

# canoo

› your provider for business web solutions ›



## Wie finde ich das passende Build-System?

Ant vs. Maven vs. Gradle

**Christoph Lipp & Etienne Studer**  
**Canoo Engineering AG**

## Referenten

- Christoph Lipp
  - ▶ +10 Jahre Erfahrung in Software-Entwicklung und IT-Consulting
  - ▶ +3 Jahre hands-on Erfahrung mit Maven/Ant Rollout in Grossprojekten
  - ▶ Branchen: Banken, Versicherungen, Elektrounternehmen
  
- Etienne Studer
  - ▶ +10 Jahre Erfahrung als Software Entwickler/Architekt in agilen Projekten
  - ▶ 5 Jahre Container Shipping Terminal Software, Navis LLC, Oakland, USA
  - ▶ 5 Jahre Banking und Linguistik Software, Canoo AG, Basel, Schweiz
  - ▶ JetBrains Evangelist (IntelliJ IDEA/TeamCity) in den USA



Abreise	Heim	Zeit	Ergebnis	Tipp Mann	Tipp Frau
16. Jun. 13:00h	Honduras	Chile	-	<input type="checkbox"/>	<input type="checkbox"/>
18. Jun. 15:00h	Spanien	Schweiz	-	<input type="checkbox"/>	<input type="checkbox"/>
21. Jun. 18:00h	Chile	Schweiz	-	<input type="checkbox"/>	<input type="checkbox"/>
23. Jun. 20:00h	Spanien	Honduras	-	<input type="checkbox"/>	<input type="checkbox"/>
25. Jun. 20:00h	Chile	Spanien	-	<input type="checkbox"/>	<input type="checkbox"/>
27. Jun. 20:00h	Schweiz	Honduras	-	<input type="checkbox"/>	<input type="checkbox"/>

**Hackergarten**  
A Computer Programming Contributor Group



**Motivation:**  
**Kontinuierliches automatisiertes Bauen der Software  
mit Hilfe eines wartbaren, erweiterbaren und skalierbaren  
Build-Systems**

## Build-System Anforderungen

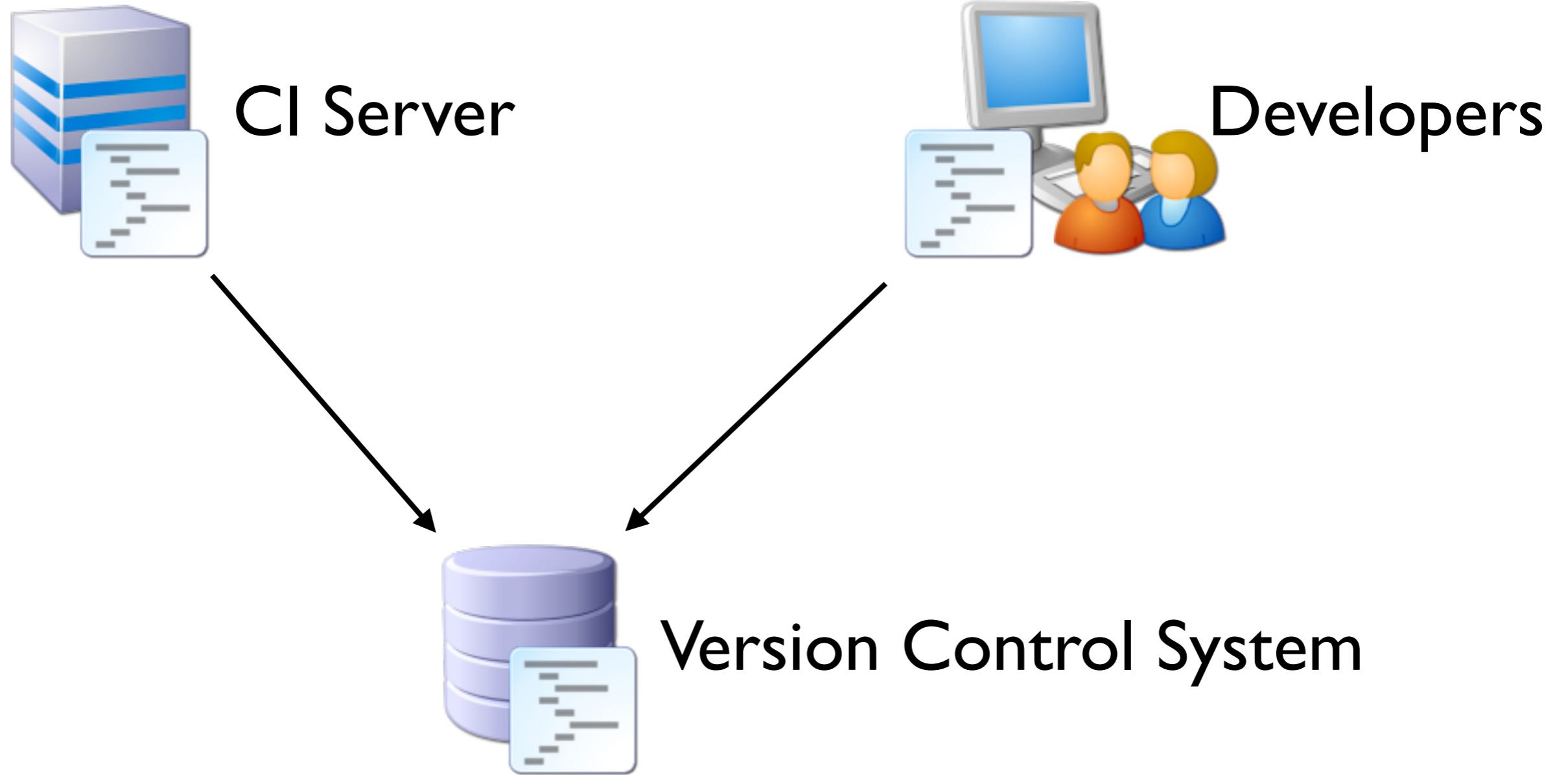
- ⦿ Wartbarkeit
  - ▶ Lesbarkeit/Verständlichkeit/Redundanzfreiheit der Build Dateien
  - ▶ Transparenz der Build Prozess Vorgänge
  - ▶ Konfigurierbarkeit
  
- ⦿ Erweiterbarkeit
  - ▶ Einbinden externer Tasks
  - ▶ Definieren eigener Tasks
  
- ⦿ Skalierbarkeit
  - ▶ Multi-Module Unterstützung
  - ▶ Dependencies Management

## Automatisiertes Bauen

- ⦿ Reproduzierbares Bauen der Software
- ⦿ Dokumentiertes Bauen der Software
- ⦿ Basis für kontinuierliches Bauen der Software

## Kontinuierliches Bauen

- ⦿ Kontinuierliches Integrieren von Software Beiträgen
- ⦿ Kontinuierliches Feedback zum Stand der Software
- ⦿ Kontinuierliches Deployen der Software



## JAXenter Umfrage *‘Welches Build-System setzen Sie ein?’*

- 55% Maven
- 22% Ant
- 12% Gradle

# Relevante Eigenschaften von Ant, Maven & Gradle

## Ant

- XML-basiert
- Unstrukturierte Sammlung von Tasks
- Viele Freiheitsgrade beim Definieren des Builds
- Einfaches Definieren und Einbinden von externen Tasks
- Sehr ausgereift und erprobt auf vielen Plattformen
- Viel und gute Dokumentation/Literatur

## Ant - Code Beispiel

```
<project name="sample" default="compile">
  <property name="srcDir" value="src/main/java"/>
  <property name="classesDir" value="build/classes/main"/>
  <property name="depDir" value="lib"/>
  <property name="artifact" value="build/libs/sample-1.0.jar"/>

  <target name="compile" description="compile java source code to class files">
    <mkdir dir="${classesDir}"/>
    <javac srcdir="${srcDir}" destdir="${classesDir}"/>
  </target>

  <target name="jar" depends="compile" description="bundle class files in jar file">
    <jar destfile="${artifact}">
      <fileset dir="${classesDir}" includes="**/*.class"/>
    </jar>
  </target>

  <taskdef resource="cloverlib.xml" classpath="${depDir}/clover.jar"/>

</project>
```

## Maven

- XML-basiert
- Echtes Build-System mit Lebenszyklus und definierten Phasen
- Hierarchische Build-Struktur (POM)
- Plugin-Architektur
- Einfaches Definieren und Einbinden von externen Plugins
- Eingebautes Dependency Management inkl. Artefakt-Repository
- Sehr ausgereift und erprobt auf vielen Plattformen
- Gewisse Dokumentation/Literatur vorhanden

## Maven - Code: Explizite Konfiguration

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.canoo.sample</groupId>
  <artifactId>sample</artifactId>
  <version>1.0</version>
  <packaging>jar</packaging>

  <build>
    <sourceDirectory>src/main/java</sourceDirectory>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.3</version>
        <executions><execution><phase>compile</phase><goal>compile</goal></execution></executions>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <version>2.3</version>
        <executions><execution><phase>package</phase><goal>jar</goal></execution></executions>
      </plugin>
    </plugins>
  </build>
</project>
```

## Maven - Code: Konvention über Konfiguration

```
<project>  
  <modelVersion>4.0.0</modelVersion>  
  <groupId>com.canoo.sample</groupId>  
  <artifactId>sample</artifactId>  
  <version>1.0</version>  
</project>
```

## Gradle

- ⊙ Groovy-basiert
- ⊙ Echtes Build-System mit vordefinierten Task Abhängigkeiten
- ⊙ Viele Freiheitsgrade beim Definieren des Builds
- ⊙ Plugin-Architektur
- ⊙ Einfaches Definieren und Einbinden von externen Plugins
- ⊙ Eingebautes Dependency Management inkl. Artefakt-Repository
- ⊙ Plattformunabhängig durch Delegation der Basis-Operationen an Ant
- ⊙ (Noch) wenig Dokumentation/Literatur vorhanden

## Gradle - Code Beispiel: Explizite Konfiguration

```
apply plugin: 'java'

version = 1.0

compile {
    source = 'src/main/java'
}

jar {
    baseName = "sample.jar"
    manifest {
        attributes("Implementation-Title": "Sample")
    }
}

repositories {
    mavenCentral()
}

dependencies {
    testCompile "junit:junit:4.4"
}
```

## Gradle - Code Beispiel: Konvention über Konfiguration

apply plugin: 'java'

# Direkter Vergleich von Ant, Maven & Gradle

## Wartbarkeit

- ◉ Ant
  - ▶ Imperatives XML schwer lesbar
  - ▶ Komplexe Verkettung von Targets schwer nachvollziehbar
  - ▶ Konfiguration durch (überschreibbare) System Properties unübersichtlich
  
- ◉ Maven
  - ▶ Standardisierte Lebenszyklus Phasen und Konvention vorhanden
  - ▶ POM-Hierarchien ermöglichen Vererbung von Build Eigenschaften  
Projekt-übergreifende Standards realisieren
  - ▶ Behandeln von Spezialfällen über Plugins mit Overhead verbunden
  
- ◉ Gradle
  - ▶ Standardisierte Task Abhängigkeiten und Konventionen vorhanden
  - ▶ Groovy DSL kompakt und gut lesbar
  - ▶ Verschiedene Codierungsmöglichkeiten führen ans gleiche Ziel  
Gefahr der Inkonsistenz innerhalb der Build Files

## Erweiterbarkeit

- ◉ Ant
  - ▶ Viele Open Source Tasks verfügbar
  - ▶ Eigene Tasks definierbar über Makros, Skripts, und Java Implementierung
  
- ◉ Maven
  - ▶ Viele Open Source Plugins verfügbar
  - ▶ Plugins über zentrales Maven Repository einfach ins Projekt integrierbar
  - ▶ Eigene Plugins definierbar über Ant und Java Implementierung
  
- ◉ Gradle
  - ▶ Viele Open Source Plugins verfügbar
  - ▶ Eigene Plugins definierbar im Build Skript selbst, im Projekt und über kompilierte (Java/Groovy/etc.) Implementierung

## Skalierbarkeit

- ◉ Dependency Management, Multi-Module Unterstützung und Speed sind zentral
- ◉ Ant
  - ▶ Dependency Management via Ivy Integration möglich
  - ▶ Beschränkter Support für Multi-Module Projekte
  - ▶ Bedingtes Ausführen von Tasks über Module Grenzen hinaus schwierig
- ◉ Maven
  - ▶ Dependency Management integraler Bestandteil
  - ▶ Modul-Aggregation und Modul-Vererbung gut unterstützt
  - Multi-Module Builds und Hierarchische POM-Struktur
- ◉ Gradle
  - ▶ Dependency Management integraler Bestandteil
  - ▶ Multi-Module Support integraler Bestandteil
  - ▶ *gradlew* zum Skalieren über viele Entwickler ohne Gradle Vorinstallation

## Standardisierung vs. Flexibilität

- ◎ Zweischneidiges Schwert
- ◎ Ant
  - ▶ flexibles Reagieren auf neue Build Requirements einfach
  - ▶ Gefahr von Inkonsistenz und Ad-hoc Lösungen
- ◎ Maven
  - ▶ Reagieren auf neue Build Requirements immer über Plugins
- ◎ Gradle
  - ▶ flexibles Reagieren auf neue Build Requirements einfach
  - ▶ Gefahr von Inkonsistenz und Ad-hoc Lösungen

## IDE Unterstützung

- Bessere IDE Unterstützung für deklarative Build-Systeme möglich
- Ant
  - ▶ Relativ gute Unterstützung solange Build Files nicht zu komplex
- Maven
  - ▶ Sehr gute Unterstützung für Single- und Multi-Module Projekte
  - ▶ Plugins zur Generierung der Projekt Files (IntelliJ IDEA/Eclipse)
- Gradle
  - ▶ Beschränktes semantisches Verständnis der Build Files für IDEs
  - ▶ Sehr gute IDE Unterstützung für Groovy Sprache
  - ▶ Tasks zur Generierung der Projekt Files (IntelliJ IDEA/Eclipse)  
basierend auf der Struktur nach der Konfigurationsphase

## Lernkurve

- Reifere Build-Systeme haben mehr Dokumentation und Best Practices
- Ant
  - ▶ Flache Lernkurve
  - ▶ Viel Dokumentation und Best Practices aus Literatur/Internet
- Maven
  - ▶ Steile Lernkurve
  - ▶ Einige Dokumentation und Best Practices aus Literatur/Internet
- Gradle
  - ▶ Steile Lernkurve
  - ▶ (Noch) wenig Dokumentation im Internet, keine Literatur
  - ▶ Schulungen angeboten

## Migration

- ⊙ Schlagartiges Wechseln zu anderem Build-System kann vermieden werden
- ⊙ Ant
  - ▶ Migration von Maven nach Ant technisch möglich
- ⊙ Maven
  - ▶ sanfte Migration von Ant nach Maven unterstützt
  - ▶ alle Core Ant Tasks und eigene Ant Skripts via Plugin verfügbar
- ⊙ Gradle
  - ▶ sanfte Migration von Ant nach Gradle unterstützt
  - ▶ alle Core Ant Tasks direkt verfügbar
  - ▶ eigene Ant Skripts einfach importierbar

# Entscheidung treffen

## Fazit

- ⊙ Entscheidung treffen unter Berücksichtigung der erwähnten Aspekte im Kontext des Projektes und der Unternehmensorganisation
- ⊙ Viele Best Practices sind unabhängig vom eingesetzten Build System
- ⊙ Build-System integraler Bestandteil des Software Projektes