



RAILWAY ORIENTED PROGRAMMING

KOMPLEXE ORCHESTRIERUNG WARTBAR MACHEN

Patrick Drechsler

Zur Person:

- "gelernter" Biologe
- Softwareentwickler bei Redheads Ltd
- .NET, JS
- aktuelle Schwerpunkte: DDD, CQRS
- Softwerkskammer
-  @drechsler
-  patrick.drechsler@redheads.de

DISCLAIMER

Ich werde nicht erklären, was eine Monade ist.

Wenn man verstanden hat, was eine Monade ist, verliert man die Fähigkeit zu erklären, was eine Monade ist.

(Monaden-Paradoxon)

Alle Beispiele sind in **C#**
(geht aber auch alles mit Java)

Umfrage:

- Java?
- C#?

ORCHESTRIERUNG

Stelle im Code, die

- selbst wenig interne Logik hat
- viele andere Module/Klassen benötigt
- z.B. den Ablauf einer User Story beschreibt
- oft in "Service" Klasse (z.B. RegistrationService)

"2 Zeilen Code"

```
var customerResult = Validate(createCustomerViewModel);  
var result = customerResult  
    .OnSuccess(c => _customerRepository.Create(c))  
    .OnSuccess(c => _mailConfirmer.SendWelcome(c))  
    .OnBoth(cResultAtEnd => cResultAtEnd.IsSuccess  
        ? new CustomerCreatedViewModel(cResultAtEnd.Value.Id)  
        : CreateErrorResponse(cResultAtEnd.Error));
```

"1 Zeile Code"

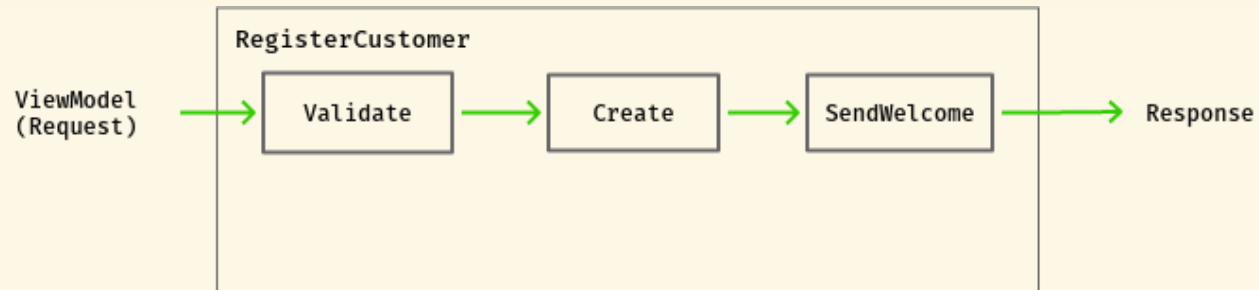
Realistischeres Beispiel

```
var result = Validate(createCustomerViewModel)
    .OnSuccess(c => _customerRepository.Create(c))
    .OnSuccess(c => createCustomerViewModel.WantsPremiumSupport
        ? _creditCardGateway.Charge(c.CreditCardNumber)
            .OnSuccess(gateway => gateway.ChargeWasBooked
                ? _customerRepository.UpgradeToPremium(c)
                : _creditCardGateway.RollbackLastTransaction(c))
        : Result.Ok(c))
    .OnSuccess(c => _mailConfirmer.SendWelcome(c))
    .OnBoth(cResultAtEnd => cResultAtEnd.IsSuccess
        ? new CustomerCreatedViewModel(cResultAtEnd.Value.Id)
        : CreateErrorResponse(cResultAtEnd.Error));
```

USER STORY: ANMELDUNG ALS NEUER BENUTZER

Wenn ein neuer Benutzer sich anmeldet,

- werden seine Eingaben validiert
- wird er im System gespeichert
- erhält er eine Bestätigungsmail



```
public CustomerCreatedViewModel RegisterCustomer(SomeVM viewModel)
{
    var customer = Validate(viewModel);
    customer = _customerRepository.Create(customer);
    _mailConfirmer.SendWelcome(customer);

    return new CustomerCreatedViewModel(customer);
}
```

- LIVE-CODING
- Cool, wir sind fertig!
- let's go live...

NO ERROR HANDLING:

**WHAT COULD POSSIBLY
GO WRONG?**

...potentielle Fallstricke...

```
// can fail
var customer = Validate(createCustomerViewModel);

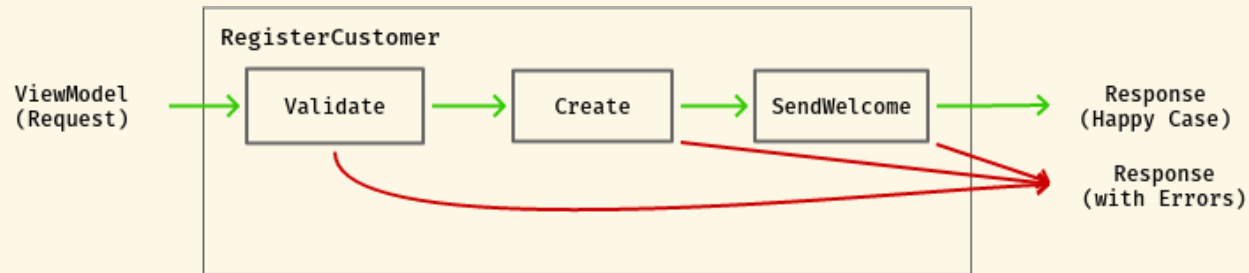
// can fail
customer = _customerRepository.Create(customer);

// can fail
_mailConfirmer.SendWelcome(customer);

return new CustomerCreatedViewModel(customer.Id) {Success = ??};
```

Nicht einfach drauflos programmieren: Zuerst mit
Kunde/Domain-Experten klären!

Dann die User Story aktualisieren (oder neue User
Story für erstellen)



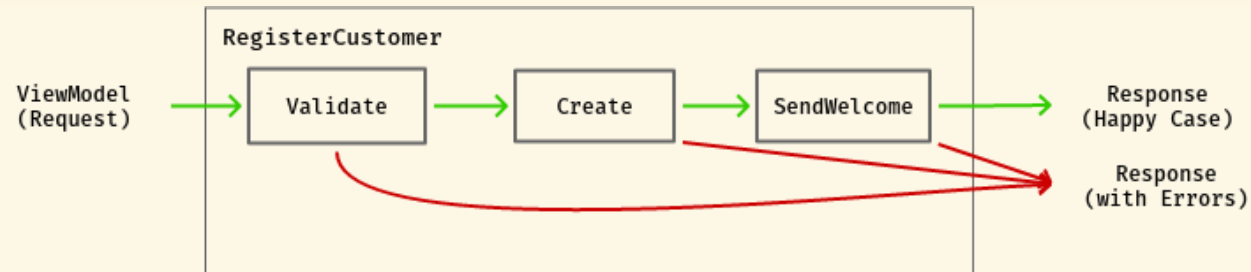
```
Customer customer;
try { customer = Validate(createCustomerViewModel); }
catch (Exception e) { return CreateErrorResponse(e); }

try { customer = _customerRepository.Create(customer); }
catch (Exception e) { return CreateErrorResponse(e); }

try { _mailConfirmer.SendWelcome(customer); }
catch (Exception e)
{
    // don't fail, but maybe: logging, retry-policy
}

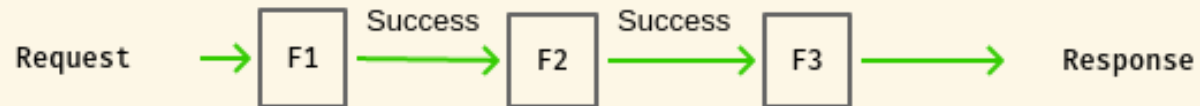
return new CustomerCreatedViewModel(customer.Id);
```

LIVE CODING

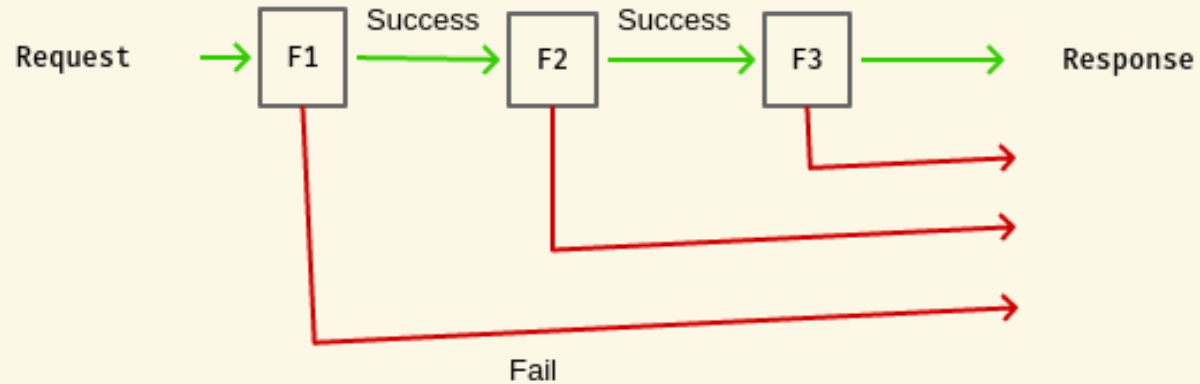


- Fehlerbehandlung macht einen Großteil des Codes aus
- Ergebnis einer vorherigen Aktion ist Grundlage für weiteres Vorgehen

...happy case...



...mit Fehlerbehandlung...



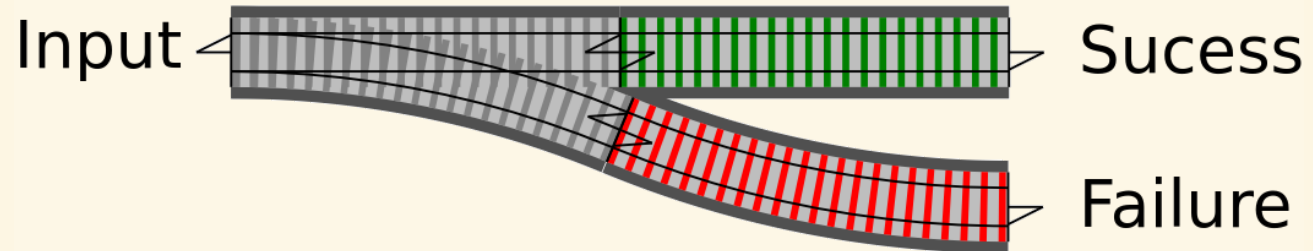
FUNKTIONALE PROGRAMMIERUNG

Wer möchte heute noch auf Lambdas verzichten?

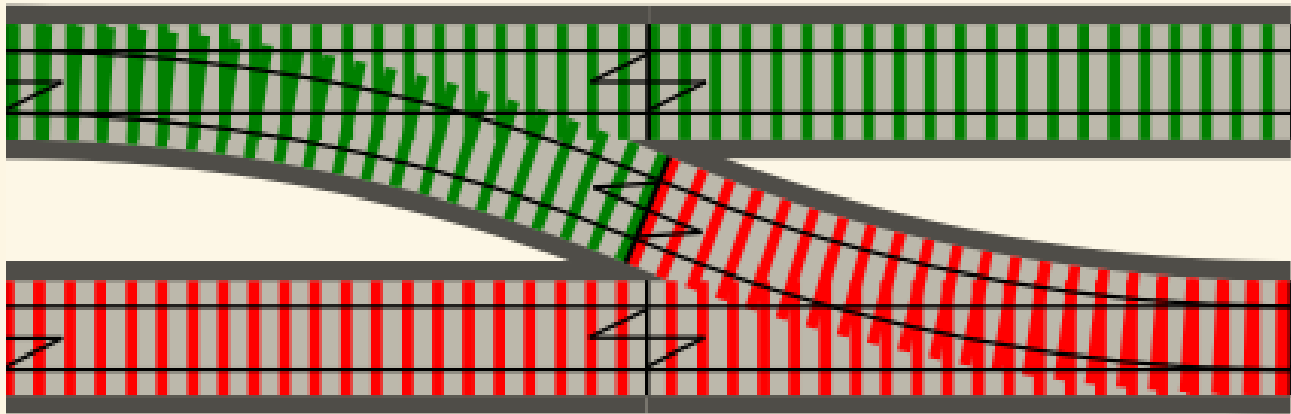
- Was unterscheidet Funktionen von Methoden?
 - entspricht eher einer math. Funktion (und nicht einer Eigenschaft einer Klasse)
 - gleiche Eingabe gibt immer gleiches Ergebnis zurück
 - (idealerweise ohne Seiteneffekte)
- Funktionen können als Eingabe- und Rückgabewert verwendet werden ("higher order functions")

WHERE ARE THE TRAINS?
YOU PROMISED RAILWAYS!

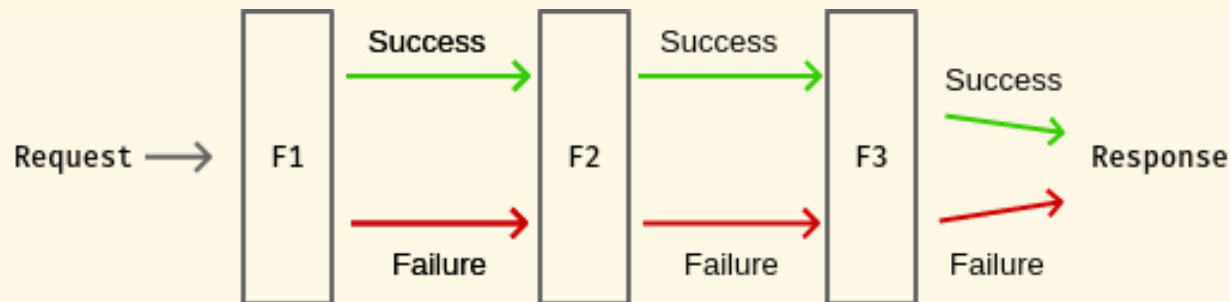
"Weiche"



(...endlich die Railway Metapher...)



Machen wir unser Bildchen "funktionaler"

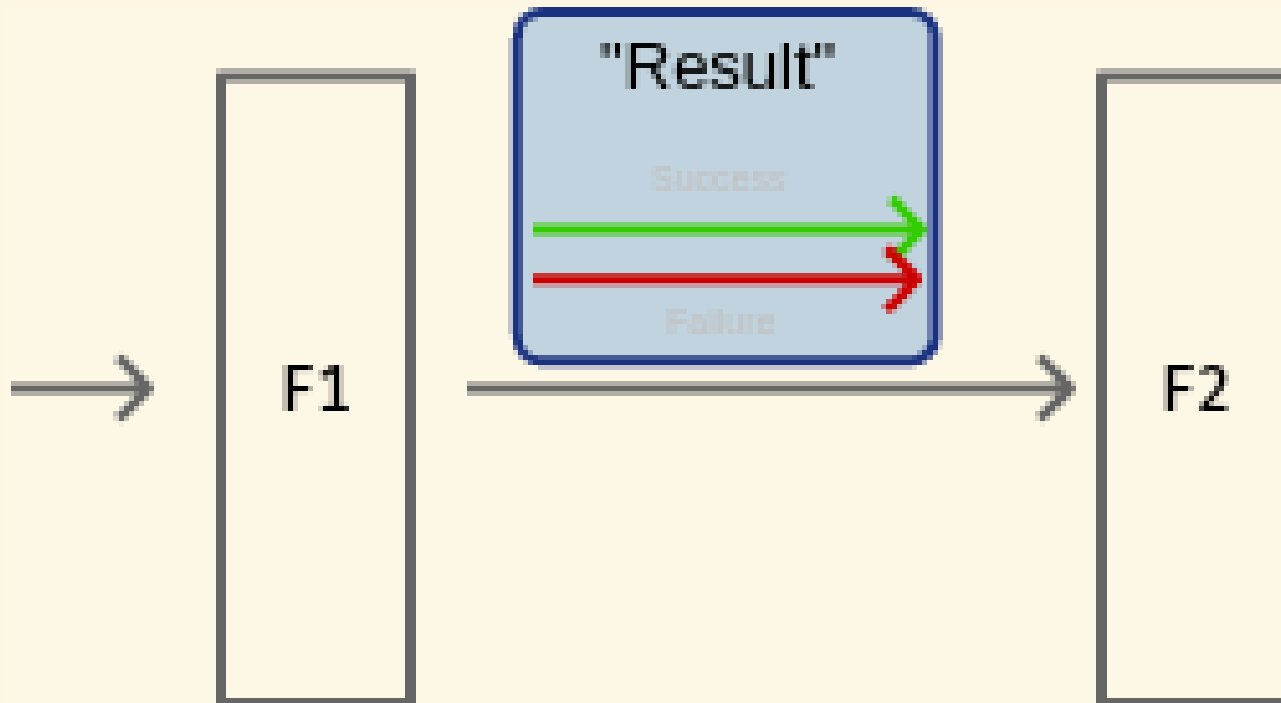


- F1: wie gehabt
- F2 und F3: können Fehler empfangen (cool)
- F2 und F3: austauschbar (sehr cool)

F1: wirklich wie gehabt? (1: Eingang, 2 Ausgänge)

Nein!

Eigentlich wollen wir das Ergebnis kapseln (damit wir es später verketteten können)



F2: kann Fehler empfangen (2 Eingänge, 2 Ausgänge)

- wir müssen ein "Result" als Eingabe verarbeiten können

Was hat es mit dieser "Result" Klasse auf sich?

"Result" basiert auf "Maybe"...

OBJEKTORIENTIERUNG UND NULL

Tony Hoare introduced Null references in ALGOL W back in 1965 "simply because it was so easy to implement", says Mr. Hoare. He talks about that decision considering it "my billion-dollar mistake".

<https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare>

darum schreiben wir...

```
Customer customer = repo.GetById(42);  
if (customer == null) {  
    // error handling  
    // is it NULL because  
    // - no result was found  
    // - or for another reason?  
}  
else {  
    // happy case  
}
```

darum brauchen wir...

```
Maybe<Customer> customer = repo.GetById(42);  
if (customer.HasValue) {  
    // happy case  
    DoSomething(customer.Value.FirstName);  
}  
else {  
    // error handling  
}
```

(zumindest ist klar, was gemeint ist)

C# VS JAVA

- Für C# spricht:
 - Lernt oft von Java
 - weniger Boilerplate (Properties vs getter/setter)
 - Lambdas
 - Pattern Matching (seit C# 7)

Aber Java holt wieder auf: Java hat eine Option Klasse!

In C# und in Java gibt es aktuell keine "Result" Klasse



- C#: CSharpFunctionalExtensions (NuGet)
- Java: auch möglich (Link im Abspann)



C# API

NuGet: CSharpFunctionalExtensions

RESULT<T> ERSTELLEN

```
Result<Customer> r1 = Result.Ok(customer);  
Result<Customer> r2 = Result.Fail<Customer>("Something failed");
```

(ja, Failure ist vom Typ String)

Result<T> als Rückgabewert

```
public Result<Customer> CreateCustomer(...)
{
    var customer = ...;
    if (IsValid(customer))
    {
        return Result.Ok(customer);
    }
    else
    {
        return Result.Fail<Customer>("Validation failed");
    }
}
```


KOMBINATION VON RESULTS

(via Extension Methods)

- OnSuccess
- OnBoth
- OnFailure

Extension Methods in C#

sind sowas wie "traits" (Scala) oder "mixins" (Ruby)

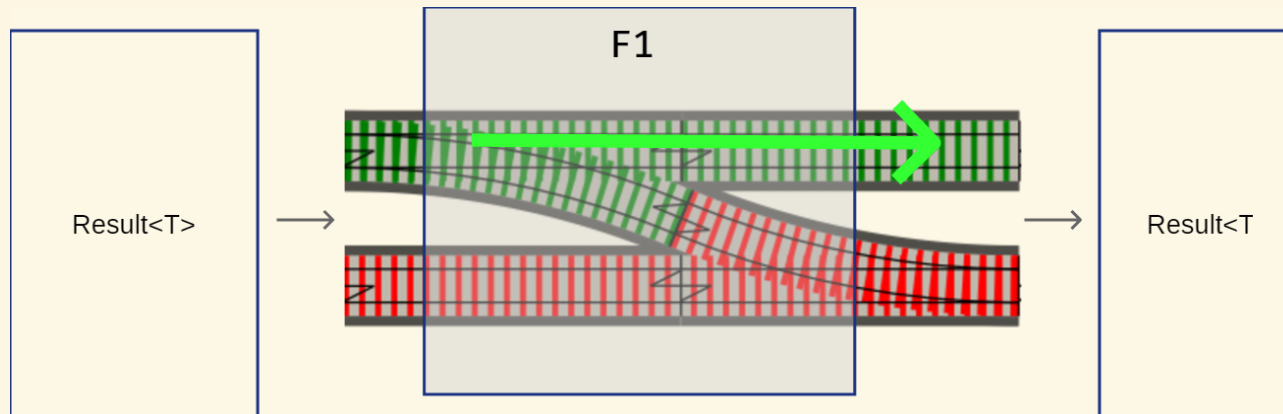
API OnSuccess:

```
public static Result OnSuccess(this Result result, Func<Result> func)
```

```
public static Result<T> OnSuccess<T>(this Result result, Func<T> func)
```

```
public static Result<K> OnSuccess<T, K>(this Result<T> result,  
                                         Func<Result<K>> func)
```

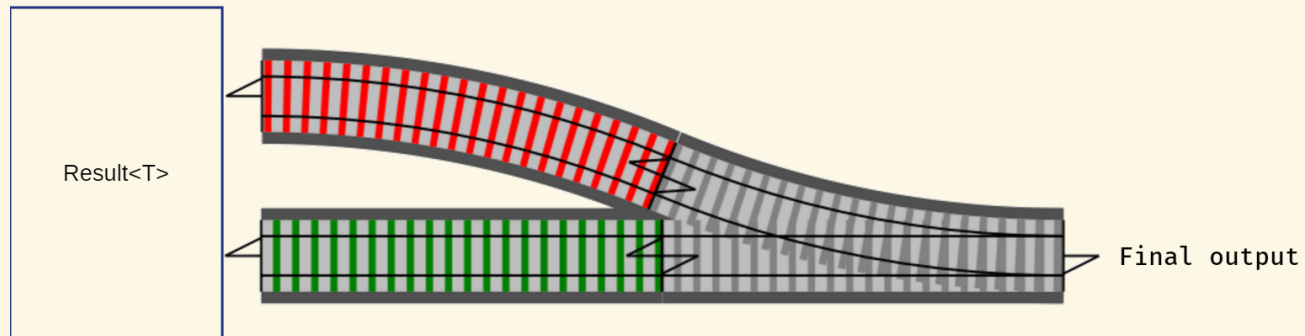
```
//...
```



API OnBoth:

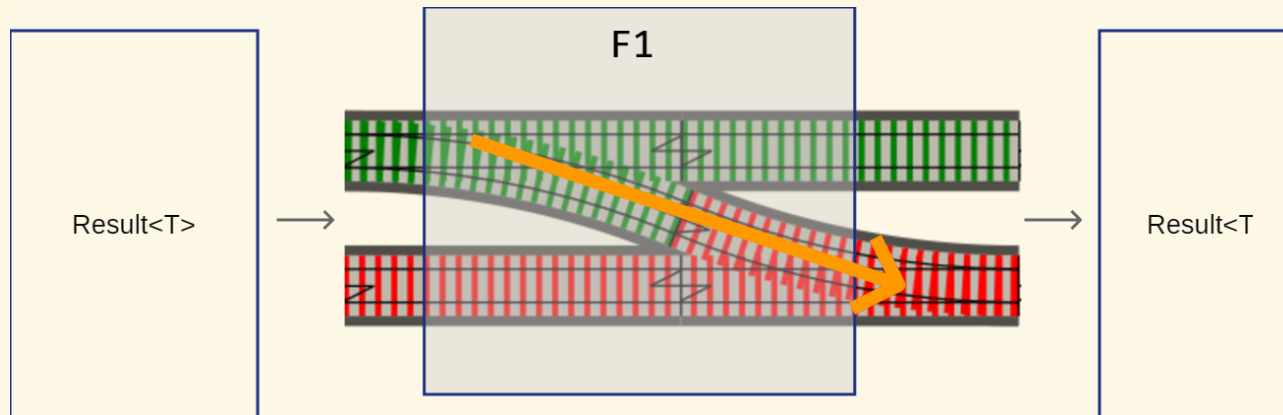
```
public static T OnBoth<T>(this Result result, Func<Result, T> func)
```

```
public static K OnBoth<T, K>(this Result<T> result,  
                             Func<Result<T>, K> func)
```



API OnFailure:

```
public static Result OnFailure(this Result result, Action action)  
  
public static Result<T> OnFailure<T>(this Result<T> result, Action<string>  
//...
```





LIVE CODING

LINKS

- Scott Wlaschin "the original talk"
<http://fsharpforfunandprofit.com/rop/>
- Stefan Macke "ROP für Java"
<https://www.heise.de/developer/artikel/Railway-Oriented-Programming-in-Java-3598438.html>
- Vladimir Khorikov "Functional C#: Handling failures..."
<http://enterprisecraftsmanship.com/2015/03/20/functional-c-handling-failures-input-errors/>
- CSharpFunctionalExtensions
<https://github.com/vkhorikov/CSharpFunctionalExtensions>

FRAGEN?

Kontaktinfos:

-  @drechsler
- 
patrick.drechsler@redheads.de