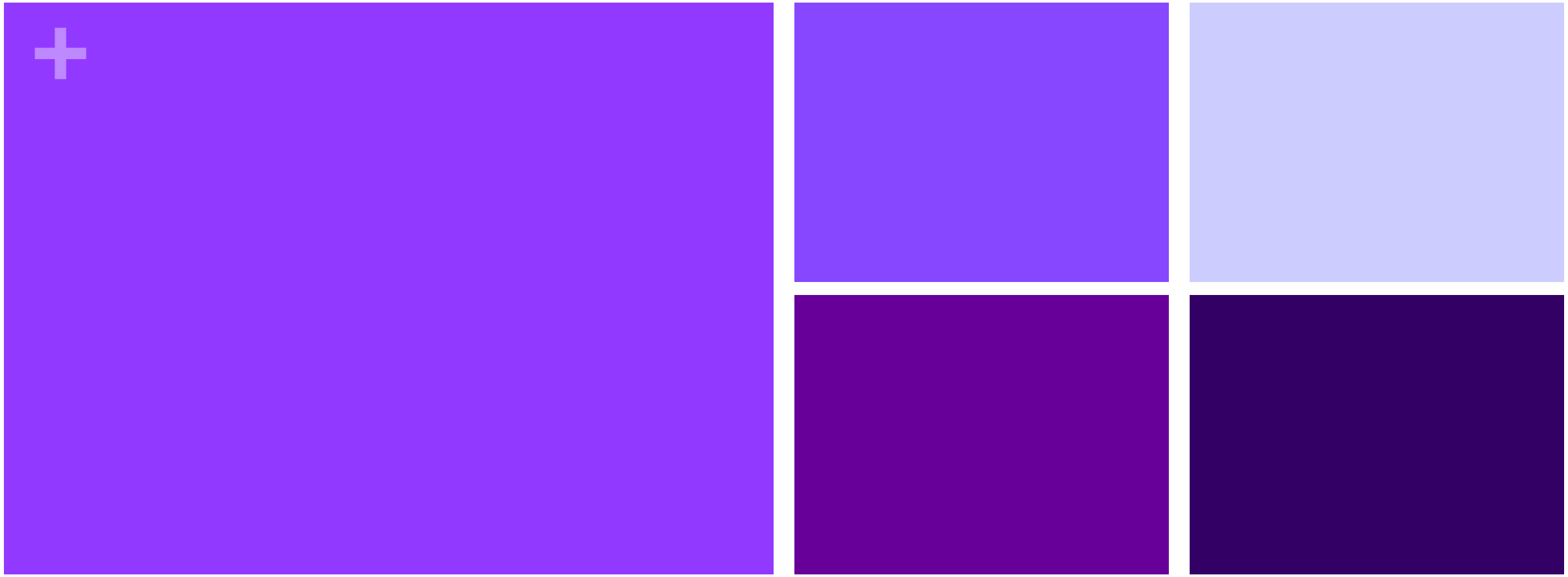


Jet loves F#. Here's why!

Rachel Reese | @rachelreese | @JetTechnology

A brief history

Programming languages evolve just like regular languages.

Mother Tongues

Tracing the roots of computer languages through the ages

Just like half of the world's spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, Cobol, Java, and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers – electronic lexicographers, if you will – aim to save, or at least document, the lingo of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten lingua francas. Among the most endangered are Ada, APL, B (the predecessor of C), Lisp, Oberon, Smalltalk, and Simula.

Code-raker Grady Booch, Rational Software's chief scientist, is working with the Computer History Museum in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time," Booch explains. "They'll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure." Here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive rundown, check out the Language List at *www.informatik.uni-freiburg.de/Java/misc/lang_list.html*. **– Michael Menduno**

Key

1954 **Year Introduced**

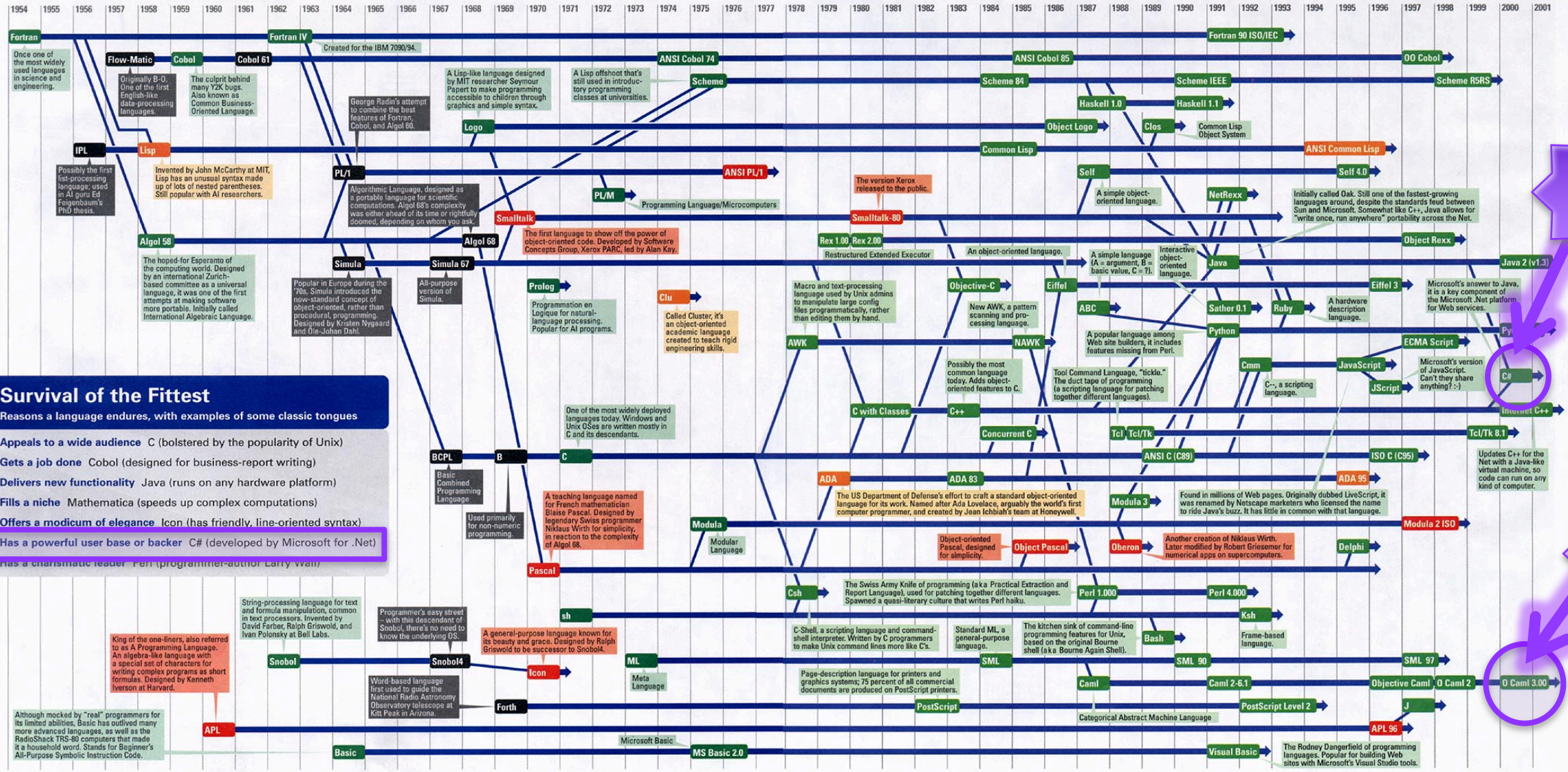
Active: thousands of users

Protected: taught at universities; compilers available

Endangered: usage dropping off

Extinct: no known active users or up-to-date compilers

Lineage continues



Survival of the Fittest

Reasons a language endures, with examples of some classic tongues

- **Appeals to a wide audience** C (bolstered by the popularity of Unix)
- **Gets a job done** Cobol (designed for business-report writing)
- **Delivers new functionality** Java (runs on any hardware platform)
- **Fills a niche** Mathematica (speeds up complex computations)
- **Offers a modicum of elegance** Icon (has friendly, line-oriented syntax)
- **Has a powerful user base or backer** C# (developed by Microsoft for .Net)
- **Has a charismatic leader** Perl (programmer-author Larry Wall)

C#

F#

Sources: Paul Boutin; Brent Hailpern, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gio Wiederhold, computer scientist, Stanford University

Why F#?

Testimonials

> I have now delivered three business critical projects written in F#. I am still waiting for the first bug to come in.

- Simon Cousins
UK-based power company

> The F# solution offers us an order of magnitude increase in productivity and allows one developer to perform the work [of] a team of dedicated developers…

- Yan Cui
Lead server Engineer, Gamesys

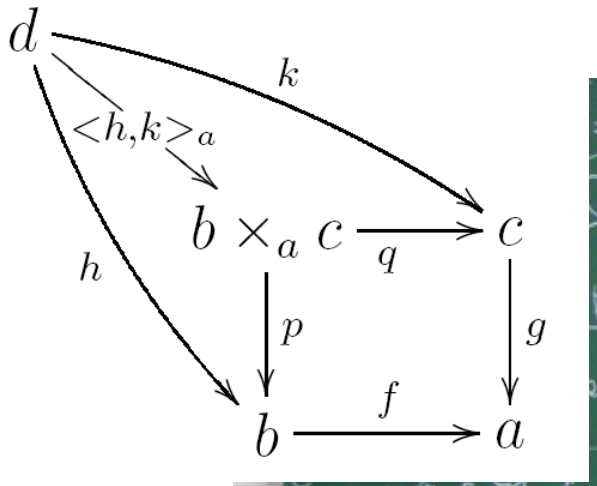
> The F# code is consistently shorter, easier to read, easier to refactor and contains far fewer bugs.

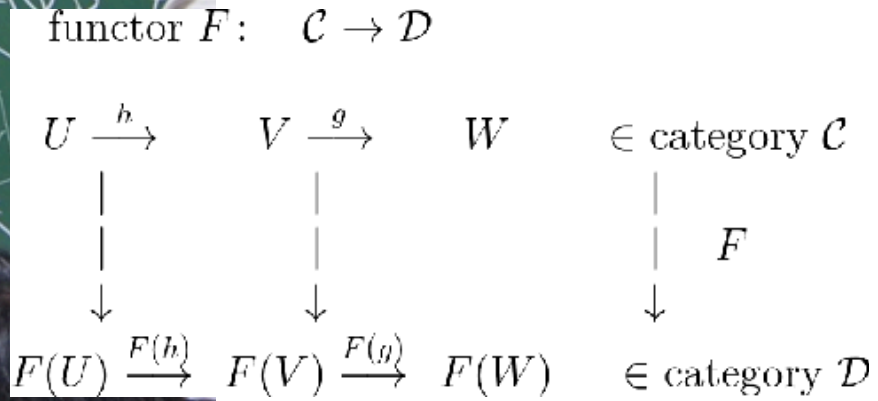
- Kaggle, a predictive modelling company

All of this seems incredible, right?

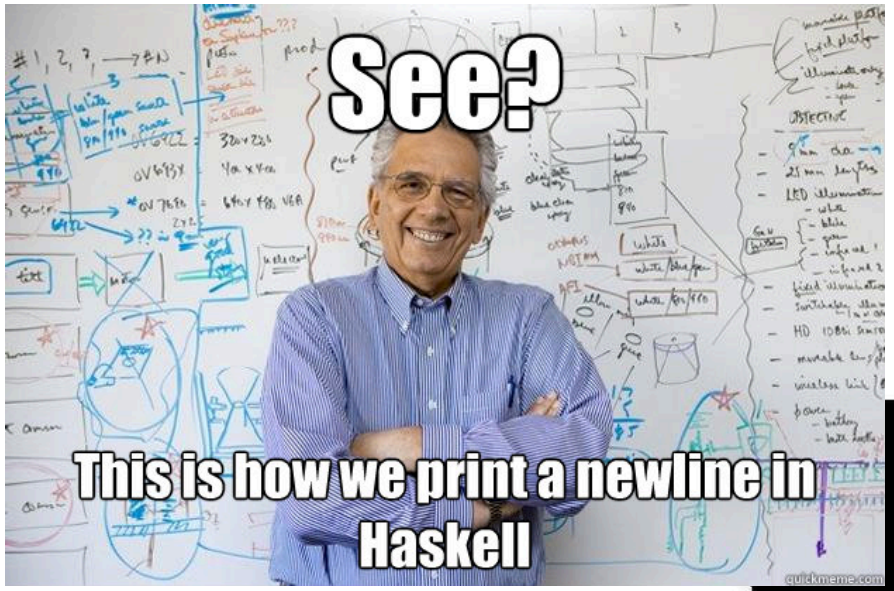
So why doesn’t everyone use F#?

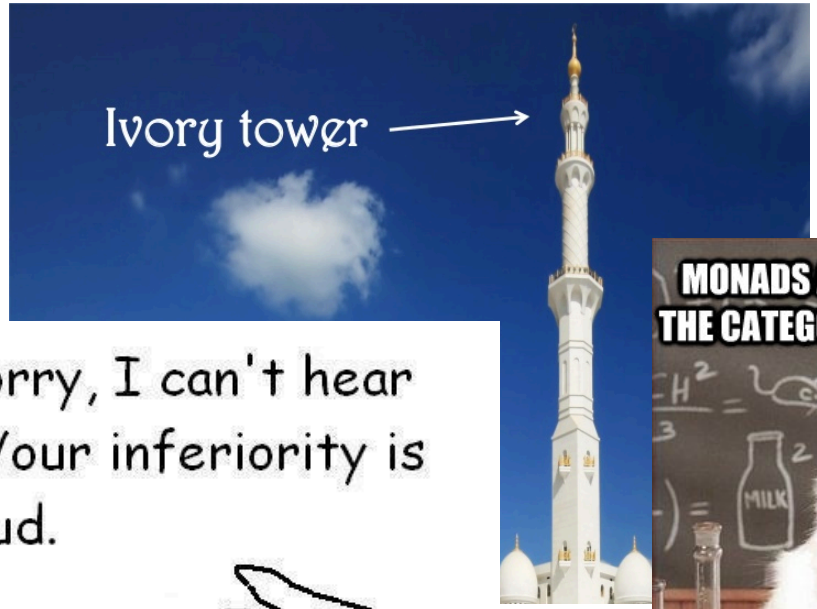
There's a lot of math behind programming languages like F#.

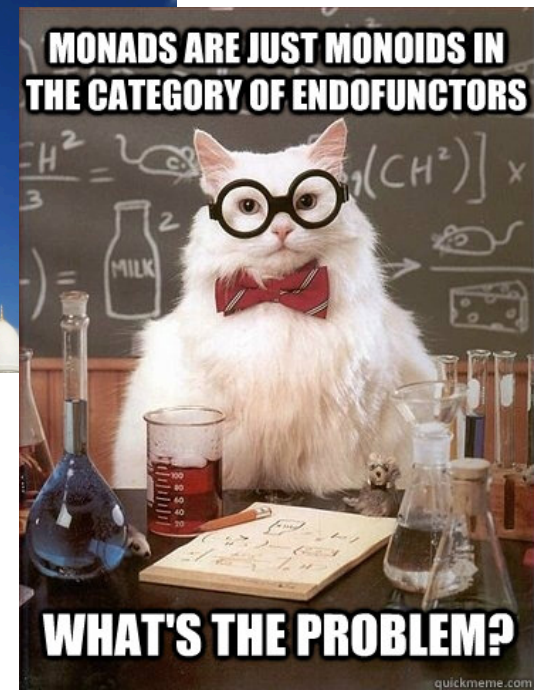


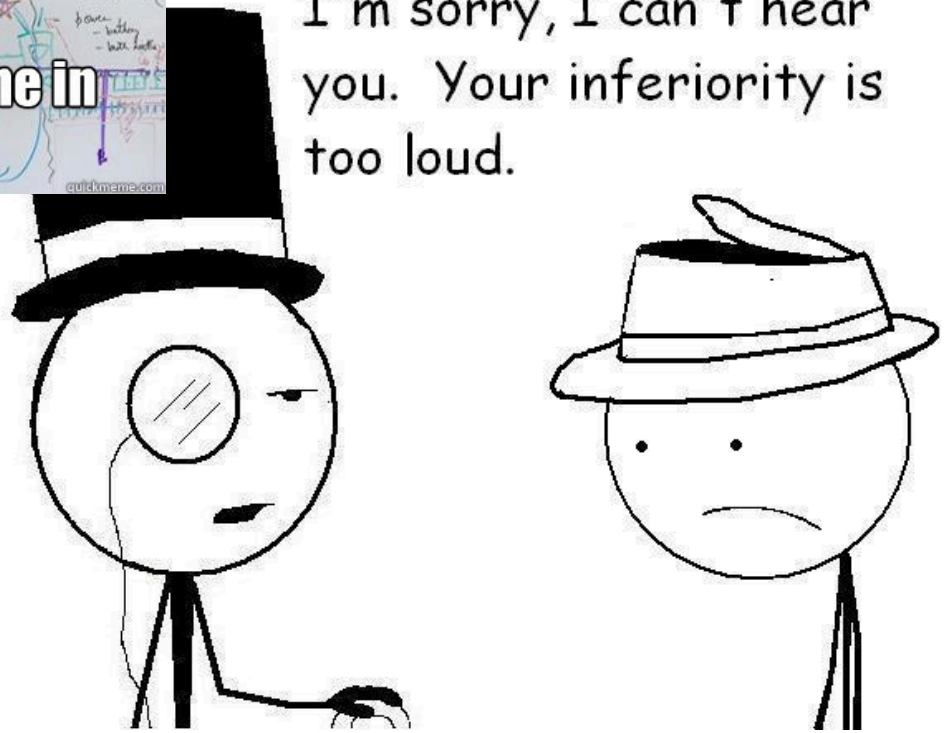


So they have this reputation.









This is Sean.

Sean is 8.

Sean spoke at a 2000-person technical conference this year. On F#.

Why F#?

Concise code & expanded feature set

Concise code

C# - 23 lines

```
using System.Net
using System
using System.IO

class WebPageDownloader
{
    public TResult FetchUrl<TResult>(
            Func<string, StreamReader, TResult> callback,
            string url)
    {
        var req = WebRequest.Create(url);
        using (var resp = req.GetResponse())
        {
            using (var stream = resp.GetResponseStream())
            {
                using (var reader = new StreamReader(stream))
                {
                    return callback(url, reader);
                }
            }
        }
    }
}
```

F# - 10 lines

```
open System.Net
open System
open System.IO

let fetchUrl callback url =
    let req = WebRequest.Create(Uri(url))
    use resp = req.GetResponse()
    use stream = resp.GetResponseStream()
    use reader = new IO.StreamReader(stream)
    callback reader url
```

From Comparing F# with C#, fsharpforfunandprofit.com

Concise and powerful code

C#

```
public abstract class Transport{ }

public abstract class Car : Transport {
  public string Make { get; private set; }
  public string Model { get; private set; }
  public Car (string make, string model) {
    this.Make = make;
    this.Model = model;
  }
}

public abstract class Bus : Transport {
  public int Route { get; private set; }
  public Bus (int route) {
    this.Route = route;
  }
}

public class Bicycle: Transport {
  public Bicycle() {
  }
}
```

F#

```
type Transport =
    | Car of Make:string * Model:string
    | Bus of Route:int
    | Bicycle
```

Same info as C#!

Trivial to pattern match on!

C#

F# pattern match

Name	Description	Example
Constant pattern	Any numeric, character, or string literal, an enumeration constant, or a defined literal identifier	`1.0`, `"test"`, `30`, `Color.Red`
Identifier pattern	A case value of a discriminated union, an exception label, or an active pattern case	`Some(x)` `Failure(msg)`
Variable pattern	*identifier*	`a`
as pattern	*pattern* as *identifier*	`(a, b) as tuple1`
OR pattern	*pattern1* | *pattern2*	`([h] | [h; _])`
AND pattern	*pattern1* & *pattern2*	`(a, b) & (_, "test")`
Cons pattern	*identifier* :: *list-identifier*	`h :: t`
List pattern	[*pattern_1*; ... ; *pattern_n*]	`[ a; b; c ]`
Array pattern	[| *pattern_1*; ..; *pattern_n* |]	`[| a; b; c |]`
Parenthesized pattern	(*pattern*)	`( a )`
Tuple pattern	(*pattern_1*, ... , *pattern_n*)	`( a, b )`
Record pattern	{ *identifier1* = *pattern_1*; ... ; *identifier_n* = *pattern_n* }	`{ Name = name; }`
Wildcard pattern	_	`_`
Pattern together with type annotation	*pattern* : *type*	`a : int`
Type test pattern	:? *type* [as *identifier*]	`:? System.DateTime as dt`
Null pattern	null	`null`

Concise and powerful code

C#

```
public abstract class Transport{ }

public abstract class Car : Transport {
  public string Make { get; private set; }
  public string Model { get; private set; }
  public Car (string make, string model) {
    this.Make = make;
    this.Model = model;
  }
}

public abstract class Bus : Transport {
  public int Route { get; private set; }
  public Bus (int route) {
    this.Route = route;
  }
}

public class Bicycle: Transport {
  public Bicycle() {
  }
}
```

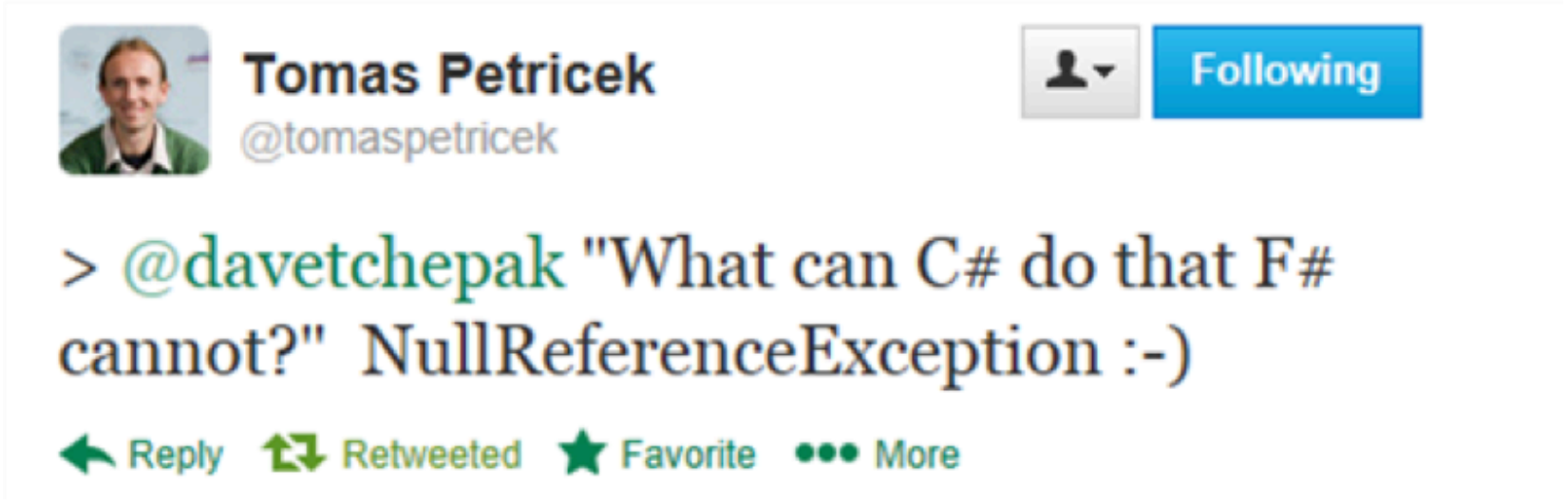
F#

```
type Transport =
    | Car of Make:string * Model:string
    | Bus of Route:int
    | Bicycle


let getThereVia (transport:Transport) =
    match transport with
       | Car (make,model) -> ...
       | Bus route -> ...
```

Warning FS0025: Incomplete pattern matches on this expression. For example, the value 'Bicycle' may indicate a case not covered by the pattern(s)

Expanded feature set: Option types



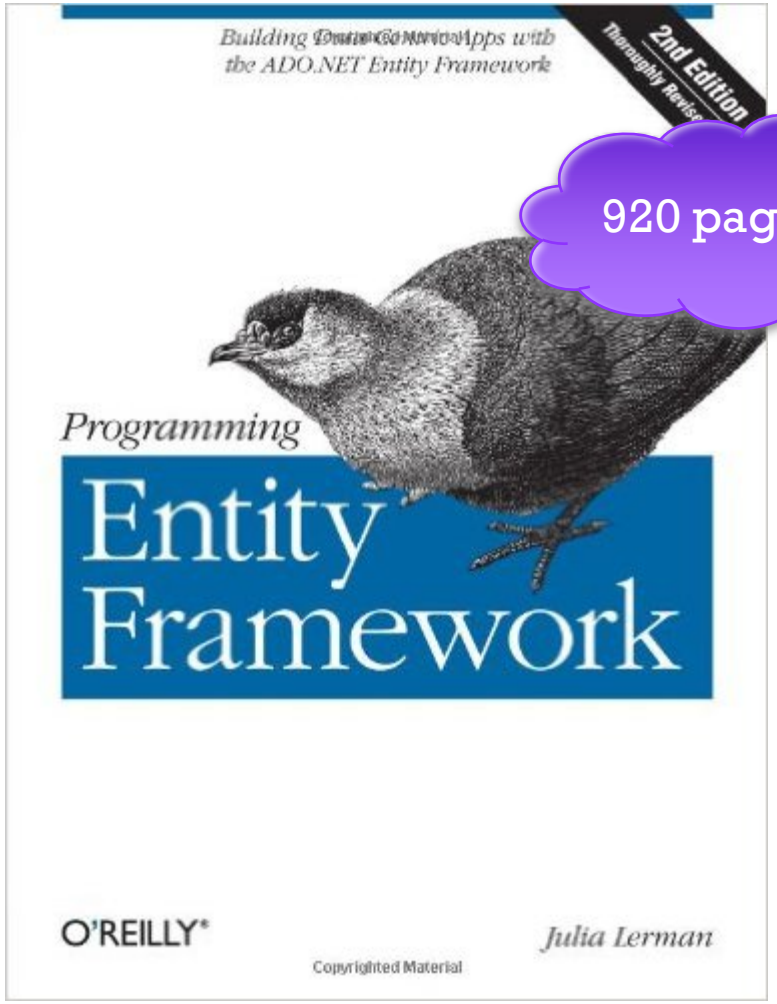
Expanded feature set: Units of measure

```
[<Measure>] type F // degrees Fahrenheit
[<Measure>] type C // degrees Celsius
[<Measure>] type mi // miles
[<Measure>] type km // kilometers
[<Measure>] type hr // hour

let WindChill_US (T:float<F>) (v:float<mi/hr>) =
    35.74<F> + 0.6215 * T-35.75<F> * float(v) ** 0.16 + 0.3965 * T * float(v) ** 0.16

let WindChill_CA (T:float<C>) (v:float<km/hr>) =
    13.12<C> + 0.6215 * T-11.37<C> * float(v) ** 0.16 + 0.4275 * T * float(v) ** 0.16
```

Expanded feature set: Type providers



920 pages!

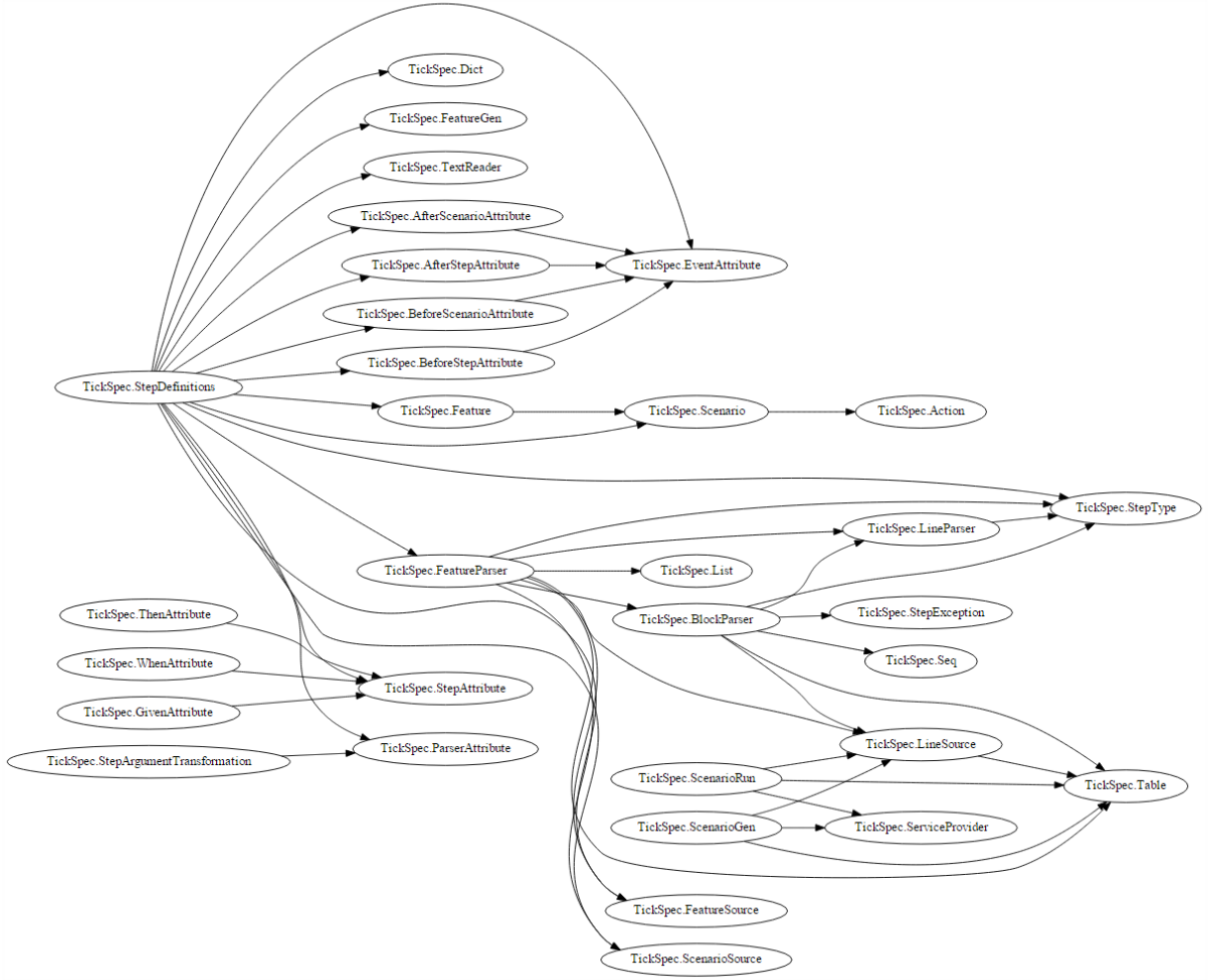
32 lines!

```
1   #r "FSharp.Data.TypeProviders"
2   #r "System.Data.Linq"
3   #r "System.Data.Entity"
4   #load @"..\packages\FSharp.Charting.0.90.9\FSharp.Charting.fsx"
5
6   open Microsoft.FSharp.Data.TypeProviders
7   open System.Data.Linq
8   open System.Data.Entity
9   open FSharp.Charting
10
11  // Sum & graph landing counts by airline -- all flights, for all time
12  module SqlDataConnectionSample =
13      type internal SFOData = SqlEntityConnection<ConnectionStringName = "SFO", ForceUpdate=true, Pl
14      let internal SFOContext = SFOData.GetDataContext()
15
16      let internal SFOInfo =
17          query { for data in SFOContext.RawLandingsData do
                    where (not (data.PublishedAirline.Contains("United Airlines"))) // again, numbers
                    groupValBy data.LandingCount data.PublishedAirline into g
                    let total = query { for group in g do sumBy group }
                    select (g.Key, total)
                    }
                    |> Chart.Column
25      // Add landing counts for a new airline.
26      let internal NewSFO =
27          SFOData.ServiceTypes.RawLandingsData.CreateRawLandingsData(0, 201501., "RachelsAirline", "
              "January")
28
29
30      SFOContext.DataContext.AddObject("RawLandingsData", NewSFO)
31      SFOContext.DataContext.SaveChanges()
32
```

Why F#?

Dependencies

TickSpec – an F# project

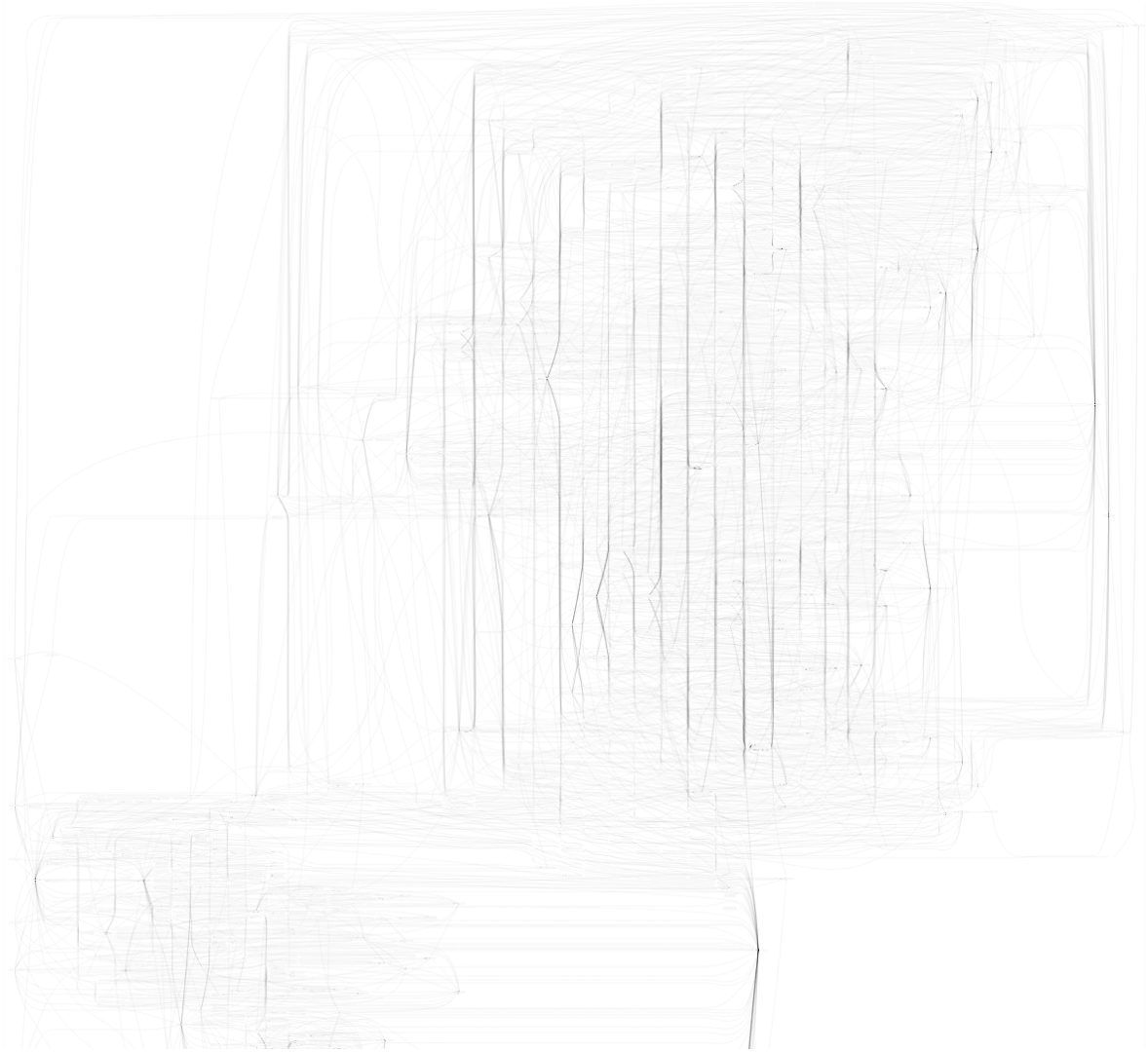


Thanks to Scott Wlaschin for his post, Cycles and modularity in the wild

SpecFlow– a comparable C# project

Thanks to Scott Wlaschin for his post, Cycles and modularity in the wild

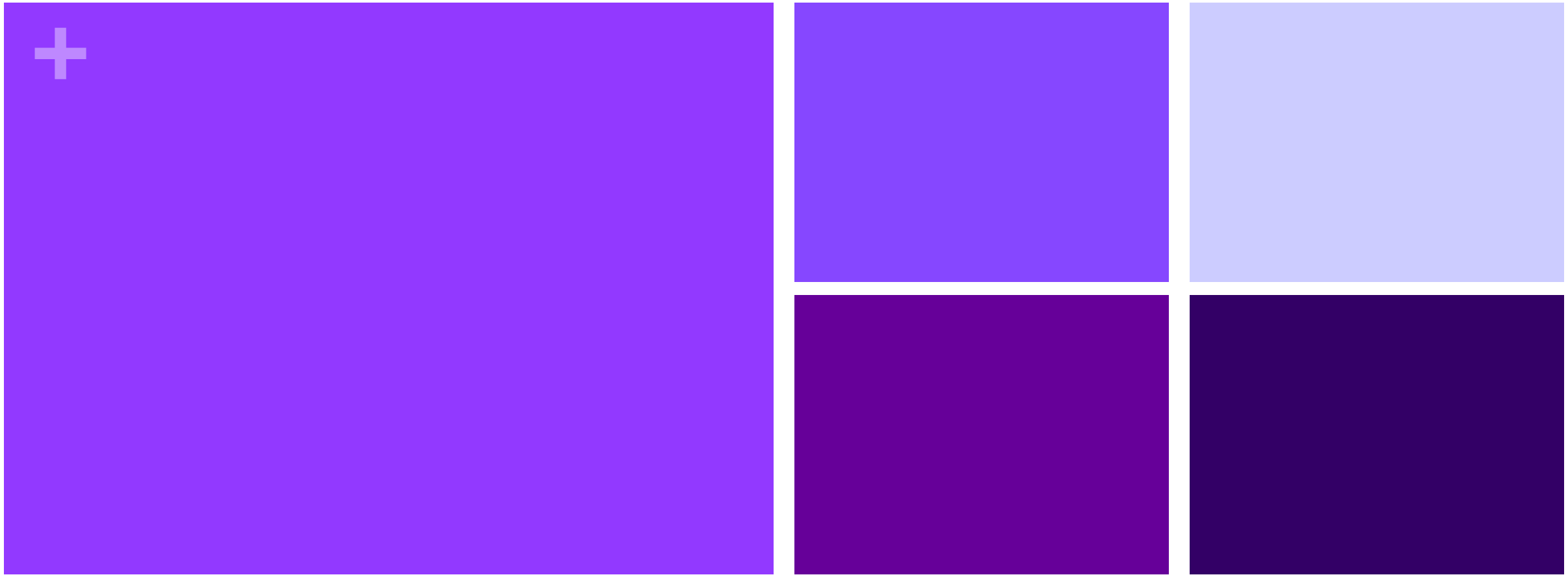
Entity Framework (C#)



Thanks to Scott Wlaschin for his post, Cycles and modularity in the wild

General resources & additional reading

- General resources
  - Me!
  - Team leads
  - Slack - #fsharp channel
  - F# chat on SO
  - http://fsharp.org/
  - Twitter: #fsharp
  - F# channel on Functional Programming Slack
- Additional reading
  - F# for Fun and Profit
  - F# Weekly
  - Try F#



Jet loves F#. Here's why!

Rachel Reese | @rachelreese