Mutation Testing

Reid Holmes

Key questions

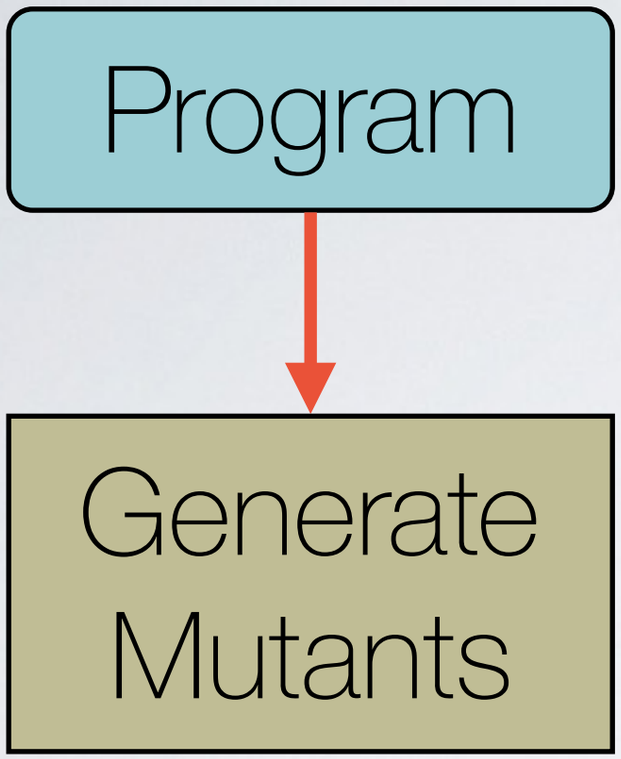
Is a test suite:

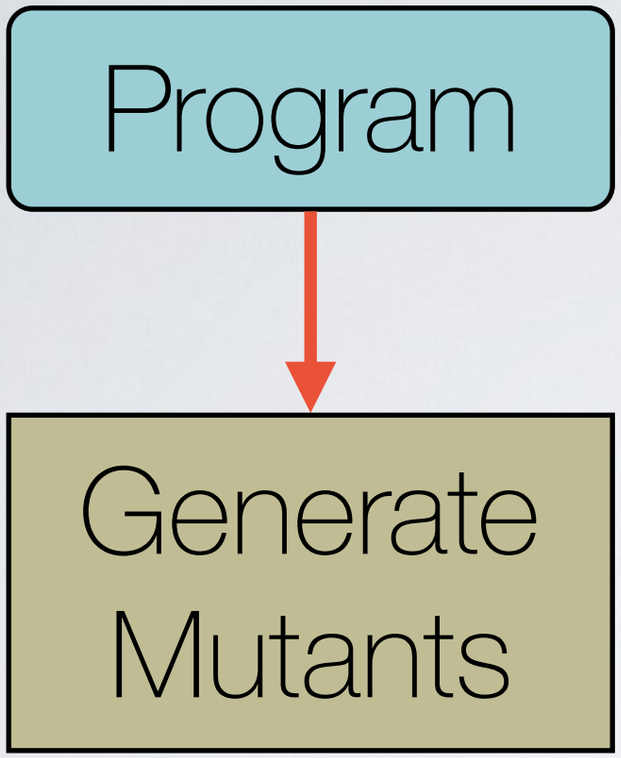
Sufficiently broad?

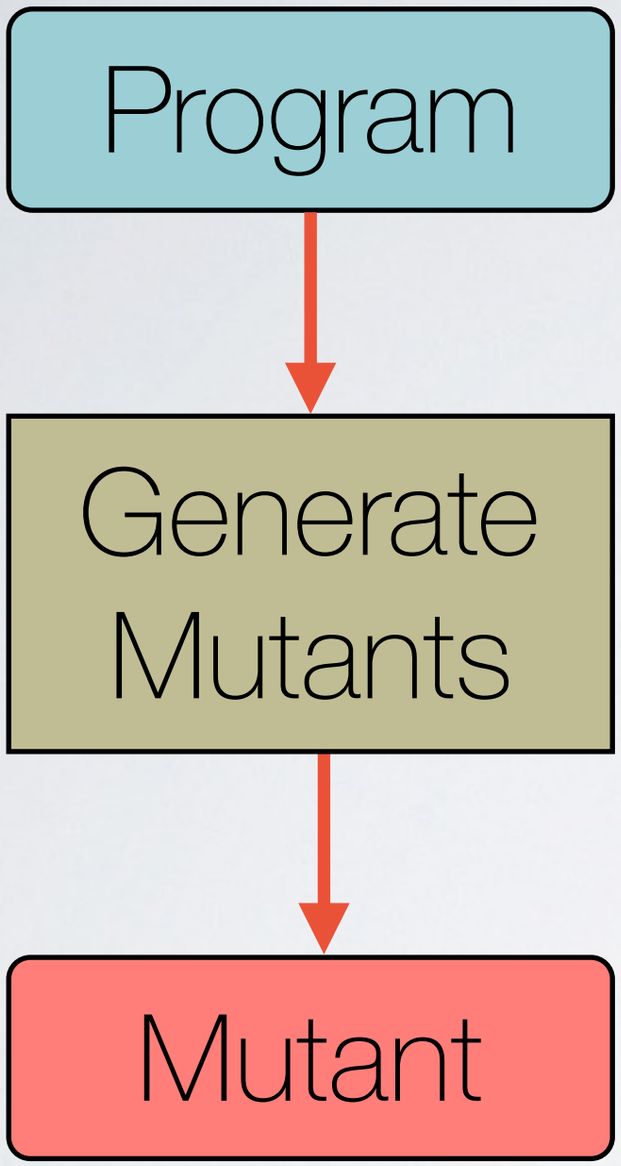
Sufficiently deep?

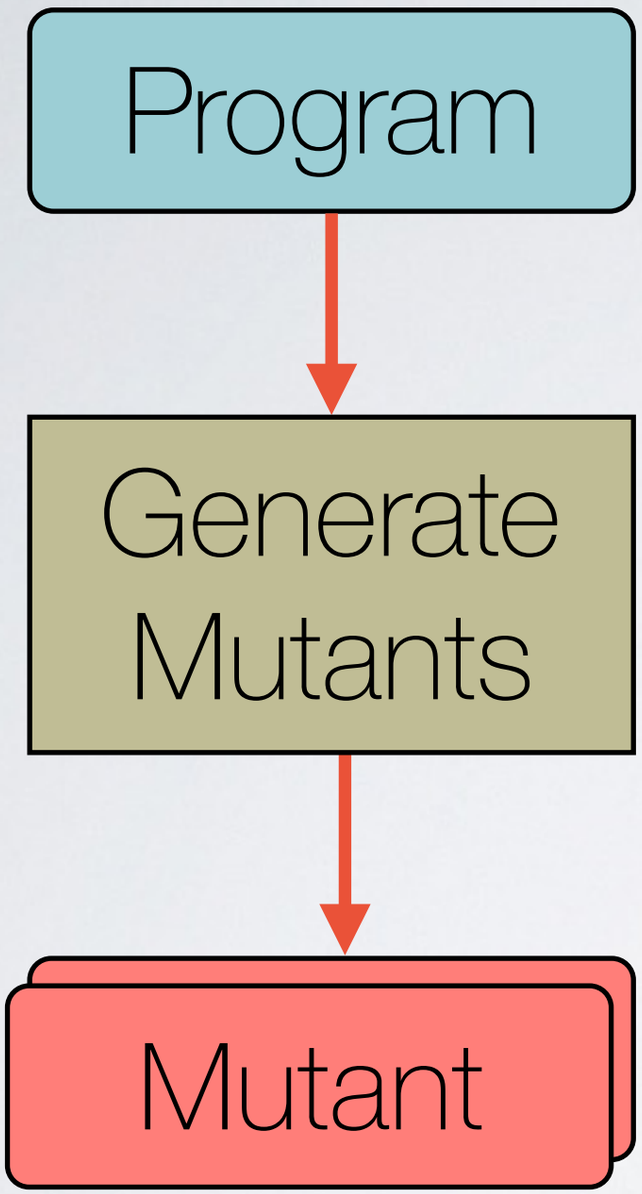
Test suite depth

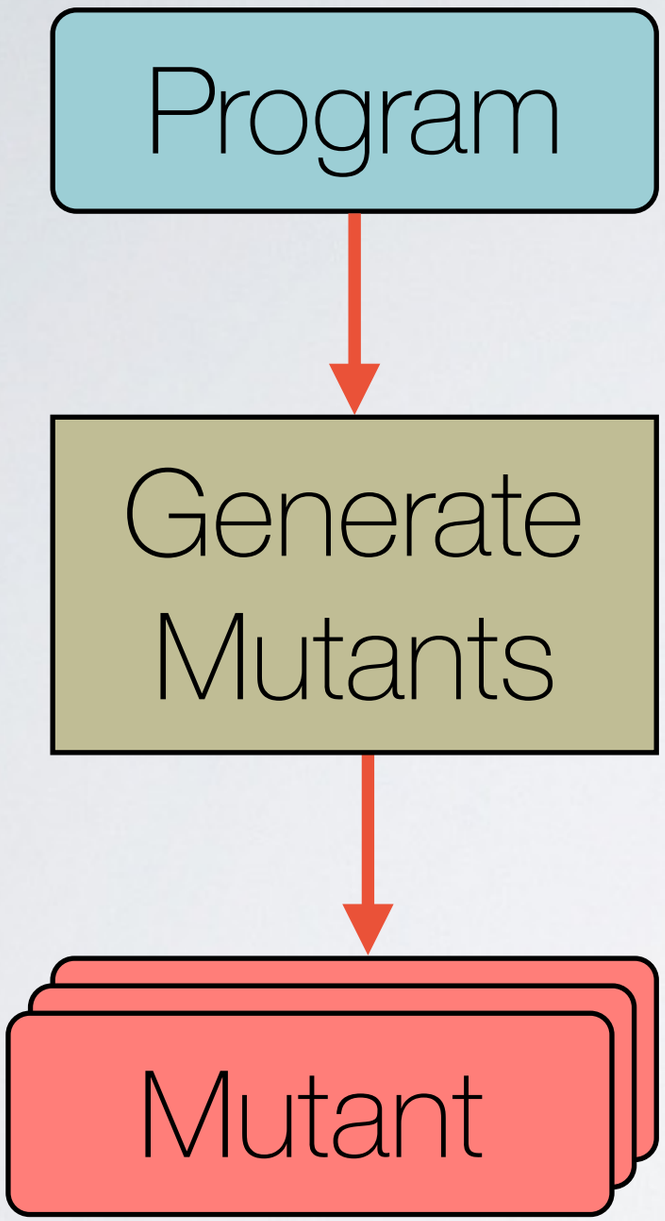
Mutation testing

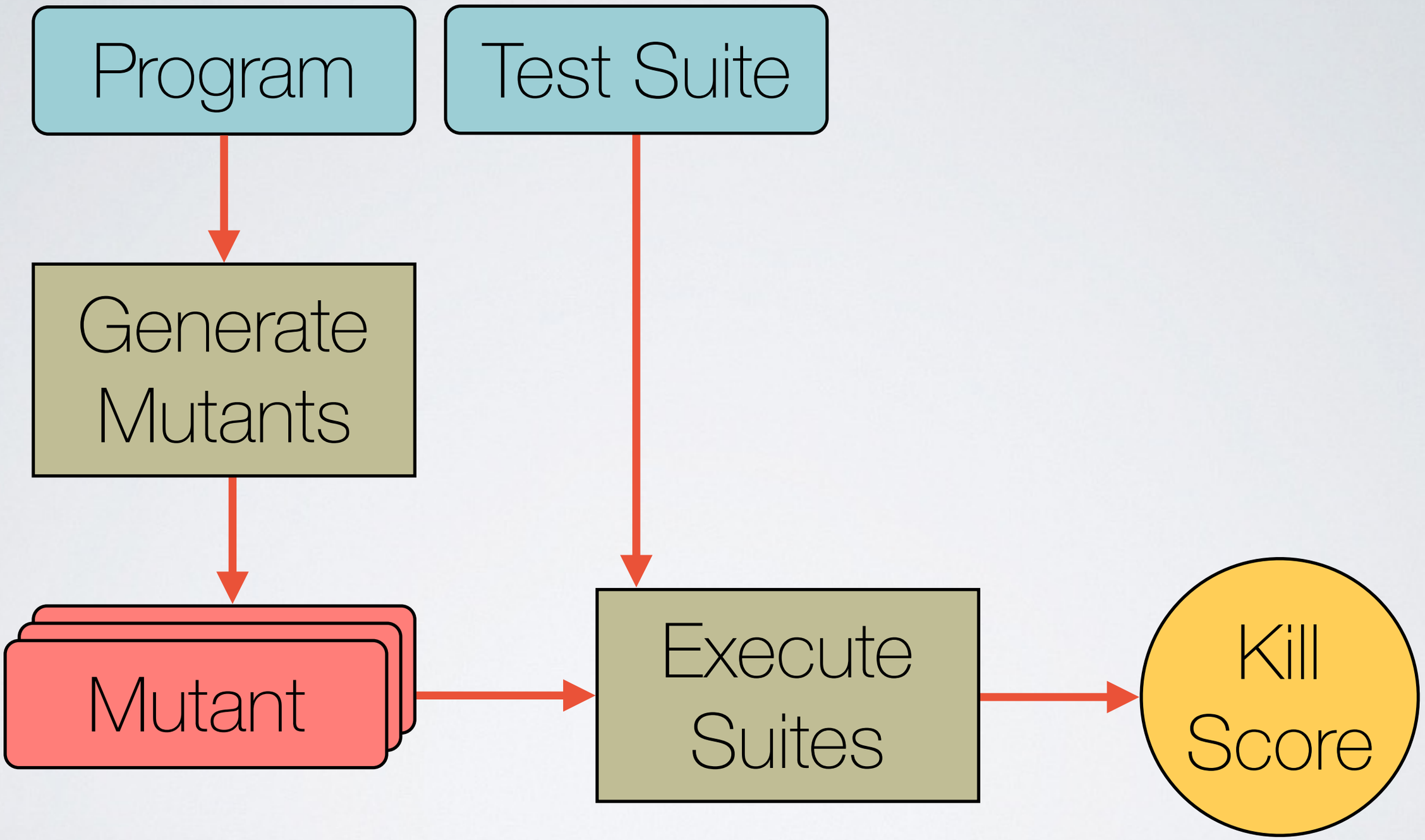

Program
Generate
Mutants


Program
Generate
Mutants


Program
Generate Mutants
Mutant


Program
Generate Mutants
Mutant


Program
Generate Mutants
Mutant


Program
Test Suite
Generate Mutants
Mutant
Execute Suites
Kill Score

what mutations?

flip boolean

boundaries (<, >=, etc)

remove conditional

increment to decrement

mutation operators

Conditional Boundary

<	—>	<=
<=	—>	<
>	—>	>=
>=	—>	>

```
if (a<b) {..}
—>
if (a<=b) {..}
```

mutation operators

Negate Conditionals

== —> !=

!= —> ==

```
if (a==b) {..}
—>
if (a!=b) {..}
```

...

mutation operators

Remove Conditionals

```
if (a==b) {..}
```

—>

```
if (true) {..}
```

mutation
operators

Math

\+ —> -

* —> /

| —> &

```
int a = b + c;
—>
int a = b - c;
```

...

mutation operators

Increments/Decrements

++ —> - -

- - —> ++

```
i++
—>
i—
```

mutation operators

Inline Constant

```
int i = 0;
—>
int i = 3;
```

mutation
operators

Return mutator

```
return o;
—>
return null;
```

mutation operators

Skip void calls

```
void somethingImportant(){..}
int foo() {
    int i = 5;
    somethingImportant();
    return i;
}
—>
int foo() {
    int i = 5;
    // somethingImportant();
    return i;
}
```

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

Test suite:

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

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

Test suite:

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

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

Test suite:

✖

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

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

Test suite:

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

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

Kill Score:
66%

Test suite:

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

```
sum = 0 -> sum = 1
```

```
sum += num -> sum += num
sum * length -> sum / length
```

New test:

✔ `assertEq(avg([1,1]), 1);`

Test suite:

✔ `assertEq(avg([1]), 1);`

```
sum = 0 -> sum = 1
```

```
sum += num -> sum += num
sum * length -> sum / length
```


should have been / not * all along

New test:

✔ `assertEq(avg([1,1]), 1);`

Test suite:

✔ `assertEq(avg([1]), 1);`

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

New test:

✔

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

Test suite:

✔

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

From the expected return of this function, this test should pass in the program; instead it reveals a fault in the program itself.

mutation assumptions

1) Competent Programmer Hypothesis:

—>Most programs are nearly correct.

2) Coupling Hypothesis:

—> Big bugs are composed of a series of small errors.

Assessing the quality of test suites

“If the program works
… on specified data,
then it will always
work on any data.
— Hoare

Mutation testing

Programmatic oracle

+

Correctness focus

Synthetic

Past studies:

Small programs

Few faults

Few mutants

	ISSTA 1996	ICSE 2005	FSE 2014
KLOC	1	6	321
Faults	12	38	357
Mutants	24	1,100	230,000
Tests	generated	generated	developer-written & generated
Coverage controlled	✖	✖	✔
Examine shortcomings	✖	✖	✔

			FSE 2014
KLOC			321
Faults	12	28	357
Mutants			30,000
Tests	generated	generated	ber-written & generated
Coverag controlle			✔
Examine shortcomings			✔

Do stronger tests detect more mutants?

Is mutant detection correlated with fault detection?

Can mutants describe all real faults?

Experimental method

Define Candidates

Compilable Faults

Triggering Tests

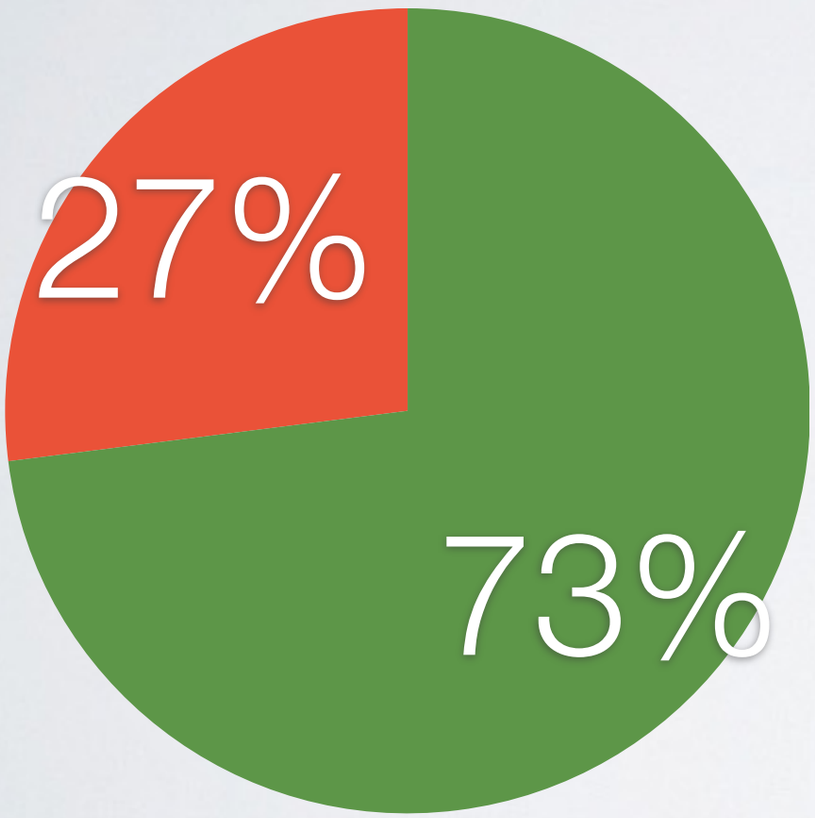
Analyze Misses

Generate Suites

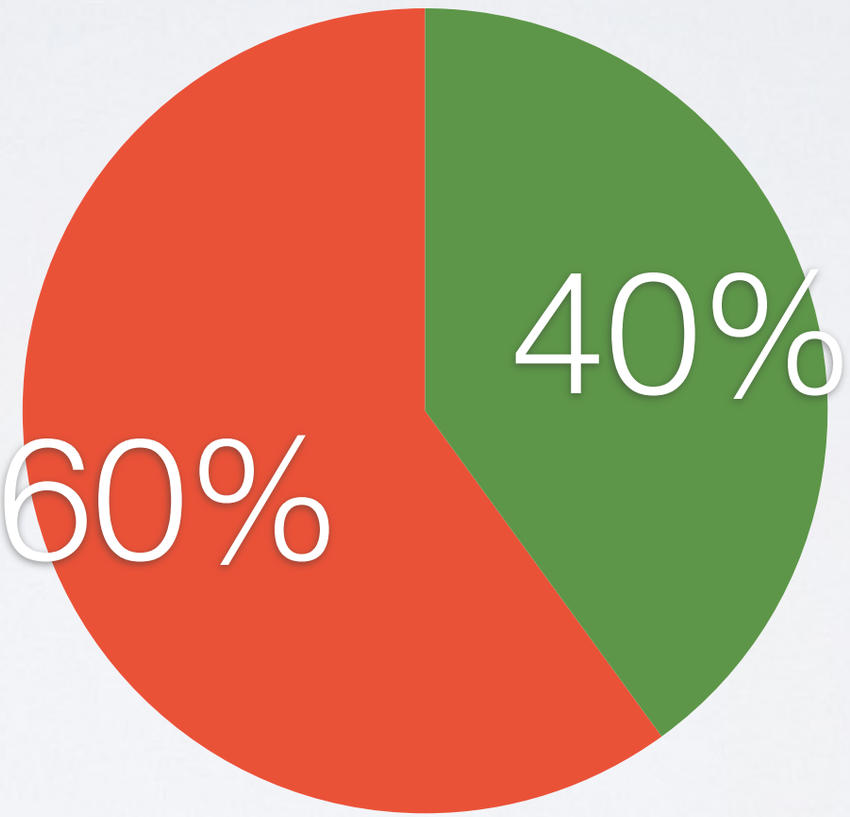
Do stronger tests detect more mutants?

Experimental results

Mutant detection



Statement coverage

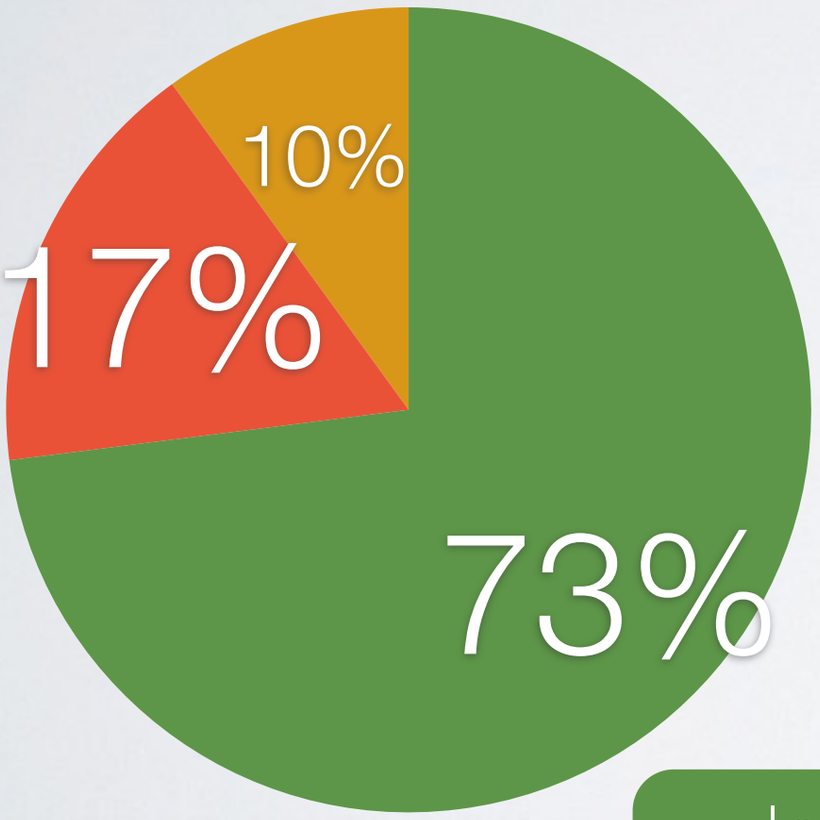


Increased

Unchanged

Experimental results

What kinds of faults are not represented by mutants?




```
if (x) {
    …
    return;
}
```

```
if (x) {
    …
    // del
}
```

```
if (cK.length !=
sD[0].length)
```

```
if (cK.length !=
getCatCount())
```

Increased

Weak/missing

No operator

Mutation
takeaway

A correlation exists between mutant detection and real fault detection.

Impact on testing

Adding tests can be more impactful than increasing coverage

Mutants can serve as effective proxies for real faults

Mutants can describe many real faults

Kill score is a better predictor of test quality than coverage

Stronger coverage criteria offer little additional insight

60% of real faults are already covered