For All Code, There Exist Properties to be Checked

(or, random testing with FsCheck)

Paulmichael Blasucci, Senior Software Engineer at

Quicken Loans
Engineered to Amaze

twitter.com/pblasucci github.com/pblasucci linkedin.com/in/pblasucci

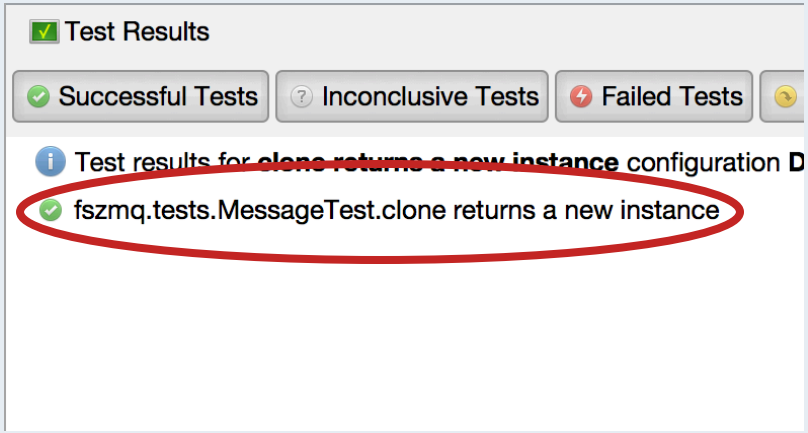
Random Testing

"Properties are described as … **functions**, and can be **automatically tested on random input**… [or] custom test data generators."

from ICFP'00 – Claessen, Hughes

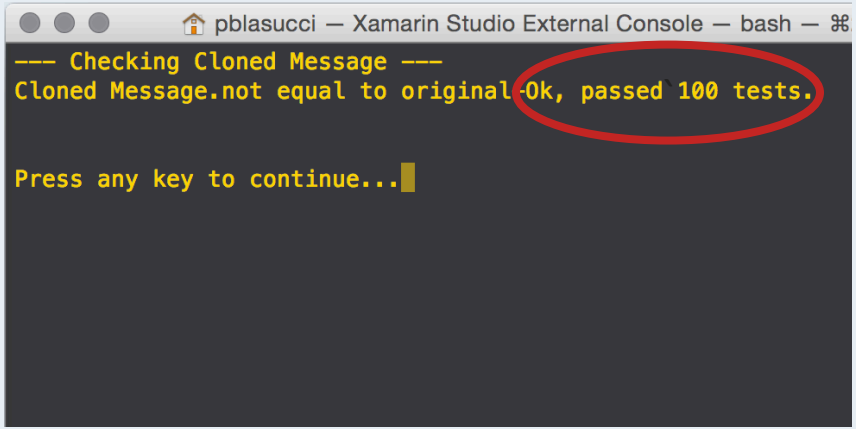
From Unit Testing…

```
[<Test>]
let ``clone returns a new instance`` ()
  use msg1 = new Message("test"B)
  use msg2 = Message.clone msg1
  Assert.That (msg2,Is.Not.EqualTo msg1)
```



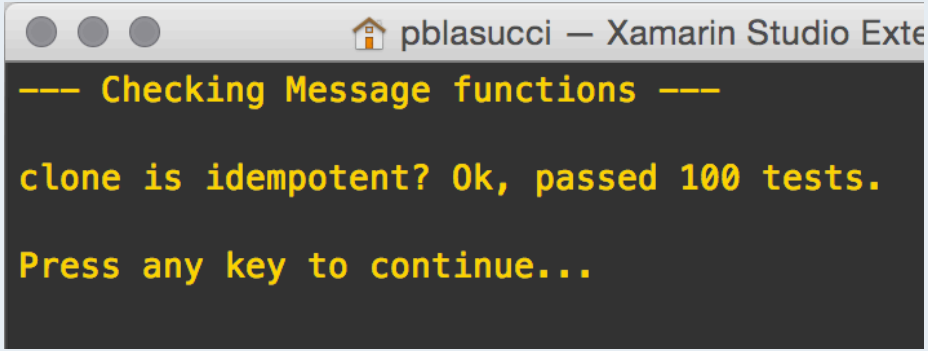
To Property Testing

```
[<Property>]
let ``not equal to original`` data =
  use msg1 = new Message (data)
  use msg2 = Message.clone msg1
  msg1 <> msg2
```



FsCheck: Properties

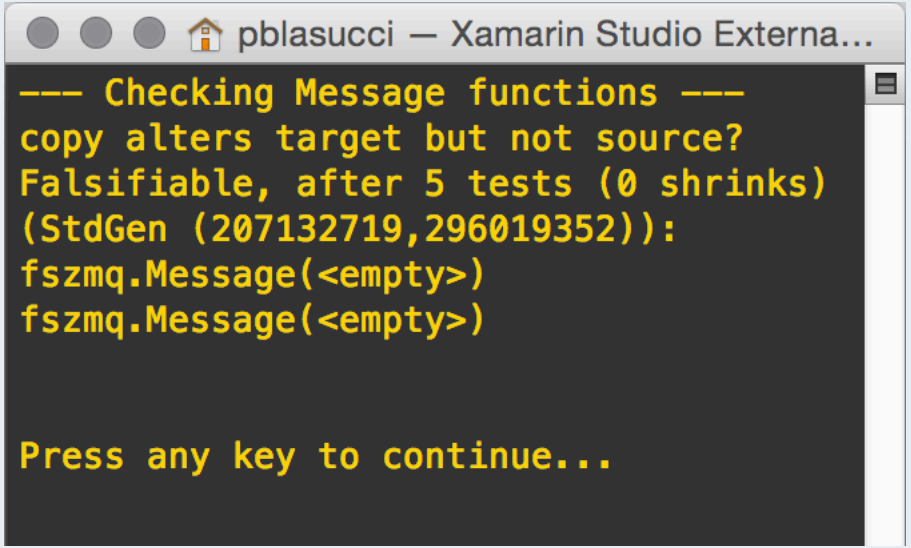
```
open fszmq
open fszmq.Message

let equalContent (msg1:Message) (msg2:Message) =
  size msg1 = size msg2 && data msg1 = data msg2

let ``clone is idempotent`` (msg:Message) =
  use once  = clone msg
  use twice = msg
              |> clone
              |> clone
  twice |> equalContent once
```



FsCheck: Properties

Combining properties (naïvely)

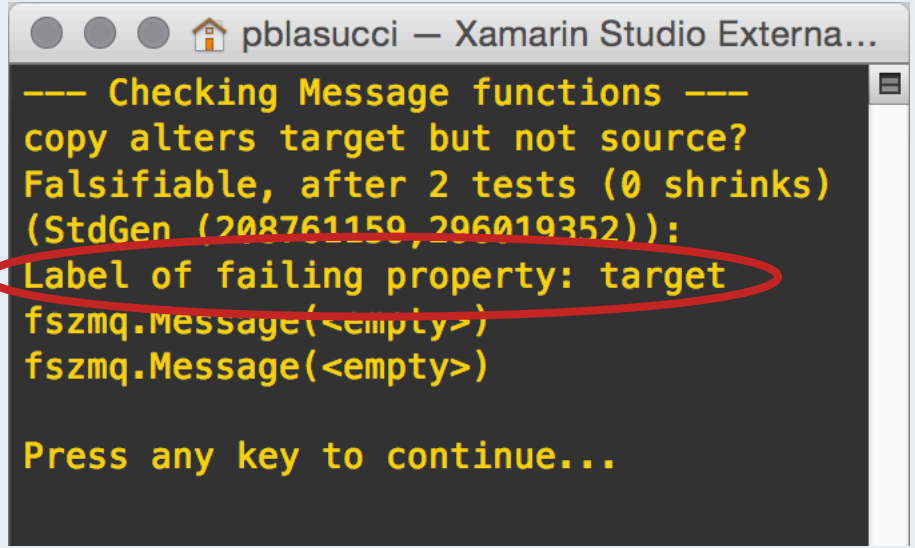
```
let ``copy alters target but not source`` msg1 msg2 =
  let data1,data2 = data msg1,data msg2
  copy msg1 msg2
  (data msg1 = data1) && (data msg2 <> data2)
```



FsCheck: Properties

Combining and ***Labeling*** properties

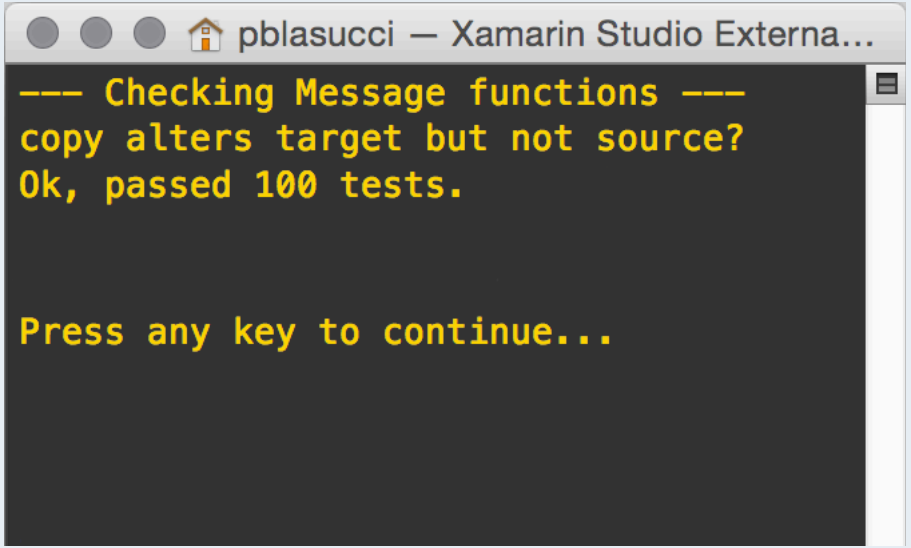
```
let ``copy alters target but not source`` msg1 msg2 =
  let data1,data2 = data msg1,data msg2
  copy msg1 msg2
  (data msg1 =  data1 |@ "source")
  .&.
  (data msg2 <> data2 |@ "target")
```

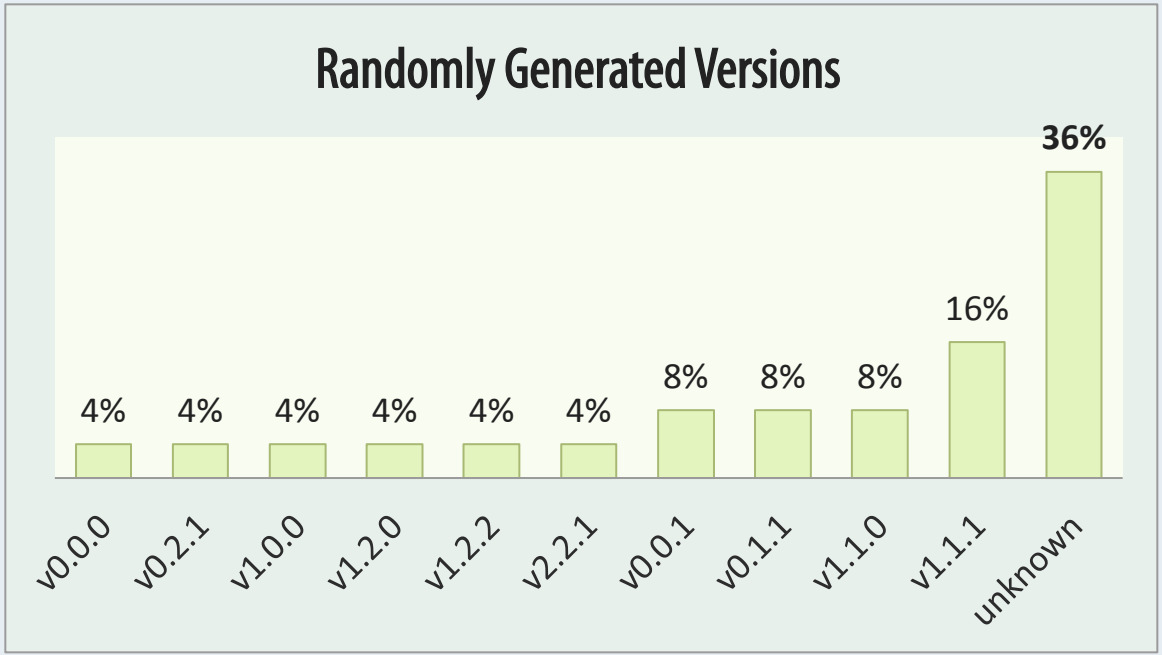


FsCheck: Properties

Conditional properties

```
let ``copy alters target but not source`` msg1 msg2 =
  (data msg1 <> data msg2) ==>
    let data1,data2 = data msg1, data msg2
    copy msg1 msg2
    (data msg1 = data1 |@ "source")
    .&.
    (data msg2 <> data2 |@ "target")
```





FsCheck: Generation

distribution of random data using a custom generator

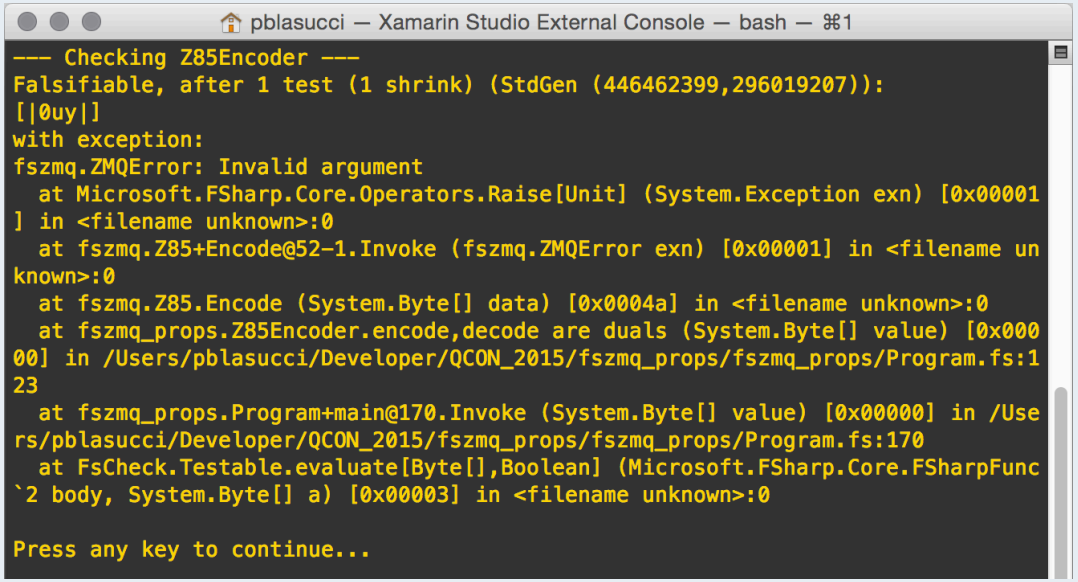
FsCheck: Generation

```
type Generators =
  static member Version =
    let unknown = Gen.constant Unknown
    let version = Arb.generate<NonNegativeInt>
                  |> Gen.three
                  |> Gen.map (fun (NonNegativeInt m
                                  ,NonNegativeInt n
                                  ,NonNegativeInt r) -> fszmq.Version (m,n,r))
    [ (1,unknown); (2,version) ]
    |> Gen.frequency
    |> Arb.fromGen
```

FsCheck: Generation

Expressing implicit business rules

```
module Z85Encoder =
  let ``encode,decode are duals`` (data:byte[]) =
    data|> Z85.encode |> Z85.decode = data
```



FsCheck: Generation

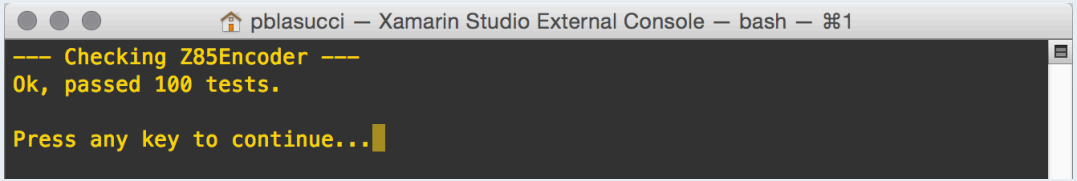
Expressing implicit business rules

"Z85 … takes a binary frame and encodes it as a printable ASCII string, or takes an ASCII encoded string and decodes it into a binary frame.

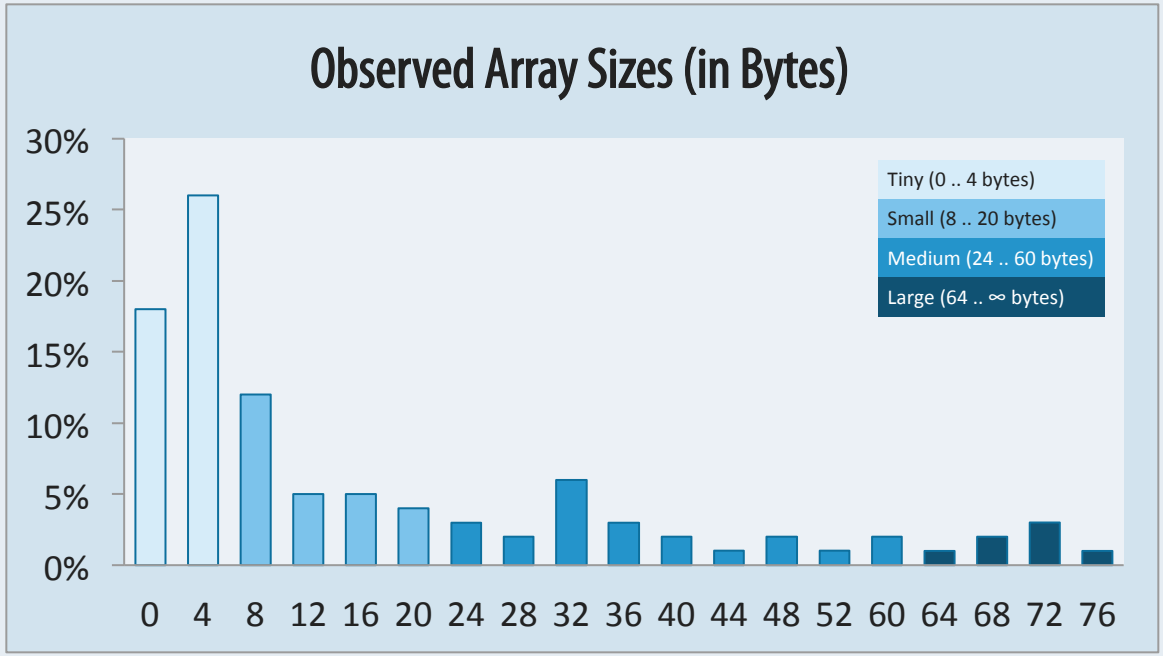
*The **binary frame** SHALL have a length that is **divisible by 4 with no remainder**. The **string frame** SHALL have a length that is **divisible by 5 with no remainder**."*

from Z85 RFC

```
module Z85Encoder =
  let ``encode,decode are duals`` (Mod4Bytes data) =
    data|> Z85.encode |> Z85.decode = data
```




```
type Mod4Bytes = Mod4Bytes of byte[]

// ... elsewhere ...

static member Mod4Bytes =
  let g = Arb.generate<byte[]>
          |> Gen.suchThat (fun b -> b <> null
                                    && b.Length % 4 = 0)
          |> Gen.map Mod4Bytes
  let s (Mod4Bytes bytes) = bytes
                            |> Arb.shrink
                            |> Seq.map Mod4Bytes
  Arb.fromGenShrink (g,s)
```

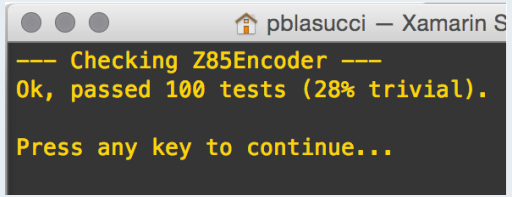


FsCheck: Observations

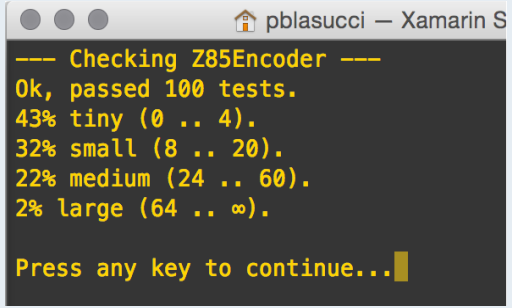
distribution of randomly generated test inputs

FsCheck: Observations

```
let ``encode,decode are duals`` (Mod4Binary value) =
  (value |> Z85.encode |> Z85.decode = value)
  |> Prop.trivial (Array.isEmpty value)
```



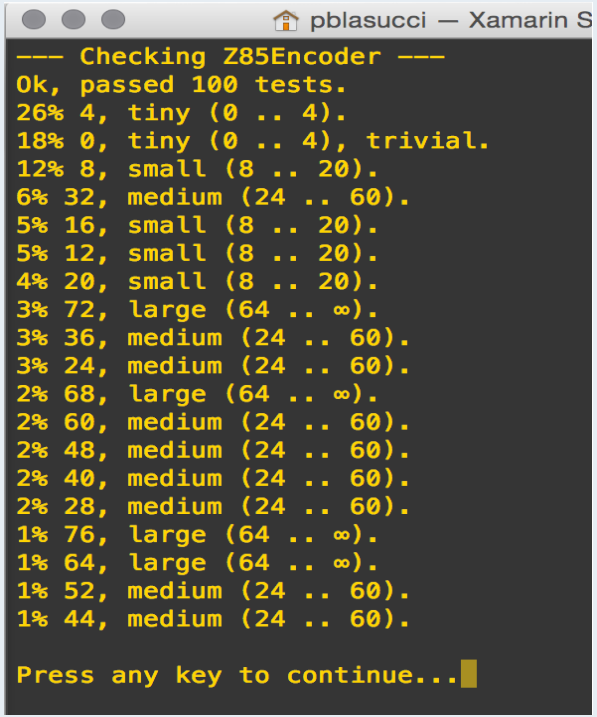

```
let ``encode,decode are duals`` (Mod4Binary value) =
  (value |> Z85.encode |> Z85.decode = value)
  |> Prop.classify (large  value) "large (64 .. ∞)”
  |> Prop.classify (medium value) "medium (24 .. 60)”
  |> Prop.classify (small  value) "small (8 .. 20)”
  |> Prop.classify (tiny   value) "tiny (0 .. 4)"
```



FsCheck: Observations

```
let ``encode,decode are duals`` (Mod4Binary value) =
  (value |> Z85.encode |> Z85.decode = value)
  |> Prop.collect (Array.length value)
```

```
let ``encode,decode are duals`` (Mod4Binary value) =
  (value |> Z85.encode |> Z85.decode = value)
  |> Prop.trivial   (Array.isEmpty value)
  |> Prop.classify  (large  value) “large (64 .. ∞)”
  |> Prop.classify  (medium value) “medium (24 .. 60)”
  |> Prop.classify  (small  value) “small (8 .. 20)”
  |> Prop.classify  (tiny   value) “tiny (0 .. 4)”
  |> Prop.collect   (Array.length value)
```



Random Testing

"One of the major advantages… is that it **encourages** us to formulate **formal specifications**, **thus improving** our **understanding**…"

from ICFP'00 – Claessen, Hughes

More Information

about F# and FsCheck

- fsharp.org
- fscheck.github.io/fscheck
- fsharpforfunandprofit.com
- meetup.com/nyc-fsharp

about Paulmichael Blasucci

- twitter.com/pblasucci
- github.com/pblasucci
- linkedin.com/in/pblasucci
- pblasucci.wordpress.com