QuviQ

QuickCheck Mini for Elixir

Thomas Arts
Quviq AB

Most developers agree that writing unit tests is useful

…. but also quickly gets boring …

An example:

Does Erlang lists:seq(N,M) do the same as

Enum.to_list(n .. m) ?

Write some unit tests

```
defmodule Examples do
  use ExUnit.Case

  test "Erlang seq the same as Enum.to_list" do
    assert :lists.seq(1, 5) == Enum.to_list(1 .. 5)
    assert :lists.seq(-10, 10) == Enum.to_list(-10 .. 10)
    assert :lists.seq(164532, 164532) ==
                          Enum.to_list(164532 .. 164532)
   end

end
```

Automated Unit tests:

```
assert :lists.seq(1, 5) == Enum.to_list(1 .. 5)
assert :lists.seq(-10, 10) == Enum.to_list(-10 .. 10)
assert :lists.seq(164532, 164532) ==
                      Enum.to_list(164532 .. 164532)
```

What is so specific for these values?

How many tests shall we write?

Generate tests from a property

```
defmodule ExamplesEqc do
  use ExUnit.Case
  use EQC.ExUnit

   property "Erlang Sequence" do
        forall {m, n} <- {int, int} do
          ensure :lists.seq(m, n) == Enum.to_list(m .. n)
        end
   end

end
```

int is a generator for an arbitrary integer value.

live demo

Automatically generated test cases

```
......Failed! After 7 tests.
{1,0}
not ensured: [] == [1, 0]

 1) test Property Erlang Sequence (ExamplesEqc)
     test/examples_eqc.exs:5
     forall({m, n} <- {int, int}) do
       ensure(:lists.seq(m, n) == Enum.to_list(m .. n))
     end
     Failed for {1, 0}
```

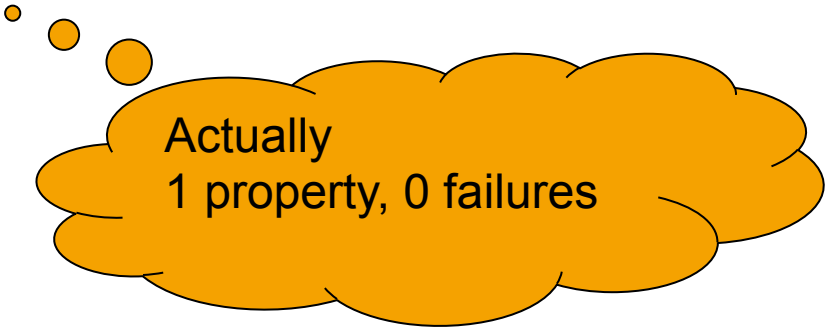
Finished in 0.1 seconds (0.06s on load, 0.05s on tests)
1 tests, 1 failures

Generate tests from a property

```
defmodule ExamplesEqc do
  use ExUnit.Case
  use EQC.ExUnit

   property "Erlang Sequence" do
        forall {m, n} <- {int, int} do
          implies n >= m do
            ensure :lists.seq(m, n) == Enum.to_list(m .. n)
         end
        end
   end

end
```

Automatically generated test cases

```
.....x..xx..x.xxx...xx.....xx..xx.x.x..xx...x.xx...x...x.xxx..x..x
xxxx.x.x.....x.x....xxx.x.x.xx.x.x(x10).x..x.x(x1)....
OK, passed 100 tests
.

Finished in 1.2 seconds (0.06s on load, 1.2s on tests)
1 test, 0 failures
```



Properties of software

A property is something that should hold

... for any valid input you supply

The classic example [ICFP2000 QuickCheck paper]

Reversing a list twice gives the original list

```
property "Reversing a list" do
  forall l <- list(int) do
      ensure Enum.reverse(Enum.reverse(l)) == l
  end
end
```

QuickCheck for Haskell since 2000

QuickCheck for Erlang since 2006

Now available for each language... also Elixir.

Not using QuickCheck is violating the rule of using "best practice"

Download QuickCheck Mini from www.quviq.com
and example from github.com/ThomasArts/Mini

General properties

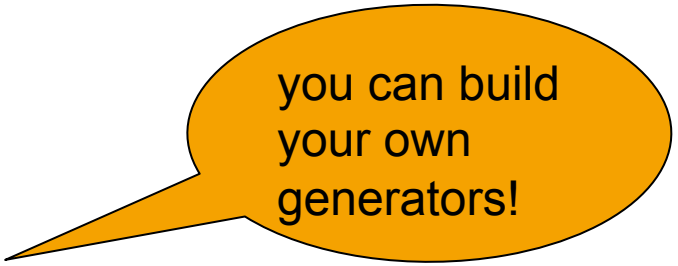
Well designed software has certain patterns... use these for testing.

Examples

- Inverse operations
- Idempotent operations
- equivalence properties
- action/observe properties

Inverse properties

```
property "List reverse" do
    forall l <- list(int) do
      ensure Enum.reverse(Enum.reverse(l)) == l
    end
  end
```




```
property "Base64" do
   forall str <- list(char()) do
      :base64.decode(:base64.encode(str)) == str
    end
end
```

Idempotent properties

```
property "Sorting" do
  forall l <- list(int) do
    Enum.sort(Enum.sort(l)) == Enum.sort(l)
  end
end
```

equivalence properties

```
property "Erlang Sequence" do
   forall {m, n} <- {int, int} do
     ensure :lists.seq(m, n) ==
                 Enum.to_list(m .. n)
   end
end
```

action/observe properties

```
property "Sorting" do
    forall l <- list(int) do
      is_sorted(Enum.sort(l))
    end
  end
```

Add eqc_ex to your mix dependencies

```
defmodule Mini.Mixfile do
  use Mix.Project

  def project do
    [ .....
      test_pattern: "*_{test,eqc}.exs"]
  end

  defp deps do
    [ .....
      {:eqc_ex, "~> 1.2.4"} ]
  end

  # Set the environment variable ERL_LIBS to PATH_OF QuickCheck Mini
end
```

Reversing things

```
defmodule StringEqc do
  use ExUnit.Case
  use EQC.ExUnit

  property "Reverse strings" do
    forall string <- utf8 do
      ensure String.reverse(String.reverse(string)) == string
    end
  end

end
```

A generator for utf8 strings

Idempotence

Reversing things

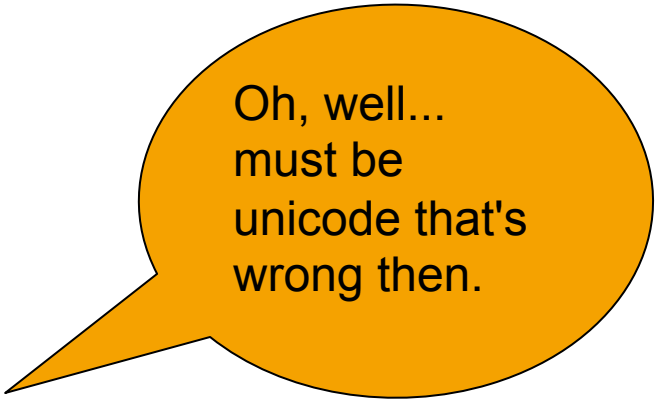
Oeps ☹

```
..................Failed! After 20 tests.
<<"ß«I">>
not ensured: "Iß«" == "ß«I"
Shrinking xxxxxxxxxxxxxxxxxxxxxxxx.....xxxxxxxxxxxxxxxxxx(5
times)
<<"ß« ">>
not ensured: " ß«" == "ß« "
```

```
1) test Property Reverse strings (StringEqc)
   test/string_eqc.exs:5
   forall(string <- utf8) do
     ensure String.reverse(String.reverse(string)) == string
   end
   Failed for "
```

This is fishy!

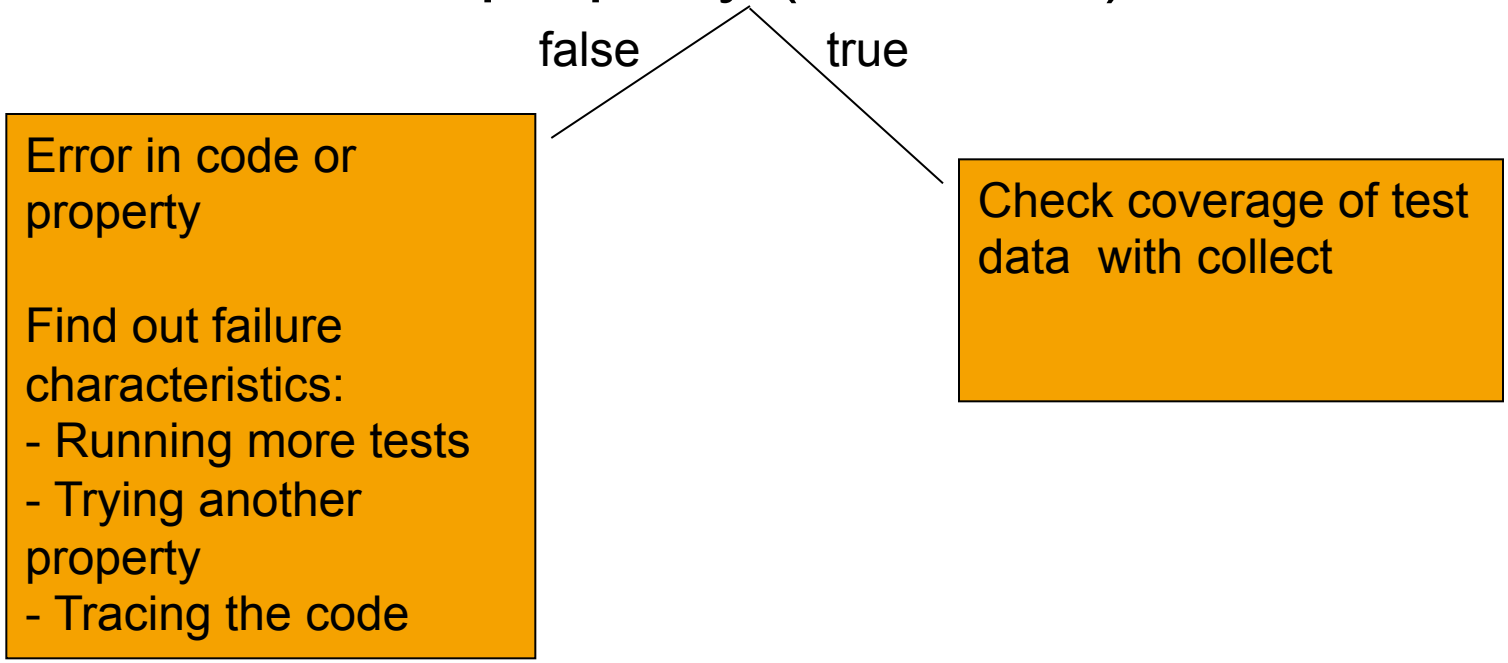
For the non-believers...




```
defmodule StringTest do
  use ExUnit.Case

  test "Reverse strings" do
    string = "\x{07eb} "
    assert String.valid?(string)
    assert String.valid?(String.reverse(string))
    assert String.reverse(String.reverse(string)) == string
  end

end
```

Practical use of QuickCheck

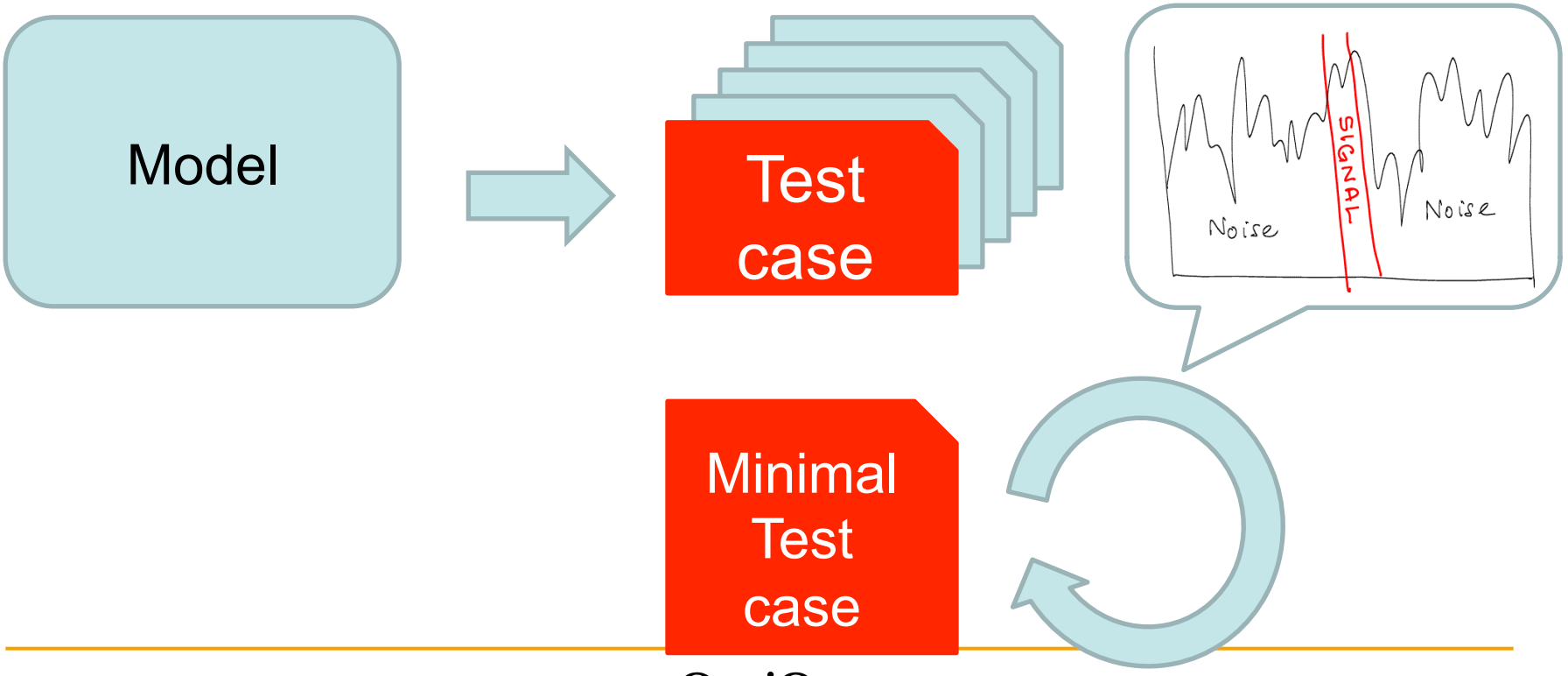
1. Consider which property should hold (not which test should pass)
2. Check the property (100 tests)



QuickCheck: property-based testing

Software has certain properties... things that should always hold for that software

QuickCheck is a tool that automatically generates test cases from these properties.



Benefits

- Less time spent writing test code
  - One model replaces many tests
- Better testing
  - Lots of combinations you'd never test by hand
- Less time spent on diagnosis
  - Failures minimized automatically