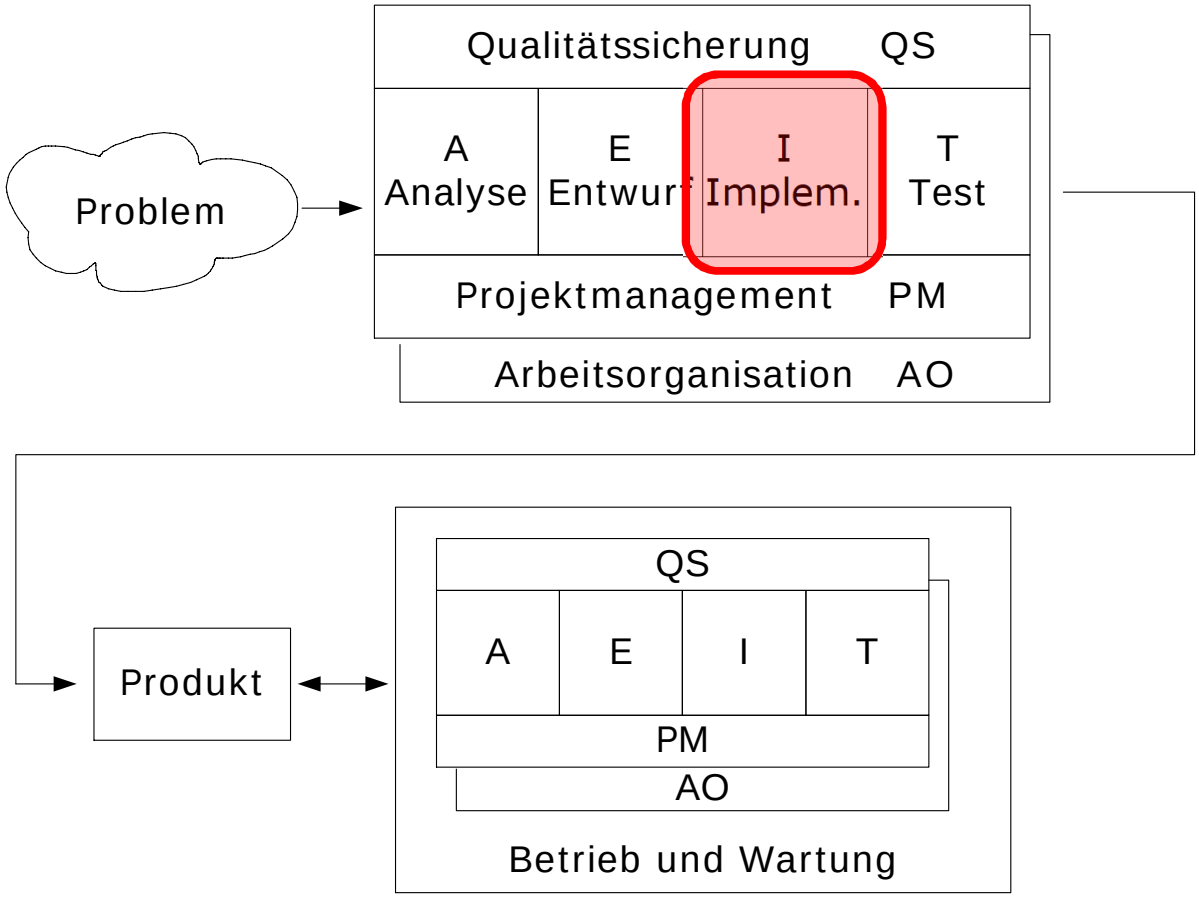
Vorlesung Softwaretechnik I (SS 2024)

9. Implementierung

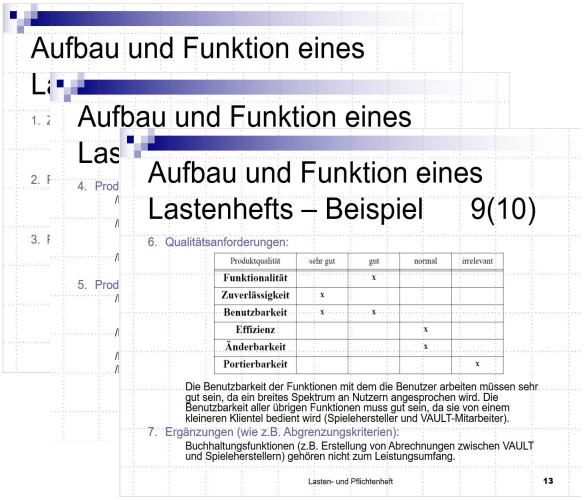
Prof. Dr. Jens Grabowski

Tel. 39 172022
grabowski@informatik.uni-goettingen.de

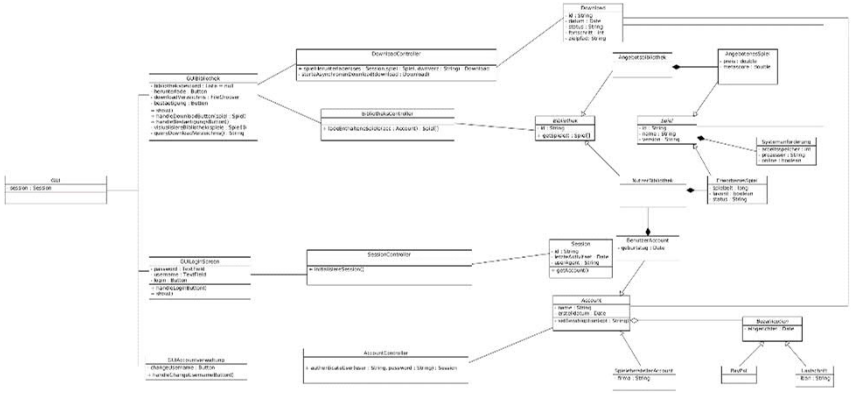
Worum geht es in diesem Kapitel?



Ausgangslage



Requirements

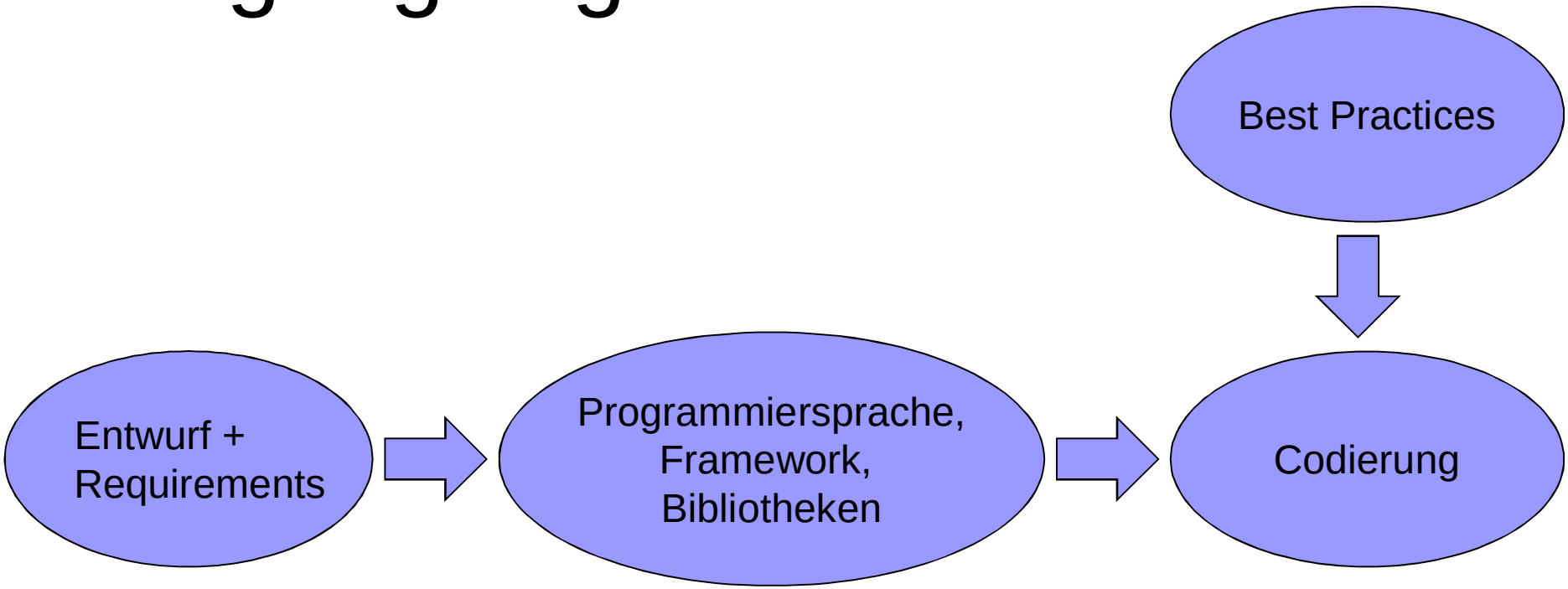


Entwurf

Inhalt

- Programmiersprachen, Frameworks und IDEs
- Grundlagen der Codierung und Best Practices
- Konfigurationsmanagement
- Anwendungsbeispiel

Ausgangslage



Implementierung

Definition: Implementierung (IEEE-Standard 610.12)

(1) The process of translating a design into hardware components, software components, or both.
(2) The result of the process in (1)

➔ Bevor Implementierung: Wahl von Programmiersprache und Technologien

Programmiersprachen

- Welche Programmiersprache für das Projekt?
  - Erfahrung der Entwickler
  - Auswahl von Bibliotheken und Frameworks
  - Systemtyp
  - Vorgabe des Kunden

Programmiersprachen

Programmiersprache	Einsatzgebiet
C++	Mobile Anwendungen, Spiele, Embedded Systems, Scientific Computing
C	Embedded Systems
Java	Enterprise-Applikationen, Webapplikationen, Mobile Applikationen
C#	Ähnlich Java
FORTRAN	Finanzapplikationen, Scientific Computing
BASIC	Client-Anwendungen
Skriptsprachen (Python, Perl, ...)	Webapplikationen, Systemadministration, Prototyping
COBOL	Legacy-Systeme

Programmiersprachen

- Erweiterungen von Sprachen
  - Annotations
  - Externe Konfiguration

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test { }
```

Technologien

- Frameworks
- Bibliotheken
- Entwicklungsumgebungen (IDEs)
- ...

Frameworks

- Was ist ein Framework?
  - Anpassbares Applikationsskelett
  - Wiederverwendbarer Entwurf
  - Verwendung ist eine effiziente Art der Softwarewiederverwendung

https://www.djangoproject.com/community/logos/

https://www.eclipse.org/jetty/

Frameworks

Gründe zur Benutzung

- Reduktion Entwicklungszeit ➔ Reduktion Entwicklungskosten
- Einhaltung von Standards
- Wiederverwendbarkeit

Framework

Auswahlkriterien

- Eigenbau vs. vorhandenes Framework
- Anforderungsabdeckung
- Sicherung von Wartung und Weiterentwicklung
- Kommerziell vs. Frei

Framework

Typen

- Persistence Frameworks
  - Abstraktion von Datenbankzugriffen
- Web-Applikations-Frameworks
  - Vereinfacht Erstellung von Web-Applikationen
- Grafik-Frameworks
  - Hilft bei der Darstellung von Strukturen
- Überkategorien:
  - Whitebox Frameworks: Innere Strukturen sind sichtbar, werden meist über Vererbung benutzt
  - Blackbox Frameworks: Innere Strukturen nicht sichtbar, werden meist über abstrakte Methoden/Annotations genutzt

Bibliotheken

Definition: Bibliothek (IEEE-Standard 610.12)

A controlled collection of software and related documentation designed to aid in software development, use, or maintenance. ...

- Bibliotheken und Frameworks sind unterschiedliche Konzepte!

Bibliotheken vs. Frameworks

Bibliotheken	Frameworks
Wiederverwendbare Funktionalität	Wiederverwendbares Verhalten
Aufruf durch eigenen Code	Eigener Code wird aufgerufen
Keine Verwendung von Frameworks	Verwendung von Bibliotheken möglich

```
public void writeTrialDataToXls(TrialData td, OutputStream os) {
    HSSFWorkbook wb = new HSSFWorkbook();
    HSSFSheet sheet = wb.createSheet("data");
    HSSFCellStyle dataStyle = wb.createCellStyle();
}

@Observer(JpaIdentityStore.EVENT_USER_AUTHENTICATED)
public void loginSuccessful(User user) {
    this.log.info("Authenticated #0", user.getUsername());
    this.user = user;
    ...
}
```

Integrated Development Environment (IDE)

Definition: IDE (IEEE-Standard 610.12)

A software development tool for creating applications, such as desktop and web applications. IDEs blend user interface design and layout tools with coding and debugging tools, which allows a developer to easily link functionality to user interface components.

- Auswahl abhängig von verwendeten Technologien und persönlichen Präferenzen

IDE

Komponenten

- Compiler
- Interpreter
- Integration Build Tools
- Syntax Highlighting
- Versionsmanagement
- Code Completion
- Refactoring Unterstützung
- Testing Unterstützung
- Debugger
- Profiler
- ...

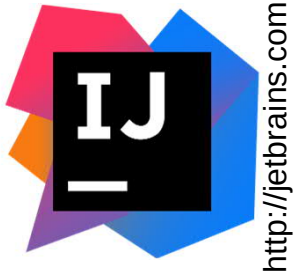
IDE

Beispiele

https://commons.wikimedia.org/

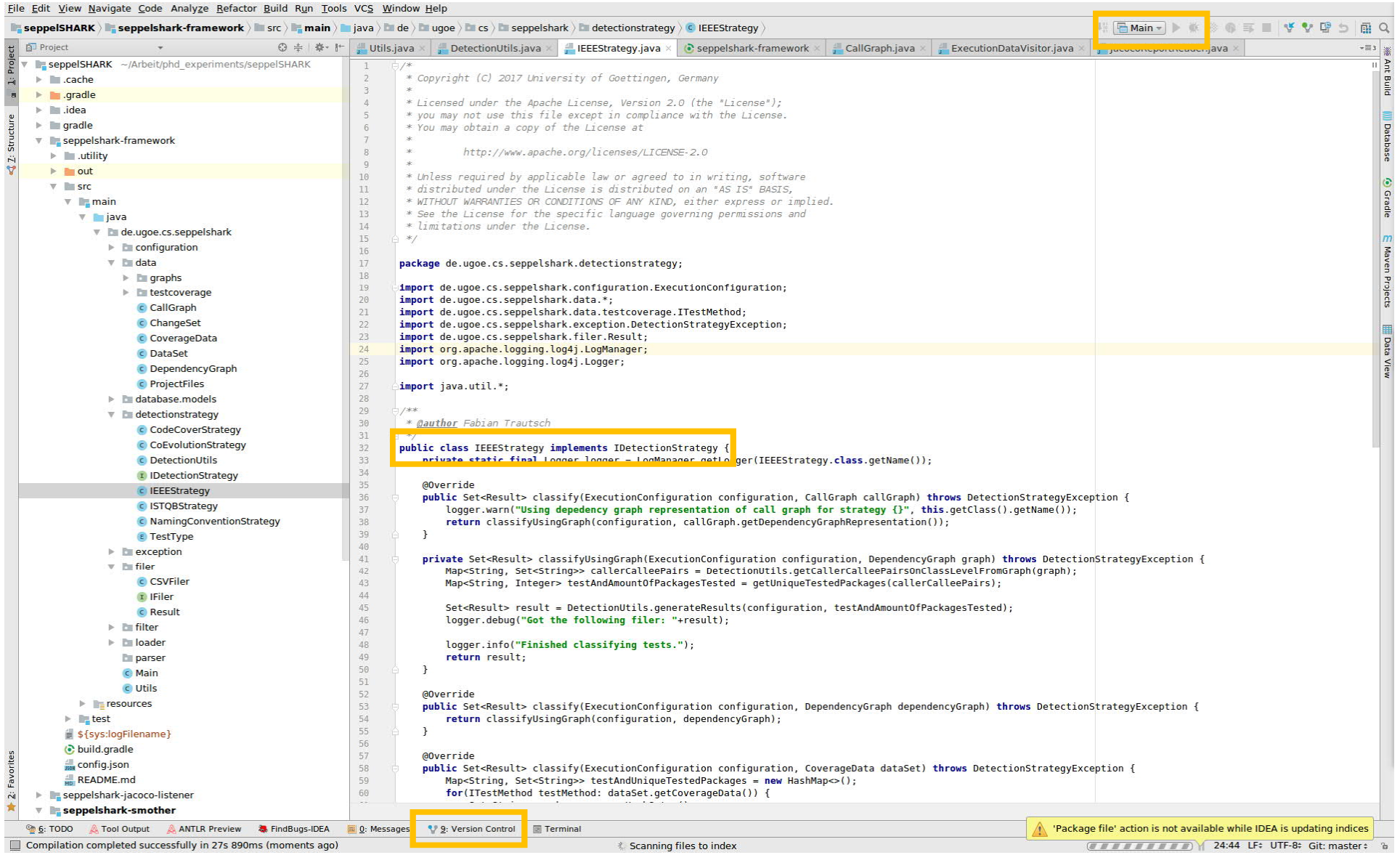
https://netbeans.org/community/teams/evangelism/collateral.html

http://www.eclipse.org/artwork/



http://jetbrains.com

http://jetbrains.com


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
seppelSHARK seppelshark-framework src main java de ugoe cs seppelshark detectionstrategy IEEEStrategy
Main
Project
seppelSHARK ~/Arbeit/phd_experiments/seppelSHARK
.cache
.gradle
.idea
gradle
seppelshark-framework
.utility
out
src
main
java
de.ugoe.cs.seppelshark
configuration
data
graphs
testcoverage
CallGraph
ChangeSet
CoverageData
DataSet
DependencyGraph
ProjectFiles
database.models
detectionstrategy
CodeCoverStrategy
CoEvolutionStrategy
DetectionUtils
IDetectionStrategy
IEEEStrategy
ISTQBStrategy
NamingConventionStrategy
TestType
exception
filer
CSVFiler
IFiler
Result
filter
loader
parser
Main
Utils
resources
test
${sys:logFilename}
build.gradle
config.json
README.md
seppelshark-jacoco-listener
seppelshark-smother
Utils.java
DetectionUtils.java
IEEEStrategy.java
seppelshark-framework
CallGraph.java
ExecutionDataVisitor.java
/*
* Copyright (C) 2017 University of Goettingen, Germany
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package de.ugoe.cs.seppelshark.detectionstrategy;
import de.ugoe.cs.seppelshark.configuration.ExecutionConfiguration;
import de.ugoe.cs.seppelshark.data.*;
import de.ugoe.cs.seppelshark.data.testcoverage.ITestMethod;
import de.ugoe.cs.seppelshark.exception.DetectionStrategyException;
import de.ugoe.cs.seppelshark.filer.Result;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import java.util.*;
/**
* @author Fabian Trautsch
*/
public class IEEEStrategy implements IDetectionStrategy {
private static final Logger logger = LogManager.getLogger(IEEEStrategy.class.getName());
@Override
public Set<Result> classify(ExecutionConfiguration configuration, CallGraph callGraph) throws DetectionStrategyException {
logger.warn("Using depedency graph representation of call graph for strategy {}", this.getClass().getName());
return classifyUsingGraph(configuration, callGraph.getDependencyGraphRepresentation());
}
private Set<Result> classifyUsingGraph(ExecutionConfiguration configuration, DependencyGraph graph) throws DetectionStrategyException {
Map<String, Set<String>> callerCalleePairs = DetectionUtils.getCallerCalleePairsOnClassLevelFromGraph(graph);
Map<String, Integer> testAndAmountOfPackagesTested = getUniqueTestedPackages(callerCalleePairs);
Set<Result> result = DetectionUtils.generateResults(configuration, testAndAmountOfPackagesTested);
logger.debug("Got the following filer: "+result);
logger.info("Finished classifying tests.");
return result;
}
@Override
public Set<Result> classify(ExecutionConfiguration configuration, DependencyGraph dependencyGraph) throws DetectionStrategyException {
return classifyUsingGraph(configuration, dependencyGraph);
}
@Override
public Set<Result> classify(ExecutionConfiguration configuration, CoverageData dataSet) throws DetectionStrategyException {
Map<String, Set<String>> testAndUniqueTestedPackages = new HashMap<>();
for(ITestMethod testMethod: dataSet.getCoverageData()) {
6: TODO
Tool Output
ANTLR Preview
FindBugs-IDEA
0: Messages
9: Version Control
Terminal
'Package file' action is not available while IDEA is updating indices
Compilation completed successfully in 27s 890ms (moments ago)
Scanning files to index
24:44 LF: UTF-8: Git: master:
Ant Build
Database
Gradle
Maven Projects
Data View
1: Project
7: Structure
2: Favorites

Inhalt

- Programmiersprachen, Frameworks und IDEs
- Grundlagen der Codierung und Best Practices
- Konfigurationsmanagement
- Anwendungsbeispiel

Grundlagen der Codierung

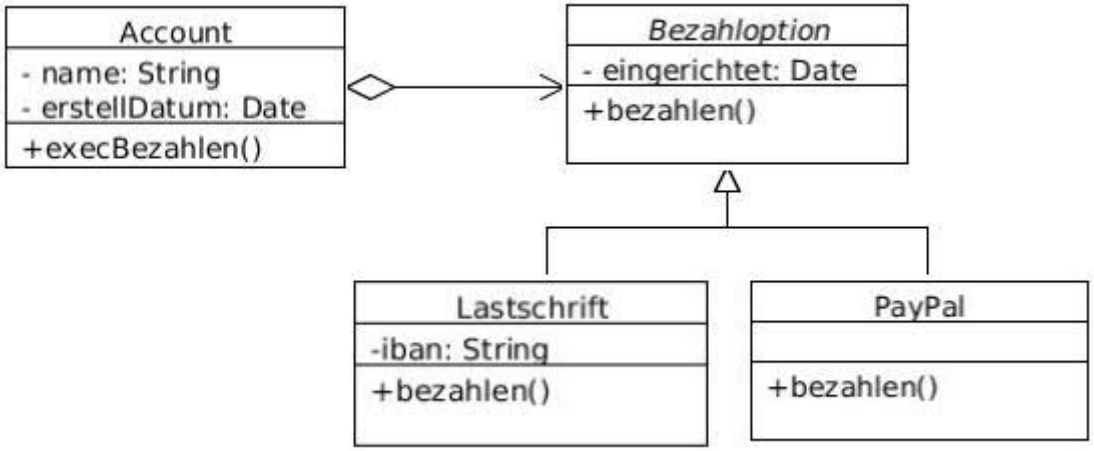
- Ziele die erreicht werden sollten
  - Minimierung der Komplexität
  - Änderbarkeit
  - Verifizierbarkeit
  - Wiederverwendbarkeit
  - Befolgen von Standards und Richtlinien

➔ Best Practices helfen diese Ziele zu erreichen

Design Patterns

- Design Pattern: Vorgefertigte Lösung für wiederkehrende Probleme
- Wichtig in Entwurf und Implementation
- Vorteile von Patterns:
  - Zeitersparnis
  - „Gute“ Lösung
- Schwierig das richtige Pattern zu finden

Strategy Pattern



Strategy Pattern

Implementation

Statt einer abstrakten Klassen (wie in UML modelliert) haben wir uns hier für ein Interface entschieden, welches auch möglich ist!

```
public class Account {
    private String name;
    private Date erstellDatum;
    private Bezahloption bezahloption;

    public Account(Bezahloption bezahloption) {
        this.bezahloption = bezahloption;
    }

    public void execBezahlung() {
        System.out.println("Bezahle mit: "+
                bezahloption.getClass().getName());
        bezahloption.bezahlen();
    }
}
```

```
public interface Bezahloption {
    public Date eingerichtet = null;
    public void bezahlen();
}
```

```
public class Lastschrift
        implements Bezahloption {
    public void bezahlen() {
        // Do Stuff
    }
}
```

```
public class PayPal
        implements Bezahloption {
    public void bezahlen() {
        // Do Stuff
    }
}
```

Strategy Pattern

Ausführung

```
public class Main {
    public static void main(String[] args) {
        Account account = new Account(new PayPal());
        account.execBezahlung();
    }
}
```

Output

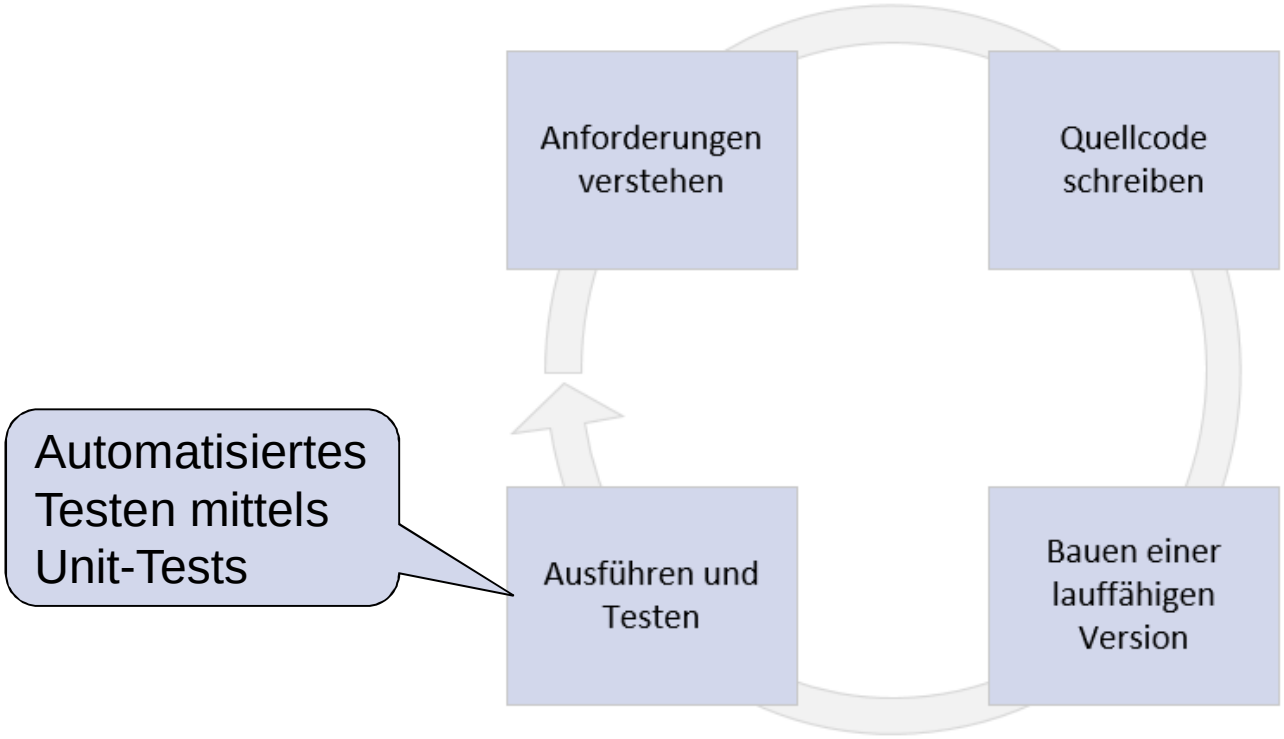
```
/usr/lib/jvm/java-8-oracle/bin/java ..
Bezahle mit: de.ugoe.PayPal

Process finished with exit code 0
```

Strukturierung von Code

- Information Hiding
  - Getter / Setter -> Private Attribute
  - Kommunikation über Interfaces
- Kopplung und Kohäsion
  - Abhängig vom Design
  - Kontrolle mittels Metriken

Automatisierte Entwicklertests



Automatisierte Entwicklertests

- Eigenschaften von Unit-/Funktionstests
  - Überprüfung einer kleinen Einheit (Unit)
  - Automatisiert & schnell wiederholbar
  - Sofortige Rückmeldung an Entwickler → Hilfe z.b. beim Refactoring
- Meist realisiert durch xUnit Testing Frameworks

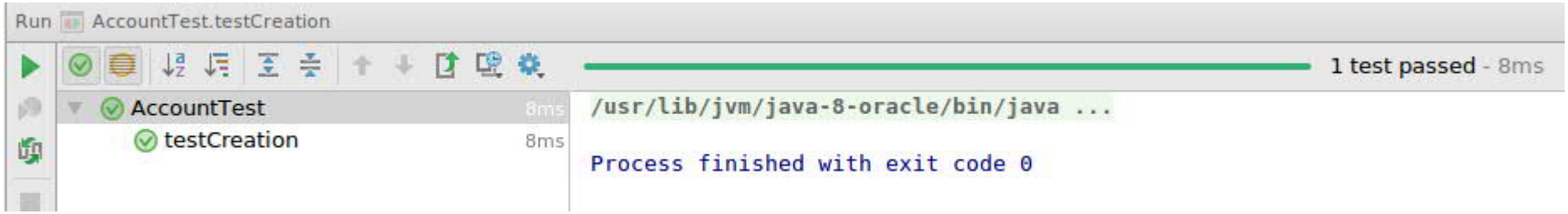
http://junit.org

Automatisierte Entwicklertests

Implementation

```
public class AccountTest {

    @Test
    public void testCreation() {
        Account acc = new Account(new PayPal());
        assertEquals( message: "Falsche Bezahloption!",
                acc.getBezahloption().getClass().getName(),
                PayPal.class.getName());
    }
}
```



Debugging

- Testen ist nicht Debuggen!
  - Testen deckt Fehler auf
- Debuggen lokalisiert Fehler
  - Z.b., Fehler die verwandt sind mit Fehlern die mittels Testen gefunden wurden

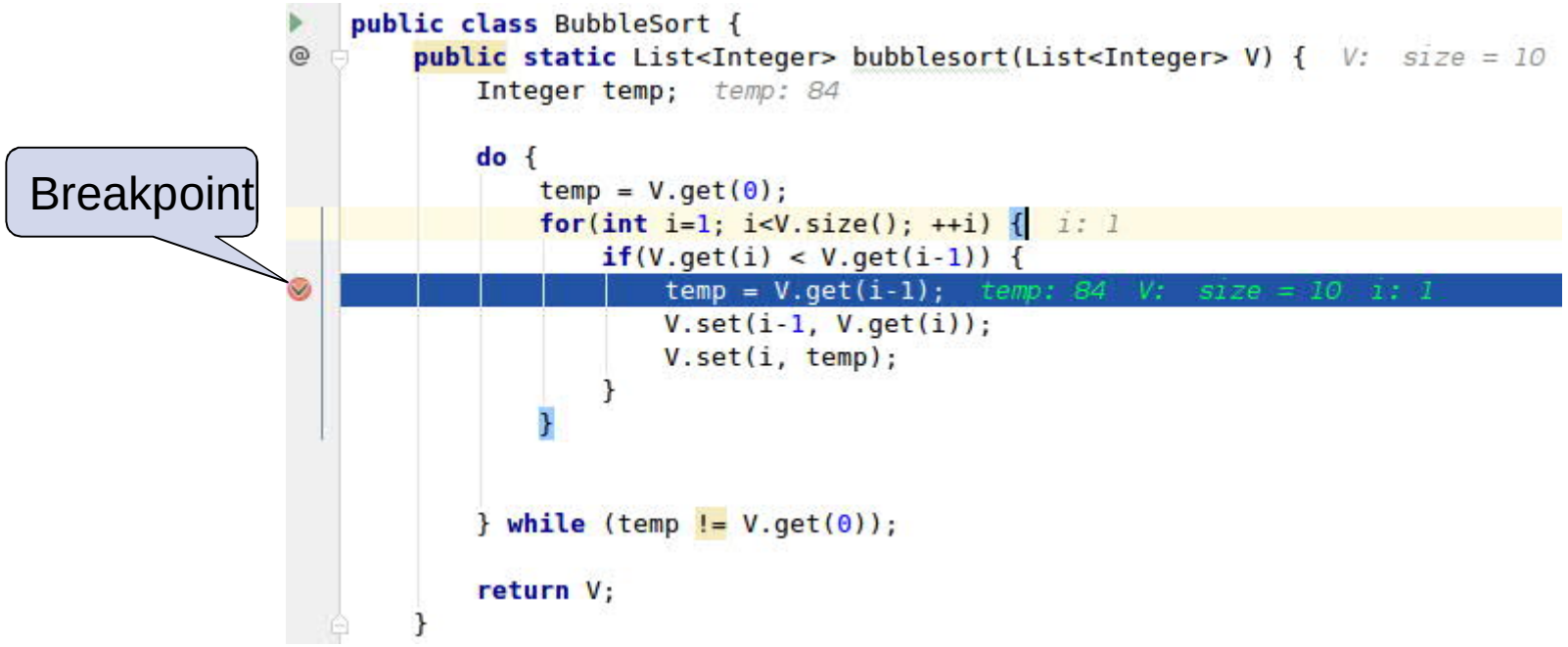
Debugging

Prozess

1. Reproduziere das Problem
   - Manchmal schwierig (z.B. Parallele Prozesse, Flaky Tests)
2. Isoliere den verantwortlichen Code
3. Simplifiziere den Testfall so dass der verantwortliche Code getestet wird
4. Benutze einen Debugger um die Quelle des Fehlers zu finden

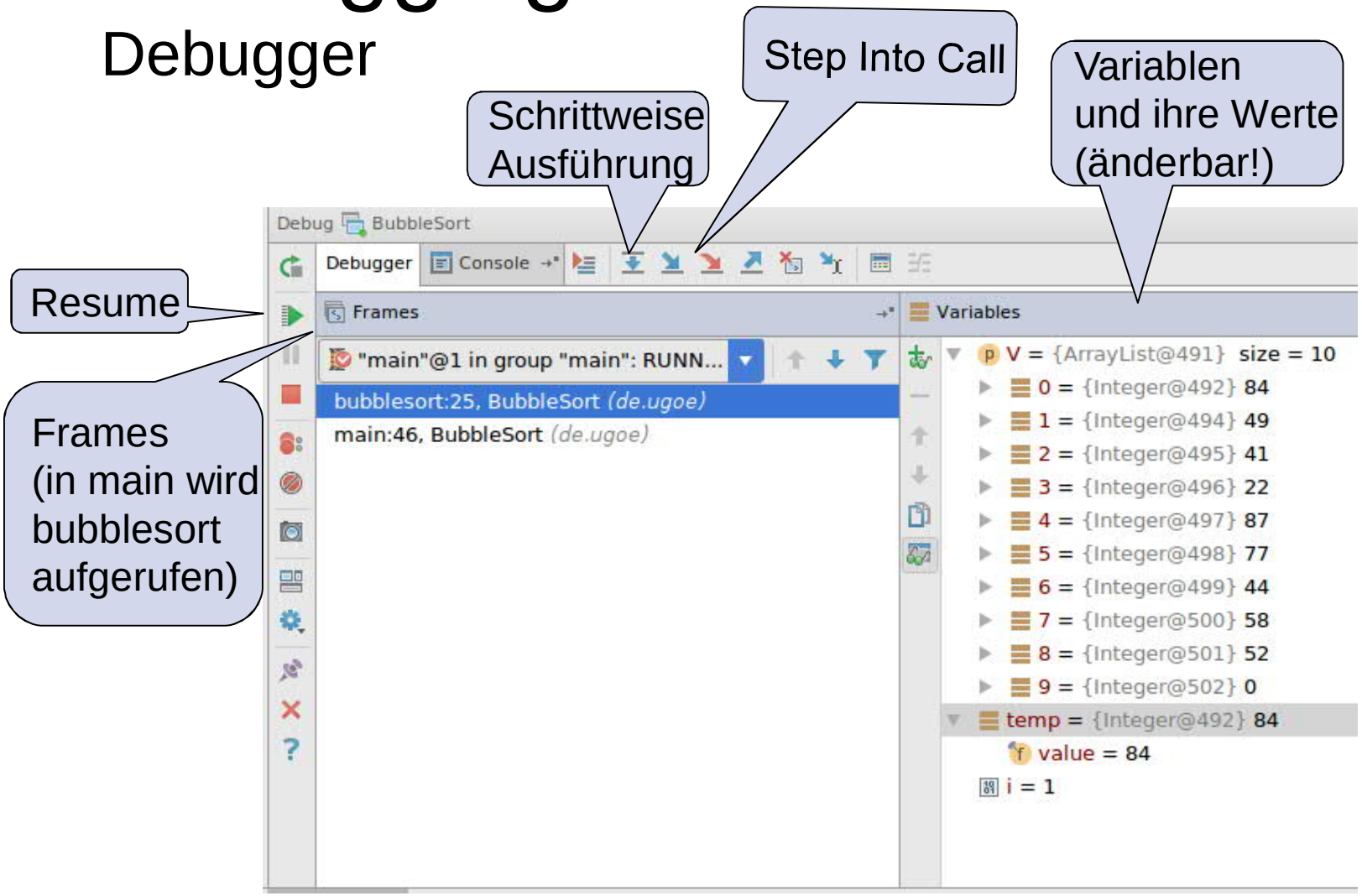
Debugging

Debugger



Debugging

Debugger



Refactoring

Definition: Refactoring (Fowler (1999))

First, the purpose of refactoring is to make the software easier to understand and modify. [...] Only changes made to make the software easier to understand are refactorings. [...] [S]econd [...] refactoring does not change the observable behavior of the software.

- Gründe für Refactorings
  - Verbesserung des Designs von Code
  - Verbesserung der Lesbarkeit von Code
  - Verstehen von fremden oder alten Code

Refactoring

Vorgehensweise

1. Identifikation von „schlecht“ strukturiertem Code
2. Sicherstellen das Bereich ausgiebig getestet ist
3. Durchführung des Refactorings
4. Ausführen aller Tests

Refactoring

Typen

- Composing Methods
  - Extract Method, Split Temporary Variable, ...
- Moving Features Between Objects
  - Move Method, Extract Class, ...
- Organizing Data
  - Replace Array with Object, Encapsulate Field, ...
- Simplifying Conditional Expressions
  - Decompose Conditional, Remove Control Flag, ...
- Making Method Calls Simpler
  - Rename Method, Add Parameter, ...
- Dealing with Generalization
  - Pull Up Field, Pull Up Method, ...

Mehr Informationen:
https://www.refactoring.com/catalog/

Refactoring

Beispiel: Split Temporary Variable

Vor dem Refactoring

```
double x = 2 * (breite + hoehe);
System.out.println("Umfang: " + x);
x = breite * hoehe;
System.out.println("Fläche: " + x);
```

Nach dem Refactoring

```
double umfang = 2 * (breite + hoehe);
System.out.println("Umfang: " + umfang);
double flaeche = breite * hoehe;
System.out.println("Fläche: " + flaeche);
```

Dokumentation von Code

- Block- und Linien-Kommentare
- Tags (z.B. TODO, NOTE, FIXME)
- Dokumentationsgeneratoren
  - Aus Kommentaren und Tags wird eine lesbare Dokumentation erstellt

https://www.konakart.com/documentation/javadoc/

http://www.import-this.de/episode/sphinx/

http://www.stack.nl/~dimitri/doxygen/index.html

Dokumentation von Code

- Wichtig: Dokumentiere **warum** etwas getan wird, nicht **was**

```
// Set the value of the age integer to 32
public int age = 32;
```

```
public Set<String> addSetEntry(Set<String> set, String value) {
    // Dont return "set.add" because it is not chainable in IE 11.
    set.add(value);
    return set;
}
```

Coderichtlinien

Definition: Coderichtlinen (Grechenig (2010))

Coderichtlinien (Coding Conventions) sind Regeln für eine Programmiersprache, die Programmierstil, Vorgehensweisen und Methoden für diese Vorschlagen.

Coderichtlinien

Gründe

- 80% der Lebenszeit einer Software ist Wartung
- Codierung und Wartung oftmals von unterschiedlichen Entwicklern
- Richtlinien erhöhen Lesbarkeit und Verständlichkeit von Code
- Code als Produkt muss sorgfältig verpackt übergeben werden

Coderichtlinien für Java (Bsp.)

- Jede Klasse hat eine eigene Datei
- Klassen- und Interfacenamen beginnen mit einem Großbuchstaben
- Konstantennamen bestehen nur aus Großbuchstaben
- Alle anderen Elementnamen beginnen mit einem Kleinbuchstaben
- Jede Zeile hat maximal 1 Statement
- Länger der Zeile < LIMIT

http://checkstyle.sourceforge.net/

Exception Management

```
try {
    BufferedReader eingabe = new BufferedReader(new
        InputStreamReader(System.in));
    String s = eingabe.readLine();
} catch (IOException e) {
    System.out.println("Fehler aufgetreten: „ +
        e.getMessage());
} finally {
    System.out.println("Programm Ende");
}
```

Exception Management

- Bereits in der Entwurfsphase muss ein Fehlerbehandlungskonzept erstellt werden
- Fehler sollen Applikation nicht zum Absturz bringen
  - Logging des Fehlers („ERROR“)
  - Fehlerbehandlung
- Konsistenter Einsatz von Fehlerbehandlungsmechanismen

Logging

- Protokollierung von Informationen zur Laufzeit
- Verwendung einer Logging-API
  - Wiederverwendbarkeit
  - Filterung der Ausgabe
  - Format der Ausgabe
  - Auswahl des Outputmediums
  - Loglevels: DEBUG, INFO, WARN, ERROR, FATAL

Logging
Beispiel

```
Set<Result> result = DetectionUtils.generateResults(
    configuration, testAndAmountOfPackagesTested);
logger.debug("Got the following filer: "+result);

logger.info("Finished classifying tests.");
```

```
2017-10-20 10:38:21,910 [main] DEBUG
de.ugoe.cs.seppelshark.detectionstrategy.IEEEStrategy - Got
the following filer: [CSVFiler]
2017-10-20 10:38:21,910 [main] INFO
de.ugoe.cs.seppelshark.detectionstrategy.IEEEStrategy -
Finished classifying tests.
```

Versionsnummern

- Ein Standard hat sich in der Praxis bewährt: *X.YY.ZZZZ*
  - *X*: Major Release
  - *YY*: Minor Release
  - *ZZZZ*: Build Nummer
- In OpenSource Entwicklung oft folgendes Schema: *A.BB.CC*
  - *A*: Major Release
  - *BB*: Minor Release
  - *CC*: Bugfix Release

Inhalt

- Programmiersprachen, Frameworks und IDEs
- Grundlagen der Codierung und Best Practices
- Konfigurationsmanagement
- Anwendungsbeispiel

Build Management Systeme

- Automatisieren von wiederkehrenden Aufgaben
  - Compilieren und Ausführen vom Programm
  - Ausführen der Tests
  - Packen des Programms zur Distribution
  - Management der Programmabhängigkeiten

Build Management Systeme

http://maven.apache.org

GNU make

http://gnu.org/software/make

http://gradle.org

http://ant.apache.org

http://bazel.build

Build Management Systeme

Beispiel (Gradle)

```
$ gradle shadowJar

:seppelshark-framework:compileJava
:seppelshark-framework:processResources
:seppelshark-framework:classes
:seppelshark-framework:shadowJar

BUILD SUCCESSFUL in 5s
3 actionable tasks: 3 executed
```

Build Management Systeme

Beispielkonfiguration (Gradle)

```
apply plugin: 'jacoco'
apply plugin: 'findbugs'
apply plugin: 'com.github.johnrengelman.shadow'


dependencies {
    testCompile group: 'junit', name: 'junit', version: '4.12'
    testCompile group: 'com.github.stefanbirkner', name: 'system-rules', version: '1.16.1'
    compile group: 'com.google.guava', name: 'guava', version: '23.2-jre'
    compile group: 'org.apache.logging.log4j', name: 'log4j-api', version: log4jVersion
    compile group: 'org.apache.logging.log4j', name: 'log4j-core', version: log4jVersion
    compile group: 'org.apache.logging.log4j', name: 'log4j-slf4j-impl', version: log4jVersion
}


javadoc {
    source = sourceSets.main.allJava
    classpath = configurations.compile
}

jar {
    manifest {
        attributes(
                'Main-Class': 'de.ugoe.cs.seppelshark.Main'
        )
    }
}
```

Versionskontrollsysteme

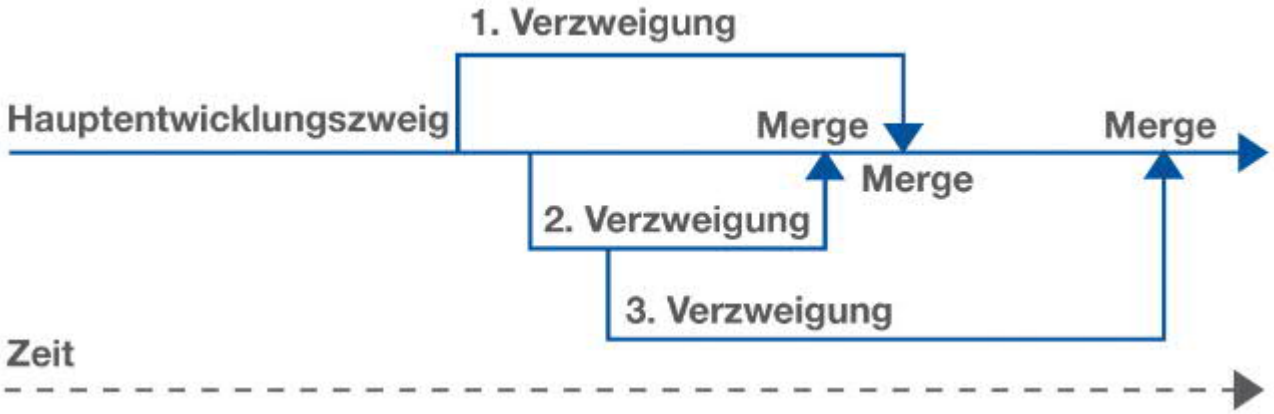
Definition: Versionskontrollsysteme (Grechenig (2010))

Versionsmanagement-Systeme sind Computersysteme, die Änderungen an Dateien und Verzeichnissen über die Zeit hinweg archivieren und in einem oder mehreren Entwicklungszweigen für die gemeinsame Bearbeitung durch mehrere Personen bereitstellen.

- Wichtige Funktionen
  - Verzweigungen (Branching)
  - Markieren (Tagging)
  - Protokollierung/Historisierung

Versionskontrollsysteme

Branching

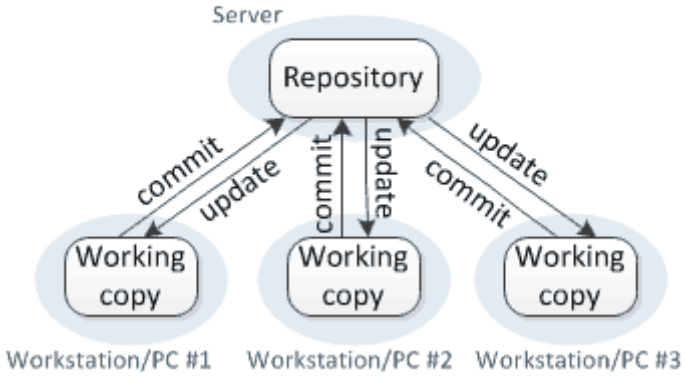


Bilder von
https://homes.cs.washington.edu/~mernst/advide/version-control.html

Versionskontrollsysteme

Typen

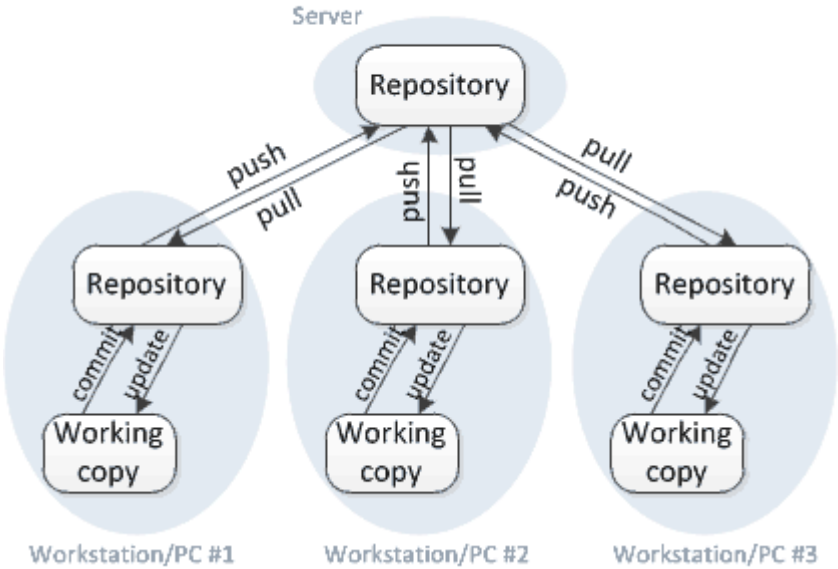
Zentrale Versionsverwaltung



SUBVERSION

http://svn.apache.org

Verteilte Versionsverwaltung



git

https://git-scm.com/downloads/logos

Zentrale Versionsverwaltung

- Client-Server System mit einem zentralem **Repository**
- Jeder Benutzer hat eine eigene **working copy** in der der Source Code lagert
- Wenn ein Benutzer **committet**, haben andere Benutzer die Möglichkeit zu **updaten** um sich die Änderungen in ihre **working copy** zu laden
- Konflikte können automatisch und manuell gelöst werden

Verteilte Versionsverwaltung

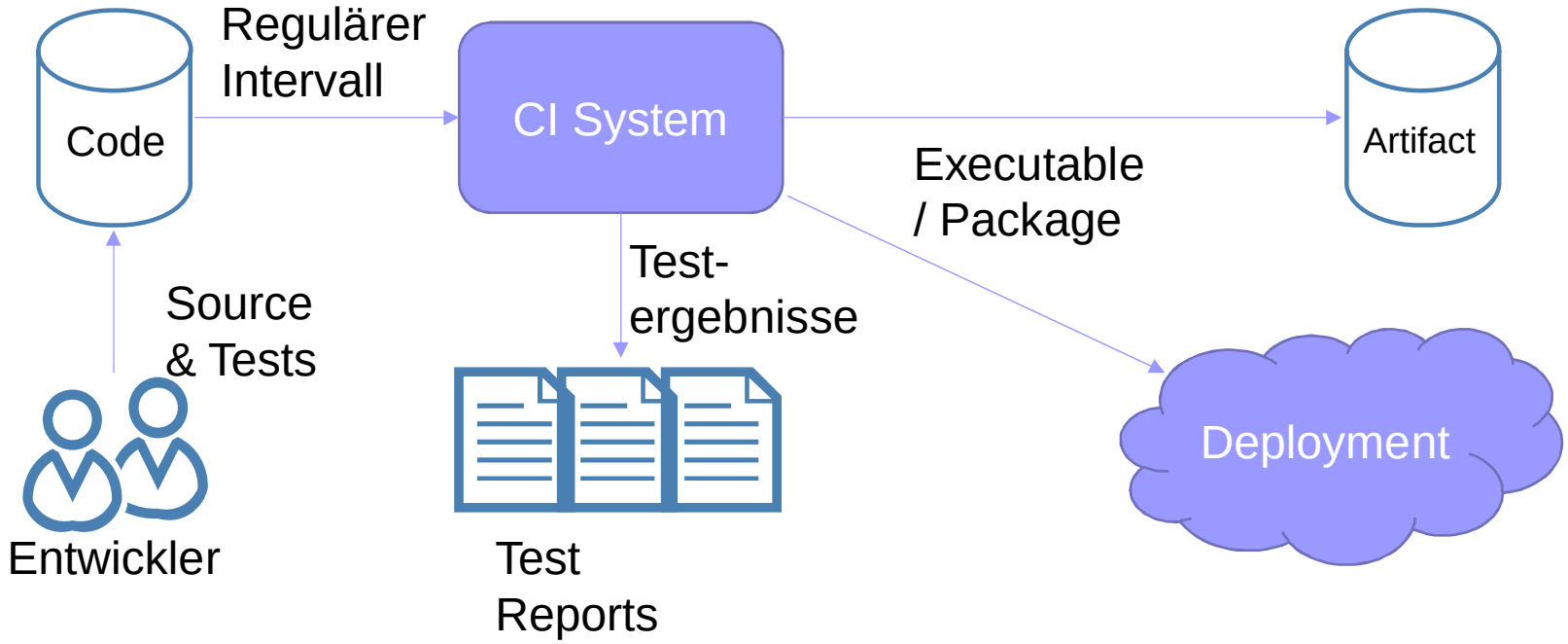
- Jeder Benutzer hat sein eigenes **Repository** und seine eigene **working copy**, welche beide lokal auf der **Workstation** liegen
- Wenn ein Benutzer **committet**, werden die Änderungen zuerst lokal von der **working copy** in das lokale **Repository** geschrieben
- Erst wenn der Benutzer **pushed**, werden die Änderungen in das **zentrale Repository** auf einem **Server** geschrieben
- Bevor ein anderer Benutzer die Änderungen sehen kann, muss dieser die Änderungen aus dem zentralen Repository **pullen**

Continous Integration

- Softwareentwicklungspraxis
- Entwickler integrieren ihre Arbeiten regelmäßig
  → Mehrere Integrationen pro Tag
- Jede Integration wird von einem Build System verifiziert
- Integrationsfehler werden somit früh erkannt

Continous Integration

Workflow



Continous Integration

Beispiel

- TravisCI: Open-Source Continous Integration System
- Enge Verbindung mit GitHub
- Konfiguration mittels *.travis.yml* in Repository
- Builds haben mehrere Jobs
  - Alle Jobs werden auf der Infrastruktur von Travis ausgeführt

Travis CI

https://travis-ci.com/logo

Inhalt

- Programmiersprachen, Frameworks und IDEs
- Grundlagen der Codierung und Best Practices
- Konfigurationsmanagement
- Anwendungsbeispiel

Anwendungsbeispiel: Aufbauen der Umgebung für die DAMPF Entwicklung

Ablauf

1. Analyse der Anforderungen für die Implementierung
2. Wahl der Programmiersprache, IDE und der Frameworks
3. Erstellung Repository auf z.B. GitHub
4. Integration eines CI Systems
5. Wahl des Build Management Systems und Einpflegen in Projekt
6. Einpflegen von Analysetools in IDE und Buildprozess
7. Anwenden eines Logging- und Exceptionhandlingkonzeptes und einpflegen einer Logging-API
8. Integration Dokumentationsgeneration
9. Tests-first Development

http://www.konakart.com/
https://logging.apache.org/log4
documentation/javadoc/

Analysetools

http://checkstyle.sourceforge.net/

http://findbugs.sourceforge.net

http://www.eclemma.org/jacoco/

Zusammenfassung

Zusammenfassung

- Programmiersprache als Werkzeug um Anforderungen umzusetzen
- Frameworks sind wichtig in der heutigen Entwicklung
- IDEs bieten viele Funktionalitäten um die Arbeit zu erleichtern
- Zur Implementierung existieren viele Best Practices: Von Design Patterns bis zum Logging
- Build Systeme bauen und testen Software automatisch
- Versionskontrollsysteme sind wichtig um eine reibungslose Entwicklung mit mehreren Entwicklern zu gewährleisten

Literatur

- Softwaretechnik: Mit Fallbeispielen aus realen Entwicklungs-projekten
- Grechenig et al. (2010)

http://amazon.de/

Lernziele

- Implementierung im Kontext des Gesamtprojektes
- Definition und Differenzierung von Frameworks und Bibliotheken
- Definition und Funktion von IDEs
- Best Practices bei der Implementierung
- Zusammenarbeit an einer Code Basis
- Dokumentation von Arbeit