Scala (Programmiersprache)

Ein Teaser und allgemeinere Gedanken

Sebastian Eidecker

16. März 2016

Scala (Programmiersprache)
Ein Teaser und allgemeinere Gedanken
Sebastian Eidecker
16. März 2016

Wer als Werkzeug nur einen Hammer hat, sieht in jedem Problem einen Nagel.

— Paul Watzlawick

2016-03-11

Scala (Programmiersprache)

Wer als Werkzeug nur einen Hammer hat, sieht in jedem Problem einen Nagel.

— Paul Watzlawick

Ich möchte heute über Scala reden. Eigentlich will ich aber nicht nur über Scala reden. Ich möchte nicht Scala verkaufen, sondern etwas völlig anderes. Scala ist spannend - für Nerds, die des Geldes wegen Java benutzen. Wir müssen aber auch über wichtigere Probleme reden. Und vielleicht passt Scala ja doch irgendwie.

In der IT wollen wir Probleme lösen und Möglichkeiten schaffen. Ich bin der festen Überzeugung, dass wir zu selten über den Kern unserer Probleme nachdenken und auch, dass sich diese Probleme gerade ändern und immer schneller ändern werden.

Ich glaube daher, dass wir unser Problem und unseren Werkzeugkasten kennen müssen. Ich glaube, dass wir zumindest wissen müssen, was im Baumarkt ausliegt und wir bei Bedarf einkaufen können. Daher kann es aus meiner Sicht nie schaden, sich neue Werkzeuge und Arbeitsweisen anzuschauen, damit man sie bei Bedarf zumindest im Hinterkopf hat. Sonst kann es passieren, dass wir das Problem gar nicht verstehen, weil wir kein passendes Werkzeug dafür besitzen und verwenden können.

Worüber reden wir?

IT im Wandel

Deswegen, möchte ich über zwei Dinge reden: Neue Herausforderungen und Scala.

Worüber reden wir?

IT im Wandel

Herausforderungen

Worüber reden wir?
IT im Wandel
Herausforderungen

Deswegen, möchte ich über zwei Dinge reden: Neue Herausforderungen und Scala.

Worüber reden wir?

IT im Wandel

Herausforderungen

Manifeste

Worüber reden wir?
IT im Wandel
Herausforderungen
Manifeste

Deswegen, möchte ich über zwei Dinge reden: Neue Herausforderungen und Scala.

Worüber reden wir?

IT im Wandel

Herausforderungen

Manifeste

Scala

Deswegen, möchte ich über zwei Dinge reden: Neue Herausforderungen und Scala.

Worüber reden wir?

IT im Wandel

Herausforderungen

Manifeste

Scala

Management Summary

Deswegen, möchte ich über zwei Dinge reden: Neue Herausforderungen und Scala.

Worüber reden wir?

IT im Wandel

Herausforderungen

Manifeste

Scala

Management Summary

Ein wenig Code

Deswegen, möchte ich über zwei Dinge reden: Neue Herausforderungen und Scala.

Worüber reden wir?

IT im Wandel

Herausforderungen

Manifeste

Scala

Management Summary

Ein wenig Code

Spannendes

Deswegen, möchte ich über zwei Dinge reden: Neue Herausforderungen und Scala.

IT im Wandel

Herausforderungen

IT im Wandel
Herausforderungen

Software Engineering

Software Engineering. Passt der Begriff?

Software Engineering

Software Engineering. Passt der Begriff?

Forderungen an IT

Forderungen an IT

- Stabilität und Resilienz

Forderungen an IT
· Stabilität und Resilienz

Magnor: Sorgen sie durch ihre Arbeit dafür, dass Systeme Kriterien der Stabilität und Resilienz erfüllen. Seien sie „dran“ an den Problemen, antizipieren sie sie, schaffen sie Fall-Back-Lösungen – auch an Wochenenden.

Forderungen an IT

- Stabilität und Resilienz
- Wertbeitrag

Forderungen an IT
- Stabilität und Resilienz
- Wertbeitrag

IT entwickelt sich zunehmend zu einem eigenen Produktionsfaktor (4-Sektor-Hypothese). Seien sie ein verlässlicher Partner für das Business, nehmen sie die Business-Seite auch an die Hand, helfen und unterstützen sie, leisten sie einen wahrgenommenen Wertbeitrag.[...]

Forderungen an IT

- Stabilität und Resilienz
- Wertbeitrag
- Businesstreiber

Forderungen an IT
- Stabilität und Resilienz
- Wertbeitrag
- Businesstreiber

Die Entwicklung der IT treibt zunehmend die Business-Seite und verändert Geschäftsmodelle. Seien sie an der Front und treiben sie das Business, entwickeln sie neue Ideen, forschen sie nach neuen Lösungen (Beispiele: Uber, 3D-Printing, CoLo21, ...

Forderungen an IT

- Stabilität und Resilienz
- Wertbeitrag
- Businesstreiber

— Matthias Magnor – CEO Surface und Contract Logistics

Forderungen an IT
- Stabilität und Resilienz
- Wertbeitrag
- Businesstreiber

— Matthias Magnor – CEO Surface und Contract Logistics

Das Bewusstsein, dass ein Wandel stattfindet, ist im Business angekommen. Diese Forderungen stammen von

Matthias Magnor!

IT im Wandel

Manifeste

Ein Manifest (lateinisch manifestus ‚handgreiflich gemacht') ist eine öffentliche Erklärung von Zielen und Absichten, oftmals politischer Natur. (Wikipedia)

Manifeste

- Antwortbereit, Widerstandsfähig, Elastisch, Nachrichtenorientiert (2013)

Manifeste
- Antwortbereit, Widerstandsfähig, Elastisch, Nachrichtenorientiert (2013)

Manifeste

- Gut gefertigt, Stets Mehrwert, Gemeinschaft aus Experten, Produktive Partnerschaften (2009)

Manifeste

- Gut gefertigt, Stets Mehrwert, Gemeinschaft aus Experten, Produktive Partnerschaften (2009)

Manifeste

- Individuen und Interaktionen, Funktionierende Software, Zusammenarbeit mit dem Kunden, Reagieren auf Veränderung (2001)

Manifeste

- Individuen und Interaktionen, Funktionierende Software, Zusammenarbeit mit dem Kunden, Reagieren auf Veränderung (2001)

Manifeste

- Antwortbereit, Widerstandsfähig, Elastisch, Nachrichtenorientiert (2013)
- Gut gefertigt, Stets Mehrwert, Gemeinschaft aus Experten, Produktive Partnerschaften (2009)
- Individuen und Interaktionen, Funktionierende Software, Zusammenarbeit mit dem Kunden, Reagieren auf Veränderung (2001)

Manifeste
- Antwortbereit, Widerstandsfähig, Elastisch, Nachrichtenorientiert (2013)
- Gut gefertigt, Stets Mehrwert, Gemeinschaft aus Experten, Produktive Partnerschaften (2009)
- Individuen und Interaktionen, Funktionierende Software, Zusammenarbeit mit dem Kunden, Reagieren auf Veränderung (2001)

Es gibt Manifeste von Softwareentwicklern, die sehr Ähnliches aussagen. (Reaktives Manifest, agiles Manifest, Manifest der Software Craftmanship-Bewegung als Beispiele) Diese sind bekannt und – ich habe zumindest das Gefühl – auch anerkannt. Ich will auch nur sehr kurz darauf eingehen, es soll ja vor allem um Scala gehen.

Wo stehen wir?

Wo stehen wir im Wettbewerb?

Scala

Management Summary

Scala
Management Summary

Scalable Language

Was bedeutet das? Scala ist sehr flexibel. Es vereint sehr viele Konzepte und ist im API-Design sehr stark. So sollen unterschiedlichste Szenarien und Anforderungen bedient werden. Schauen wir, ob das Versprechen gehalten wird.

Scalable Language

This means that Scala grows with you. You can play with it by typing one-line expressions and observing the results. But you can also rely on it for large mission critical systems [...]

— www.scala-lang.org

Scalable Language

This means that Scala grows with you. You can play with it by typing one-line expressions and observing the results. But you can also rely on it for large mission critical systems [...]

— www.scala-lang.org

Was bedeutet das? Scala ist sehr flexibel. Es vereint sehr viele Konzepte und ist im API-Design sehr stark. So sollen unterschiedlichste Szenarien und Anforderungen bedient werden. Schauen wir, ob das Versprechen gehalten wird.

Eigenschaften

Eigenschaften

- Objektorientiert

Eigenschaften
- Objektorientiert

Keine Primitive, wie in Java. Ein schönes Beispiel: Dort hat man per Autoboxing Komplexität eingeführt statt

Schönheit. NullPointer bei Primitiven anyone?

Eigenschaften

- Objektorientiert
- Funktional

Eigenschaften
- Objektorientiert
- Funktional

Eigenschaften

- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference

Eigenschaften
- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference

Aber Type Inference, wenn gewünscht

Eigenschaften

- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference
- Immutable by default

Eigenschaften
- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference
- Immutable by default

Eigenschaften

- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference
- Immutable by default
- Gewohnte Syntax („Java ohne Semikolon“)

Eigenschaften
- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference
- Immutable by default
- Gewohnte Syntax („Java ohne Semikolon“)

Eigenschaften

- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference
- Immutable by default
- Gewohnte Syntax („Java ohne Semikolon“)
- Ausdrucksstark (APIs/DSLs schreiben)

Eigenschaften
- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference
- Immutable by default
- Gewohnte Syntax („Java ohne Semikolon“)
- Ausdrucksstark (APIs/DSLs schreiben)

Bei Java bricht man sich schon die Hände, will man ein Builder-Pattern rekursiv umsetzen oder eine fluent API

Eigenschaften

- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference
- Immutable by default
- Gewohnte Syntax („Java ohne Semikolon“)
- Ausdrucksstark (APIs/DSLs schreiben)
- Jung (2004, Hype 2011)

Eigenschaften
- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference
- Immutable by default
- Gewohnte Syntax („Java ohne Semikolon“)
- Ausdrucksstark (APIs/DSLs schreiben)
- Jung (2004, Hype 2011)

Versprechen

Versprechen

- Produktivitätssteierung

Versprechen
- Produktivitätssteierung

Versprechen

- Produktivitätssteierung
- Höhere Codequalität

Versprechen
- Produktivitätssteierung
- Höhere Codequalität

Versprechen

- Produktivitätssteierung
- Höhere Codequalität
- Mehr Spaß

Versprechen
- Produktivitätssteierung
- Höhere Codequalität
- Mehr Spaß

Versprechen

- Produktivitätssteierung
- Höhere Codequalität
- Mehr Spaß

  durch
- Weniger Code

Versprechen
- Produktivitätssteierung
- Höhere Codequalität
- Mehr Spaß
  durch
- Weniger Code

Versprechen

- Produktivitätssteierung
- Höhere Codequalität
- Mehr Spaß

  durch
- Weniger Code
- Höheres Abstraktionsniveau

Versprechen
- Produktivitätssteierung
- Höhere Codequalität
- Mehr Spaß
  durch
- Weniger Code
- Höheres Abstraktionsniveau

Versprechen

- Produktivitätssteierung
- Höhere Codequalität
- Mehr Spaß

  durch
- Weniger Code
- Höheres Abstraktionsniveau
- Skalierbarkeit

Versprechen
- Produktivitätssteierung
- Höhere Codequalität
- Mehr Spaß
  durch
- Weniger Code
- Höheres Abstraktionsniveau
- Skalierbarkeit

Passt in unseren Entwicklungsprozess. Leichte Änderungen an Deployment etc.

Scala und die Java-Plattform

Scala und die Java-Plattform

- Java-Bytecode, läuft auf JVM

Scala und die Java-Plattform
- Java-Bytecode, läuft auf JVM

Scala und die Java-Plattform

- Java-Bytecode, läuft auf JVM
- Java-Bibliotheken nutzbar

Scala und die Java-Plattform
- Java-Bytecode, läuft auf JVM
- Java-Bibliotheken nutzbar

Scala und die Java-Plattform

- Java-Bytecode, läuft auf JVM
- Java-Bibliotheken nutzbar
- Bekannte IDEs

Scala und die Java-Plattform
- Java-Bytecode, läuft auf JVM
- Java-Bibliotheken nutzbar
- Bekannte IDEs

Wenn man schon eine neue Sprache lernt, wird man wenigstens bei IDE und Buildprozess abgeholt.

Scala und die Java-Plattform

- Java-Bytecode, läuft auf JVM
- Java-Bibliotheken nutzbar
- Bekannte IDEs
- Ähnlicher Paketierungs- und Buildprozess (sbt)

Scala und die Java-Plattform
- Java-Bytecode, läuft auf JVM
- Java-Bibliotheken nutzbar
- Bekannte IDEs
- Ähnlicher Paketierungs- und Buildprozess (sbt)

Wenn man schon eine neue Sprache lernt, wird man wenigstens bei IDE und Buildprozess abgeholt.

Scala

Ein wenig Code

Scala
Ein wenig Code

Alles geklaut von Wikipedia oder Heiko Seebergers Buch

Eine Java-Klasse

Eine Java-Klasse

```java
public class Person {
  private final String firstName;
  private final String lastName;
  public Person(String firstName, String lastName) {
      this.firstName = firstName;
      this.lastName = lastName;
          }
  public String getFirstName() {
      return firstName;
  }
  public String getLastName() {
      return lastName;
  }
```


Eine Java-Klasse

```java
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Person person = (Person) o;
    if (firstName != null ?
        !firstName.equals(person.firstName) :
        person.firstName != null) return false;
    if (lastName != null ?
        !lastName.equals(person.lastName) :
        person.lastName != null) return false;
    return true;
}
```


Eine Java-Klasse

```
  @Override
  public int hashCode() {
      int result = firstName != null ? firstName.hashCode() : 0;
      result =
          31 * result + (lastName != null ? lastName.hashCode() :
              0);
      return result;
  }
}
```


Businesslogik?

Wo ist die Businesslogik versteckt? Ich gebe zu, das wird nur ein Beispiel, aber es zeigt symptomatisch eine Schwäche von Java: Java ist leicht verständlich, Änderungen werden aber immer non-breaking eingeführt und fühlen sich oft deplatziert an. Scala hat den Vorteil, neuer zu sein und kennt die Schmerzen der Java-Entwickler. Daher sind viele Dinge eingebaut, die man sich schon lange wünscht oder Konzepte, die man eigentlich schon immer umsetzen wollte, wenn es denn nicht so lästig wäre. (Immutable Objects, Lambdas, Generics, checked Exceptions als Beispiel)

Ein besonders lästiges Nicht-Feature-Feature sind in meinen Augen JavaBeans, wie wir sie gerade gesehen haben.

Leicht verständlich, aber strikte Richtlinien durch Konvention. Und wehe, dein Framework mag die Namen Deiner

Getter und Setter nicht.

Dasselbe in Scala

Dasselbe in Scala

Ja, nur ein Beispiel, ein besonders krasses. Aber dennoch ein alltägliches.

Dasselbe in Scala

```scala
case class Person(firstName:String, lastName:String)
```


Dasselbe in Scala
case class Person(firstName:String, lastName:String)

Ja, nur ein Beispiel, ein besonders krasses. Aber dennoch ein alltägliches.

Es wird funktional – Quicksort

Man sieht: Ausdrucksstark, aber auch nicht mehr so straight forward wie Java (vor 8)

Es wird funktional – Quicksort

```scala
def quickSort[A <% Ordered[A]](xs: List[A]): List[A] = xs match {
  case Nil     => xs
  case y :: ys => ys partition (_ <= y) match {
    case (l1, l2) => quickSort(l1) ++ (y :: quickSort(l2))
  }
}
```


Es wird funktional – Quicksort

```scala
def quickSort[A <% Ordered[A]](xs: List[A]): List[A] = xs match {
  case Nil     => xs
  case y :: ys => ys partition (_ <= y) match {
    case (l1, l2) => quickSort(l1) ++ (y :: quickSort(l2))
  }
}
```

Man sieht: Ausdrucksstark, aber auch nicht mehr so straight forward wie Java (vor 8)

2016-03-11

Scala (Programmiersprache)
└Scala
└Ein wenig Code

Jetzt endlich ein Schnelldurchlauf durch einige grundlegende Features von Scala. Leider nur kurz, ich habe meine Zeit ja mit Management Summary und meiner politischen Agenda vertrödelt. Alle Beispiele sind natürlich geklaut.

Also hier nur ein paar Dinge zum Grundverständnis und ein paar nette Sachen, die Java-Entwickler interessieren dürften.

val und var – Immutables

i ist val per Default, also immutable. Letzter Ausdruck ist return

val und var – Immutables

```
1 def addOne(i: Int): Int = { i += 1; i }
```


i ist val per Default, also immutable. Letzter Ausdruck ist return

val und var – Immutables

```
1 def addOne(i: Int): Int = { i += 1; i }
```

Compile-Fehler, da i immutable

val und var – Immutables
def addOne(i: Int): Int = { i += 1; i }
Compile-Fehler, da i immutable

i ist val per Default, also immutable. Letzter Ausdruck ist return

Funktionen – Benannte Parameter

Funktionen – Benannte Parameter

```
1       class
```


Funktionen – Benannte Parameter
1 class

Klassen und Objekte

Klassen und Objekte

```
1 class
```


Klassen und Objekte
class

Listen

Listen

```
1       class
```


Listen
class

Pattern Matching

Pattern Matching

```
1       class
```


Pattern Matching
class

Type Inference

Type Inference

```
1 def f() = 3 * 2
2
3 def f() : Int = 3 * 2
```


Type Inference
def f() = 3 * 2
def f() : Int = 3 * 2

Implizites return

Kein return, letzte Anweisung wird zurückgegeben. Die Methode ist statisch typisiert! Der Compiler kann den Typen der Rückgabe ermitteln: String – Type Inference Wenn wir wollen, können wir ihn deklarieren. Bei öffentlichen Methoden sollte man das auch tun.

Implizites return

```
1 def f() = {
2   if (something)
3     "A"
4   else
5     "B"
6 }
```


Implizites return
1 def f() = {
2   if (something)
3   "A"
4   else
5   "B"
6 }

Kein return, letzte Anweisung wird zurückgegeben. Die Methode ist statisch typisiert! Der Compiler kann den Typen der Rückgabe ermitteln: String – Type Inference Wenn wir wollen, können wir ihn deklarieren. Bei öffentlichen Methoden sollte man das auch tun.

Implizites return

```
1 def f() = {
2   if (something)
3     "A"
4   else
5     "B"
6 }
```

Letzte Anweisung wird zurückgegeben, impliziter Typ String.

Implizites return

```
1 def f() = {
2   if (something)
3     "A"
4   else
5     "B"
6 }
```

Letzte Anweisung wird zurückgegeben, impliziter Typ String.

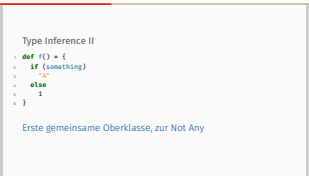
Kein return, letzte Anweisung wird zurückgegeben. Die Methode ist statisch typisiert! Der Compiler kann den Typen der Rückgabe ermitteln: String – Type Inference Wenn wir wollen, können wir ihn deklarieren. Bei öffentlichen Methoden sollte man das auch tun.

Type Inference II

Type Inference II

```scala
def f() = {
  if (something)
    "A"
  else
    1
}
```

Erste gemeinsame Oberklasse, zur Not Any



Vererbung und Traits

Vererbung und Traits

```
1       class
```


Vererbung und Traits
class

Funktionen funktional – Lambdas schön

Funktionen funktional – Lambdas schön

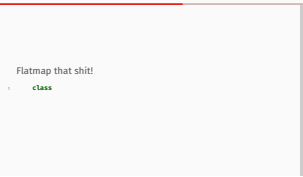
```
1       class
```


Funktionen funktional – Lambdas schön
class

Flatmap that shit!

Flatmap that shit!

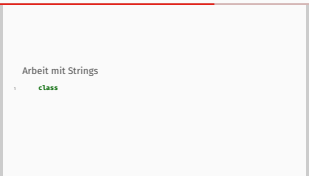
```
1       class
```

Arbeit mit Strings

Arbeit mit Strings

```
1       class
```

Tupel

Tupel

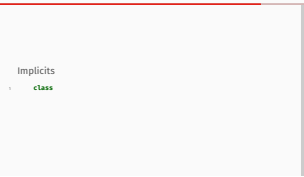
```
1       class
```


Tupel
class

Implicits

Implicits

```
1       class
```

??? – Mein heimlicher Star

??? – Mein heimlicher Star

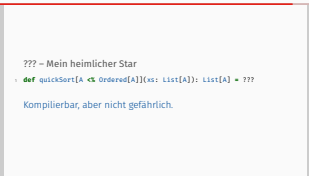
```scala
def quickSort[A <% Ordered[A]](xs: List[A]): List[A] = ???
```


??? – Mein heimlicher Star
def quickSort[A <% Ordered[A]](xs: List[A]): List[A] = ???

??? – Mein heimlicher Star

```scala
def quickSort[A <% Ordered[A]](xs: List[A]): List[A] = ???
```

Kompilierbar, aber nicht gefährlich.



2016-03-11

Scala (Programmiersprache)
└Scala
└Ein wenig Code

```
1 class
```


class

Scala

Spannendes

Scala
Spannendes

Akka

Akka

- Scalable real-time transaction processing

Akka
- Scalable real-time transaction processing

Akka

- Scalable real-time transaction processing
- Will die aktuellen Probleme lösen

Akka
- Scalable real-time transaction processing
- Will die aktuellen Probleme lösen

Akka

- Scalable real-time transaction processing
- Will die aktuellen Probleme lösen
-

Akka
- Scalable real-time transaction processing
- Will die aktuellen Probleme lösen
-

ScalaTest

ScalaTest

- Fachlich verständliche Tests

ScalaTest
- Fachlich verständliche Tests

ScalaTest

- Fachlich verständliche Tests
- Testdatengenerierung

ScalaTest
- Fachlich verständliche Tests
- Testdatengenerierung

Scalatest – Beispiel

```
"Creating a Time" should {
    "throw an IllegalArgumentException for hours less than 0 or
       greater equal 24" in {
      forAll("hours") { (hours: Int) =>
        whenever(hours < 0 || hours >= 24) {
          an[IllegalArgumentException] should be thrownBy Time(
             hours)
        }
      }
    }
  }
```


Domain Specific Languages

2016-03-11

Scala (Programmiersprache)
└Scala
└Spannendes

Domain Specific Languages

-

Domain Specific Languages

-
-

Domain Specific Languages
.
.

Domain Specific Languages

-
-
-

Domain Specific Languages
-
-
-

Meine wenig qualifizierte Meinung

2016-03-11

Scala (Programmiersprache)
└ Scala
└ Spannendes
└ Meine wenig qualifizierte Meinung

Meine wenig qualifizierte Meinung

Meine wenig qualifizierte Meinung

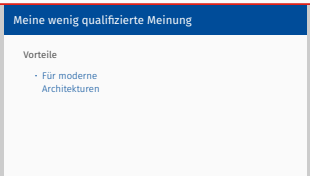
Vorteile

Meine wenig qualifizierte Meinung
Vorteile

Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen



Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional

Meine wenig qualifizierte Meinung
Vorteile
- Für moderne Architekturen
- Verständlich funktional

Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem

Meine wenig qualifizierte Meinung
Vorteile
- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem

Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß

Meine wenig qualifizierte Meinung
Vorteile
- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß

Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

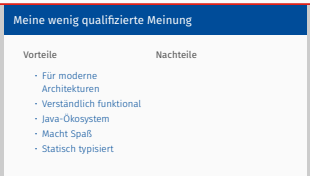
Meine wenig qualifizierte Meinung
Vorteile
- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

Nachteile



Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

Nachteile

- Komplex

Meine wenig qualifizierte Meinung
Vorteile
- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert
Nachteile
- Komplex

Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

Nachteile

- Komplex
- Zukunftssicher?

Meine wenig qualifizierte Meinung

Vorteile
- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

Nachteile
- Komplex
- Zukunftssicher?

Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

Nachteile

- Komplex
- Zukunftssicher?
- Anzahl Entwicklungssklaven

Meine wenig qualifizierte Meinung

Vorteile
- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

Nachteile
- Komplex
- Zukunftssicher?
- Anzahl Entwicklungssklaven

Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

Nachteile

- Komplex
- Zukunftssicher?
- Anzahl Entwicklungssklaven
- Binärkompatibilität nicht in alle Ewigkeit

Meine wenig qualifizierte Meinung

Vorteile
- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

Nachteile
- Komplex
- Zukunftssicher?
- Anzahl Entwicklungssklaven
- Binärkompatibilität nicht in alle Ewigkeit

We've found that Scala has enabled us to deliver things faster with less code. It's reinvigorated the team.

— Graham Tackley, Guardian

2016-03-11

Scala (Programmiersprache)
└ Scala
└ Spannendes

We've found that Scala has enabled us to deliver things faster with less code. It's reinvigorated the team.

— Graham Tackley, Guardian

Der Guardian ist der Inbegriff für den Technologiewandel in einer klassischen, behäbigen Branche. Lange, bevor Springer wach geworden ist.

Mehr für Nerds

2016-03-11

Scala (Programmiersprache)
└ Scala
  └ Spannendes

Wie gesagt habe ich wenig Ahnung von Scala, da ich es noch nicht produktiv eingesetzt habe. Ich lasse mich aber gerne zwingen, für Interessierte ein Hands on durchzuführen, bei dem wir ein paar Stunden mit Scala in einer echten IDE spielen, nicht auf Folien. Ich würde natürlich auch dafür aus Büchern klauen.

Mehr für Nerds

- Sprecht mich an

2016-03-11

Scala (Programmiersprache)
└Scala
└Spannendes

Mehr für Nerds
- Sprecht mich an

Wie gesagt habe ich wenig Ahnung von Scala, da ich es noch nicht produktiv eingesetzt habe. Ich lasse mich aber gerne zwingen, für Interessierte ein Hands on durchzuführen, bei dem wir ein paar Stunden mit Scala in einer echten IDE spielen, nicht auf Folien. Ich würde natürlich auch dafür aus Büchern klauen.

Mehr für Nerds

- Sprecht mich an
- Hands on-Termin bei Interesse

2016-03-11

Scala (Programmiersprache)
└Scala
└Spannendes

Mehr für Nerds
- Sprecht mich an
- Hands on-Termin bei Interesse

Wie gesagt habe ich wenig Ahnung von Scala, da ich es noch nicht produktiv eingesetzt habe. Ich lasse mich aber gerne zwingen, für Interessierte ein Hands on durchzuführen, bei dem wir ein paar Stunden mit Scala in einer echten IDE spielen, nicht auf Folien. Ich würde natürlich auch dafür aus Büchern klauen.

Mehr für Nerds

- Sprecht mich an
- Hands on-Termin bei Interesse
- Heiko Seeberger: „Durchstarten mit Scala. Tutorial für Einsteiger (2. Aufl.)“

2016-03-11

Scala (Programmiersprache)
└Scala
└Spannendes

Mehr für Nerds

- Sprecht mich an
- Hands on-Termin bei Interesse
- Heiko Seeberger: „Durchstarten mit Scala. Tutorial für Einsteiger (2. Aufl.)“

Wie gesagt habe ich wenig Ahnung von Scala, da ich es noch nicht produktiv eingesetzt habe. Ich lasse mich aber gerne zwingen, für Interessierte ein Hands on durchzuführen, bei dem wir ein paar Stunden mit Scala in einer echten IDE spielen, nicht auf Folien. Ich würde natürlich auch dafür aus Büchern klauen.

2016-03-11

Scala (Programmiersprache)
└Scala
└Spannendes