Mutation Testing in Java

Wojciech Langiewicz @ 33rd Degree 4 Charity

https://github.com/wlk/mutation-testing-demo

How do you make sure your tests are any good?

How do you make sure your tests are any good?

- Code Review
- TDD
- Code coverage
  - What does it really measure?
  - What does it prove?
  - Line/Statement/Branch coverage
  - Tests without assertions

https://github.com/wlk/mutation-testing-demo

Mutation Testing

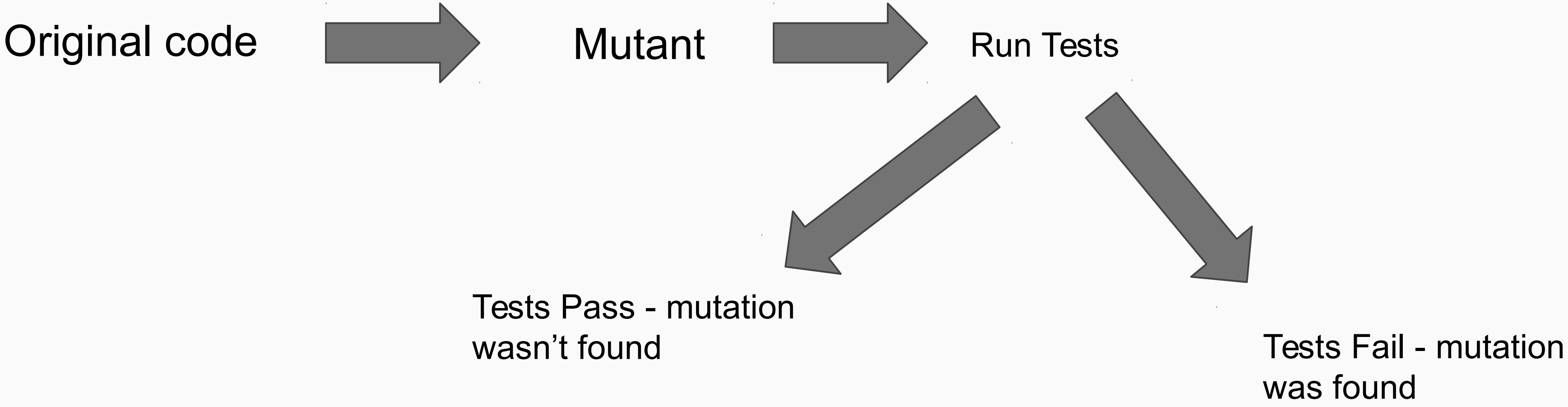
- Technique to measure quality of tests
- Injects a fault into a system and uses our tests to find it
- Proposed in 1971 by Richard Lipton
- Competent programmer hypothesis
- Few new concepts:
    - Mutants
    - Mutation Operators

https://github.com/wlk/mutation-testing-demo

Mutation Testing - problems

- Forgotten for many years (only some academic work)
  - Performance problems
  - Lack of tooling
- Performance problem:
  - Test suite takes 5 minutes to run
  - 500 classes, 10 tests per class, testing each class takes 0.6s
  - Naive: 10 mutants per class gives 10 * 5 * 500 ~ **70 hours**
  - Fast: 10 * 0.6 * 500 = **50 minutes**

How It Works



Original code

Mutant

Run Tests

Tests Pass - mutation wasn’t found

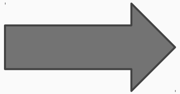
Tests Fail - mutation was found

https://github.com/wlk/mutation-testing-demo

How It Works

Original code

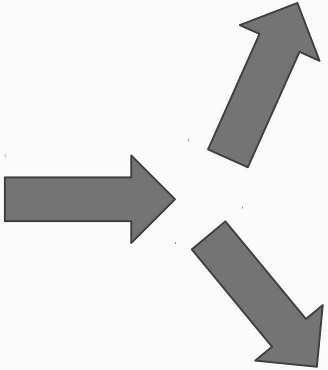
```
if ( i>= 0 ) {
    return "foo";
} else {
    return "bar";
}
```

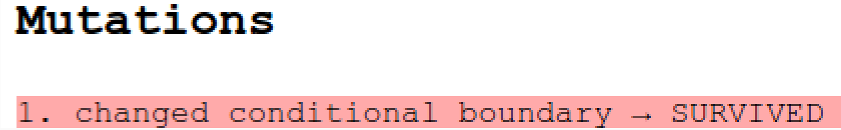


Mutant

```
if ( i> 0 ) {
    return "foo";
} else {
    return "bar";
}
```

Run Tests



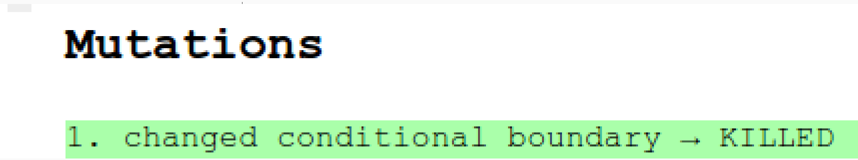


Mutations

1. changed conditional boundary → SURVIVED

Tests Pass - mutation wasn’t found

Tests Fail - mutation was found



Mutations

1. changed conditional boundary → KILLED

https://github.com/wlk/mutation-testing-demo

Mutation Operators (Mutators)

https://github.com/wlk/mutation-testing-demo

Mutator: Conditionals Boundary Mutator

```
<     →   <=
<=        <
>         >=
>=        >
```

```
if (a < b) {
  //do something
}
```

→

```
if (a <= b) {
  //do something
}
```

https://github.com/wlk/mutation-testing-demo

Mutator: Negate Conditionals Mutator

==	!=
!=	++
<=	>
>=	<
<	>=
>	<=

```
if (a == b) {
  //do something
}
```

```
if (a != b) {
  //do something
}
```

https://github.com/wlk/mutation-testing-demo

Mutator: Void Method Call

```
public void someVoidMethod(int i) {
  //does something
}

public int foo() {
  int i = 5;
  doSomething(i);
  return i;
}
```

```
public void someVoidMethod(int i) {
  //does something
}

public int foo() {
  int i = 5;
  //don't do anything
  return i;
}
```

https://github.com/wlk/mutation-testing-demo

Mutator: Constructor Call

```
public Object foo() {
 Object o = new Object();
 return o;
}
```

→

```
public Object foo() {
 Object o = null;
 return o;
}
```

Mutators: Many More

- Replace constants
- Replace return values to defaults
- And many others

Tooling

- PIT - http://pitest.org/
- Ruby: Mutant: https://github.com/mbj/mutant
- Popular in communities where testing (TDD) is already popular

PIT

https://github.com/wlk/mutation-testing-demo

PIT Features

- Bytecode modifications (to avoid recompilation)
- Integrates easily with:
    - Java 6, 7, 8
    - JUnit, TestNG
    - Eclipse, Intellij
    - Gradle, Maven, Ant
    - Sonar, Jenkins
    - Mocking frameworks
- For each mutation, it tries to minimize number of tests to run
- Allows to choose which mutators we want to use
- Doesn't work with Scala
- Doesn't store mutated code
- Generates simple HTML report, or XML report for other tools

Performance Nowadays:

- Not really a problem on CI server when using modern tools (PIT)
- PIT can analyze only changed code (looking at your SCM)
- Practical tests:
  - Apache commons-math:
  - 177k lines of code
  - 109k lines of tests
  - 8 minutes to test
  - PIT takes 1:15h with 4 threads

https://github.com/wlk/mutation-testing-demo

Demo

https://github.com/wlk/mutation-testing-demo

Problems With Mutation Testing

Mutants can be dangerous:

```
if(false){
 Runtime.getRuntime().exec("rm -rf /");
}
```

Defensive programming:

```
if(i> 0){
 throw new IllegalArgumentException(
   “argument must be positive”
 );
}
return Math.sqrt(i);
```

Equivalent Mutants:

```
int i= 2;
if ( i>= 1 ) {
   return "foo";
}
// is equivalent to
int i= 2;
if ( i> 1 ) {
   return "foo";
}
```

https://github.com/wlk/mutation-testing-demo

Summary

- Mutation testing tests your tests
- Code coverage gives you false sense of security
- PIT is extremely easy to introduce into Java project

https://github.com/wlk/mutation-testing-demo

Time for questions

https://github.com/wlk/mutation-testing-demo

Thank You!

https://github.com/wlk/mutation-testing-demo