

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

Worüber reden wir?

IT im Wandel

Worüber reden wir?

IT im Wandel

Herausforderungen

Worüber reden wir?

IT im Wandel

Herausforderungen

Manifeste

Worüber reden wir?

IT im Wandel

Herausforderungen

Manifeste

Scala

Worüber reden wir?

IT im Wandel

Herausforderungen

Manifeste

Scala

Management Summary

Worüber reden wir?

IT im Wandel

Herausforderungen

Manifeste

Scala

Management Summary

Ein wenig Code

Worüber reden wir?

IT im Wandel

Herausforderungen

Manifeste

Scala

Management Summary

Ein wenig Code

Spannendes

IT im Wandel

Herausforderungen

Software Engineering

Software Engineering

Forderungen an IT

Forderungen an IT

- Stabilität und Resilienz

Forderungen an IT

- Stabilität und Resilienz
- Wertbeitrag

Forderungen an IT

- Stabilität und Resilienz
- Wertbeitrag
- Businessstreiber

Forderungen an IT

- Stabilität und Resilienz
- Wertbeitrag
- Businessstreiber

— Matthias Magnor – CEO Surface und Contract Logistics

IT im Wandel

Manifeste

Manifeste

Manifeste

- Antwortbereit, Widerstandsfähig, Elastisch, Nachrichtenorientiert (2013)

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

- 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)

Wo stehen wir?

Scala

Management Summary

Scalable Language

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

Eigenschaften

Eigenschaften

- Objektorientiert

Eigenschaften

- Objektorientiert
- Funktional

Eigenschaften

- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference

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“)
- Ausdrucksstark (APIs/DSLs)

Eigenschaften

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

Versprechen

Versprechen

- Produktivitätssteigerung

Versprechen

- Produktivitätssteigerung
- Höhere Codequalität

Versprechen

- Produktivitätssteigerung
- Höhere Codequalität
- Mehr Spaß

Versprechen

- Produktivitätssteigerung
- Höhere Codequalität
- Mehr Spaß
- durch
- Weniger Code

Versprechen

- Produktivitätssteigerung
- Höhere Codequalität
- Mehr Spaß
- durch
- Weniger Code
- Höheres Abstraktionsniveau

Versprechen

- Produktivitätssteigerung
- Höhere Codequalität
- Mehr Spaß
- durch
- Weniger Code
- Höheres Abstraktionsniveau
- Skalierbarkeit

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
- 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
- Ähnliches Toolset, oft wiederverwendbar

Scala

Ein wenig Code

Eine Java-Klasse

Eine Java-Klasse

```
1 public class Person {  
2     private final String firstName;  
3     private final String lastName;  
4     public Person(String firstName, String lastName) {  
5         this.firstName = firstName;  
6         this.lastName = lastName;  
7     }  
8     public String getFirstName() {  
9         return firstName;  
10    }  
11    public String getLastName() {  
12        return lastName;  
13    }
```

Eine Java-Klasse

```
14  @Override
15  public boolean equals(Object o) {
16      if (this == o) return true;
17      if (o == null || getClass() != o.getClass()) return false;
18      Person person = (Person) o;
19      if (firstName != null ?
20          !firstName.equals(person.firstName) :
21          person.firstName != null) return false;
22      if (lastName != null ?
23          !lastName.equals(person.lastName) :
24          person.lastName != null) return false;
25      return true;
26  }
```

Eine Java-Klasse

```
27  @Override
28  public int hashCode() {
29      int result = firstName != null ? firstName.hashCode() : 0;
30      result =
31          31 * result + (lastName != null ? lastName.hashCode() :
32              0);
32      return result;
33  }
34  }
```

Businesslogik?

Dasselbe in Scala

Dasselbe in Scala

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

Es wird funktional – Quicksort

Es wird funktional – Quicksort

```
1 def quickSort[A <% Ordered[A]](xs: List[A]): List[A] = xs match {  
2   case Nil      => xs  
3   case y :: ys => ys partition (_ <= y) match {  
4     case (l1, l2) => quickSort(l1) ++ (y :: quickSort(l2))  
5   }  
6 }
```

Schnelldurchlauf Scala – Endlich!

Variablen – val und var

Variablen – val und var

```
1 var j = 3  
2 j = 4
```

Variablen – val und var

1 **var** j = 3

2 j = 4

3 **val** k = 3

4 k = 4

Variablen – val und var

```
1 var j = 3  
2 j = 4  
  
3 val k = 3  
4 k = 4
```

Compile-Fehler für k, da immutable

Variablen – val und var

```
1 var j = 3  
2 j = 4  
  
3 val k = 3  
4 k = 4
```

Compile-Fehler für k, da immutable

```
5 val k: Int = 3
```

Methoden

Methoden

```
1 def f = 3 * 2
```

Methoden

1 **def** f = 3 * 2

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

Methoden

1 **def** f = 3 * 2

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

3 **def** f(i: Int) = 3 * i

Methoden

```
1 def f = 3 * 2
```

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

```
3 def f(i: Int) = 3 * i
```

```
4 def f(i: Int) = {  
5     3 * i  
6 }
```

Implizites return

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 }
```

Letzte Anweisung wird zurückgegeben, impliziter Typ String.

Type Inference – Any

Type Inference – Any

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

Type Inference – Any

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

Erste gemeinsame Oberklasse, zur Not Any

Unit

Unit

```
1 def print(s: String): Unit = println(s)
```

Listen

Listen

```
1 val numbers = List(1, 2, 3, 4)
```

Listen

```
1 val numbers = List(1, 2, 3, 4)
```

```
2 val bagContent = List(1, "a", Time(12,30), 4)
```

Listen

```
1 val numbers = List(1, 2, 3, 4)
```

```
2 val bagContent = List(1, "a", Time(12,30), 4)
```

Immutable-Listen bevorzugen

Funktionen höherer Ordnung

Funktionen höherer Ordnung

```
1 val numbers = List(1, 2, 3, 4)
```

Funktionen höherer Ordnung

```
1 val numbers = List(1, 2, 3, 4)
```

```
2 val numbersDecreasing = numbers.sortWith((x, y) => x > y)
```

Funktionen höherer Ordnung

```
1 val numbers = List(1, 2, 3, 4)
2 val numbersDecreasing = numbers.sortWith((x, y) => x > y)
3 val numbersPlusOne = numbers.map(x => x + 1)
```

Funktionen als Typen

Funktionen als Typen

```
1 val numbersPlusOne = numbers.map(x => x + 1)
```

Funktionen als Typen

```
1 val numbersPlusOne = numbers.map(x => x + 1)  
2 val addTwo = (n: Int) => n + 2
```

Funktionen als Typen

```
1 val numbersPlusOne = numbers.map(x => x + 1)
2 val addTwo = (n: Int) => n + 2
3 val numbersPlusTwo = numbers.map(addTwo)
```


Funktionen als Typen

```
1 val numbersPlusOne = numbers.map(x => x + 1)
```

```
2 val addTwo = (n: Int) => n + 2
```

```
3 val numbersPlusTwo = numbers.map(addTwo)
```

```
4 val descending = (n: Int, m: Int) => n > m
```

Funktionen als Typen

```
1 val numbersPlusOne = numbers.map(x => x + 1)
2 val addTwo = (n: Int) => n + 2
3 val numbersPlusTwo = numbers.map(addTwo)
4 val descending = (n: Int, m: Int) => n > m
5 val sortedNumbers = numbers.sortWith(descending)
```

Klassen

Klassen

```
1 class Time(val hours: Int, val minutes: Int) {  
2     // Primärer Konstruktor  
3     require(hours < 24 && hours >= 0)  
4     require(minutes < 60 && minutes >= 0)  
5  
6     def this() = this(0)  
7 }
```

Klassen

```
1 class Time(val hours: Int, val minutes: Int) {  
2     // Primärer Konstruktor  
3     require(hours < 24 && hours >= 0)  
4     require(minutes < 60 && minutes >= 0)  
5  
6     def this() = this(0)  
7 }  
  
8 val t = new Time(1, 14)  
9 val t2 = new Time
```

Benannte Parameter und Standardwerte

Benannte Parameter und Standardwerte

```
1 class Time(val hours: Int = 0, val minutes: Int = 0)
```

Benannte Parameter und Standardwerte

```
1 class Time(val hours: Int = 0, val minutes: Int = 0)

2 val t = new Time(1)

3 val t2 = new Time(minutes = 13)
```


Singleton/Companion Objects

Singleton/Companion Objects

```
1 object Time {  
2     def fromMinutes(minutes: Int): Time =  
3         new Time(minutes / 60, minutes % 60)  
4 }
```

Singleton/Companion Objects

```
1 object Time {  
2     def fromMinutes(minutes: Int): Time =  
3         new Time(minutes / 60, minutes % 60)  
4 }  
  
5 val t = Time.fromMimutes(100)
```

Case classes

Case classes

```
1 case class Person(nachname: String, vorname: String)
```

Case classes

```
1 case class Person(nachname: String, vorname: String)

5 val ich = Person("Eidecker", "Sebastian")

6 val sohn = ich.copy(vorname = "Nils")
```

Vererbung und Traits

Vererbung und Traits

- Geht

Vererbung und Traits

- Geht
- Mehrfachvererbung durch Traits

Vererbung und Traits

- Geht
- Mehrfachvererbung durch Traits
- Zu wenig Zeit heute

Schönere Methodenaufrufe

Schönere Methodenaufrufe

```
1 "Test".startsWith("T")  
2 List(1,2,3).isEmpty
```

Schönere Methodenaufrufe

```
1 "Test".startsWith("T")  
2 List(1,2,3).isEmpty
```

```
1 "Test" startsWith "T"  
2 List(1,2,3) isEmpty
```

Definition eigener Operatoren

Definition eigener Operatoren

```
1 case class Time(hour:Int, minute:Int) {  
2   def minus(time: Time) = {new Time(this.hour - time.hour, this.  
    minute - time.minute)}  
3  
4 }
```

Definition eigener Operatoren

```
1 case class Time(hour:Int, minute:Int) {  
2     def minus(time: Time) = {new Time(this.hour - time.hour, this.  
        minute - time.minute)}  
3  
4 }  
  
5 Time(10,20).minus(Time(1,10))
```


Definition eigener Operatoren

```
1 case class Time(hour:Int, minute:Int) {  
2     def minus(time: Time) = {new Time(this.hour - time.hour, this.  
        minute - time.minute)}  
3  
4 }  
  
5 Time(10,20).minus(Time(1,10))  
6 Time(10,20) minus Time(1,10)
```

Definition eigener Operatoren

```
1 case class Time(hour:Int, minute:Int) {  
2     def minus(time: Time) = {new Time(this.hour - time.hour, this.  
        minute - time.minute)}  
3     def -(time: Time) = minus(time)  
4 }
```

```
5 Time(10,20).minus(Time(1,10))
```

```
6 Time(10,20) minus Time(1,10)
```

Definition eigener Operatoren

```
1 case class Time(hour:Int, minute:Int) {  
2     def minus(time: Time) = {new Time(this.hour - time.hour, this.  
        minute - time.minute)}  
3     def -(time: Time) = minus(time)  
4 }
```

```
5 Time(10,20).minus(Time(1,10))  
6 Time(10,20) minus Time(1,10)  
7 Time(10,20) - Time(1,10)
```

Pattern Matching

Pattern Matching

```
1 val alice = new Person("Alice", 25)
2 val bob = new Person("Bob", 32)
3 val charlie = new Person("Charlie", 32)
4
5 for (person <- List(alice, bob, charlie)) {
6   person match {
7     case Person("Alice", 25) => println("Hallo Alice!")
8     case Person("Bob", 32) => println("Hallo Bob!")
9     case Person(name, age) => println("Alter: " + alter + ", Name:
      " + name)
10  }
11 }
```

Pattern Matching – Funktional Länge einer Liste

Pattern Matching – Funktional Länge einer Liste

```
1 def length[A](list : List[A]) : Int = {  
2   list match {  
3     case _ :: tail => 1 + length(tail)  
4     case Nil => 0  
5   }  
6 }
```

Tupel

Tupel

```
1 val ichMitAlter = (Person("Eidecker", "Sebastian"), 37)
```

Tupel

```
1 val ichMitAlter = (Person("Eidecker", "Sebastian"), 37)
2 val ich = ichMitAlter._1
3 val alter = ichMitAlter._2
```

Tupel

```
1  val ichMitAlter = (Person("Eidecker", "Sebastian"), 37)

2  val ich = ichMitAlter._1
3  val alter = ichMitAlter._2

4  def personMitAlter(person: Person, alter: Int): (Person, Int) = {
5      (person, alter)
6  }
```

Und noch viel, viel mehr ...

Scala

Spannendes

Domain Specific Languages

Domain Specific Languages

- Fachliche Abstraktion

Domain Specific Languages

- Fachliche Abstraktion
- Verständlicher

Domain Specific Languages

- Fachliche Abstraktion
- Verständlicher
- Entwicklung schwierig

Domain Specific Languages

- Fachliche Abstraktion
- Verständlicher
- Entwicklung schwierig
- Einschränkungen vorhanden

Scalatest

```
1 "Creating a Time" should {
2     "throw an IllegalArgumentException for hours less than 0 or
3     greater equal 24" in {
4         forAll("hours") { (hours: Int) =>
5             whenever(hours < 0 || hours >= 24) {
6                 an[IllegalArgumentException] should be thrownBy Time(
7                     hours)
8             }
9         }
10    }
```

Aktoren

Aktoren

- Nebenläufige Einheiten

Aktoren

- Nebenläufige Einheiten
- Empfangen Nachrichten (Ereignisse)

Aktoren

- Nebenläufige Einheiten
- Empfangen Nachrichten (Ereignisse)
- Abarbeitung FIFO

Aktoren

- Nebenläufige Einheiten
- Empfangen Nachrichten (Ereignisse)
- Abarbeitung FIFO
- Verhaltensänderung

Aktoren

- Nebenläufige Einheiten
- Empfangen Nachrichten (Ereignisse)
- Abarbeitung FIFO
- Verhaltensänderung
- Asynchrone Kommunikation mit Aktoren

Akka

Akka

- Aktoren

Akka

- Aktoren
- Fehlertoleranz

Akka

- Aktoren
- Fehlertoleranz
- Standort-Transparenz

Akka

- Aktoren
- Fehlertoleranz
- Standort-Transparenz
- Nachrichten-Persistenz

Akka

- Aktoren
- Fehlertoleranz
- Standort-Transparenz
- Nachrichten-Persistenz
- Reaktiv laut Manifest

Akka – Ping-Aktor

Akka – Ping-Aktor

```
1 case object PingMessage
2 case object PongMessage
3 case object StartMessage
4 case object StopMessage
5
6 class Ping(pong: ActorRef) extends Actor {
7     var count = 0
8     def incrementAndPrint { count += 1; println("ping") }
```

Akka – Ping-Aktor

```
9  def receive = {
10    case StartMessage =>
11      incrementAndPrint
12      pong ! PingMessage
13    case PongMessage =>
14      incrementAndPrint
15      if (count > 99) {
16        sender ! StopMessage
17        println("ping stopped")
18        context.stop(self)
19      } else {
20        sender ! PingMessage
21      }
22  }
23 }
```

Akka – Pong-Aktor

```
19 class Pong extends Actor {  
20     def receive = {  
21         case PingMessage =>  
22             println("  pong")  
23             sender ! PongMessage  
24         case StopMessage =>  
25             println("pong stopped")  
26             context.stop(self)  
27     }  
28 }
```

Bedeutung?

Was haben wir gesehen

Was haben wir gesehen

- Start, Stop, Nachrichten, Zustand

Was haben wir gesehen

- Start, Stop, Nachrichten, Zustand
- Klingt verdächtig nach Prozessen

Was haben wir gesehen

- Start, Stop, Nachrichten, Zustand
- Klingt verdächtig nach Prozessen
- Nebenläufig, skalierbar, resilient durch Akka

Was haben wir gesehen

- Start, Stop, Nachrichten, Zustand
- Klingt verdächtig nach Prozessen
- Nebenläufig, skalierbar, resilient durch Akka
- Mein Traum: Direkte Abbildung

Was haben wir gesehen

- Start, Stop, Nachrichten, Zustand
- Klingt verdächtig nach Prozessen
- Nebenläufig, skalierbar, resilient durch Akka
- Mein Traum: Direkte Abbildung
- Scheint machbar

Was haben wir gesehen

- Start, Stop, Nachrichten, Zustand
 - Klingt verdächtig nach Prozessen
 - Nebenläufig, skalierbar, resilient durch Akka
-
- Mein Traum: Direkte Abbildung
 - Scheint machbar
 - Umgewöhnung

Warum in Scala?

Warum in Scala?

- Pattern Matching

Warum in Scala?

- Pattern Matching
- Funktional

Warum in Scala?

- Pattern Matching
- Funktional
- Weniger Code

Warum in Scala?

- Pattern Matching
- Funktional
- Weniger Code
- Besser verständlich

Scala – Meine Meinung

Scala – Meine Meinung

Vorteile

Scala – Meine Meinung

Vorteile

- Für moderne Architekturen

Scala – Meine Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional

Scala – Meine Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem

Scala – Meine Meinung

Vorteile

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

Scala – Meine Meinung

Vorteile

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

Scala – Meine Meinung

Vorteile

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

Nachteile

Scala – Meine Meinung

Vorteile

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

Nachteile

- Komplex

Scala – Meine Meinung

Vorteile

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

Nachteile

- Komplex
- Zukunftssicher?

Scala – Meine Meinung

Vorteile

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

Nachteile

- Komplex
- Zukunftssicher?
- Anzahl Entwicklungssklaven

Scala – Meine 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

Mehr für Nerds

Mehr für Nerds

- Sprecht mich an

Mehr für Nerds

- Sprecht mich an
- Hands on-Termin bei Interesse

Mehr für Nerds

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

Mehr für Nerds

- Sprecht mich an
- Hands on-Termin bei Interesse
- Heiko Seeberger: „Durchstarten mit Scala. Tutorial für Einsteiger“
- Martin Odersky: „Programming in Scala“

