Build-Tools und Continuous Integration/Deployment

Fortgeschrittene Softwaretechnik WS 2024/25

Lucas Kreber – kreberl@uni-trier.de

Technologie Dschungel

Programmiersprachen

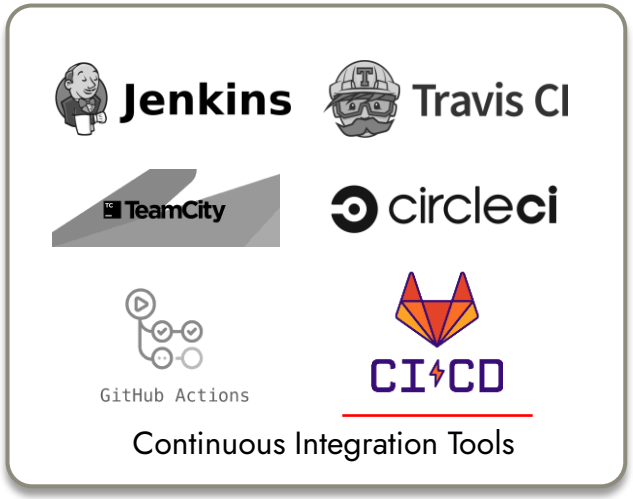
Entwicklungsumgebungen

Versionsverwaltung

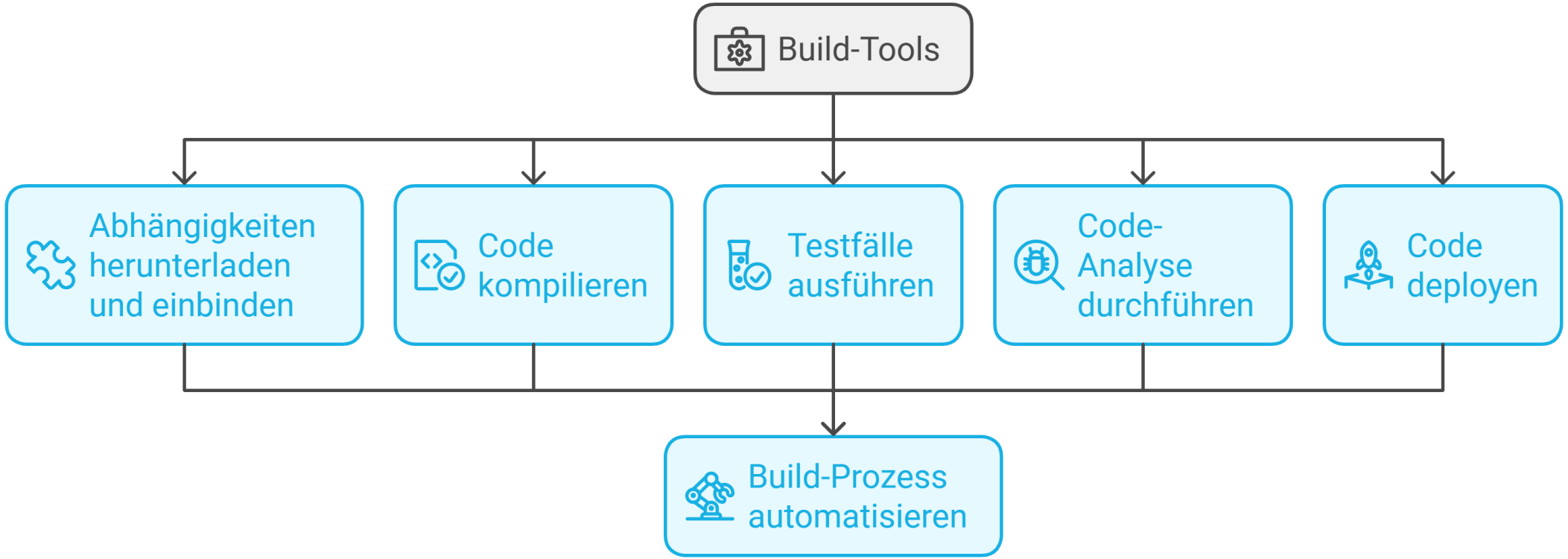
Continuous Integration Tools

Build-Skripte

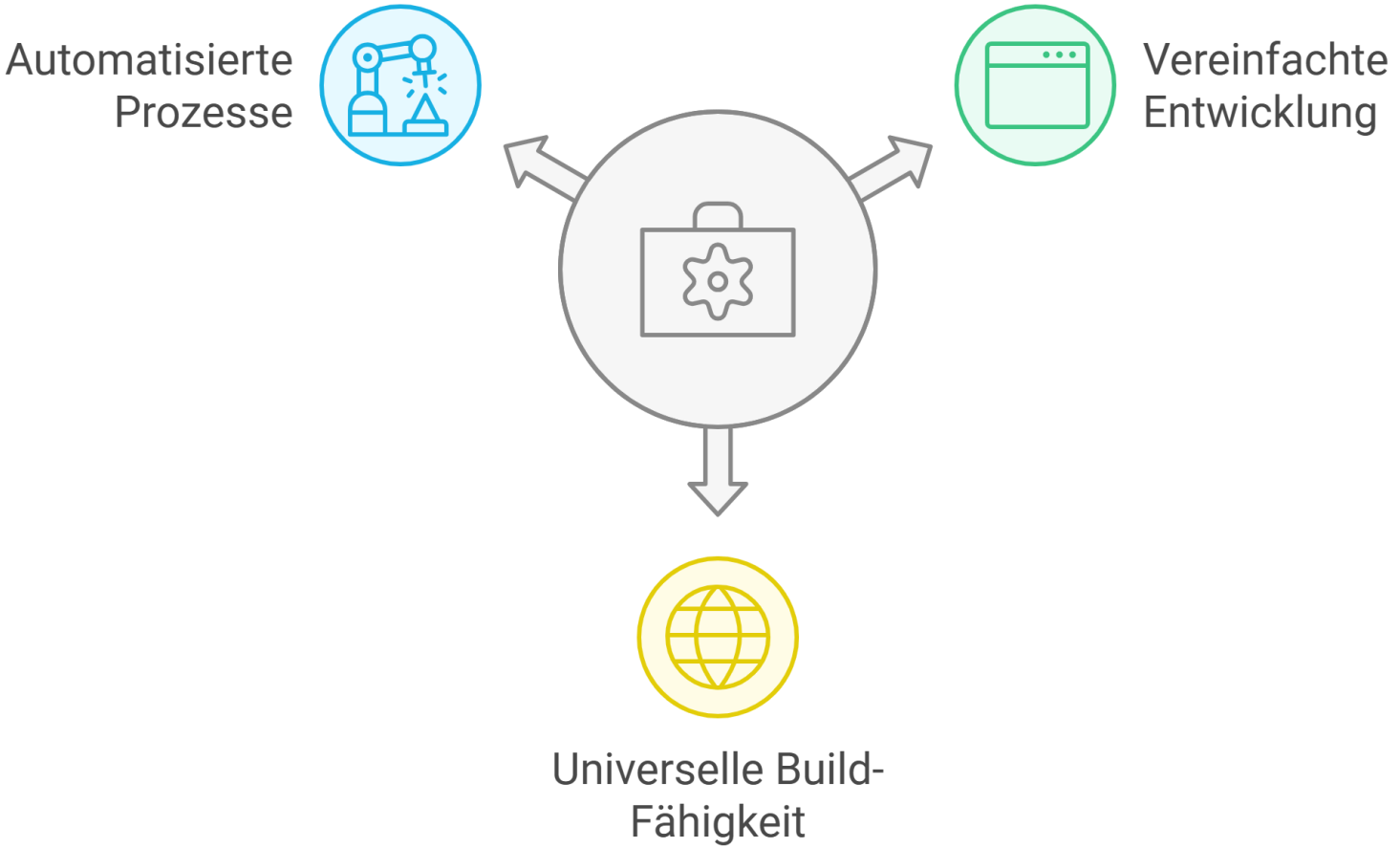
Technologie Dschungel



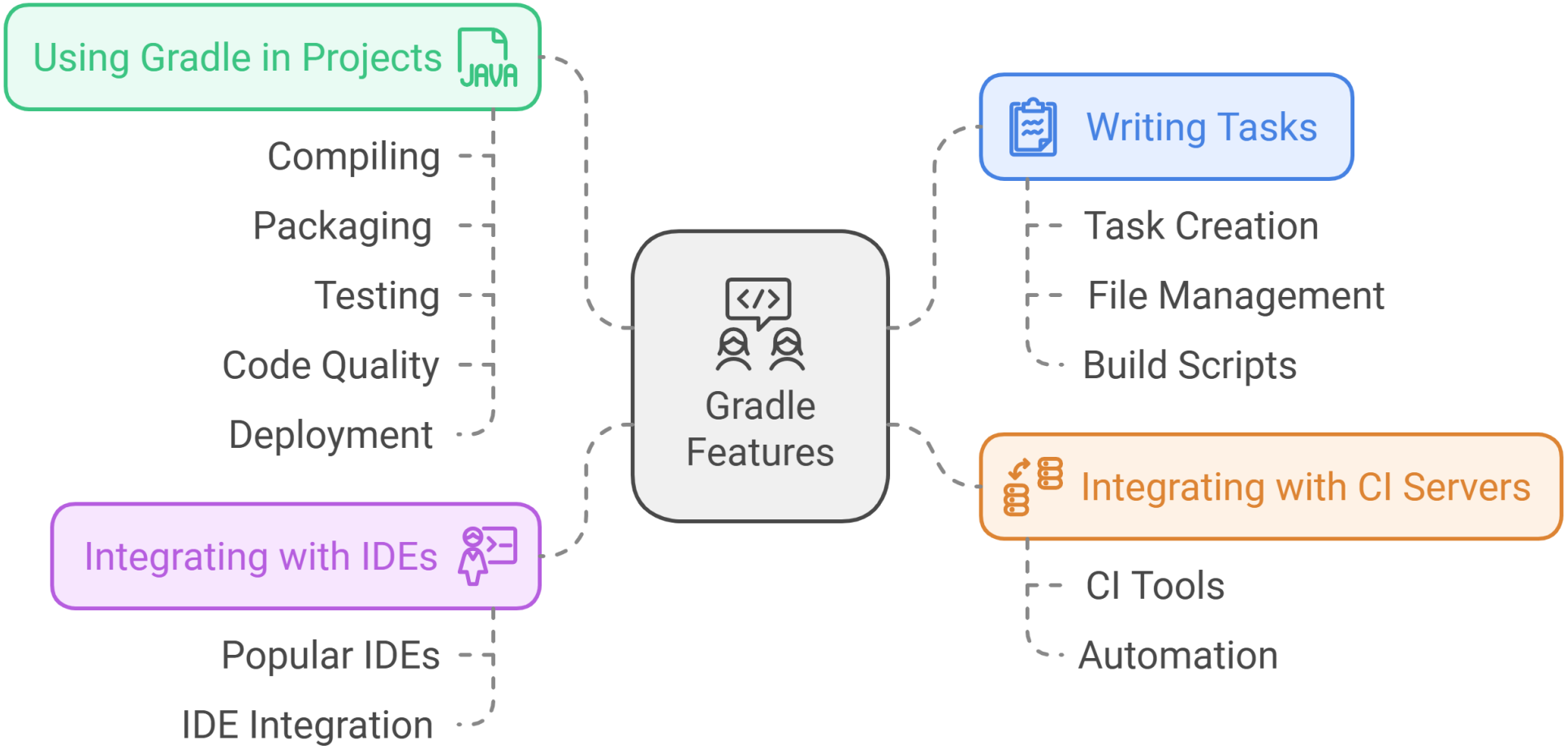
Build Tools


Build-Tools
Abhängigkeiten herunterladen und einbinden
Code kompilieren
Testfälle ausführen
Code-Analyse durchführen
Code deployen
Build-Prozess automatisieren

Build Tools


Automatisierte Prozesse
Vereinfachte Entwicklung
Universelle Build-Fähigkeit

Inhalt (z.T.) dieser Vorlesung



Installation von Gradle: https://gradle.org/install/

Gradle vs. Maven

Gradle

Maven

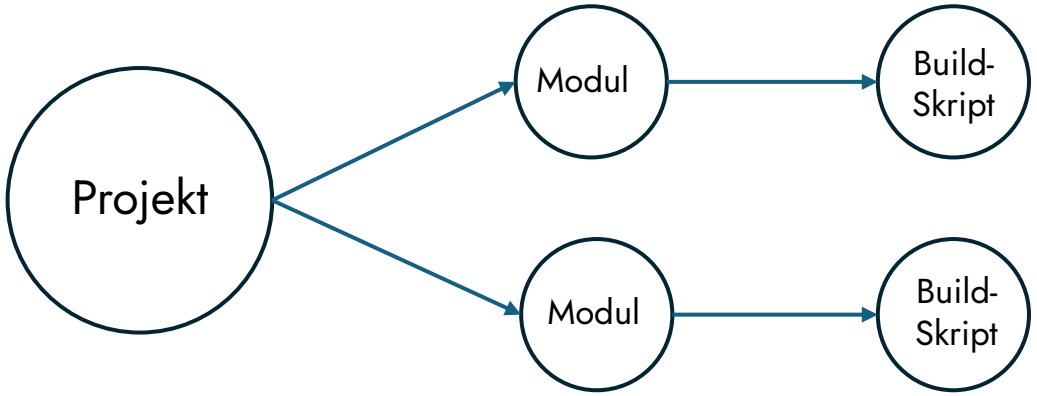
- **Convention over configuration** (zum Beispiel Projektstruktur)
- sehr flexibel, Build-Skript ist Code
- In großen Projekten schneller

- **Convention over configuration** (weniger flexible)
- Im Vergleich zu Gradle eher langsam

Skripte zur Vorlseung

Alle Skripte stehen in Plaintext oder gepackt als Zip über das Icon  zur Verfügung

Erstes Gradle Build-Skript



Ein Projekt enthält ein oder mehrere Module

Ein **Subprojekt**, das eigenständig gebaut werden kann, aber Teil eines größeren **Multi-Module-Projekts** ist. Gibt es nur ein Modul, ist dieses das ganze Projekt

Ein Modul enthält ein Build-Skript mit einem oder mehreren Build-Tasks

Definition einzelner Schritte (kompilieren, testen, deployen, …)

Erstes Gradle Build-Skript (1.1)

1. Erzeugen einer Datei 'build.gradle'

Gradle sucht nach Datei `build.gradle` im aktuellen Verzeichnis

2. `build.gradle` öffnen und Task erstellen

```
task helloWorld {
   println "Hello World."
}
```

3. Ausführen:
   - `gradle helloWorld` → Task ´helloWorld´ ausführen
   - `gradle -quiet helloWorld` → Task ´helloWorld´ ausführen und nur Ausgaben des Tasks in der Konsole ausgeben
   - `gradle -q task1 task2 …` → Mehrere Tasks hintereinander ausführen
   - `gradle -dry-run helloWorld` → Simuliert Build-Prozess. Ausgabe aller Tasks, die Durchlaufen werden, ohne diese wirklich auszuführen

Beispiel 2.1

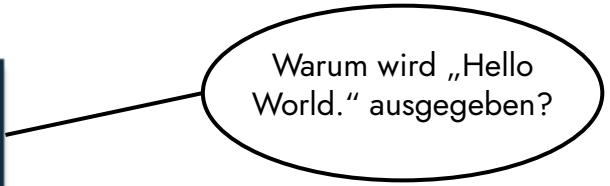
```
project.description = "Simple project"

task simple {
    println "Running simple task for project " + project.description
}
```

- Gradle liest das Build-File und erzeugt ein Projekt-Objekt
- Projekt-Objekt hat verschiedene Eigenschaften und Methoden

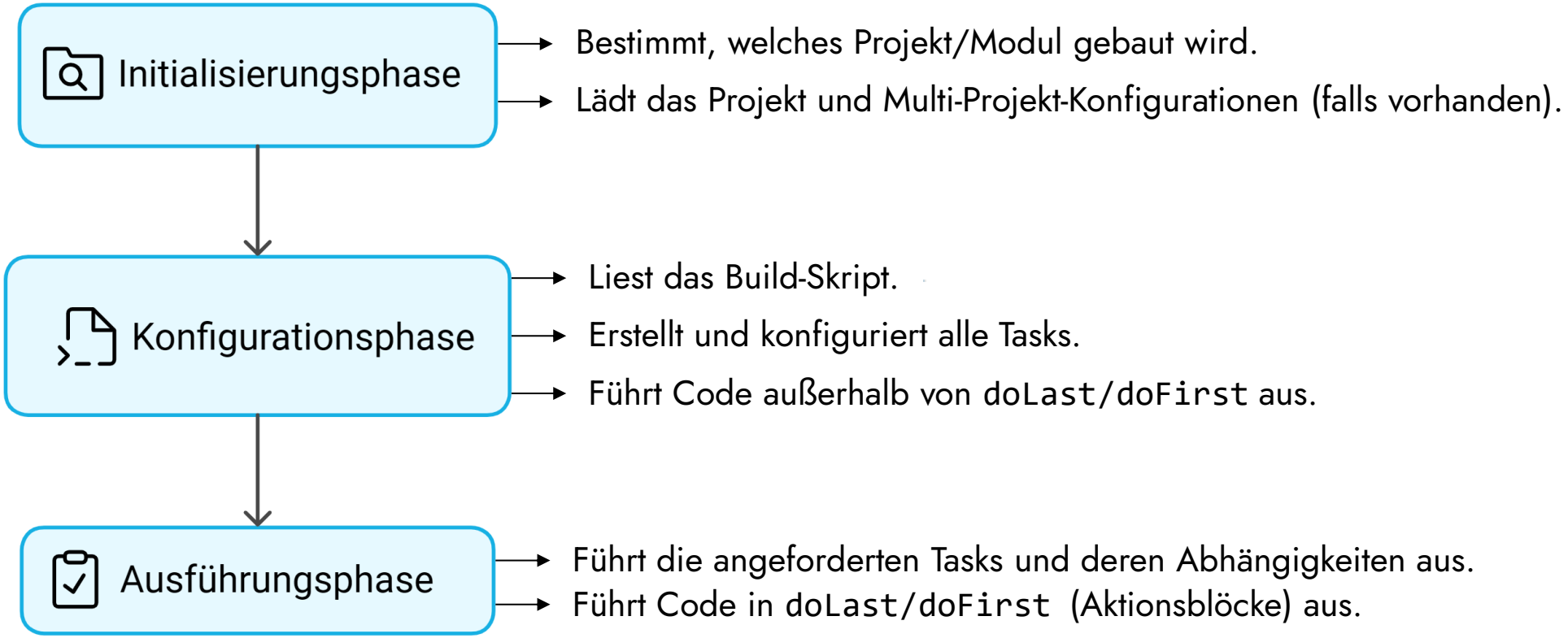
Beispiel 2.2

```
project.description = "Simple project"

task simple {
    println "Running simple task for project " + project.description
}

Task helloWorld {
    println "Hello World."
}
```

```
$ gradle -q simple
  Running simple task for project Simple project
  Hello World.
```



Beispiel 2.2


Initialisierungsphase
Bestimmt, welches Projekt/Modul gebaut wird.
Lädt das Projekt und Multi-Projekt-Konfigurationen (falls vorhanden).
Konfigurationsphase
Liest das Build-Skript.
Erstellt und konfiguriert alle Tasks.
Führt Code außerhalb von doLast/doFirst aus.
Ausführungsphase
Führt die angeforderten Tasks und deren Abhängigkeiten aus.
Führt Code in doLast/doFirst (Aktionsblöcke) aus.

Beispiel 2.2

```
project.description = "Simple project"
ext.msg = 'Running first in project '

task first {
    doFirst {
        println msg + project.description
    }
}

task second {
    doLast { Task task ->
        println “Running ${task.name}”
    }
}

task third {
    doLast {
        println “Running “ + it.name
    }
}
```

Mit ext.varName können global Variablen definiert werden. Zugriff ohne ext.

Zugriff auf Variablen in Strings mit ${var}

it ist vergleichbar mit this in Java und bezieht sich immer auf den konkreten Task

```
$ gradle –q first second
  > Task :first
  Running first

  > Task :second
  Running second
```

Beispiel 2.3 (Exkurs)

```
task numbers {
    doFirst {
        (1..4).each { number ->
            def squared = number * number
            println "Square of ${number} = ${squared}"
        }
    }
}
```

```
$ gradle -q numbers
  > Task :numbers
  Square of 1 = 1
  Square of 2 = 4
  Square of 3 = 9
  Square of 4 = 16
```

Beispiel 2.4

```
defaultTasks 'third'

task first {
    doFirst {
        println "Run ${it.name}"
    }
}

task second {
    doFirst {
        println "Run ${it.name}"
    }
}

task third(dependsOn: "second") {
    doFirst {
        println "Run ${it.name}"
    }
}

second.dependsOn 'first'
```

Dieser Task wird ausgeführt, wenn beim Aufruf von `gradle` kein spezifischer Task angegeben wird. Es können mehrere Default-Tasks festgelegt werden.

Bevor third ausgeführt wird, muss second ausgeführt werden

Bevor second ausgeführt wird, muss first ausgeführt werden

```
$ gradle
  > Task :first
  Run first

  > Task :second
  Run second

  > Task :third
  Run third
```

Beispiel 2.5

```
task writeFile {
    doFirst {
        def content = “Hello World!“
        file(“hello.txt").text = content
    }
}

task readFile {
    doFirst {
        def content = file(“hello.txt”).text
        if (content == “Hello World!”) {
            println content
        } else {
            throw new GradleException(“Wrong File!”)
        }
    }
}

task deleteFile {
    doFirst {
        delete file(“hello.txt”)
    }
}
```

→ Definiere eine Variable content

→ Erstelle eine Datei hello.txt mit dem Inhalt der Variable content

→ Öffne eine Datei hello.txt und lese deren Inhalt

→ Werfe eine Exception

Übung 1

Schreibe ein Gradle-Skript, das mehrere Aufgaben zur Verwaltung eines Schlosscodes automatisiert. Die Schritte umfassen das Festlegen eines Codes, das Überprüfen des Codes und das Ausgeben einer Erfolgsmeldung. Die Reihenfolge der Tasks soll durch Abhängigkeiten der Tasks sichergestellt werden.

1. Erstelle eine Datei mit einem Code: Der erste Task soll eine Datei erstellen und einen Code darin speichern.
2. Überprüfe den Code: Der zweite Task soll sicherstellen, dass der Code ausgelesen wird und korrekt ist. Falls nicht, soll der Prozess abgebrochen werden.
3. Gebe eine Erfolgsmeldung aus: Der dritte Task soll abhängig von Task 2 ausgeführt werden und eine Meldung ausgeben, dass das Schloss erfolgreich geöffnet wurde.

Beispiel 3.1

```
task folderCopy(type: Copy) {
    from "src/files"
    into "destination"
}

task fileCopy(type: Copy) {
    from "src/files/File1.txt"
    into "destination"
}

task extensionCopy(type: Copy) {
    from "src/files"
    include "**/*.java"
    into "destination"
}

task excludeCopy(type: Copy) {
    from "src/files"
    exclude "**/*.java"
    into "destination"
}
```

- Verwende Copy als vordefinierten Task
- Kopiere alle Dateien aus dem Ordner src/files in den Ordner destination
- Kopiere nur die Datei File1.txt in den Ordner destination
- Kopiere alle Dateien mit der Endung .java (include) in den Ordner destination
- Kopiere alle Dateien, welche die Endung .java nicht haben (exclude), in den Ordner destination

Beispiel 3.2

```
task zip(type: Zip) {
    from "src/Files"

    archiveBaseName = "src-files"
    archiveAppendix = "archive"
    archiveExtension = "zip"
    archiveVersion = "1.0"
    archiveClassifier = "sample"

    destinationDirectory.set(file("$buildDir/archives"))
}

task unzip(type: Copy) {
    def oldLocation = file("$buildDir/file.zip")
    def newLocation = file("$buildDir/dest")

    from zipTree(oldLocation)
    into newLocation
}
```

→ Verwende Zip als vordefinierten Task

→ Setze Informationen zum Benennen der .zip Datei

→ Definiere das Ziel, in der die Datei gespeichert wird

Beispiel 3.3

```
plugins {
    id "de.undercouch.download" version "4.1.1"
}

task download(type: Download) {
    src 'https://www.lpi.usra.edu/resources/mars_maps/1083/300dpi.jpg'
    dest buildDir
    onlyIfModified true
}
```

→ Verwende das Plugin de.undercouch.download

→ URL angeben

→ Speicherort angeben

→ Nur herunterladen, wenn sich die Datei verändert hat

Übung 2

Ein Dieb möchte ein geheimes Kuchenrezept stehlen. Das Rezept befindet sich in der Konditorei (entspricht dem Download einer Datei). Dein Ziel ist es, die Datei herunterzuladen, den Inhalt zu lesen und danach alles wieder so zu verpacken, dass niemand merkt, dass du das Rezept gesehen hast.

1. Rezept herunterladen: Lade die ZIP-Datei mit dem geheimen Rezept von folgender URL herunter: https://www.st.uni-trier.de/Gradle/Rezept.zip
2. Entpacke Rezept: Entpacke die ZIP-Datei in ein Verzeichnis und lösche diese danach.
3. Lese Rezept: Lese die Rezeptdatei (Rezept.txt), um den Inhalt anzuzeigen.
4. Zip Erstellen: Packe die entpackte Datei erneut in eine neue ZIP-Datei, um keine Spuren zu hinterlassen.

Beispiel 4.1

```
plugins {
    id "java"
}
```

Verwende das Plugin java, um vordefinierte, java spezifische Tasks zu erhalten

```
$ gradle tasks

Build tasks
-----------
assemble - Assembles the outputs of this project.
build - Assembles and tests this project.
buildDependents - Assembles and tests this project and all projects that depend on it.
buildNeeded - Assembles and tests this project and all projects it depends on.
classes - Assembles main classes.
clean - Deletes the build directory.
jar - Assembles a jar archive containing the classes of the 'main' feature.
testClasses - Assembles test classes.

Build Setup tasks
-----------------
init - Initializes a new Gradle build.
wrapper - Generates Gradle wrapper files.

Documentation tasks
-------------------
javadoc - Generates Javadoc API documentation for the 'main' feature.

Help tasks
----------
buildEnvironment - Displays all buildscript dependencies declared in root project '1'.
dependencies - Displays all dependencies declared in root project '1'.
dependencyInsight - Displays the insight into a specific dependency in root project '1'.
help - Displays a help message.
javaToolchains - Displays the detected java toolchains.
outgoingVariants - Displays the outgoing variants of root project '1'.
projects - Displays the sub-projects of root project '1'.
properties - Displays the properties of root project '1'.
resolvableConfigurations - Displays the configurations that can be resolved in root project '1'.
tasks - Displays the tasks runnable from root project '1'.

Verification tasks
------------------
check - Runs all checks.
test - Runs the test suite.
```

Beispiel 4.2

```
plugins {
    id "java"
}

task run(type: JavaExec) {
    main = 'com.example.HelloWorld'
    classpath = sourceSets.main.runtimeClasspath
}
```

Verwende das Plugin java

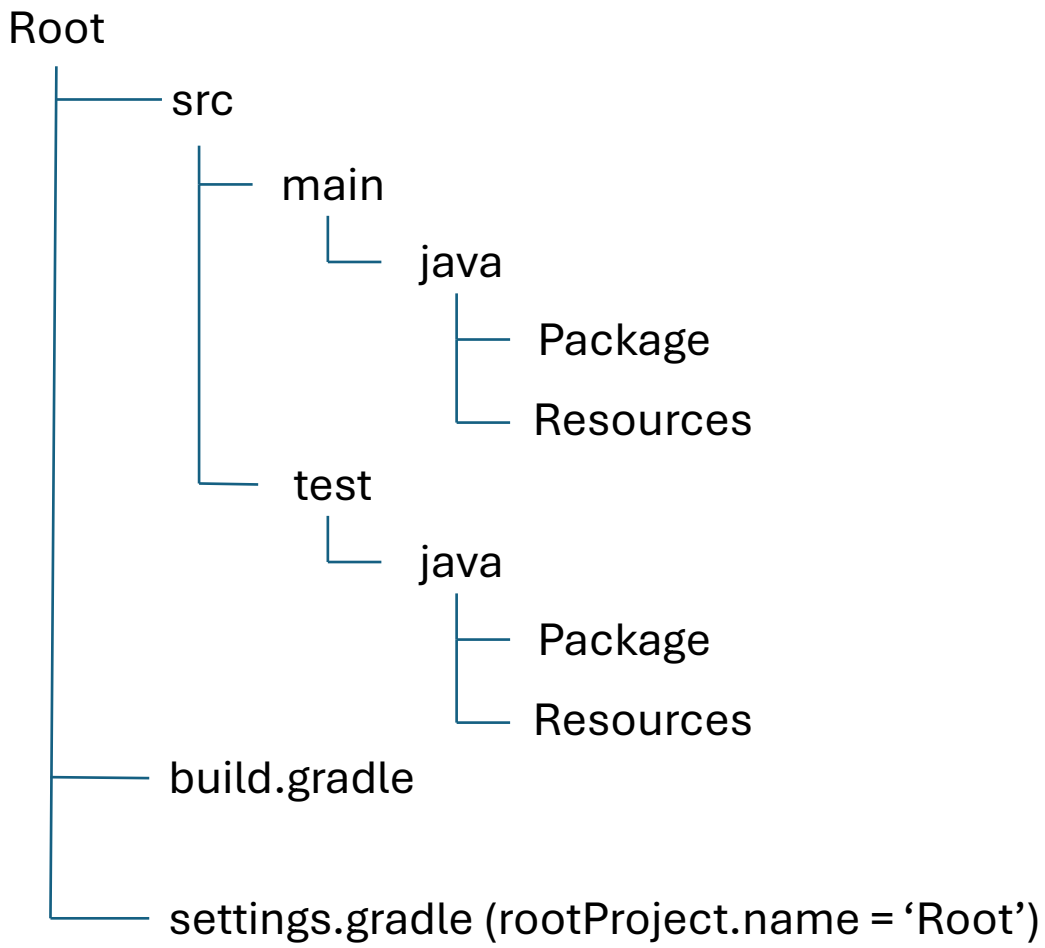
Definiere einen Task run zum Ausführen. Hier wird die Main-Klasse und der Klassenpfad gesetzt

```
plugins {
    id "java„
    id „application“
}

mainClassName = ‘com.example.HelloWorld’

//oder

application {
    mainClass = ‘com.example.HelloWorld'
}
```

Alternative das Plugin application verwenden. Dieses Plugin definiert u.a. den Task run. Die Main-Klasse muss weiterhin angegeben werden

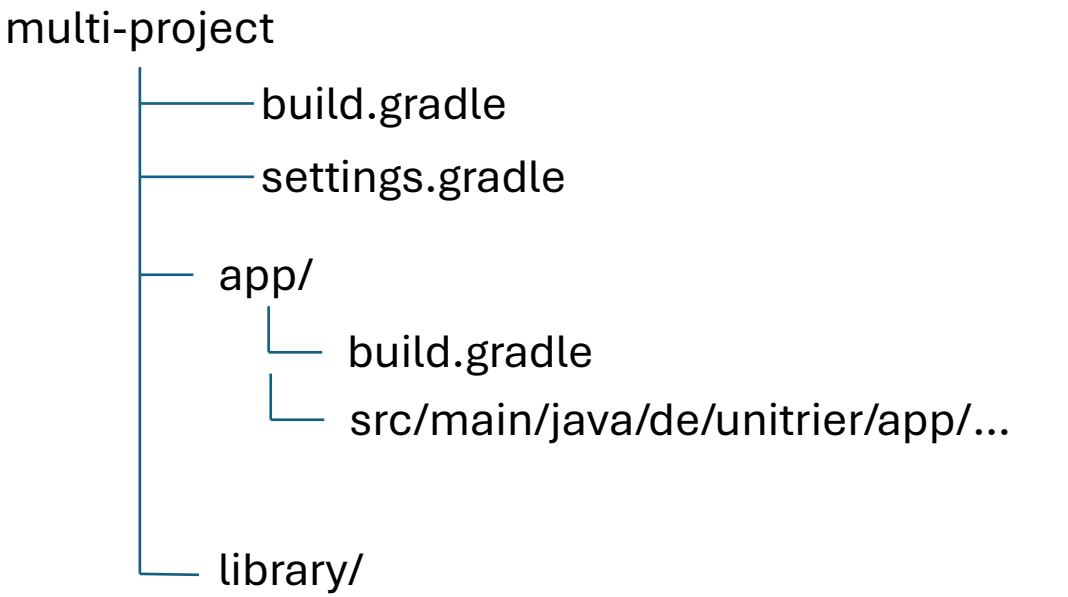
```
$ gradle run
```

Projektstruktur und sourceSets




```
//Default:
source: ’src/main/java’
test: ’test/main/java’

sourceSets {
    main {
        java {
            srcDir ‘src’
            exclude '**/ui/*.java'
        }
    }
    test
        java {
            srcDir ’test’
            include ,**/core/*.java'
        }
    }
}
```

Multi-Module-Projekte



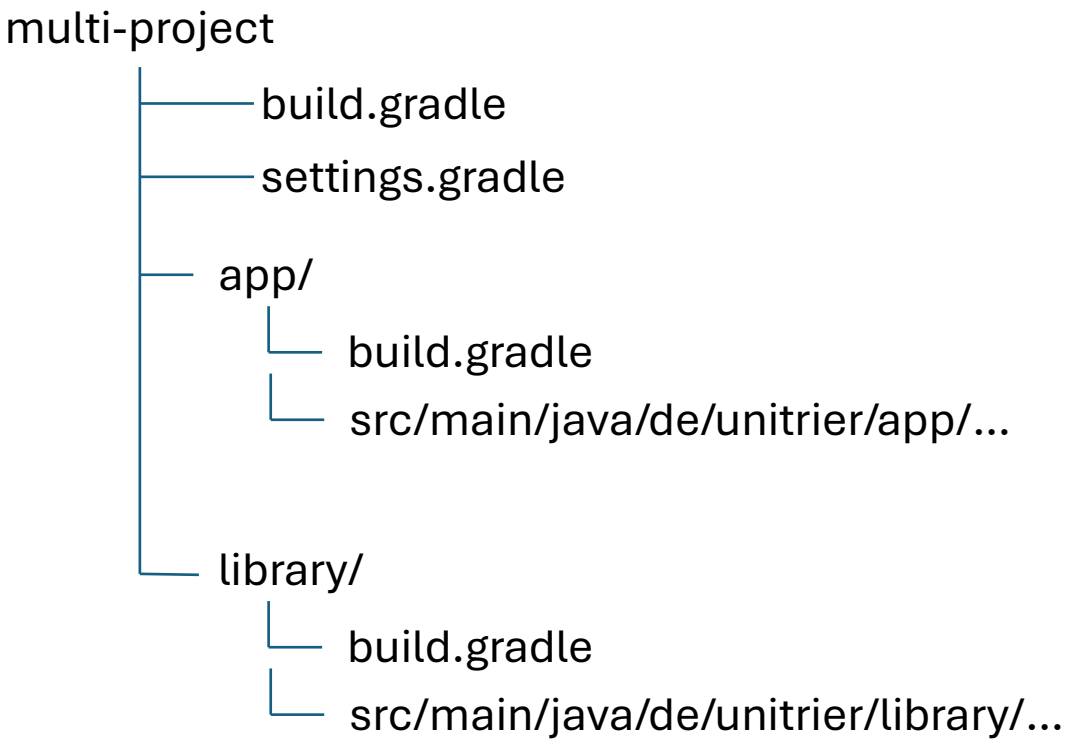

```
plugins {
    id 'java'
}

allprojects {
    repositories {
        mavenCentral()
    }
}

subprojects {
    apply plugin: 'java'

    dependencies {
        // Gemeinsame Test-Abhängigkeit für alle Subprojekte
        testImplementation 'junit:junit:4.13.2'
    }
}
```

Root build.gradle

```
rootProject.name = 'multi-project'
include 'app', 'library'
```

Settings.gradle

Multi-Module-Projekte




```
dependencies {
    implementation project(':library')
}
```

app/build.gradle

```
// Spezifische Abhängigkeiten für das library-Modul können hier
definiert werden
```

library/build.gradle

```
$ gradle build // Project bauen
$ gradle :app:run // App-Modul ausführen
```

ausführen

Beispiel 4.3

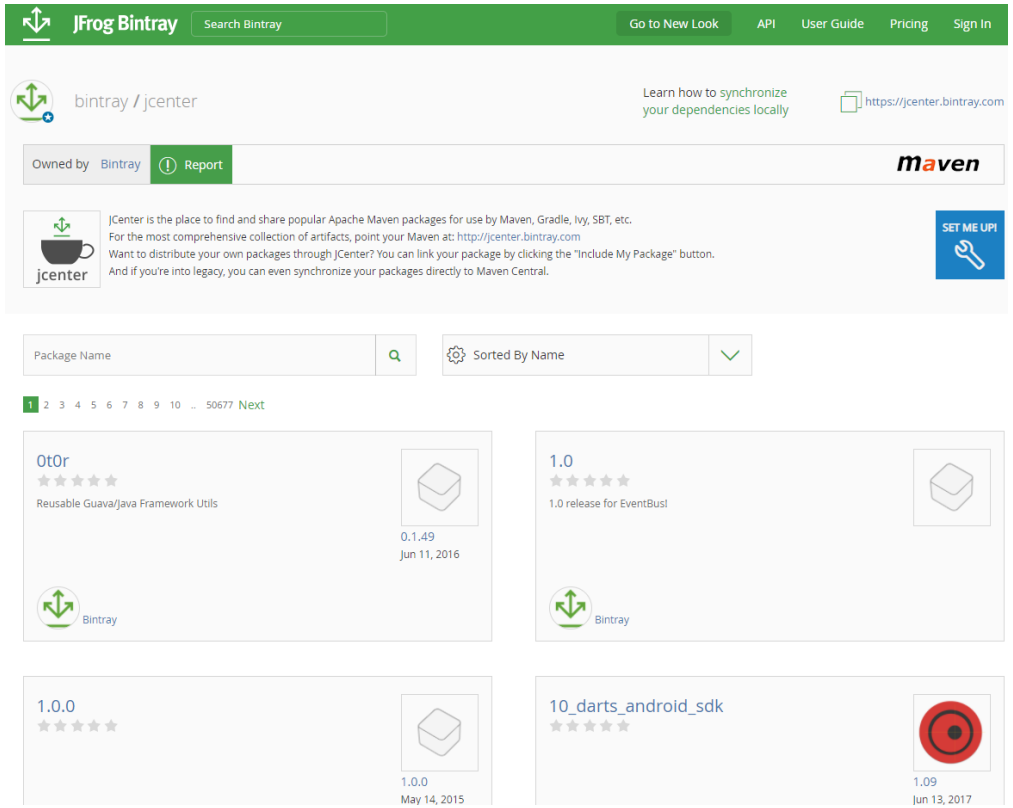
```
plugins {
    id 'org.openjfx.javafxplugin' version '0.0.8'
    id 'application'
}

javafx {
    version = '11.0.2'
    modules = [ 'javafx.controls‘]
}

application {
    mainClass = 'com.example.StockChartApp'
}

repositories {
    mavenCentral()
}
```

→ Verwende das Plugin javafxplugin, um JavaFX Anwendungen starten zu können

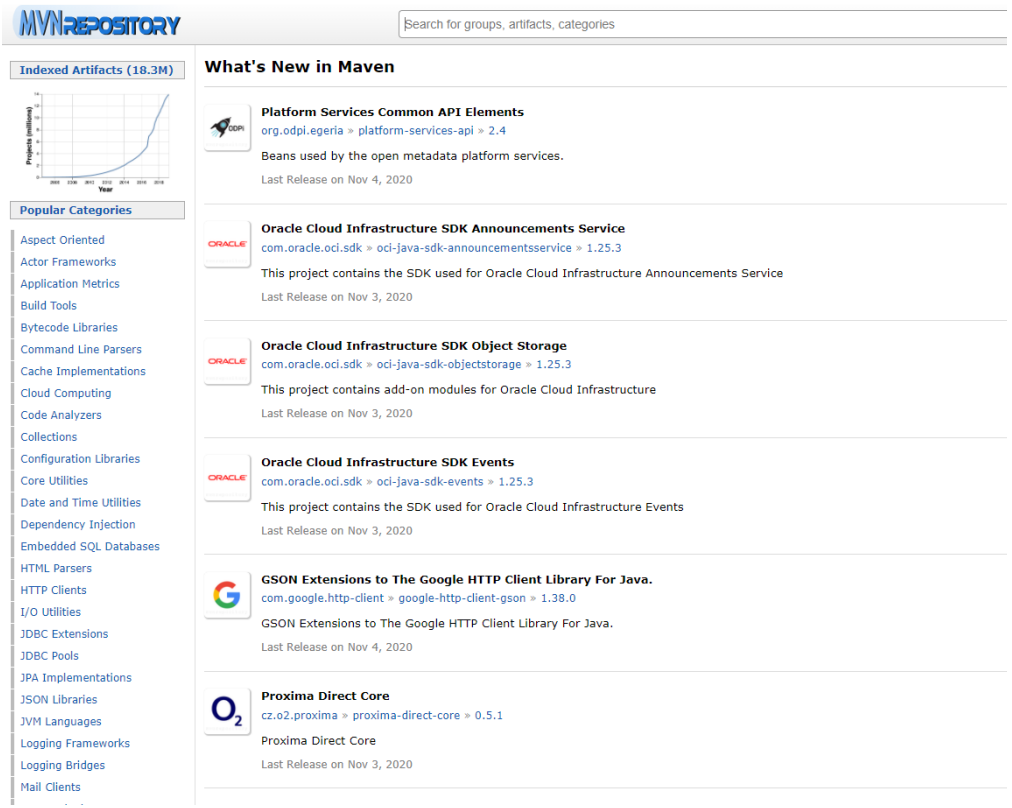
→ Angabe der benötigten Module

```
$ gradle run
```

Beispiel 5.1



jcenter()
(https://bintray.com/bintray/jcenter)



mavenCentral()
(https://mvnrepository.com/)

Beispiel 5.1

```
repositories {
    jcenter()
    mavenCentral()
}

dependencies {
    implementation group: 'group id', name: 'dependency name', version: 'version number’
    implementation 'com.google.code.gson:gson:2.11.0'
    implementation files('libs/lib1.jar', 'libs/lib2.jar’)
    implementation fileTree(dir: 'libs', include: '*.jar’)
    testImplementation group: 'junit', name: 'junit', version: '4.13.1’
}
```

→ Quellen, aus welchen Abhängigkeiten heruntergeladen werden sollen

→ Definiere Abhängigkeiten innerhalb des Projektes

Beispiel 5.2

```
plugins {
    id 'pmd'
}
```

Fügt die Tasks pmdMain, pmdTest und pmdSourceSet hinzu

PMD ist ein Analysetool zur statischen Codeanalyse

```
$ gradle pmdMain
$ gradle pmdTest
```

PMD report

Problems found

#	File	Line	Problem
1	C:\Users\lucas\IdeaProjects\FST-Gradle-Project\src\main\java\de\unitrier\st\core\DataLoader.java	12	Ensure that resources like this BufferedReader object are closed after use
2	C:\Users\lucas\IdeaProjects\FST-Gradle-Project\src\main\java\de\unitrier\st\core\DataLoader.java	14	Avoid assignments in operands
3	C:\Users\lucas\IdeaProjects\FST-Gradle-Project\src\main\java\de\unitrier\st\core\DataLoader.java	14	Found 'DU'-anomaly for variable 'str' (lines '14'-'27').

Zusätzlich existieren noch folgende Tools:

- Checkstyle
- Jacoco
- findbugs
- codenarc
- Jdepend

Beispiel 5.3

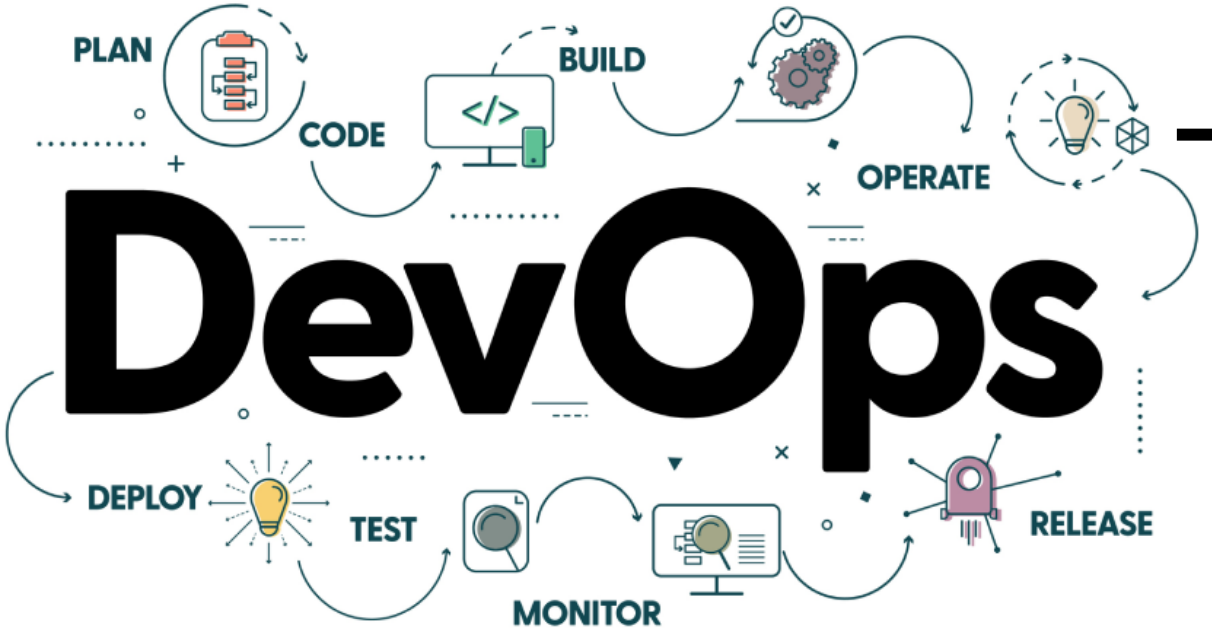
```
plugins {
    id 'java'
}

repositories {
    mavenCentral()
}

dependencies {
    implementation 'com.google.code.gson:gson:2.10.1'

    testImplementation group: 'junit', name: 'junit', version: '4.13.1'
}

task run(type: JavaExec) {
    main = 'com.example.GsonExample'
    classpath = sourceSets.main.runtimeClasspath
}
```

→ Testfälle werden standardmäßig in src/test/java/package gesucht

```
$ gradle test
```

DevOps

CD
Continuous Deployment



CI
Continuous Integration

https://www.datenschutz-praxis.de/wp-content/uploads/sites/2/2022/11/Datenschutz-DevOps-iStock-Chavapong-Prateep-Na-Thalang-full.webp

DevOps

DevOps ist eine Kultur und ein Ansatz, der Entwicklung (Development) und Betrieb (Operations) zusammenbringt, um die Softwarebereitstellung zu beschleunigen und die Qualität zu verbessern.

↓

Ziel ist es, die Zusammenarbeit zwischen Entwicklern und Betriebsteams zu fördern, Prozesse zu automatisieren und Feedback-Zyklen zu verkürzen.

↓

DevOps umfasst verschiedene Praktiken und Tools, darunter CI und CD, um die Automatisierung und Effizienz über den gesamten Lebenszyklus einer Anwendung zu verbessern.

	1970s–1980s	1990s	2000s–Present
Era	Mainframes	Client/Server	Commoditization and Cloud
Representative technology of era	COBOL, DB2 on MVS, etc.	C++, Oracle, Solaris, etc.	Java, MySQL, Red Hat, Ruby on Rails, PHP, etc.
Cycle time	1–5 years	3–12 months	2–12 weeks
Cost	$1M–$100M	$100k–$10M	$10k–$1M
At risk	The whole company	A product line or division	A product feature
Cost of failure	Bankruptcy, sell the company, massive layoffs	Revenue miss, CIO's job	Negligible

The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations

CI vs. CD

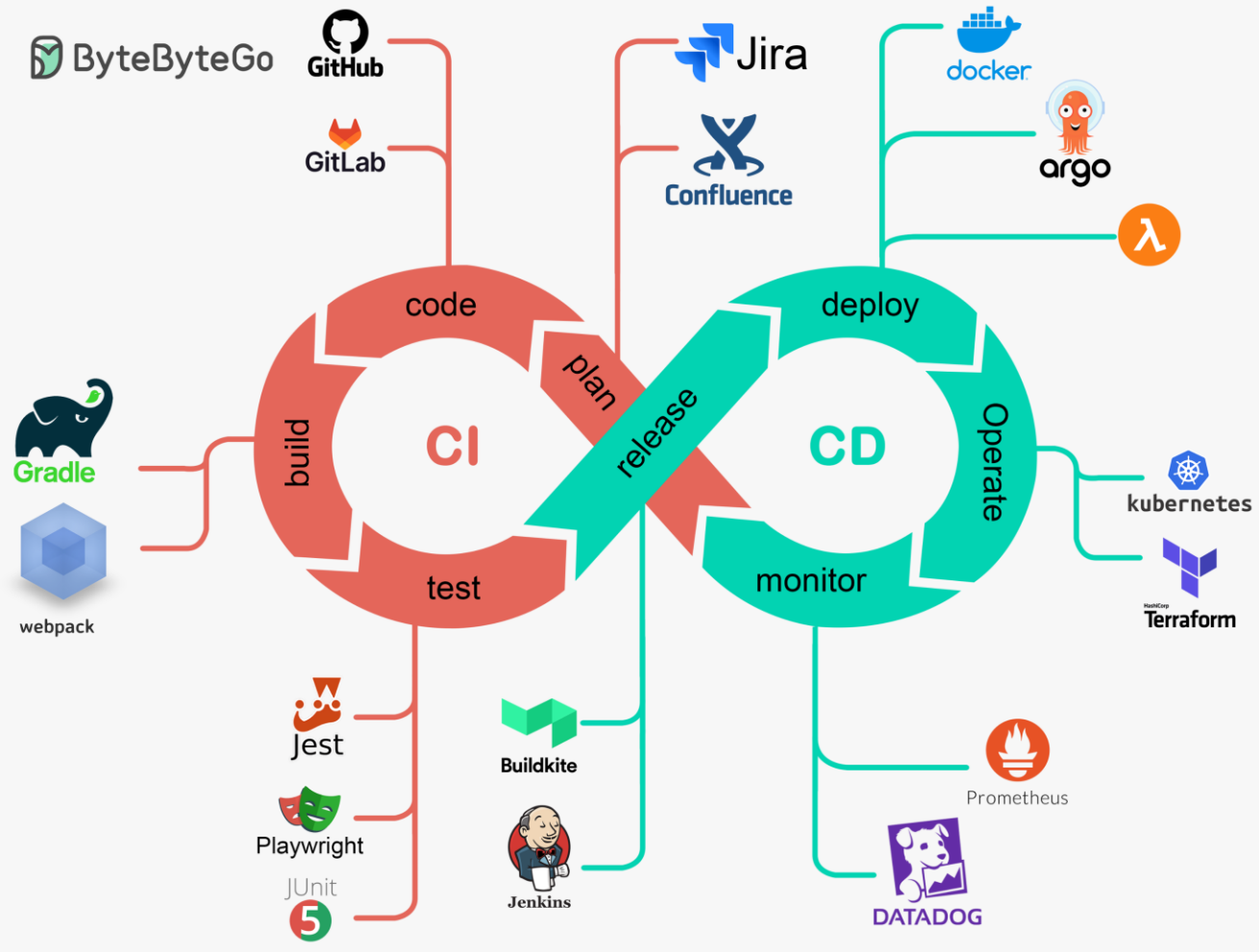
Continuous Integration (CI)

- Ziel: Automatisches Testen und Integrieren von Code-Änderungen
- Fokus auf Entwicklerproduktivität und Codequalität
- Integriert Code in ein gemeinsames Repository mehrmals täglich
- Automatisiert Tests, um Fehler frühzeitig zu erkennen
- Ermöglicht schnelles Feedback an Entwickler

Continuous Delivery/Deployment (CD)

- Ziel: Automatische Bereitstellung von Software in verschiedenen Umgebungen
- Fokus auf die Lieferung von funktionsfähiger Software
- Ermöglicht automatische Bereitstellung in Test-, Staging- oder Produktionsumgebungen
- Unterstützt kontinuierliche Auslieferung von neuen Funktionen an Endnutzer
- Minimiert Risiken durch häufige, kleine Releases

CI/CD Pipeline



https://blog.bytebytego.com/p/a-crash-course-in-cicd

Operation InVersion at LinkedIn (2011)

A Case Study:

The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations

Sechs Monate nach dem erfolgreichen Börsengang 2011 hatte LinkedIn weiterhin Probleme mit fehlerhaften Deployments. Diese wurden so gravierend, dass für zwei Monate die Feature-Entwicklung komplett gestoppt wurde, um die Computing-Umgebungen, Deployments und Architektur grundlegend zu überarbeiten.

LinkedIn wurde 2003 gegründet. Bereits in der ersten Woche existierten 2.700 Mitglieder, nach einem Jahr über eine Million. Seitdem wuchs die Plattform exponentiell. Im November 2015 hatte LinkedIn über 350 Millionen Mitglieder, die zehntausende Anfragen pro Sekunde und Millionen von Abfragen auf den Backend-Systemen erzeugten.

Anfangs lief LinkedIn hauptsächlich auf ihrer selbst entwickelten monolithischen Java-Anwendung "Leo", die Seiten über Servlets bereitstellte und JDBC-Verbindungen zu Oracle-Datenbanken verwaltete. Um jedoch dem steigenden Traffic gerecht zu werden, wurden zwei wichtige Dienste aus "Leo" ausgegliedert: einer für In-Memory-Abfragen des Mitgliedernetzwerks und ein weiterer für die Mitgliedersuche, der darauf aufbaute.

Bis 2010 fand die meiste Neuentwicklung in neuen Services statt und fast hundert Services liefen außerhalb von Leo. Das Problem war, dass **Leo nur alle zwei Wochen aktualisiert wurde**.

Josh Clemm, ein Senior Engineering Manager bei LinkedIn, erklärte, dass das Unternehmen bis 2010 erhebliche Probleme mit Leo hatte. Trotz vertikaler Skalierung durch zusätzlichen Speicher und CPUs fiel Leo häufig aus, Fehler waren schwer zu beheben und neue Code-Releases waren schwierig. Es wurde klar, dass man "Leo töten" und in viele kleine, funktionale und zustandslose Services aufteilen musste.

Operation InVersion at LinkedIn (2011)

2013 beschrieb die Journalistin Ashlee Vance von Bloomberg, dass „wenn LinkedIn versuchte, viele neue Dinge gleichzeitig hinzuzufügen, die Website in sich zusammenbrach, was die Ingenieure zwang, bis spät in die Nacht zu arbeiten, um die Probleme zu beheben.“ Im Herbst 2011 waren diese langen Nächte keine Übergangsriten oder Teamaktivitäten mehr, da die Probleme unerträglich geworden waren. Einige der besten Ingenieure, darunter Kevin Scott, der drei Monate vor dem Börsengang als VP of Engineering zu LinkedIn kam, beschlossen, die gesamte Entwicklungsarbeit an neuen Features zu stoppen und sich vollständig auf die Behebung der Kerninfrastruktur zu konzentrieren. Diese Initiative wurde "Operation InVersion" genannt.

„Man geht an die Börse, die ganze Welt schaut auf einen, und dann sagen wir dem Management, dass wir in den nächsten zwei Monaten nichts Neues liefern werden, während das gesamte Engineering-Team an diesem [InVersion]-Projekt arbeitet. Das war eine beängstigende Sache.“

Vance beschrieb die äußerst positiven Ergebnisse von Operation InVersion: „LinkedIn entwickelte eine ganze Reihe von Software und Tools, um den Code für die Website besser zu erstellen. Anstatt Wochen zu warten, bis neue Funktionen auf die Hauptseite von LinkedIn gelangten, konnten Ingenieure einen neuen Service entwickeln, automatisierte Systeme den Code auf Fehler und Probleme prüfen lassen und ihn direkt live schalten. … LinkedIns Engineering-Team führt nun dreimal täglich große Upgrades durch.“ Durch die Schaffung eines sichereren Arbeitssystems konnten sie weniger nächtliche Notfalleinsätze und mehr Zeit für die Entwicklung neuer, innovativer Funktionen gewinnen.

Wie Josh Clemm in seinem Artikel über Skalierung bei LinkedIn beschrieb: „Skalierung kann in vielen Dimensionen gemessen werden, einschließlich der Organisation. … [Operation InVersion] ermöglichte es der gesamten Engineering-Organisation, sich auf die Verbesserung von Tools und Deployments, Infrastruktur und Entwicklerproduktivität zu konzentrieren. Sie war erfolgreich darin, die notwendige Agilität zu schaffen, um die skalierbaren neuen Produkte von heute zu entwickeln. … 2010 hatten wir bereits über 150 separate Services. Heute sind es über 750 Services.“

Indem LinkedIn fast ein Jahrzehnt technischer Schulden abbauen konnte, ermöglichte Project InVersion Stabilität und Sicherheit und bereitete das Unternehmen auf die nächste Wachstumsphase vor. Allerdings erforderte dies zwei Monate völliger Konzentration auf nicht-funktionale Anforderungen, auf Kosten aller versprochenen Features, die während des Börsengangs angekündigt wurden. Indem Probleme als Teil der täglichen Arbeit gefunden und behoben werden, wird das technische Schuldenmanagement so gestaltet, dass solche „Nahtoderfahrungen“ vermieden werden.