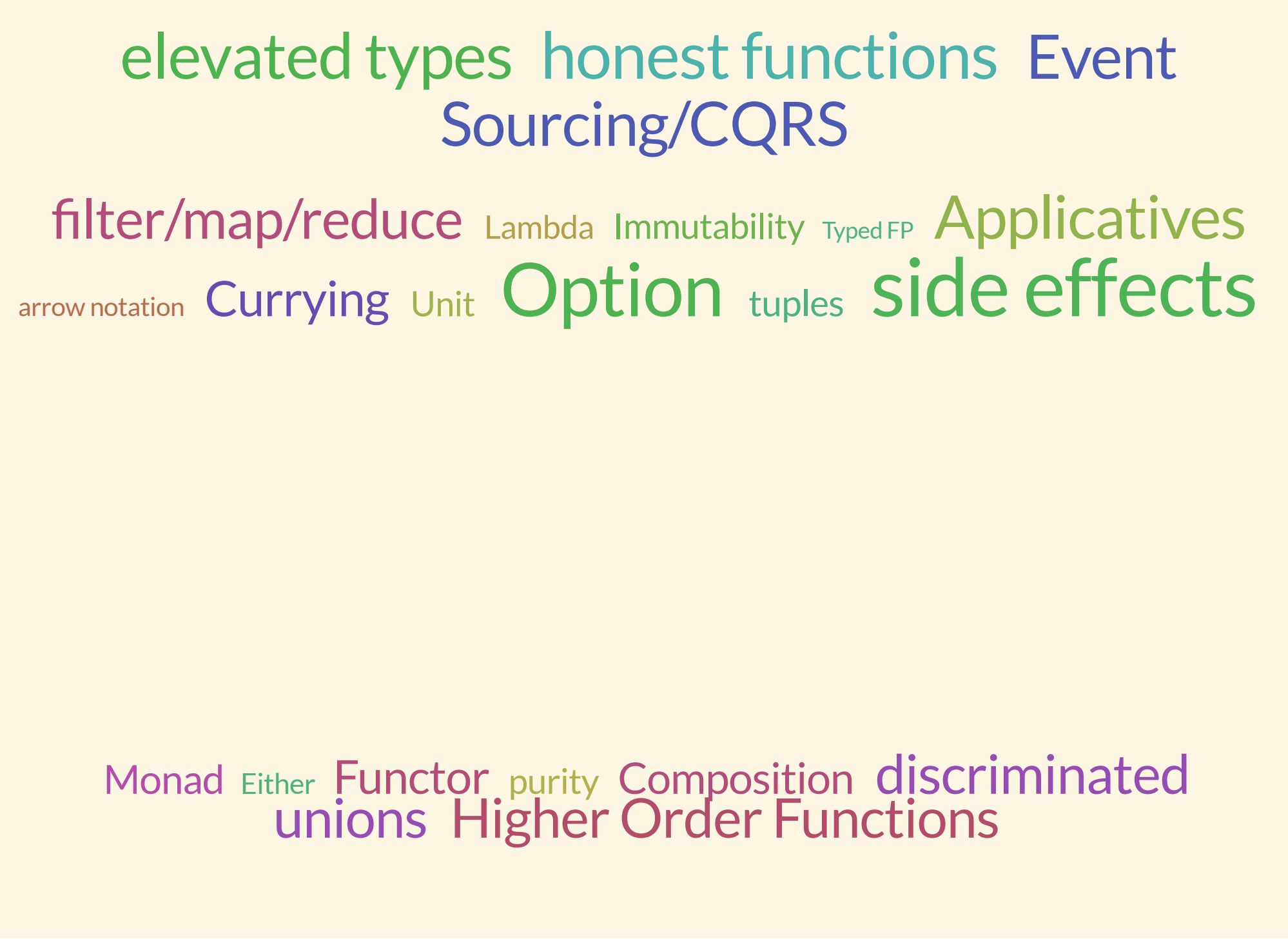
# FP FOR OO DEVELOPERS

Patrick Drechsler

Softwerkskammer Nürnberg 1.2.2018

# WAS IST FUNKTIONALE PROGRAMMIERUNG?

- Sprachunabhängig
- **Nur ein Paradigma!**
  - andere Paradigmen:
    - Prozedural
    - Objektorientiert
    - Logisch


elevated types
honest functions
Event Sourcing/CQRS
filter/map/reduce
Lambda
Immutability
Typed FP
Applicatives
arrow notation
Currying
Unit
Option
tuples
side effects
Monad
Either
Functor
purity
Composition
discriminated unions
Higher Order Functions

Monoid bind

# FP KONZEPTE

# IMMUTABILITY

- Lambdas: Sprachfeatures verwenden (LINQ, Streaming API)
- Value Objects ("fight primitive obsession")

# das ist ok:

```
let list = [1, 2, 3, 4, 5];
for (let i = 0; i < list.length; i++) {
    list[i] = list[i] + 1;
}
console.log(list)
```

# ...aber das einfacher:

```
let list = [1, 2, 3, 4, 5];
let result = list.map(x -> x + 1); // oder eine "addOne" Funktion nehmen
console.log(list)
```

# ok...

```
public Risk CheckRisk(int age)
{
    if (age <= 0) { /* error handling */ }
    else if (age > 120) { /* error handling */ }
    else if (age < 20) { return Risk.Low }
    else if (age < 40) { return Risk.Medium }
    else { return Risk.High }
}
```

# ...weniger "Krach":

```
public Risk CheckRisk(Age age)
{
    if (age < 20) { return Risk.Low }
    else if (age < 40) { return Risk.Medium }
    else { return Risk.High }
}
```

# MEHR RECHTE FÜR FUNKTIONEN!

- Expressions statt Statements
- Higher Order Functions: Methoden können auch Funktionen zurückgeben
  - → Currying/Applicative Functions

# EXPRESSIONS

```
// statement
public int AddOne(int i)
{
    return i + 1;
}
```

```
// expression
public int AddOne(int i) => i + 1;
```

# HIGHER ORDER FUNCTIONS

```
Func<int, bool> IsDivisibleBy(int divisor) => number
    => number % divisor == 0;

var isDivisibleByFive = IsDivisibleBy(5);
isDivisibleByFive(10); // TRUE
```

# COMPOSITION

- Funktionen miteinander kombinieren (Alternative zu Ableitung in OO)
  - z.B. Method Chaining (LINQ)
  - → kann IoC ersetzen

# COMPOSITION

```
Func<int, bool> isLargerThanFive = x => x > 5;
Func<int, bool> isSmallerThenTen = x => x < 10;

Func<int, bool> isBetweenFiveAndTen = x =>
    isLargerThanFive(x) && isSmallerThenTen(x);

isBetweenFiveAndTen(7).Should().BeTrue();
```

# COMPOSITION

```
static string Abbreviate(string s) => s.SubString(0, 2).ToLower();

static string AbbreviateName(Person p)
    => Abbreviate(p.FirstName) + Abbreviate(p.LastName);

static string AppendDomain(string localPart)
    => $"{localPart}@company.com";

// composition
Func<Person, string> emailFor = p => AppendDomain(AbbreviateName(p));

var joe = new Person("Joe", "Smith")
emailFor(joe).Should().Be("josm@company.com");
```

```
// method chaining (using C# Extensions)
static string AbbreviateName(this Person p)
    => Abbreviate(p.FirstName) + Abbreviate(p.LastName);

static string AppendDomain(this string localPart)
    => $"{localPart}@company.com";

joe.AbbreviateName().AppendDomain().Should().Be("josm@company.com");
```

# SAFETY THROUGH TYPES

- Stärkeres Typsystem kann Entwicklung erleichtern
  - Discriminated Union
  - Wrapper wie Option, Either, etc

# TYPSYSTEM

```
public Option<Customer> GetCustomer(int id) { /* ... */ }

public string Greet(int id)
    => GetCustomer(id).Match(
           None: () => "Sorry, who?",
           Some: (customer) => $"Hello, {customer.Name}");
```

# TYPSYSTEM (BSP. F#)

```
type AccountStatus = // discriminated union
    Requested | Active | Frozen | Dormant | Closed

type CurrencyCode = string // "type alias"

type Transaction = { // record type
    Amount: decimal
    Description: string
    Date: DateTime
}

type AccountState = {
    Status: AccountStatus
    Currency: CurrencyCode
    AllowedOverdraft: decimal
    TransactionHistory: Transaction list
}

type AccountState with
member this.WithStatus(status) = { this with Status = Active }
member this.Add(transaction) =
    { this with TransactionHistory =
        transaction :: this.TransactionHistory }
```

# ZUSAMMENFASSUNG

- Immutability
- Expressions
- HOF
- Composition
- Typsystem

# DANKE!

Kontaktinfos:

- @drechsler
- socialcoding@pdrechsler.de