embarc
Software Consulting GmbH

JVM Functional Language Battle

FALK SIPPACH // EMBARC

Developer Week 2021

Dienstag, 29. Juni 2021, 18:00 Uhr

DWX DEVELOPER WEEK '21

JVM Functional Language Battle

Zusammenfassung

Funktionale Programmierung soll so viel ausdrucksstärker sein, aber leider ist dieses Programmier-Paradigma nicht ganz kompatibel zu der prozedural- und objektorientierten Denkweise von uns Java-Entwicklern. Anhand eines kleinen Algorithmus werden wir uns verschiedene Lösungen zunächst im klassischem imperativen Java (vor Java 8) und als Vergleich dazu in alternativen JVM-Sprachen (Groovy, Frege, ggf. Scala bzw. JavaScript) anschauen und die verschiedenen Lösungen diskutieren.

Herauskommen soll eine saubere und verständlichere Struktur, die zu besser les- und wartbarem Code führen wird. Die gewonnenen Erkenntnisse wollen wir dann letztendlich in Java 8 mittels Streams und Lambda-Ausdrücken umsetzen, so dass jeder Zuhörer die Grundideen der funktionalen Programmierung mit in seine tägliche Arbeit nehmen kann. Es sind keine speziellen Vorkenntnisse in den angesprochenen alternativen Sprachen notwendig, ein solides Verständnis für die Programmiersprache Java genügt.

Falk Sippach

- Softwarearchitekt, Berater, Trainer bei embarc
- früher bei Orientation in Objects (OIO), Trivadis

Schwerpunkte:

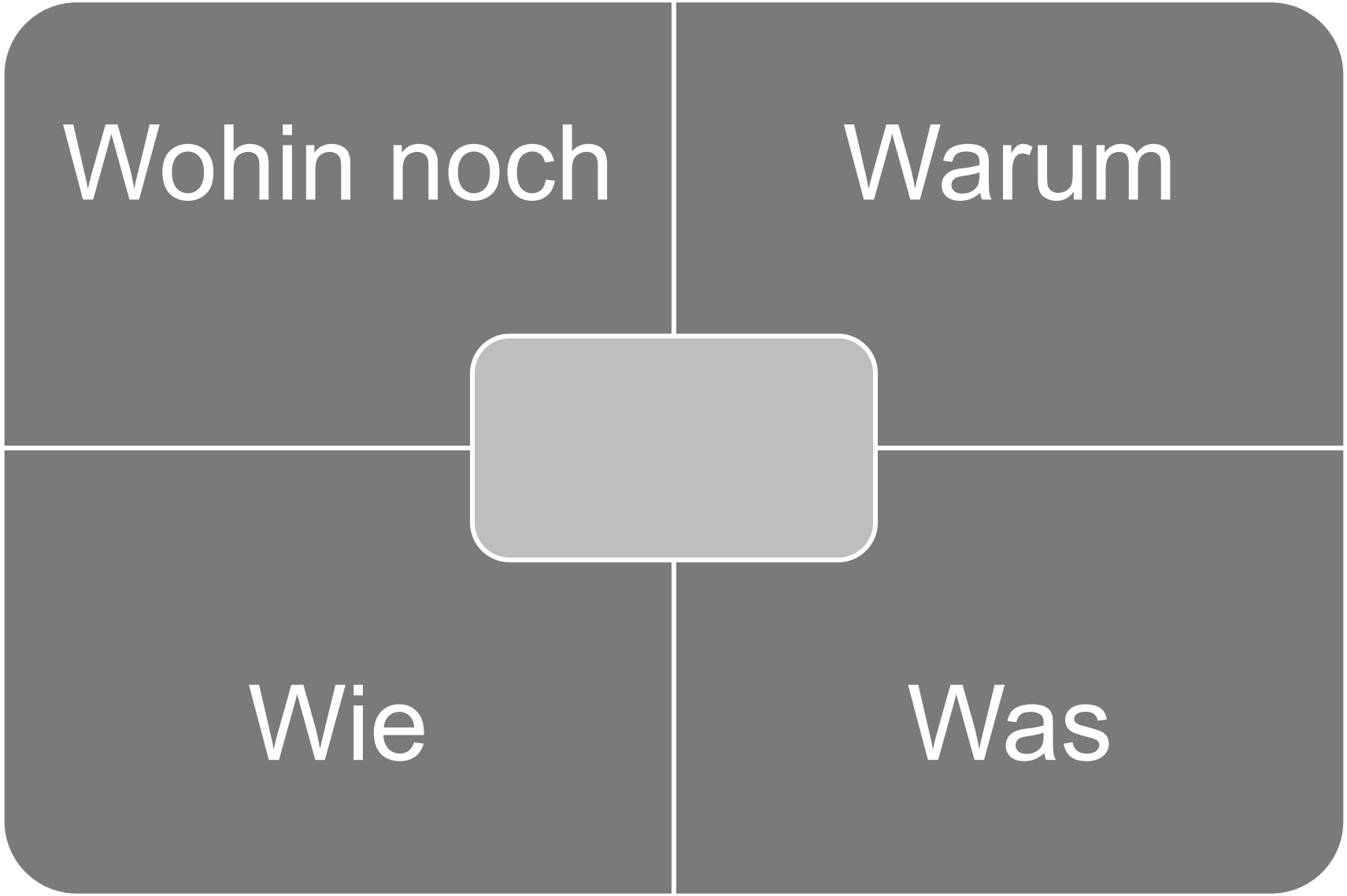
- Architekturberatung und -bewertung
- Cloud- und Java-Technologien

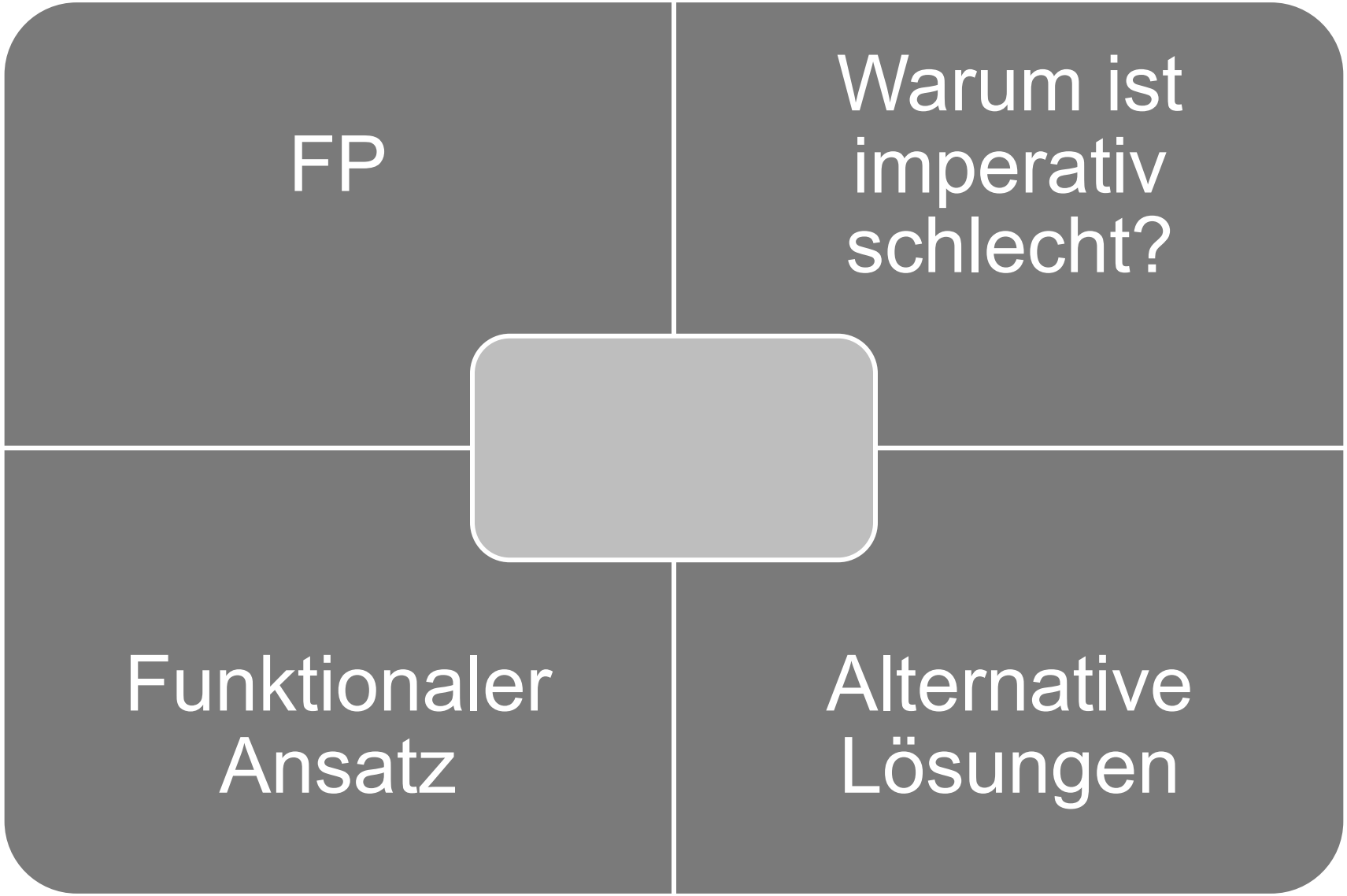
fs@embarc.de

@sippsack

➔ xing.to/fsi

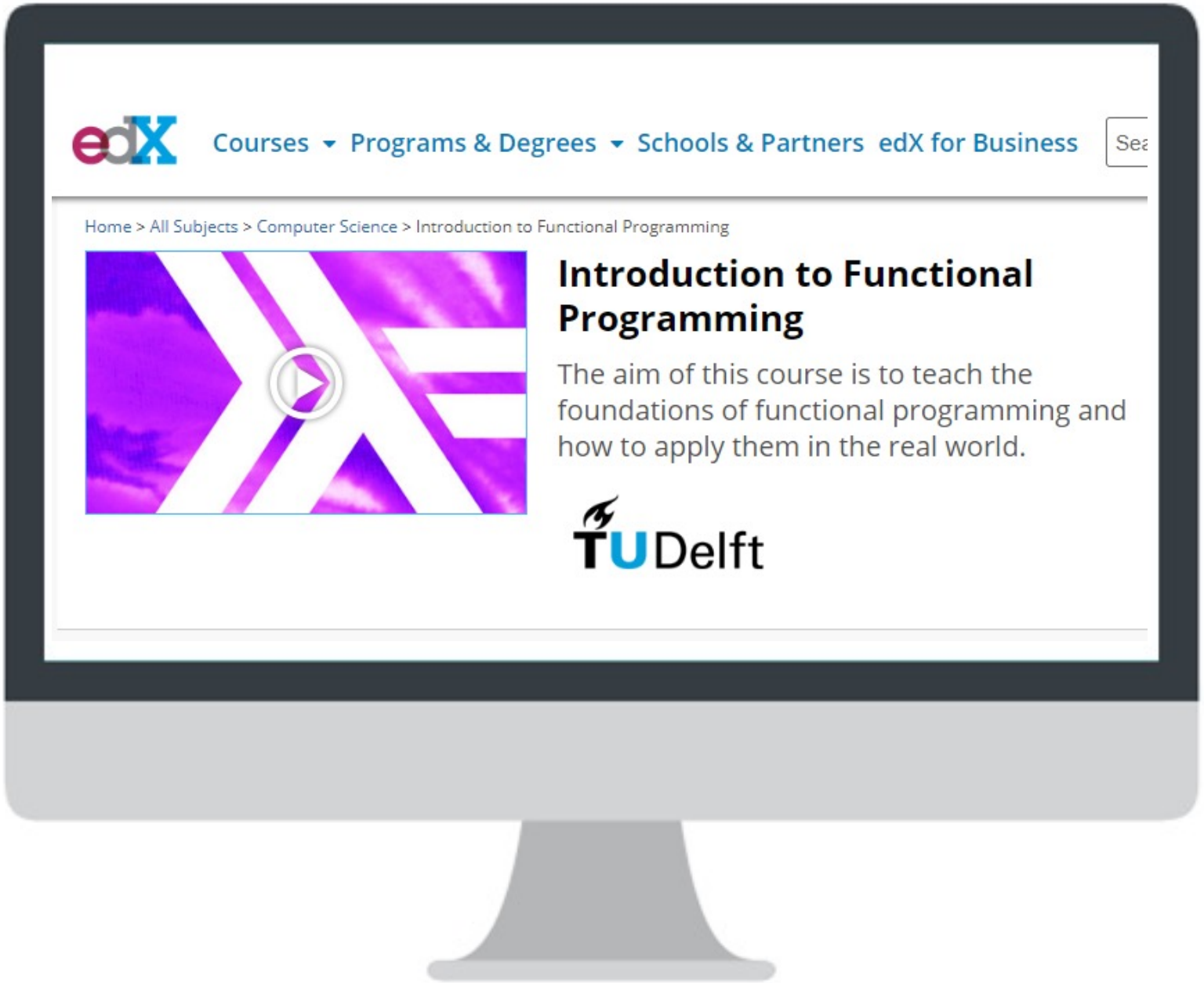
embarc
Software Consulting GmbH

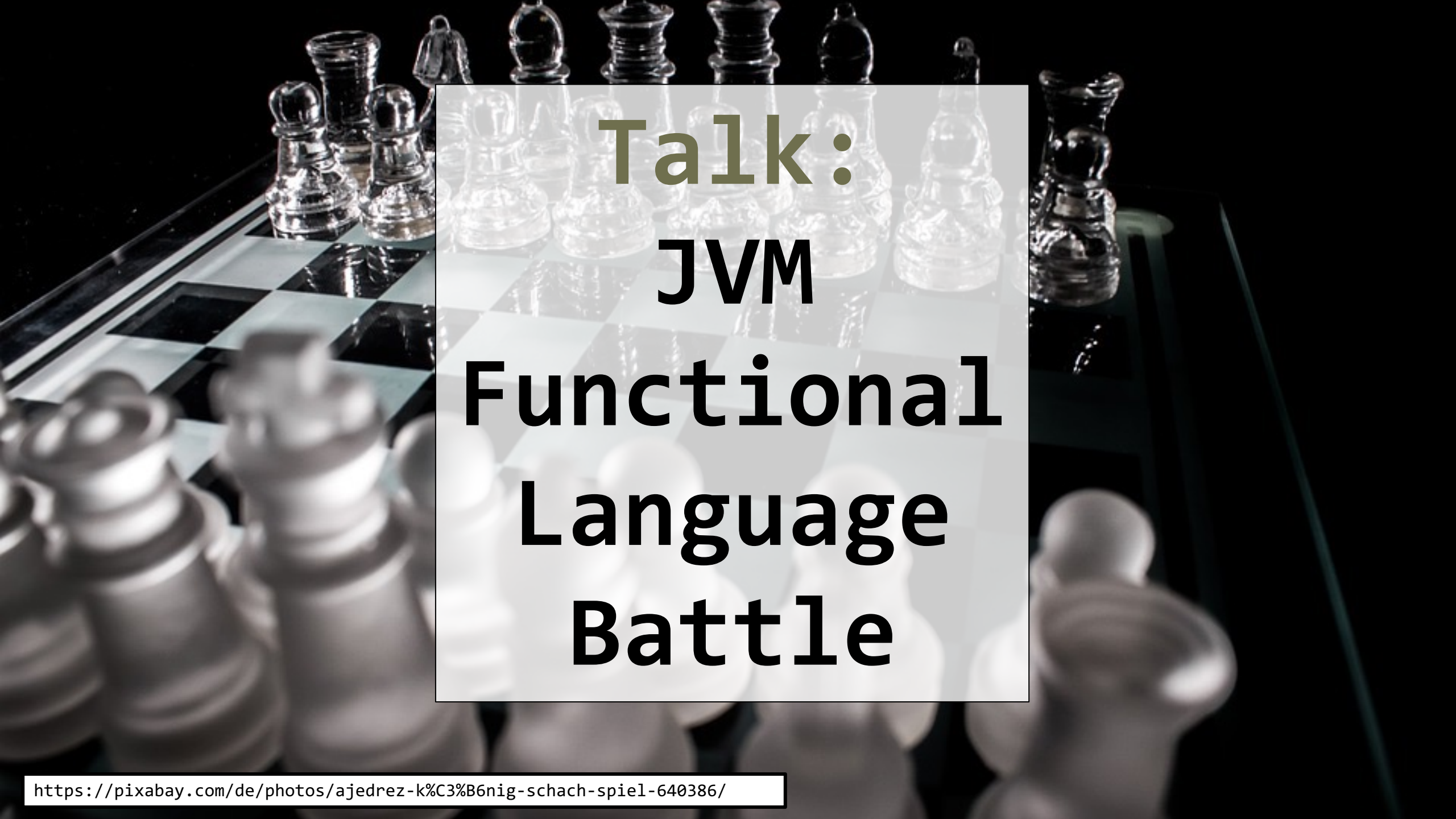

Wohin noch
Warum
Wie
Was


FP
Warum ist imperativ schlecht?
Funktionaler Ansatz
Alternative Lösungen


Groovy


Softwerkskammer
Software Craftsmanship Communities
Überblick
Aktivitäten
Gruppen
Wiki
Lerngruppe Funktionale Programmierung
Themengruppe
Adresse: lernfp@softwerkskammer.org
Wir interessieren uns für Funktionale Programmierung und wollen uns gegenseitig soviel FP wie möglich beibringen. Wer Funktionale Programmierung lernen möchte, oder wer Funktionale Programmierung kann und sich berufen fühlt sein Wissen zu teilen sollte hier mal reinschauen :)
Mitglieder:
Diese Gruppe hat 182 Mitglieder.


edX
Courses
Programs & Degrees
Schools & Partners
edX for Business
Home > All Subjects > Computer Science > Introduction to Functional Programming
Introduction to Functional Programming
The aim of this course is to teach the foundations of functional programming and how to apply them in the real world.
TUDelft



Talk: JVM Functional Language Battle

https://pixabay.com/de/photos/ajedrez-k%C3%B6nig-schach-spiel-640386/

Agenda

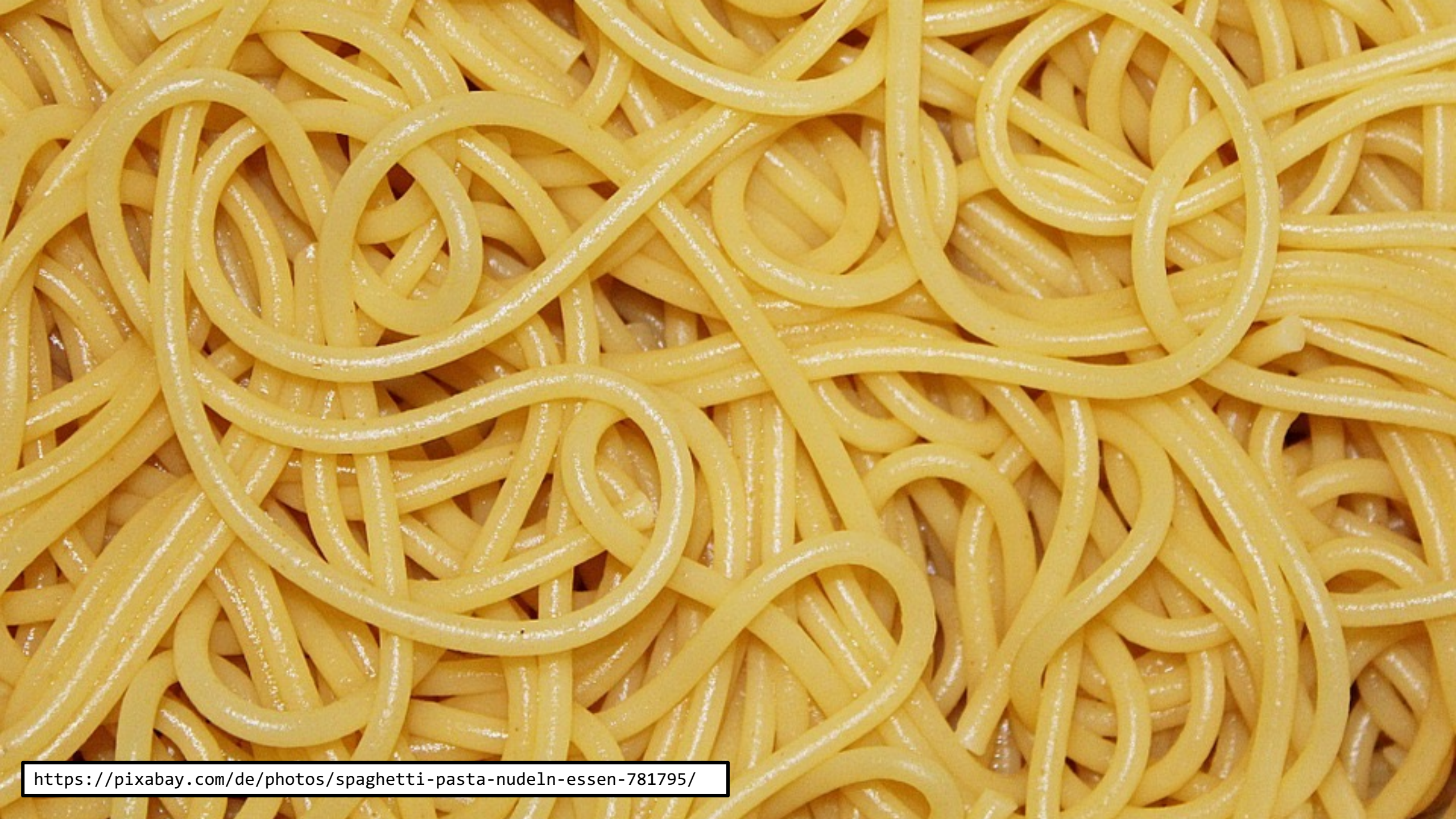
1. Probleme mit imperativer Programmierung?
2. Let’s battle …
3. Lösung in funktionaler Programmierung
4. Fazit und Ausblick

Agenda

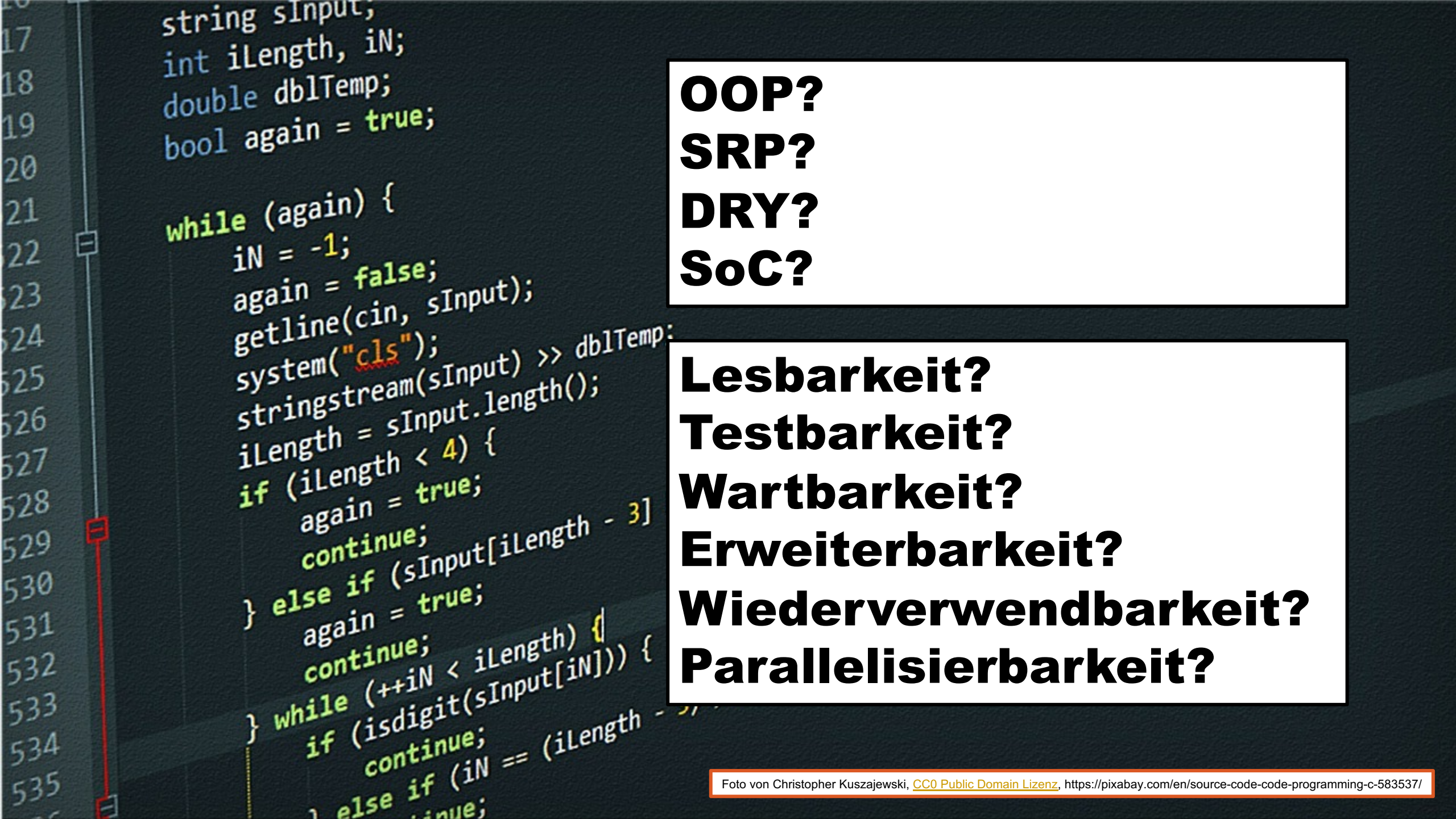
1. **Probleme mit imperativer Programmierung?**
2. Let’s battle …
3. Lösung in funktionaler Programmierung
4. Fazit und Ausblick

1

```
3  public class FooAlgorithm{
4      public static boolean isValid(long number) {
5          int sum = 0;
6          boolean alternate = false;
7          while(number > 0) {
8              long digit = number % 10;
9              if (alternate) {
10                 sum += 2 * digit;
11                 if (digit >= 5) {
12                     sum -= 9;
13                 }
14             } else {
15                 sum += digit;
16             }
17             number = number / 10;
18             alternate = !alternate;
19         }
20         return sum % 10 == 0;
21     }
22 }
```



https://pixabay.com/de/photos/spaghetti-pasta-nudeln-essen-781795/



```
3  public class FooAlgorithm{
4      public static boolean isValid(long number) {
5          int sum = 0;
6          boolean alternate = false;
7          while(number > 0) {
8              long digit = number % 10;
9              if (alternate) {
10                 sum += 2 * digit;
11                 if (digit >= 5) {
12                     sum -= 9;
13                 }
14             } else {
15                 sum += digit;
16             }
17             number = number / 10;
18             alternate = !alternate;
19         }
20         return sum % 10 == 0;
21     }
22 }
```

Prüfsummen-Berechnung

```
3  public class LuhnAlgorithm {
4      public static boolean isValid(long number) {
5          int sum = 0;
6          boolean alternate = false;
7          while(number > 0) {
8              long digit = number % 10;
9              if (alternate) {
10                 sum += 2 * digit;
11                 if (digit >= 5) {
12                     sum -= 9;
13                 }
14             } else {
15                 sum += digit;
16             }
17             number = number / 10;
18             alternate = !alternate;
19         }
20         return sum % 10 == 0;
21     }
22 }
```

Aufspalten in Ziffern

Jede zweite verdoppeln

Aufsummieren

Validierungsprüfung

	4716347184862961															
①	4	7	1	6	3	4	7	1	8	4	8	6	2	9	6	1
②	1	6	9	2	6	8	4	8	1	7	4	3	6	1	7	4
③	1	12	9	4	6	16	4	16	1	14	4	6	6	2	7	8
④	1	1 2	9	4	6	1 6	4	1 6	1	1 4	4	6	6	2	7	8
⑤	1	3	9	4	6	7	4	7	1	5	4	6	6	2	7	8
⑥	1 + 3 + 9 + 4 + 6 + 7 + 4 + 7 + 1 + 5 + 4 + 6 + 6 + 2 + 7 + 8															
	80															
⑦	80 % 10 == 0															
	gültig															

```java
public class LuhnAlgorithm {
    public static boolean isValid(long number) {
        int sum = 0;
        boolean alternate = false;
        while(number > 0) {
            long digit = number % 10;
            if (alternate) {
                sum += 2 * digit;
                if (digit >= 5) {
                    sum -= 9;
                }
            } else {
                sum += digit;
            }
            number = number / 10;
            alternate = !alternate;
        }
        return sum % 10 == 0;
    }
}
```

- Variablen
- Schleifen
- Verzweigungen
- Tiefe der Verschachtelung
- Zustands-änderungen
- Reihenfolge nicht beliebig
- Fehler-behandlung?

Der Kampf beginnt

https://pixabay.com/de/photos/ajedrez-k%C3%B6nig-schach-spiel-640386/

Agenda

1. Probleme mit imperativer Programmierung?
2. **Let’s battle …**
3. Lösung in funktionaler Programmierung
4. Fazit und Ausblick

2

JavaScript

objektorientierte Skriptsprache

aber klassenlos (Prototyp-basiert)

dynamisch typisiert

Funktionen sind First Class Citizens

geeignet für Client (Browser) und Server (Node.js, Nashorn)

Beispiel JavaScript

```
1   console.log(isValid(3782822463100005)) // true
2   console.log(isValid(76009244561)) // false
3
4   function isValid(number) {
5       let even = false
6       return number.toString()
7           .split('')
8           .reverse()
9           .map(c => parseInt(c))
10          .map(d => (even = !even) ? d : d < 5 ? d * 2 : (d - 5) * 2 + 1)
11          .reduce((a, b) => a + b, 0)
12          % 10 === 0
13  }
```

Kotlin

statisch typisiert

objektorientiert

Übersetzung in Java Bytecode oder JavaScript Quellcode

interoperabel mit Java

entwickelt bei Jetbrains (IntelliJ IDEA)

Kotlin

```kotlin
package de.oio.luhn

fun main(args: Array<String>) {
    println(isValid(378282246310005)) // true
    println(isValid(76009244561)) // false
}

fun isValid(number: Long): Boolean {
    var even = false
    return number.toString()
            .split("")
            .reversed()
            .filter { !it.isBlank() }
            .map { it.toInt() }
            .map { even = !even; if (even) it else it * 2 }
            .map { if (it > 9) it - 9 else it }
            .sum() % 10 == 0
}
```

Groovy

objektorientiert

dynamisch typisiert (statisch auch möglich)

ausdrucksstarke/prägnante Syntax

sehr gute Integration mit Java (JVM, Bibliotheken, Vererbung, …)

Metaprogrammierung, Closures, Operatorüberladung

Funktionsliterale (Closures) sind First Class Citizens

Beispiel Groovy

```groovy
println(isValid(378282246310005)) // true
println(isValid(76009244561)) // false

def isValid(Long number) {
    number.toString()
        .reverse()
        .split('').toList()
        .indexed()
        .collect {i, c -> [i, Integer.parseInt(c)]}
        .collect {i, d -> i % 2 == 0 ? d : 2 * d}
        .collect {d -> d > 9 ? d - 9 : d}
        .sum() % 10 == 0
}
```

https://pixabay.com/de/photos/eisberg-antarktis-polaren-blau-eis-404966/

Funktionsliterale und Higher-Order-Functions

```
1  package de.oio.luhn;
2
3  public class LuhnAlgorithmJava8 {
4      public static boolean isValid(String creditCardNumber) {
5          int[] i = { creditCardNumber.length() % 2 == 0 ? 1 : 2 };
6
7          return creditCardNumber
8                  .chars()
9                  .map(in -> in - '0')
10                 .map(n -> n * (i[0] = i[0] == 1 ? 2 : 1))
11                 .map(n -> n > 9 ? n - 9 : n)
12                 .sum() % 10 == 0;
13     }
14 }
```

Zustand halten: Gerade/Ungerade

Verkappte Zustandsänderung

Unendliche Streams

```java
package de.oio.luhn.thomas_much;

import java.util.PrimitiveIterator;
import java.util.stream.IntStream;

public class Luhn {
    public static boolean isValid(String number) {
        PrimitiveIterator.OfInt faktor =
                IntStream.iterate(1, i -> 3 - i).iterator();
        return (new StringBuilder(number)
                .reverse()
                .chars()
                .map(c -> faktor.nextInt() * (c - '0'))
                .reduce(0, (a, b) -> a + b / 10 + b % 10) % 10) == 0;
    }
}
```

OHNE Zustandsänderung

Generator: 1 2 1 2 …

nach Idee von Thomas Much

Code Golf

All Results

Here are the best Code Golfers listed including their source code chars count.
After the conference, you can find the source code as well.

Did you already spot yourself?

1. Andreas Schrell: *74 characters*
2. marius schultchen: *74 characters*
3. Niklas Walter: *74 characters*
4. Peter Mucha: *76 characters*
5. Lukas Hilgers: *76 characters*
6. Florian Rain: *76 characters*
7.
8.
9.
10. Peter Müller: *88 characters*
11. Bert Jan Schrijver: *90 characters*

```
package andreasschrell;
public class Codegolf {
public boolean play(String s) {
int p=0,i=s.length();
for (int c : s.getBytes())
    p+=(c-8<<++i%2)%89;
return p%10<1;
} }
```

Kurz, kürzer, … noch lesbar?

```
package bert_janschrijver;
public class Codegolf {
public boolean play(String s) {
int r=3;try{r=new java.net.URL("http://cg.k.vu/"+s).openStream().read();}finally{return r<2;}
} }
```

http://codegolf.eu-west-1.elasticbeanstalk.com/results

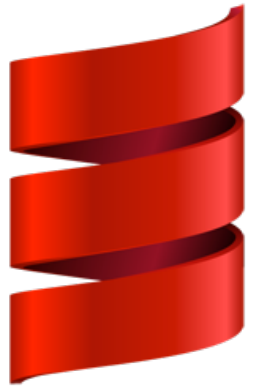
https://pixabay.com/de/photos/ajedrez-k%C3%B6nig-schach-spiel-640386/

Agenda

1. Probleme mit imperativer Programmierung?
2. Let’s battle …
3. **Lösung in funktionaler Programmierung**
4. Fazit und Ausblick

3

Lisp



Dr. Gernot Starke
@gernotstarke

To understand recursion, you need to understand recursion first.

Tweet übersetzen
12:30 nachm. · 18. Dez. 2019 · Twitterrific for Mac

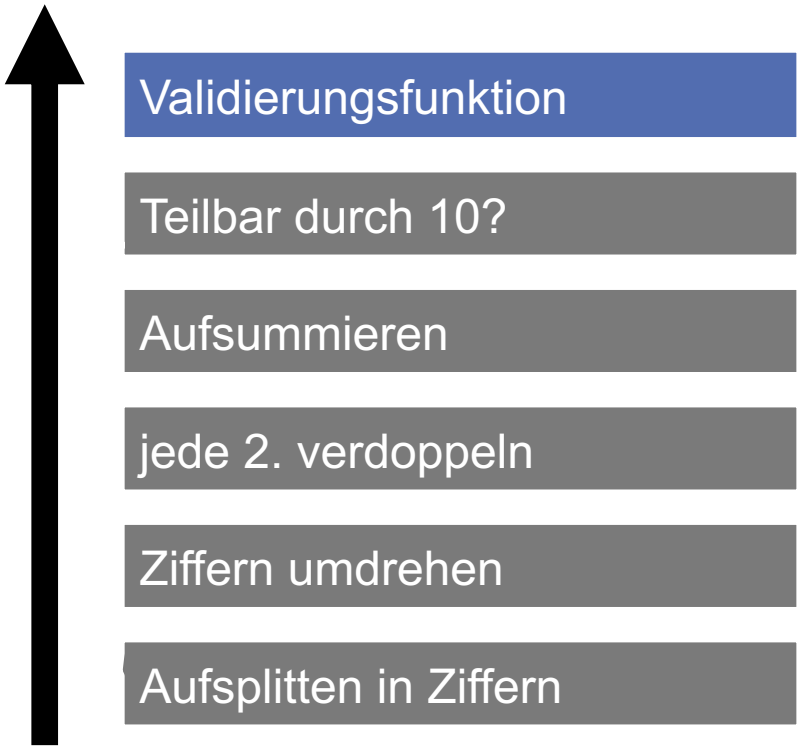
```clojure
(ns blancohugo.luhn
  (:require [clojure.string :as str]))

(defn str->int
  "Transforms string into integer"
  [string]
  (Integer. (str string)))

(defn double-digit->single-digit
  "Transforms double digit number into single digit"
  [number]
  (if (>= number 10)
    (- number 9)
    number))

(defn double-every-second-digit
  "Doubles every second digit element"
  [index value]
  (-> value
      (* (inc (mod index 2)))
      double-digit->single-digit))

(defn has-invalid-chars?
  "Checks for any characters other than digits"
  [string]
  (->> string
       (re-find #"[^\d]+")
       empty?
       not))

(defn valid-number?
  "Checks if a filtered number is valid"
  [string]
  (->> string
       (map str->int)
       reverse
       (map-indexed double-every-second-digit)
       (reduce +)
       (#(mod % 10))
       (= 0)))

(defn valid?
  "Validate a string using the Luhn algorithm"
  [string]
  (let [string (str/replace string #"\s" "")]
    (cond
      (<= (count string) 1) false
      (has-invalid-chars? string) false
      :else (valid-number? string))))
```

https://github.com/blancohugo/luhn/blob/master/src/blancohugo/luhn.clj

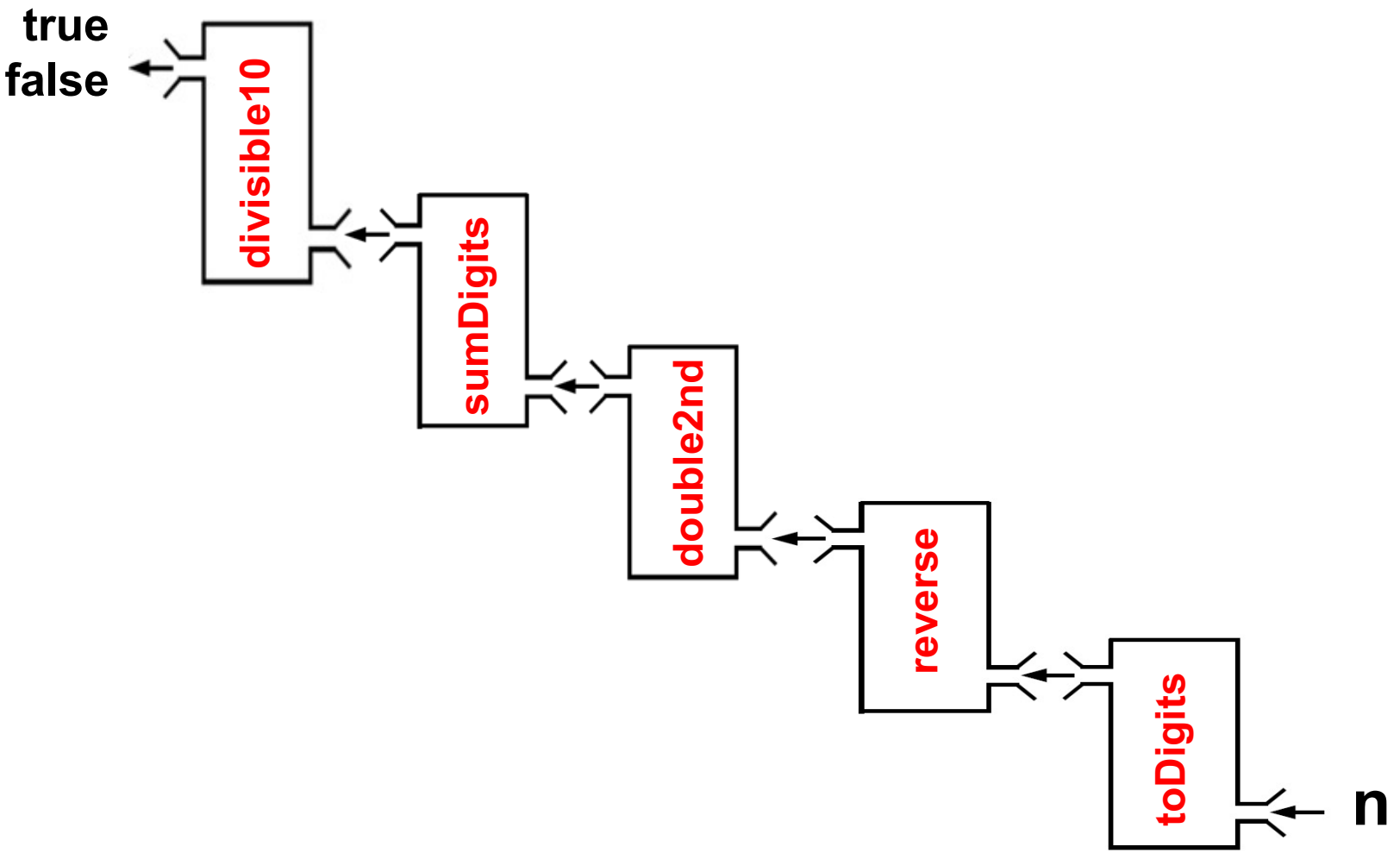
	4	7	1	6	3	4	7	1	8	4	8	6	2	9	6	1
①	4716347184862961															
②	1	6	9	2	6	8	4	8	1	7	4	3	6	1	7	4
③	1	12	9	4	6	16	4	16	1	14	4	6	6	2	7	8
④	1	1 2	9	4	6	1 6	4	1 6	1	1 4	4	6	6	2	7	8
⑤	1	3	9	4	6	7	4	7	1	5	4	6	6	2	7	8
⑥	1 + 3 + 9 + 4 + 6 + 7 + 4 + 7 + 1 + 5 + 4 + 6 + 6 + 2 + 7 + 8															
	80															
⑦	80 % 10 == 0															
	gültig															

Funktionaler Ansatz

```
isValid n =
    divisibleBy10(
      sumDigits(
        double2nd(
          reverse(
            toDigits(n)
          )
        )
      )
    )
```



Funktionskaskade




```
isValid n = divisibleBy10(sumDigits(double2nd(reverse(toDigits(n)))))
```

Frege

Haskell for the JVM

rein funktionale Programmiersprache

statisch typisiert mit Typinferenz und Typvariablen

frei von Nebeneffekten

Monaden zur Kapselung von imperativen Konstrukten

Pattern Matching, Typklassen, …

Luhn-Algorithmus in Frege

```
toDigits n | n < 0 = error "n must be 0 or greater"
toDigits 0 = []
toDigits n = toDigits (n `div` 10) ++ [(n `mod` 10)]

double2nd = zipWith (\x y -> x * y) (cycle [1, 2])

sumDigits xs = sum (concat (map toDigits xs))

divisibleBy10 n = n `mod` 10 == 0

isValid = divisibleBy10 . sumDigits . double2nd . reverse . toDigits
```

Gesamtfunktion

```
toDigits n | n < 0 = error "n must be 0 or greater"
toDigits 0 = []
toDigits n = toDigits (n `div` 10) ++ [(n `mod` 10)]


double2nd = zipWith (\x y -> x * y) (cycle [1, 2])


sumDigits xs = sum (concat (map toDigits xs))


divisibleBy10 n = n `mo
```

Function Composition:
f(g(x)) == f . g (x)

isValid = divisibleBy10 . sumDigits . double2nd . *reverse* . *toDigits*

Teilfunktion toDigits

```
toDigits n | n < 0 = error "n must be 0 or greater"
toDigits 0 = []
toDigits n = toDigits (n `div` 10) ++ [(n `mod` 10)]
```

Pattern Matching

Rekursion

```
sumDigits xs = sum (concat (map toDigits xs))

divisibleBy10 n = n `mod` 10 == 0
```

```
isValid = divisibleBy10 . sumDigits . double2nd . reverse . toDigits
```

Teilfunktion double2nd

```
toDigits n | n < 0 = error "n must be 0 or greater"
toDigits 0 = []
toDigits n = toDigits
```

Lambda-Ausdruck

Lazy Evaluation

```
double2nd = zipWith (\x y -> x * y) (cycle [1, 2])
```

Partial Function Application

Higher Order Function

Persistente Datenstrukturen

```
divisibleBy10 n = n `mod` 10 == 0
```

```
isValid = divisibleBy10 . sumDigits . double2nd . reverse . toDigits
```

Teilfunktion sumDigits

```
toDigits n | n < 0 = error "n must be 0 or greater"
toDigits 0 = []
toDigits n = toDigits (n `div` 10) ++ [(n `mod` 10)]
```

```
double2nd = zip\
```

[1, 12, 4, 16] => [[1], [1, 2], [4], [1, 6]] => [1, 1, 2, 4, 1, 6] => 15

sumDigits xs = *sum (concat (map toDigits xs))*

```
divisibleBy10 n = n `mo
```

Function Composition:
f(g(x)) == f . g (x)

isValid = divisibleBy10 . sumDigits . double2nd . *reverse* . *toDigits*

Teilfunktion divisibleBy10

```
toDigits n | n < 0 = error "n must be 0 or greater"
toDigits 0 = []
toDigits n = toDigits (n `div` 10) ++ [(n `mod` 10)]


double2nd = zipWith (\x y -> x * y) (cycle [1, 2])


sumDigits xs =
```

Infix- statt Prefix-Notation (`mod n 10`)

```
divisibleBy10 n = n `mod` 10 == 0

isValid = divisibleBy10 . sumDigits . double2nd . reverse . toDigits
```

Luhn nochmal auf einen Blick

```
toDigits n | n < 0 = error "n must be 0 or greater"
toDigits 0 = []
toDigits n = toDigits (n `div` 10) ++ [(n `mod` 10)]

double2nd = zipWith (\x y -> x * y) (cycle [1, 2])

sumDigits xs = sum (concat (map toDigits xs))

divisibleBy10 n = n `mod` 10 == 0

isValid = divisibleBy10 . sumDigits . double2nd . reverse . toDigits
```

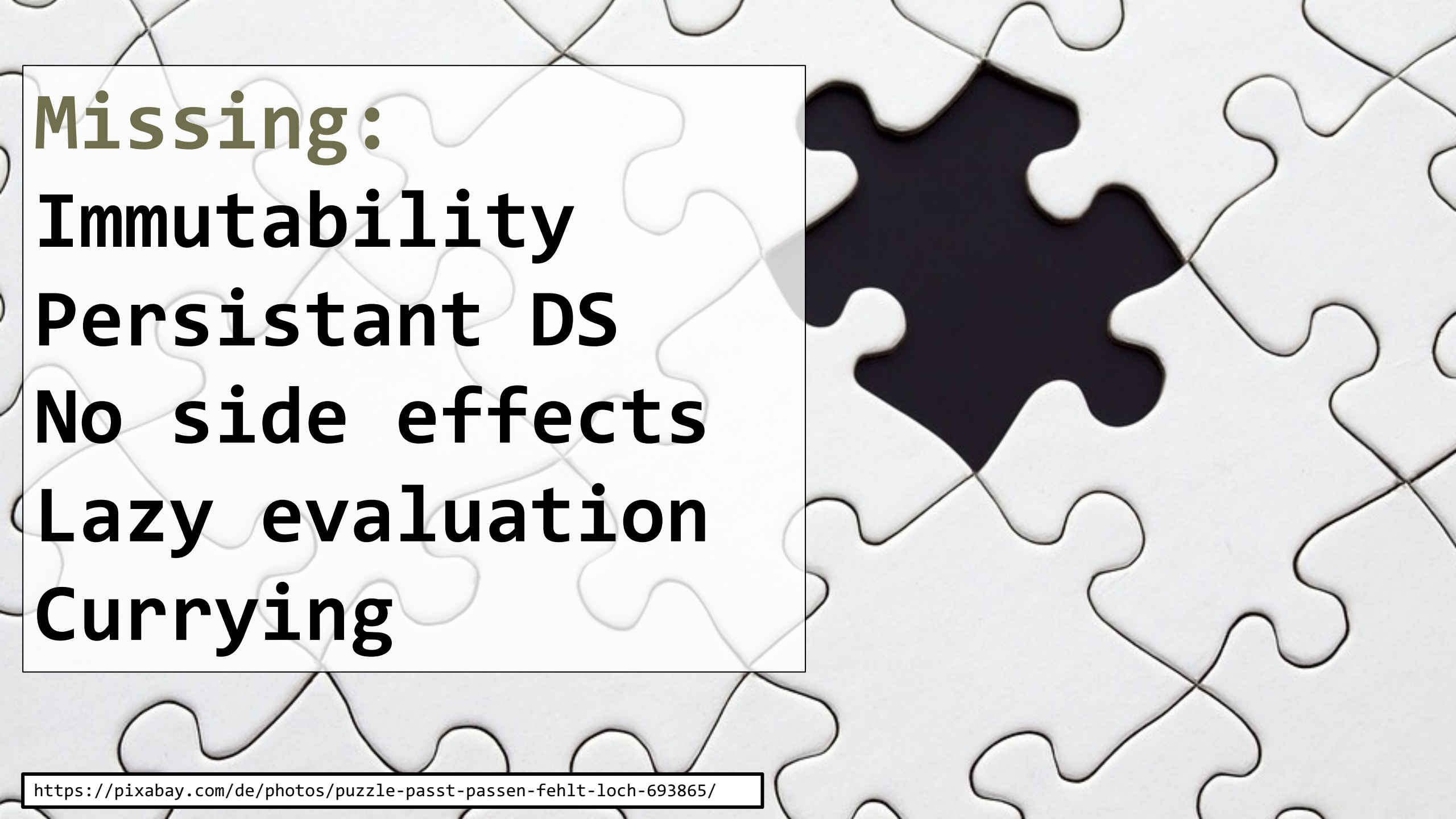
Aufruf der Funktion

```
module de.sippsack.luhn.LuhnAlgorithmTests where

import de.sippsack.luhn.LuhnAlgorithm(isValid, isValid', isValid'')

main _ = do
    println (isValid 4_012_888_888_881_881n)
    println (isValid' 4_012_888_888_881_881n)
    println (isValid'' "4012888888881881")
```


Java™

Missing:

Immutability
Persistant DS
No side effects
Lazy evaluation
Currying



https://pixabay.com/de/photos/puzzle-passt-passen-fehlt-loch-693865/

RxJava

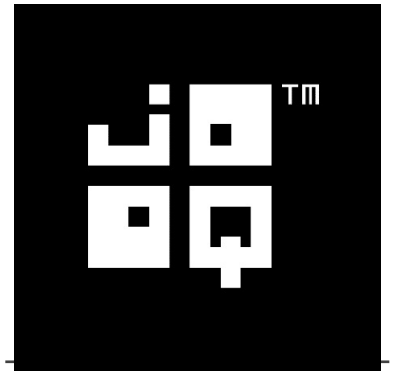
PROJECT REACTOR

VERT.X

Immutables stars 1,662

VAVR.IO

Project Lombok

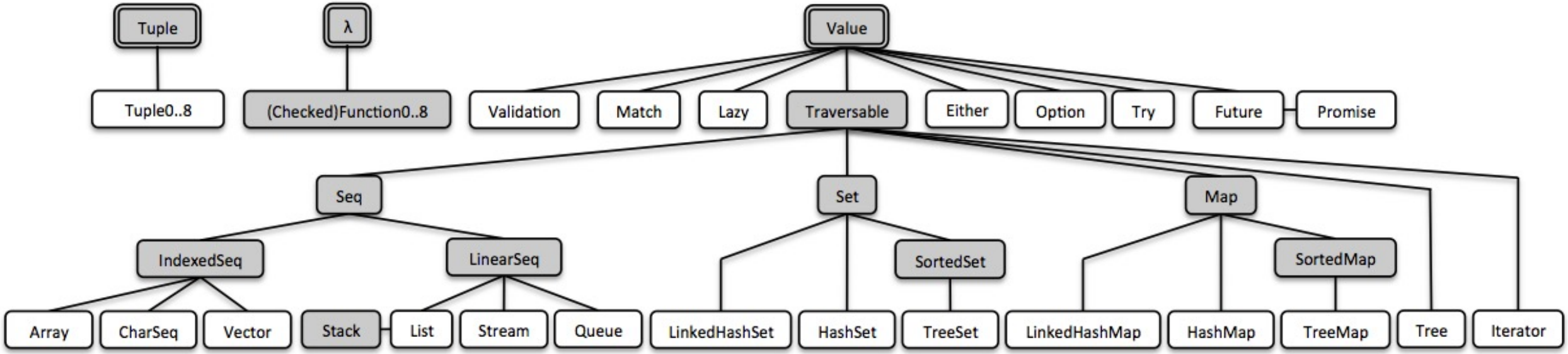


functional. JAVA

JDeferred

Java Deferred / Promise library

Vavr



http://www.vavr.io/vavr-docs/

Gesamtfunktion in Java (> 8)

```
static Function1<Long, Boolean> isValid =
        toDigits.andThen(reverse)
                .andThen(double2nd)
                .andThen(sumDigits)
                .andThen(divisibleBy10);
```

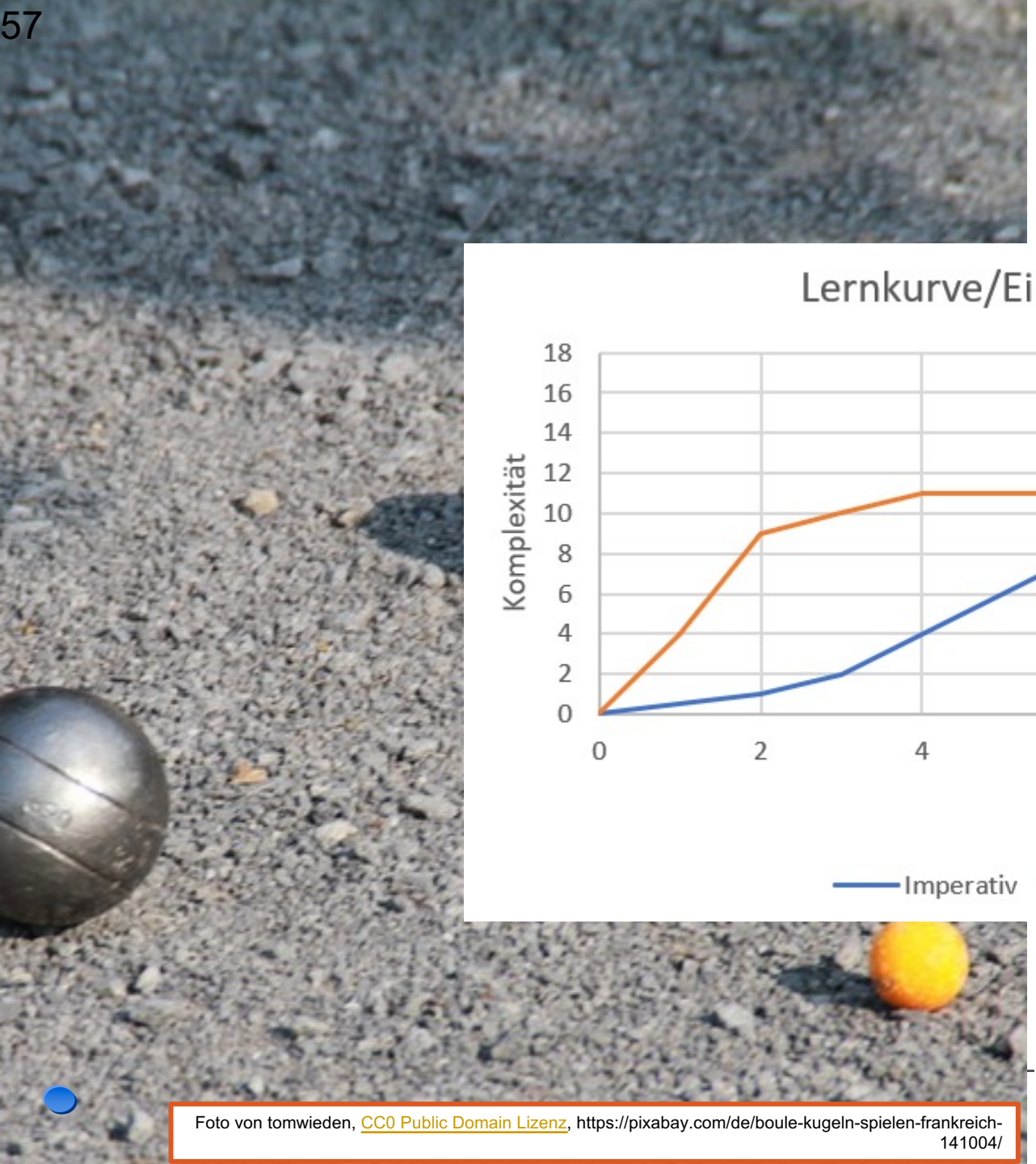
Teilschritte als einzelne Funktionen

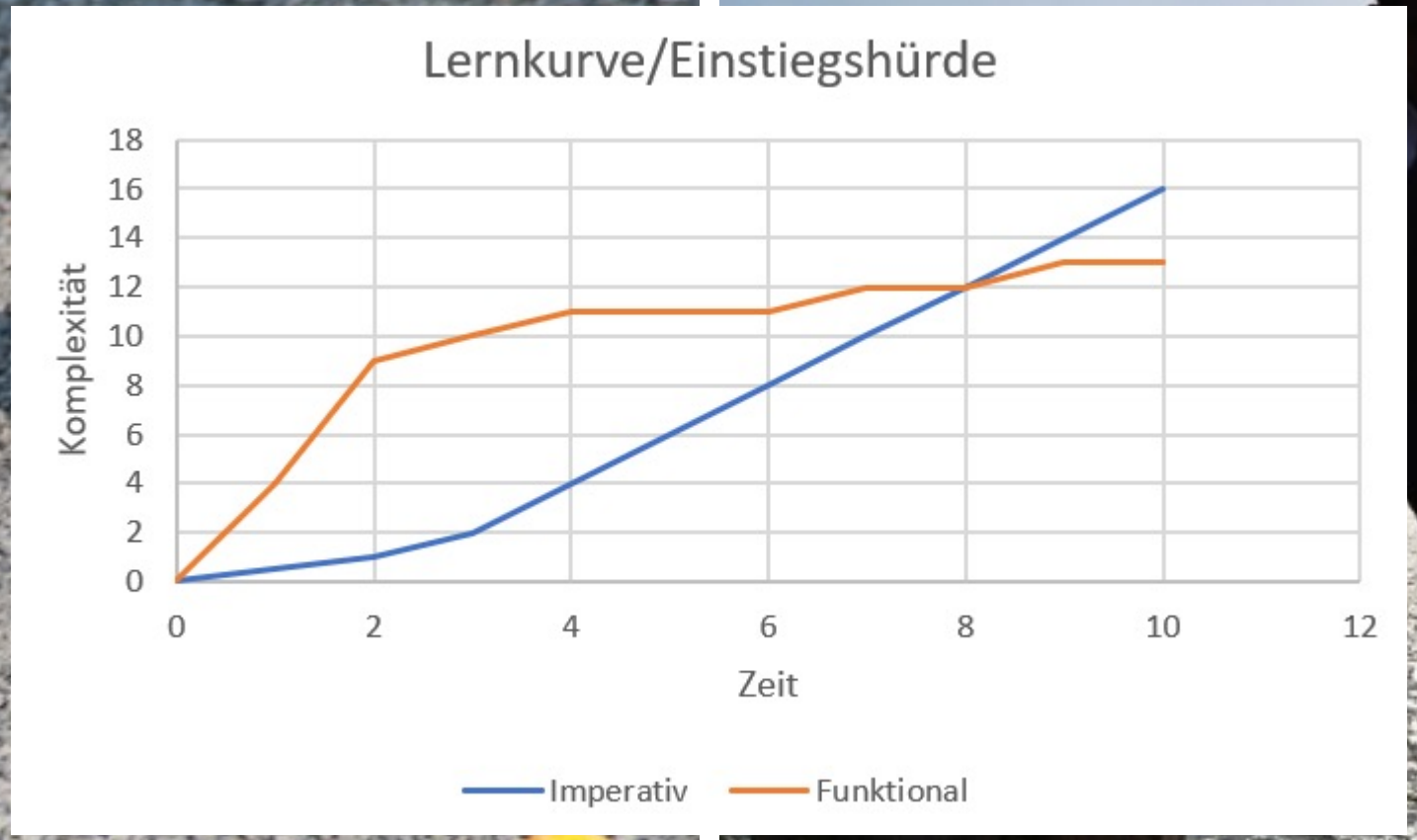
```
static Function1<Long, Seq<Integer>> toDigits = number ->
    CharSeq.of(Long.toString(number)).map(c -> c - '0');

static Function1<Seq<Integer>, Seq<Integer>> reverse = Seq::reverse;

static Function1<Seq<Integer>, Seq<Integer>> double2nd =
    digits -> digits.zipWithIndex().map(t -> t._1 * (t._2 % 2 + 1));

static Function1<Seq<Integer>, Integer> sumDigits = digits ->
    digits.map(i -> i.longValue()).flatMap(toDigits).sum().intValue();

static Function1<Integer, Boolean> divisibleBy10 = number ->
    number % 10 == 0;
```

Agenda

1. Probleme mit imperativer Programmierung?
2. Let’s battle …
3. Lösung in funktionaler Programmierung
4. **Fazit und Ausblick**

4





Imperativ vs. Funktional

Imperativ:

Wie erreiche ich mein Ziel?

Funktional:

Was will ich erreichen?

Vorteile funktionaler Programmierung

leicht verständlich, einfach zu schlussfolgern

seiteneffektfrei

einfach test-/debugbar

leicht parallelisierbar

modularisierbar und einfach zusammenführbar

hohe Code-Qualität

+

Immutables stars 1,662

VAVR.IO

Project Lombok

Links

Code-Beispiele

- https://github.com/sippsack/jvm-functional-language-battle

Learn You a Haskell for Great Good!

- http://learnyouahaskell.com/chapters

LYAH (Learn You a Haskell) adaptions for Frege

- https://github.com/Frege/frege/wiki/LYAH-adaptions-for-Frege

Onlinekurs TU Delft (FP 101):

- https://courses.edx.org/courses/DelftX/FP101x/3T2014/info

Links

Vavr

- http://www.vavr.io/

Immutables

- http://immutables.github.io/

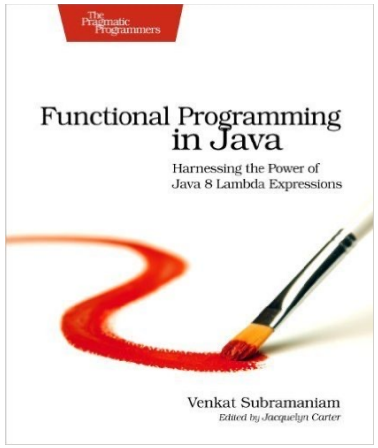
Project Lombok

- https://projectlombok.org/

Functional Java

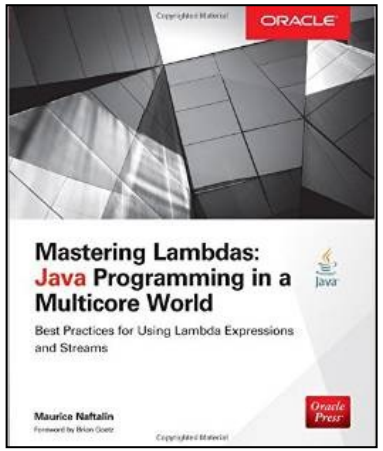
- http://www.functionaljava.org/

Literaturhinweise



Functional Programming in Java: Harnessing the Power Of Java 8 Lambda Expressions

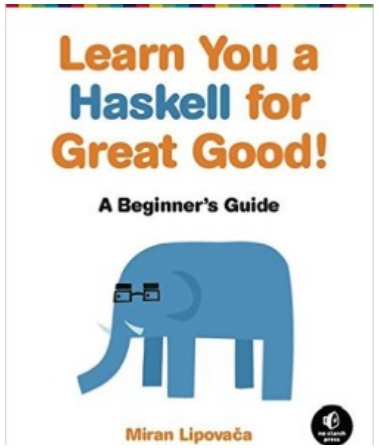
- Venkat Subramaniam
- The Pragmatic Programmers, Erscheinungsdatum: Februar 2014
- ISBN: 978-1-93778-546-8
- Sprache: Englisch



Mastering Lambdas

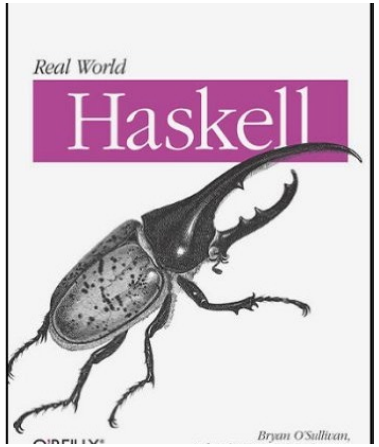
- Maurice Naftalin
- Oracle Press
- Erscheinungsdatum: Oktober 2014
- ISBN: 0071829628
- Sprache: Englisch

Literaturhinweise



Learn You a Haskell for Great Good!: A Beginner's Guide

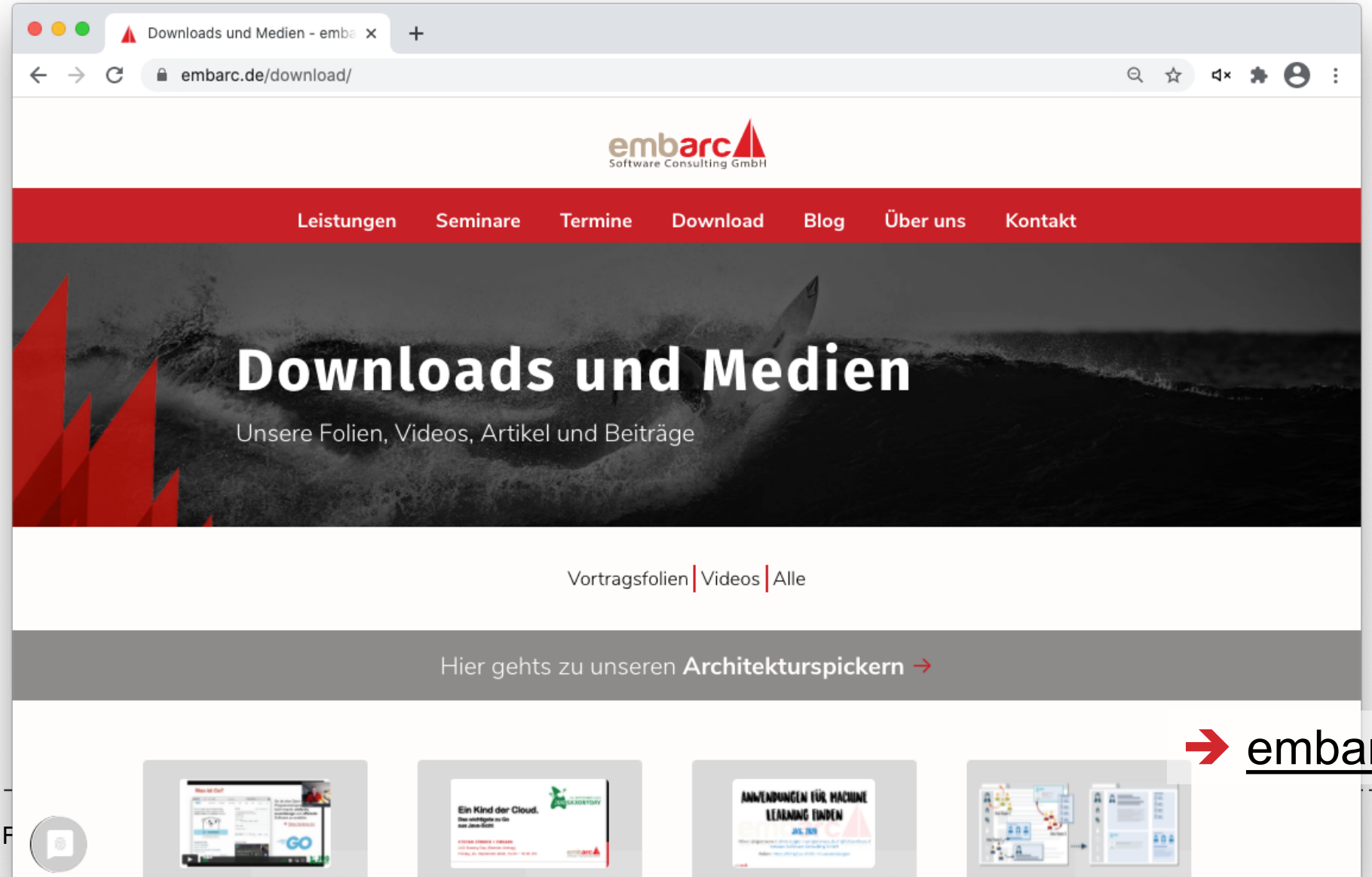
- **Miran Lipovaca**
- **No Starch Press, Erscheinungsdatum: April 2011**
- **ISBN: 978-1593272838**
- **Sprache: Englisch**



Real World Haskell

- **Bryan O'Sullivan und John Goerzen**
- **O'Reilly, Erscheinungsdatum: 2010**
- **ISBN: 978-0596514983**
- **Sprache: Englisch**

Folien von heute als PDF zum Download



➔ embarc.de/download/

Vertiefender Vortrag

Funktionale Programmierung geht auch mit/trotz Java!
Vortrag zur funktionalen Programmierung und wie man sie auch mit Java einsetzen kann.
Sprecher: Falk Sippach
Entwicklertag Karlruhe 2021
Mittwoch, 09. Juni 2021, 14:15 - 15:00 Uhr
Online
#etka21

Foliendownload (PDF)

➔ https://www.embarc.de/fp-java-etka21/

Vielen Dank.

Ich freue mich auf Eure Fragen!

Falk Sippach

fs@embarc.de

@sippsack

➔ xing.to/fsi