embarc Software Consulting GmbH

# Architekturüberblick Gradle

**STEFAN ZÖRNER, EMBARC**
Stefan.Zoerner@embarc.de | @StefanZoerner

## Über diese Folien

- In unserem Architektur-Spicker #1 („Der Architekturüberblick“) schlagen wir u.a. eine **Struktur für Folienvorträge** vor.
- Diese Folien sind eine **„Beispielbefüllung“** dieser Gliederung für das Build-System **Gradle**.
- Die Inhalte stammen größtenteils aus meinen Blog-Beiträgen bei Hanser-Update („**Starschnitt** Gradle“)


*Stefan Zörner, Oktober 2015*

Der Spicker #1 kostenlos als PDF:
➔ http://www.embarc.de/spicker

Hanser-Update Blog
➔ http://update.hanser-fachbuch.de

# Agenda

1 Aufgabenstellung
2 Lösungsstrategie
3 Die Lösung im Detail
4 Weitere Informationen



# Agenda

1 **Aufgabenstellung**
2 Lösungsstrategie
3 Die Lösung im Detail
4 Weitere Informationen

1

# Was ist Gradle?

Gradle ist fortentwickelte Build-Automatisierung.

- Es automatisiert das Bauen, Testen und Ausliefern von Software.
- Gradle ist für den Unternehmenseinsatz konzipiert.
- Es kombiniert die Mächtigkeit und Flexibilität von Ant mit dem Abhängigkeitsmanagement und den Konventionen von Maven.
- Hocheffiziente Builds steigern die Produktivität im Team.
- Bestehende Builds lassen sich leicht auf Gradle umstellen.
- Gradle-Builds lassen sich sogar dort ausführen, wo Gradle nicht installiert ist.



# Wesentliche Features von Gradle


- Open Source, lizensiert unter der Apache Software License
- Java-basiert
- unterstützt Java-, Scala-, Groovy-Projekte und vieles mehr
- erweiterbar durch exzellent dokumentierte API
- Einbindung in gängige Entwicklungs-umgebungen (IDEs) und Continous Integration (CI) Server
- Offizielles Build-System für Android Studio

# Architekturtreiber Gradle (1/2)

Im Folgenden die zentralen Qualitätsziele von Gradle mit kurzen Erläuterungen.

| | |
|---|---|
| Erweiterbarkeit | Gradle lässt sich leicht um neue Funktionalität erweitern. Es kann auf lange Sicht dem technologischen Fortschritt bei Tools und Entwicklungsmethodik folgen. |
| Effizienz | Teams und einzelne Entwickler erhalten durch kurze Buildzeiten schnelle Rückmeldungen; Gradle steigert so ihre Produktivität. |
| Interoperabilität | Gradle arbeitet mit bestehenden Werkzeugen wie Ant und Maven und deren Öko-Systemen nahtlos zusammen. ➔ . . . |



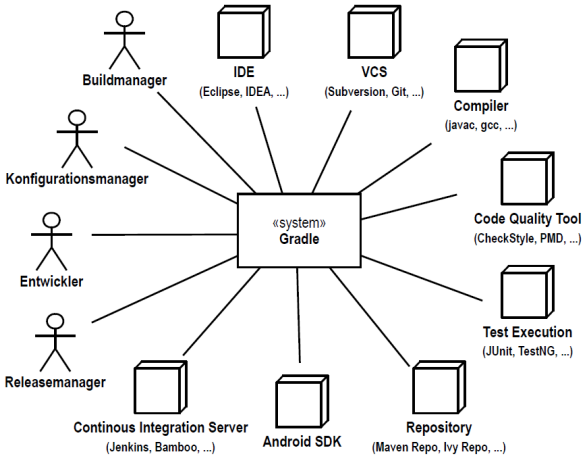
# Architekturtreiber Gradle (2/2)

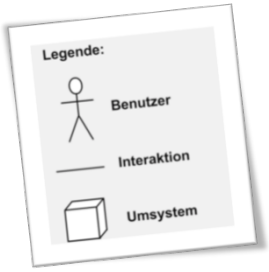
Fortsetzung der Ziele. Die Reihenfolge gibt eine grobe Orientierung bezüglich der Wichtigkeit vor.

| | |
|---|---|
| Erlernbarkeit | Entwickler und Buildmanager finden sich schnell in Gradle zurecht, der Einstieg und das Erstellen erster Build-Skripte fallen ihnen leicht. |
| Installierbarkeit | Der Aufwand, der zum Installieren von Gradle notwendig ist, um einen Build auszuführen, ist sehr gering. |
| Skalierbarkeit | Gradle bleibt auch bei sehr umfangreichen Builds und in Multi-Projekt-Szenarien handhabbar und effizient. |

# Kontextabgrenzung

Das Diagramm zeigt Gradle im Zusammenspiel mit wichtigen Akteuren.




Die dargestellten Umsysteme stellen oft Kategorien dar; in Klammern sind beispielhafte Vertreter des Werkzeugtyps notiert.



# Agenda

2

1 Aufgabenstellung
2 **Lösungsstrategie**
3 Die Lösung im Detail
4 Weitere Informationen

# Lösungsstrategie

Im Folgenden stellen wir den Qualitätszielen die Architekturansätze von Gradle gegenüber, die diese adressieren. Wir schlagen so eine Brücke zwischen Aufgabenstellung und Lösung.


Erweiterbarkeit


Effizienz


Interoperabilität


Erlernbarkeit


Installierbarkeit


Skalierbarkeit

- ...
- ...
- ...



# Lösungsstrategie (1/3)

## Erweiterbarkeit

*„Gradle lässt sich leicht um neue Funktionalität erweitern. ...“*

- Build-Files sind **Groovy-Skripte**, kleine Erweiterungen ohne aufwändige Programmierung
- Ausgereiftes **Plugin-Konzept**, Entwicklung von Custom Plugins in Groovy, Java, Scala …
- Exzellent dokumentierte Gradle API und erweiterbare **DSL**

## Effizienz

*“Teams und einzelne Entwickler erhalten durch kurze Buildzeiten schnelle Rückmeldungen. …”*

- Unterstützung **inkrementeller Builds**
- Dependency-Cache zur Reduktion der Download-Zeiten
- Betrieb im Hintergrund für kürzere Start- und Ausführungszeiten (**Gradle Daemon**)

# Lösungsstrategie (2/3)

## Interoperabilität

*„Gradle arbeitet mit bestehenden Werkzeugen wie Ant und Maven und deren Öko-Systemen nahtlos zusammen.“*

- Direkte Verwendung von **Ant**-Tasks und Ant-Projekten in Gradle möglich
- Konverter für **Maven** pom.xml nach Gradle Build-Skript verfügbar
- Unterstützung von Maven-Repositories
- **Gradle API** erlaubt Einbetten von Gradle, zum Beispiel in IDEs

## Erlernbarkeit

*“Entwickler und Buildmanager finden sich schnell in Gradle zurecht, ...”*

- Verwendung der Sprache **Groovy** für Build-Skripte
- Deklarative Builds mit Groovy DSL und „build-by-convention“
- Zielgruppengerechte Dokumentation (**Tutorials**, User-Guide)



# Lösungsstrategie (3/3)

## Installierbarkeit

*„Der Aufwand, der zum Installieren von Gradle notwendig ist, um einen Build auszuführen ist, sehr gering.“*

- Java einzige Systemvoraussetzung
- **Wrapper** lädt Gradle automatisch herunter, Builds ohne weitere Installation möglich

## Skalierbarkeit

*“Gradle bleibt auch bei sehr umfangreichen Builds und in Multi-Projekt-Szenarien handhabbar und effizient. ...”*

- Umfassende Unterstützung für **Multi-Project Builds**
- darüber hinaus: Ansätze für Effizienz (siehe oben)

# Agenda

3

1 Aufgabenstellung
2 Lösungsstrategie
3 **Die Lösung im Detail**
4 Weitere Informationen



# Warum Groovy?

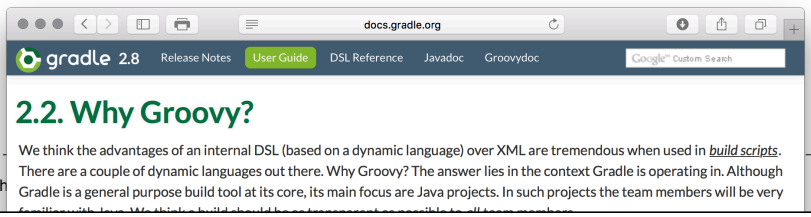
Gradle-Buildskripte basieren auf Groovy. Anders als bei Ant und Maven, wo XML zum Einsatz kommt.

## Was spricht für Groovy?

- einfache Erlernbarkeit für Entwickler mit Java-Background
- DSL-Features von Groovy ermöglichen
  - deskriptive Build-Skripte
  - gute Erweiterbarkeit

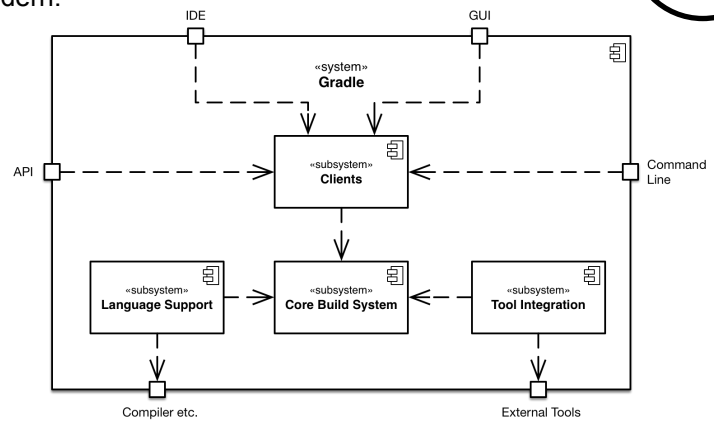
## Zum Nachlesen:

→ https://docs.gradle.org/current/userguide/overview.html#sec:why_groovy



# Der Aufbau von Gradle

Gradle lässt sich in 4 grobe Blöcke („Subsysteme“) zergliedern:




Das Diagramm zeigt sie mit ihrer Interaktion untereinander und mit außen, die gestrichelten Pfeile stellen Abhängigkeiten bzw. Verwendungen dar.



# Subsysteme: Verantwortlichkeiten

| | |
|---|---|
| «subsystem» **Clients** | Gradle stellt Verwendern seine Funktionalität auf verschiedene Weise bereit. **Kommandozeile**, einfache **graphische Oberfläche**, Integration in **Entwicklungsumgebungen** und **CI-Server** … |
| «subsystem» **Core Build System** | Der Kern von Gradle umfasst die **Basisfunktionalität** (Interpretation von Skripten, Abhängigkeitsmanagement, Ausführung von Tasks, Java-Unterstützung …) sowie das **Plugin-Konzept** als Erweiterungspunkt. |
| «subsystem» **Tool Integration** | Plugins, die **Werkzeuge** anderer Projekte **integrieren**, etwa zur Qualitätssicherung, zur Veröffentlichung von Artefakten in Repositories, etc. |
| «subsystem» **Language Support** | Plugins zur Unterstützung weiterer **Programmiersprachen** neben Java (Scala, JavaScript, …) |

# Wichtige Konzepte

Der Gradle User Guide beschreibt viele in der Lösungsstrategie genannte Konzepte.

Hier einige Beispiele:

## Abhängigkeitsmanagement

Gradle User Guide: Kapitel 8. “Dependency Management Basics”
Kapitel 52. “Dependency Management”

## Gradle Wrapper

Gradle User Guide: Kapitel 64. “The Gradle Wrapper”

## Multi-project Builds

Gradle User Guide: Kapitel 59. ”Multi-project Builds”

➔ https://docs.gradle.org/current/userguide/userguide.html



# Demo

An dieser Stelle eine kurze Live-Demo von Gradle ...

# Agenda

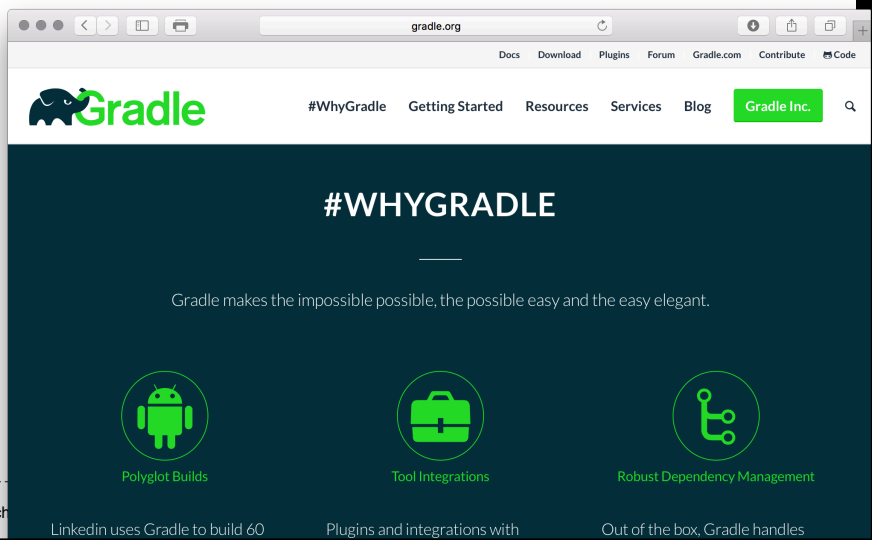
4

1 Aufgabenstellung
2 Lösungsstrategie
3 Die Lösung im Detail
4 **Weitere Informationen**



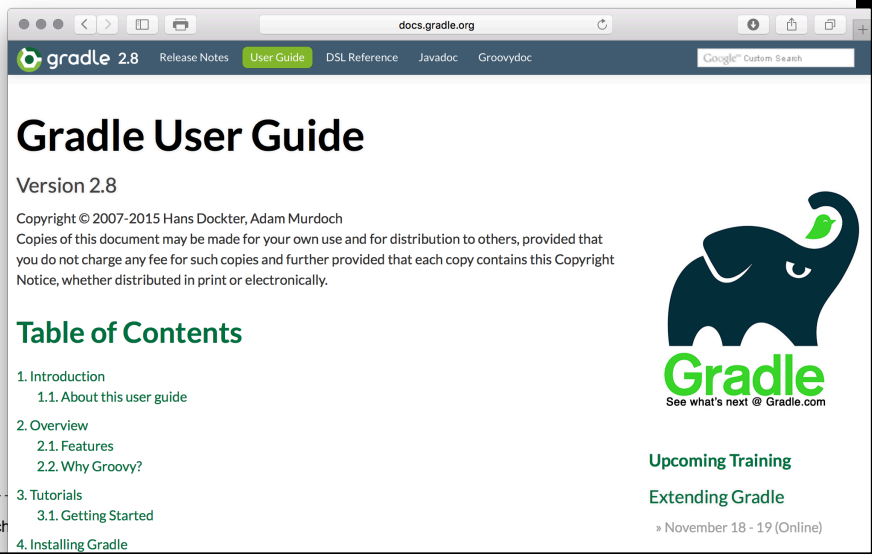
# Gradle Webseite

➔http://www.gradle.org



# Gradle Dokumentation

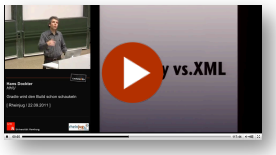
➔https://docs.gradle.org






# Videos

## „Gradle wird den Build schon schaukeln“

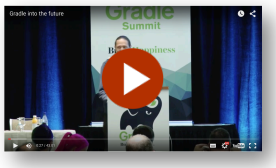



rheinjug
JAVA USER GROUP DÜSSELDORF

Hans Dockter, CEO / Founder Gradle
rheinjug Düsseldorf, 22.09.2011 (118 Minuten)

➔ http://www.rheinjug.de/videos/gse.lectures.app/Talk.html#Gradle

## „Gradle into the future“




gradle summit

Adam Murdoch, CTO Gradle
Gradle Summit, Juni 2015 (43 Minuten)

➔ https://www.youtube.com/watch?v=4yWIejvC_I0

## Mitmachen, mitdiskutieren …

### Quelltextverwaltung von Gradle bei GitHub

github SOCIAL CODING

➔ https://github.com/gradle

### Foren und Defect-Tracking

➔ http://forums.gradle.org/gradle

➔ https://issues.gradle.org/ (JIRA)



# Vielen Dank.

## Ich freue mich auf Eure Fragen!

stefan.zoerner@embarc.de

@StefanZoerner

xing.to/szr

embarc Software Consulting GmbH

**DOWNLOAD FOLIEN:**
http://www.embarc.de/blog/