William Blum

Principal Development Lead

Microsoft

Building a cloud service with F#

Lambda Days 2017, Kraków

Microsoft

Structure of the talk

- My journey to FP
- Project Springfield
- Why and how we built it in F#

My FP journey

Caml Light

$$e := x \mid \lambda x.\, e \mid e\ e$$

F#

.NET Blog

A first-hand look from the .NET engineering teams

Project Springfield: a Cloud Service Built Entirely in F#

December 13, 2016 by Visual FSharp Team [MSFT] // 17 Comments

This post was written by **William Blum**, a Principal Software Engineering Manager on the Springfield team at Microsoft Research.

Earlier this year, Microsoft announced a preview of Project Springfield, one of the most sophisticated tools Microsoft has for rooting out potential security vulnerabilities in software. Project Springfield is a fuzz testing service which finds security-critical bugs in your code.

One of the amazing things about Springfield is that it's built on top of Microsoft's development platforms – F#, .NET, and Azure! This post will go over some of the what, why, and how of Project Springfield, F#, .NET, and Azure.

What is Project Springfield?

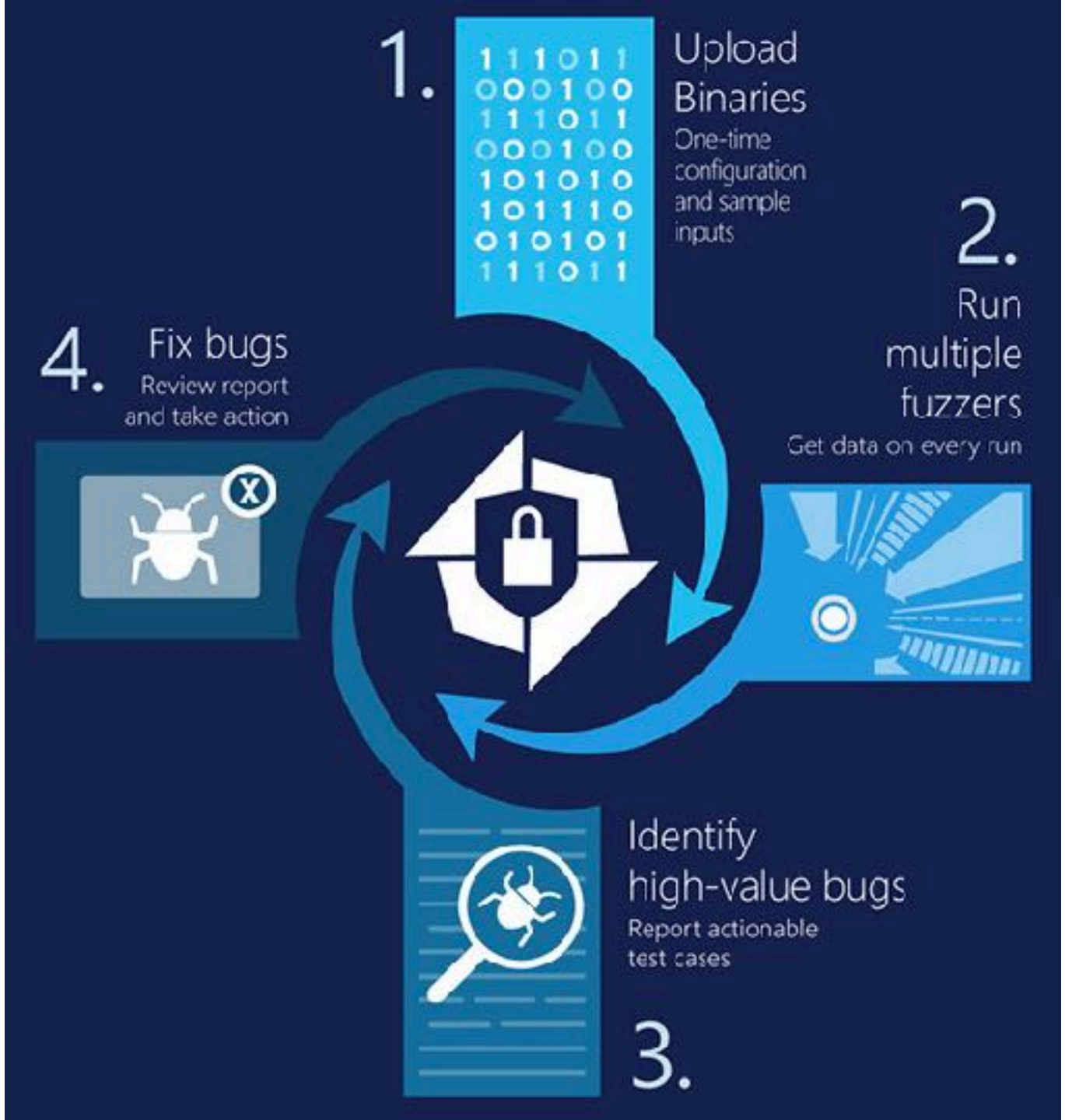
Project Springfield is Microsoft's unique fuzz testing service for finding security critical bugs in software. It helps you quickly adopt practices and technology from Microsoft. The service leverages the power of the Azure cloud to scale security testing using a suite of security testing tools from Microsoft. It's currently in preview, and you can sign up at Project Springfield if you want to give it a try!

Microsoft Research NExT

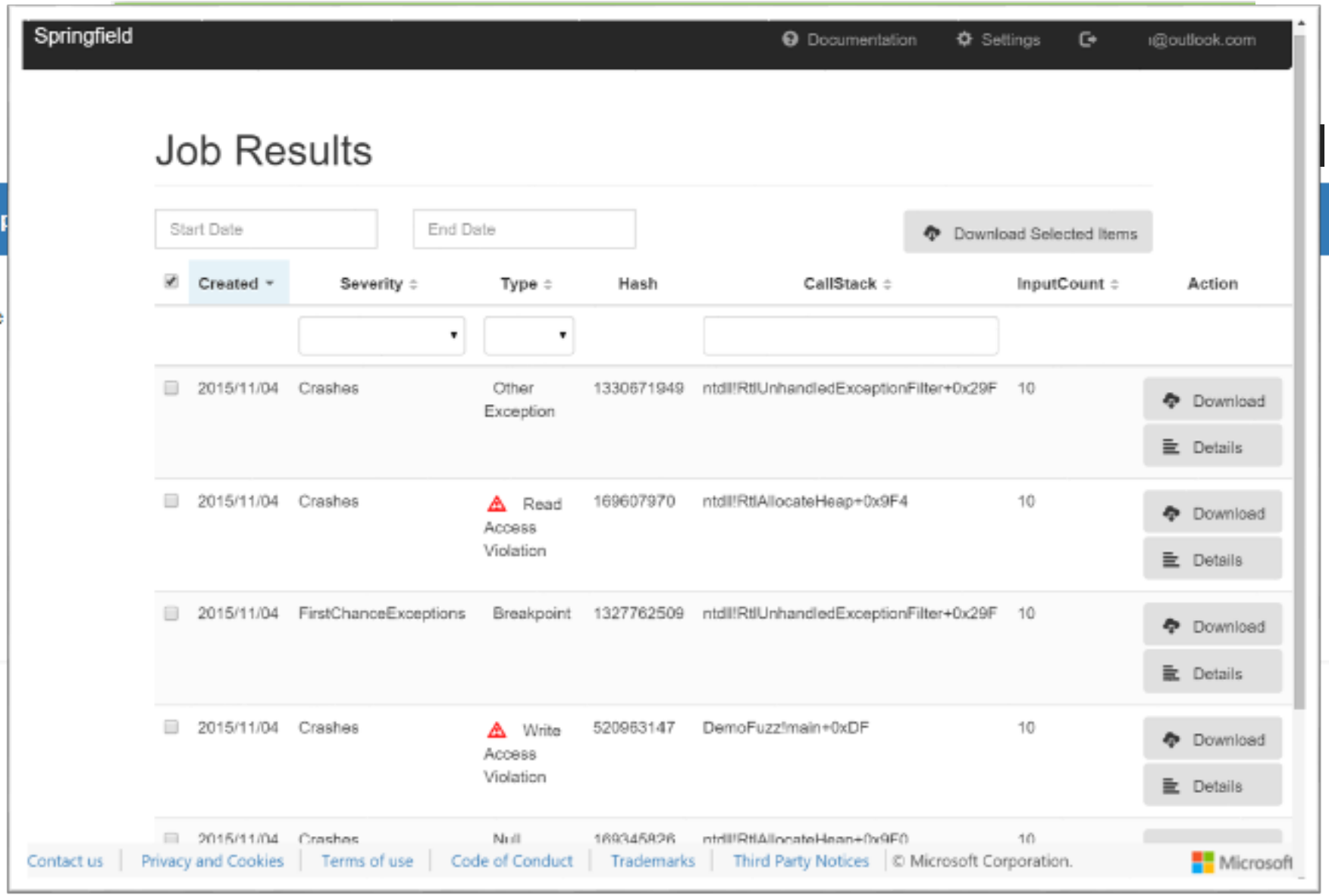
- **Natick** Undersea data center
- **Skype translator**
- **Cognitive services** API for NLP and Image recognition
- **DNA storage** “storing petabytes of data in a shoe box”
- **Holoportation** teleportation via HoloLens
- **Springfield**

Project Springfield

Security testing in the cloud running Microsoft fuzzing technologies



Springfield



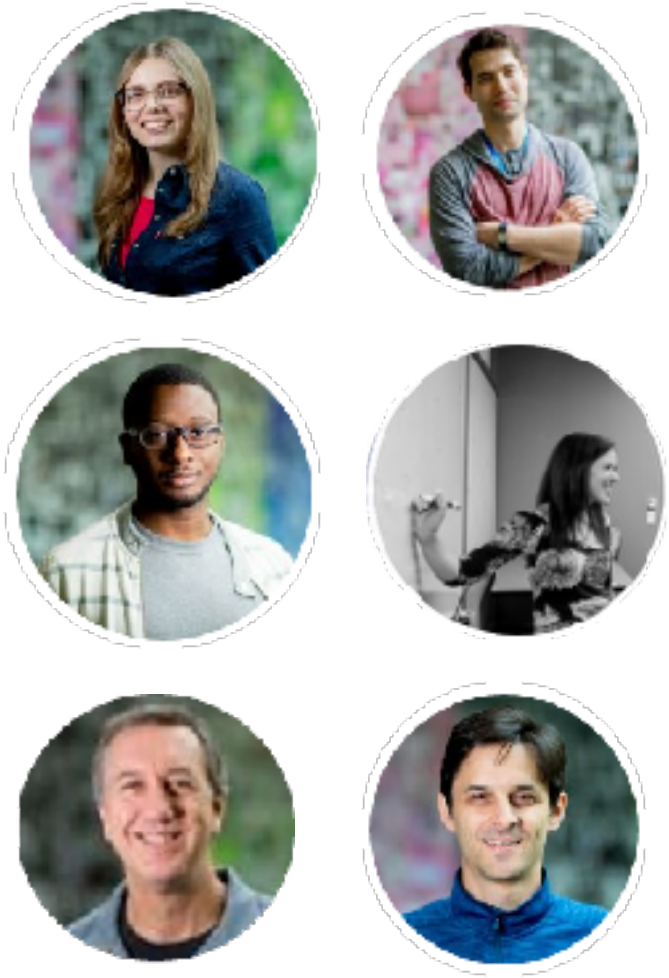
Springfield Dev Team

Started 2 years ago

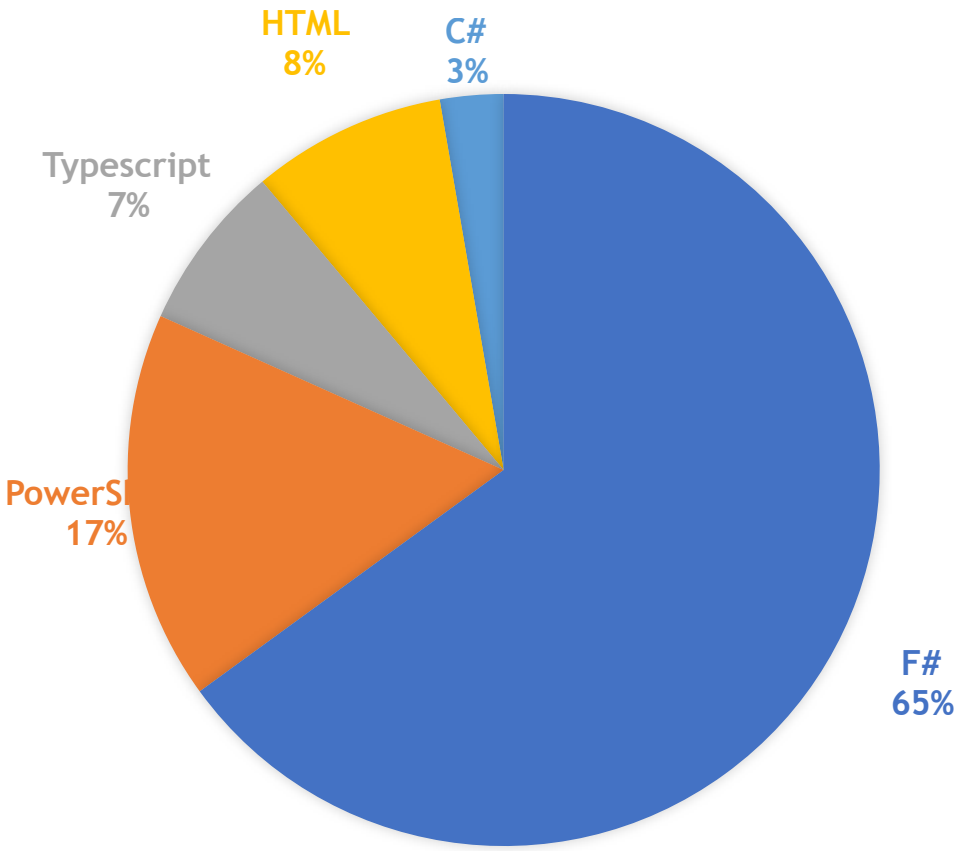
Initially 3 devs - Now 6 F# devs

Team background:

- C/C++, C#, Java
- Some previous F# experience
- All interested in learning more about FP!



Springfield Tech Stack



F#
- Frontend, Backend, Client, Operation, Core API

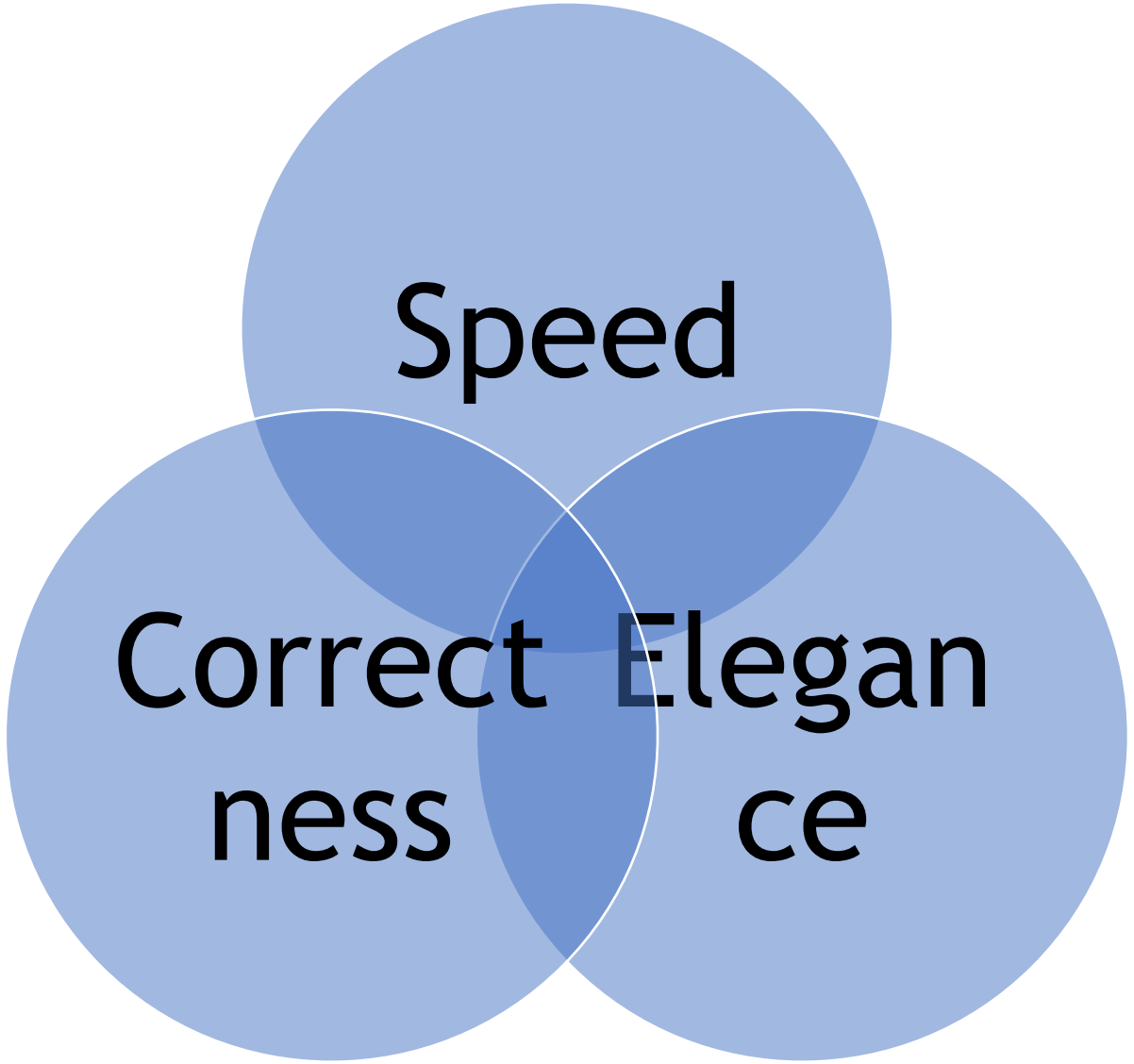
Azure
- Compute, Network, Storage, App Services
- Service Fabric

TypeScript
- Web client

PowerShell
- Deployment

Fuzzing tools
- Tools from Office Team and MSR (written in C# and C++)

Why F#?



Fast prototyping
Fast onboarding
=> MVP in 3 months

Lean, focused team, no dedicated test resource.

Fun programming!

Functional/F# benefits

Speed

- Concise
- Intentional
- Interactive REPL
- **Tooling/IDE support**
- **C#/.NET Interop**
- **.NET libraries**
- **Community support**

Correctness

- Strong type checking
- Immutability
- option type
- printf type checking
- Theoretical foundation
- **Unit of measure**

Elegance

- Lambda abstractions
- Type inference
- Let-polymorphism
- Recursion
- Pattern matching
- Algebraic types
- **Type providers**
- **Computational expressions**
- **Async worfklows**
- **Active patterns**

“Why bother with F# now that C# has lambdas?”

Feature	F#	C#
Type inference	1.0 (built in)	3.0 (2007)
Lambda	1.0 (built in)	3.0 (2007)
REPL	1.0 (built in)	6.0
Asynchronous programming	2007	5.0 (2012)
Immutability by default	1.0 (built in)	Not planned
Safe printf	1.0 (built in)	Not planned
Algebraic types, option type	1.0 (built in)	Not planned
null safety	1.0 (built in)	Non-nullable? 7.0
Type providers	3.0, 2012	Not planned
Computational Expression	2.0	Not planned
Local functions, Tuples, records, pattern matching	1.0 (Built in)	7.0?

Correctness: option type

“NULL pointers are the biggest problem in C/C++, at least in our code. The dual use of a single value as both a flag and an address causes an incredible number of fatal issues. C++ references should be favored over pointers (…) Perform NULL checks when pointers are turned into references, then you can ignore the issue thereafter. There are a lot of deeply ingrained game programming patterns that are just dangerous, but I'm not sure how to gently migrate away from all the NULL checking.”

John Carmack

http://www.gamasutra.com/view/news/128836/InDepth_Static_Code_Analysis.php

```
type 'a Option =
| None
| Some of 'a
```

```
match o with
| None ->
| Some v ->
```

Correctness: typed-checked printf

“Printf format errors were the second biggest issue in our codebase, (...) passing an idStr almost always results in a crash, but annotating all our variadic functions with kills this problem dead. There were dozens of these hiding in informative warning messages that would turn into crashes (…), which is also a comment about how the code coverage of our general testing was lacking.”

John Carmack

http://www.gamasutra.com/view/news/128836/InDepth_Static_Code_Analysis.php

```
open Microsoft.FSharpLu.TraceLogging
Trace.info "Created %d machines for %s at %O"
                machines name System.DateTime.Now
```

Recruiting: FP attracts talents!

Growing adoption

- Scala used at Twitter, Amazon
- Elixir growing popularity in the Bay Area
- Elm and PureScript are growing for web front-end
- Haskell/Ocaml at Facebook, Jane Street Capital
- Apple Swift taking over Objective C
- F# at Jet.com

Hired 4 programmers because of F#:

- 3 C#/Java dev who wanted to do F# full time
- 1 C#, C/C++ who show interest in our team because of F# blog post

Domain types expressed with Records + ADT + Discriminated Unions

```
type VmParameters =
    {
        customerId : CustomerId
        jobId : JobId
        runId : JobRunId
        submitDate : System.DateTime
        appInsightsKey: string option
        vmPurposeParameters : VmPurposeParameters
        version : Version
    }
```

```
type VmPurposeParameters =
| CustomerVM of CustomerVmParameters
| FuzzingVM of FuzzingVmParameters
```

```
type VmSize =
    | ExtraSmall
    | Small
    | Medium
    | Large
    | ExtraLarge
```

Reliability via agent-based programming

```
type Transition<'s, 't> =
| RetryIn of System.TimeSpan
| Goto of 's
| Return of 't
```

```
type Channel<'m> =
 {
    send : 'm -> Async<unit>
    sendIn : 'm -> System.TimeSpan -> Async<unit>
 }
```

```
type Agent<'s, 't, 'm> =
  {
    transition : Channel<'m> -> 's
                 -> Async<Transition<'s, 't>>
    serializeState : 's -> 'm
    persist : 'm -> System.TimeSpan -> unit
    channel : Channel<'m>
  }
```

```
'a -> ('b -> Async<'c>)
let createJob (channel:Agent.Channel<_>) = function
    | JobCreationRequest ->
        async {
            let! name = createAzureVM ()
            return Goto <| WaitForVM(name)
        }
    | WaitForVM name ->
        async {
            let! ready = isVMDeployed name
            if ready then
                return Goto <| JobCreated(name)
            else
                return RetryIn 5<s>
        }
    | JobCreated name ->
        async {
            do! NotifyUserVMIsReady()
            return Return name
        }
```

Resumable monad

```
let myServiceApi =
    resumable {
        let! machineName = resumable { return Environment.getMachineName () }
        let! requestId = resumable { return Environment.provisionVM machineName }

        let! vmready = resumable {
            while not <| Environment.vmRequestSucceeded requestId do
                printfn "Waiting for cloud request to complete..."
                System.Threading.Thread.Sleep(1000)
            return true
        }

        printfn "Request completed for machine %s!" machineName
        return machineName, requestId, vmready
    }
```

http://blumu.github.io/ResumableMonad/

JSON serialization

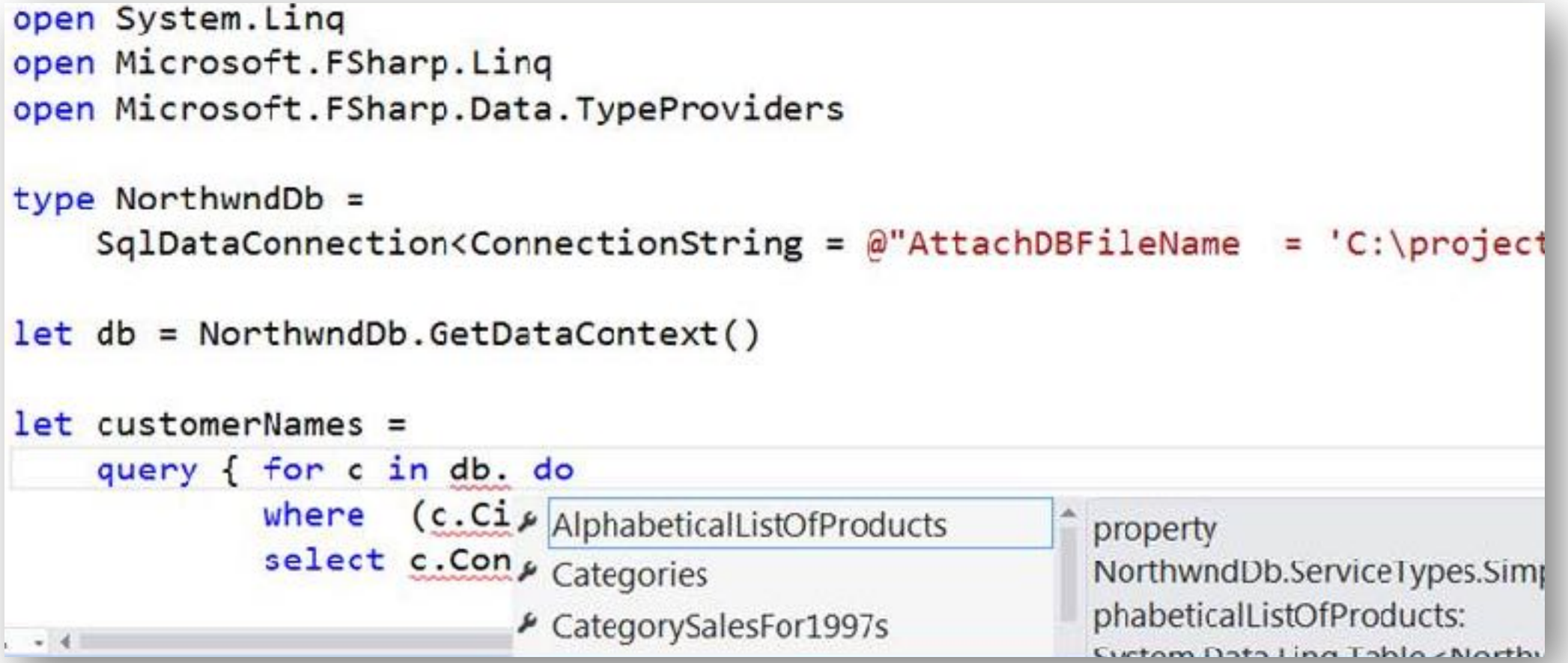
```
type 'a Tree =
  Leaf of 'a | Node of 'a Tree * 'a Tree

let x = Node (Node((Leaf 1), (Leaf 4)), Leaf
6)
```

```
Default.serialize x  =

"{ "Case": "Node", "Fields":
[ { "Case": "Node", "Fields":
[ { "Case": "Leaf", "Fields": [ 1 ] },
{ "Case": "Leaf", "Fields":
[ 4 ] } ] }, { "Case": "Leaf",
"Fields": [ 6 ] } ] }"
```

```
Compact.serialize x =

 "{ "Node": [ {
      "Node": [ { "Leaf": 1 },
                { "Leaf": 4 } ] },
              { "Leaf": 6 } ] }"
```

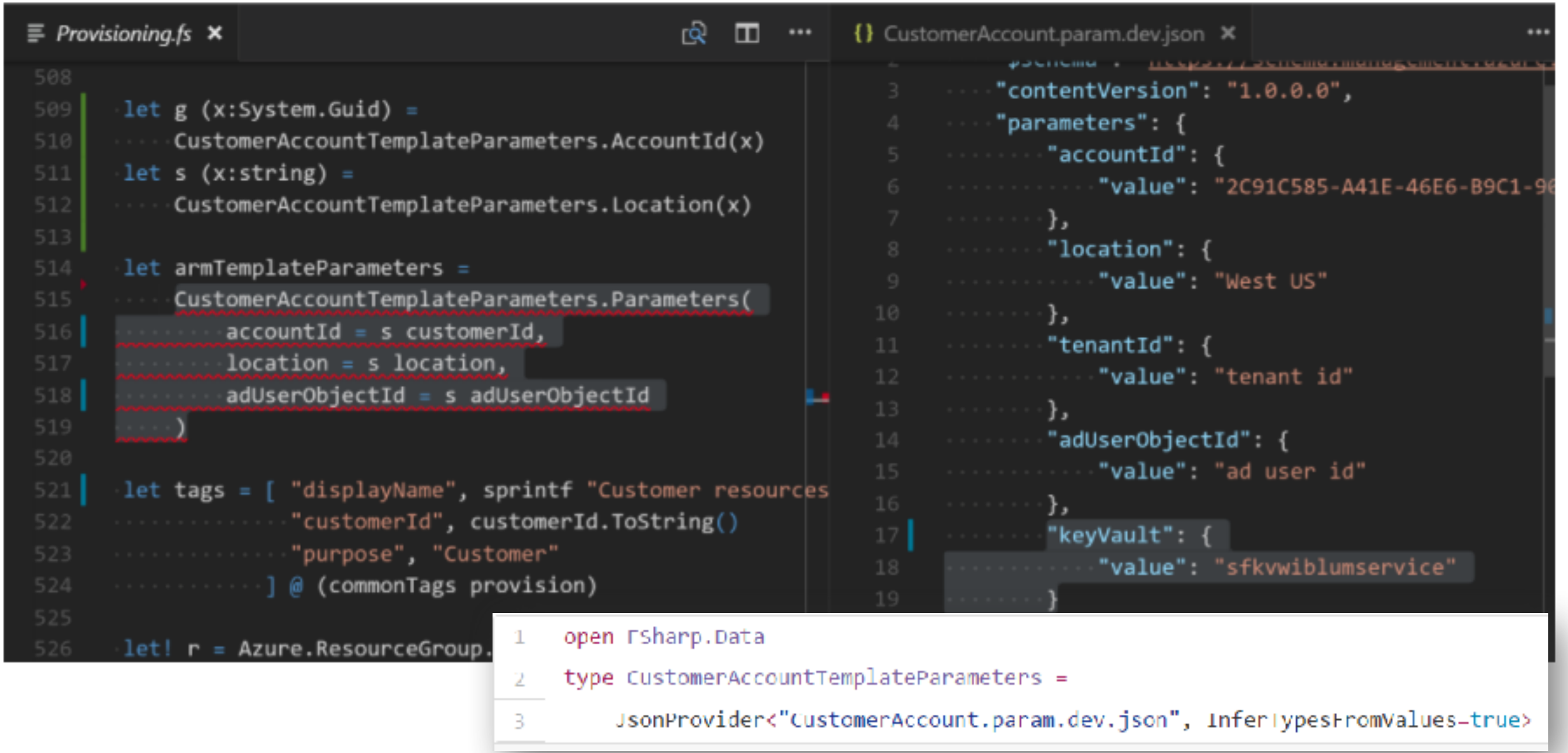
https://github.com/Microsoft/fsharplu/wiki/fsharplu.json

F# Type Providers

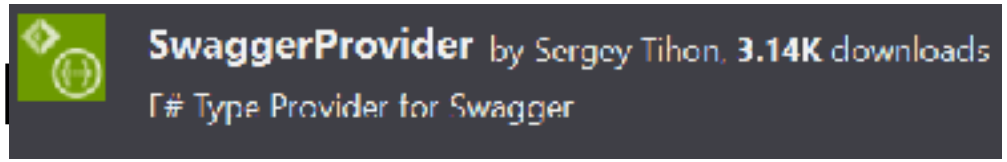


F# type providers in action: https://blogs.msdn.microsoft.com/dsyme/2013/01/30/twelve-f-type-providers-in-action/

Statically checking data with Type Providers



Azure and Type Provider




```
open SwaggerProvider
type Azure =
    SwaggerProvider<
        "https://raw.githubusercontent.com/Azure/azure-rest-api-specs/master/arm-compute/2016-03-30/swagger/compute.json">

let a = new Azure(Headers = authHeader)

a.Vi
```

VirtualMachineExtensionImagesGet
VirtualMachineExtensionImagesListTypes
VirtualMachineExtensionImagesListVersions

Azure.VirtualMachineExtensionImagesGet(location: string, publisherName: string, type: string, version: st
subscriptionId: string) : Azure.VirtualMachineExtensionImage

```
a.VirtualMachinesListAll(sdkVersion,subscriptionId).Value
|> Seq.countBy (fun vm-> vm.Properties.HardwareProfile.VmSize)
```

Asynchronous programming with F#

- Springfield code entirely async
- C# pitfall avoided
- Implemented as a computational expression, not a compiler trick

```
'a -> 'b -> Async<unit>
let deleteVhd (storage:BlobStorageContext) path =
    async {
        let container = storage.client.GetContainerReference("container")
        let blob = container.GetBlobReference(path)
        let! r = blob.DeleteIfExistsAsync().AsAsync
        if not r then
            invalidOp "VHD blob does not exist"
    }
```

Handling asynchronous exceptions with Active patterns

```
try
    client.ResourceGroups.Delete(groupName)
with
| :? AzureException as e
  when e.code = "ResourceGroupNotFound" ->
    return failwith "Aggregated NotFound exception!"
```

Exception not caught!

Because .Net Async Tasks wrap exceptions in ***System.AggregateException***

Active Pattern matching aggregated exceptions

```
(exn -> 'a option) -> System.Exception -> 'a option
let rec (|IsAggregateOf|_|) condition (e:System.Exception) =
    match e with
    | :? System.AggregateException as e ->
        Seq.tryPick (IsAggregateOf condition) e.InnerExceptions
    | e -> condition e
```

Matching condition

```
System.Exception -> string option
let SomeResourceGroupNotFoundException (e:System.Exception) =
    match e with
    | :? AzureException as e
        when e.code = "ResourceGroupNotFound" -> Some e.code
    | _ -> None
```

Exception properly caught!

```
try
    client.ResourceGroups.Delete(groupName)
with
| IsAggregateOf SomeResourceGroupNotFoundException e ->
    return failwith "Aggregated NotFound exception!"
```

Port to F# stories

PowerShell scripts

statically checked, strongly typed, **code 2.73 smaller**

Perl scripts

37% smaller, more readable, better logging (.Net), ETW telemetry, testable, …

C# frontend

Web, Database Abstraction Layer (EF)

40% smaller (thousands LOC)

Testimonials from Springfield dev team

- *“F# equipped me with new tools to see and decompose problems in new ways.”*
- *“F# changed the way I look at solving problems and writing code (…) allows me to start thinking in code“*
- *“Refactoring became an enjoyable process.”*

F# at Microsoft

Product teams

Bing: ad sales simulation

Xbox Live: Matchmaking

Windows: Compatibility Testing & Tools

Azure Batch

Product Supporting F#

Visual Studio, Azure Notebooks, Azure Function

Research

Program verification: termination analysis (SLAM), memory safety checker (SLAyer), verification language (F*)

Quantum Computing: Liquid tool suite

Bioinformatics: DNA and genetic programming (DSD, GEC, SPiM)

Distributed Computing/Big Data: Prajna

Free fsharp.org stickers!

Appropriate use of C++

Thanks

- Lambda Days organizers
- Springfield Team at Microsoft
- Phillip Carter (F# team)

Relevant Links

https://blogs.msdn.microsoft.com/dotnet/2016/12/13/project-springfield-a-cloud-service-built-entirely-in-f/

http://fsharpforfunandprofit.com/posts/conciseness-intro/

http://blumu.github.io/ResumableMonad/

https://azure.microsoft.com/en-us/blog/scaling-up-project-springfield-using-azure/

https://github.com/Microsoft/fsharplu/wiki

Backup slides

Features coming in VS 2017

- Tooling support in VS (Roslyn)
- Feature parity with C#/VB
- F# the best-tooled functional programming language in the market
- .Net Core Linux support

F# tool to reduce beta-redex with traversals

http://william.famille-blum.org/research/tools.html