5.– 8. September 2011
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Einfacher bauen

SBT im Vergleich zu Ant, Maven & Co.

Joachim Hofer

imbus AG

„Zeige mir, wie du baust, und ich sage dir, wer du bist.“
– Christian Morgenstern

Kurzvorstellung

- 10 Jahre Teamleiter Softwareentwicklung
- 10 Jahre zu viel Erfahrung mit Enterprise Java
- Schwerpunkte Softwaretest und Code Coverage
- **Scala-Enthusiast**
- Autor von eCobertura (schlechte EclEmma-Konkurrenz)
- Autor diverser **sbt-Plugins**
  - findbugs4sbt
  - cobertura4sbt
  - cpd4sbt

Fahrplan

- „Historische“ Wurzeln: make, Ant, Maven
- Moderne Alternativen: Buildr, Gradle, sbt
- Umbruch: sbt 0.7.x vs sbt 0.10.x
- Einstieg in sbt
- Anpassen von sbt
- Erweitern von sbt
- Fazit

Fahrplan

- **„Historische“ Wurzeln**
  - **make**
  - **Ant**
  - **Maven**
- Moderne Alternativen: Buildr, Gradle, sbt
- Umbruch: sbt 0.7.x vs sbt 0.10.x
- Einstieg in sbt
- Anpassen von sbt
- Erweitern von sbt
- Fazit

„Historische“ Wurzeln (0): **make**

Aaaaargh!

„Historische“ Wurzeln (1): **Ant**

- „Another Neat Tool“
  - James Duncan Davidson, 1999
- XML-basiert
- JVM als Engine
- große Bibliothek an Tasks

- **keine** Vorgaben bzgl Projektstruktur
- **kein** Dependency Management
  - üblicherweise: Ergänzung durch Ivy

„Historische“ Wurzeln (2): **Maven**

- „Accumulator of Knowledge“, seit 2003
- wie Ant:
  - XML-basiert (oder: Polyglot Maven)
  - JVM als Engine
  - große Bibliothek an Tasks
- aber:
  - strikt deklarativ
  - **Convention over Configuration**
  - eigenes Dependency Management

Intermezzo: XML ist doof!

- Buildfiles sollen **lesbar** sein
  - XML ist zu verbos
  - deklarative Alternativen: YAML, JSON

„It's not meant to be a nice language for humans“
– Steve Loughran (über Ant)

- Builds sind **Skripte**
  - Overhead für deklarative Builds hoch
  - Domain-Specific Language (DSL) für Builds
  - Skriptsprachen

Fahrplan

- „Historische“ Wurzeln: make, Ant, Maven
- **Moderne Alternativen**
  - **Buildr**
  - **Gradle**
  - **sbt**
- Umbruch: sbt 0.7.x vs sbt 0.10.x
- Einstieg in sbt
- Anpassen von sbt
- Erweitern von sbt
- Fazit

Moderne Alternativen (1): **Buildr**

- seit 2007
- wie Maven
- aber: **Ruby** als Build-DSL

- gute Scala-Unterstützung
- gute Dokumentation

```
define 'killer-app' do
  project.version = '0.1.0'
  compile.with 'commons-cli:commons-cli:jar:1.2'
  test.with 'commons-collections:commons-collections:jar:3.2'
  test.using :testng
  package :jar
end
```

Moderne Alternativen (2): **Gradle**

- seit 2008, JVM-basiert
- wie Maven
- aber: **Groovy** als Build-DSL

- gute Dokumentation

> *„I can honestly say if someone had shown me the Programming in Scala book by Martin Odersky, Lex Spoon & Bill Venners back in 2003 I'd probably have never created Groovy.“*
> *– James Strachan*

Moderne Alternativen (3): **sbt**

- „Simple Build Tool“, seit 2008, von Mark Harrah
- wie Maven
- aber: **Scala** als Build-DSL

- sehr gute Scala-Unterstützung
- (bisher) mäßige Dokumentation
- interaktiver Modus
- Bestandteil des Typesafe-Stacks

Typesafe

Fahrplan

- „Historische“ Wurzeln: make, Ant, Maven
- Moderne Alternativen: Buildr, Gradle, sbt
- **Umbruch: sbt 0.7.x vs sbt 0.10.x**
  - **sbt 0.7.x**
  - **sbt 0.10.x: Der Umbruch**
  - **sbt 0.10.x: Die Änderungen**
- Einstieg in sbt
- Anpassen von sbt
- Erweitern von sbt
- Fazit

sbt 0.7.x – der Anfang

- Fokus auf **Einfachheit**
  - einfach bedienbar
  - (relativ) einfach gemacht
- Build-Definition in Scala
  - → `project/build/Project.scala`

sbt 0.7.x – die Probleme

- wenig DSL-Elemente (viele `overrides`, `vals`)
- Dependencies werden pro Projekt heruntergeladen
- Build-Engine intern
  - objektorientiert
  - basierend auf Traits

sbt 0.10.x (1): Der Umbruch

- Umstellung 2010/2011
- erster Release (0.10.0) am 2. Juni 2011

→ *Vergessen Sie alles, was Sie über sbt wissen!*

→ *Schreiben Sie all Ihre Buildskripte neu!*

→ *Schreiben Sie all Ihre Plug-ins neu!*

- Lohnt sich das denn?

sbt 0.10.x (2): Die Änderungen

- Build-Definition weniger versteckt
  - `build.sbt` (einfache Builds)
  - oder: `project/Build.scala` (komplexe Builds)
- etwas mehr DSL-Elemente
- Build-Engine intern
  - funktional
  - eigenes Settings-Framework
- → leider nicht mehr ganz so einfach

Fahrplan

- „Historische“ Wurzeln: make, Ant, Maven
- Moderne Alternativen: Buildr, Gradle, sbt
- Umbruch: sbt 0.7.x vs sbt 0.10.x
- **Einstieg in sbt**
- Anpassen von sbt
- Erweitern von sbt
- Fazit

Fahrplan – Einstieg in sbt

- Dokumentation
- Installation
- Erstes Projekt
- Beispiel-Settings
- Zuweisungs-Syntax
- Befehle
- Testen
- Tipps & Tricks

Einstieg in sbt (1): Dokumentation

- Projekt `harrah/xsbt` auf GitHub:
  - Wiki
  - API-Dokumentation
  - Quelltexte, speziell Keys.scala
- Mailing-Liste via Google Groups
- Interaktive Hilfe:
  - `help`
  - `tasks`
  - `commands`

Einstieg in sbt (2): Installation

- Download von `sbt-launch.jar`
- Schreiben eines Aufruf-Skripts
- keine gesonderte Scala-Installation nötig

Einstieg in sbt (3): Erstes Projekt

- `sbt` im Projektverzeichnis
- Setzen der Projekteigenschaften
  - interaktiv: `set` und `session save`
  - per `build.sbt`

Einstieg in sbt (4): Beispiel-Settings

- Beispiele für Projekteigenschaften
  - Basisinformationen: `name`, `version`, `organization`
  - Scala-Version: `scalaVersion`
  - Abhängigkeiten: `libraryDependencies`, `resolvers`

  - konfigurations- bzw taskabhängig:
    - `parallelExecution in Test`
    - `logLevel in compile`
    - `publishArtifact in (Compile, packageDoc)`

Einstieg in sbt (5): Zuweisungs-Syntax

- einfache Zuweisung: :=
  - `name := „FirstProject“`
- Hinzufügen zu Listen: += und ++=
  - `scalacOptions += "-deprecation"`
  - `javacOptions ++= Seq(`
    `"-source", "1.5", "-target", "1.5")`
- Bezugnehmen auf existierende Settings: Präfix mit <
  - `libraryDependencies <+= scalaVersion(`
    `"org.scala-lang" % "scala-compiler" % _ )`
  - `watchSources <+= baseDirectory map { _ / "input" }`

Einstieg in sbt (6): Befehle

- übliche den Stages entsprechende Befehle:
  - `update`, `clean`, `compile`, `test`, `package`, `publish` etc
- mehrere Befehle hintereinander:
  - `; reload ; update ; compile`
- durch Quelltextänderung getriggert:
  - `~ test`

Einstieg in sbt (7): Testen

- Java – Einbinden von JUnit:
  - `"com.novocode" % "junit-interface" % "0.6" % "test"`
  - Testklassen nach `src/test/java`
- Scala – Specs:
  - `"org.specs2" %% "specs2" % "1.6" % "test"`
  - Spezifikationen nach `src/test/scala`
  - zusätzliche Optionen siehe zB `asflierl/eulersolutions`
- Laufen lassen zB mit
  - `test`
  - `test-only <Klasse>`
  - `~test`

Einstieg in sbt (8): Tipps & Tricks (1)

- Interaktiv arbeiten!
  - Startup-Zeiten vermeiden
    - JVM
    - Scala-Compiler
  - getriggerten Compile/Test nutzen
  - aber:
    - hierzu Classunloading einschalten, PermSize erhöhen
    - `-XX:+CMSClassUnloadingEnabled -XX:MaxPermSize=256m`

Einstieg in sbt (9): Tipps & Tricks (2)

- `console-project` für REPL in Buildumgebung
- `inspect` zeigt Zusammenhänge
- ivy-Home setzen: `-Dsbt.ivy.home=/my/ivy/home`

Fahrplan

- „Historische“ Wurzeln: make, Ant, Maven
- Moderne Alternativen: Buildr, Gradle, sbt
- Umbruch: sbt 0.7.x vs sbt 0.10.x
- Einstieg in sbt
- **Anpassen von sbt**
  - **Build-Definition in Scala**
  - **Plug-ins (Auswahl)**
  - **Plug-ins einbinden**
- Erweitern von sbt
- Fazit

Anpassen von sbt (1): Build in Scala

- Rahmen schaffen
  - `object MyBuild extends Build`
- Projekt definieren
  - `lazy val project = Project(`

    `"name", file("."), settings = mySettings)`
- Settings einbinden
  - `val mySettings = Defaults.defaultSettings ++ Seq (`

    `organization := buildOrganization,`

    `version := buildVersion,`

    `libraryDependencies ++= myDependencies,`

    `...)`
- ggf eigene Anpassungen vornehmen

Anpassen von sbt (2): Plug-ins (Auswahl)

- IDE-Integration: IDEA, NetBeans, Eclipse
- Web-Plugins: Webstart, GWT, CloudBees...
- Code Coverage: Cobertura, SCCT
- Statische Analyse: FindBugs (nur Java), CPD
- Sonstiges: Android, Cucumber, Growl, LWJGL...

- siehe Plugin-Übersicht im Wiki
- täglich (naja) werden es mehr...

Anpassen von sbt (3): Plug-ins einbinden

- Abhängigkeit einfügen (evtl auch Resolver)
  - `libraryDependencies +=`

    ```
    "com.github.retronym" %% "sbt-xjc" % "0.3"
    ```
  - globale Plug-ins in `~/.sbt/plugins/build.sbt`
  - projektlokale Plug-ins in `project/plugins/build.sbt`

- ggf Settings importieren
  - `seq(findbugsSettings : _*)`

Fahrplan

- „Historische“ Wurzeln: make, Ant, Maven
- Moderne Alternativen: Buildr, Gradle, sbt
- Umbruch: sbt 0.7.x vs sbt 0.10.x
- Einstieg in sbt
- Anpassen von sbt
- **Erweitern von sbt**
  - **Grundlagen zu Plug-ins**
  - **Command Plug-ins**
  - **Task Plug-ins**
- Fazit

Erweitern (1): Grundlagen zu Plug-ins

- minimal: eingebundene JAR
  - manuell in `project/plugins/lib` oder via Ivy
- „echte“ Plug-ins
  - in `build.sbt`: „`sbtPlugin := true`“
  - abgeleitet von `sbt.Plugin`
  - können Settings definieren

Erweitern (2): Grundlagen – Plugin-Arten

- Command Plug-ins
  - definieren zusätzliche Befehle (zB „`eclipse`“)
- Task Plug-ins
  - definieren zusätzliche Build Tasks (zB „`instrument-code`“)

Erweitern (3): Command Plug-ins

- Aufgebaut aus:
  - Action: `val action: (State, T) => State = ...`
  - Parser: `val parser: State => Parser[T] = ...`
  - Command:

```
val command: Command = Command("name")(parser)(action)
```

- Command definieren:
  -
```
def hello = Command.command("hello") { state =>
    println("Hi!"); state }
```

- Command zu bestehenden `commands` hinzufügen:
  - `override lazy val settings = Seq(commands += hello)`
- Beispiel: `typesafehub / sbteclipse`

Erweitern (4): Task Plug-ins

- Task(schlüssel) deklarieren:
  - `val hello = TaskKey[Unit]("hello", "<Task Description>")`
- Task definieren:
  - `def helloTask = { println("Hello World") }`
- Task einbinden:
  - `override lazy val settings = Seq(hello := helloTask)`
- Task mit Abhängigkeiten:
  - `hello <<= hello dependsOn otherTask`
- Task mit Settings:
  - ```
    private def someTask = { (baseDirectory, name) map {
      (baseDir, name) => doSomething(baseDir, name)
    } }
    ```
    ```

Fahrplan

- „Historische“ Wurzeln: make, Ant, Maven
- Moderne Alternativen: Buildr, Gradle, sbt
- Umbruch: sbt 0.7.x vs sbt 0.10.x
- Einstieg in sbt
- Anpassen von sbt
- Erweitern von sbt
- **Fazit**

Fazit zu sbt (1)

- aus Java-Sicht: sbt ist Magie, Finger weg!
  - → Pragmatic Programmer –
    „Lerne jedes Jahr eine neue Programmiersprache!“
- mit Scala-Kenntnissen eine gute Alternative
- für Scala-Projekte klar gesetzt

Fazit zu sbt (2)

- nicht mehr ganz „simple“
- Änderungsrate noch hoch
- es bleibt spannend: sbt 0.11.x ist bereits auf dem Weg
  - Scala 2.9.1 (statt 2.8.1) als Basis für den Build
  - besserer Bezug Plugin-Version ↔ sbt-Version
  - ???

Vielen Dank!

Joachim Hofer
@johofer
imbus AG

imbus AG

Spezialisierter Lösungsanbieter für Software-Qualitätssicherung und Software-Test

Innovativ seit 1992

Erfahrung und Know-how aus über 3.000 erfolgreichen Projekten

180 Mitarbeiter an vier Standorten in Deutschland

Beratung, Test-Services, Training, Tools, Datenqualität

Für den gesamten Software-Lebenszyklus

www.imbus.de