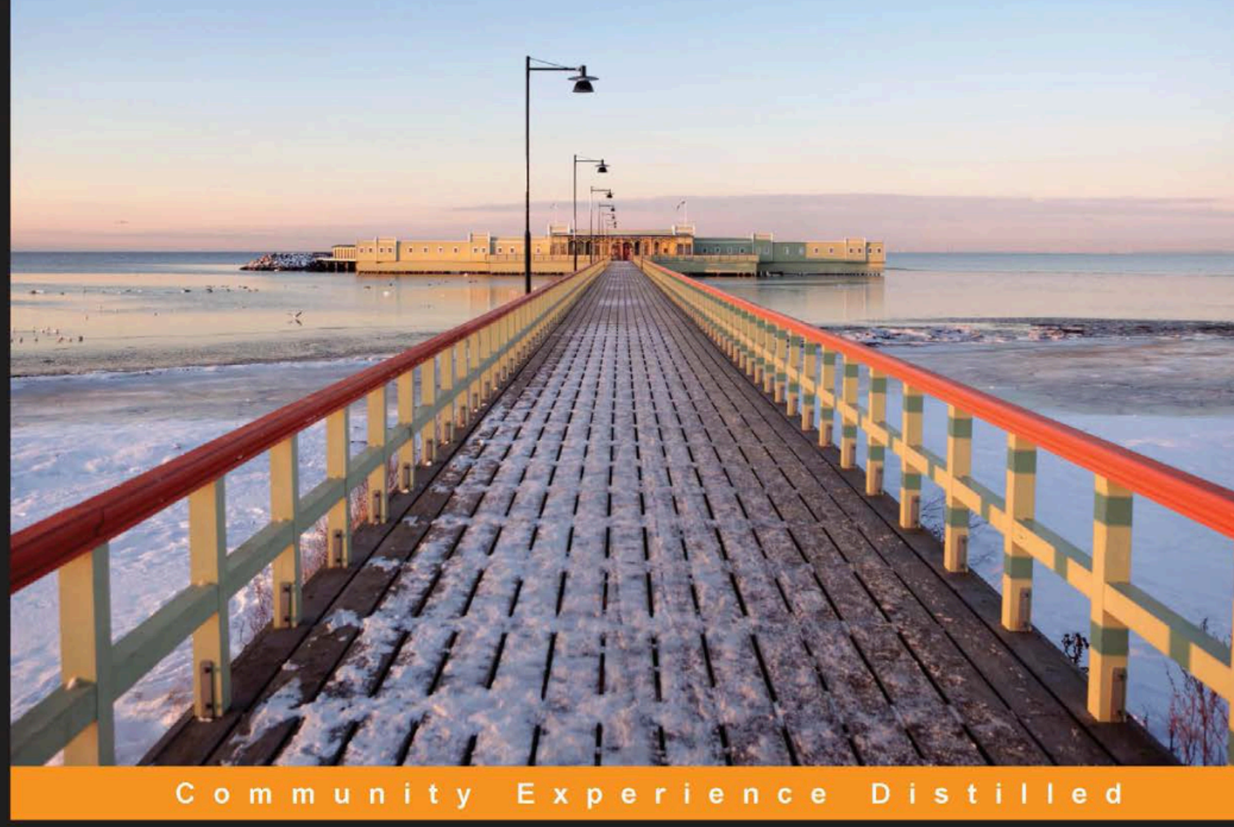
Testing with F#
Mikael Lundin

2015-05-13

valtech_



Testing with F#

Deliver high-quality, bug-free applications by testing them with efficient and expressive functional programming

F# Crash Course

```
tils
" ["Hello"; "FSharp"]
lo FSharp"
separator : string) =
educe (fun s1 s2 -> s1 + s
```

valtech_

F# Crash Course

```fsharp
module StringUtils

    // join " " ["Hello"; "FSharp"]
    // => "Hello FSharp"
    let join (separator : string) =
        List.reduce (fun s1 s2 -> s1 + separator + s2)
```

```dotnet
public static class StringUtils
{
    public static string Join(string separator, IEnumerable<string> words)
    {
        return words.Aggregate((s1, s2) => s1 + separator + s2);
    }
}
```

valtech_

F# Crash Course

```
module StringUtils

    // join " " ["Hello"; "FSharp"]
    // => "Hello FSharp"
    let join (separator : string) =
        List.reduce (fun s1 s2 -> s1 + separator + s2)



public static class StringUtils
{
    public static string Join(string separator, IEnumerable<string> words)
    {
        return words.Aggregate((s1, s2) => s1 + separator + s2);
    }
}
```

valtech_

F# Crash Course

```
module StringUtils

    // join " " ["Hello"; "FSharp"]
    // => "Hello FSharp"
    let join (separator : string) =
        List.reduce (fun s1 s2 -> s1 + separator + s2)



public static class StringUtils
{
    public static string Join(string separator, IEnumerable<string> words)
    {
        return words.Aggregate((s1, s2) => s1 + separator + s2);
    }
}
```

valtech_

F# Crash Course

```
module StringUtils

    // join " " ["Hello"; "FSharp"]
    // => "Hello FSharp"
    let join (separator : string) =
        List.reduce (fun s1 s2 -> s1 + separator + s2)
```

```
public static class StringUtils
{
    public static string Join(string separator, IEnumerable<string> words)
    {
        return words.Aggregate((s1, s2) => s1 + separator + s2);
    }
}
```

valtech_

F# Crash Course

```fsharp
module StringUtils

    // join " " ["Hello"; "FSharp"]
    // => "Hello FSharp"
    let join (separator : string) =
        List.reduce (fun s1 s2 -> s1 + separator + s2)
```

```dotnet
public static class StringUtils
{
    public static string Join(string separator, IEnumerable<string> words)
    {
        return words.Aggregate((s1, s2) => s1 + separator + s2);
    }
}
```

valtech_

F# Crash Course

```fsharp
module StringUtils

    // join " " ["Hello"; "FSharp"]
    // => "Hello FSharp"
    let join (separator : string) =
        List.reduce (fun s1 s2 -> s1 + separator + s2)
```

```dotnet
public static class StringUtils
{
    public static string Join(string separator, IEnumerable<string> words)
    {
        return words.Aggregate((s1, s2) => s1 + separator + s2);
    }
}
```

valtech_

F# Crash Course

```
module StringUtils

    // join " " ["Hello"; "FSharp"]
    // => "Hello FSharp"
    let join (separator : string) =
        List.reduce (fun s1 s2 -> s1 + separator + s2)



public static class StringUtils
{
    public static string Join(string separator, IEnumerable<string> words)
    {
        return words.Aggregate((s1, s2) => s1 + separator + s2);
    }
}
```

valtech_

F# Crash Course

```
module StringUtils

    // join " " ["Hello"; "FSharp"]
    // => "Hello FSharp"
    let join (separator : string) =
        List.reduce (fun s1 s2 -> s1 + separator + s2)



public static class StringUtils
{
    public static string Join(string separator, IEnumerable<string> words)
    {
        return words.Aggregate((s1, s2) => s1 + separator + s2);
    }
}
```

valtech_

F# Crash Course

```
module StringUtils

    // join " " ["Hello"; "FSharp"]
    // => "Hello FSharp"
    let join (separator : string) =
        List.reduce (fun s1 s2 -> s1 + separator + s2)
```

```
public static class StringUtils
{
    public static string Join(string separator, IEnumerable<string> words)
    {
        return words.Aggregate((s1, s2) => s1 + separator + s2);
    }
}
```

valtech_

F# Crash Course

```fsharp
module StringUtils

    // join " " ["Hello"; "FSharp"]
    // => "Hello FSharp"
    let join (separator : string) =
        List.reduce (fun s1 s2 -> s1 + separator + s2)
```

```dotnet
public static class StringUtils
{
    public static string Join(string separator, IEnumerable<string> words)
    {
        return words.Aggregate((s1, s2) => s1 + separator + s2);
    }
}
```

valtech_

F# Crash Course

```
module StringUtils

    // join " " ["Hello"; "FSharp"]
    // => "Hello FSharp"
    let join (separator : string) =
        List.reduce (fun s1 s2 -> s1 + separator + s2)



public static class StringUtils
{
    public static string Join(string separator, IEnumerable<string> words)
    {
        return words.Aggregate((s1, s2) => s1 + separator + s2);
    }
}
```

valtech_

Write our first Test

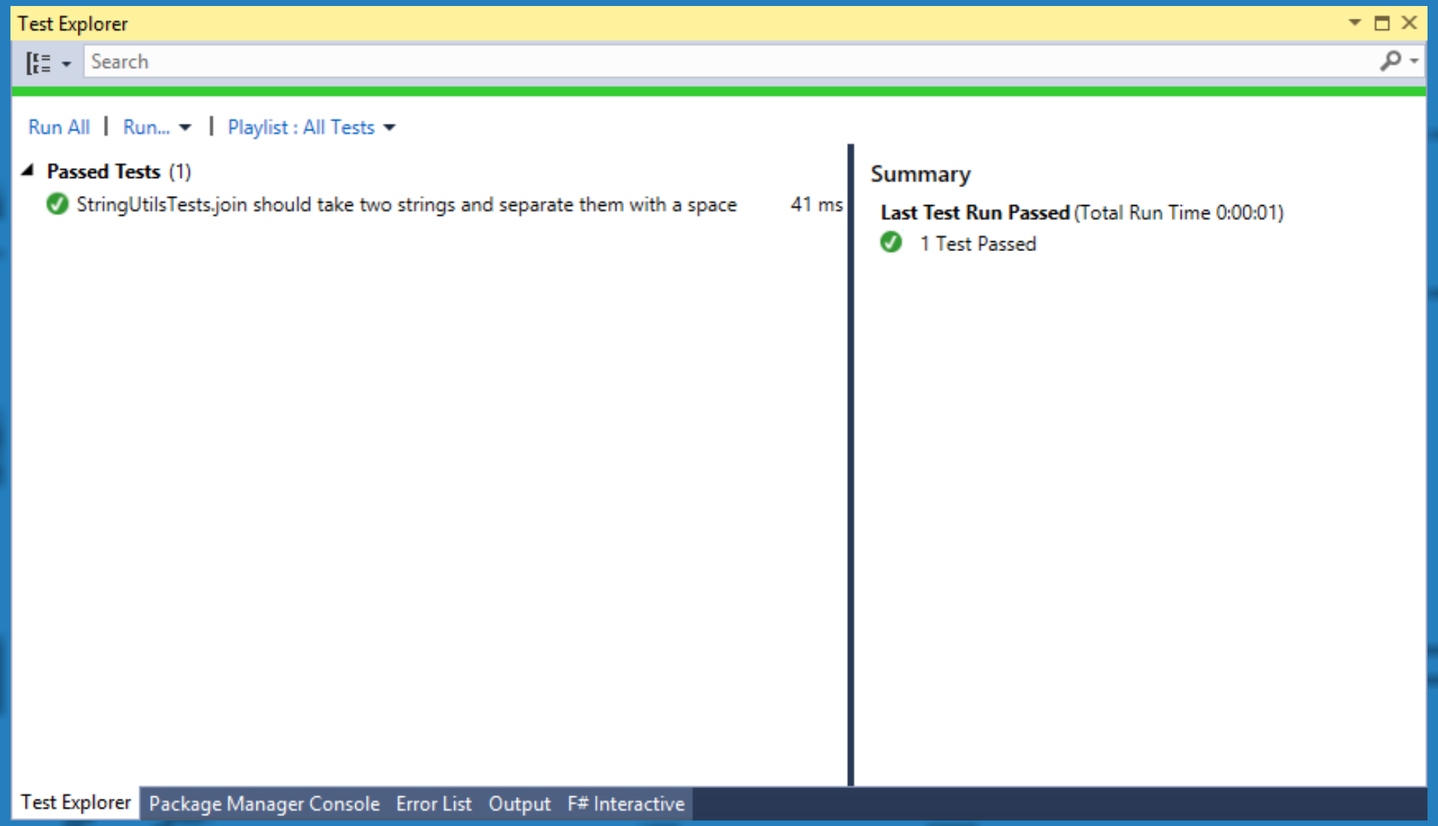
```
module StringUtilsTests

    open Xunit
    open StringUtils

    [<Fact>]
    let ``join should take two strings and separate them with a space`` () =
        Assert.Equal<string>("Hello FSharp", join " " ["Hello"; "FSharp"])
```

valtech_

Running our first Test



valtech_

FsUnit

```
open FsUnit.Xunit

[<Fact>]
let ``join should return same string when only one string is supplied`` () =
    join "!!!" ["Hello"] |> should equal "Hello"

[<Fact>]
let ``join should return empty string for empty list`` () =
    join "" [] |> should equal System.String.Empty
```

valtech_

Implementation does not fulfill test specification

StringUtilsTests.join should return empty string for empty list

Source: StringUtilsTests.fs line 18

Test Failed - StringUtilsTests.join should return empty string for empty list

Message: System.ArgumentException : The input list was empty.
Parameter name: list

Elapsed time: 6 ms

StackTrace:

```
ListModule.Reduce[T](FSharpFunc`2 reduction, FSharpList`1 list)
join@6-1.Invoke(FSharpList`1 list)
StringUtilsTests.join should return empty string for empty list()
```

valtech

New implementation

```
// join " " ["Hello"; "FSharp"]
// => "Hello FSharp"
let join (separator : string) list =
    if list = List.empty then
        System.String.Empty
    else
        list |> List.reduce (fun s1 s2 -> s1 + separator + s2)
```

valtech_

Custom asserts

```
[<Fact>]
let ``join should put whitespace between words`` () =
    ["How"; "much"; "wood"; "would"; "a"; "woodchuck"; "chuck";
      "if"; "a"; "woodchuck"; "could"; "chuck"; "wood"]
    |> join " "
    |> should match' @"^\w+(\s\w+){12}"
```

valtech_

Implementing regex assert

```
open System.Text.RegularExpressions
open NHamcrest
open NHamcrest.Core

let match' pattern =
    CustomMatcher<obj>(sprintf "Matches %s" pattern, fun c ->
        match c with
        | :? string as input -> Regex.IsMatch(input, pattern)
        | _ -> false)
```

valtech_

F# Code Quotations

```
<@ 1 + 2 = 3 @>
```

```
val it : Quotations.Expr<bool> =
  Call (None, op_Equality,
      [Call (None, op_Addition, [Value (1), Value (2)]), Value (3)])
    {CustomAttributes = [NewTuple (Value ("DebugRange"),
          NewTuple (Value ("stdin"), Value (1), Value (3), Value (1), Value (12)))];
```

valtech_

Unquote

```
unquote <@ (30 + 6) / 3 = (3 * 7) - 9 @>

(30 + 6) / 3 = 3 * 7 - 9
36 / 3 = 21 - 9
12 = 12
true

val it : unit = ()
```

valtech_

Unquote

```
[<Fact>]
let ``join should ignore empty strings`` () =
    test <@ join " " ["hello"; ""; "fsharp"] = "hello fsharp" @>
```

valtech_

Unquote

StringUtilsTests.join should ignore empty strings

Source: StringUtilsTests.fs line 41

Test Failed - StringUtilsTests.join should ignore empty strings

Message: StringUtils.join " " ["hello"; ""; "fsharp"] = "hello fsharp"
"hello fsharp" = "hello fsharp"
false

Elapsed time: 189 ms

StackTrace:

Operators.Raise[T](Exception exn)
StringUtilsTests.join should ignore empty strings()

valtech

Mocking Functional Dependencies

```
module HighScore

    type CsvReader = string -> string list list

    type Score = { Name : string; Score : int }

    let getHighScore (csvReader : CsvReader) =
        csvReader "highscore.txt"
        |> List.map (fun row ->
            match row with
            | name :: score :: [] -> { Name = name; Score = score |> int }
            | _ -> failwith "Expected row with two columns" )
        |> List.sortBy (fun score -> score.Score)
        |> List.rev
```

valtech_

Mocking Functional Dependencies

```
[<Fact>]
let ``should return highscore in descending order`` () =
    // arrange
    let getData (s : string) = [
        ["Mikael"; "1234"];
        ["Steve"; "321"];
        ["Bill"; "4321"]]

    // act
    let result = getHighScore getData

    // assert
    result |> List.map (fun row -> row.Score)
    |> should equal [4321; 1234; 321]
```

valtech_

Mocking Interface Dependencies

```
type ICsvReader =
    abstract member FileName : string
    abstract member ReadFile : unit -> string list list

type Score = { Name : string; Score : int }

let getHighScore (csvReader : ICsvReader) =
    csvReader.ReadFile()
    |> List.map (fun row ->
        match row with
        | name :: score :: [] -> { Name = name; Score = score |> int }
        | _ -> failwith "Expected row with two columns" )
    |> List.sortBy (fun score -> score.Score)
    |> List.rev
```

valtech_

Mocking Interface Dependencies

```
[<Fact>]
let ``should return highscore in descending order`` () =
    // arrange
    let csvReader =
        { new ICsvReader with
            member this.FileName = "highscore.txt"
            member this.ReadFile () = [
                ["Mikael"; "1234"];
                ["Steve"; "321"];
                ["Bill"; "4321"]]
        }

    // act
    let result = getHighScore csvReader

    // assert
    result |> List.map (fun row -> row.Score)
    |> should equal [4321; 1234; 321]
```

valtech_

Mocking with Foq

```
open Foq

[<Fact>]
let ``should return highscore in descending order`` () =
    // arrange
    let csvReader =
        Mock<ICsvReader>()
            .Setup(fun da -> <@ da.ReadFile() @>)
                .Returns([["Mikael"; "1234"];
                           ["Steve"; "321"];
                           ["Bill"; "4321"]])
            .Create()

    // act
    let result = getHighScore csvReader

    // assert
    result |> List.map (fun row -> row.Score)
    |> should equal [4321; 1234; 321]
```

valtech_

Functional Testing with TickSpec

```
Feature: Conway's Game of Life

Scenario 1: Any live cell with fewer than two live neighbours dies, as if caused by under-population.
    Given a live cell
    And has 1 live neighbour
    When turn turns
    Then the cell dies

Scenario 2: Any live cell with two or three live neighbours lives on to the next generation.
    Given a live cell
    And has 2 live neighbours
    When turn turns
    Then the cell lives

Scenario 3: Any live cell with more than three live neighbours dies, as if by overcrowding.
    Given a live cell
    And has 4 live neighbours
    When turn turns
    Then the cell dies

Scenario 4: Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.
    Given a dead cell
    And has 3 live neighbours
    When turn turns
    Then the cell lives
```

valtech_

Functional Testing with TickSpec

```
let mutable cell = Dead(0, 0)
let mutable cells = []
let mutable result = []

let [<Given>] ``a (live|dead) cell`` = function
    | "live" -> cell <- Live(0, 0)
    | "dead" -> cell <- Dead(0, 0)
    | _ -> failwith "expected: dead or live"

let [<Given>] ``has (\d) live neighbours?`` (x) =
    let rec _internal x =
        match x with
        | 0 -> [cell]
        | 1 -> Live(-1, 0) :: _internal (x - 1)
        | 2 -> Live(1, 0) :: _internal (x - 1)
        | 3 -> Live(0, -1) :: _internal (x - 1)
        | 4 -> Live(0, 1) :: _internal (x - 1)
        | _ -> failwith "expected: 4 >= neighbours >= 0"
    cells <- _internal x

let [<When>] ``turn turns`` () =
    result <- GameOfLife.next cells

let [<Then>] ``the cell (dies|lives)`` = function
    | "dies" -> Assert.True(GameOfLife.isDead (0, 0) result, "Expected cell to die")
    | "lives" -> Assert.True(GameOfLife.isLive (0, 0) result, "Expected cell to live")
    | _ -> failwith "expected: dies or lives"
```

valtech_

Functional Testing with TickSpec

TickSpec.Features.GameOfLifeFeature

Source: TickFact.fs line 51

Test Passed - Scenario 1: Any live cell with fewer than two live neighbours dies, as if caused by under-population.
Elapsed time: 69 ms
Output

Test Passed - Scenario 2: Any live cell with two or three live neighbours lives on to the next generation.
Elapsed time: 40 ms
Output

Test Passed - Scenario 3: Any live cell with more than three live neighbours dies, as if by overcrowding.
Elapsed time: 32 ms
Output

Test Passed - Scenario 4: Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.
Elapsed time: 32 ms
Output

valtech_

WebTests with Canopy

```
open canopy
open runner

[<EntryPoint>]
let main argv =
    start firefox

    context "Testing Valtech Start Page"

    "should redirect first visit to introduction" &&& fun _ ->
        // navigate
        url "http://www.valtech.se"

        // assert
        on "https://valtech.se/vi-gor/introduktion"

    "should have a valtech logo" &&& fun _ ->
        // navigate
        url "http://www.valtech.se"

        // assert
        ".header__logo" == "Valtech"

    run()
    quit()

    0 // return an integer exit code
```

valtech_

WebTests with Canopy

C:\Users\mikaellundin\Documents\Visual Studio 2013\Projects\TestingWithFSh...

```
context: Testing Valtech Start Page
Test: should redirect first visit to introduction
Passed
Test: should have a valtech logo
Passed

0 minutes 4 seconds to execute
2 passed
0 failed
```

valtech_

Property-Based Testing

#VTD15

valtech_

Thank you!

- http://valte.ch/TestingFSharpSlides
- http://valte.ch/TestingFSharpPaper

valtech_