Breaking your code in new and exciting ways

Michael Newton (@mavnn)

Who am I?

- 15below – telling you your flight's been delayed for 15 years and counting
- Keeping up with the latest hammers

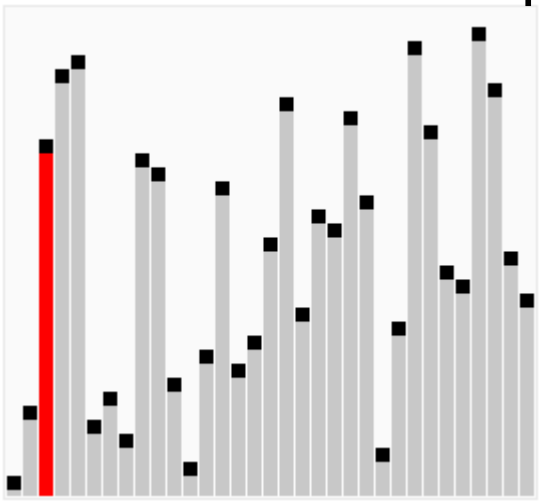
- Open source (personal and 15below)

15below

Property based testing?

Property based testing

- Define a property (“sorting a list should return a list of the same length”)
- Generate input values to try and break the property
- Shrink inputs to a minimal reproduction



15below

A bit of history

QuickCheck

- Not new: QuickCheck released 1999 (Claesson and Hughes)
- QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs – ICFP 2000
- At least 25 other implementations as of 2015
- .net version: FsCheck

15below

FsCheck

- Mature, high quality code (first commit in 2008)
- Code to test and property code in any .net language
- Generators easier to write in F# (although possible in C#)
- Maintained by Kurt Schelfthout

15below

A brief interlude

Some of our examples will not look like the others…

F# in 5 minutes or less…

```
using System;

namespace BYCINAEW.CSharp
{
    public class Class1
    {
        static public int Adder(int x, int y)
        {
            return x + y;
        }

        static public void CallAdder()
        {
            var first = 10;
            var second = 20;

            var result = Adder(first, second);

            Console.WriteLine(result);
        }
    }
}
```

```
module BYCINAEW.FSharp.Module1

let adder x y =
    x + y

let first = 10
let second = 20

let result =
    adder 10 20

printfn "%d" result
```

15below

F# in 5 minutes or less…

```
using System;
using System.Net;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

namespace BYCINAEW.CSharp
{
    public class Downloader
    {
        static public async Task<int> Count(string url, string word)
        {
            using (var client = new WebClient())
            {
                var uri = new Uri(url);

                var html = await client.DownloadStringTaskAsync(uri);

                return Regex.Matches(html, word).Count;
            }
        }
    }
}
```

15below

F# in 5 minutes or less…

```
module BYCINAEW.FSharp.Module1

open System
open System.Net
open System.Text.RegularExpressions

let count url word =
    use client = new WebClient()
    let uri = Uri(url)
    async {
        let! html =
            client.AsyncDownloadString uri
        return Regex.Matches(html, word).Count
    } // returns Async<int>
```

15below

Enterprise All the Things

Sometimes it just has to be XML

Update collector

- Receives a series of enhancement commands
- Receives a commit command
- Saves an XML file with all enhancements

15below

Update collector

Properties!

- Idempotent: Submitting the same command multiple times should have no effect
- Append only: a command should never reduce the size of the document
- Safe: input is not under our control, may be untrustworthy

15below

Enterprise All the Things

The code!

Serialization

Exploiting common properties

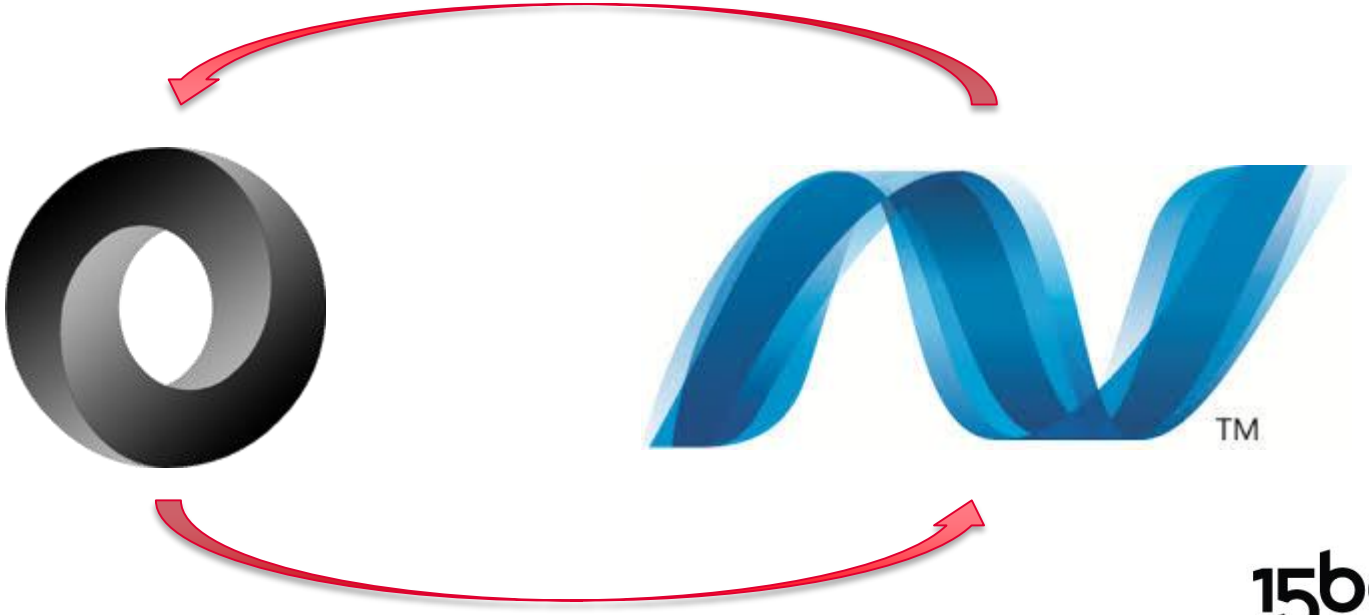
Choosing Properties – what to look for

- Functions that are inverses
  - Serialize/Deserialize
- Things that don't change
  - Running the same command repeatedly
- Partial functions
  - Where the input type (we're looking at you, 'string') is broader than the meaningful input space: check nonsense input

15below

Chiron – Type safe Json Serialization

- Not reflection based
- “Safe”
- Blindingly simple property: can you roundtrip from Json and back?



15below

Serialization

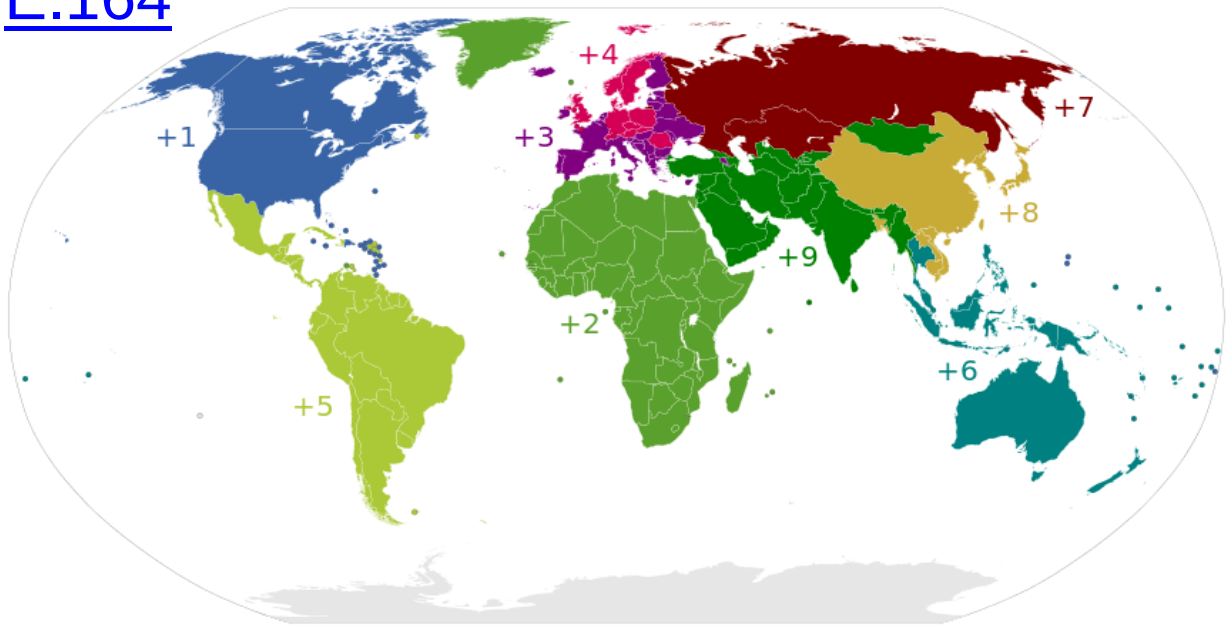
The code!

Phone Numbers

When the business is kind enough to tell you the properties

E.164 International Telephone Numbers

- Users supply things like “07788 123 987”
- SMS Senders like “+447788123987”
- E.164



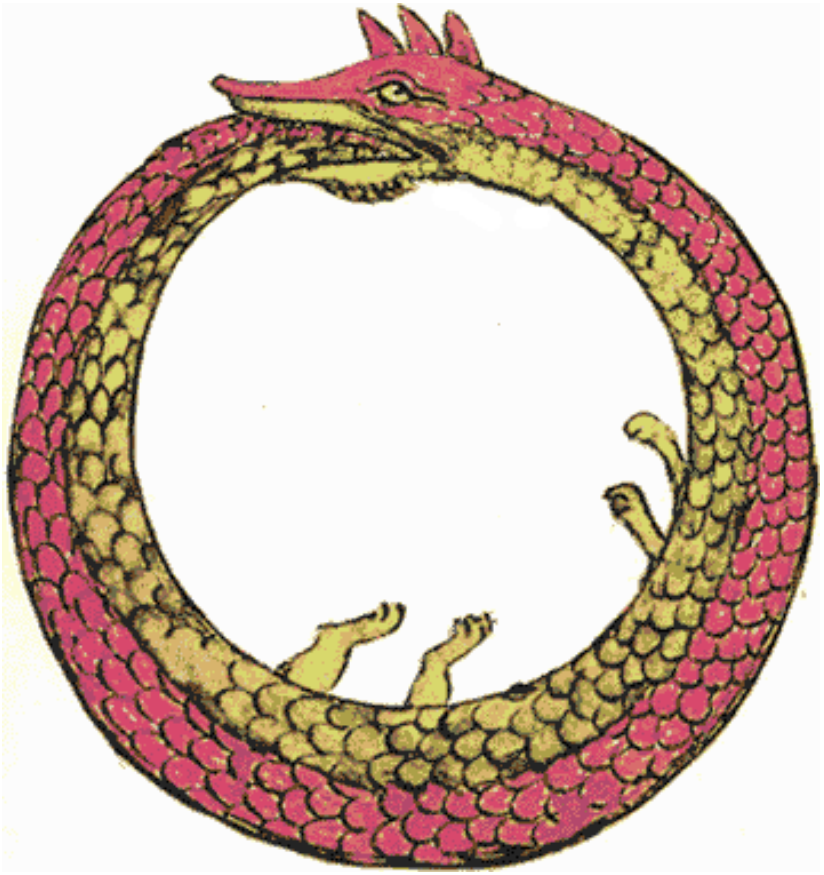
15below

E.164 Properties

- We want a “ValidE164” class that can **only** contain valid international numbers
- 1st property: ValidE164.TryCreate(validNumber) creates an instance
- 2nd property: ValidE164.TryCreate(invalidNumber) doesn’t

15below

E.164 Properties



15below

E.164 Generators

- Three separate generators
  1. Default string generator (this is a partial function)
  2. Correct number generator
  3. Incorrect number generator

Phone Numbers

The code!

Distributed Locking

Only if we have time…

Sproc.Lock

- Distributed locks
- Environment with no locking server deployed
- Implemented on top of SQL Server

- QA department reported no bugs on testing (1st time for everything)

15below

Distributed Locking

The code!

Wrapping it all up

Hooking into your test runner

- Xunit and Nunit plugs available (check FsCheck docs)
- Or: just call “QuickCheckThrowOnFailure” in a test for your framework

15below

Things to watch out for

- Keep ‘em fast
- Learn to create Generator
- Use labels!
- Model based testing
  - Refactoring
  - Performance tuning
  - Distributed system testing

15below

When (not) to use property based testing

- The bad
  - Creating properties can be time consuming – make sure it's worthwhile
  - Properties only make sense if you have a known specification (formal or informal)
  - Run times start adding up – be sparing
- The good
  - Get rid of 100's of "example" unit tests
  - Test things you'd never thought of (invalid XML characters, anyone?)
  - Build a **lot** of confidence in code with a accurate specification

15below

Tell me more

- FsCheck specific:
  - My blog – Intro and practical tutorial
  - Scott Wlaschin – An introduction to property-based testing
- General QuickCheck resources:
  - John Hughes – QuickCheck Evolution (video)

15below

Any questions?

Answers on a postcard, please

15below