Mutation Testing

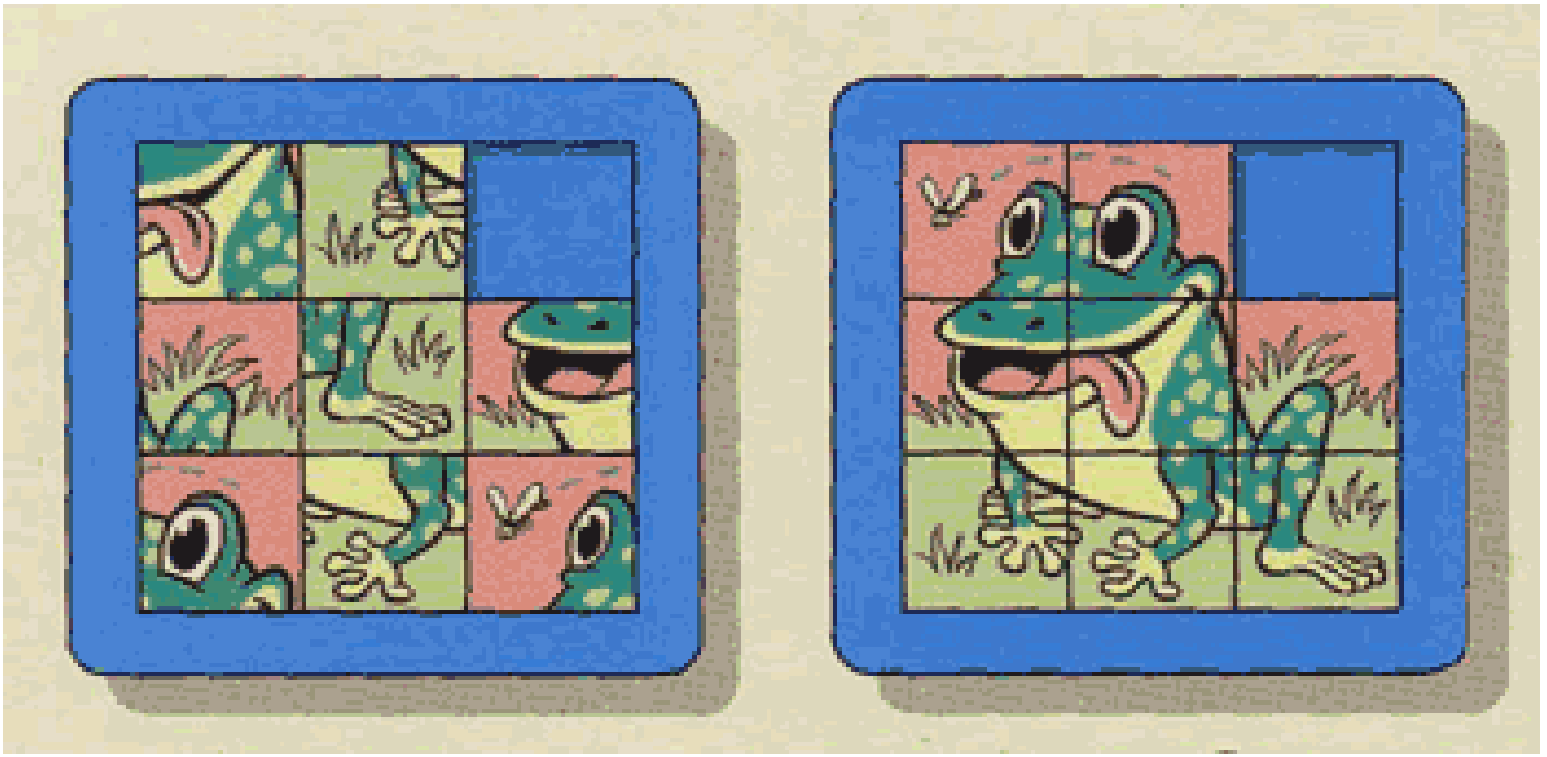
TOC

- What It Is?
- For What Purpose?
- How It Works?
- Types Of Mutations
- Code In Java
- Code In JavaScript
- Recommendations
- Future Directions

What is Mutation Testing?

- Mutation is Alteration
- So, it is a testing technique that relies on Program Alteration
- Aims to reveal a form of relationship between the specified behavior (through tests) and the actual code
- It tests your tests

What is Mutation Testing



Relies on program mutation/alteration/modification

What is Mutation Testing

Mutation testing tells you if your tests protect against regression

What is Mutation Testing

TESTSTHETESTS

Mutation From History

- Geist et al in “Estimation and Enhancement of Realtime Software Reliability through Mutation Analysis”:

  - *In practice, if the software contains a fault, there will be a set of mutants that can only be killed by a test case that also detects that fault*

Mutation From History

- K. Wah, 1995 in “Fault Coupling in finite bijective functions”:

  - *Complex faults are coupled to simple faults in such a way that a test data set that detects all simple faults in a program will detect most complex faults (coupling effect)*

Example Program Mutation

```
deposit(amount){
   balance += amount
}
```

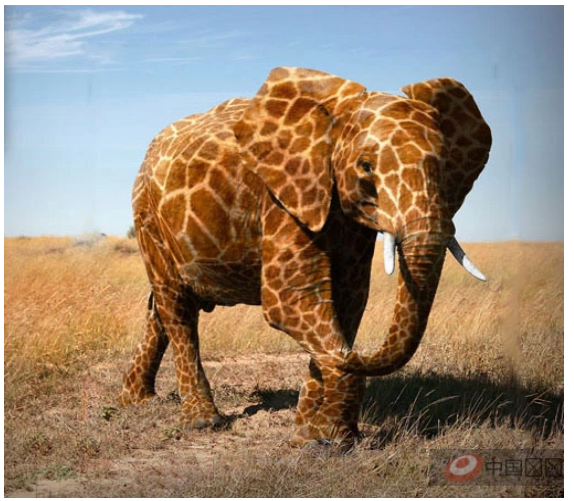
```
deposit(amount){
   balance -= amount
}
```

Example Program Mutation

```
getName(){
   return name
}
```

```
getName(){
   return null
}
```

Example Program Mutation



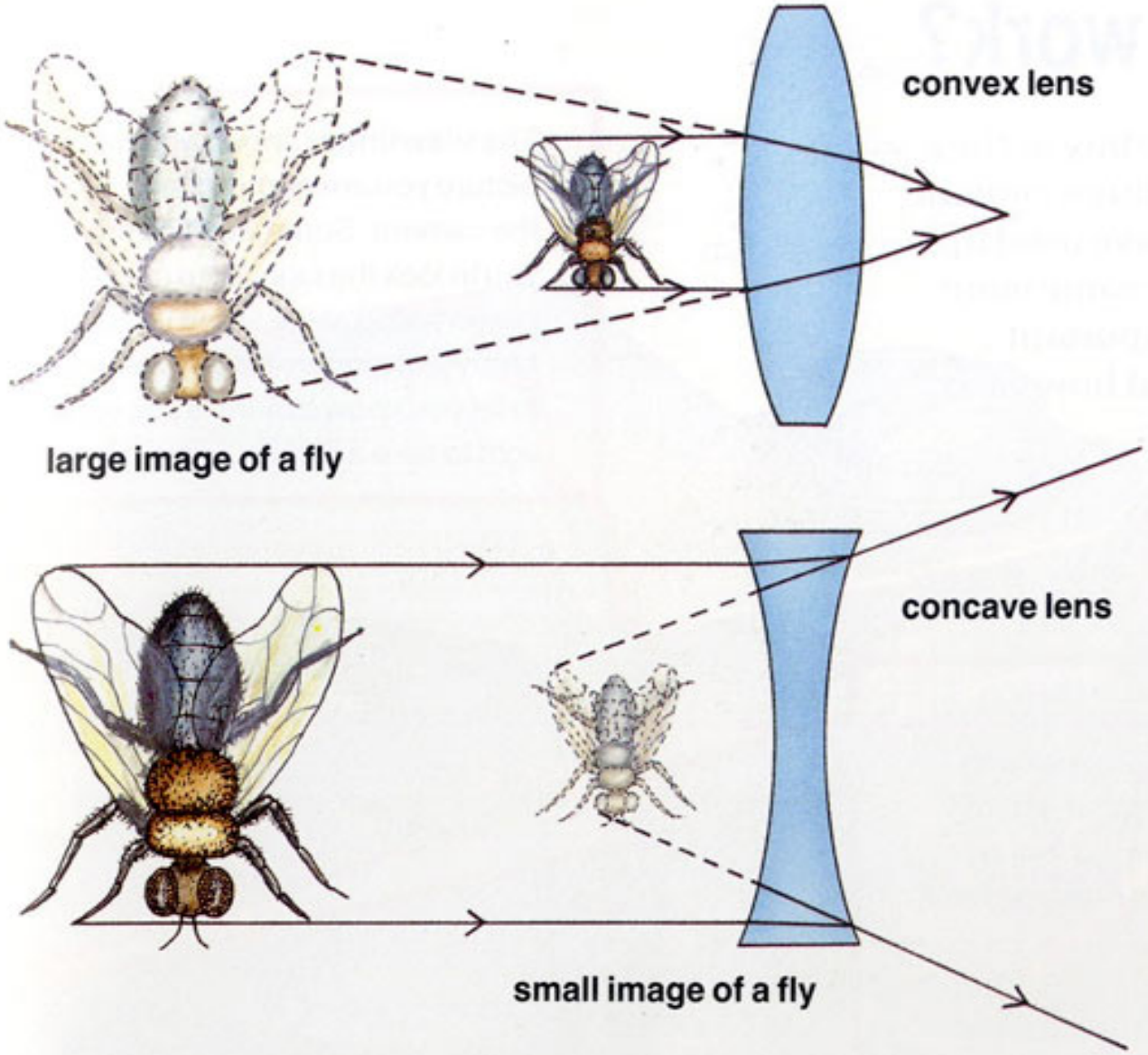
```
const int MIN_AGE = 18
```

```
const int MIN_AGE = 100
```

What about Test Coverage?

- Can be Useful, but can be just a cover up
- It relies on code line execution regardless of the (lack of) impact of that execution
- Aims at revealing a relationship between the specified behavior (through tests) and the line code execution
- Some methods do not generate mutants, but still need to be exercised (logging?)

What about Test Coverage?



Mutation Coverage

Convex lens

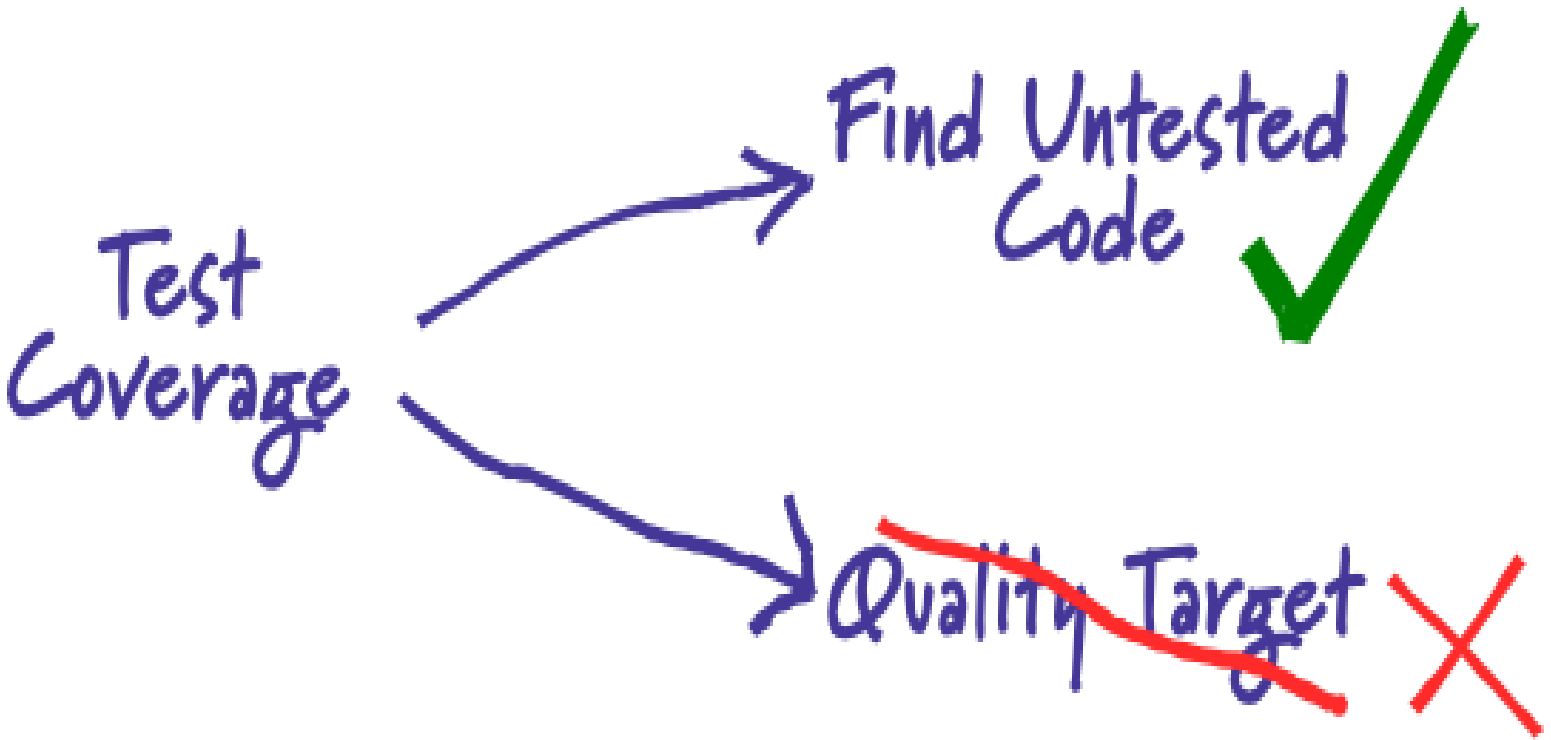
Look at an object, such as a fly, through a convex lens. The fly will be magnified and appear bigger than it really is.

Code Coverage

Concave lens

Look at a fly through a concave lens and it will appear smaller than it really is.

What about Test Coverage?



Example Mutation Relationship vs Coverage Relationship

```
boolean authenticate(User user) {
 authUsersCount++;
 LOGGER.info("user {} authenticated", user);
 authenticatedUsers.add(user);
 return true;
}

//assume this will pass as is (coverage is 100%, but mutation
//coverage? 50% only .. mmm ..
@Test
void test_users_can_be_authenticated(){
  User user = createUser();
  boolean result = authenticate(user);
  assertTrue(result);
  assertTrue(secService.getAuthenticatedUsers().contains(user));
}
```

Example Mutation Relationship vs Coverage Relationship

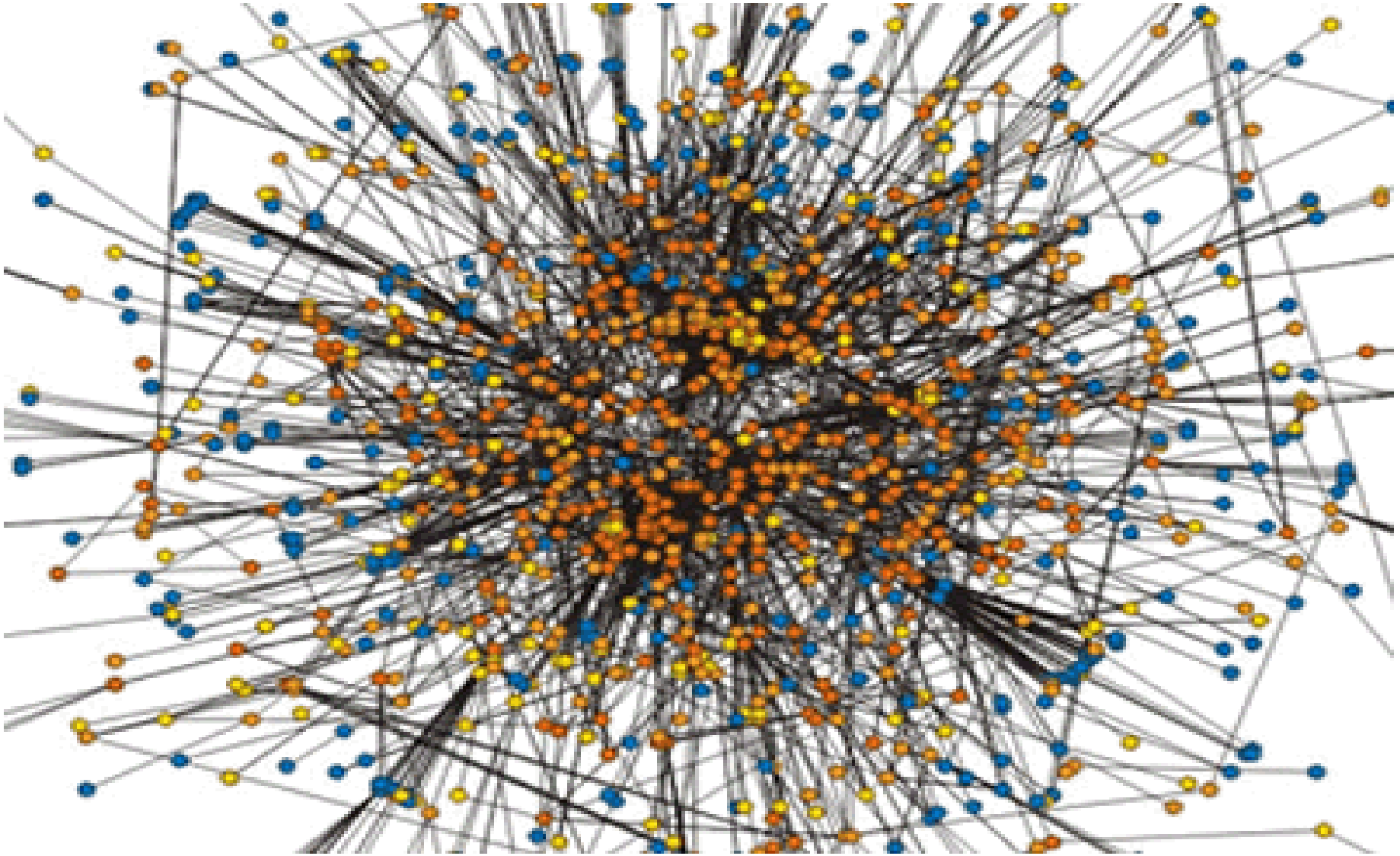
```
boolean authenticate(User user) {
 authUsersCount++;
 LOGGER.info("user {} authenticated", user);
 authenticatedUsers.add(user);
 return true;
}

//assume this will pass as is (coverage is 100%, but mutation
//coverage? 0% only .. mmm ..
@Test
void test_users_can_be_authenticated(){
  User user = createUser();
  boolean result = authenticate(user);
  //assertTrue(result);
  //assertTrue(secService.getAuthenticatedUsers().contains(user));
}
```

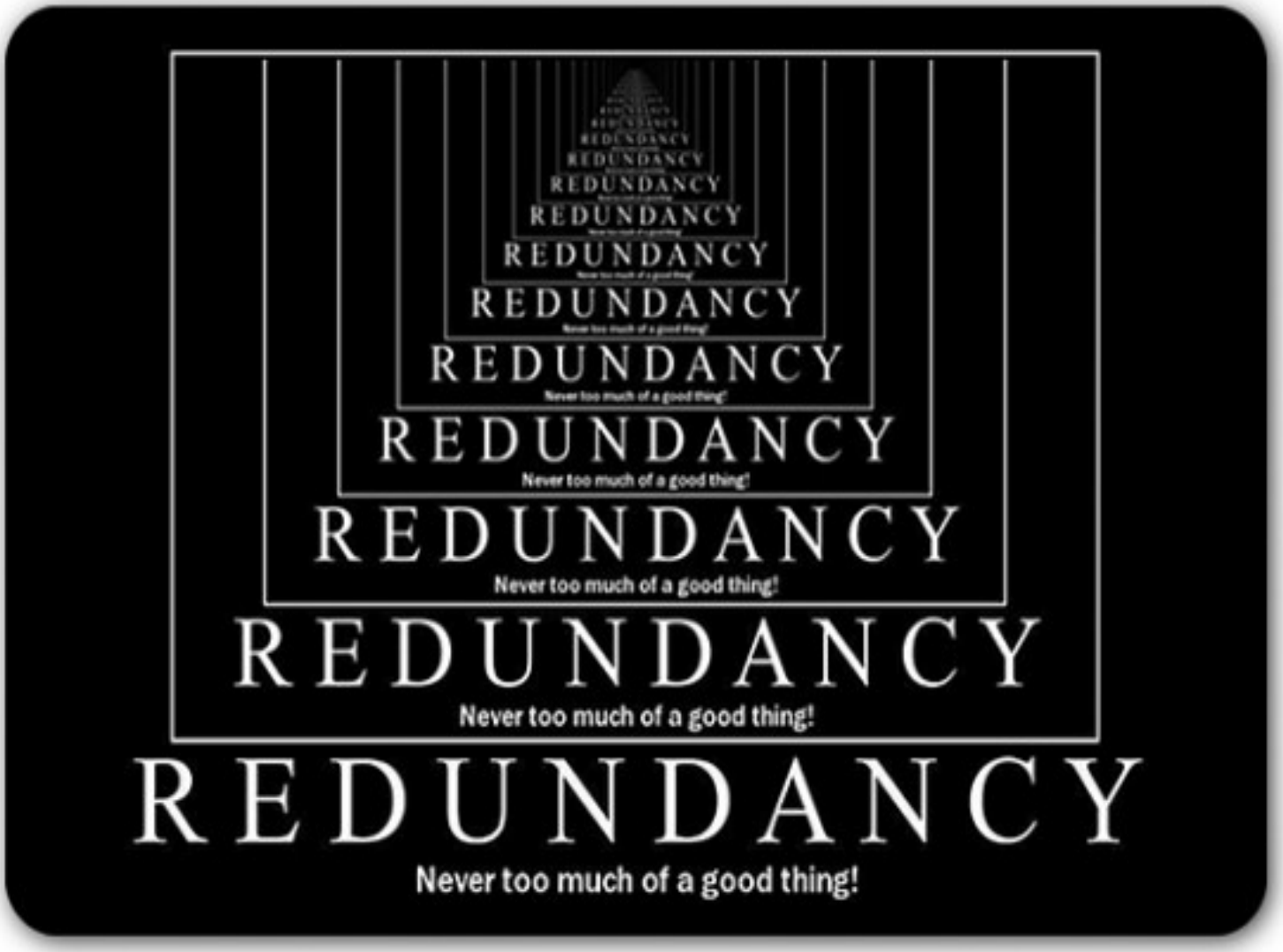
For What Purpose?

- You either have:
- Inadequate code:
  - ***Redundant code***
  - ***Complex code***
- Inadequate tests:
  - ***assertions are incomplete***
  - ***Scenarios are incomplete***
- So, helps understand the specs and the code better and deeper

For What Purpose?



For What Purpose?



For What Purpose?

Missing assertions

For What Purpose?

How It Works

- Mutation Killed: tests failed

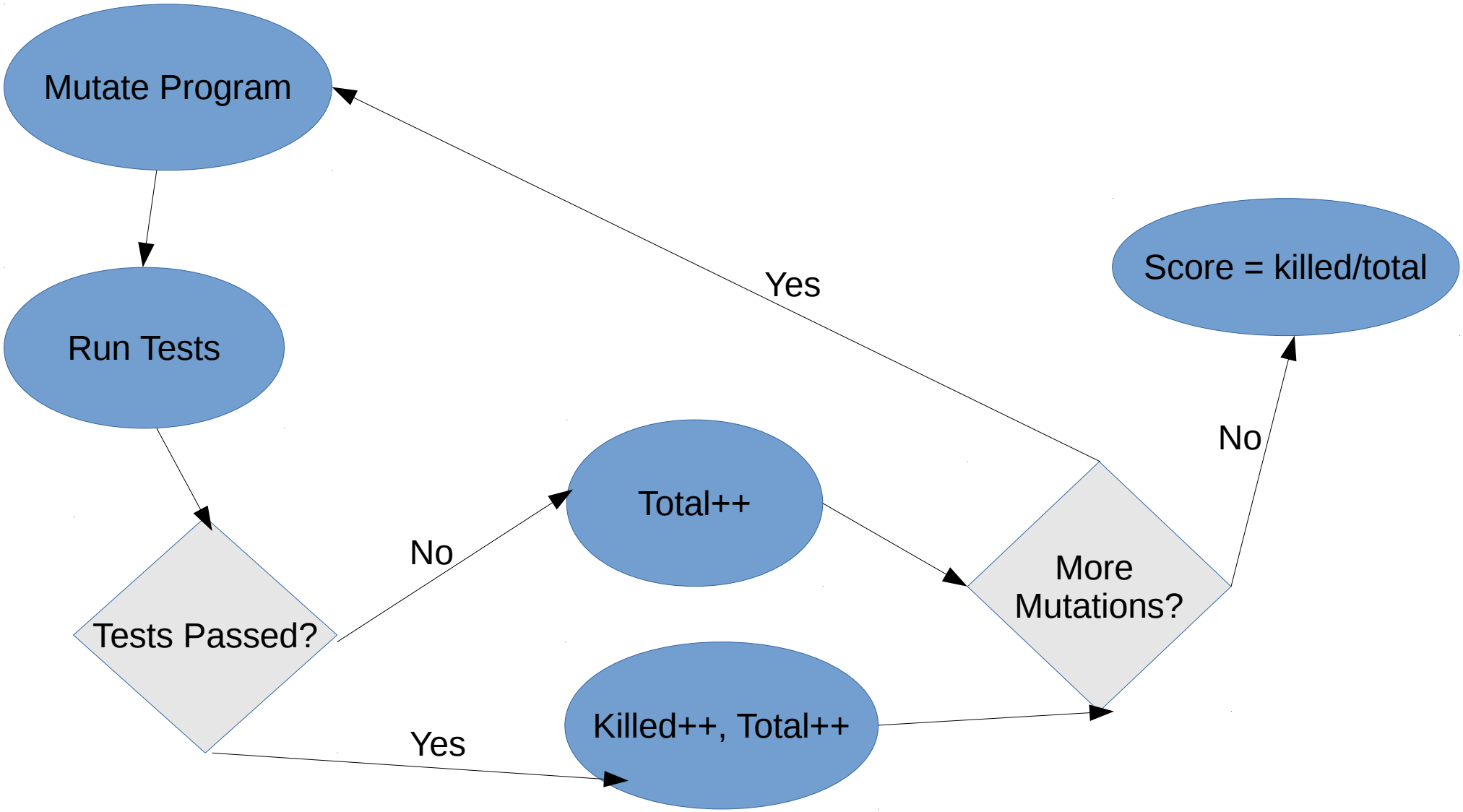
- Mutation Survived: tests succeeded

- We want the tests to always fail here (unlike coverage)

How It Works

- Mutate Program
- Run tests
- If mutation killed → increment killed and total counts
- If mutation survived → increment survived and total count
- If more mutations to attempt, go to 'Mutate Program'
- If no more mutations to attempt, calculate: mutation score = killed / total

How It Works


Mutate Program
Run Tests
Tests Passed?
No
Total++
Yes
Killed++, Total++
More Mutations?
Yes
No
Score = killed/total

Type Of Mutations

CONDITIONALS_BOUNDARY

`A > B`

to

`A < B`

Type Of Mutations

CONSTRUCTOR_CALLS

```
new X();
```

to

```
null;
```

Type Of Mutations

INCREMENTS

`i++`

to

`i--`

Type Of Mutations

INLINE_CONSTS

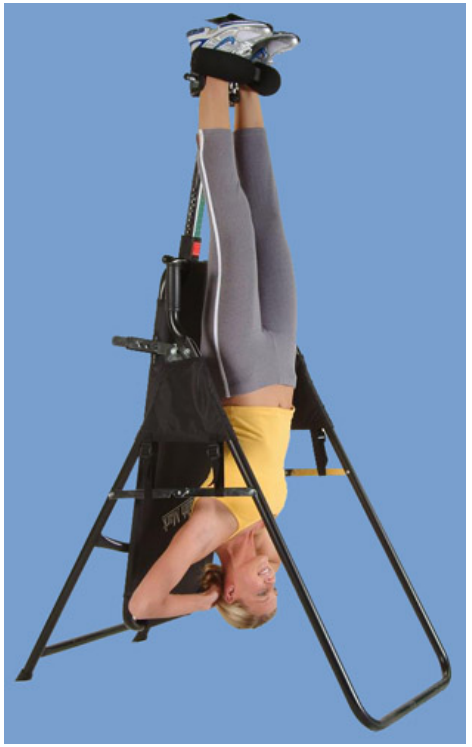
“A”

to

“B”

Type Of Mutations

INVERT_NEGS



–1

to

1

Type Of Mutations

MATH:

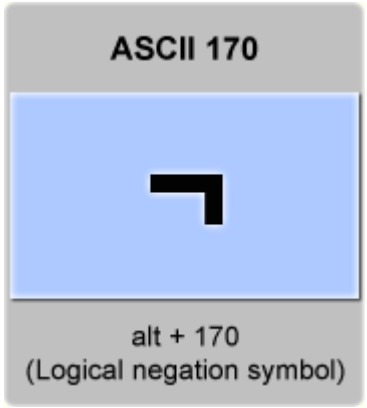
`a + b`

to

`a / b`

Type Of Mutations

NEGATE_CONDITIONALS




```
if(!condition())
```

to

```
if(condition())
```

Type Of Mutations

VOID_METHOD_CALLS

```
log()
```

to

```
//nothing
```

Type Of Mutations

NON_VOID_METHOD_CALLS

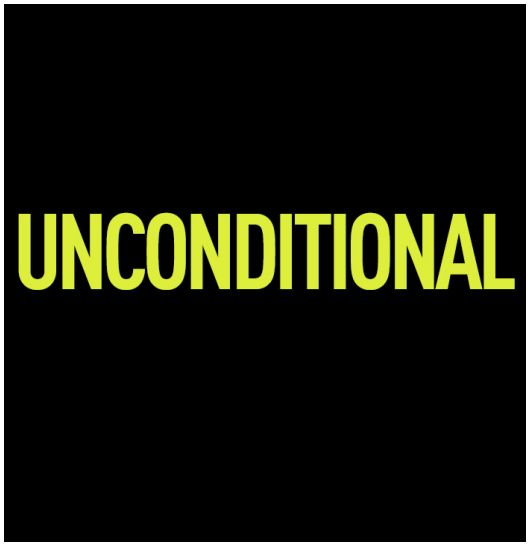
```
int a = b();
```

to

```
int a = 0;
```

Type Of Mutations

REMOVE_CONDITIONALS




```
if(a & b)
```

to

```
if(true)
```

Type Of Mutations

RETURN_VALS

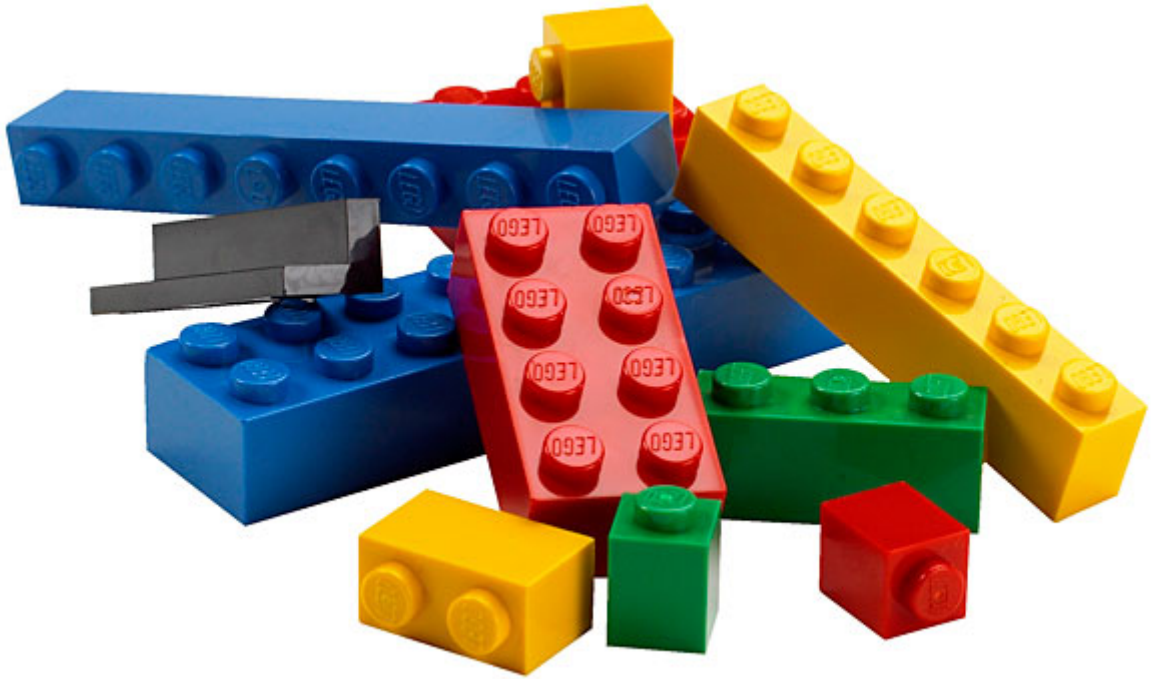
`return a`

to

`return null`

Type Of Mutations

- There can be more mutations and
- We can write our own



Recommendations

Do not use as a commit build step, might be too slow for now

Make it part of the pipeline though

Recommendations

Focus on mutations that makes sense to you

Recommendations

Automate and make easy to obtain mutation reports

Recommendations

Useful for thinking

Gives you insight into your code, even if not part of the build

Recommendations

Useful for thinking

Gives you insight into your code, even if not part of the build

Recommendations

Re-assess relationship with coverage

Do not focus on high coverage (remember we can achieve it with no assertions at all)

Recommendations

Think about other relationships that we might uncover between specifications (test or otherwise) and our code

Let's create meaningful relationships

Recommendations

Keep coverage up because
methods without coverage cannot kill any mutants

Some methods do not genuinely generate mutants (logging?)

Future Directions

We need to invest in the tools

make them faster, smarter, more flexible and more mature

Future Directions

We need to invest in the process

When and how to incorporate, what to replace, how to combine to get best ROI

Future Directions

We need to think of other ways to map code to requirements and understand the quality of automatic validation tools/techniques

We need to innovate

Future Directions

We need to educate

This is a new way of doing things

Socializing the ideas (more important than the tools) is fundamental to improving things on the ground

Questions, Feedback, and Debate