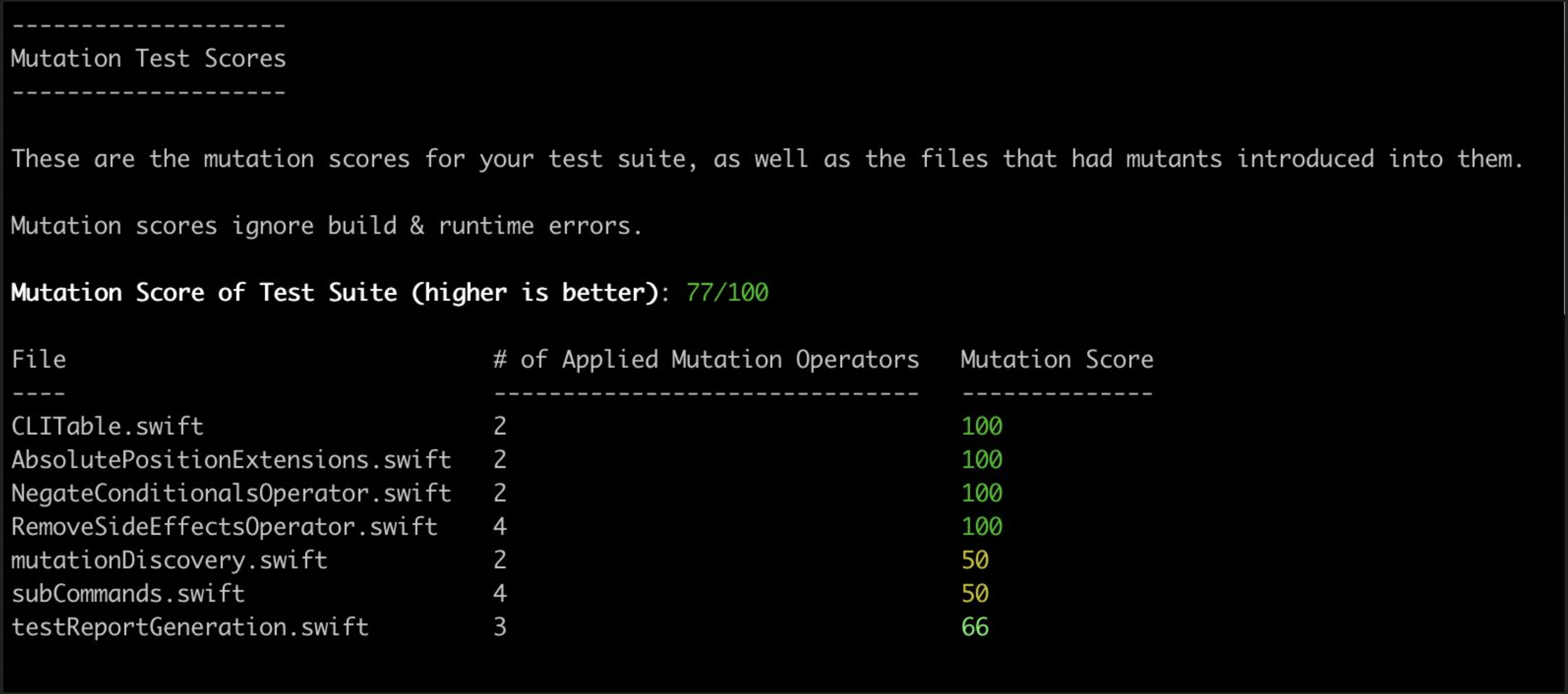
$$\text{mutation score} = \frac{\text{\# of mutants killed}}{\text{total \# of mutants}}$$

The ideal mutation score is 100*

* this is usually not achievable

And this is okay

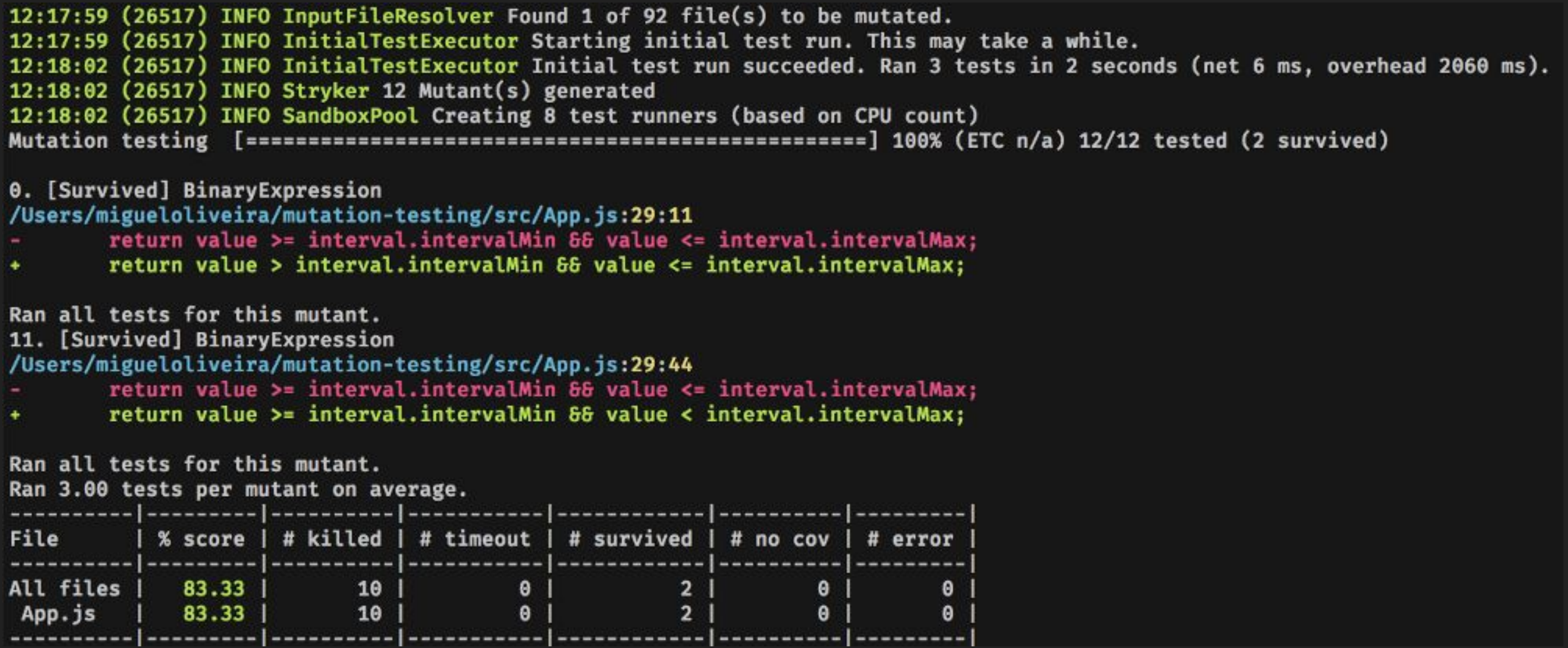
So what’s a report look like?



```
--------------------
Mutation Test Scores
--------------------

These are the mutation scores for your test suite, as well as the files that had mutants introduced into them.

Mutation scores ignore build & runtime errors.

Mutation Score of Test Suite (higher is better): 77/100

File                               # of Applied Mutation Operators   Mutation Score
----                               -------------------------------   --------------
CLITable.swift                     2                                 100
AbsolutePositionExtensions.swift   2                                 100
NegateConditionalsOperator.swift   2                                 100
RemoveSideEffectsOperator.swift    4                                 100
mutationDiscovery.swift            2                                 50
subCommands.swift                  4                                 50
testReportGeneration.swift         3                                 66
```

bit.ly/muter-swift

@__chefski__



```
12:17:59 (26517) INFO InputFileResolver Found 1 of 92 file(s) to be mutated.
12:17:59 (26517) INFO InitialTestExecutor Starting initial test run. This may take a while.
12:18:02 (26517) INFO InitialTestExecutor Initial test run succeeded. Ran 3 tests in 2 seconds (net 6 ms, overhead 2060 ms).
12:18:02 (26517) INFO Stryker 12 Mutant(s) generated
12:18:02 (26517) INFO SandboxPool Creating 8 test runners (based on CPU count)
Mutation testing  [==================================================] 100% (ETC n/a) 12/12 tested (2 survived)

0. [Survived] BinaryExpression
/Users/migueloliveira/mutation-testing/src/App.js:29:11
-       return value >= interval.intervalMin && value <= interval.intervalMax;
+       return value > interval.intervalMin && value <= interval.intervalMax;

Ran all tests for this mutant.
11. [Survived] BinaryExpression
/Users/migueloliveira/mutation-testing/src/App.js:29:44
-       return value >= interval.intervalMin && value <= interval.intervalMax;
+       return value >= interval.intervalMin && value < interval.intervalMax;

Ran all tests for this mutant.
Ran 3.00 tests per mutant on average.
----------|---------|----------|-----------|------------|----------|---------|
File      | % score | # killed | # timeout | # survived | # no cov | # error |
----------|---------|----------|-----------|------------|----------|---------|
All files |   83.33 |       10 |         0 |          2 |        0 |       0 |
 App.js   |   83.33 |       10 |         0 |          2 |        0 |       0 |
----------|---------|----------|-----------|------------|----------|---------|
```

bit.ly/muter-swift

@__chefski__

How is a mutation score different from code coverage?

Code Coverage

a measure of how much application code is executed by a test suite

indicates what code is exposed to a test

Mutation Score

a measure of how sensitive your test suite is to changes in your source code

indicates how a test interacts with code

How do we use this stuff anyway?

A low mutation score indicates the need to write different assertions, or add test cases

How you can improve your test will depend on the mutation operator

For example

Muter has a mutation operator that prevents (some) side effects from occurring

You may add a test which uses a test double to observe the side effect

Enabling you to kill that mutant in the future

How should we begin mutation testing?

Do‘s & Don’ts of Mutation Testing

Do

- Incrementally mutation test your project
- Set up a scheduled CI job
- Incorporate a review of metrics

Don’t

- Attempt to address all surviving mutants at once
- Institute mutation score requirements
- Ignore other test suite metrics
- Mutation test when your code coverage is low

What tools exist to help us mutation test?

Mutation Testing Tools

Name	Language(s)	Link
Muter	Swift	bit.ly/muter-swift
Stryker	Typescript, .NET, Scala	bit.ly/stryker-js
PITest	JVM Languages	bit.ly/pitest-jvm
Mull	LLVM Code (C/C++)	bit.ly/mull-llvm

Thank you!
Questions?

Sean Olszewski