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
- Businesstreiber

Forderungen an IT

- Stabilität und Resilienz
- Wertbeitrag
- Businesstreiber

— Matthias Magnor – CEO Surface und Contract Logistics

IT im Wandel

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?

Wo stehen wir im Wettbewerb?

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 schreiben)

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
- Höhere Codequalität

Versprechen

- Produktivitätssteierung
- Höhere Codequalität
- Mehr Spaß

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
- 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
- Ähnlicher Paketierungs- und Buildprozess (sbt)

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

```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?

Dasselbe in Scala

Dasselbe in Scala

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

Es wird funktional – Quicksort

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))
  }
}
```

val und var – Immutables

val und var – Immutables

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

val und var – Immutables

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

Compile-Fehler, da i immutable

Funktionen – Benannte Parameter

Funktionen – Benannte Parameter

```
1     class
```

Schönere Methodenaufrufe

Schönere Methodenaufrufe

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

Schönere Methodenaufrufe

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

```
"Test" startsWith "T"
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 }
```

```
1 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 }
```

```
1 Time(10,20).minus(Time(1,10))
2 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 }
```

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

Definition eigener Operatoren

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

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

Klassen und Objekte

Klassen und Objekte

```
1    class
```

Listen

Listen

```
1     class
```

Pattern Matching

Pattern Matching

```
1     class
```

Type Inference

Type Inference

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

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 II

Type Inference II

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

Erste gemeinsame Oberklasse, zur Not Any

Vererbung und Traits

Vererbung und Traits

```
1       class
```

Funktionen funktional – Lambdas schön

Funktionen funktional – Lambdas schön

```
1       class
```

Flatmap that shit!

Flatmap that shit!

```
1       class
```

Arbeit mit Strings

Arbeit mit Strings

```
1       class
```

Tupel

Tupel

```
1     class
```

Implicits

Implicits

```
1     class
```

??? – Mein heimlicher Star

??? – Mein heimlicher Star

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

??? – Mein heimlicher Star

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

Kompilierbar, aber nicht gefährlich.

Scala

Spannendes

Aktoren

Aktoren

- Scalable real-time transaction processing

Aktoren

- Scalable real-time transaction processing
- Nebenläufige Berechnungen

Aktoren

- Scalable real-time transaction processing
- Nebenläufige Berechnungen
- Kommunikation ausschließlich über Nachrichten

Aktoren

- Scalable real-time transaction processing
- Nebenläufige Berechnungen
- Kommunikation ausschließlich über Nachrichten
- Elastizität und Resilienz

Aktoren

- Scalable real-time transaction processing
- Nebenläufige Berechnungen
- Kommunikation ausschließlich über Nachrichten
- Elastizität und Resilienz
- Einfaches Mapping auf Prozessdefinitionen

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

DSL – Beispiel

```
100 PRINT "Distance " % 'dist % "km, " % "Velocity " % 'v % "km/s,
    " % "Fuel " % 'fuel
110 INPUT 'burn
120 IF ABS('burn) <= 'fuel THEN 150
130 PRINT "You don't have that much fuel"
140 GOTO 100
150 LET ('v := 'v + 'burn * 10 / ('fuel + 'mass))
```

DSL sinnvoll – ScalaTest

DSL sinnvoll – ScalaTest

- Fachlich verständliche Tests

DSL sinnvoll – ScalaTest

- Fachlich verständliche Tests
- Testdatengenerierung

Scalatest – Beispiel

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

Meine wenig qualifizierte Meinung

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
- 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
- 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
- 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
- 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 (2. Aufl.)“