Java Überblick

Ein Seminar für die Commerzbank

Warum ist Java so herausragend?

- Sie hatte das Glück, zum richtigen Zeitpunkt (1996) erfunden zu werden
- Java war die erste wirklich plattform-unabhängige Programmiersprache
  - Native Programmierung
    - Jede Fachvorgabe muss für jede Plattform eigens programmiert werden
    - Aufwand!
  - Abstraktion: Ausführbare Code ist “Bytecode” wird interpretiert von der Java Virtual Machine (JVM)

Vorsicht

- Heute ist Plattform-unabhängigkeit nicht mehr eine besonderes Merkmal von Java…
  - Rechner Installation mit CPU, RAM und Storage “auf Knopfdruck von einer Virtualisierungs-Software
  - VMware-Player oder Oracle Virtual Box stellen auch beliebige Umgebungen auf einer Desktop-Maschine zur Verfügung
  - Container-Lösungen z.B. mit Docker

Die Java Runtime

- Die Java Virtual Machine (JVM)
  - Wahrscheinlich die aktuell ausgereifteste Laufzeitumgebung für die Ausführung von Geschäftslogik
  - Plattform-unabhängige Ausführung von Bytecode
  - Ausgestattet mit ausgefeilten Sicherheitsmechanismen, die eine fehlerhafte Programmausführung “unkritisch” macht
    - Exception Mechanismus
  - Automatische “Garbage Collection”
  - HotSpot-Optimierungen

Java Spezifikationen

- Bytecode ist öffentlich einsehbar spezifiziert
- Auch die JVM ist spezifiziert
  - Kein Produkt!
    - Oracle JVM
    - IBM
- Frage: Was ist denn die aktuelle Java-Version?
  - Java 8 (häufig im Unternehmen noch der aktuelle Stand, obwohl von 2009)
  - Java 16
  - Long Time Support: Java 8, 11, 15
  - Die eigentlich Version ist Java **1**.8, **1**.11, **1**.15
- Damit ist Java auch heute noch komplett Abwärts-kompatibel

Java ist auch eine Programmiersprache

- Features
  - Statisch typisiert
    - Compiler prüft die Einhaltung von einer Menge von Regeln
      - var lastname = “Sawitzki”
      - lastname = 42 -> Compiler-Fehler!
  - Objekt-Orientiert
    - “Menschlich nachvollziehbare” Programmierung
      - new Person(“Sawitzki”, “Rainer”)
  - Java Compiler übersetzt ein Java-Programm in Bytecode

Es gibt auch andere Sprachen...

- Flexible, untypisierte Programmierung z.B. Scala
  - Vorteil: Weniger Programmcode notwendig
- Skript-Sprachen
  - Vorteil: Änderung des Programms “zieht sofort” z.B. Groovy
- Funktionale Programmierstil statt Objekt-Orientierung
  - Vorteil: Einfache Umsetzung von “Workflows” z.B. JavaScript

die aber auch Bytecode produzieren können!

- Scala-Compiler
  - .scale -> Bytecode
- Groovy-Compiler
  - .groovy -> Bytecode
- ...

Kategorien der Java Virtual Machine

Java Virtual Machine

- Standard Runtime, Java Standard Edition (JSE)
- Features
  - Bytecode-Interpreter
  - HotSpot
  - Garabage Collection
  - Robuster Umgang mit Fehler-Situationen
  - Überwachung und Erfassung von Metriken
- Programme
  - Berechnungen, “Algorithmen”
  - Ressourcen-Zugriffe auf das lokale Dateisystem, Netzwerk
  - Optional: Benutzeroberflächen (UI-Anwendungen)

Enterprise Java Virtual Machine = Application Server

- beruht auf der Standard-Edition
- Zusätzlich
  - Transaktioneller Zugriff auf Datenbank-Systeme
  - Server-Anwendungen für eine Vielzahl von Netzwerk-Protokollen
    - http
    - TCP/IP
    - REST
    - SOAP
    - Messaging
  - Authentifizierung und Autorisierung
  - Deployment einer Anwendung
    - bei Bedarf im laufenden Betrieb möglich

Hersteller und Produkte

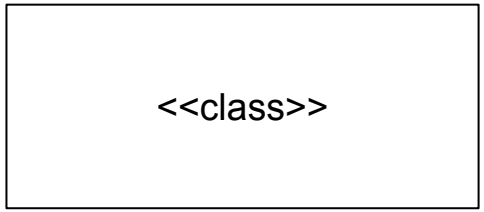
- Standard Edition
  - Oracle JVM
  - IBM (JRockit)
  - Einige wenige Nischen-Produkte
- Application Server mit einer reichhaltigen Produktpalette
  - IBM WebSphere
    - Recht groß (4Gbyte Download-Größe…)
    - Lizenz-pflichtig, kommerzieller Support verfügbar
  - Wildfly/JBoss
    - 200MByte
    - Open Source, kommerzieller Support von Red Hat
  - Apache Tomcat
    - 50 MByte
    - Ein Open Source http-Server
  - …

Die meisten Java Virtual Machines:

- Java Card Virtual Machines
  - praktisch auf allen SIM-Karten, Cards, Thermostaten, …
- Diese beruht natürlich nicht auf der Standard-Edition
  - Garbage Collection & HotSpot geht auf einer Karte natürlich nicht
- Trotzdem:
  - Interpreter für einen etwas eingeschränkten Satz von Bytecode-Anweisungen

Objekt-orientierte Programmierung

Modellierung



Exkurs: Klassen? Nicht Objekt-orientiert?

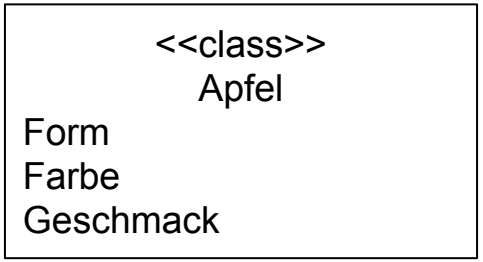
ein Apfel

rund
grün
süß

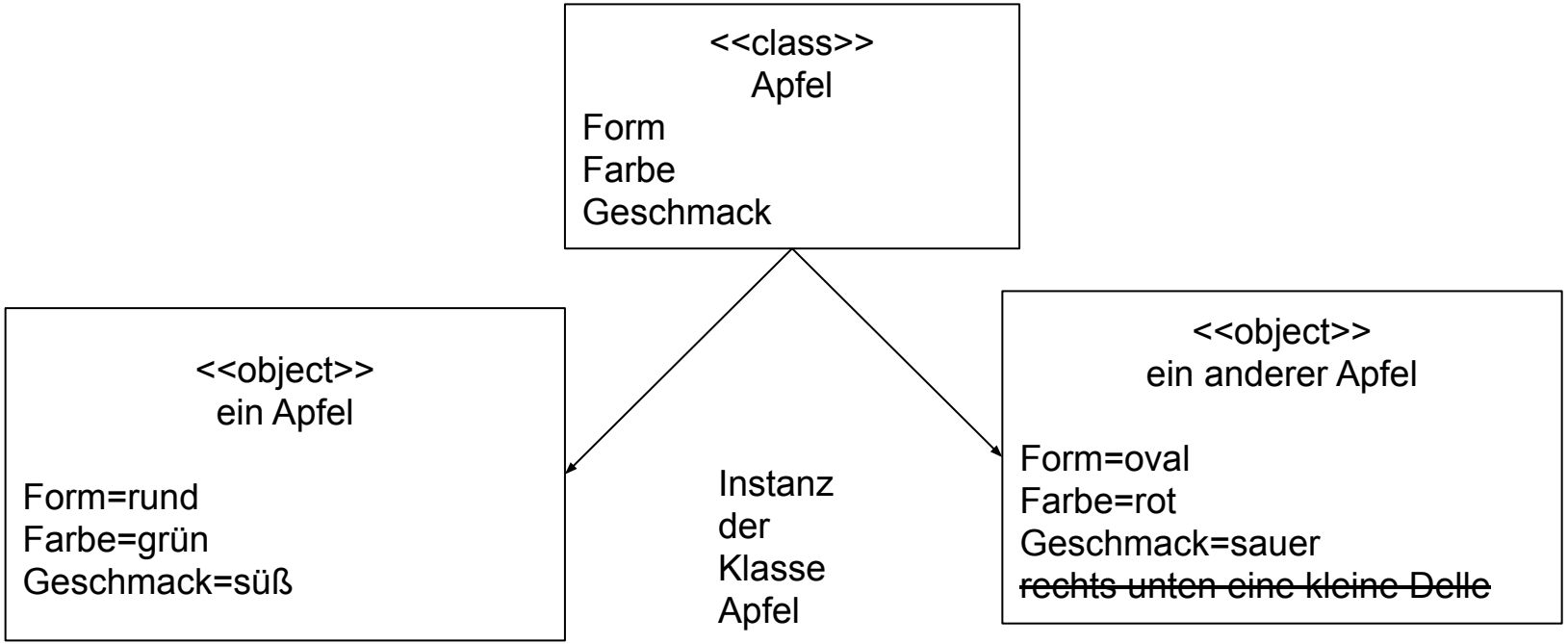
ein anderer Apfel

oval
rot
sauer
rechts unten eine kleine Delle

Modellierung: Die Klasse

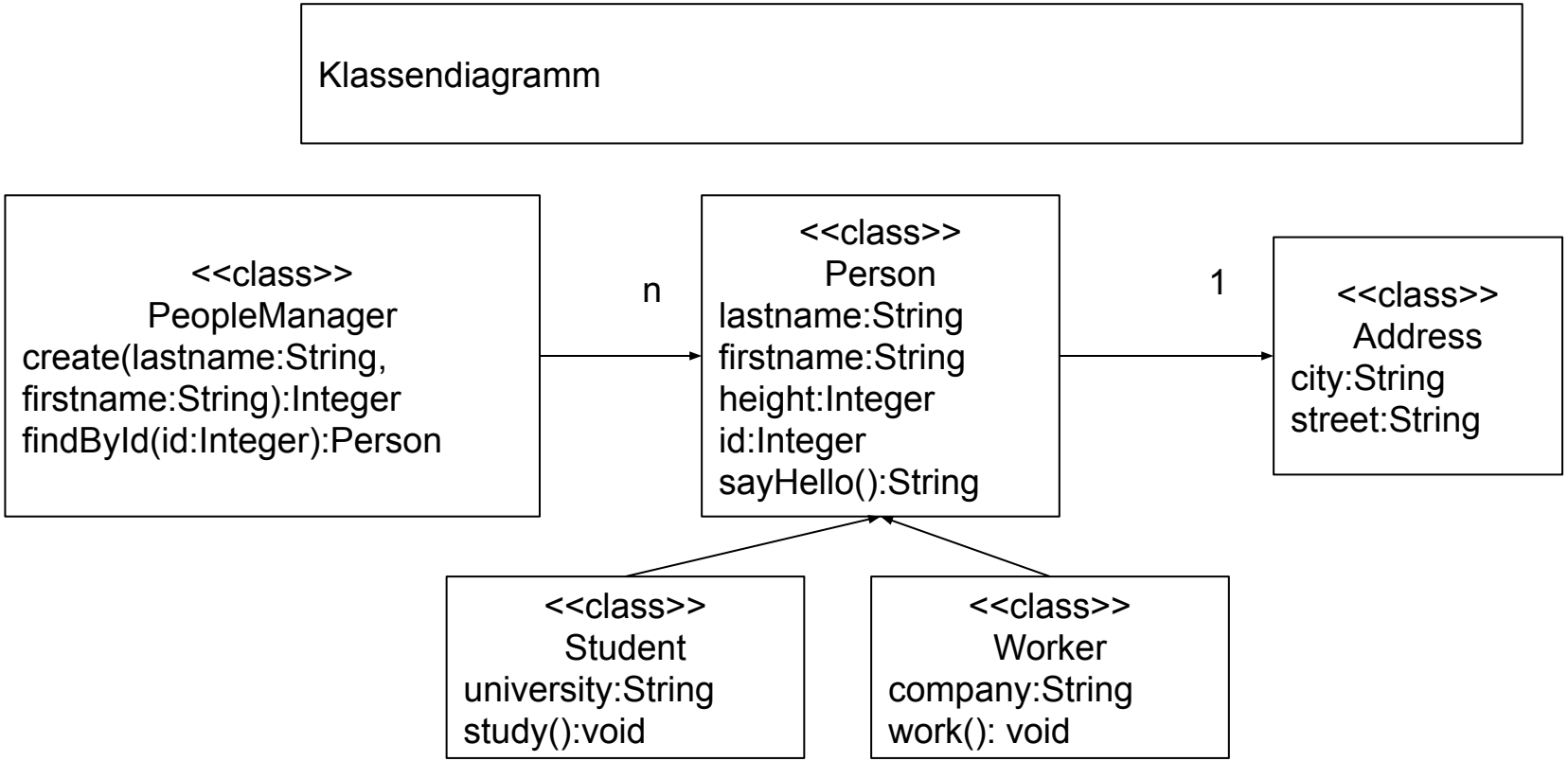


Modellierung: Klassen und Objekte



Eine reale Modellierung einer Anwendung

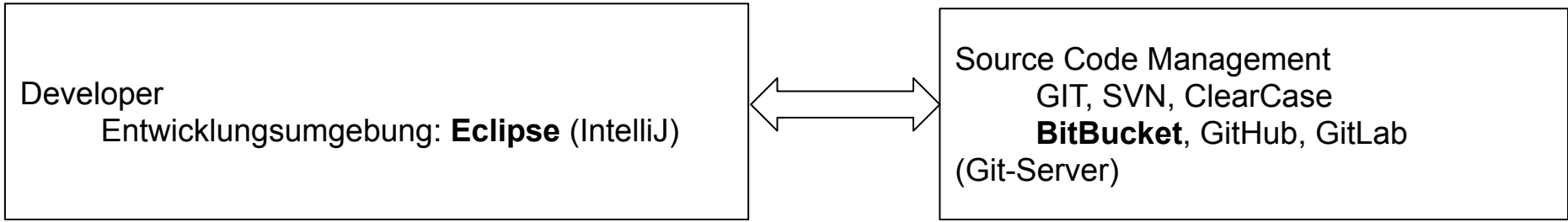
JAVACREAM
Training
Consulting
Projectmanagement



Exkurs UML

- Formale Sprache mit eindeutiger Syntax
  - “Pfeil mit einer geschlossenen Pfeilspitze -> Beziehung”
  - “Pfeil mit einer offenen Pfeilspitze -> Vererbung”
- Werkzeuge
  - Enterprise Architect
    - Sehr kompliziert…
  - Visio als Beispiel für ein Zeichenprogramm
  - Umletino (auch Online)
  - Whiteboard/Flipchart

Toolchain für die Anwendungs-Entwicklung



Realisierung

- Initial
  - https://github.com/Javacream/org.javacream.training.java.overview/commit/ccd45db8d42994c31ec4e89d70860c5dd60b8b94
- Verbose Java code
  - https://github.com/Javacream/org.javacream.training.java.overview/tree/0f3586407eec5705030080f057d2a8f56c0bd80d
- PeopleManager
  - https://github.com/Javacream/org.javacream.training.java.overview/tree/8c72c7f34e2e8abd3d74f39e7d7a9ef912f003a6

Beteiligte Rollen

- Fachlichabteilung
  - “Wir brauchen eine Anwendung zur Personen-Verwaltung”
  - Liefert: Prosa-Text
- Technische Designer
  - Übersetzt das Pflichtenheft
    - in ein Klassendiagramm
    - Spezifikation des Ablaufs
      - NEU, fehlt bis jetzt...
- Java-Entwickler
  - Schreibt den Quellcode für die konkrete Programmierung

Eine simple Ablauf-Spezifikation

- Eine angelegte Person kann über die ID wieder gefunden werden
- Körpergröße einer Person muss im Bereich 50 - 280 sein

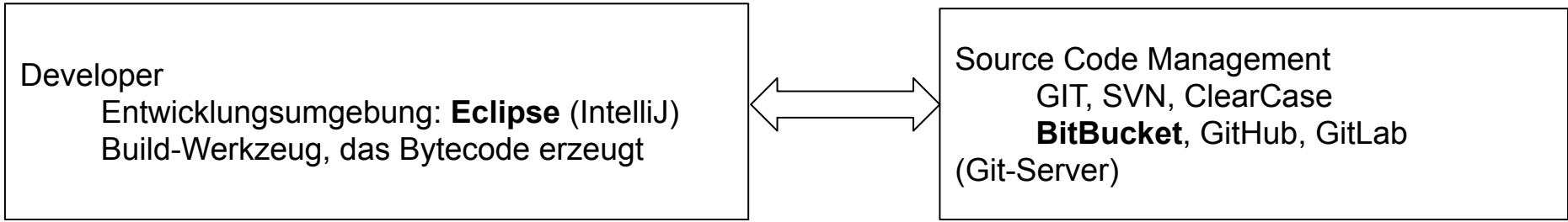
Beteiligte Rollen

- Fachlichabteilung
  - “Wir brauchen eine Anwendung zur Personen-Verwaltung”
  - Liefert: Prosa-Text
- Technische Designer
  - Übersetzt das Pflichtenheft
    - in ein Klassendiagramm
    - Spezifikation des Ablaufs
      - NEU, fehlt bis jetzt...
- Java-Entwickler
  - Schreibt den Quellcode für die konkrete Programmierung
- Tester
  - Schreibt den Test-Code an Hand der Spezifikation
  - Bevorzugt NICHT identisch mit dem Java Entwickler

Realisierung

- Initial
  - https://github.com/Javacream/org.javacream.training.java.overview/commit/ccd45db8d42994c31ec4e89d70860c5dd60b8b94
- Verbose Java code
  - https://github.com/Javacream/org.javacream.training.java.overview/tree/0f3586407eec5705030080f057d2a8f56c0bd80d
- PeopleManager
  - https://github.com/Javacream/org.javacream.training.java.overview/tree/8c72c7f34e2e8abd3d74f39e7d7a9ef912f003a6
- Tests
  - https://github.com/Javacream/org.javacream.training.java.overview/tree/5c0e5dcf1b0a5113757d71b556e4890b68b21fb6

Toolchain für die Anwendungs-Entwicklung



Test/QS

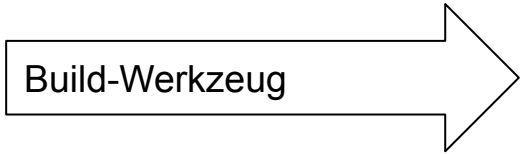
Production

Build-Prozess im Detail

- Java Quellcode
  - .java
- Compiler
  - Konfigurierbares Werkzeug
  - Portabel
    - Compilieren unter Windows/Linux/Mac/… erzeugt binär eindeutiges Ergebnis liefern
  - Reproduzierbarkeit
- Java Bytecode
  - .class

Build-Prozess

Werke
- Quellcode
- Konfigurationsdatei
- **Build-Konfiguration**
- …
- Bestandteil des SCM

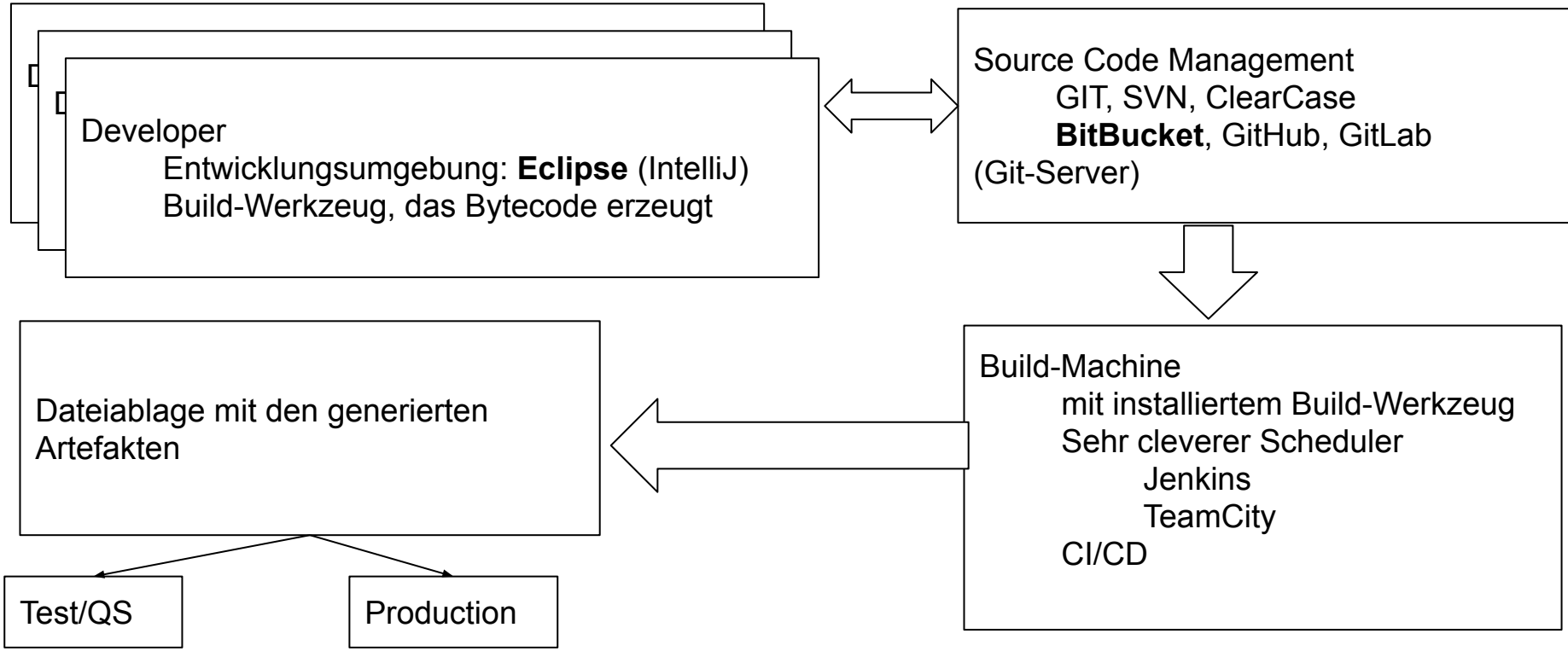


Artefakt
- z.B. Bytecode
- Sind NICHT Bestandteil des Source Code Managements

Build-Konfiguration

- Sollte unabhängig von einer proprietären Entwicklungsumgebung sein
- Java
  - Apache Maven
    - pom.xml
  - Gradle
    - build.gradle
  - (Apache ANT)
    - build.xml

Toolchain für die Anwendungs-Entwicklung



CI/CD

- Continuous Integration
- Continuous Delivery/Deployment
- Developer stellt seine Werke neu im SCM zur Verfügung
  - Build Machine baut die Artefakte neu (CI)
  - Optional: Artefakte werden automatisch an Test/QS bzw Production geliefert (Continuous Delivery) oder sogar im laufenden Betrieb neu installiert (Continuous Deployment)

Einzelne Schritte eines Build-Prozesses

- Config-Management, z.B. Profiles
- compile
- testing
- automatisierte Bestimmung relevanter Qualitäts-Metriken
- Reporting
- In der Summe ist das eine so genannte CI/CD-”Pipeline”, die aus einzelnen “Stages” besteht

Beispiel

- Maven-Projekt unter
  - https://github.com/Javacream/org.javacream.training.java.overview/tree/79e85501681981fdf0e13cadb2a1e4be1a178d09
-

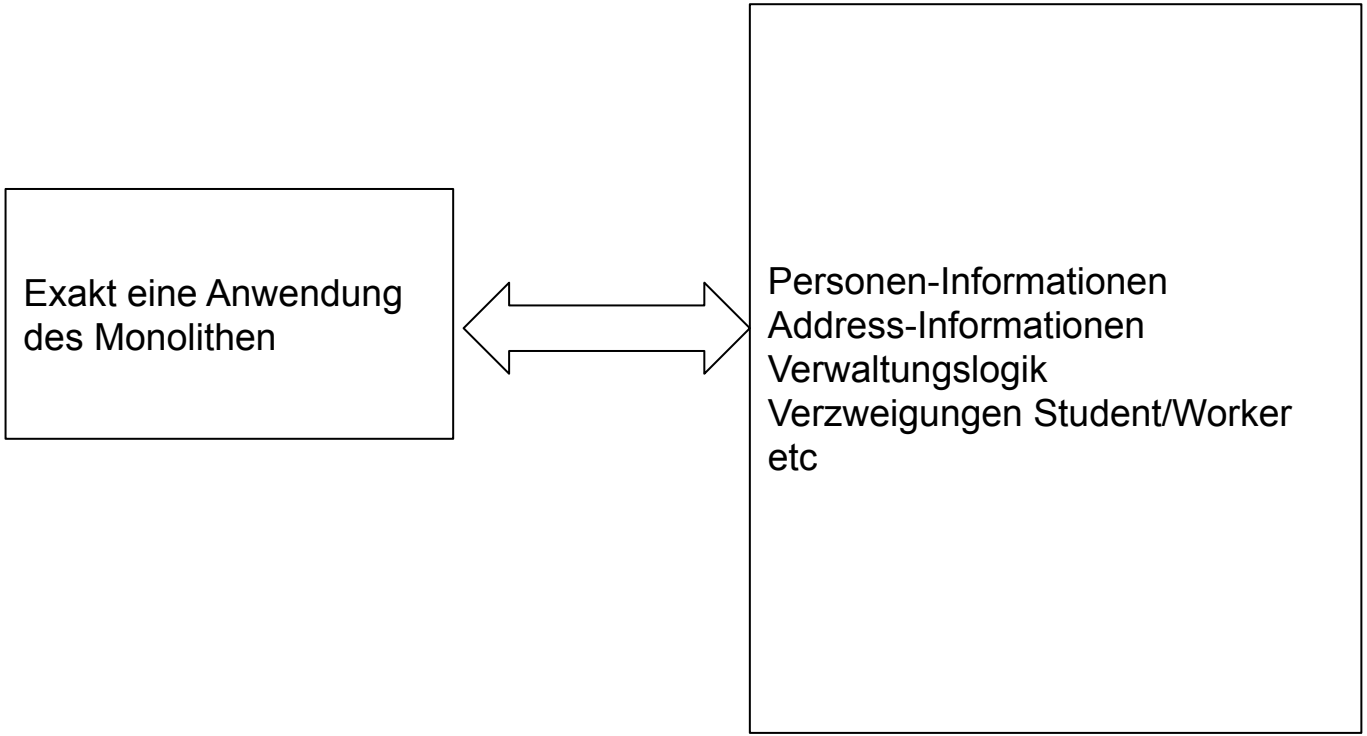
Werke und OOP

Elementare Regel der Modellierung

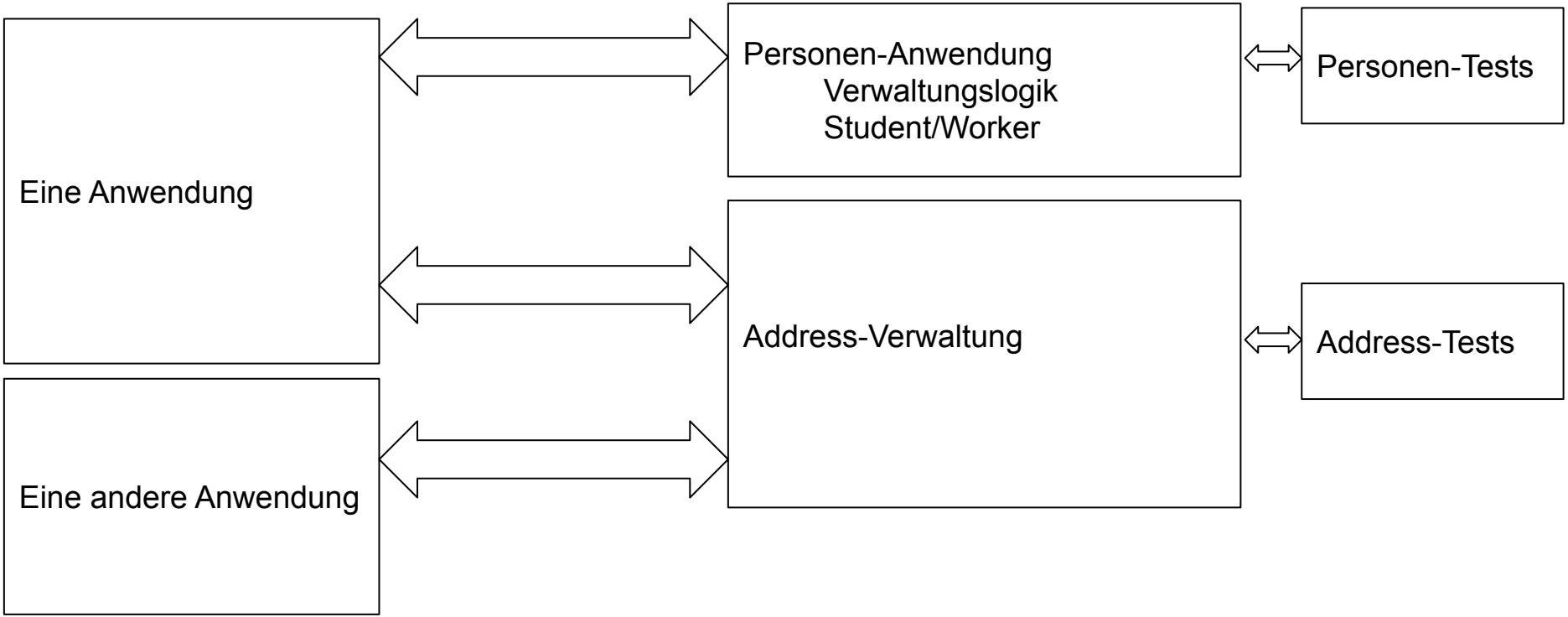
JAVACREAM
Training
Consulting
Projectmanagement

- “Jede Klasse sollte exakt eine wohl definierte Aufgabe übernehmen”
- Konsequenzen
  - Selbst einfache Anwendungen haben eine Vielzahl von Werken und damit Klassen
  - Wartbarer, Wiederverwendbarer und Testbarer Code ist damit “sehr gut erreichbar”

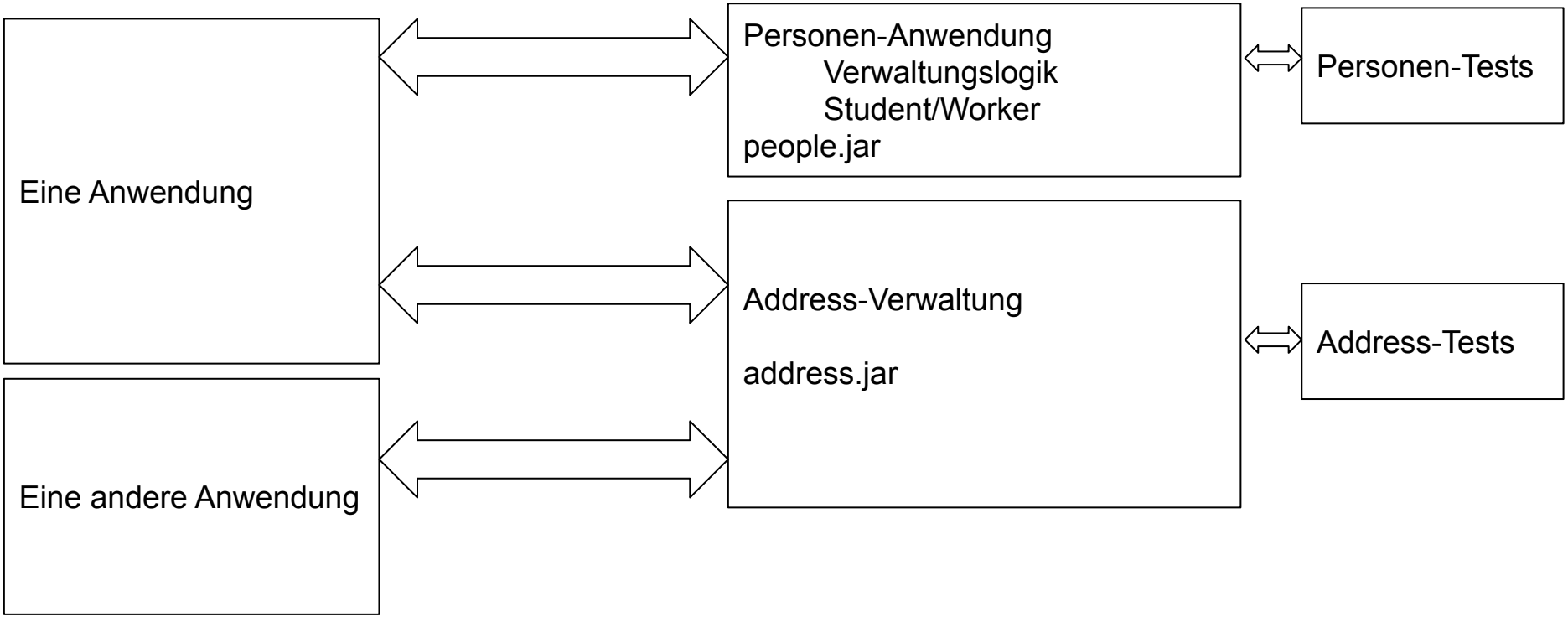
Monolith


Exakt eine Anwendung des Monolithen
Personen-Informationen
Address-Informationen
Verwaltungslogik
Verzweigungen Student/Worker
etc

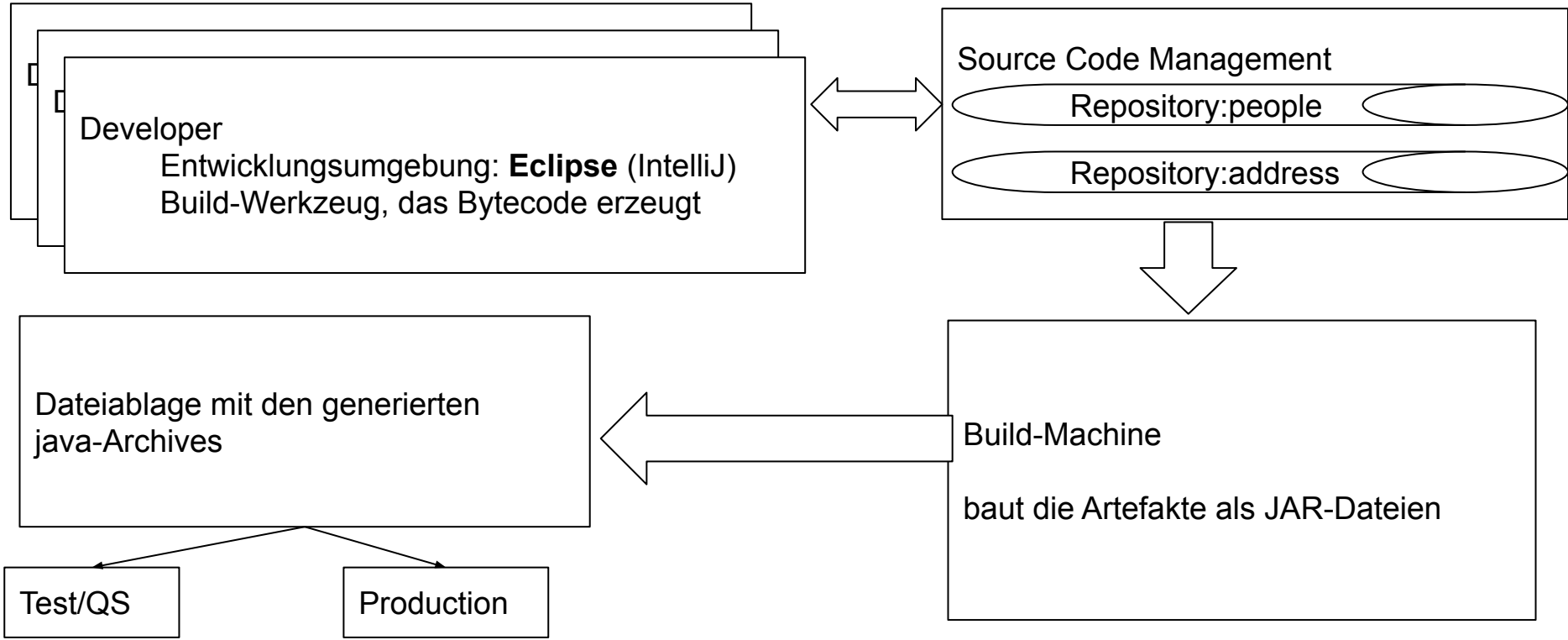
Modulare Anwendung



Module und Artefakte in der Java-Welt



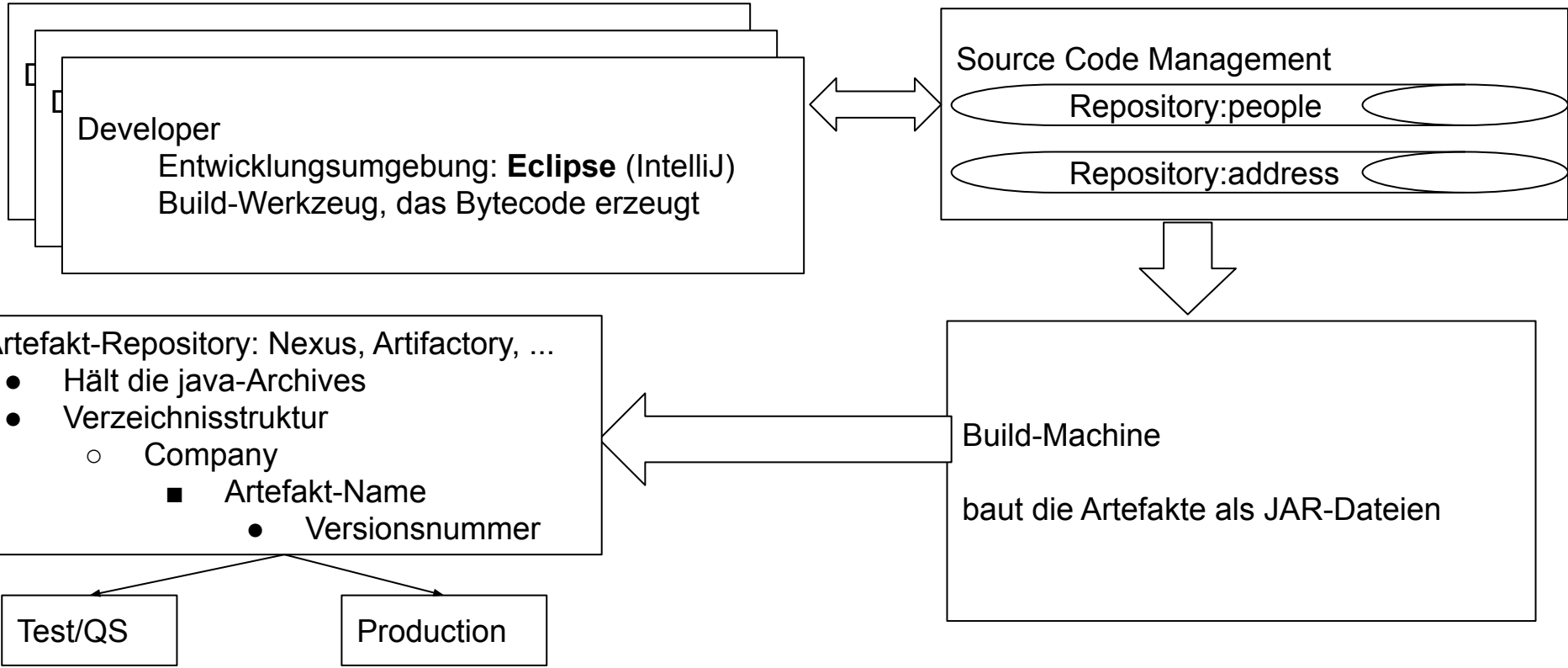
Toolchain für die Anwendungs-Entwicklung



Beispiel: Personen-Anwendung

- Repository im GitHub nimmt die Werke auf
  - people-Repository
- Build-Machine erzeugt daraus das Java-Archiv mit den Bytecodes der Anwendung
  - people**-1.0.3**.jar
    - Person.class
    - Student.class
    - PeopleManager.class
    - Worker.class

Toolchain für die Anwendungs-Entwicklung

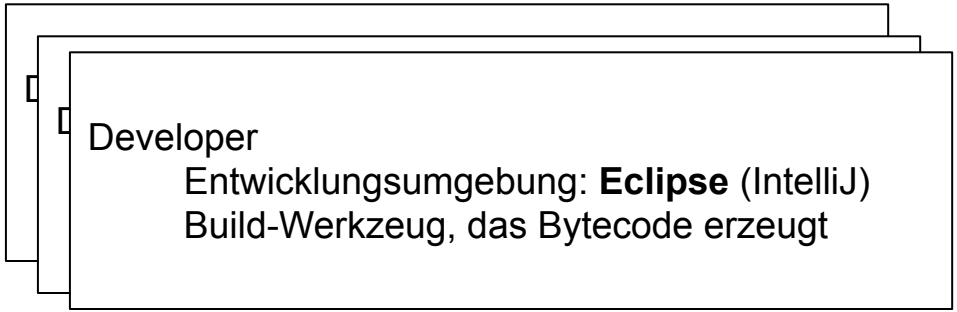


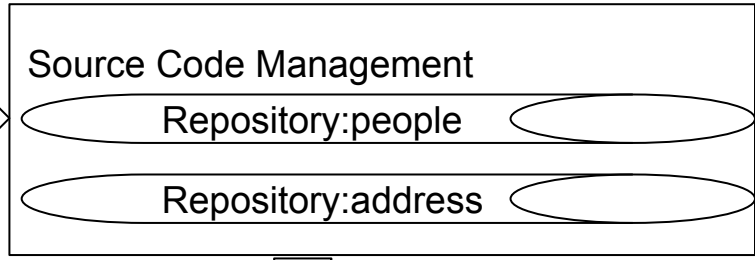
Ein Artefakt-Repository eines Software-Projekts

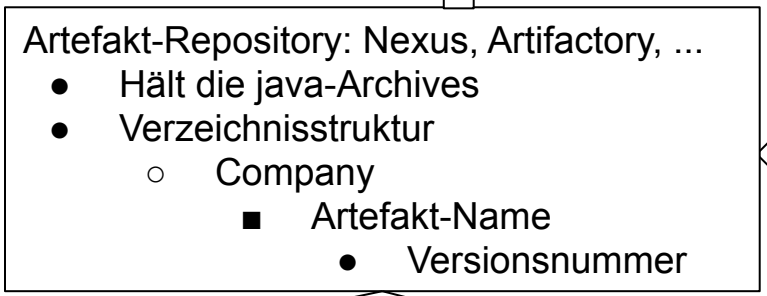
- Open Source Community in Java
  - Vielzahl von Produkten und Frameworks/Libraries
    - Zig-Tausende
  - https://repo.maven.apache.org/maven2/
-

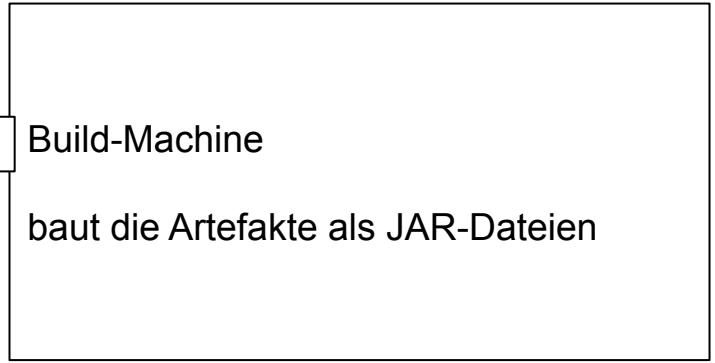
Toolchain für die Anwendungs-Entwicklung

JAVACREAM
Training
Consulting
Projectmanagement









Einzelne Schritte eines Build-Prozesses

- **Dependency Management**
  - **Auflösen aller in der Build-Definition enthaltenen Dependencies auf andere Bibliotheken**
  - **Inklusive Transitive Dependencies**
- Config-Management, z.B. Profiles
- compile
- testing
- automatisierte Bestimmung relevanter Qualitäts-Metriken
- Reporting
- In der Summe ist das eine so genannte CI/CD-”Pipeline”, die aus einzelnen “Stages” besteht

Ein paar kleine Fragen

- Kann ein Entwickler eine beliebige Bibliothek aus dem Internet laden und in sein Projekt einbinden?
  - Ist das wünschenswert?
  - Wie kann das Verhindert werden
  - Was würde passieren, wenn er es trotzdem tut?
- Was sollte passieren, wenn für eine Bibliothek eine Security-Lücke (WARNING, CRITICAL) gemeldet wird?
  - Wer sollte diese Meldung verarbeiten?
  - Was sollte dann passieren?

Eine komplexere Anwendung

Fachliche/Technische Anforderungen

- Anbindung an ein Backend
  - Datenbank wie Oracle, MySQL, …
  - Viele aufwändige und komplexe Codezeilen
    - Connection Pool
    - Transaction Manager
    - SQL-Statements erzeugen und absetzen
- Implementierung der Verarbeitungslogik
  - Kann auch sehr komplex sein...
- Zugriff über Netzwerk
  - Client-Server-Architektur
  - Viele aufwändige und komplexe Codezeilen
    - Authentifizierung
    - Parallelisierung
    - Streaming

Fachliche/Technische Anforderungen

- Anbindung an ein Backend
  - Datenbank wie Oracle, MySQL, …
  - Viele aufwändige und komplexe Codezeilen
    - Connection Pool
    - Transaction Manager
    - SQL-Statements erzeugen und absetzen
- Implementierung der Verarbeitungslogik
  - Kann auch sehr komplex sein…
  - Komplexer durch die Integration der Frameworks
- Zugriff über Netzwerk
  - Client-Server-Architektur
  - Viele aufwändige und komplexe Codezeilen
    - Authentifizierung
    - Parallelisierung
    - Streaming

In der Java-Welt

- Sehr etablierte Frameworks sind vorhanden
- (Java) -> Jakarta Enterprise Edition
  - Programmiermodell mit diversen Spezifikationen
    - z.B. “Java Persistence API), JPA
    - z.B. JAX-RS für http-basierte Anwendungen
    - …
  - Spezifikation für den Application Server
  - Wertung: Application Server und die einzelnen Spezifikationen sind OK, der Rest ist tot
- De Facto-Standard: Spring Framework
  - Programmiermodell, das die JEE-Spezifikationen verbessert und erweitert

Beispiel mit Spring

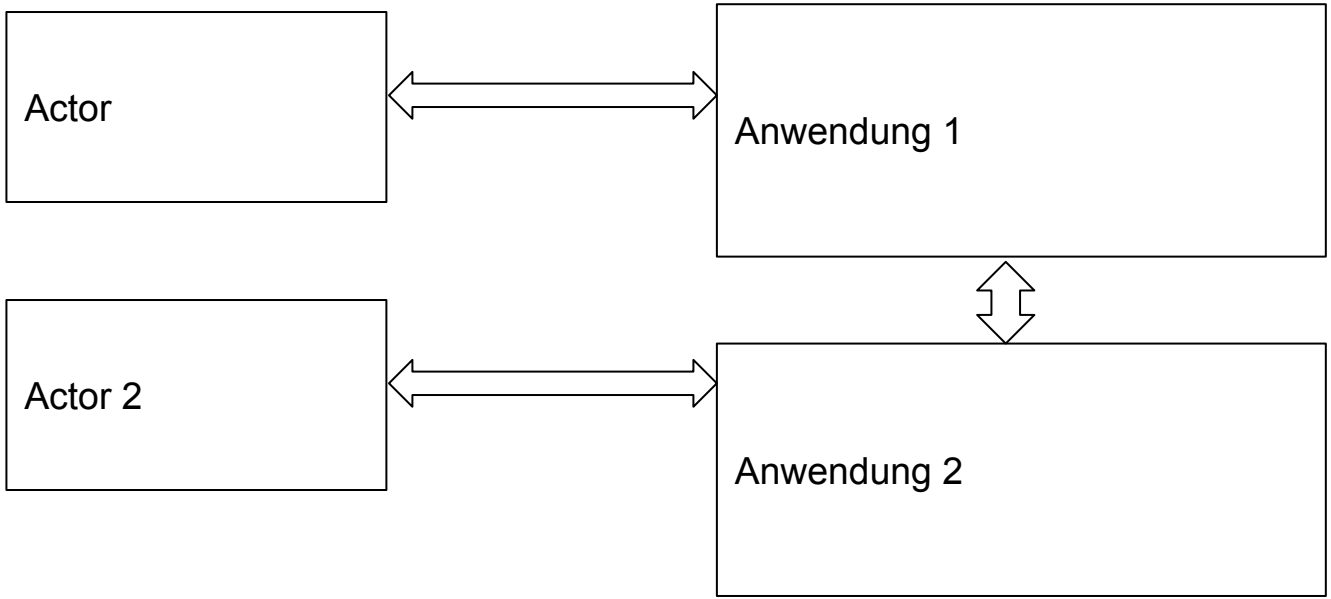
- https://github.com/Javacream/org.javacream.training.java.overview/tree/d8dba31d73f6f393911ab7ca1aade4b9abd1c8c6
-

Services

Kriterien für eine qualitativ hochwertige Software

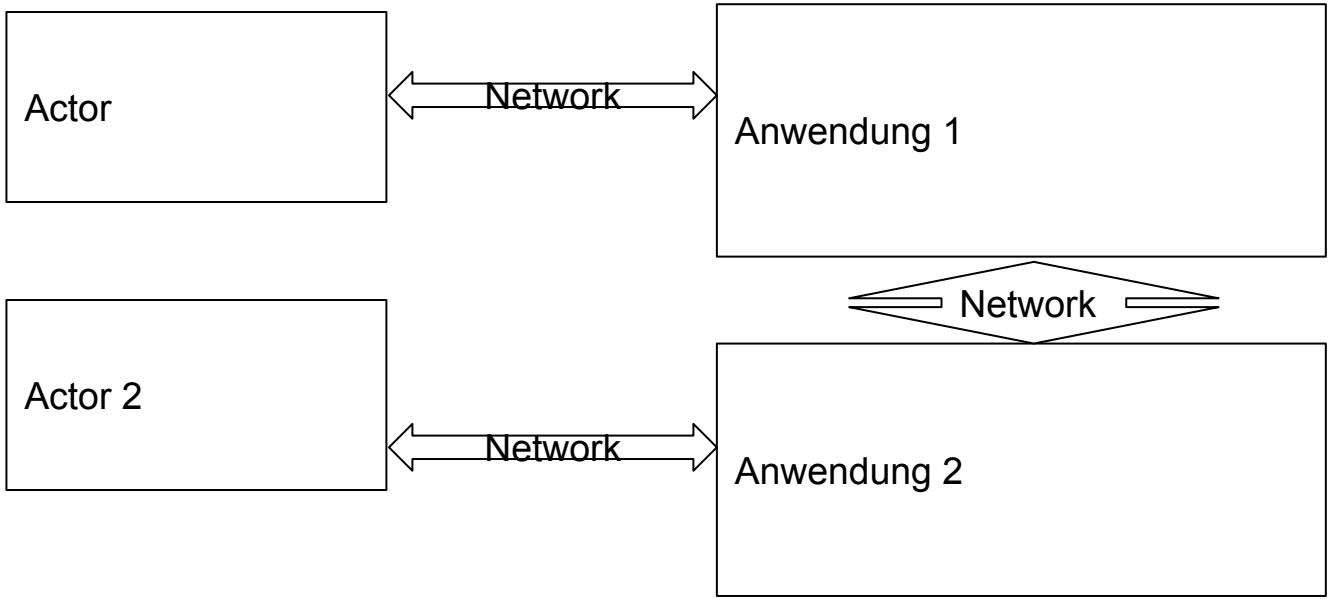
- Wartbar
- Testbar
- Wiederverwendbar

Wiederverwendbarkeit von Anwendungen



Anwendung 2 hat zwei Actors, die damit die Logik wiederverwenden

Wiederverwendbarkeit Services



Anwendung 2 hat zwei Actors, die damit die Logik wiederverwenden

Notwendige Vorbedingung der Wiederverwendbarkeit

- “Wenn ich einen Service verwenden möchte, muss ich natürlich wissen, was er eigentlich macht...”
- Service-Beschreibung ist unbedingt notwendig!
  - = vollständige Dokumentation
- Minimal:
  - Datenstrukturen
  - Operationen

Beispiel: Datenstrukturen

PeopleService

Person
- lastname
- firstname
- id

Operations:

BooksService

Book
- isbn (= id)
- title
- price

Operations:

InvoiceService

Invoice
- id
- totalPrice
- date

Operations:

Service-Operationen

- CRUD
  - Create, Read, Update, Delete
  - Read-Operationen müssen wohl noch gesondert parametrisiert werden
    - findByLastname - findBytitle - findByDateRange
- Rückgabe:
  - Datenstruktur oder eine Liste davon ergänzt um einen Status
- Daten-Format
  - Was schicke ich? (z.B. Plain Text)
  - Was erwarte ich? (z.B. PDF)
- Versions-Identifier, z.B. ein Hashwert
- Cache Policy
- Authentifizierung

Abstrakte Service-Architektur: REST

- Damit ist eine vereinfachte Dokumentation möglich!

Konkrete Realisierung von REST: RESTful WebServices

- Mapping von REST nach http
  - CREATE -> POST
  - READ -> GET
  - UPDATE -> PUT, auch möglich PATCH
  - DELETE -> DELETE
- Status sind die HttpStatus, z.B. 404 (Not found)
- Daten-Format sind die “MediaTypes”, text/plain, application/pdf
  - De Facto Standard: JSON
- Versions-Identifier durch das ETag
- Cache Policy
- Authentifizierung: (BASIC, FORM, CERTIFICATE), OpenId (oAuth)

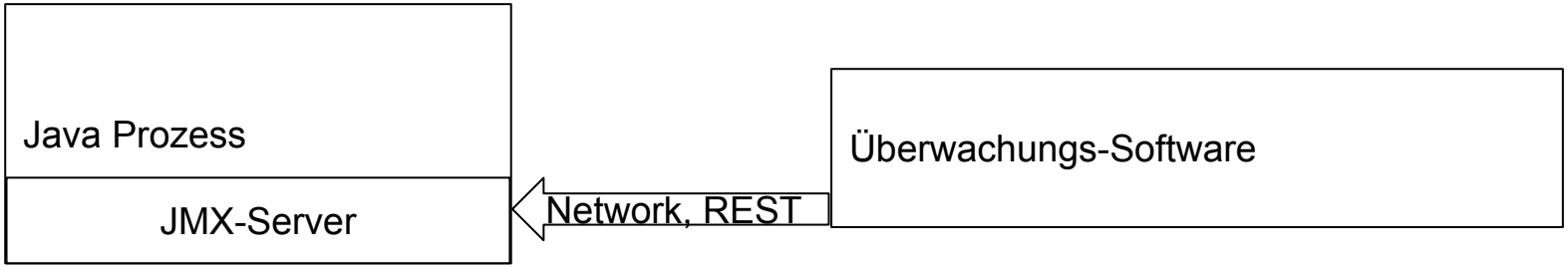
Service-Beschreibung bei REST ist bereits standardisiert

- Open API als Spezifikation eines Industrie-Konsortiums
- Open-API Beschreibungen werden
  - Erstellt und Modelliert mit Werkzeugunterstützung
    - Swagger

Zurück in den Betrieb von Java-Anwendungen

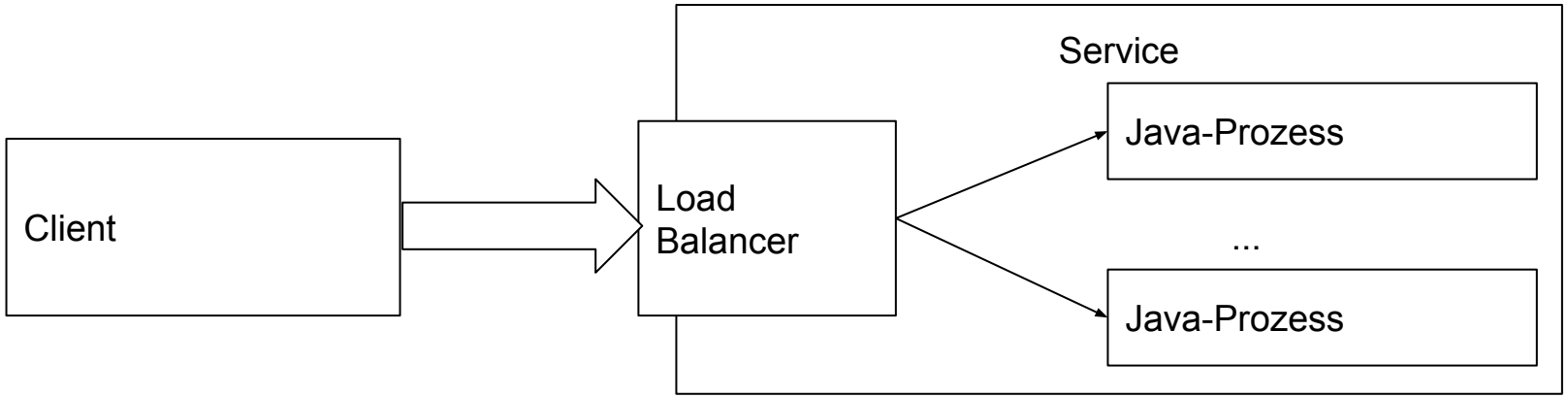
Überwachung

- Faktisch wieder standardisiert
  - JMX (Java Management Extension)
  -



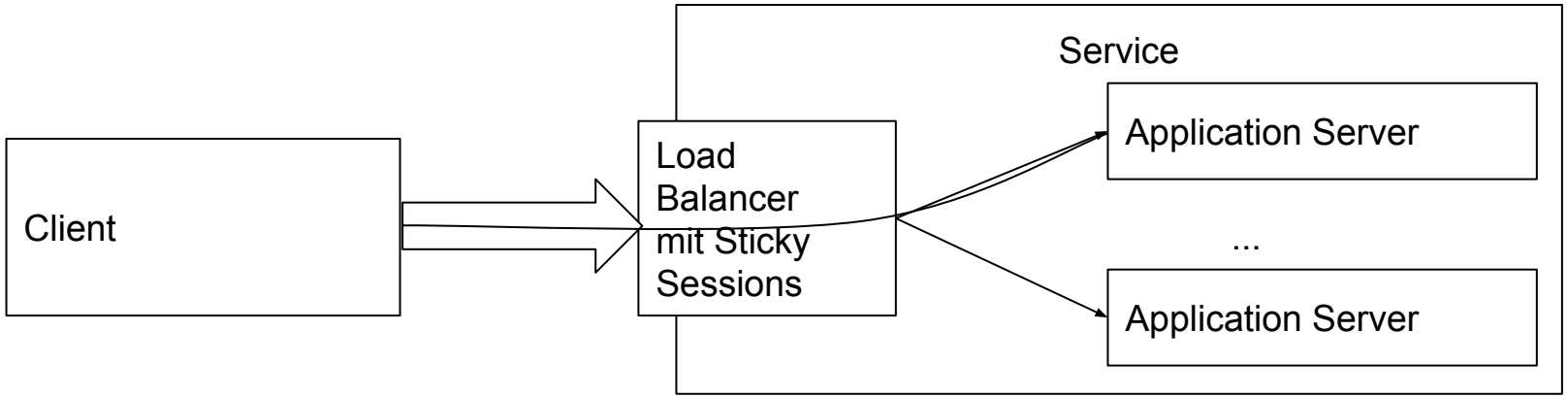
Betriebssystem-nahe Informationen
    CPU, Speicher, IO
Java-typisch
    Garbage Collection: Wie oft, wie effizient
Anwendung
    Anzahl und durchschnittliche Antwortzeit für Request

Skalierbarkeit von Java-Anwendungen



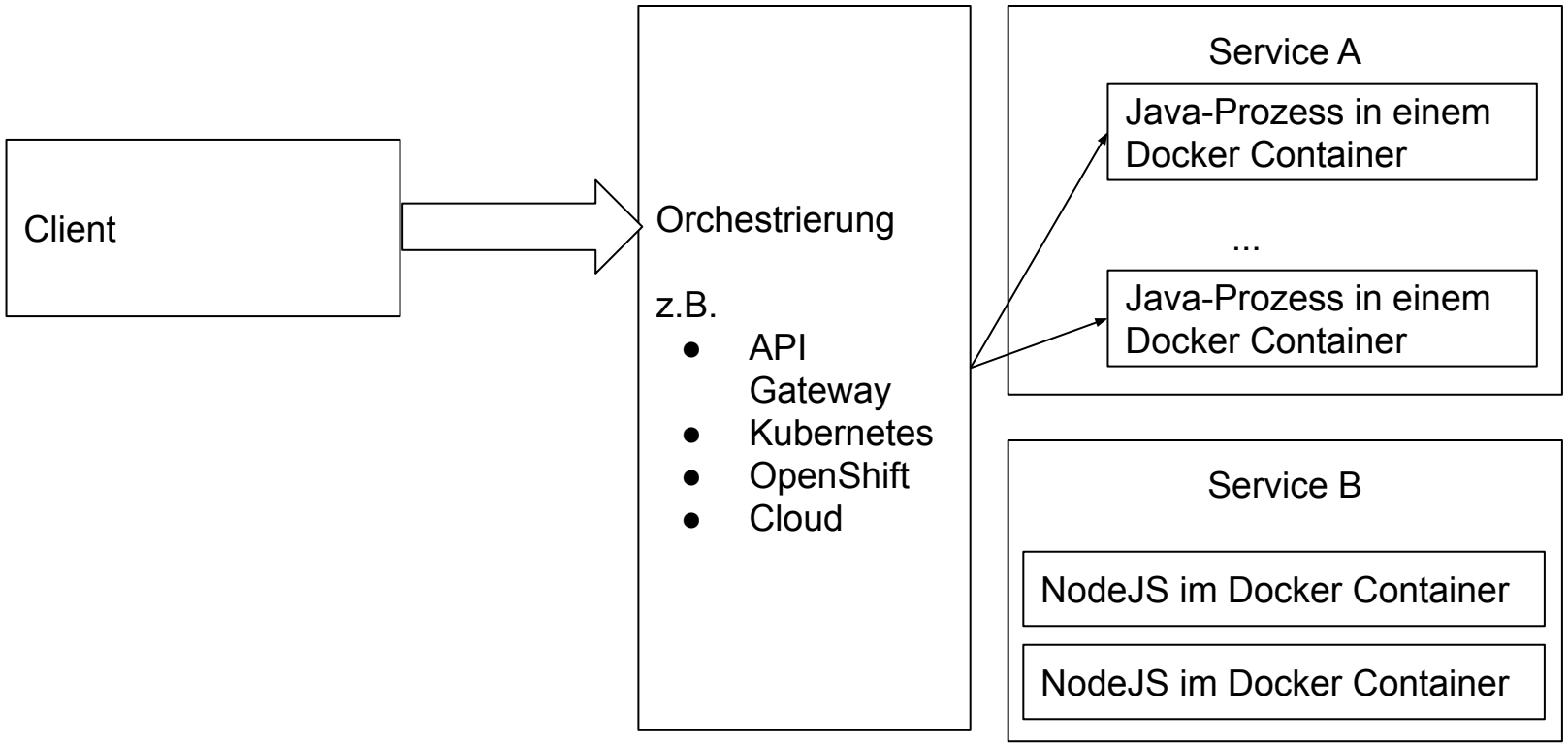
“Stateless Architecture”

Skalierbarkeit von Java-Anwendungen



“Stateful Architecture” mit so genannten “Sessions”

Dynamische Cluster mit Orchestrierung



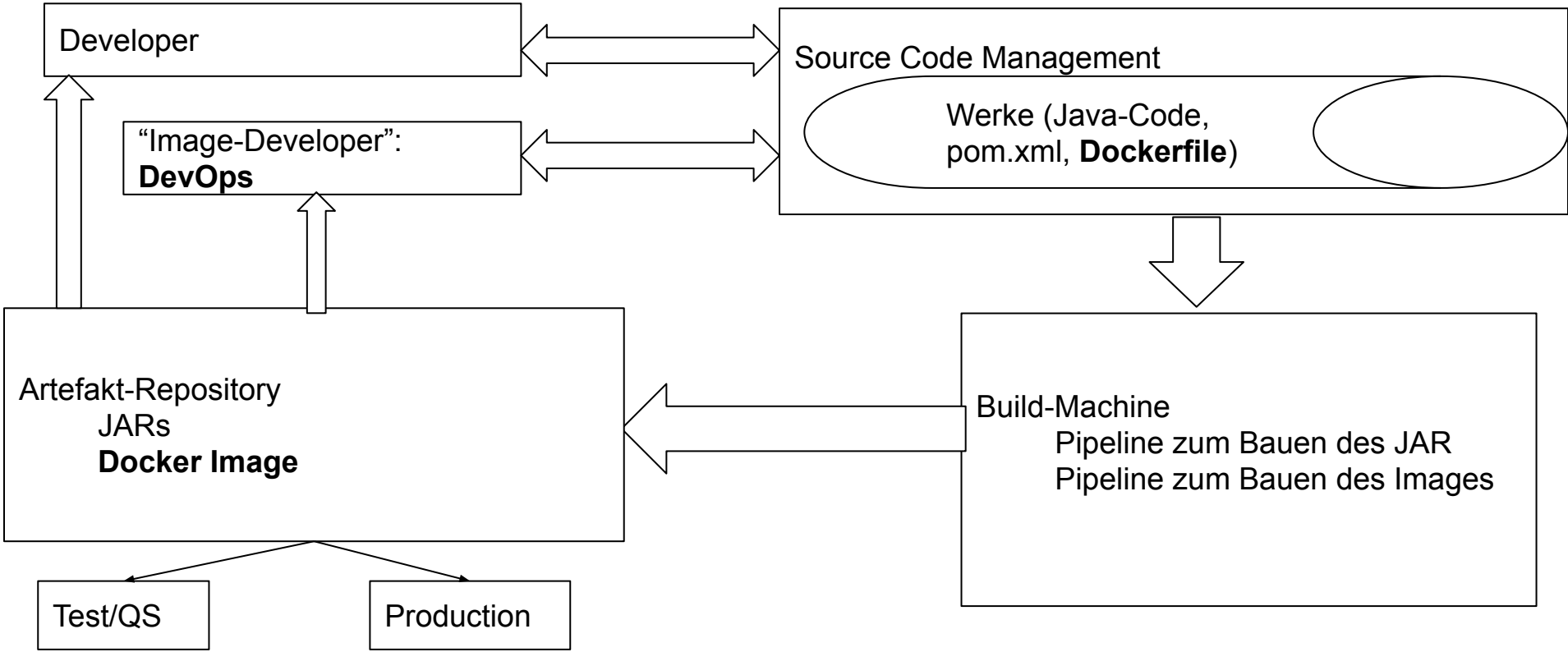
Exkurs: Docker Container

- Wird erzeugt aus einem Docker Image
- Ein Docker Image ist definiert über ein “Dockerfile”
  - Ein minimalistisches Beispiel für ein Image my_app:1.0

```
FROM openjdk:1.8
ADD application-1.0.jar .
CMD java -jar application.jar
```

- Container: docker create -p 9090:8080 my_app:1.0

Toolchain


Developer
“Image-Developer”:
DevOps
Source Code Management
Werke (Java-Code,
pom.xml, Dockerfile)
Artefakt-Repository
JARs
Docker Image
Build-Machine
Pipeline zum Bauen des JAR
Pipeline zum Bauen des Images
Test/QS
Production

Aufwände

Ausbildung

- Java Grundausbildung
  - 5 - 8 Tage
  - Hinweis: Viele klassische Programmier-Themen sind irrelevant
    - Native Ressourcen-Zugriffe auf Dateisystem und Netzwerk
    - Graphische Benutzeroberflächen in Java sind “old fashioned”
      - Swing oder JavaFX macht heute fast keiner mehr
      - Browser-basierte Applikationen sind “State of the Art”
        - React oder Angular
      - OK ist noch JavaServer Faces (JSF) oder Spring MVC, Struts ist schon sehr alt
    - Optimierung und Tuning
- Spring (oder JEE)
  - Spring Boot/Spring Core: 4 Tage
  - Spring REST: 2 Tage
  - Spring Date: 3 Tage

Ausbildung

- Build-Management mit Apache Maven
  - 3 Tage
- DevOps
  - 3 - 4 Tage

- FRAME