Mutation Testing to the rescue of your tests

@nicolas_frankel

I T.A.K.E. Unconference | 2016

An event proudly designed by

Mozaic Works
Think. Design. Work smart.

Me, Myself and I

- By day
  - Hybris consultant
- By night
  - Teacher/trainer
  - Book Author: Vaadin, Integration Testing
  - Blogger: https://blog.frankel.ch/
  - Speaker

Many kinds of testing

- Unit Testing
- Integration Testing
- End-to-end Testing
- Performance Testing
- Penetration Testing
- Exploratory Testing
- etc.

Their only single goal

- Ensure the Quality of the production code
- How to check the Quality of the testing code?

Code coverage

- “Code coverage is a measure used to describe the degree to which the source code of a program is tested”
  - *--Wikipedia http://en.wikipedia.org/wiki/Code_coverage*

	# Classes	Line Coverage		Branch Cov
	6	89%	50/56	81%
	15	75%	45/60	N/A
	28	67%	197/295	60%
	7	67%	286/426	40%
	28	61%	156/254	39%
	50	52%	327/634	37%
	5	52%	33/63	46%
	18	51%	133/259	37%
	5	50%	27/54	17%
	27	50%	343/684	32%
	125	49%	1393/2820	43%
	303	48%	3602/7534	39%
	68	46%	623/1363	32%
	84	45%	529/1172	28%
	17	43%	218/511	36%
	79	43%	894/2056	32%
	167	35%	1030/2907	29%
	83	35%	754/2150	25%
	10	34%	159/473	24%
	3	31%	12/39	7%
	37	29%	167/579	19%
	15	19%	54/288	6%
	20	8%	16/200	2%
	6	8%	13/172	4%
	10	6%	13/226	5%
	4	3%	4/123	0%
	11	2%	6/252	2%
	20	0%	0/157	0%
	3	0%	0/39	0%
defs.cvslib	8	0%	0/402	0%
defs	1	0%	0/246	0%
	5	0%	0/59	0%

Measuring Code Coverage

- Check whether a source code line is executed during a test
  - Or Branch Coverage

Computing Code Coverage

- CC: Code Coverage (in percent)
- $L_{executed}$: Number of executed lines of code
- L_{total}: Number of total lines of code

$$CC = \frac{L_{executed}}{L_{total}} * 100$$

Java Tools for Code Coverage

- JaCoCo
- Clover
- Cobertura
- etc.

100% Code Coverage?

“Is 100% code coverage realistic? Of course it is. If you can write a line of code, you can write another that tests it.”

Robert Martin (Uncle Bob)
https://twitter.com/unclebobmartin/status/559666205096673 28

Assert-less testing

```
@Test
public void add_should_add() {
  new Math().add(1, 1);
}
```

But, where is the assert?

As long as the Code Coverage is OK…

Code coverage as a measure of test quality

- Any metric can be gamed!
- Code coverage is a metric…
- ⇒ Code coverage can be gamed
  - On purpose
  - Or by accident

Code coverage as a measure of test quality

- Code Coverage lulls you into a false sense of security…

The problem still stands

- Code coverage cannot ensure test quality
  - Is there another way?

The Cast

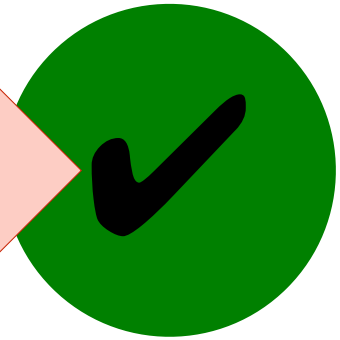
Original Source Code

Modified Source Code

a.k.a "The Mutant"

Standard testing

Execute Test



Mutation testing

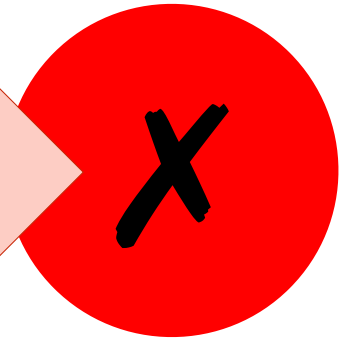
MUTATION

Execute SAME Test

?

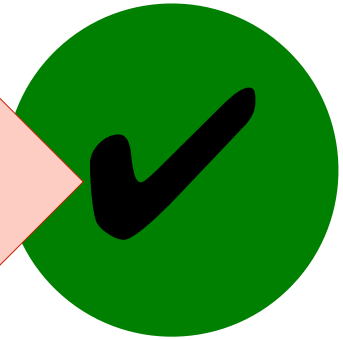
Mutation testing

Execute SAME Test



Mutant Killed

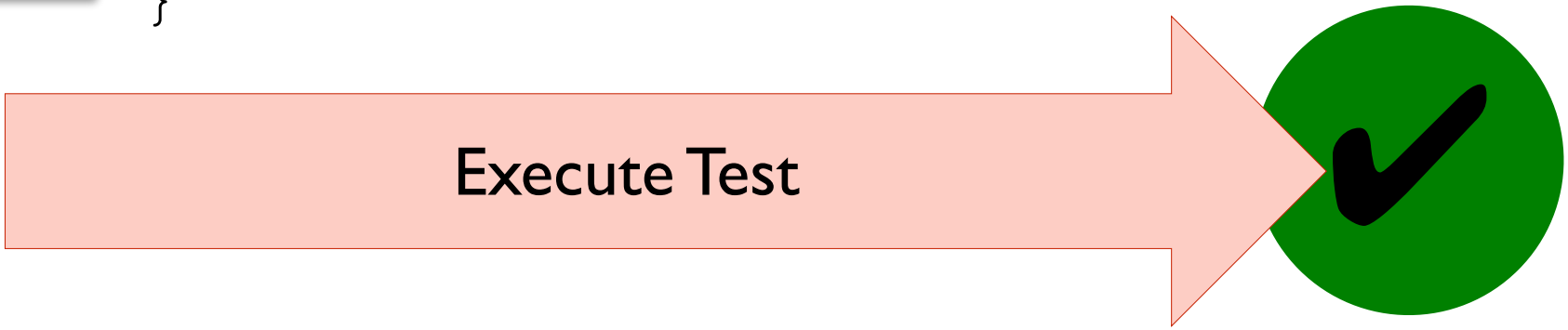
Execute SAME Test



Mutant Survived

Test the code

```
public class Math {
   public int add(int i1, int i2) {
      return i1 + i2;
   }
}
```

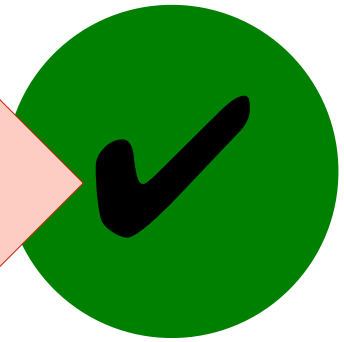



```
@Test
public void add_should_add() {
   new Math().add(1, 1);
}
```

Surviving mutant

```
public class Math {
   public int add(int i1, int i2) {
      return i1 - i2;
   }
}
```

Execute SAME Test

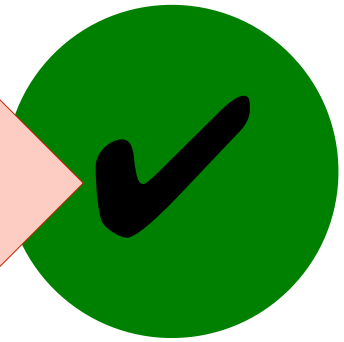


```
@Test
public void add_should_add() {
   new Math().add(1, 1);
}
```

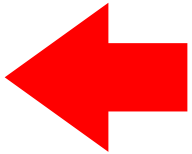
Test the code

```
public class Math {
   public int add(int i1, int i2) {
      return i1 + i2;
   }
}
```

Execute Test



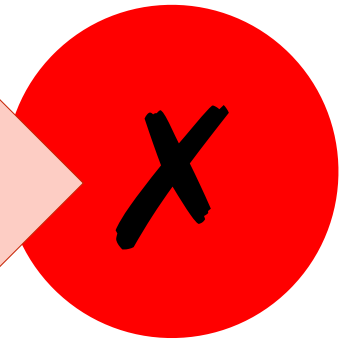
```
@Test
public void add_should_add() {
   new Math().add(1, 1);
   Assert.assertEquals(sum, 2);
}
```



Killed mutant

```
public class Math {
   public int add(int i1, int i2) {
      return i1 - i2;
   }
}
```

Execute SAME Test



```
@Test
public void add_should_add() {
   new Math().add(1, 1);
   Assert.assertEquals(sum, 2);
}
```

Mutation Testing in Java

- PIT is a tool for Mutation testing
- Available as
  - Command-line tool
  - Ant target
  - Maven plugin

Mutators

- Mutators are patterns applied to source code to produce mutations

PIT mutators sample

Name	Example source	Result
Conditionals Boundary	>	>=
Negate Conditionals	==	!=
Remove Conditionals	foo == bar	true
Math	+	-
Increments	foo++	foo--
Invert Negatives	-foo	foo
Inline local variable	int foo= 42	int foo= 43
Return Values	return true	return false
Void Method Call	System.out.println("foo")	
Non Void Method Call	long t = System.currentTimeMillis()	long t = 0
Constructor Call	Date d = new Date()	Date d = null;

Enough talk!

Drawbacks

- Slow
- Sluggish
- Crawling
- Sulky
- Lethargic
- etc.

Metrics (kind of)

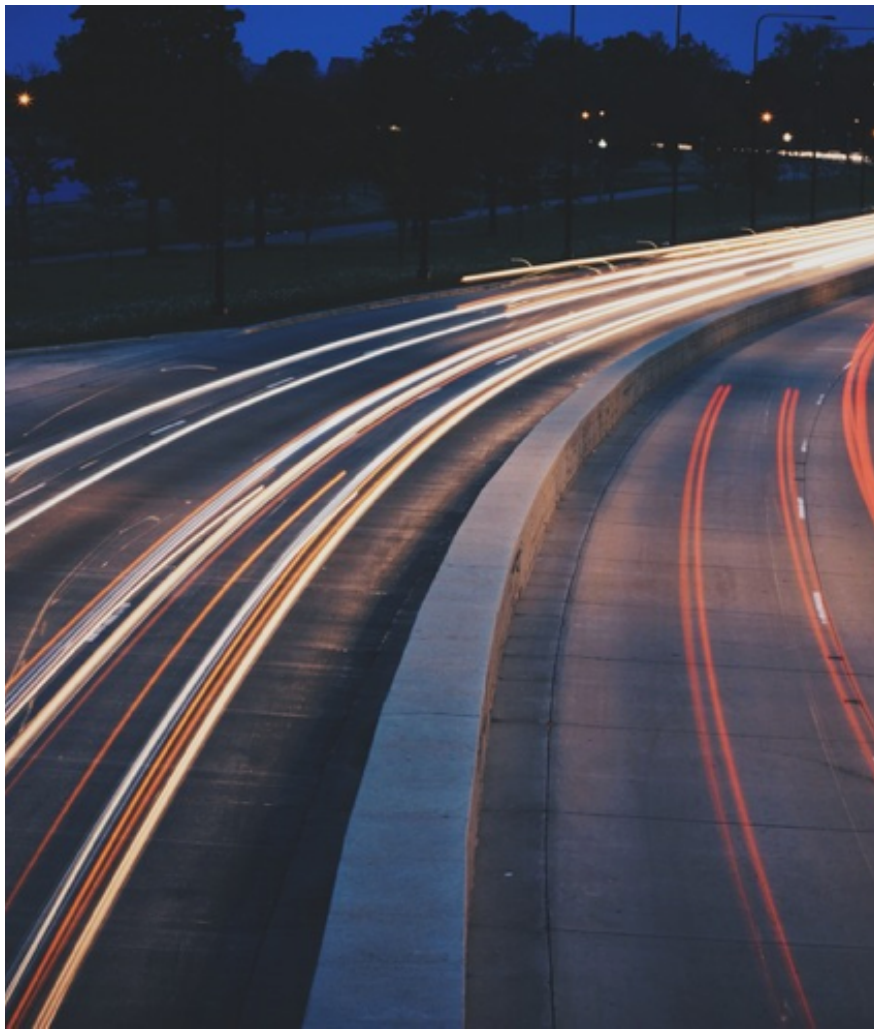
- On *joda-money*
- `mvn clean test-compile`
- `mvn surefire:test`
  - Total time: 2.181 s
- `mvn pit-test...`
  - Total time: 48.634 s

Why so slow?

- Analyze test code
- For each class under test
  - For each mutator
    - Create mutation
  - For each mutation
    - Run test
    - Analyze result
    - Aggregate results

Workarounds

- Increase number of threads → ☺
- Set a limited a set of mutators
- Limit scope of target classes
- Limit number of tests
- Limit dependency distance
- Don’t bind to the test phase → ☺
- Use scmMutationCoverage
- Use incremental analysis → ☺



Incremental analysis

- Metadata stored between runs
- During each following run mutant will not be checked again, if the last time it:
    - timed out, and class has not changed
    - was killed, and neither class nor test have changed
    - survived, and there are no new/changed tests for it

False positives

- Mutation Testing is not 100% bulletproof
- Might return false positives
- Be cautious!

Pit is imperfect

```
if (p < 0)
...
// changed condition boundary
// -> survived:
if (p > 0)
...
return 0;
```

Pit might be dangerous

```
void reboot() throws IOException {
  // removed method call:
  checkUserPermissions();
  Runtime.getRuntime()
                        .exec("reboot");
}
```

Testing is about ROI

- Don’t test to achieve 100% coverage
- Test because it saves money in the long run
- Prioritize:
  - Business-critical code
  - Complex code

Q&A

- https://git.io/vznQK
- http://blog.frankel.ch/
- @nicolas_frankel
- https://leanpub.com/integrationtest/