Testing What Must Work

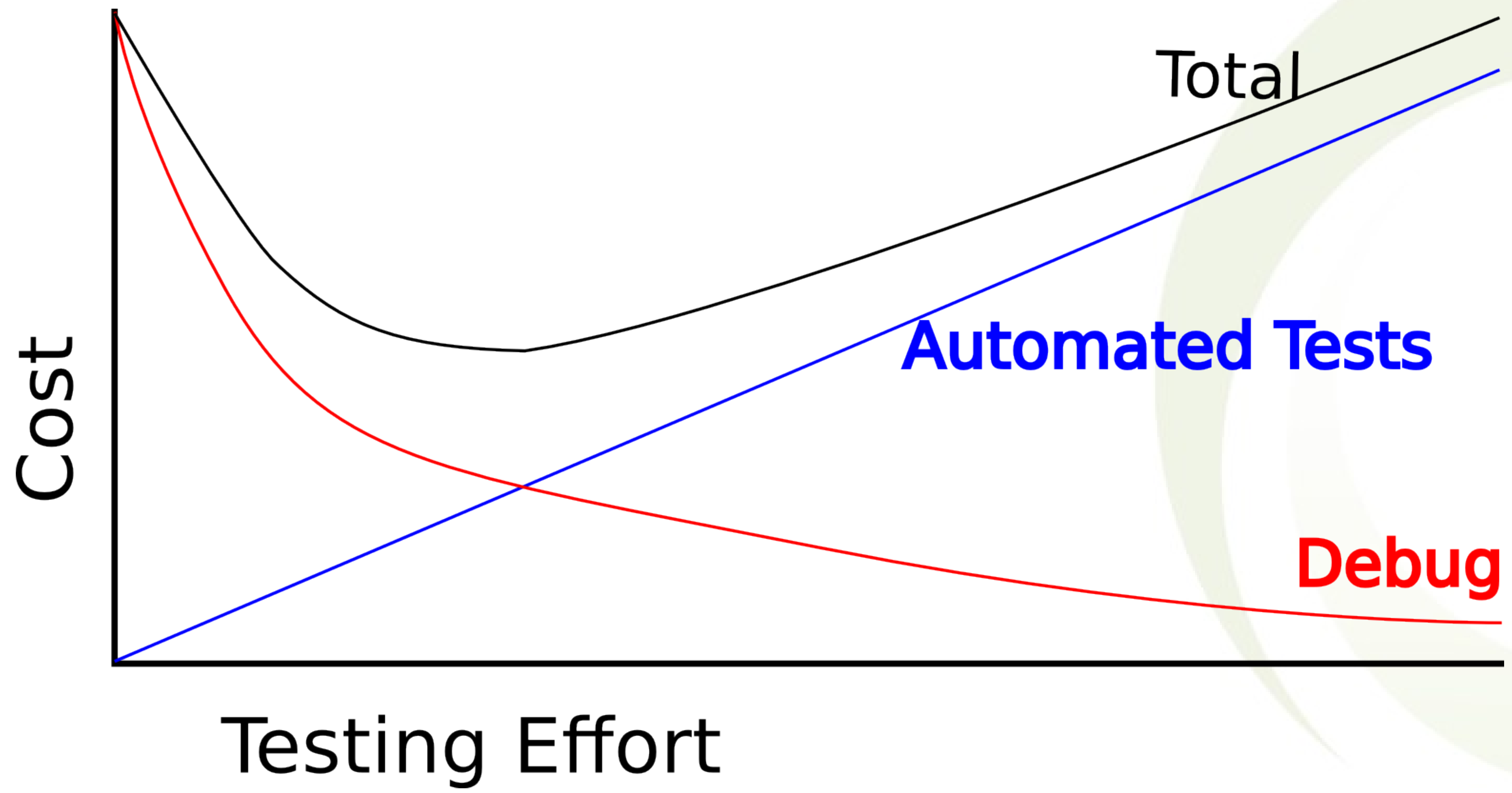
Not Only What Souldn't Fail

by Samuel Rivas

samuel.rivas@interoud.com

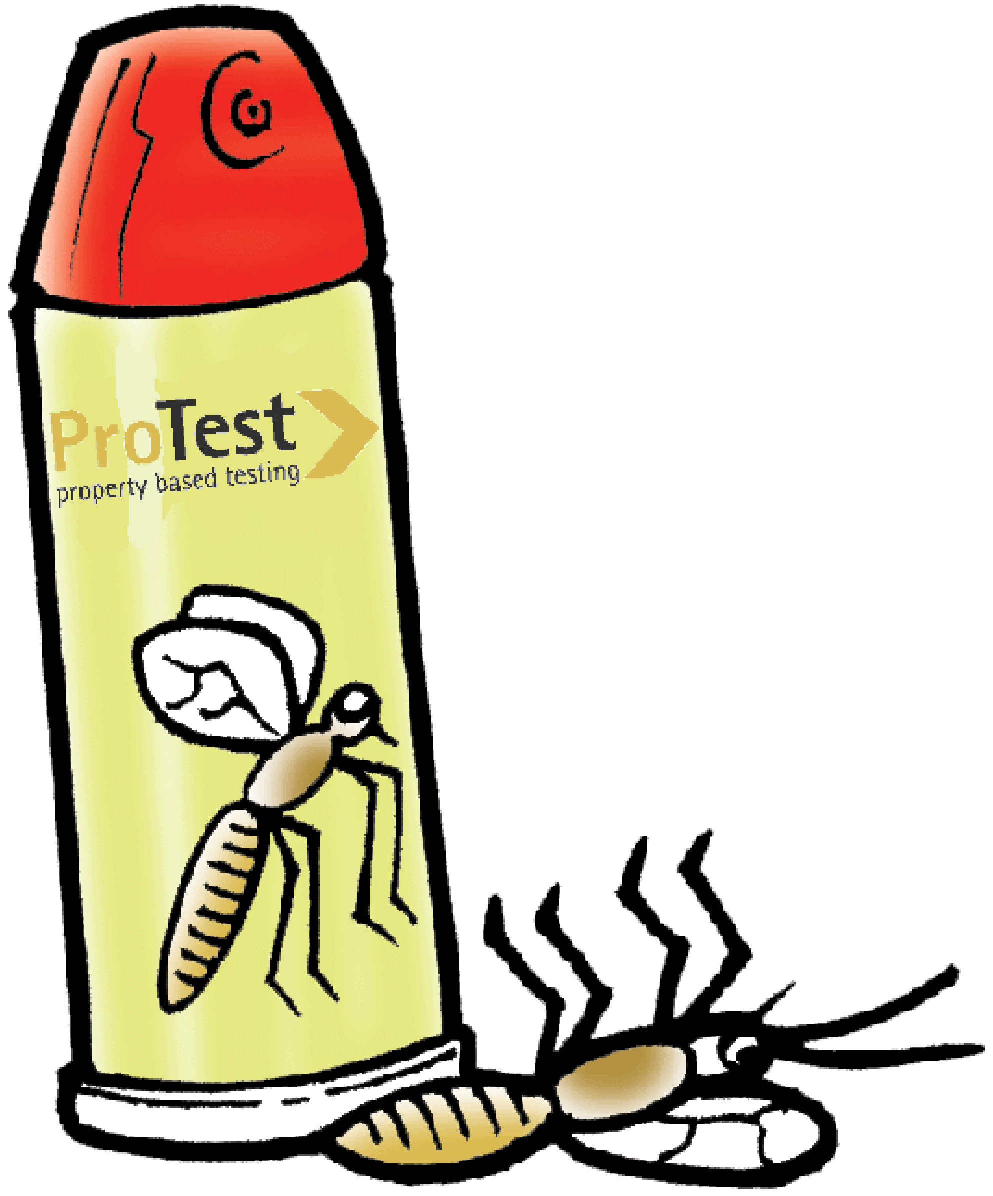
Testing plays an important role in development, specially when many developers team together

You know testing is a powerful tool, what you want to know is how to test more effectively



Even with zealous TDD, skilled development teams can still introduce bugs without doing anything stupid

You can use property based testing to build stronger tests



You need to understand what are the advantages of property based testing to apply it effectively

Testing

a classical story

Traditional test cases are concrete samples of behaviour that developers, not computers, will generalise

√ sum(0,0) == 0

√ sum(0,1) == 1

√ sum (0,2) == 2

√ sum(1,1) == 2

√ sum(1,2) == 3

```
Sum(0,0) → 0;
Sum(0,1) → 1;
Sum(0,2) → 2;
Sum(1,1) → 2;
Sum(1,2) → 3.
```

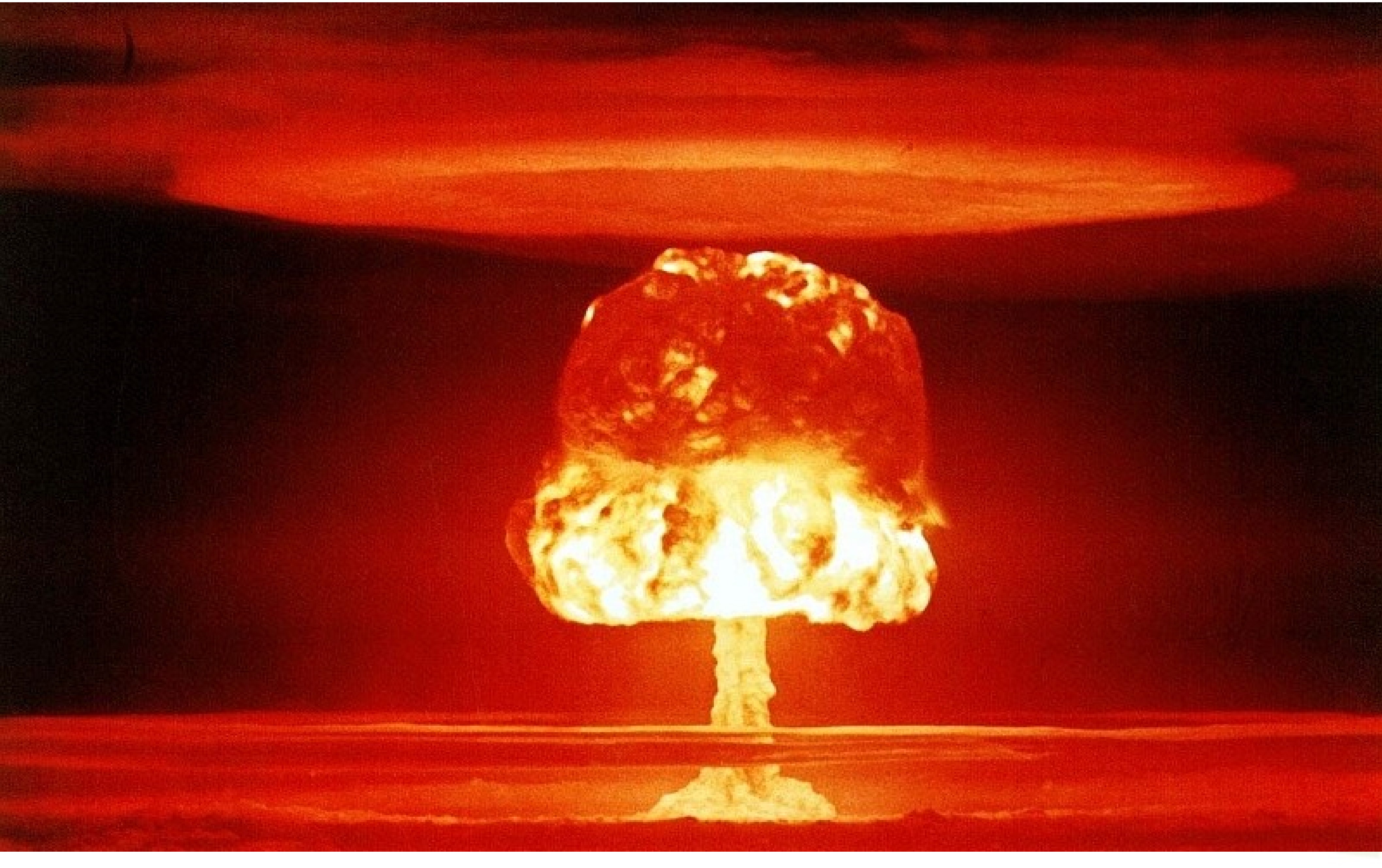
A team builds a suite of test cases, and works towards a generic solution that passes all of them

```
?assertEqual(
  "Erlang rulez!",
  template:apply(
    "@lang@ rulez!",[{"lang","Erlang"}])
```

MONEY

A second team enhances the program, passing new tests plus the former ones

```
?assertEqual("@",template:apply("@@",[]))
```



The new feature interacts with the old one in a way existing tests don't describe

```
1> template:apply(
     "@name@@@@domain@",
     [{"name", "samuel.rivas" },
     {"domain", "interoud.com"}]).
** exception error:
   {variable_not_found,"name@@domain"}
```

Testing Reloaded

A Property Based Testing Remake

We would like to describe the computer the behaviour we want, and let it test it automatically

For all X, X+0 == X

For all X, X+1 == next(X)

For all X, Y, X+(Y+1) == (X+Y)+1

~~Sum(0,0) → 0;~~
~~Sum(0,1) → 1;~~
~~Sum(0,2) → 2;~~
~~Sum(1,1) → 2;~~
~~Sum(1,2) → 3.~~

A team gets a description of the same template library and writes a property

For all template T, list of variables X, and list of substitutions X', apply(T, X) yields T with X values switched to those of X'

Extending the library involves extending the property

For all template T, list of variables X,
and list of substitutions X', apply(T, X) yields
T with X values switched to those of X'
and all escaped at symbols
turned into at symbols

The new property doesn't allow the bug slip through like in the previous story

```
Template: "@ @@ @"
Substs : [{" ", ""}]
Expected: ""
Got: {error, {variable_not_found," @ "}}
```

ProTest researchers explored different practical ways of achieving this type of testing

ProTest
property based testing

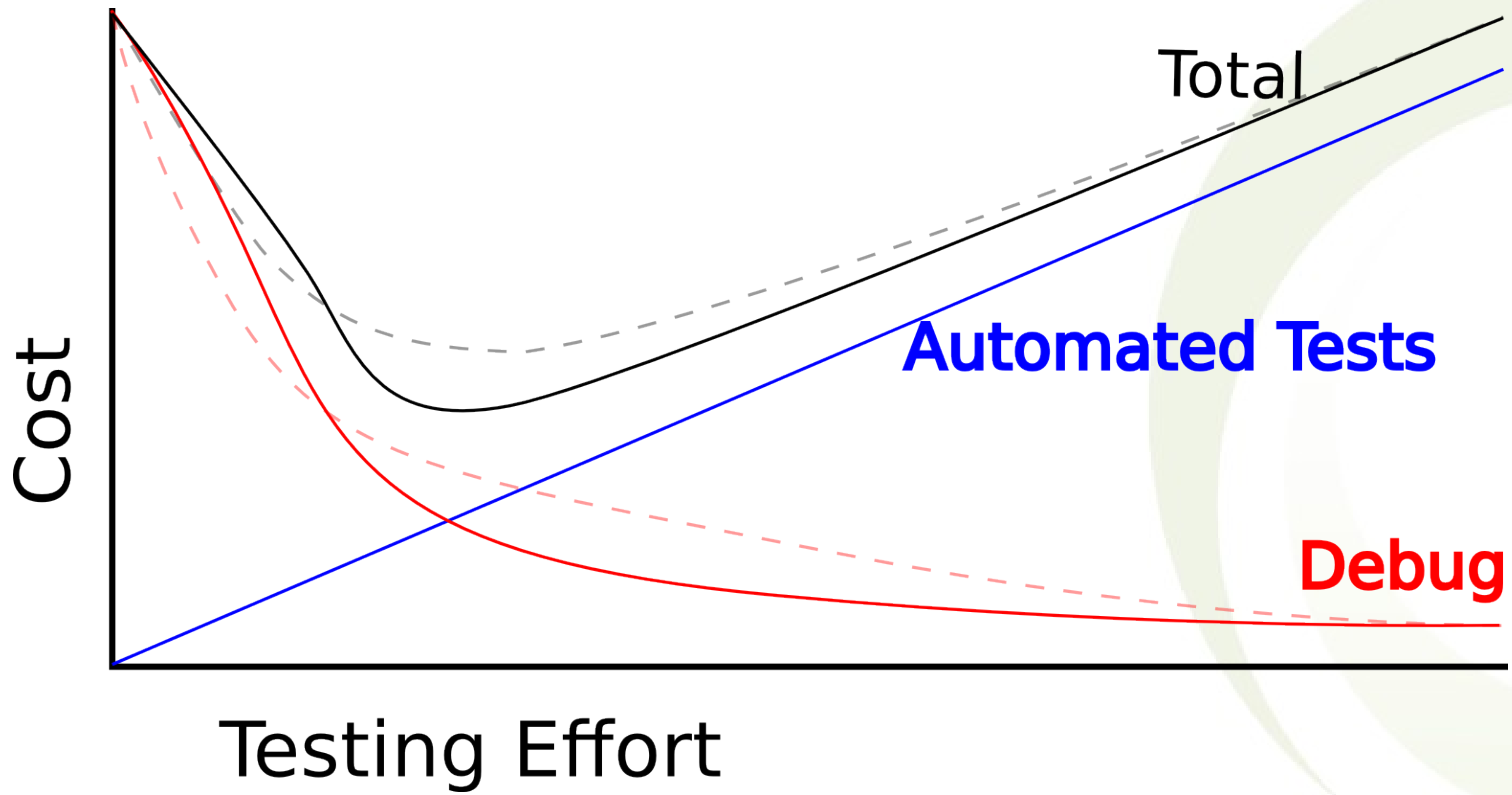
With QuickCheck we define data generators so that the computer can randomly look for a counterexample

```
?FORALL(T,template(),
  template:apply(
    to_string(T), to_subs(T))
  == to_result(T)).
```

The trick in this case is that we derive inputs and outputs from the same abstract representation

$$\begin{tikzcd}
 & \text{template()} \arrow[dl, "\text{to\_strting}"'] \arrow[d, "\text{to\_subs}"] \arrow[dr, "\text{to\_result}"] & \\
\text{"Hi @name@"} \arrow[dr] & \text{[\{"name","Samuel"\}]} \arrow[d] & \text{"Hi Samuel"} \arrow[dl, leftrightarrow, "=="] \\
 & \text{apply("Hi @name@", [\{"name","Samuel"\}])} &
\end{tikzcd}$$

You know testing is a powerful tool, what you want to know is how to test more effectively



You need to understand what are the advantages of property based testing to apply it effectively

interoud
innovation

Thank You!