

Matthew Butt

# Unit Testing the Hard Stuff

matthewbutt.com    @bnathyuw

NB

There will be code

It will be C#

I will explain it

Axiom:

# Unit Tests are Important

feedback for delivering quality software

Speed

Control

Precision

Observation:

**Some projects have  
no Unit Tests**

Why?

# Why?

not Designed for Test

few Techniques

few Tools

not even many Workarounds

# Unit Testing the *Easy* Stuff



Designed for Test

Tools

Techniques

or at least **Workarounds**

.NET Web API

[Whiteboard]

Designed for Test

**Directly Executable**

**Object Oriented**

**Abstractions**

**DI**

Techniques

**SOLID design**

**Mocking**

**Architecture Patterns**

(MVC, ports & adapters, hexagonal...)

Tools

**NUnit...**

test framework

**NSubstitute...**

mocking framework

**OWIN...**

in-memory host

Workarounds

# Adapters

around leaky abstractions

# Mock Clock

and other system dependencies

Speed **Run locally**

Control **Test doubles**

Precision **Class level**

**Clear purpose**



Outside-in approach

**Acceptance test** outer loop

**Unit test** inner loop

Acceptance Test

# In-Memory Host Stub Externals

Controller Test

# In-Memory Host

Treat as adapter

# Mock Dependencies

Test interactions

# Keep it thin

No domain logic

Domain Object Tests

# State or Interaction?

State: stubs

Interaction: mocks

# Single Responsibility

Listen to your tests!

External dependencies

# Abstractions

Interfaces in Domain

## Integration tests

## No test if trivial

Clock

```
namespace EasyStuff.Api.Domain {  
    public interface IClock {  
        DateTime Now { get; }  
    }  
}
```

```
namespace EasyStuff.Api.Adapters {  
    public class SystemClock : IClock {  
        public DateTime Now => DateTime.UtcNow;  
    }  
}
```

```
var knownDate = new DateTime(2001, 2, 3);  
var clock = Substitute.For<IClock>();  
clock.Now.Returns(knownDate);
```

# Unit Testing the **Hard** Stuff

# Microsoft Azure Data Lake Analytics



~~Financial Transactions~~

Weather Data

## aberporthdata.txt

### Aberporth

Location: 224100E 252100N, Lat 52.139 Lon -4.570, 133 metres amsl

Estimated data is marked with a \* after the value.

Missing data (more than 2 days missing in month) is marked by ---.

Sunshine data taken from an automatic Kipp & Zonen sensor marked with a #, otherwise sunshine data taken from a Campbell Stokes recorder.

yyyy	mm	tmax degC	tmin degC	af days	rain mm	sun hours
1970	1	7.5	3.1	7	97.5	40.2
1970	2	6.2	1.9	7	79.2	96.2
1970	3	6.6	2.0	8	76.1	101.2
1970	4	8.8	4.2	2	67.0	135.2
1970	5	14.5	8.5	0	28.1	148.9
1970	6	18.5	11.5	0	47.3	206.5
1970	7	16.6	11.3	0	57.4	150.3
1970	8	17.8	12.3	0	42.6	151.8
1970	9	16.8	11.5	0	49.4	120.4
1970	10	13.3	8.6	0	108.3	75.4
1970	11	10.8	6.4	0	181.4	41.7
1970	12	7.5	3.3	6	42.9	69.4
etc.			etc.			etc.

Extract

Transform

Load

[Whiteboard]

Not Designed for Test

Hosted in Cloud

Not Directly Executable

Hybrid Code

Closely Coupled

Few Tools

A library from MS

abstractions too leaky

Local Test Runner

this *is* useful

Few techniques

# SQL unit tests

fairly gruesome

Few workarounds

**Google doesn't help**



Outside-in approach

**Acceptance test** outer loop

**Unit test** inner loop

Acceptance Test

# Local Run Helper

```
}namespace WeatherData.III.AcceptanceTests
{
    ...[TestFixture]
}...public class MonthlyMaximumShould
    ...{
    ...private static readonly char[] LineSeparators = {'\r', '\n'};

    ...[Test]
}...public void ShowMaximumTemperatureForEachMonthOfTheYear()
    ...{
    ...CopyToDataRoot("input\\metOfficeObservations\\aberporthdata.txt");

    ...Run(AnalyticsScript("monthlyMaximum.usql"));

    ...var output = ReadOutput("monthlyMaximum.csv");
    ...var lines = output.Split(LineSeparators, RemoveEmptyEntries);
    ...lines.Length.Should().Be(12);
    ...}
    ...}
}
}
```

```
.....public static void Run(string script)
.....{
.....    var localRunHelper = new LocalRunHelper
.....    {
.....        ScriptPath = script,
.....        DataRoot = DataRoot
.....    };

.....    localRunHelper.DoRun().Should().BeTrue("script should execute successfully");
.....}
```

# U-SQL Script Test

## Hybrid code

U-SQL & C#

## Data on file system

```
USE DATABASE [WeatherData];

REFERENCE ASSEMBLY [Objects];

USING Extractors := WeatherData.[III].Objects.Extractors;
USING Outputters := WeatherData.[III].Objects.Outputters;

DECLARE @inputFiles := "input\metOfficeObservations\{location}data.txt";
DECLARE @outputFile := "output\monthlyMaximum.csv";

@observations :=
... EXTRACT location string,
... year int,
... month int,
... maximumTemperature double?
... FROM @inputFiles
... USING Extractors.MetOfficeObservations;

@maxima :=
... SELECT month,
... MAX(maximumTemperature) AS maximumTemperature
... FROM @observations
... GROUP BY month;

@output :=
... SELECT ANY_VALUE(@observations.location) AS location,
... ANY_VALUE(@observations.year) ?? 0 AS year,
... @observations.month,
... @observations.maximumTemperature
```

Responsibility of Script

**Orchestrator** or  
**Query**

Seam U-SQL // C#

**Inline code**

Nope!

**Code-behind**

Nah...

**Assemblies**

Now you're talking!



# Substitute Assemblies

Code is compiled for each execution

Duck typing

Substitute at runtime

Solution Explorer



Search Solution Explorer (Ctrl+;)



Solution 'WeatherData.III' (7 projects)

- ▶ **WeatherData.III.AcceptanceTests**
- ▶ WeatherData.III.Analytics
- ▶ WeatherData.III.Analytics.Tests
- ▶ WeatherData.III.Analytics.Tests.DD
- ▶ WeatherData.III.Analytics.Tests.Objects
- ▶ WeatherData.III.DataDefinition
- ▶ WeatherData.III.Objects

```
USE DATABASE [WeatherData];
```

```
DROP ASSEMBLY IF EXISTS [Objects];
```

```
CREATE ASSEMBLY [Objects]
```

```
FROM @"WeatherData.III.Analytics.Tests.Objects.dll";
```

```
.....[OneTimeSetUp]
|.....public void OneTimeSetUp()
|.....{
|.....    CreateDirectory(DataRoot);
|.....    Run(DataDefinitionScript("CreateDatabase.usql"));
|.....
|.....    CopyToDataRoot("WeatherData.III.Analytics.Tests.Objects.dll");
|.....    Run(DataDefinitionScript("RegisterObjectsAssembly.usql"));
|.....}
|.....}
```

# Stub or Mock?

Not directly available

Fakes against file system

```
namespace WeatherData.III.Objects
{
    ... public class MetOfficeObservationExtractor : IExtractor
    ... {
    ...     ... public override IEnumerable<IRow> Extract(IUnstructuredReader input, IUpdatableRow output)
    ...     ... {
    ...     ...     ... foreach (var lineStream in input.Split(Encoding.UTF8.GetBytes("\r\n")))
    ...     ...     ... {
    ...     ...     ...     ... var serializer = new DataContractJsonSerializer(typeof(MaximumTemperatureInput));
    ...     ...     ...     ... var inputObject = (MaximumTemperatureInput)serializer.ReadObject(lineStream);
    ...     ...     ...     ... yield return inputObject.WriteTo(output);
    ...     ...     ...     ... }
    ...     ...     ... }
    ...     ... }
    ... }
}
```

---

---

```
namespace WeatherData.III.Objects
```

```
{  
    ...public class MetOfficeObservationOutputter : IOutputter  
    ...{  
        ...public override void Output(IRow input, IUnstructuredWriter output)  
        ...{  
            ...using (var streamWriter = new StreamWriter(output.BaseStream))  
            ...{  
                ...streamWriter.WriteLine(SerializeObject(MaximumTemperatureOutput.ReadFrom(input)));  
            ...}  
        ...}  
    ...}  
}
```

# User-Defined Objects



```

public class MetOfficeObservationExtractor : IExtractor
{
    public override IEnumerable<IRow> Extract(IUnstructuredReader input, IUpdatableRow output)
    {
        using (var streamReader = new StreamReader(input.BaseStream))
        {
            string line;
            while ((line = streamReader.ReadLine()) != ".degC .degC .days .mm .hours")
            {
            }
            while (!string.IsNullOrEmpty(line = streamReader.ReadLine()))
            {
                var parts = line.Split(new[] { '.' }, StringSplitOptions.RemoveEmptyEntries);
                output.Set("year", int.Parse(parts[0]));
                output.Set("month", int.Parse(parts[1]));
                output.Set("maximumTemperature", ParseNullableDouble(parts[2]));
                yield return output.AsReadOnly();
            }
        }
    }

    private static double? ParseNullableDouble(string part)
    {
        return part == "---" ? (double?) null : double.Parse(part);
    }
}

```

Leaky Abstraction

Use of Streams

Temporal Coupling

Strange Idiom

>1 Responsibility

```

4
5 [TestFixture]
6 public class MetOfficeObservationExtractorShould
7 {
8     private MetOfficeObservationExtractor _metOfficeObservationExtractor;
9     private IUnstructuredReader _input;
10    private IUpdatableRow _output;
11
12    [SetUp]
13    public void Setup()
14    {
15        _metOfficeObservationExtractor = new MetOfficeObservationExtractor();
16        _input = Substitute.For<IUnstructuredReader>();
17
18        var memoryStream = new MemoryStream(Encoding.UTF8.GetBytes(text));
19        _input.BaseStream.Returns(memoryStream);
20
21        _output = Substitute.For<IUpdatableRow>();
22    }
23
24    [Test]
25    public void InteractWithOutput()
26    {
27        _metOfficeObservationExtractor.Extract(_input, _output).ToList();
28
29        Received.InOrder(() =>
30        {
31            CallsToReturnValue(_output, 1958, 3, 7.6);
32            CallsToReturnValue(_output, 1958, 4, 10.6);
33            CallsToReturnValue(_output, 1958, 5, 13.4);
34            CallsToReturnValue(_output, 1941, 12, null);
35        });
36    }
37
38    [Test]
39    public void ReturnRowsFromOutput()
40    {
41        var row1 = Substitute.For<IRow>();
42        var row2 = Substitute.For<IRow>();
43        var row3 = Substitute.For<IRow>();
44        var row4 = Substitute.For<IRow>();
45        _output.AsReadonly().Returns(row1, row2, row3, row4);
46
47        var actualRows = _metOfficeObservationExtractor.Extract(_input, _output).ToArray();
48
49        actualRows.Should().BeEquivalentTo(row1, row2, row3, row4);
50    }
51
52    private static void CallsToReturnValue(IUpdatableRow output, int year, int month, double? maximumTemperature)
53    {
54        output.Set("year", year);
55        output.Set("month", month);
56        output.Set("maximumTemperature", maximumTemperature);
57        output.AsReadonly();
58    }
59
60    private const string Text = @"Aberporth
61Location: 224100E 252100N, Lat 52.139 lon -4.570, 133 metres amsl
62Estimated data is marked with a * after the value.
63Missing data (more than 2 days missing in month) is marked by ----.
64Sunshine data taken from an automatic Kipp & Zonen sensor marked with a #, otherwise sunshine data taken from a Campbell Stokes recorder.
65--- yyyy-mm---tmax---tmin---af---rain---sun
66--- degC---degC---days---mm---hours
671958-3---7.6---1.7---8---21.1---128.8
681958-4---10.6---4.6---0---0---17.8---169.0
691958-5---13.4---7.8---0---0---95.3---190.8
701941-12-----86.5-----
71";
72
73    }
74
75    }
76
77    }
78
79    }
80
81    }
82
83    }
84
85    }
86
87    }
88
89    }
90
91    }
92
93    }
94
95    }
96
97    }
98
99    }
100
101    }
102
103    }
104
105    }
106
107    }
108
109    }
110
111    }
112
113    }
114
115    }
116
117    }
118
119    }
120
121    }
122
123    }
124
125    }
126
127    }
128
129    }
130
131    }
132
133    }
134
135    }
136
137    }
138
139    }
140
141    }
142
143    }
144
145    }
146
147    }
148
149    }
150
151    }
152
153    }
154
155    }
156
157    }
158
159    }
160
161    }
162
163    }
164
165    }
166
167    }
168
169    }
170
171    }
172
173    }
174
175    }
176
177    }
178
179    }
180
181    }
182
183    }
184
185    }
186
187    }
188
189    }
190
191    }
192
193    }
194
195    }
196
197    }
198
199    }
200
201    }
202
203    }
204
205    }
206
207    }
208
209    }
210
211    }
212
213    }
214
215    }
216
217    }
218
219    }
220
221    }
222
223    }
224
225    }
226
227    }
228
229    }
230
231    }
232
233    }
234
235    }
236
237    }
238
239    }
240
241    }
242
243    }
244
245    }
246
247    }
248
249    }
250
251    }
252
253    }
254
255    }
256
257    }
258
259    }
260
261    }
262
263    }
264
265    }
266
267    }
268
269    }
270
271    }
272
273    }
274
275    }
276
277    }
278
279    }
280
281    }
282
283    }
284
285    }
286
287    }
288
289    }
290
291    }
292
293    }
294
295    }
296
297    }
298
299    }
300
301    }
302
303    }
304
305    }
306
307    }
308
309    }
310
311    }
312
313    }
314
315    }
316
317    }
318
319    }
320
321    }
322
323    }
324
325    }
326
327    }
328
329    }
330
331    }
332
333    }
334
335    }
336
337    }
338
339    }
340
341    }
342
343    }
344
345    }
346
347    }
348
349    }
350
351    }
352
353    }
354
355    }
356
357    }
358
359    }
360
361    }
362
363    }
364
365    }
366
367    }
368
369    }
370
371    }
372
373    }
374
375    }
376
377    }
378
379    }
380
381    }
382
383    }
384
385    }
386
387    }
388
389    }
390
391    }
392
393    }
394
395    }
396
397    }
398
399    }
400
401    }
402
403    }
404
405    }
406
407    }
408
409    }
410
411    }
412
413    }
414
415    }
416
417    }
418
419    }
420
421    }
422
423    }
424
425    }
426
427    }
428
429    }
430
431    }
432
433    }
434
435    }
436
437    }
438
439    }
440
441    }
442
443    }
444
445    }
446
447    }
448
449    }
450
451    }
452
453    }
454
455    }
456
457    }
458
459    }
460
461    }
462
463    }
464
465    }
466
467    }
468
469    }
470
471    }
472
473    }
474
475    }
476
477    }
478
479    }
480
481    }
482
483    }
484
485    }
486
487    }
488
489    }
490
491    }
492
493    }
494
495    }
496
497    }
498
499    }
500
501    }
502
503    }
504
505    }
506
507    }
508
509    }
510
511    }
512
513    }
514
515    }
516
517    }
518
519    }
520
521    }
522
523    }
524
525    }
526
527    }
528
529    }
530
531    }
532
533    }
534
535    }
536
537    }
538
539    }
540
541    }
542
543    }
544
545    }
546
547    }
548
549    }
550
551    }
552
553    }
554
555    }
556
557    }
558
559    }
560
561    }
562
563    }
564
565    }
566
567    }
568
569    }
570
571    }
572
573    }
574
575    }
576
577    }
578
579    }
580
581    }
582
583    }
584
585    }
586
587    }
588
589    }
590
591    }
592
593    }
594
595    }
596
597    }
598
599    }
600
601    }
602
603    }
604
605    }
606
607    }
608
609    }
610
611    }
612
613    }
614
615    }
616
617    }
618
619    }
620
621    }
622
623    }
624
625    }
626
627    }
628
629    }
630
631    }
632
633    }
634
635    }
636
637    }
638
639    }
640
641    }
642
643    }
644
645    }
646
647    }
648
649    }
650
651    }
652
653    }
654
655    }
656
657    }
658
659    }
660
661    }
662
663    }
664
665    }
666
667    }
668
669    }
670
671    }
672
673    }
674
675    }
676
677    }
678
679    }
680
681    }
682
683    }
684
685    }
686
687    }
688
689    }
690
691    }
692
693    }
694
695    }
696
697    }
698
699    }
700
701    }
702
703    }
704
705    }
706
707    }
708
709    }
710
711    }
712
713    }
714
715    }
716
717    }
718
719    }
720
721    }
722
723    }
724
725    }
726
727    }
728
729    }
730
731    }
732
733    }
734
735    }
736
737    }
738
739    }
740
741    }
742
743    }
744
745    }
746
747    }
748
749    }
750
751    }
752
753    }
754
755    }
756
757    }
758
759    }
760
761    }
762
763    }
764
765    }
766
767    }
768
769    }
770
771    }
772
773    }
774
775    }
776
777    }
778
779    }
780
781    }
782
783    }
784
785    }
786
787    }
788
789    }
790
791    }
792
793    }
794
795    }
796
797    }
798
799    }
800
801    }
802
803    }
804
805    }
806
807    }
808
809    }
810
811    }
812
813    }
814
815    }
816
817    }
818
819    }
820
821    }
822
823    }
824
825    }
826
827    }
828
829    }
830
831    }
832
833    }
834
835    }
836
837    }
838
839    }
840
841    }
842
843    }
844
845    }
846
847    }
848
849    }
850
851    }
852
853    }
854
855    }
856
857    }
858
859    }
860
861    }
862
863    }
864
865    }
866
867    }
868
869    }
870
871    }
872
873    }
874
875    }
876
877    }
878
879    }
880
881    }
882
883    }
884
885    }
886
887    }
888
889    }
890
891    }
892
893    }
894
895    }
896
897    }
898
899    }
900
901    }
902
903    }
904
905    }
906
907    }
908
909    }
910
911    }
912
913    }
914
915    }
916
917    }
918
919    }
920
921    }
922
923    }
924
925    }
926
927    }
928
929    }
930
931    }
932
933    }
934
935    }
936
937    }
938
939    }
940
941    }
942
943    }
944
945    }
946
947    }
948
949    }
950
951    }
952
953    }
954
955    }
956
957    }
958
959    }
960
961    }
962
963    }
964
965    }
966
967    }
968
969    }
970
971    }
972
973    }
974
975    }
976
977    }
978
979    }
980
981    }
982
983    }
984
985    }
986
987    }
988
989    }
990
991    }
992
993    }
994
995    }
996
997    }
998
999    }
1000

```

# Adapter layer

Translate between framework & domain

```
·internal·class·MetOfficeObservationExtractor···IExtractor
·{
····private·readonly·MetOfficeDatasetParser·_metOfficeDatasetParser;
····private·readonly·InputReader·_inputReader;
····private·readonly·RowFactory·_rowFactory;

····internal·MetOfficeObservationExtractor(InputReader·inputReader,
········MetOfficeDatasetParser·metOfficeDatasetParser,·RowFactory·rowFactory)
····{
········_metOfficeDatasetParser·=·metOfficeDatasetParser;
········_inputReader·=·inputReader;
········_rowFactory·=·rowFactory;
····}

····public·override·IEnumerable<IRow>·Extract(IUnstructuredReader·input,·IUpdatableRow·output)
····{
········var·inputLines·=·_inputReader·.ReadLines(input);
········var·metOfficeObservations·=·_metOfficeDatasetParser·.Parse(inputLines);
········foreach·(var·metOfficeObservation·in·metOfficeObservations)
········{
··········yield·return·_rowFactory·.Create(output,·metOfficeObservation);
········}
····}
·}
```

---

```

[TestFixture]
public class MetOfficeObservationExtractorShould
{
    ... private MetOfficeDatasetParser _metOfficeDatasetParser;
    ... private MetOfficeObservationExtractor _metOfficeObservationExtractor;
    ... private IUpdateableRow _output;

    ... private readonly IEnumerable<string> _lines = new string[] { };
    ... private readonly IRow _row1 = Substitute.For<IRow>();
    ... private readonly IRow _row2 = Substitute.For<IRow>();
    ... private readonly IRow _row3 = Substitute.For<IRow>();
    ... private readonly MetOfficeObservation _observation1 = new MetOfficeObservation();
    ... private readonly MetOfficeObservation _observation2 = new MetOfficeObservation();
    ... private readonly MetOfficeObservation _observation3 = new MetOfficeObservation();

    ... private InputReader _inputReader;
    ... private IUnstructuredReader _input;
    ... private RowFactory _rowFactory;

    ... [SetUp]
    ... public void SetUp()
    ... {
    ...     _input = Substitute.For<IUnstructuredReader>();
    ...     _output = Substitute.For<IUpdateableRow>();

    ...     _inputReader = Substitute.For<InputReader>();
    ...     _metOfficeDatasetParser = Substitute.For<MetOfficeDatasetParser>((MetOfficeObservationParser) null);
    ...     _rowFactory = Substitute.For<RowFactory>();

    ...     _metOfficeObservationExtractor = new MetOfficeObservationExtractor(_inputReader, _metOfficeDatasetParser, _rowFactory);
    ... }

    ... [Test]
    ... public void ReadParseWriteAndReturnValues()
    ... {
    ...     _inputReader.ReadLines(_input).Returns(_lines);
    ...     _metOfficeDatasetParser.Parse(_lines).Returns(new[] { _observation1, _observation2, _observation3 });
    ...     _rowFactory.Create(_output, _observation1).Returns(_row1);
    ...     _rowFactory.Create(_output, _observation2).Returns(_row2);
    ...     _rowFactory.Create(_output, _observation3).Returns(_row3);

    ...     var actualRows = _metOfficeObservationExtractor.Extract(_input, _output).ToList();

    ...     actualRows.Should().BeEquivalentTo(_row1, _row2, _row3);
    ... }
}

```

```
internal class InputReader
{
    public virtual IEnumerable<string> ReadLines(IUnstructuredReader input)
    {
        using (var streamReader = new StreamReader(input.BaseStream))
        {
            while (!streamReader.EndOfStream)
            {
                yield return streamReader.ReadLine();
            }
        }
    }
}
```

---

```
[TestFixture]
public class InputReaderShould
{
    private readonly IEnumerable<string> _lines = new[] { "Line1", "Line2", "Line3" };
    private InputReader _inputReader;
    private IUnstructuredReader _unstructuredReader;

    [SetUp]
    public void SetUp()
    {
        _unstructuredReader = Substitute.For<IUnstructuredReader>();

        _inputReader = new InputReader();
    }

    [Test]
    public void ReturnEachLineFromInputStream()
    {
        _unstructuredReader.BaseStream.Returns(StreamWithLines(_lines));

        var linesReturned = _inputReader.ReadLines(_unstructuredReader).ToList();

        linesReturned.Should().BeEquivalentTo(_lines);
    }

    private static MemoryStream StreamWithLines(IEnumerable<string> lines) => new MemoryStream(Encoding.UTF8.GetBytes(string.Join("\r\n", lines)));
}
```



```
·internal·class·RowFactory
·{
····public·virtual·IRow·Create(IUpdatableRow·output,·MetOfficeObservation·metOfficeObservation)
····{
·······output.Set("year",·metOfficeObservation.Year);
·······output.Set("month",·metOfficeObservation.Month);
·······output.Set("maximumTemperature",·metOfficeObservation.MaximumTemperature);
·······return·output.AsReadOnly();
····}
·}
```

---

```

[TestFixture]
public class RowFactoryShould
{
    private RowFactory _rowFactory;
    private IUpdatableRow _output;
    private readonly IRow _expectedRow = Substitute.For<IRow>();

    [SetUp]
    public void Setup()
    {
        _rowFactory = new RowFactory();
        _output = Substitute.For<IUpdatableRow>();
    }

    [Test]
    public void CollaborateWithOutputToCreateOutputRow(
        [Values(1999, 2001)] int year,
        [Values(1, 12)] int month,
        [Values(null, -12.3, 12.3)] double? maximumTemperature)
    {
        var metOfficeObservation = new MetOfficeObservation { Year = year, Month = month, MaximumTemperature = maximumTemperature };
        _rowFactory.Create(_output, metOfficeObservation);

        Received.InOrder(() =>
        {
            _output.Set("year", year);
            _output.Set("month", month);
            _output.Set("maximumTemperature", maximumTemperature);
            _output.AsReadOnly();
        });
    }

    [Test]
    public void ReturnCreatedRow()
    {
        _output.AsReadOnly().Returns(_expectedRow);

        var row = _rowFactory.Create(_output, new MetOfficeObservation());

        row.Should().Be(_expectedRow);
    }
}

```

Domain Tests

# Familiar Territory

```
internal class MetOfficeObservationParser
{
    private const string NoObservation = "---";

    public virtual MetOfficeObservation Parse(string line) =>
        ObservationFromParts(PartsSeparatedBySpaces(line));

    private static string[] PartsSeparatedBySpaces(string line) => line.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

    private static MetOfficeObservation ObservationFromParts(string[] parts) => new MetOfficeObservation
    {
        Year = ReadYear(parts),
        Month = ReadMonth(parts),
        MaximumTemperature = ReadMaximumTemperature(parts)
    };

    private static int ReadYear(string[] parts) => int.Parse(parts[0]);

    private static int ReadMonth(string[] parts) => int.Parse(parts[1]);

    private static double? ReadMaximumTemperature(string[] parts) => ParseNullableDouble(parts[2]);

    private static double? ParseNullableDouble(string part) => part == NoObservation ? (double?) null : double.Parse(part);
}
```

```
[TestFixture]
public class MetOfficeObservationParserShould
{
    private MetOfficeObservationParser _metOfficeObservationParser;

    [SetUp]
    public void SetUp()
    {
        _metOfficeObservationParser = new MetOfficeObservationParser();
    }

    [Test]
    public void ParseYearFromFirstElement()
    {
        var metOfficeObservation = _metOfficeObservationParser.Parse("1979-2-3");

        metOfficeObservation.Year.Should().Be(1979);
    }

    [Test]
    public void ParseMonthFromSecondElement()
    {
        var metOfficeObservation = _metOfficeObservationParser.Parse("1979-11-3");

        metOfficeObservation.Month.Should().Be(11);
    }

    [Test]
    public void ParseMaximumTemperatureFromThirdElement()
    {
        var metOfficeObservation = _metOfficeObservationParser.Parse("1979-11-12.3");

        metOfficeObservation.MaximumTemperature.Should().Be(12.3);
    }

    [Test]
    public void ParseNullMaximumTemperatureFromWhenNoObservationSupplied()
    {
        var metOfficeObservation = _metOfficeObservationParser.Parse("1979-11-");

        metOfficeObservation.MaximumTemperature.Should().Be(null);
    }
}
```

You may still need to  
Mock the Clock!

What have we learnt?

Speed **Run locally**

Control **Seams**

**Test doubles**

Precision **Small pieces**

**Clear purpose**

**Adapters**



Patterns

Patterns

Patterns

# Resources

Testing Patterns

[github.com/bnathyuw/testing-patterns/wiki](https://github.com/bnathyuw/testing-patterns/wiki)

Example implementation

[github.com/bnathyuw/weather-data-iii](https://github.com/bnathyuw/weather-data-iii)

Dank je wel!