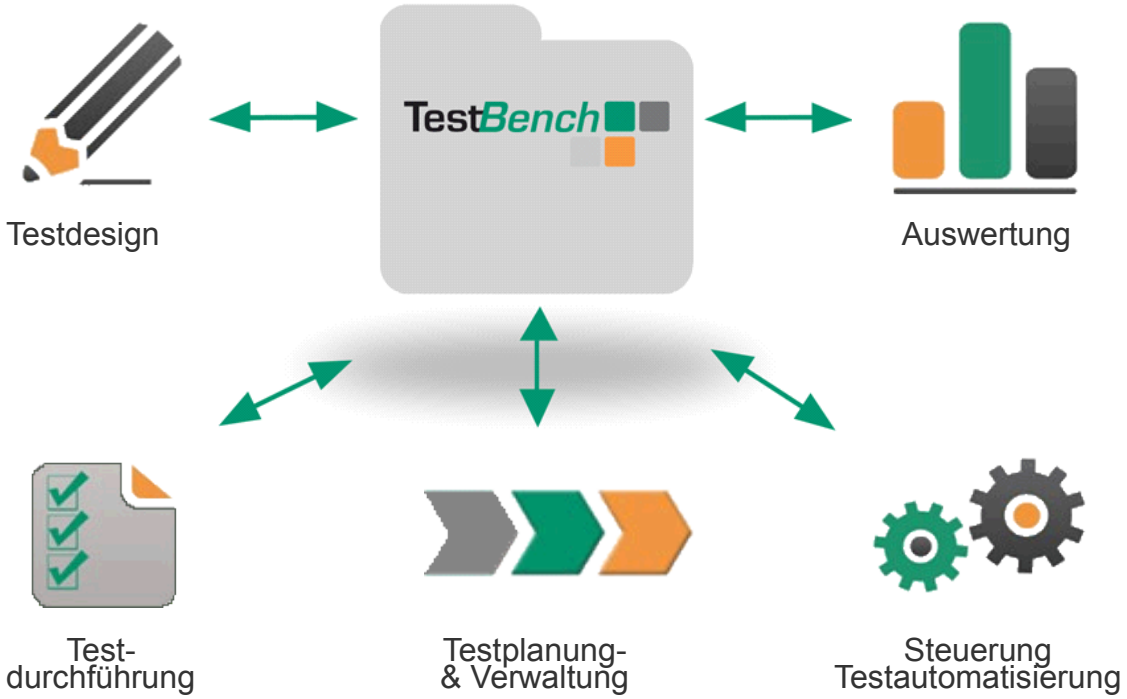
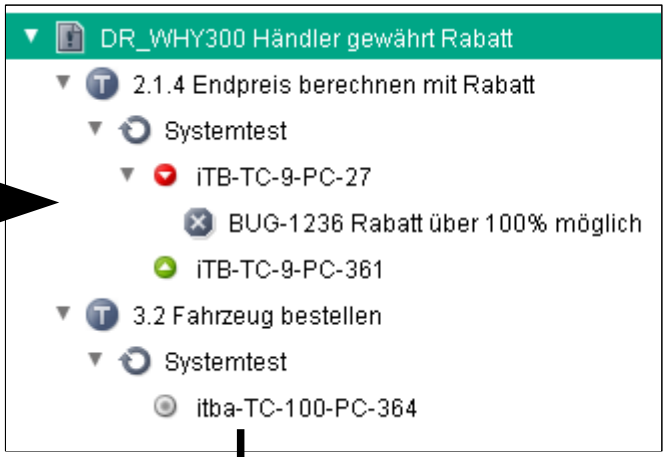
Test*Bench*: Ein Projekt steigt um von Java auf Scala

Georg Dietrich

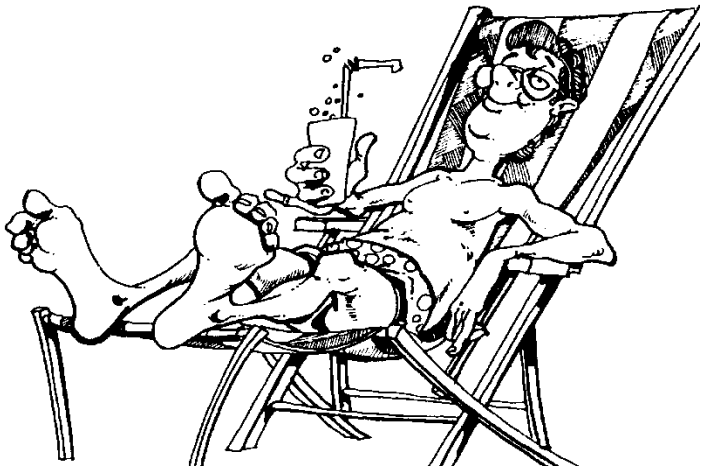
georg.dietrich (at) imbus.de



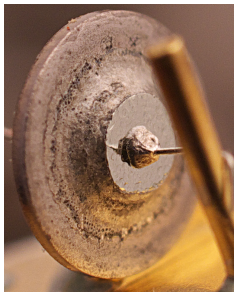


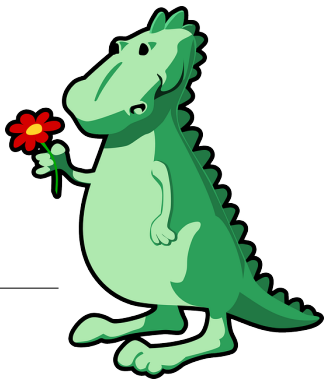
Aktionen und erwartete Reaktionen

	Testsequenz	1. Parameter	Bemerkung
1	Test Elements.Interactions.starte CarConfigurator		
2	Test Elements.Interactions.wähle Fahrzeugtyp	fahrzeug	
3	Test Elements.Interactions.wähle Zusatzausstattung	zubehörliste	
4	Test Elements.Interactions.gewähre Rabatt	rabatt	
5	Test Elements.Interactions.beende CarConfigurator		



- 2060: ???
- 2014-01: Scala 2.10.3; Test*Bench* 2.3.1
- 2011-12: Scala IDE for Eclipse 2.0
- 2011-11: Scala in Test*Bench*
- 2011-07: Java 7
- 2011-05: Scala 2.9
- 2004-01: Scala; Test*Bench* (Java 1.4)
- 1995: Java
- 1992: imbus AG
- 1972: Ich
- 1945: Transistor
- 100.000 v. Chr.: Dinosaurier





2011, nach sieben Jahren Test*Bench*-Entwicklung in Java ...

451.000 Java

Codezeilen

130.000
XSL & XML

10.700 SQL

25.800 Rest
(12 Sprachen)

TestBench

Java als Sprache

relativ alt

extrem
konservativ

viel
Boilerplate

Schwächen im
Typsystem

fehlerträchtige
Mutability

Keine Closures
Keine Lambdas

Schlecht geeignet
für funktionale
Programmierung

2011, nach sieben Jahren Test*Bench*-Entwicklung in Java ...

Typinferenz

Erweiterbar (DSLs)

Collections

kaum Boilerplate

Scala als Sprache

ausgefeiltes Typsystem

kein null

OOP / FP: Vorteile beider Welten

programmieren ohne „var“

Beispiel SOAP-Client:
1 var in 85 kB Quellcode

TestBench

relativ alt

extrem konservativ

Java als Sprache

viel Boilerplate

Schwächen im Typsystem

fehlerträchtige Mutability

Keine Closures
Keine Lambdas

Schlecht geeignet für funktionale Programmierung

„Boilerplate“
http://www.flickr.com/photos/prettyuglydesign

Was ist Boilerplate Code?

Beispiel:
Invertieren einer Map<A,B> zu Map<B,A>

(von Hand – es gibt dafür natürlich auch Libraries)

```java
public static final <K, V> Map<V, K> invertMap(Map<K, V> map) {
    final Map<V, K> inverted = new HashMap<>(map.size());
    for (Map.Entry<K, V> entry : map.entrySet())
        inverted.put(entry.getValue(), entry.getKey());
    return Collections.unmodifiableMap(inverted);
}

public static final Map<String, Integer> wordToInteger = invertMap(integerToWord);
```

```scala
val wordToInt = intToWord map { _ swap }
```

Alle Code-Beispiele:
https://bitbucket.org/ugdietrich/javaland2014

Reaktionen des Teams

Zweifel

Begeisterung

Alltag

http://reynaruina.deviantart.com/art/Vector-Manga-Avatar-Faces-250742792

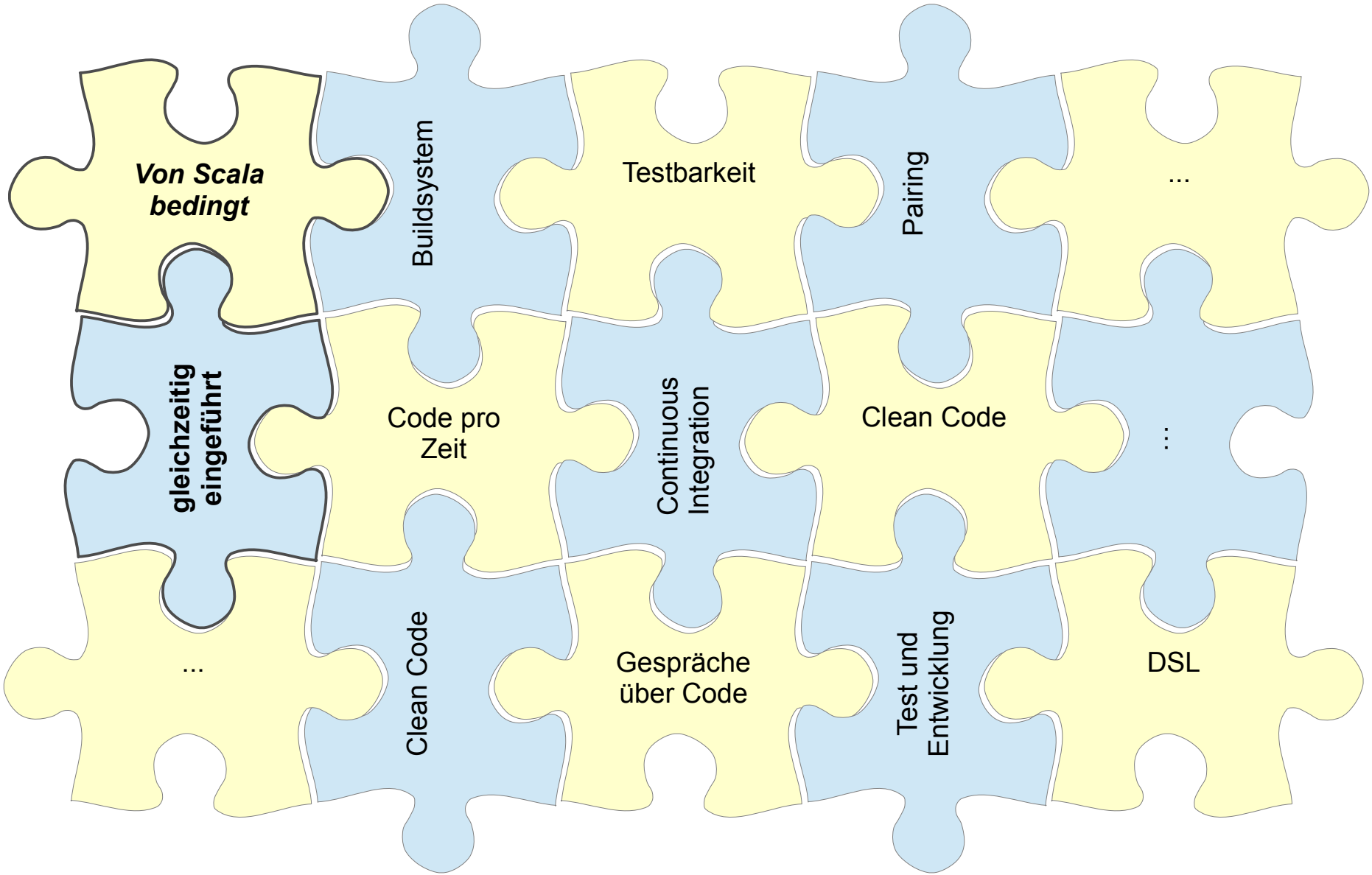
Eine neue Sprache muss man lernen...

```
"abc/def".lastIndexOf('/') + 1                    > res0: Int =  4

"abc/def" lastIndexOf '/'  + 1                    > res1: Int = -1

("abc/def" lastIndexOf '/') + 1                   > res2: Int =  4
```

```
case class Issue(id: String, title: String, description: String)
```

- Äquivalent zu einem POJO
- Autogenerierte Getter
- Immutable, deshalb keine Setter
- Vernünftig autogenerierte equals und hashCode Methode
- Vernünftig autogenerierte toString Methode
- Autogenerierte copy-Methoden
- Instanziierung ohne „new“
- Pattern matching

Unterschiede im Alltag



Einfach asynchron ...

Aus einem Issue Tracking System sollen mit Paging alle Items ausgelesen werden.

Erster Ansatz: Rekursiv, synchron.

```
def items: List[Item] = {
  @annotation.tailrec
  def pageRecursion(pageStart: Long, results: List[Item]): List[Item] = {
    itemsPage(pageStart, pageSize) match {
      case List() => results
      case items => pageRecursion(pageStart + pageSize, results ++ items)
    }
  }
  pageRecursion(1, List())
}
```

… Dauer: 6380ms

Verbesserter Ansatz: Paralleles Lesen asynchron.

```
def itemsAsync: Future[List[Item]] =
  for {
    count <- numberOfItemsAsync
    pageStarts = 1 to count by pageSize
    futurePages = pageStarts.toList map( itemsPageAsync(_, pageSize) )
    pages <- Future sequence futurePages
  } yield pages.flatten
```

… Dauer: 1650ms

Zahlen aus der Entwicklung

	12.12.2011	22.05.2013	11.03.2014
.java	451.000	443.000	460.000
.scala	2.770	10.800	25.600
Rest	168.000	288.000	290.000

Codezeilen (NLOC) von Main und Test ohne Test-Ressourcen

SELECT id, parent, name FROM Tree WHERE parent = ?

is a modern database query and access library for Scala.

```
def queryChildren(parent: Option[TreeEntryID]) =
  TableQuery[Tree] filter (_.parent === parent) list

println(queryChildren(Some(TreeEntryID(1))) mkString "\n")
```

```
TreeEntry(TreeEntryID(2),Some(TreeEntryID(1)),Bestellung)
TreeEntry(TreeEntryID(703),Some(TreeEntryID(1)),Fotos)
TreeEntry(TreeEntryID(61010),Some(TreeEntryID(1)),Sonstiges)
```

Selbst die Tabellendefinition ist Scala-Code:

```
class Tree(tag: Tag) extends Table[TreeEntry](tag, "TREEENTRIES") { import Tree._
  def id      = column[TreeEntryID]          ("ID", O.PrimaryKey)
  def parent  = column[Option[TreeEntryID]] ("PARENT")
  def name    = column[String]               ("NAME", O.DBType("VARCHAR(256)"))
  def *       = (id, parent, name) <> (TreeEntry.tupled, TreeEntry.unapply)
}
```

```
SELECT id, parent, name FROM Tree WHERE parent = ?
```

… hat auch ganz eigene Herausforderungen …

```
private def queryChildren(parent: Option[TreeEntryID]) =
  Tree.tableQuery filter (_.parent === parent)

private val compiledQuery = Compiled(queryChildren _)
```

```
[error] Computation of type Option[TreeEntryID] =>
scala.slick.lifted.Query[TreeDemo.TreeEntries,TreeDemo.TreeEntries#TableElementType]
cannot be compiled (as type C)
```

Wie bei jeder Sprache muss man bei sich auch bei einer Domain Specific Language an die Besonderheiten gewöhnen. Hier die Korrektur – strikt nach Slick-Dokumentation:

```
private def queryChildren(parent: Column[Option[TreeEntryID]]) =
  Tree.tableQuery filter (_.parent === parent)

private val compiledQuery = Compiled(queryChildren _)

def children(parent: TreeEntryID): List[TreeEntry] =
  queryChildren(Some(parent)) list
```

Noch mehr Zahlen aus der Entwicklung

	SOAP DM Integration Java	SOAP DM Integration Scala
Dateien	13	10
Zeilen Main-Code	3858	1205
Utility-Dateien	7	-
Zeilen Utility Main-Code	1353	-

Codezeilen (NLOC) von Main ohne Test

1 var in 85 kB Quellcode

… wird wegen Java-Interoperabilität benötigt.

Test: Beispiel für Schönheit

http://specs2.org

```
class ExampleSpec extends SpecBasics { def is = s2"""
  ${s"Results of issue synchronization" title}

  With the example data, during synchronization
    one new issue should be created $issuesCreated
    one issue should be deleted $issuesDeleted
    two issues should be updated $issuesUpdated
    one issue should be imported $issuesImported
"""

  lazy val result = IssueManagementStub synchronizeIssues exampleData

  def issuesCreated =
    (result.created should haveExactlyDescriptions("sense")) and
    (result.protocol should contain("created issue 006"))

  def issuesDeleted = todo
  def issuesUpdated = todo
  def issuesImported = todo
}
```

Test: Beispiel für (zu?) viel Aufwand für Schönheit

http://specs2.org

```
def haveExactlyDescriptions(descriptions: String*): Matcher[Issues] =
  contain(exactly(descriptions:_*)) ^^
  { issues: Issues => issues map (_ description) aka "the issue descriptions" }

def beHttpReference(httpSuffix: String): Matcher[String] =
  beMatching("""http\:\/\/.*""" + Pattern.quote(httpSuffix) + "/") ^^
  { (_: String) aka s"the HTTP reference that should end with $httpSuffix/" }

def haveOnlySomeHttpReferences(httpSuffixes: List[String]): Matcher[Option[Issue]] =
  beSome(contain(exactly((httpSuffixes map beHttpReference): _*)) ^^
  { (_: Issue).references })
```

… die Dinge richtig tun
… die richtigen Dinge tun
… die richtigen Dinge richtig tun

Scala: Die beste aller möglichen Welten?

Wenn man keine Probleme mit der Programmiersprache hat,
dann heißt das nicht,
dass man keine Probleme hat.

Einige Probleme von Scala:

- Binärkompatibilität von Bibliotheken mit spezifischen Scala-Versionen
- Typinferenz an Schnittstelle OOP/FP nicht perfekt (→ Boilerplate Code)
- Erasure von Generics
- Spezialisierung generischer Typen ist umständlich

Fazit

Sicht des Projektmanagements:

- Mit Scala fällt es leichter, über lange Zeiträume hinweg eine hohe Code-Qualität zu erhalten.

Sicht des Teams:

- Uns macht programmieren in Scala mehr Spaß als in Java.
- Es kann noch besser werden – und dafür müssen wir uns auch selbst mit einsetzen.

Einer unserer Entwickler soll wieder Java statt Scala schreiben:

Als erstes muss ich in Java wieder mental den ganzen Boilerplate-Code ausblenden um die Stelle zu finden, an der das passiert, was mich interessiert. [...] In Scala ist es allerdings schon so, dass man mit der funktionalen Schreibweise oft länger braucht um zu verstehen, worum es geht – in Java ist das einfacher. Zumindest für einen Abschnitt von X Zeilen.