

Video 1

Software-Qualität

Prof. Dr. Stephan Kleuker
Hochschule Osnabrück

- Prof. Dr. Stephan Kleuker, geboren 1967, verheiratet, 2 Kinder
- seit 1.9.09 an der FH, Professur für Software-Entwicklung
- vorher 4 Jahre FH Wiesbaden
- davor 3 Jahre an der privaten FH Nordakademie in Elmshorn
- davor 4 ½ Jahre tätig als Systemanalytiker und Systemberater in Wilhelmshaven
- s.kleuker@hs-osnabrueck.de oder Zoom

- 2h Vorlesung + 2h Praktikum = 5 CP
- Vorlesung auf folgender Seite verlinkt
<http://kleuker.iui.hs-osnabrueck.de/index.html>
- Praktikum online:
 - Anwesenheit = (Übungsblatt vorliegen + Lösungsversuche zum vorherigen Aufgabenblatt)
 - Übungsblätter mit Punkten ($\Sigma \geq 100$), 2-4 Studis
 - Praktikumsteil mit 85 oder mehr Punkten bestanden
- Prüfung: Hausarbeit (Kenntnisse aus der Vorlesung anwenden und/oder eigene Untersuchungen)
- Unterlagen zur Folgeveranstaltung liegen 6 Tage vor der Veranstaltung vor

- Vorlesung bis vorgegebenen Vorlesungsende durcharbeiten; sinnvoll eher fertig sein, um früh Fragen stellen zu können
- Folienveranstaltungen sind schnell, bremsen Sie mit der Stopp-Taste, sehen sie in Gruppen, diskutieren Sie Gesehenes, stellen sie Fragen, die noch beantwortet werden sollen
- Fragen zur Vorlesungszeit oder sonst per E-Mail
- von Studierenden wird hoher Anteil an Eigenarbeit erwartet

- Probleme sofort melden
- Wer aussteigt teilt mit warum

Ordentliche Programmierkenntnisse in Java

- Klasse
- Methode
- Klassenvariable und Klassenmethode
- Sichtbarkeit
- Vererbung
- abstrakte Klassen
- Collections (z. B. Liste, Menge)
- Polymorphie

- schön: weitere Sprachen

- Verfahren zur Qualitätssicherung (QS) kennen lernen, anwenden lernen, situationsabhängig bewerten lernen
- Prozess zur systematischen QS kennen und nutzen lernen
- Umgang und selbständige Einarbeitung in Beispielwerkzeuge
- Fehler kennenlernen und bei eigenen Entwicklungen vermeiden

Randbedingungen

- Programmierung in Java (Erkenntnisse übertragbar auf andere Sprachen)
- Open Source Werkzeuge
- keine Software-Ergonomie

Themenbereiche der Qualitätssicherung

Funktionales Testen

verhält sich mein Programm gemäß der funktionalen Anforderungen

weitergehende QS-Maßnahmen

welche weiteren Maßnahmen gibt es, wie organisiere ich den QS-Prozess

Last- und Performance Tests

wie schnell ist mein Programm, wieviele Aufrufe gleichzeitig sind möglich

Usability Tests

erfüllt das Programm die Erwartungen an das Bedien-Erlebnis

Sicherheitstests

Veranstaltungsinhalt

Themengebiete (Planung)



- 1 Fehlerquellen von Software
- 2 Unit - Tests
- 3 Äquivalenzklassentests
- 4 Überdeckungstests
- 5 Vorgehensmodelle und Testen
- 6 Mocking
- 7 Test von WebServices
- 8 Test von Software mit Nutzungsoberflächen
- 9 Metriken
- 10 Konstruktive Qualitätssicherung
- 11 Performance und Speicherauslastung
- 12 Testautomatisierung
- 13 Organisation des QS-Prozesses in IT-Projekten

Video 1

Video 2

Video 3

Video 4

Video 5

Video 6

Video 7

Video 8

Video 9

Video 10

Video 11

Video 12

Video 13

Video 14

- Stephan Kleuker, Qualitätssicherung durch Softwaretests, 2. Auflage Springer Vieweg, Wiesbaden, 2019 (freier Download über Bibliothek von HS-Rechnern)
- Peter Liggesmeyer, Software-Qualität: Testen, Analysieren und Verifizieren von Software, Spektrum Akademischer Verlag, 2. Auflage, 2009
- Stephan Kleuker, Grundkurs Software-Engineering mit UML, Kapitel 11, Springer Vieweg, 4. Auflage, 2018
- Andreas Spillner, Tilo Linz, Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester - Foundation Level nach ISTQB-Standard, dpunkt.verlag, 6. Auflage, 2019

1. Fehlerquellen von Software

- Software-Fehler und ihre dramatischen Folgen
- Typische Entwicklungsphasen von Software
- Fehler in der Anforderungsanalyse
- Fehler im Design
- Fehler bei der Implementierung
- Fehler bei der Qualitätssicherung
- Fehler im Projektumfeld

technische Grundlagen:

- reguläre Ausdrücke
- functional Interfaces

Beispiele markanter Software-Fehler (1/3)

1962	Trägerrakete Mariner 1, Selbstzerstörung nach Kursabweichung; in SW ein Komma statt eines Punktes
1971	Eole 1, Satellit zur Steuerung von Wetterballons, 71 Ballons bekommen irrtümlich Befehl zur Selbstzerstörung
1982	Größte nicht-nukleare Explosion durch Fehler in Steuerungs-SW einer russischen Gas-Pipeline
1982	Flugabwehrsystem der Fregatte Sheffield versagt; anfliegender argentinischer Flugkörper als Freund erkannt
1991	Flugabwehrsystem Patriot verfehlt Scud-Rakete; notwendiger Systemneustart nach max. 100 h vergessen
1992	Londoner Ambulance Service, neue SW führt zu Wartezeiten von bis zu drei Stunden
1994	Mondsonde Clementine, Sonde sollte einen Asteroiden ansteuern, durch einen Software-Fehler wurden die Triebwerke nach der Kurskorrektur nicht mehr abgestellt
1993	Pentium-Chip, fehlerhafte Divisionsberechnung
1995	Automatischer Gepäcktransport im Flughafen Denver funktioniert nicht; zu viele Informationen im Netz

Beispiele markanter Software-Fehler (2/3)

1996	Notwendige Selbstzerstörung der Ariane 5; falsche Steuersignale durch SW-Übernahme von Ariane 4
2001	Fehlerhafte Bestrahlung von Krebspatienten durch nicht bedachten Bedienfehler
2003	Patriot schießt befreundetes englisches Flugzeug ab
2003	Stromausfall USA, falsche Reaktion in SW auf starke Lastschwankungen
2004	Rückruf Mercedes-Transporter, Fehler in Steuerungs-SW schalten evtl. Motor ab (auch andere Fahrzeughersteller)
2005	ALG 2: keine Auszahlung bei neun-stelliger Kontonummer
2005	T-Mobile: SMS für 80 statt 19 Cent abgerechnet
2005	Russische Trägerrakete Cryosat; Signal zum Abschalten der Triebwerke der zweiten Stufe nicht weiter geleitet
2011	Intel den 8MByte-Bug bei SSDs der Serie 320 führt zu plötzlichem Kapazitätsverlust, auch nach ersten Firmware-Update
2012	Schaltsekunde zur Synchronisation der Uhren mit Erdrotation bereitet Servern Probleme, legt Webseiten und Datenbanken lahm
2013	neue Software eines Börsenhandelsunternehmens verursacht einen Schaden von 440 Millionen US-Dollar, innerhalb von 45 Min. große Menge verschiedener Aktiensorten an- und wieder verkauft

Beispiele markanter Software-Fehler (3/3)

2013	acht Zeichen - schon stürzen einige Anwendungen unter Mac OS X 10.8 ab (File:///), Schuld wohl systemweite Rechtschreibkontrolle
2013	Computerfehler im Buchungssystem von American Airlines hat tausende Passagiere stranden lassen, über Stunden keine Abfertigung
2014	Fehler der neuen Playstation 4, löscht Spielstände, zerstört Spiele
2014	Durch doppeltes „goto fail“ gravierende Sicherheitslücke bei Apple
2014	Apple iOS 8.0.2 löscht teilweise alle User-Daten in iCloud
2016	Googles selbstfahrendes Auto rammt Linienbus
2019	iPhone-Bug: Apple schaltet FaceTime-Gruppenanrufe ab
2019	Microsoft: mehrtägige Großstörung bei Office365.com
2019	Software-Panne bei der Deutschen Flugsicherung, >500 Flüge fallen aus
2019	Nach Flugzeugabstürzen Boeing Typ 737 Max: Boeing räumt weiteres Softwareproblem ein
2020	Nasa findet mehrere schwere Software-Fehler in Boeings Starliner (Raumschiff)
1999 - 2015	England: mehr als 700 Post-Filialleiter zu unrecht wegen Veruntreuung verurteilt, Fehler lag in der SW Horizon

goto fail in sslKeyExchange.c



```
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;

if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

err = sslRawVerify(ctx,
                  ctx->peerPubKey,
                  dataToSign,
                  dataToSignLen,
                  signature,
                  signatureLen);
```

Foto: SPIEGEL ONLINE

fail:

```
SSLFreeBuffer(&signedHashes);
SSLFreeBuffer(&hashCtx);
return err;
```

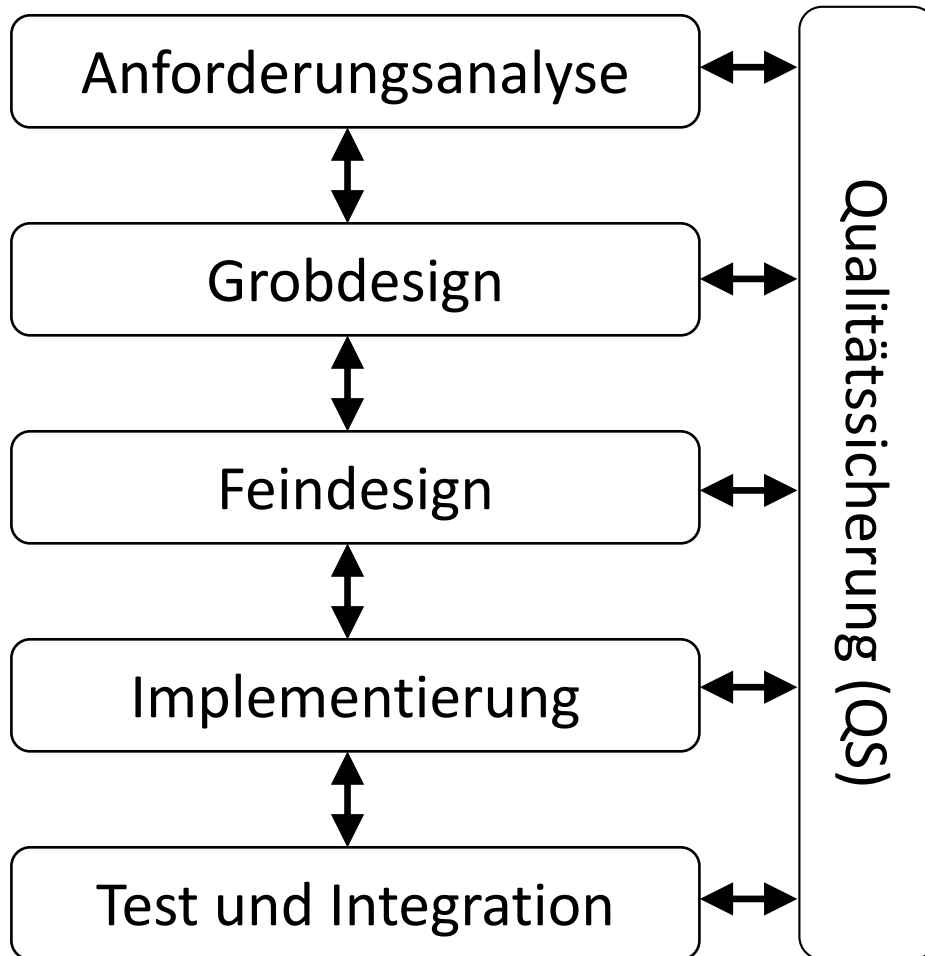
Fotostrecke

Apple-Software: Probleme bei gesicherten Verbindungen

4 Bilder

<https://www.spiegel.de/netzwelt/web/goto-fail-apples-furchtbarer-fehler-a-955154.html>

- erste Computer von Personen mit sehr hohem Bildungsniveau (Mathematik, Physik) entwickelt und programmiert
- Computer und Programme waren Individuelleistungen
- Mit steigender Verfügbarkeit von Computern stieg auch die Anzahl der SW- und HW-Fehler
- Software-Entwicklung ist hoch kreativer (künstlerischer) Prozess, der mit der Zeit in einen ingenieur-wissenschaftlichen Rahmen (Software-Engineering) eingebettet wurde
- Zentrale Ergebnisse: Vorgehensmodelle mit Prozessmodellen (wer macht wann, was, mit wem, mit welchen Hilfsmitteln, warum)



- QS ist eigenständiges Teilprojekt
- zu jedem Projekt gehört ein QS-Plan
- QS Plan: was wird wann von wem wie überprüft
- QS unabhängig vom Projekt organisiert
- Form der QS sehr stark projektabhängig
- häufig Forderung nach Normen (z.B. ISO 9000) und QS-Standards (z.B. DO 178B)

konstruktive Qualitätssicherung:

- QS vor Projekt planen
- Auswahl von Methoden, Werkzeugen, Prozessen, Schulungen
- Normen, Guidelines und Styleguides

analytische Qualitätssicherung:

- manuelle Verfahren (z.B: Inspektion und Audit) zur Prüfung der Einhaltung konstruktiver Ansätze
- Reviews von Teilprodukten (z.B. Anforderungen, Design)
- Einsatz von Testverfahren (Äquivalenzklassen, Überdeckungen) in unterschiedlichen Testphasen (Entwicklungstests, Integrationstests, Systemtests, Abnahmetests)

- Kundschaft wird zu wenig in Planungen eingebunden
- Kundschaft versteht Analysedokumente der IT nicht; nickt sie trotzdem ab
- Rahmenbedingungen (HW, umgebende SW) nicht geklärt
- Stakeholder (in irgendeiner Form Projektbetroffene) nicht in Analyse involviert (z. B. Endnutzer vergessen)
- Analyse der vom Kunden genutzten SW nicht durchgeführt; z. B. Look-and-Feel an branchenüblicher Software orientieren
- Grundregel für IT-Projekte: Garbage in - Garbage out

- keine Prüfung, ob genau die Anforderungen umgesetzt werden
- Design-Entscheidungen werden nicht dokumentiert (UML: Aktivitätsdiagramme, Klassendiagramme, Zustandsdiagramme, Sequenzdiagramme)
- mangelndes Design-Know-how macht resultierende Software langfristig unwartbar, nicht erweiterbar
- Randbedingungen z. B. der HW nicht berücksichtigt (Performance, Kommunikationsprotokolle, ...)

- Laufendes Programm – unerwünschte Funktionalität
 - Spezifikation war missverständlich
 - Überflüssige Goldrandlösung
- Mögliche Alternativen werden vergessen
- Programme zu komplex zum Testen (Schachteln oder Ketten von if, while, switch)
- Programm bei Wiederverwendung/Erweiterung unwartbar (versteckte Konstanten, versteckte Abhängigkeiten)
- Existierende Lösungen missachtet
- Kein Gedanke an Laufzeit

- alle Tests werden von den entwickelnden Personen selbst durchgeführt
- Fehlermöglichkeiten vergessen
- Testfälle können nicht nachvollzogen oder wiederholt werden; mangelnde Dokumentation der Testfälle
- Anforderungen nicht konsequent in Testfälle umgesetzt
- fehlende Werkzeuge zur Testautomatisierung; manuelle Tests zu zeitaufwändig
- Testumgebung passt nicht zur realen Zielumgebung des Kunden
- Informationen zum Performance- und Lastverhalten angeschlossener Systeme nicht berücksichtigt

- Kundschaft wird zu selten über Projektstand informiert
- Kundschaft will vor Fertigstellung der Software „nicht belästigt“ werden
- Der Umgang mit Änderungswünschen der Kundschaft und des beauftragten Unternehmens ist unterspezifiziert
- beauftragtes Unternehmen wird über projektrelevante Änderungen bei Kundschaft nicht informiert
- kein Risikomanagement (beim beauftragten Unternehmen und bei Kundschaft)

- kleines Unternehmen geht mit viel Know-how und neuer Individual-SW auf den Markt
- SW wird bei der Kundschaft direkt vor Ort angepasst
- mit Kundenzahl wächst Zahl der Anpassungen und weiterer Kundenwünsche

- dadurch, dass zentrale Daten mehrfach in Modulen gehalten werden, Datenabhängigkeiten schwer analysierbar sind und Individual-SW nicht dokumentiert ist, wird SW unwartbar

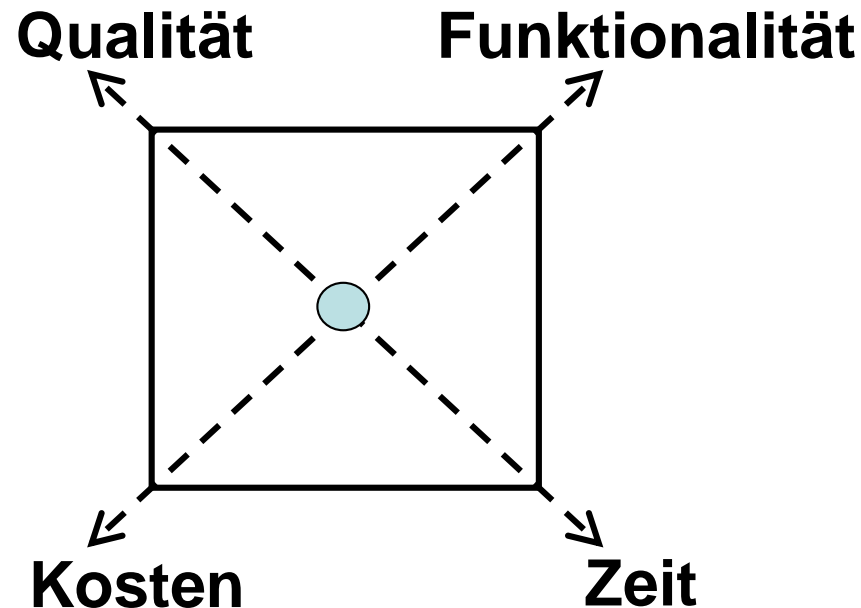
- typisches Problem vieler SW-Systeme: am Anfang erfolgreich werden sie irgendwann nicht mehr weiterentwickelbar

- mangelndes Verständnis von Anforderungen
- Übersehen von Ablaufmöglichkeiten
- Programmierfehler

- zu wenig Zeit für Tests
- mangelnde Qualifikation der Mitarbeitenden
- unpassende SW-Werkzeuge

- mangelndes Managementverständnis für IT-Entwicklung
- hoher Kostendruck, starker Preiskampf

Teufelsquadrat für IT-Projekte (Sneed)

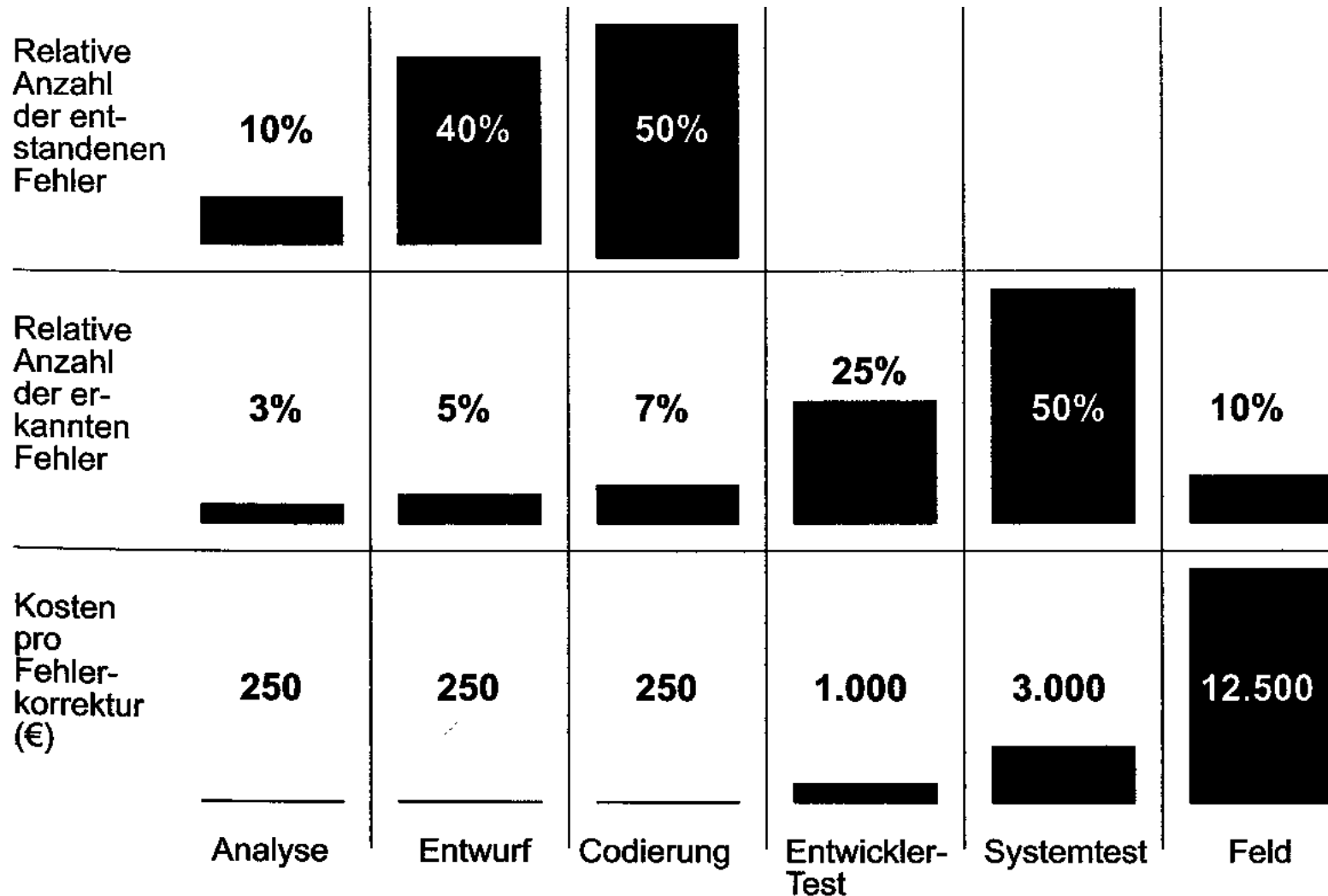


- Konzentration auf ein Ziel benötigt bei gleichen Rahmenbedingungen Vernachlässigung mindestens eines anderen Ziels
- generelle Verbesserung nur durch Verbesserung der Rahmenbedingungen (z.B. der Prozesse)

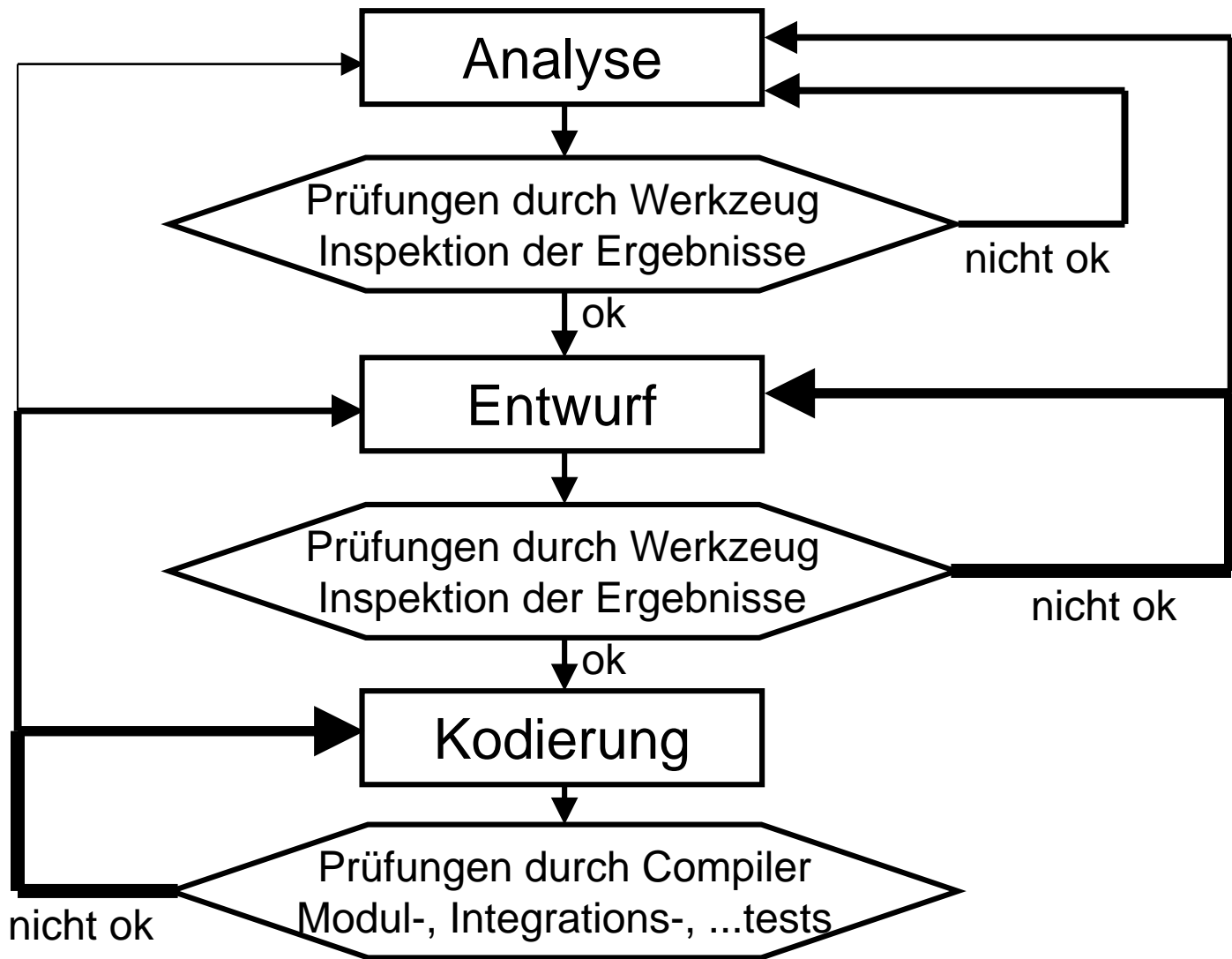
- Qualitätssicherung (allgemein)
Alle geplanten und systematischen Tätigkeiten, die notwendig sind, um ein angemessenes Vertrauen zu schaffen, dass ein Produkt oder eine Dienstleistung die gegebenen Qualitätsanforderungen erfüllt
- Qualitätssicherung (softwarespezifisch)
Die Gesamtheit aller Maßnahmen und Hilfsmittel, die mit dem Ziel eingesetzt werden, die gestellten Anforderungen an den Entwicklungs- und Wartungsprozess sowie an das Softwareprodukt zu erreichen

aus DIN 55350-11: Begriffe zu Qualitätsmanagement und Statistik, Teil 11, 8/95

Wann werden Fehler gefunden?



Prinzip der integrierten Qualitätsprüfung [Lig]



- Qualität (DIN 55350-11) definiert als „Beschaffenheit einer Einheit bezüglich ihrer Eignung, festgelegte und abgeleitete Erfordernisse (Qualitätsanforderungen) zu erfüllen“
- Qualitätsanforderung: Gesamtheit der Einzelanforderungen an eine Einheit, die die Beschaffenheit dieser Einheit betreffen
- Qualitätsmerkmal: Die konkrete Beurteilung von Qualität geschieht durch so genannte Qualitätsmerkmale. Diese stellen Eigenschaften einer Funktionseinheit dar, anhand derer ihre Qualität beschrieben und beurteilt wird, die jedoch keine Aussage über den Grad der Ausprägung enthalten. Ein Qualitätsmerkmal kann über mehrere Stufen in Teilmerkmalen verfeinert werden.

- Qualitätsmaß: konkrete Ausprägung eines Qualitätsmerkmals geschieht durch so genannte Qualitätsmaße; dies sind Maße, die Rückschlüsse auf die Ausprägung bestimmter Qualitätsmerkmale zulassen (Beispiel: Durchschnittliche Antwortzeit für Laufzeiteffizienz)
- Fehlverhalten oder Ausfall (failure) zeigt sich dynamisch bei der Benutzung eines Produkts, beim dynamischen Test einer SW erkennt man keine Fehler, sondern Fehlverhalten bzw. Ausfälle. (auch: Fehlerwirkung)
- Fehler oder Defekt (fault, defect) ist bei SW die statisch im Programmcode vorhandene Ursache eines Fehlverhaltens oder Ausfalls

funktional

- Korrektheit

nicht funktional

- Sicherheit
- Zuverlässigkeit
- Verfügbarkeit
- Robustheit
- Speicher- und Laufzeiteffizienz
- Änderbarkeit
- Portierbarkeit
- Prüfbarkeit
- Benutzbarkeit

Anmerkung: Qualitätsmerkmale können Wechselwirkungen haben

Video 2

- Reguläre Ausdrücke beschreiben eine Menge von Worten, auch Strings genannt
- Typische Fragestellung: Hat ein String den gewünschten Aufbau? Der Aufbau wird mit regulären Ausdrücken beschrieben.
- genauer: Theoretische Informatik, Aufbau regulärer Ausdruck
- benötigt: erlaubte Zeichen, Oder-Verknüpfung, Sequenz von regulären Ausdrücken, beliebige Wiederholung eines regulären Ausdrucks, leere Menge
- jede Programmiersprache bietet Bibliotheken für reguläre Ausdrücke zur Analyse und Aufteilung von Strings
- <https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html>

reguläre Ausdrücke in Java (1/5)

```
public class Main {

    public static void pruefe(String regAusdruck, String s){
        System.out.println(Pattern.matches(reAusdruck, s));
    }

    public static void main(String[] arg){
        pruefe("q","q");           // true
        pruefe("qq","q");          // false
        pruefe("q*","q");          // true    // beliebig oft
        pruefe("q*","");           // true
        pruefe("q+","");           // false  // mindestens einmal
        pruefe("q+","q");          // true
        pruefe("q|r","q");         // true    // Alternative (oder)
        pruefe("q|r","r");         // true
        pruefe("q|r","s");         // false
        pruefe("(q|r)*","qrrrq"); // true
    }
}
```

reguläre Ausdrücke in Java (2/5)

```
pruefe("..", "qr");           // true // ein bel. Zeichen
pruefe("..", "qrs");          // false
pruefe(".*", "blubb");        // true
pruefe("q?", "");             // true // ein- oder keinmal
pruefe("q?", "q");            // true
pruefe("q?", "qq");           // false
pruefe("q{3}", "qq");         // false // genau 3-mal
pruefe("q{3}", "qqq");        // true
pruefe("q{3,}", "qq");        // false // mindestens 3-mal
pruefe("q{3,}", "qqq");       // true
pruefe("q{3,5}", "qq");       // false // zwischen 3 u 5-mal
pruefe("q{3,5}", "qqqqqq");   // false
pruefe("q\\*", "qq");         // false // Fluchtsymbol
pruefe("q\\*", "q*");         // true
pruefe("q(\\*)", "q*");       // true // Klammern erlaubt
pruefe("q\\\\", "q\\");        // true // \\ immer \\
```

reguläre Ausdrücke in Java (3/5)

```
pruefe("[qwe]", "w"); // true Alternative (oder)
pruefe("[^qwe]", "w"); // false // keines der Zeichen
pruefe("[0-9][^a-f]", "1A"); // true // von - bis
pruefe("[0-9][^a-f]", "1a"); // false
pruefe("\\d+", "12345"); // true // \\d Ziffer
pruefe("\\D+", "12345"); // false // \\D keine Ziffer
pruefe("\\s*", " \n\t\f\r "); // true // \\s Weißraum
pruefe("\\S*", "hall\noo"); // false // \\S kein Weißraum
pruefe("\\w*", "i_a"); // true // \\w = [a-zA-Z_0-9]
pruefe("\\W*", " \n "); // true // nicht \\w
pruefe("\\p{Lower}\\p{Upper}", "aA"); // true // klein groß
pruefe("\\p{Lower}\\p{Upper}", "Aa"); //false // gibt mehr
// dieser Zeichenklassen
pruefe("(?i)aAa(?-i)Aa", "aaaAa"); // true // (?i) Flag Case
// insensitive
pruefe("(?i)aAa(?-i)Aa", "aaaaa"); // false // (?-i) Flag
// ausschalten
```

reguläre Ausdrücke in Java (4/5)

```
pruefe(".*", "bl\nub\nb"); // false // . kein Zeilenumbruch
pruefe("(?s).*", "bl\nub\nb"); // true // . auch für
// Steuerzeichen
pruefe("^a.*", "aaa"); // true // ^ markiert Zeilenanfang
pruefe("^a.*", " aaa"); // false
pruefe(".*a$", "aaa"); // true // $ markiert Zeilenende
pruefe(".*a$", "aaa "); // false
```

```
Pattern pat = Pattern.compile("\\d+");
Matcher mat = pat.matcher("1. Die Antwort ist 42");
while(mat.find()){
    System.out.println(mat.group() + " : "
        + mat.start() + "-" + mat.end());
}
```

1 : 0-1
42 : 19-21

reguläre Ausdrücke in Java (5/5)

```
Pattern pat2 = Pattern.compile("a+");  
Matcher mat2 = pat2.matcher("aaaa");  
while(mat2.find()) { // greedy: Suche maximaler Wortlänge  
    System.out.println(mat2.group() + " : "  
        + mat2.start() + "-" + mat2.end());  
}
```

aaaa : 0-4

```
Pattern pat3 = Pattern.compile("b+?");  
Matcher mat3 = pat3.matcher("bbbb");  
while(mat3.find()){ // mit ? nicht greedy  
    System.out.println(mat3.group() + " : "  
        + mat3.start() + "-" + mat3.end());  
}
```

b : 0-1
b : 1-2
b : 2-3
b : 3-4

```
Pattern pat4 = Pattern.compile("a|o"); //[ao]  
String[] splits = pat4.split("Hallo Costa!");  
for(String s:splits){  
    System.out.println(s);  
}
```

H
ll
C
st
!

- Ansatz: Funktionen als Parameter übergeben
- Vereinfachung für Interfaces, die genau eine Methode enthalten (auch SAM-Types für Single Abstract Method)
- selber explizit definierbar mit Annotation `@FunctionalInterface`

`(Parameterliste) -> {Ausdruck bzw. Programmanweisungen}`

- einige Notationsvarianten
- hier interessant, da in JUnit 5 genutzt
- Spezifikation: JSR 335: Lambda Expressions for the Java™ Programming Language, <https://jcp.org/en/jsr/detail?id=335>

Möglichkeiten der Interface-Nutzung (1/6)

```
package interfaces;
```

```
@FunctionalInterface  
public interface BspInterface {  
    public int mach(int x, int y);  
}
```

```
public class Plus implements BspInterface{  
  
    public int mach(int x, int y){  
        System.out.println("plus");  
        return x+y;  
    }  
}
```

Möglichkeiten der Interface-Nutzung (2/6)

```
// Beispielklasse fuer Interface-Nutzung
```

```
package nutzer;
```

```
import interfaces.BspInterface;
```

```
public class BspNutzer {  
    public int nutzen (BspInterface b1, BspInterface b2  
        , BspInterface b3){  
        return b3.mach(6, 9) - b2.mach(6, 9) - b1.mach(6, 9);  
    }  
}
```

Möglichkeiten der Interface-Nutzung (3/6)

```
// Beispielklasse fuer Interface-Nutzung
public static void main(String[] args) {
    BspInterface klassisch = new Plus();
    int erg = klassisch.mach(6, 9);
    System.out.println("Klassisch: " + erg);
```

```
plus
Klassisch: 15
```

```
BspInterface direktesObjekt = new BspInterface() {
    @Override
    public int mach(int x, int y) {
        System.out.println("minus");
        return x - y;
    }
};
erg = direktesObjekt.mach(6, 9);
System.out.println("direktes Objekt: " + erg);
```

```
minus
direktes Objekt: -3
```

Möglichkeiten der Interface-Nutzung (4/6)

```
System.out.println("anonymes Objekt: "  
    + (new BspInterface() {  
        @Override  
        public int mach(int x, int y) {  
            System.out.println("mal");  
            return x * y;  
        }  
    }).mach(6, 9));
```

```
mal  
anonymes Objekt: 54
```

```
BspInterface direktesObjektMitLambda = (a,b) -> {  
    System.out.println("oder");  
    return a | b;  
};
```

```
oder  
direktes Objekt mit Lambda: 15
```

```
erg = direktesObjektMitLambda.mach(6, 9);  
System.out.println("direktes Objekt mit Lambda: " + erg);
```

Möglichkeiten der Interface-Nutzung (5/6)

```
BspInterface direktesObjektMitLambda2 =  
    (a,b) -> a & b;  
erg = direktesObjektMitLambda2.mach(6, 9);  
System.out.println("direktes Objekt mit Lambda: " + erg);
```

direktes Objekt mit Lambda: 0

```
BspNutzer nutzer = new BspNutzer();  
System.out.println(nutzer.nutzen(  
    klassisch  
    , (a,b) -> {  
        System.out.println("minus");  
        return a - b;  
    }  
    , (a,b) -> a * b));
```

minus
plus
42

Möglichkeiten der Interface-Nutzung (6/6)

```
BspNutzer nutzer2 = new BspNutzer();
System.out.println(nutzer2.nutzen(
    (int a, int b) -> { // Typen angebar
        System.out.println("spielerei");
        return 2 * b + 4 * a;
    }
    , (a,b) -> Math.addExact(a,b)
    , Math::addExact));
}
```

```
spielerei
-42
```

```
// letzter Fall zeigt Abkuerzung bei identischen
// Parametertypen
```

Beispiel: Programmanalyse (1/3)

- Ansatz: Übergabe beliebiger und beliebig vieler Programmstücke, die ausgeführt und deren Exceptions gefangen werden
- Ansatz: Interface für parameterlose Methoden

```
@FunctionalInterface
public interface Programm {
    public void ausfuehren();
}
```

- Erinnerung: Parameterlisten mit dynamischer Länge
- `public int mach (double d, int... x){ //normal weiter`
- Aufruf mit 0 bis beliebig vielen int-Werten möglich
- x ist Variable vom Typ `int[]`

Beispiel: Programmanalyse (2/3)

```
public class Analyse {
    public static List<String> analysieren(Programm... progs) {
        List<String> ergebnisse = new ArrayList<>();
        for (Programm p : progs) {
            try {
                p.ausfuehren();
                ergebnisse.add("ok");
            } catch (Throwable e) {
                ergebnisse.add(e.getClass().getSimpleName()
                    + ": " + e.getMessage());
            }
        }
        return ergebnisse;
    }
}
```


Beispiel: Programmanalyse (3/3)

```
public static void main(String[] args) {
    System.out.println(Analyse.analysieren(
        () -> System.out.println("Hallo")
    ), () -> {
        System.out.println("durch 0");
        int x = 7 / 0;
    }
    , () -> {
        System.out.println("Array");
        int[] x = {1, 2, 3};
        System.out.println(x[3]);
    }
    , () -> {
        throw new IllegalArgumentException(
            "Kein Mensch ist illegal");
    }
    ));
```

```
Hallo
durch 0
Array
[ok,
ArithmeticException: /
by zero,
ArrayIndexOutOfBoundsException: 3,
IllegalArgumentException: Kein Mensch ist
illegal]
```

2. Unit - Tests



- Grundidee des Testens
- Annotationen
- Varianten von JUnit
- Testfälle mit JUnit
- Testsuite mit JUnit
- Parametrisierte Tests

- Hinweis: Um im Praktikum experimentieren zu können, erst „wie schreibe ich Tests“ und danach „welche Tests sind sinnvoll“
- Warnung für dieses und die weiteren Kapitel:
Der Programmcode ist meist ein Beispiel für schlechte Formatierung; hier für mich erlaubt, um möglichst viele Details auf einer Folie mit Ihnen zu diskutieren

- Nach ersten Programmierschritten und erfolgreicher Kompilierung (erste Qualitätssicherung) wird Programm ausgeführt
- (Nicht-Informatik affine Person: Programm ist fertig)
- Es wird visuell geprüft, ob erwartete Ergebnisse, gegebenenfalls bei zufällig gewählten Eingaben herauskommen
- kritische Personen lassen das Programm mehrfach laufen, überlegen sich kritische Eingaben (später formaler als Äquivalentklassenanalyse) und schauen dann Ausgaben an
- Fazit: erste Fehler können so gefunden werden, Tests typischerweise nicht dokumentiert (eventuell immerhin Testcode) und nicht einfach wiederholbar

Beispiel: eine der ersten Programmierübungen

- Übergebe 3 int-Werte und gebe den maximalen Wert zurück.

```
public class Maximum {  
    public static int max(int x, int y, int z) {  
        int max = 0; // lokale Variable muss initialisiert sein  
        if (x > z) { // zuerst x am groessten  
            max = x;  
        }  
        if (y > x) { // pruefe ob nicht y noch groesser  
            max = y;  
        }  
        if (z > y) { // oder z noch groesser  
            max = z;  
        }  
        return max;  
    }  
}
```

- Anmerkung: Code leider nicht hochwertig kommentiert

Beispiel: Hilfsprogramm für Maximum

```
public class Analyse {  
    public static void main(String[] args) {  
        System.out.println("Maximum von 7, 4, 3: erwartet: 7 "  
            + "gefunden: " + Maximum.max(7, 4, 3));  
        System.out.println("Maximum von 3, 7, 4: erwartet: 7 "  
            + "gefunden: " + Maximum.max(3, 7, 4));  
        System.out.println("Maximum von 3, 3, 7: erwartet: 7 "  
            + "gefunden: " + Maximum.max(3, 3, 7));  
    }  
}
```

```
Maximum von 7, 4, 3: erwartet: 7 gefunden: 7  
Maximum von 3, 7, 4: erwartet: 7 gefunden: 7  
Maximum von 3, 3, 7: erwartet: 7 gefunden: 7
```

- Ausgabe liefert erwartete Ergebnisse, Programm „wohl“ korrekt
- positiv: lesbare und später nutzbare Ausgaben
- positiv: prüfe Maximum an allen drei Positionen und doppelte Werte

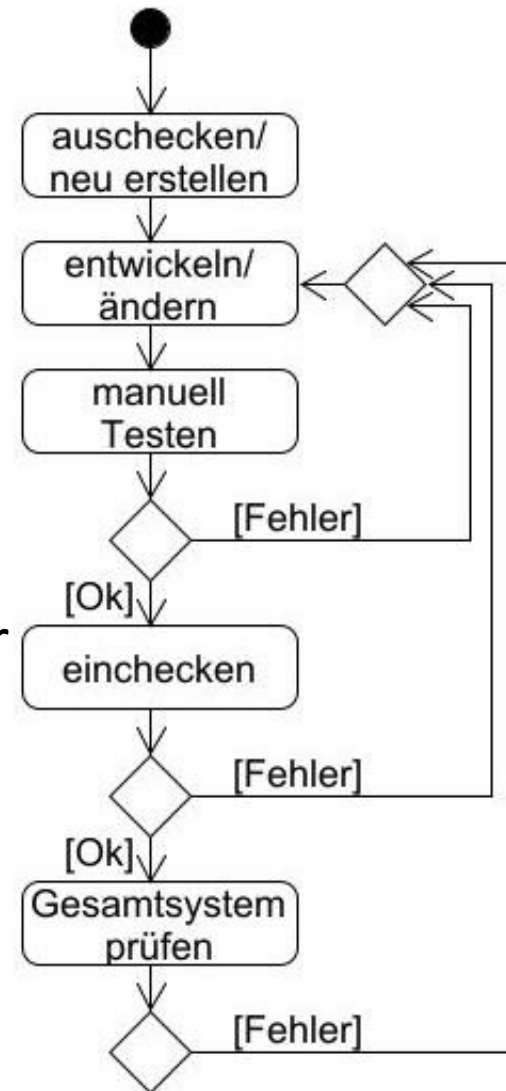
- was wird zum Testen benötigt?
- Vorbereitung: Testobjekt, hier Methode, die genutzt werden soll, sonst keine Randbedingungen
- Durchführung:
 - Testidee bzw. Testparameter zur Ausführung
 - Durchführung
 - Protokollierung
- Analyse der Ausgabe:
 - Kenntnis des erwarteten Ergebnisses
 - Prüfung des Ergebnisses

Testfall (AAA)

- Vor dem Testen müssen Testfälle spezifiziert werden
- Vorbedingungen (Arrange)
 - Zu testende Software in klar definierte Ausgangslage bringen (z. B. Objekte mit zu testenden Methoden erzeugen)
 - Angeschlossene Systeme in definierten Zustand bringen
 - Weitere Rahmenbedingungen sichern (z. B. HW)
- Durchführung /Ausführung (Act)
 - Was muss wann gemacht werden (einfachster Fall: Methodenaufruf)
- Nachbedingungen (Assert)
 - Welche Ergebnisse sollen vorliegen (einfachster Fall: Rückgabewerte)
 - Zustände anderer Objekte / angeschlossener Systeme

Einschub: Entwicklungs-Pipeline (minimalst)

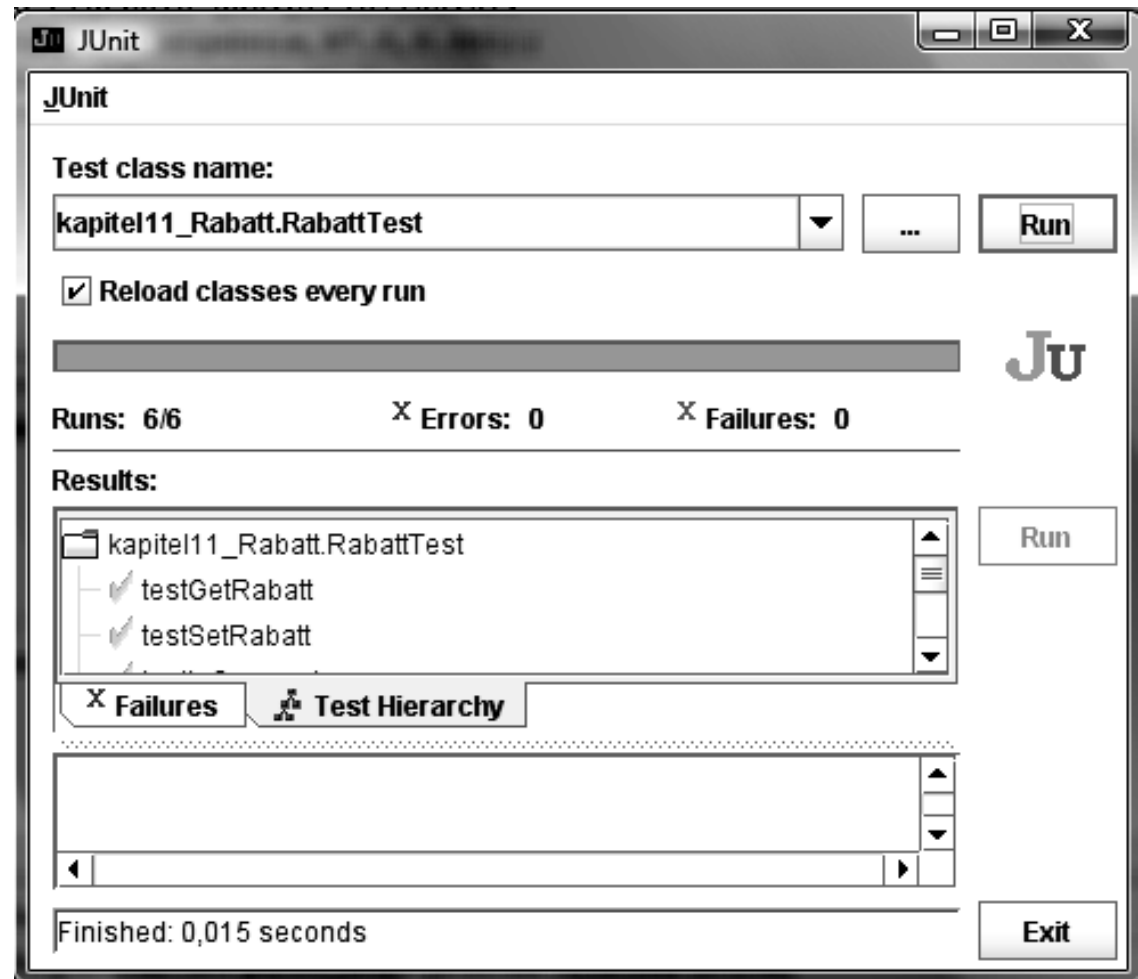
- entwickelnde Person checkt zu bearbeitende Software aus Versionsmanagement (z. B. Git) aus
- entwickelnde Person führt Änderungen aus
- entwickelnde Person testet eigenen Code, die von Korrektheit der Änderungen überzeugt
- entwickelnde Person checkt Software ein, es werden erste Überprüfungen durchgeführt (z. B. Coding-Guidelines, wie Einrückungen, zu if immer {, keine Umlaute, kein break in for)
- Test-Team überprüft Code mit eigenen Tests, ob Gesamtsystem Anforderungen erfüllt
- Fehler werden zurückgemeldet
- (Pipeline-Betrachtung später genauer)



- eine nicht getestete Software ist nicht existent (Kartenhaus)
- Testerstellung kann sehr aufwändig werden
- manuelle Testausführung kann sehr aufwändig, lästig, langweilig und selbst fehlerträchtig werden (Ansatz: Outsourcing)
- generelles Testkonzept aber einfach AAA
- Wunsch: Tests automatisiert ausführbar zu machen (und später in die Pipeline einzubauen)

- Framework, um den Unit-Test eines Java-Programms zu automatisieren
- einfacher Aufbau
- leicht erlernbar

- geht auf SUnit (Smalltalk) zurück
- mittlerweile für viele Sprachen verfügbar (JUnit, NUnit, CPPUnit)



JUnit 3.x [nicht mehr genutzt]

- entwickelt für „klassisches“ Java vor Java 5
- Testklassen müssen von einer Klasse TestCase erben
- Tests müssen in Testklassen stehen

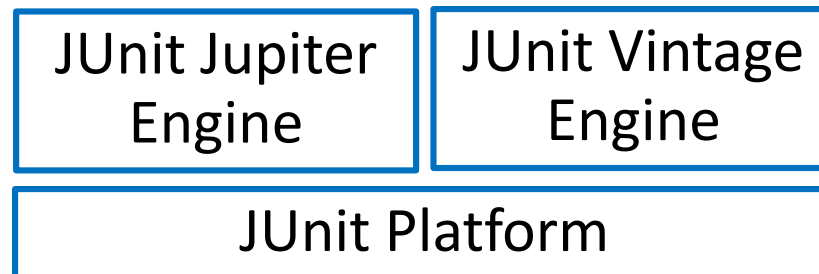
JUnit 4.x

- nutzt Annotationen
- Tests können in zu testenden Klassen stehen, keine Vererbung

JUnit 5.x

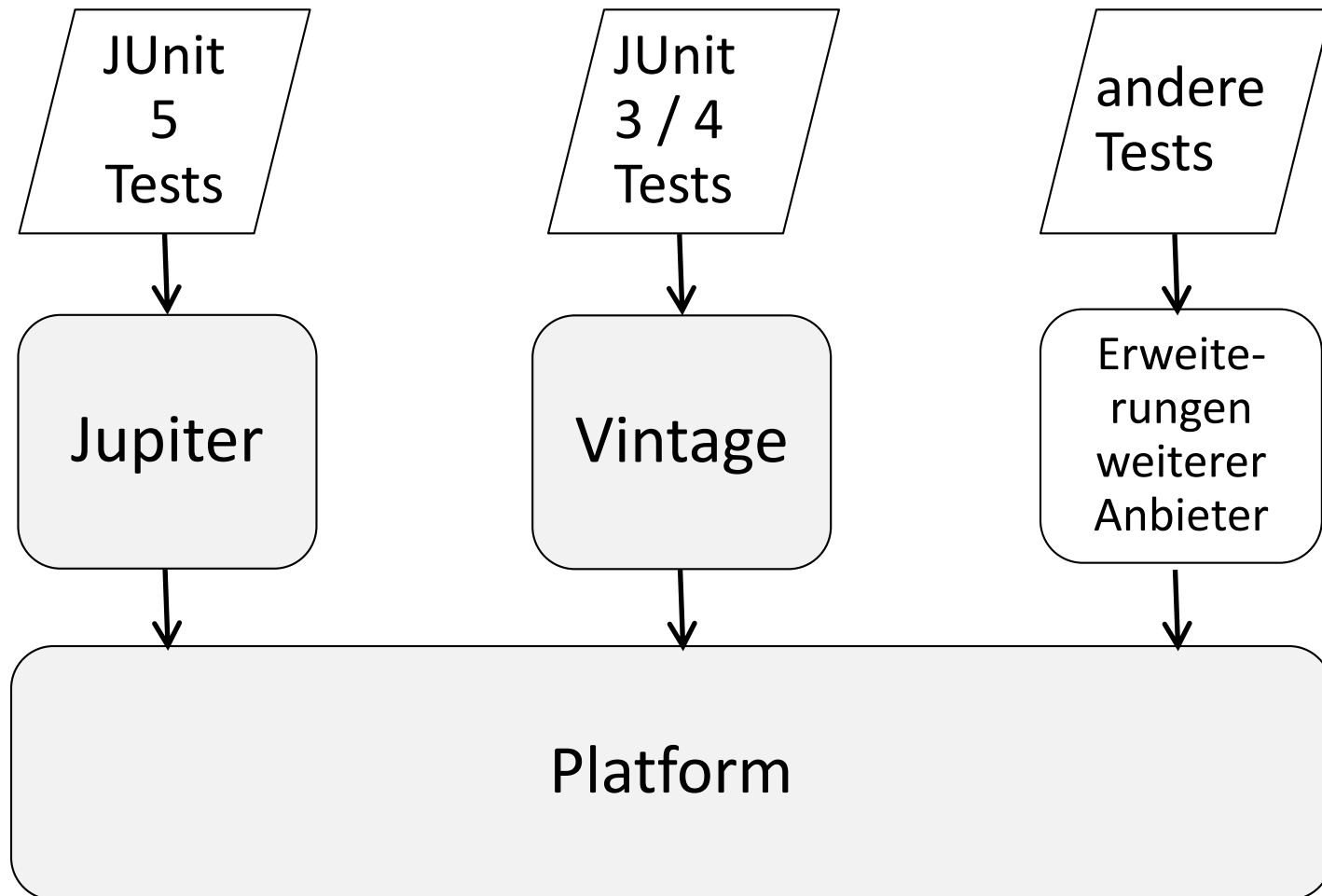
- Erweiterungen und Vereinfachungen z. B. durch Lambda-Ausdrücke
- echtes Framework mit Erweiterungsmöglichkeiten
- (viele kleine ärgerliche Änderungen wodurch JUnit 4 Tests nicht laufen), JUnit 4 kann parallel genutzt werden

- Modularer Aufbau aus Teilprojekten für erweiterbare Funktionalität
- JUnit Platform: Ausführung von Tests (TestEngines), Ausführung in Konsole, Plugins für Build-Werkzeuge (Maven, Gradle)
- JUnit Jupiter: Aufbau der Testlogik, Modell zur Erweiterung, Parametrisierung
- JUnit Vintage: Möglichkeit zur Ausführung von JUnit 3 und 4 Tests (wenn diese keinen eigenen TestRunner benötigen!)



- <https://junit.org/junit5/docs/current/user-guide/>
- B. Garcia, Mastering Software Testing with JUnit 5, Packt Publishing, Birmingham (UK), 2017

JUnit 5 (grau) – Architektur-Idee



```
// HelloWorldService.java
import javax.jws.WebMethod;
import javax.jws.WebService;
@WebService
public class HelloWorldService {
    @WebMethod
    public String helloWorld() {
        return "Hello World!";
    }
}
```

- Annotationen beginnen mit einem @ und können, z. B. von anderen Programmen zur Weiterverarbeitung genutzt werden
- hier z. B. soll die Erzeugung eines Web-Services durch die Kennzeichnung relevanter Teile durch Annotationen erfolgen
- Java besitzt seit Version 5 Annotationen, man kann selbst weitere definieren

- Annotationen können Parameter haben
- ohne Parameter: `@EinfacheAnnotation`
- mit einem Parameter `@EinAnno(par="Hallo")` oder `@EinAnno("Hallo")` (bei nur einem Parameter kann der Parametername weggelassen werden)
- mit mehreren Parametern werden Wertepaare (Parametername, Wert) angegeben
`@KomplexAnno(par1="Hi", par2=42, par3={41,43})`
- Annotationen können bei/vor anderen Modifiern (z.B. public, abstract) bei folgenden Elementen stehen:
 - package
 - class, interface, enum
 - Methode
 - Exemplar- und Klassenvariable
 - lokale Variablen
 - Parameter
- Man kann einschränken, wo welche Annotation erlaubt ist

Beispiel: Nutzung vordefinierter Annotation

```
public class Oben {  
    public void supertolleSpezialmethode(){  
        System.out.println("Ich bin oben");  
    }  
}
```

```
public class Unten extends Oben{  
    @Override public void superTolleSpezialmethode(){  
        System.out.println("Ich bin unten");  
    }  
    public static void main(String[] s){  
        Unten u = new Unten();  
        u.supertolleSpezialmethode();  
    }  
}
```

ohne Annotation in Java 1.4:
Ich bin oben

Compilermeldung:

```
..\..\netbeans\Annotationen\src\Unten.java:2: method  
does not override a method from its superclass  
    @Override public void superTolleSpezialmethode(){  
1 error
```


- Testfälle werden in Java programmiert, keine spezielle Skriptsprache notwendig
- Idee ist inkrementeller Aufbau der Testfälle parallel zur Entwicklung
 - Pro Klasse wird mindestens eine Test-Klasse implementiert (oder in Klasse ergänzt)
 - Pro Methode mindestens ein Test (später genauer)
- JUnit in allen führenden Entwicklungsumgebungen integriert
- sonst muss JUnit zu CLASSPATH hinzugefügt werden
- seit JUnit 4.11 wird neben JUnit auch Hamcrest-Matcher-Bibliothek benötigt (vorher fest integriert)
- (in JUnit 5 bereits integriert), andere Matcher-Bibliotheken nutzbar

Video 3

- Gegeben sei eine Aufzählung mit den folgenden Werten

```
package verwaltung.mitarbeit;
public enum Fachgebiet {
    ANALYSE, DESIGN, JAVA, C, TEST
}
```
- Zu entwickeln ist eine Klasse `Mitarbeit`, wobei jedes `Mitarbeitobjekt`
 - eine eindeutige Kennzeichnung (`id`) hat
 - einen änderbaren Vornamen haben kann
 - einen änderbaren Nachnamen mit mindestens zwei Zeichen hat
 - eine Informationssammlung mit maximal drei Fachgebieten hat, die ergänzt und gelöscht werden können

Klasse Mitarbeit (1/4) - fast korrekt



```
package verwaltung.mitarbeit;
import java.util.HashSet;
import java.util.Set;
public class Mitarbeit {
    private int id;
    private static int idGenerator = 100;
    private String vorname;
    private String nachname;
    private Set<Fachgebiet> fachgebiete;

    public Mitarbeit(String vorname, String nachname) {
        if (nachname == null || nachname.length() < 2)
            throw new IllegalArgumentException(
                "Nachname mit mindestens zwei Zeichen");
        this.vorname = vorname;
        this.nachname = nachname;
        this.id = idGenerator++;
        this.fachgebiete = new HashSet<>();
    }
}
```

Klasse Mitarbeit (2/4)



```
public int getId() { return id; }
public void setId(int id) { this.id = id; }
public String getVorname() { return vorname; }
public void setVorname(String vorname) {
    this.vorname = vorname; }
public String getNachname() { return nachname; }
public void setNachname(String nachname) {
    this.nachname = nachname;}
public Set<Fachgebiet> getFachgebiete() {
    return fachgebiete;}
public void setFachgebiete(Set<Fachgebiet> fachgebiete) {
    this.fachgebiete = fachgebiete;
}
```

```
public void addFachgebiet(Fachgebiet f){
    this.fachgebiete.add(f);
    if(this.fachgebiete.size() > 3){
        this.fachgebiete.remove(f);
        throw new IllegalArgumentException(
            "Maximal 3 Fachgebiete");
    }
}
```

```
public void removeFachgebiet(Fachgebiet f){
    this.fachgebiete.remove(f);
}
```

```
public boolean hatFachgebiet(Fachgebiet f){
    return this.fachgebiete.contains(f);
}
```

Klasse Mitarbeit (4/4)



```
@Override
public int hashCode() { return id; }

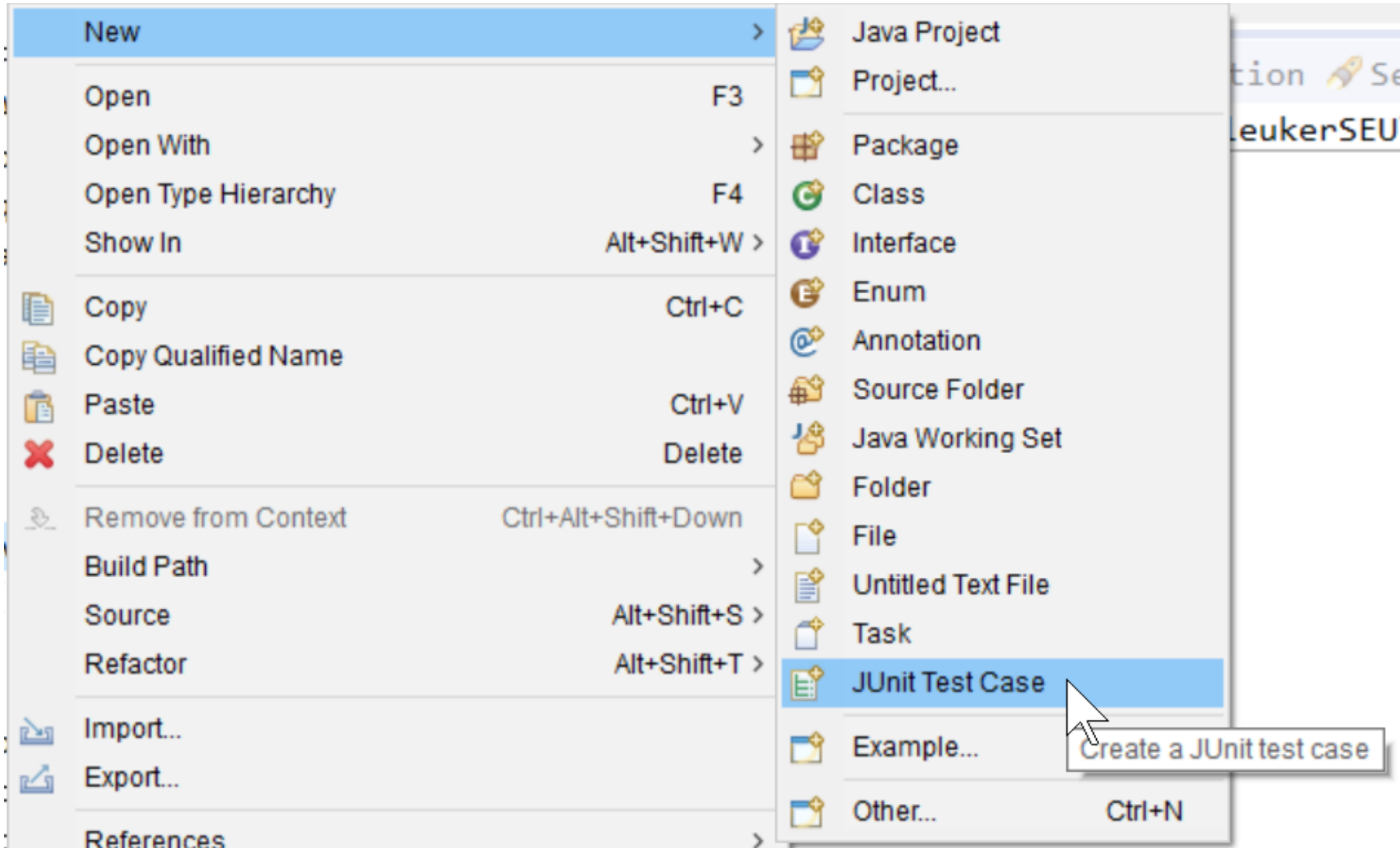
@Override
public boolean equals(Object obj) {
    if (obj == null || getClass() != obj.getClass())
        return false;
    Mitarbeit other = (Mitarbeit) obj;
    return (id == other.id);
}

@Override
public String toString(){
    StringBuilder erg = new StringBuilder(this.vorname +
        " " + this.nachname + " (" + this.id + ")[ ");
    for(Fachgebiet f: this.fachgebiete)
        erg.append(f + " ");
    erg.append("]");
    return erg.toString();
} }
```

```
public static void main(String... s){  
    Mitarbeit m = new Mitarbeit("Uwe", "Mey");  
    m.addFachgebiet(Fachgebiet.ANALYSE);  
    m.addFachgebiet(Fachgebiet.C);  
    m.addFachgebiet(Fachgebiet.JAVA);  
    System.out.println(m);  
    m.addFachgebiet(Fachgebiet.TEST);  
}
```

```
Uwe Mey (100)[ C JAVA ANALYSE ]  
Exception in thread "main"  
java.lang.IllegalArgumentException: Maximal 3  
Fachgebiete  
    at  
verwaltung.mitarbeit.Mitarbeit.addFachgebiet(Mitarbe  
it.java:58) at  
verwaltung.mitarbeit.Mitarbeit.main(Mitarbeit.java:9  
9)
```

Anlegen einer Testklasse (1/2)



Anlegen einer Testklasse (2/2)

New JUnit Test Case

JUnit Test Case

The use of the default package is discouraged.

New JUnit 3 test New JUnit 4 test New JUnit Jupiter test

Source folder: Browse..

Package: (default) Browse..

Name:

Superclass: Browse..

Which method stubs would you like to create?

setUpBeforeClass() tearDownAfterClass()

setUp() tearDown()

constructor

Do you want to add comments? (Configure templates and styles)

Generate comments

Class to test: Browse..

< Back Next > Finish Cancel

JUnit5

entweder gleiches Paket
wie zu testende Klasse
oder „test.“ davor

erzeugt Beispiel-Code;
auch weglassbar

- Tests werden mit Methoden durchgeführt, die mit Annotation `@Test` markiert sind
- Testmethoden haben typischerweise keinen Rückgabewert (nicht verboten)
- Tests stehen typischerweise in eigener Klasse; für Klasse X eine Testklasse XTest (können auch in zu testender Klasse stehen)
- Mit Klassenmethoden der Klasse Assertions werden gewünschte Eigenschaften geprüft

```
Assertions.assertTrue(m.getVorname().equals("Ute"),  
    "erwarteter Vorname Ute");
```

- auch generell nutzbar: `Assertions.assertEquals(erwartet, gefunden)`, Fehlermeldung optional, guter Fehlerausgabe
- JUnit steuert Testausführung, Verstöße bei Prüfungen werden protokolliert

Test des Konstruktors und der eindeutigen Id

```
package verwaltung;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import verwaltung.mitarbeit.Mitarbeit;

public class Mitarbeit1Test {
    @Test
    public void testKonstruktor() {
        Mitarbeit m = new Mitarbeit("Ute", "Mai");
        Assertions.assertTrue(m.getVorname().equals("Ute")
            , "korrekter Vorname waere Ute");
        Assertions.assertTrue(m.getNachname().equals("Mai")
            , "korrekter Nachname waere Mai");
    }
    @Test
    public void testEindeutigeId() {
        Assertions.assertTrue(new Mitarbeit("Ute", "Mai").getId()
            != new Mitarbeit("Ute", "Mai").getId()
            , "unterschiedliche ID gefordert");
    }
}
```

- keine gute Idee: Assertions – Code – Assertions (scheitert das Erste, wird zweites nicht betrachtet -> Tests trennen)
- Assertions-Methoden haben optionalen zweiten String-Parameter
- String kann genauere Informationen über erwartete Werte enthalten, z. B. über toString-Methoden der beteiligten Objekte kritische Details ausgeben
- generell reicht `Assertions.assertTrue()` aus, gibt viele weitere darauf basierende sinnvolle Methoden (auch eigene Bibliotheken)
- Um nicht immer die Klasse Assertions angeben zu müssen, kann man auch (persönlich unschön) folgendes nutzen:

```
import static org.junit.jupiter.api.Assertions.*;
```

- nutzt Lambda-Ausdrücke

```
@Test // bessere Variante, alle Assertions ausgeführt
public void testKonstruktor2() {
    Mitarbeit m = new Mitarbeit("Ute", "Mai");
    Assertions.assertThat("Ueberschrift optional"
        , () -> Assertions.assertTrue(m.getVorname().equals("Ute")
            , "korrekter Vorname")
        , () -> Assertions.assertTrue(m.getNachname().equals("Mai")
            , "korrekter Nachname")
    );
}
```

Ausschnitt: Klassenmethoden von Assertions

[assertArrayEquals](#)(java.lang.Object[] expecteds,
java.lang.Object[] actuals)
Asserts that two object arrays are equal.

[assertEquals](#)(double expected, double actual, double delta)
Asserts that two doubles or floats are equal to within
a positive delta.

[assertFalse](#)(boolean condition)

[assertNotNull](#)(java.lang.Object object)

[assertNull](#)(java.lang.Object object)

[assertSame](#)(java.lang.Object expected,
java.lang.Object actual)
Asserts that two objects refer to the same object.

[assertThat](#)(T actual, org.hamcrest.Matcher<T> matcher)
Asserts that actual satisfies the condition
specified by matcher.

[assertTrue](#)(boolean condition)

[fail](#)() Fails a test with no message.

- Testfall sieht in der Regel so aus, dass eine bestimmte Konfiguration von Objekten aufgebaut wird, gegen die der Test läuft
- Menge von Testobjekten wird als Test-Fixture bezeichnet
- Damit fehlerhafte Testfälle nicht andere Testfälle beeinflussen können, wird die Test-Fixture für jeden Testfall neu initialisiert
- In der mit **@BeforeEach** annotierten Methode werden Exemplarvariablen initialisiert
- In der mit **@AfterEach** annotierten Methode werden wertvolle Testressourcen wie zum Beispiel Datenbank- oder Netzwerkverbindungen wieder freigegeben

Tests von hatFähigkeit



```
public class Mitarbeit2Test {
    private Mitarbeit m1;
    @BeforeEach // JUnit4 @Before
    public void setUp() throws Exception {
        this.m1 = new Mitarbeit("Uwe", "Mey");
        this.m1.addFachgebiet(Fachgebiet.ANALYSE);
        this.m1.addFachgebiet(Fachgebiet.C);
        this.m1.addFachgebiet(Fachgebiet.JAVA);
    }
    @Test
    public void testHatFaehigkeit1() {
        Assertions.assertTrue(this.m1
            .hatFachgebiet(Fachgebiet.C), "vorhandene Faehigkeit");
    }
    @Test
    public void testHatFaehigkeit2(){
        Assertions.assertFalse(this.m1
            .hatFachgebiet(Fachgebiet.TEST), "falsche Faehigkeit");
    }
}
```


Testablaufsteuerung (1/4)

```
public class Ablaufanalyse {  
    @BeforeAll // JUnit4 @BeforeClass  
    public static void setUpBeforeClass() throws Exception {  
        System.out.println("setUpBeforeClass");  
    }  
  
    @AfterAll  
    public static void tearDownAfterClass() throws Exception {  
        System.out.println("tearDownAfterClass");  
    }  
  
    @BeforeEach  
    public void setUp() throws Exception {  
        System.out.println("setUp");  
    }  
}
```

wird einmal für alle Tests vor allen Tests ausgeführt (static !)

wird einmal für alle Tests vor nach Tests ausgeführt (static !)

```
@AfterEach
```

```
public void tearDown() throws Exception {  
    System.out.println("tearDown");  
}
```







```
@Test
```

```
public void test1(){  
    System.out.println("test1");  
}
```

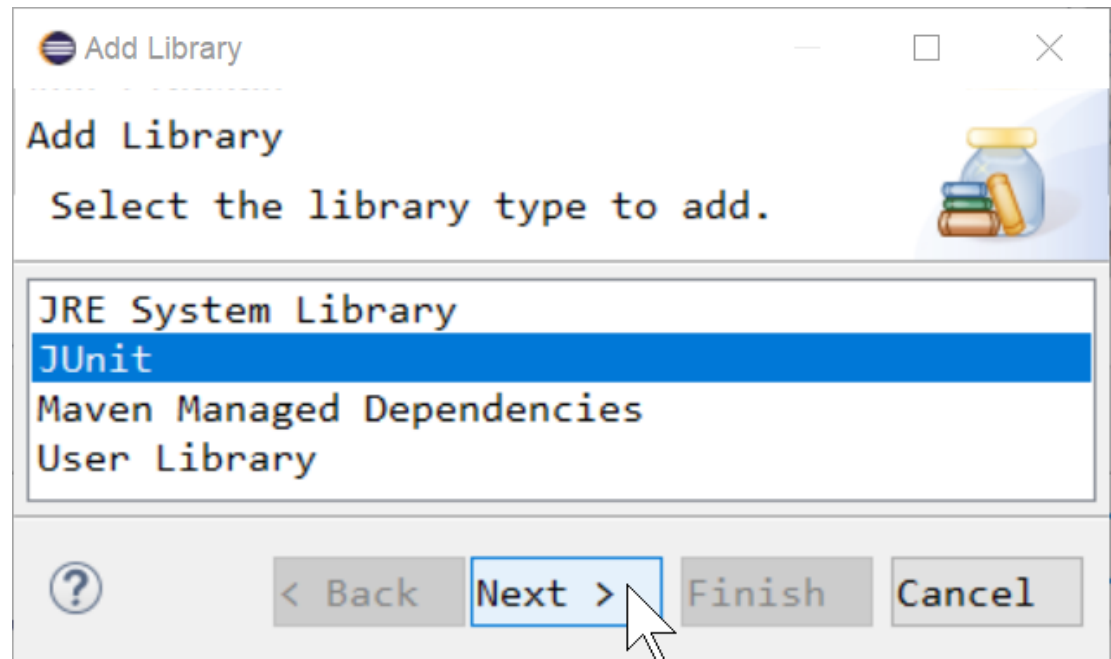
```
@Test
```

```
public void test2(){  
    System.out.println("test2");  
}  
}
```

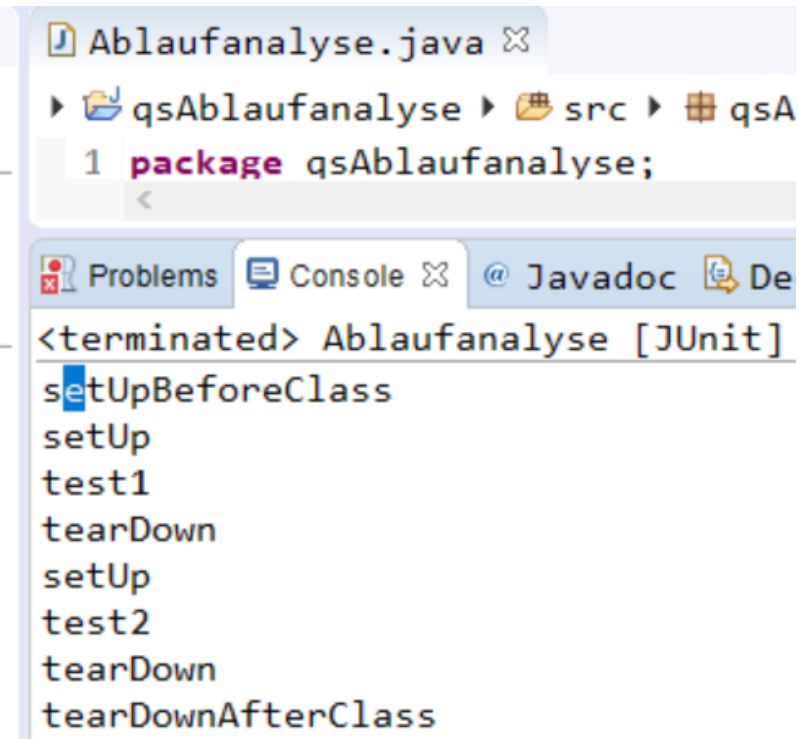
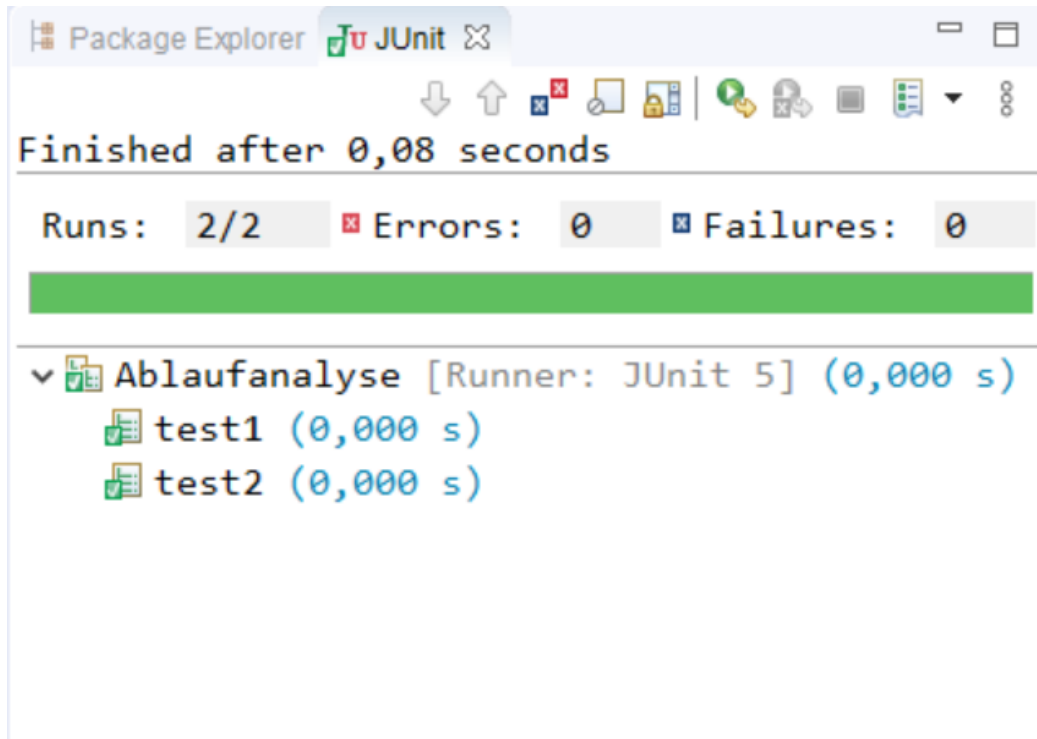
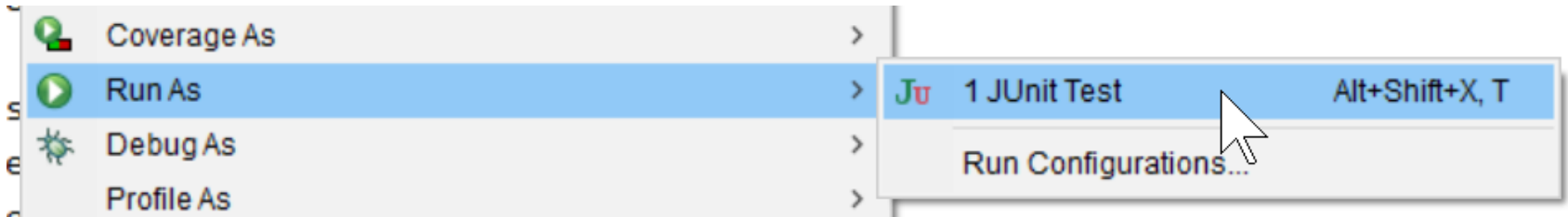
Testablaufsteuerung (3/4)

- ▼  qsAblaufanalyse
 - ▼  src
 - ▼  qsAblaufanalyse
 - >  Ablaufanalyse.java
 - >  JRE System Library [JavaSE-11]
 - >  JUnit 5

bei erster JUnit-Nutzung hinzuzufügende Bibliothek (Library) JUnit 5 kann auch getrennte JUnit 4-Tests laufen lassen



Testablaufsteuerung (4/4)



- man markiert Stellen, von denen man erwartet, dass sie nicht ausgeführt werden mit fail()
- Erinnerung: Exception bei viertem Fachgebiet

@Test

```
public void testAddFaehigkeit1a(){
    try{
        this.m1.addFachgebiet(Fachgebiet.TEST);
        Assertions.fail("fehlende Exception");
    }catch(IllegalArgumentException e){
        // oder Block einfach leer lassen
        Assertions.assertNotNull(e.getMessage());
    }catch(Exception e){
        Assertions.fail("unerwartet " + e);
    }
}
```

- wenn keine Exception auftreten soll

```
@Test
public void testAddFaehigkeit2(){
    this.m1.addFachgebiet(Fachgebiet.C);
}

@Test
public void testAddFaehigkeit2a(){
    try{
        this.m1.addFachgebiet(Fachgebiet.C);
    }catch(Exception e){
        Assertions.fail("unerwartet " + e);
    }
}
```

Testen von Exceptions (3/4) – JUnit5-Style

```
@Test // klassisch mit Assertions.fail() geht weiterhin
public void testKonstruktor() {
    Executable auszufuehren
        = () -> new Mitarbeiter(null, null);
    Assertions.assertThrows(IllegalArgumentException.class
        , auszufuehren
        , "erwartete Exception nicht geworfen");
}

// Schreibvariante
@Test
public void testKonstruktor2() {
    Assertions.assertThrows(IllegalArgumentException.class
        , () -> {new Mitarbeiter(null, null);}
        // geschweifte Klammern koennen weggelassen werden
        , "erwartete Exception nicht geworfen");
}
```

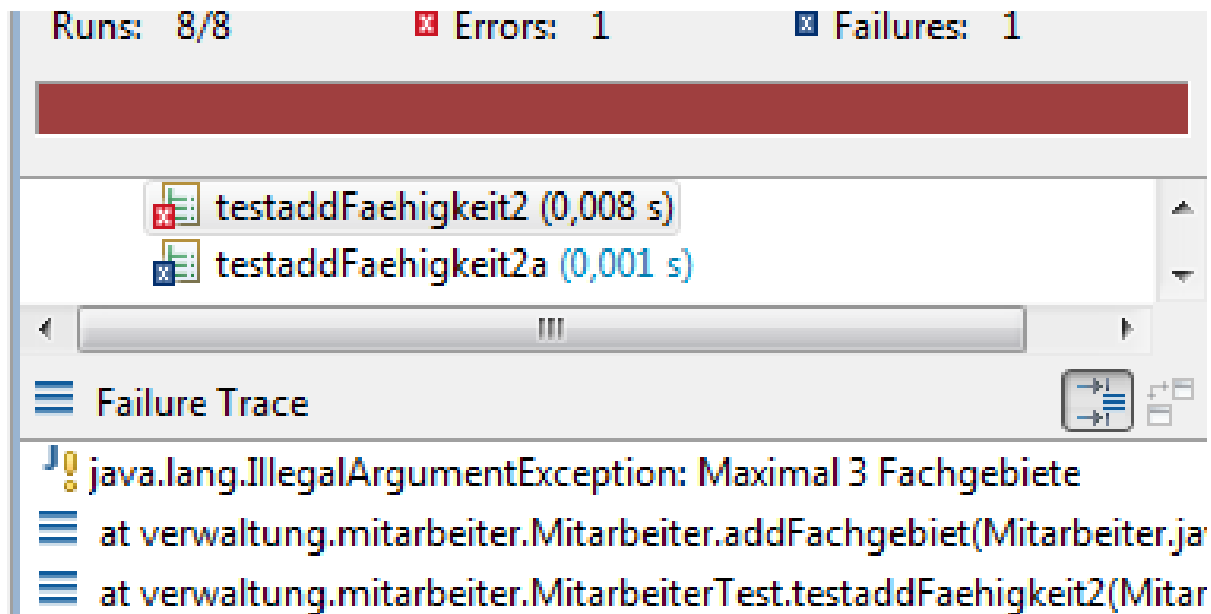
Testen von Exceptions (4/4) – JUnit5-Style

```
// Analyse der erhaltenen Exception
@Test
public void testKonstruktor3() {
    Throwable ex =
        Assertions.assertThrows(IllegalArgumentException.class
            , () -> {new Mitarbeit(null, null);}
            , "erwartete Exception nicht geworfen");
    Assertions.assertTrue(ex.getMessage().contains("Nachname")
        , " 'Nachname fehlt in Exception-Text");
}
@Test
public void testAddFaehigkeit1() {
    Executable code = () -> m1.addFachgebiet(Fachgebiet.TEST);
    Assertions.assertThrows(IllegalArgumentException.class
        , code
        , "erwartete Exception nicht geworfen");
}
```


Ausblick: langfristige Testnutzung

- Bisher geschriebene Tests werden typischerweise von Entwickelnden geschrieben
- Tests müssen archiviert und bei jedem Release neu ausführbar sein
- Beispiel: unerfahrener Neuling ersetzt

```
private List<Fachgebiet> fachgebiete = new ArrayList<>();
```



Für große Projekte sind folgende Wünsche bzgl. der Tests typisch:

- a) mehrere Testklassen sollen zusammen laufen können
- b) man möchte Tests flexibel kombinieren können;
Testwiederholung soll davon abhängen, welche Klassen betroffen sein können [schwierige Weissagung]
- c) man möchte Tests automatisiert ablaufen lassen können
möglich z. B.:
 - JUnit hat Klassen zum einfachen Start von der Konsole aus
 - Nutzung eines cron-Jobs
 - Nutzung von Continuous Integration Frameworks

zu a) und b) folgende Folien: Tests zusammenfassen zu TestSuites

Tests zusammenfassen (1/3) - weitere Testklasse

```
package verwaltung.mitarbeit;

public class Mitarbeit2Test {

    @Test
    public void testHinzuUndWeg(){
        Mitarbeit m = new Mitarbeit("Hu", "Go");
        m.addFachgebiet(Fachgebiet.C);
        m.addFachgebiet(Fachgebiet.C);
        m.removeFachgebiet(Fachgebiet.C);
        Assertions.assertFalse(m.hatFachgebiet(Fachgebiet.C));
    }
}
```

Tests zusammenfassen (2/3) - einzelne TestSuite

- Anmerkung: Aufbau nicht ganz intuitiv
package verwaltung;

```
import org.junit.platform.suite.api.SelectClasses;  
import org.junit.platform.suite.api.Suite;  
import org.junit.platform.suite.api.SuiteDisplayName;
```

@Suite

```
@SuiteDisplayName("Tests zusammengefasst") // optional
```

```
@SelectClasses({ Mitarbeit1Test.class  
                  , Mitarbeit2Test.class})
```

```
public class MitarbeitAllTest {
```

```
}
```

Tests zusammenfassen (3/3) - Suite in Suite

```
package verwaltung;
```

```
import org.junit.platform.suite.api.SelectClasses;
```

```
import org.junit.platform.suite.api.Suite;
```

```
import org.junit.platform.suite.api.SuiteDisplayName;
```

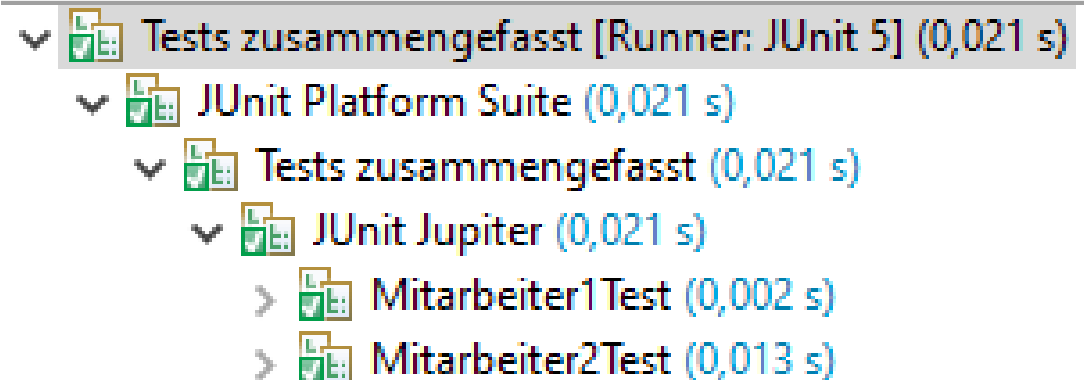
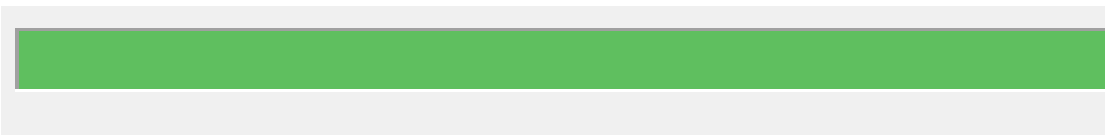
```
@Suite
```

```
@SuiteDisplayName("Tests zusammengefasst")
```

```
@SelectClasses({ MitarbeiterAllTest.class})
```

```
public class VerwaltungAllTest {
```

```
}
```



Test weiterer Anforderung (1/2)

- Anforderung „einen änderbaren Nachnamen mit mindestens zwei Zeichen hat“

```
@Test
```

```
public void testAddKonstruktor2(){  
    try{  
        new Mitarbeit(null,null);  
        Assertions.fail("fehlt Exception ");  
    }catch(IllegalArgumentException e){  
    }  
}
```

```
@Test
```

```
public void testAddKonstruktor3(){  
    try{  
        new Mitarbeit(null,"X");  
        Assertions.fail("fehlt Exception ");  
    }catch(IllegalArgumentException e){  
    }  
}
```

Test weiterer Anforderung (2/2)

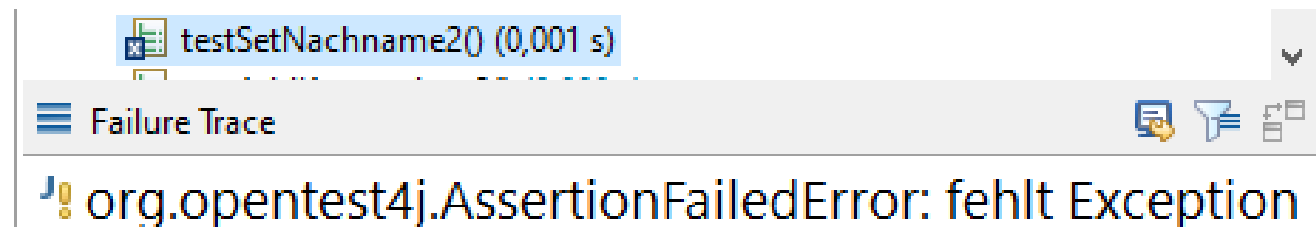
- fehlt noch: Nachnamenänderung prüfen

```
@Test
```

```
public void testSetNachname1(){  
    this.m1.setNachname("Mai"); // m1 wie vorher  
    Assertions.assertEquals(this.m1.getNachname(), "Mai");  
}
```

```
@Test // hier wird ein Fehler gefunden!
```

```
public void testSetNachname2(){  
    try{  
        this.m1.setNachname("X");  
        Assertions.fail("fehlt Exception ");  
    }catch(IllegalArgumentException e){  
    }  
}
```



The screenshot shows a failure trace in an IDE. The top part of the trace is highlighted in blue and reads "testSetNachname2() (0,001 s)". Below this, the text "Failure Trace" is visible. At the bottom of the trace, the error message is displayed: "org.opentest4j.AssertionFailedError: fehlt Exception".

Test von equals



```
@Test
public void testEquals1(){
    Assertions.assertTrue(this.m1.equals(m1)
        , this.m1.toString());
}

@Test
public void testEquals2(){
    Assertions.assertFalse(this.m1
        .equals(new Mitarbeit("Ute", "Mey")));
}

@Test
public void testEquals3(){
    Mitarbeit m2 = new Mitarbeit("Ufo", "Hai");
    m2.setId(this.m1.getId());
    Assertions.assertTrue(this.m1.equals(m2));
}
```


- Beispiel zeigt bereits, dass man sehr viele sinnvolle Tests schreiben kann
- Frage: Wieviele Tests sollen geschrieben werden?
 - jede Methode testen
 - doppelte Tests vermeiden
 - je kritischer eine SW, desto mehr Tests
 - Suche nach Kompromissen: Testkosten vs Kosten von Folgefehlern
 - ein Kompromiss: kein Test generierter Methoden (Konstruktoren, get, set, equals, hashCode)
- (nächster Block, sinnvolle Testerstellung)

- verschiedene Ansätze, Stellen zu markieren, die noch entwickelt werden (//TODO)
- häufiger kann man Tests schreiben, bevor Implementierung vorliegt (-> Design by Contract durch Interfaces)
- Tests können auch sonst so inaktiv geschaltet werden

```
@Disabled("im naechsten Release lauffaehig")
```

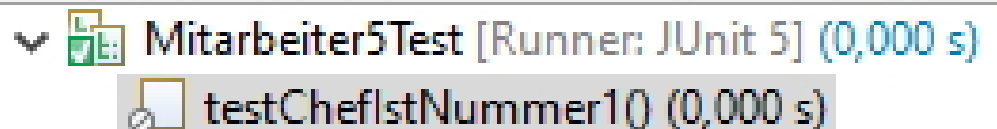
```
@Test
```

```
public void testChefIstNummer1() {
```

```
    Mitarbeit chef = new Mitarbeit("Ego", "Ich");
```

```
    Assertions.assertEquals(chef.getId(), 1, "Nr.1");
```

```
}
```

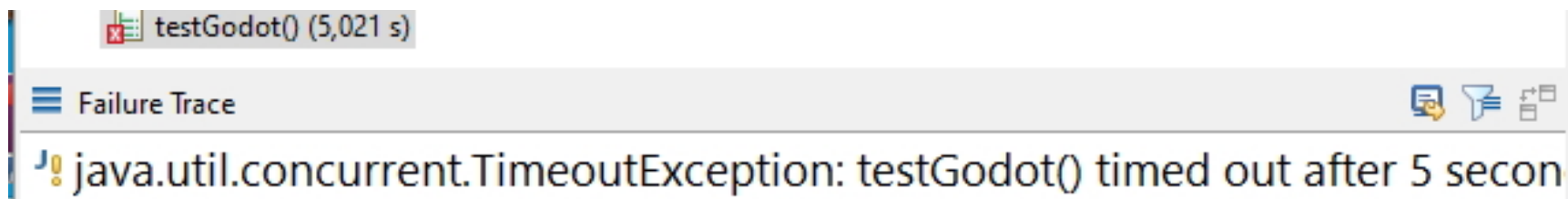


- Tests laufen nacheinander, endet einer nicht, werden andere nicht ausgeführt (default sind Sekunden, z. B. `@Timeout(5)`)

```
@Test
```

```
@Timeout(value = 5, unit = TimeUnit.SECONDS)
```

```
public void testGodot() throws InterruptedException {  
    Mitarbeit godot = new Mitarbeit("Uwe", "Godot");  
    godot.addFachgebiet(Fachgebiet.C);  
    while (!godot.getFachgebiete().isEmpty()) {  
        Thread.sleep(2_000); // 2000 ms  
    }  
}
```



```
testGodot() (5,021 s)  
Failure Trace  
java.util.concurrent.TimeoutException: testGodot() timed out after 5 secon
```

```
@Test
```

```
public void testWarteMax2Minuten() {  
    Assertions.assertTimeout(Duration.ofMinutes(2), () -> {});  
}
```

```
@Test
```

```
public void testMitTimeout() {  
    Assertions.assertTimeout(Duration.ofMillis(10)  
        , () -> {Thread.sleep(100);}) ;  
}
```

```
@Test
```

```
public void testTimeOutMitErgebnispruefung() {  
    String erg = Assertions.assertTimeout(Duration.ofMinutes(1)  
        , () -> { return "moin";}) ;  
    Assertions.assertEquals("moin", erg);  
}
```

- aufwändige Tests in weiteren Threads ausführbar
- Tests müssen dies erlauben, z. B. Serverzugriffe, DB-Nutzung
- (fast gleicher Test wie vorher)

```
@Test
```

```
@Timeout(value = 5, unit = TimeUnit.MINUTES  
    , threadMode = Timeout.ThreadMode.SEPARATE_THREAD)
```

```
public void testGodot2() throws InterruptedException {  
    Mitarbeit godot = new Mitarbeit("Uwe", "Godot");  
    godot.addFachgebiet(Fachgebiet.C);  
    while (!godot.getFachgebiete().isEmpty()) {  
        Thread.sleep(2_000); // 2000 ms  
    }  
}
```

- bereits gesehen: häufig wird eine Methode mit verschiedenen Parametern getestet
- das Schreiben gleichartiger Tests mit ähnlichen Parametern ist langweilig und zeitaufwändig
- Ansatz: Testdaten irgendwo kompakt speichern und für Testfälle einlesen
- JUnit 5 bietet mittlerweile viele Möglichkeiten Testdaten zu spezifizieren (folgen einige Beispiele)
- oft wird Hilfsmethode benötigt um Elementardaten (z. B. int, String) möglichst einfach in Objekte zu wandeln
- Erinnerung: systematische Testentwicklung später; Tests mit gleicher Aussage vermeiden

JUnit 5 – Nutzungsbeispiele – Parameter (1/7)

```
// ab JUnit 5.1 eine Methode angebar, die Stream<Arguments>  
// liefern muss; jedes Element steht fuer einen Satz  
// Testdaten
```

```
public static Stream<Arguments> daten() {  
    Arguments[] testdaten = {  
        Arguments.of(Fachgebiet.ANALYSE, Fachgebiet.C  
                    , Fachgebiet.C),  
        Arguments.of(Fachgebiet.ANALYSE, Fachgebiet.C  
                    , Fachgebiet.ANALYSE),  
        Arguments.of(Fachgebiet.C, Fachgebiet.C, Fachgebiet.C)  
    };  
    return Arrays.asList(testdaten).stream();  
}
```

JUnit 5 – Nutzungsbeispiele – Parameter (2/7)

```
@ParameterizedTest
```

```
@MethodSource({"daten"})
```

```
public void testHat( Fachgebiet f1, Fachgebiet f2  
                    , Fachgebiet f3) {  
    System.out.println("testHat");  
    Mitarbeit m1 = new Mitarbeit("Oh", "Ha");  
    m1.addFachgebiet(f1);  
    m1.addFachgebiet(f2);  
    Fachgebiet hat = f3;  
    Assertions.assertTrue( m1.hatFachgebiet(hat) );  
}
```

```
testHat  
testHat  
testHat
```


JUnit 5 – Nutzungsbeispiele – Parameter (3/7)

```
// eigene Umwandlungsklasse (hier mit merkwuerdiger Rueckgabe)
public class FachgebietConverter
    extends SimpleArgumentConverter {

    @Override
    protected Object convert(Object o, Class<?> type)
        throws ArgumentConversionException {
        System.out.println("o: " + o
            + " type: " + type.getSimpleName());
        // ueblich waere aus o passendes Objekt zu konstruieren
        return Fachgebiet.C;
    }
}
```

JUnit 5 – Nutzungsbeispiele – Parameter (4/7)

```
@ParameterizedTest
@MethodSource("daten")
public void testHatNicht(
    @ConvertWith(FachgebietConverter.class) Fachgebiet f1,
    @ConvertWith(FachgebietConverter.class) Fachgebiet f2,
    @ConvertWith(FachgebietConverter.class) Fachgebiet f3) {
    System.out.println("testHatNicht: " + f1 + f2 + f3);
    Mitarbeit m1 = new Mitarbeit("Oh", "Ha");
    m1.addFachgebiet(f1);
    m1.addFachgebiet(f2);
    Fachgebiet hat = f3;
    Assertions.assertFalse(m1
        .hatFachgebiet(
            Fachgebiet.ANALYSE));
}
```

```
o: ANALYSE type: Fachgebiet
o: C type: Fachgebiet
o: C type: Fachgebiet
testHatNicht: CCC
o: ANALYSE type: Fachgebiet
o: C type: Fachgebiet
o: ANALYSE type: Fachgebiet
testHatNicht: CCC
o: C type: Fachgebiet
o: C type: Fachgebiet
o: C type: Fachgebiet
testHatNicht: CCC
```

JUnit 5 – Nutzungsbeispiele – Parameter (5/7)

```
@ParameterizedTest
```

```
@ValueSource(strings = {"Ich", "XY"})
```

```
void test(String name) {
```

```
    System.out.println("P1 " + new Mitarbeit(name, name));
```

```
}
```

```
P1 Ich Ich (103)[ ]  
P1 XY XY (104)[ ]
```

```
@ParameterizedTest
```

```
@CsvFileSource(resources = {"/bsp.csv"  
    ,"/bsp.csv"}, numLinesToSkip = 1)
```

```
public void testWithCsvFileSource(  
    String v, String n, int a)
```

```
    System.out.printf("csv: %s %s %d\n"  
        , v, n, a);
```

```
}
```

```
bsp.csv:  
Vorname, Nachname, alter  
James T., Kirk, 87  
{, Spock, 83
```

```
csv: James T. Kirk 87  
csv: null Spock 83  
csv: James T. Kirk 87  
csv: null Spock 83
```

JUnit 5 – Nutzungsbeispiele – Parameter (6/7)

```
// kommaseparierte interne Listen nutzbar
@ParameterizedTest(name = "{0} and {1}")
@CsvSource({"Edna, 'de, Meijer'", "Kemal, Schmidt"})
public void testCsvIntern(String vor, String nach) {
    System.out.println("csv intern: "
        + new Mitarbeit(vor, nach));
}
```

```
csv intern: Edna de, Meijer (116)[ ]
csv intern: Kemal Schmidt (117)[ ]
```

```
@ParameterizedTest
@CsvSource({ "1, 2, 3", "A , 4, 3.5" })
public void testWithCsvSource(String s, int i, double d) {
    System.out.println("P2 " + s + " " + i + " " + d);
}
```

```
P2 1 2 3.0
P2 A 4 3.5
```

JUnit 5 – Nutzungsbeispiele – Parameter (7/7)

```
// Tests fuer alle Werte eines Enums
@ParameterizedTest
@NullSource // auch sonst verwendbar
@EnumSource(Fachgebiet.class)
public void testMitEnumSource(
    Fachgebiet f) {
    System.out.println(" Fachgebiet: " + f);
}
```

```
Fachgebiet: null
Fachgebiet: ANALYSE
Fachgebiet: DESIGN
Fachgebiet: JAVA
Fachgebiet: C
Fachgebiet: TEST
```

```
@ParameterizedTest
@EnumSource(value = Fachgebiet.class, names = {"JAVA", "TEST"},
    mode = Mode.INCLUDE)
public void testMitEnumSource2(Fachgebiet f) {
    System.out.println(" Fachgebiet2: " + f);
}
// INCLUDE ist default-Wert
```

```
Fachgebiet2: JAVA
Fachgebiet2: TEST
```


- bekannt: mir @Suite können Tests in Suiten und Suiten in Suiten strukturiert werden
- in großen Projekten werden oft weitere Möglichkeiten benötigt, nur Teilmengen von Tests laufen zu lassen
- da Tests zu mehreren Suiten gehören können, kennen wir eine Strukturierungsmöglichkeit
- weitere Möglichkeit @Tag(„String“), dabei gibt der String einen willkürlichen Namen, typischer einer Testartengruppe an
- Tags werden bei der Ausführung ein- oder ausgeschaltet
- (gibt in JUnit 5 weitere Strukturierungsmöglichkeiten)
- ersetzt (warum?) Kategorien @Category, @Categories aus JUnit 4, dieses basierten Interfaces
- Testausführung auch von außen (Maven, Gradle) konfigurierbar

Testauswahl mit Tags (1/3)

- man kann einzelne Tests und Testklassen markieren

```
package test.meine;  
import org.junit.jupiter.api.Tag;  
import org.junit.jupiter.api.Test;
```

```
@Tag("A")
```

```
public class ATest {
```

```
    @Test  
    public void testa1() {  
        System.out.println("a1");  
    }
```

```
    @Test  
    @Tag("Basic")  
    public void testa2() {  
        System.out.println("a2");  
    }
```

```
}
```

<<A>> ATest
testa1() <<Basic>>testa2()

Testauswahl mit Tags (2/3)

```
// Testklassen und Tests mit @Tag bzw. @Tags markierbar  
package test.meine;
```

```
import org.junit.jupiter.api.Tag;  
import org.junit.jupiter.api.Tags;  
import org.junit.jupiter.api.Test;
```

```
@Tags({@Tag("Basic"), @Tag("Increment2")})  
public class BTest {  
  
    @Test  
    public void testb() {  
        System.out.println("b");  
    }  
}
```

<<Basic>>
<<Inkrement2>>
BTest
testb()

Testauswahl mit Kategorien (3/3)

- weitere Strukturierung über **Testpakete**
package test.alle;

```
@Suite
```

```
@SelectPackages("test.meine")
```

```
@IncludeTags({"Basic"})
```

```
public class BasicSuiteTest {  
}
```

```
@Suite
```

```
@SelectPackages("test.meine")
```

```
@IncludeTags({"Basic"})
```

```
@ExcludeTags({"Inkrement2"})
```

```
public class OnlyBasicSuiteTest {  
}
```

<<Basic>> <<Inkrement2>>
BTest
testb()

a2 b

ATest
testa1() <<Basic>>testa2()

a2

Video 4

Tests können Informationen über den Test selbst erhalten und nutzen

@BeforeEach

```
public void init(TestInfo testInfo) {  
    String displayName = testInfo.getDisplayName();  
    System.out.printf("@BeforeEach %s %n", displayName);  
}
```

@RepeatedTest(2)

```
public void testMehrfach(RepetitionInfo repetitionInfo) {  
    System.out.println(repetitionInfo.getCurrentRepetition());  
}
```

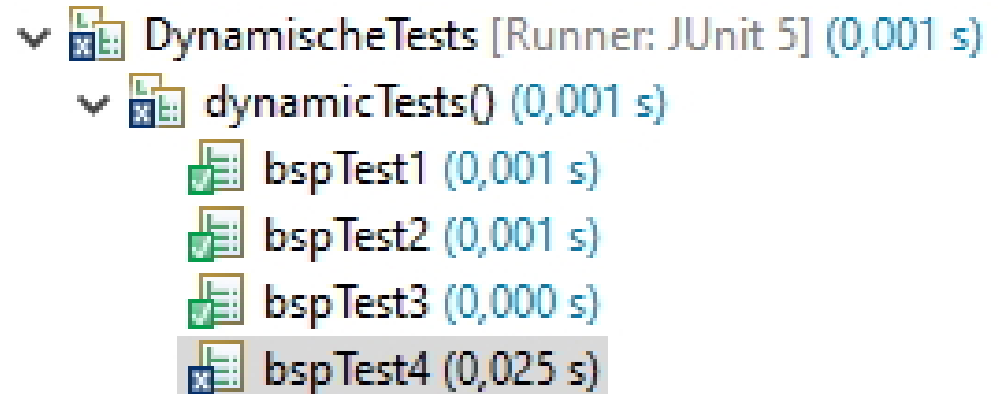
@Test // z. B. Infos fuer IDE

```
public void testFuegeErgebnisinfoHinzu(TestReporter testRepo){  
    testRepo.publishEntry("key", "value");  
}
```

JUnit 5 – kurz: dynamische Tests

```
@TestFactory
```

```
Stream<DynamicTest> dynamicTests() {  
    return IntStream  
        .range( 1, 5)  
        .mapToObj( i -> DynamicTest.dynamicTest(  
            "bspTest"+i  
            , () -> Assertions.assertTrue(i<4)  
        )  
    );  
}
```



- JUnit wird in Java programmiert, ist normales Java-Programm
- JUnit selbst in Java zu programmieren ist kein Problem (Reflexion, evtl. ByteCode-Bearbeitung)

viele Vorteile

- gute Java-Programmierende können gute Testende sein
- alle Bibliotheken auch in Tests nutzbar
- alle Strukturierungsideen, siehe Test-Architektur, übertragbar
- Nachteil: schlechte Programmierung führt zu schlechten Tests

JUnit möglichst ohne Klassenvariablen (1/2)



```
public class Static {  
    public static int val = 42;  
  
    public static int inc() {  
        return ++Static.val;  
    }  
}
```

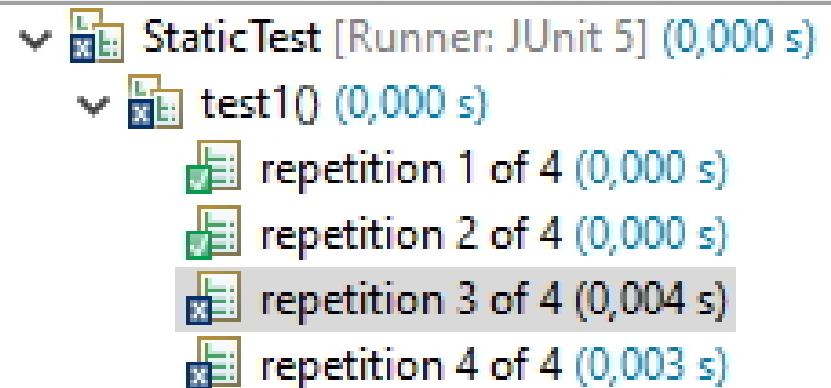
JUnit möglichst ohne Klassenvariablen (2/2)

```
public class StaticTest {  
  
    // lokale Methoden nutzbar  
    private int methode() {  
        return Static.inc();  
    }  
  
}
```

@RepeatedTest(4)

```
public void test1() {  
    Assertions.assertTrue(this.methode() < 45);  
}  
  
}
```

Runs: 4/4 ❌ Errors: 0 ❌ Failures: 2



StaticTest [Runner: JUnit 5] (0,000 s)
 test1() (0,000 s)
 repetition 1 of 4 (0,000 s) ✓
 repetition 2 of 4 (0,000 s) ✓
 repetition 3 of 4 (0,004 s) ❌
 repetition 4 of 4 (0,003 s) ❌

Testdogma von JUnit

- nach Assertions wird kein Code zu testender Code mehr geschrieben, da dieser im Fehlerfall nicht ausgeführt wird

- Programm:



- benötigt 4 JUnit-Tests:



- erhöht Testanzahl drastisch, auch wenn keine Auswirkung des Fehlers auf nachfolgenden Code besteht
- Kompromiss: Fehler werden in Liste gesammelt, die nach dem Test individuell ausgewertet werden kann
- moderne Programmiersprache GO erlaubt Teststrategie-Auswahl
- JUnit 5-Kompromiss – eigene Testrunner möglich (z. B. AssertJ)

vollständige Testausführung

```
@ExtendWith(SoftAssertionsExtension.class)
```

```
public class VollTest {
```

```
    @InjectSoftAssertions
```

```
    private SoftAssertions s;
```

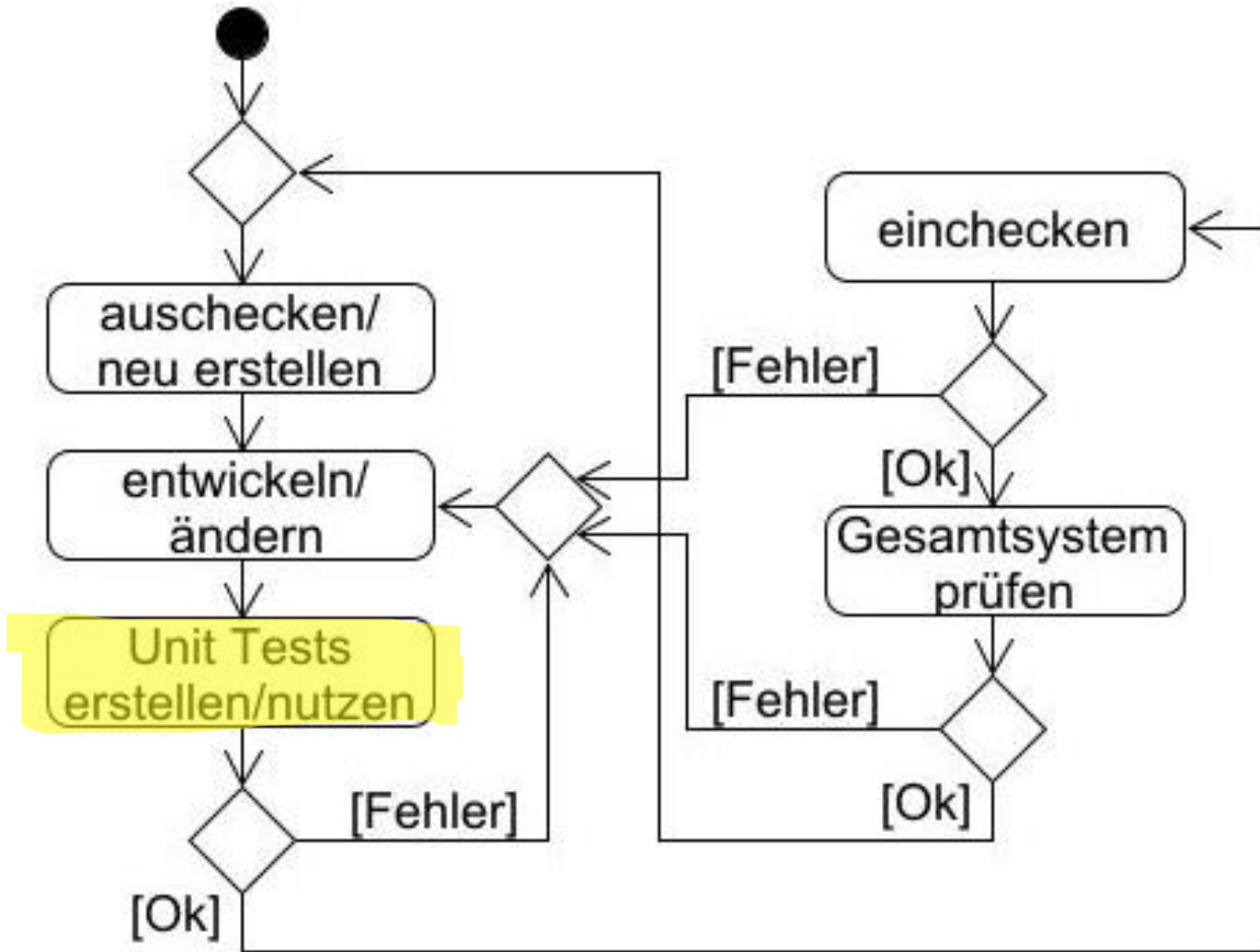
```
    @Test
```

```
    void testExample() { // moeglich: (final SoftAssertions so){  
        s.assertThat("QS").isEqualTo("Testen"); //AssertJ  
        s.assertThat(new int[]{1,2}[1] == 1).isTrue();  
        // keine Exceptions fangbar  
        // s.assertThat(7/0 == 0).isTrue();  
        System.out.println(soft.assertionErrorsCollected());  
    }
```

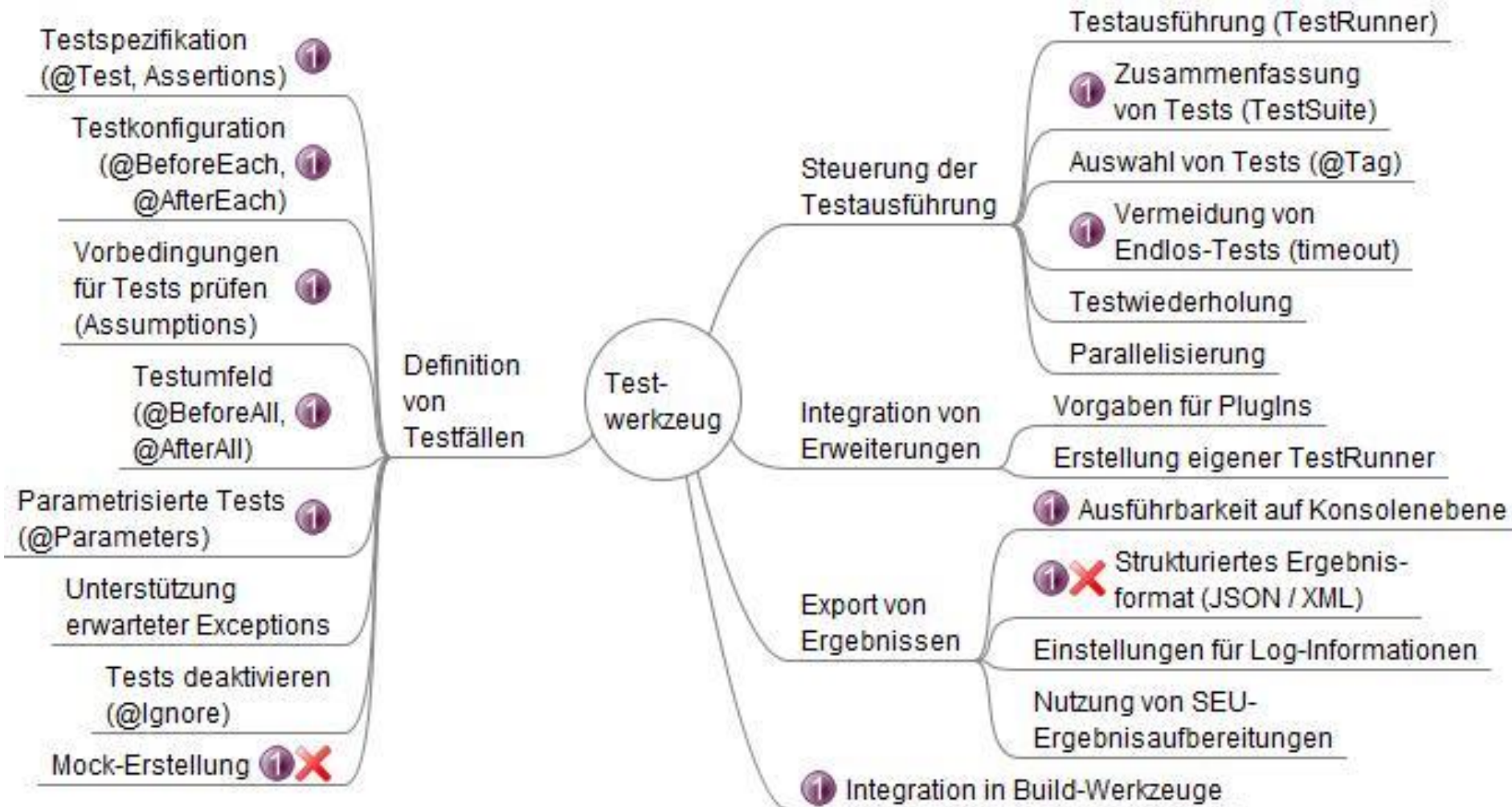
```
} // JUnit-Test scheitert am Ende als failure
```

```
[org.opentest4j.AssertionFailedError:  
expected: "Testen" but was: "QS"  
at ErrorHandlerTest.java:19,  
org.opentest4j.AssertionFailedError:  
Expecting value to be true but was false  
at ErrorHandlerTest.java:20]
```

erweiterter Entwicklungsprozess



Anforderungen an ein funktionales Testwerkzeug



① elementar wichtig
X nicht in JUnit 5

3. Äquivalenzklassentests



- Motivation der Äquivalenzidee
- typische Findung von Äquivalenzklassen
- integrierte Grenzwertanalyse
- abgeleitete Testfälle

- zentraler Satz: Für jede nicht-triviale Anforderung kann **keine Software geschrieben werden**, die für ein beliebiges Programm als Eingabe überprüft, ob diese Anforderung erfüllt ist oder nicht.
- (ausgehend von Turing-Maschinen und **Entscheidbarkeit** des Halteproblems)
- vollständiges Testen ist nicht möglich
- Tests zeigen nur die Anwesenheit von Fehlerzuständen
- für Teilmengen beliebiger Programme können Anforderungen theoretisch automatisch überprüft werden (was sehr oft an Laufzeit- oder Speichergrenzen praktisch scheitert)
- Programmverifikation für kleinere Programme möglich, aber sehr aufwendig

In der Literatur gibt es recht unterschiedlich detaillierte Klassifizierungen von Testfällen, eine erste grobe Einteilungsmöglichkeit ist :

- Datenbezogene Testfälle: Ausgehend von der Spezifikation des zu untersuchenden Objekts werden verschiedene Eingaben überlegt, deren gewünschtes Resultat aus der Spezifikation abzuleiten ist
- Ablaufbezogene Testfälle: Es wird die Struktur des zu untersuchenden Programms analysiert und versucht, möglichst alle Ablaufalternativen (if, while) durchzuspielen

- Äquivalenzklassenbildung zerlegt Menge in disjunkte Teilmengen
- jeder Repräsentant einer Teilmenge hat das gleiche Verhalten bzgl. einer vorgegebenen Operation
- Beispiel: Restklassen (modulo x), werden zwei beliebige Repräsentanten aus Restklassen addiert, liegt das Ergebnis immer in der selben Restklasse

- Übertragungsidee auf Tests: Eingaben werden in Klassen unterteilt, die durch die Ausführung des zu testenden Systems zu „gleichartigen“ Ergebnissen führen

Beispiele für Äquivalenzklassen von Eingaben

- erlaubte Eingabe: $1 \leq \text{Wert} \leq 99$ (Wert sei ganzzahlig)
 - eine gültige Äquivalenzklasse: $1 \leq \text{Wert} \leq 99$
 - zwei ungültige Äquivalenzklassen: $\text{Wert} < 1$, $\text{Wert} > 99$
- erlaubte Eingabe in einer Textliste: für ein Auto können zwischen einem und sechs Besitzer eingetragen werden
 - eine gültige Äquivalenzklasse: ein bis sechs Besitzer
 - zwei ungültige Äquivalenzklassen: kein Besitzer, mehr als sechs Besitzer
- erlaubte Eingabe: Instrumente Klavier, Geige, Orgel, Pauke
 - vier gültige Äquivalenzklassen: Klavier, Geige, Orgel, Pauke
 - eine ungültige Äquivalenzklasse: alles andere, z.B. Zimbeln

- man muss mögliche Eingaben kennen (aus Spezifikation)
- für einfache Zahlenparameter meist einfach:
 - Intervall mit gültigen Werten
 - eventuell Intervall mit zu kleinen und Intervall mit zu großen Werten (wenn z. B. alle `int` erlaubt, gibt es nur eine Äquivalenzklasse, etwas schwieriger bei `double`)
- explizit eine Menge von Werten vorgegeben:
 - jeder Wert eine Äquivalenzklasse dar
 - andere Eingaben möglich: zusätzliche Äquivalenzklasse
- falls nach Analyse der Spezifikation Grund zur Annahme besteht, dass Elemente einer Äquivalenzklasse unterschiedlich behandelt werden, ist die Klasse aufzuspalten

Spezifikation:

- Als Beispiel dient eine Methode, genauer ein Konstruktor, zur Verwaltung von Studierendendaten, der ein Name, ein Geburtsjahr und ein Fachbereich übergeben werden. Dabei darf das Namensfeld nicht leer sein, das Geburtsjahr muss zwischen 1900 und 2010 liegen und es können nur die Fachbereiche FBING, FBBWL und FBPOL aus einer Aufzählung übergeben werden.

Beispiel (2/5)

Äquivalenzklassen:

Eingabe	gültige Äquivalenzklassen	ungültige Äquivalenzklassen
Name	Ä1) nicht leer	Ä2) leer
Geburtsjahr	Ä4) $1900 < =$ Geburtsjahr $< = 2010$	Ä3) Geburtsjahr < 1900 Ä5) Geburtsjahr > 2010
Fachbereich	Ä6) FBING Ä7) FBBWL Ä8) FBPOL	

- Die Äquivalenzklassen sind eindeutig zu nummerieren. Für die Erzeugung von Testfällen aus den Äquivalenzklassen sind zwei Regeln zu beachten:
- gültige Äquivalenzklassen:
 - möglichst viele Klassen in einem Test kombinieren
- ungültige Äquivalenzklassen:
 - Auswahl eines Testdatums aus einer ungültigen Äquivalenzklasse
 - Kombination mit Werten, die ausschließlich aus gültigen Äquivalenzklassen entnommen sind
 - Grund: für alle ungültigen Eingabewerte muss eine Fehlerbehandlung existieren

Beispiel (3/5)

Testfälle nach einer Äquivalenzklassenanalyse: (jeder Klasse wird [mindestens] einmal getestet, die Testanzahl soll möglichst gering sein)

Test-nummer	1	2	3	4	5	6
geprüfte Äquivalenzklassen	Ä1 Ä4 Ä6	(Ä1) (Ä4) Ä7	(Ä1) (Ä4) Ä8	Ä2	Ä3	Ä5
Name	„Mei“	„Max“	„Schulz“	„“	„Meier“	„Meier“
Geburtsjahr	1987	1989	1985	1988	1892	2026
Fachbereich	FBING	FBBWL	FBPOL	FBING	FBING	FBING
Ergebnis	ok	ok	ok	Abbruch	Abbruch	Abbruch

- Viele Software-Fehler sind auf Schwierigkeiten in Grenzbereichen der Äquivalenzklassen zurück zu führen (z.B. Extremwert nicht berücksichtigt, Array um ein Feld zu klein)
- Untersuchung von Äquivalenzklassen um die Untersuchung der Grenzen ergänzt
- Beispiel: $1 \leq \text{Wert} \leq 99$ (wobei Wert ganzzahlig ist)
 - Äquivalenzklasse $\text{Int-Wert} < 1$: obere Grenze $\text{Wert} = 0$ (untere Grenze spielt hier keine Rolle)
 - Äquivalenzklasse $\text{Int-Wert} > 99$: untere Grenze $\text{Wert} = 100$ (obere Grenze spielt keine Rolle)
 - Äquivalenzklasse $1 \leq \text{Int-Wert} \leq 99$: untere Grenze $\text{Wert} = 1$ und obere Grenze $\text{Wert} = 99$
- Grenzfallbetrachtung geht direkt in die Testfallerzeugung ein (es gibt Ansätze, bei denen zusätzlich ein Fall mit einem Wert aus der „Mitte“ der Äquivalenzklasse genommen wird)

Beispiel (4/5)

- Testfälle nach einer Äquivalenzklassenanalyse und Grenzwertanalyse
- Anmerkung: Testfallanzahl erhöht sich meist

Test-nummer	1	2	3	4	5	6
geprüfte Äquivalenzklassen	Ä1 Ä4U Ä6	(Ä1) Ä4O Ä7	(Ä1) (Ä4) Ä8	Ä2	Ä3O	Ä5U
Name	„Mei“	„Max“	„Schulz“	„“	„Meier“	„Meier“
Geburtsjahr	1900	2010	1985	1988	1899	2011
Fachbereich	FBING	FBBWL	FBPOL	FBING	FBING	FBING
Ergebnis	ok	ok	ok	Abbruch	Abbruch	Abbruch

Beispiel (5/5)

- mögliche Übersetzung nach JUnit (Ausschnitt)

```
public class ImmatrikulationTest {  
    ...  
    @Test public void test1(){  
        try{  
            new Immatrikulation("Mei",1900,Bereich.FBING);  
        }catch(ImmatrikulationsException e){  
            Assertions.fail("falsche Exception");  
        }  
    }  
  
    @Test public void test4(){  
        try{  
            new Immatrikulation("",1988,Bereich.FBING);  
            Assertions.fail("fehlende Exception");  
        }catch(ImmatrikulationsException e){  
        }  
    }  
    ...  
}
```


Erinnerung: Zahlenbereiche (1/2)



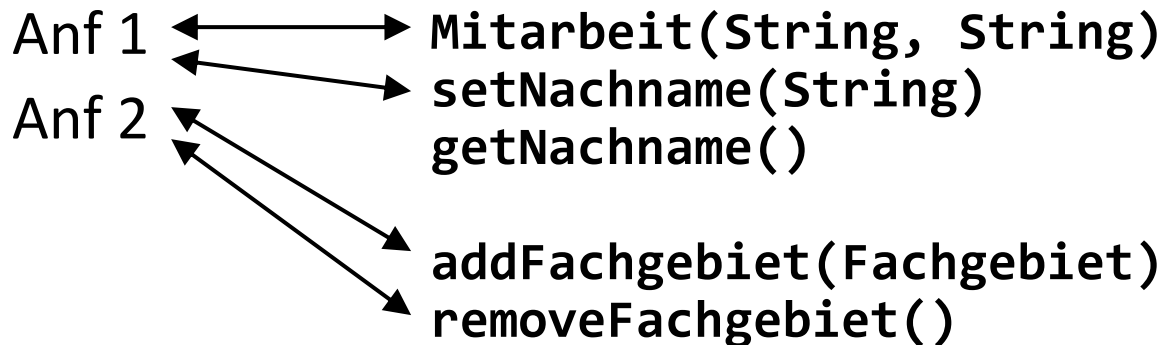
```
public static void main(String[] args) {
    Double dmax = Double.MAX_VALUE;
    Double dmin = -dmax;
    Double c = 1.0E300;
    System.out.println("dmax: "+dmax);
    System.out.println("dmin: "+dmin);
    System.out.println("dmax+c: "+(dmax+c));
    System.out.println("dmin+c: "+(dmin+c));
    System.out.println("dmax-c: "+(dmax-c));
    System.out.println("dmin-c: "+(dmin-c));
    System.out.println("dmax durch 0.0: "+(dmax/0.0));
    System.out.println("dmin durch 0.0: "+(dmin/0.0));
    System.out.println("0.0 durch 0.0: "+(0.0/0.0));
    int max = Integer.MAX_VALUE;
    int min = Integer.MIN_VALUE;
    System.out.println("max: "+max);
    System.out.println("min: "+min);
    System.out.println("max+1: "+(max+1));
    System.out.println("min+1: "+(min+1));
    System.out.println("max-1: "+(max-1));
    System.out.println("min-1: "+(min-1));
    System.out.println("durch 0: "+(max/0));
}
```

Erinnerung: Zahlenbereiche (2/2)

```
dmax: 1.7976931348623157E308
dmin: -1.7976931348623157E308
dmax+c: Infinity
dmin+c: -1.7976931248623157E308
dmax-c: 1.7976931248623157E308
dmin-c: -Infinity
dmax durch 0.0: Infinity
dmin durch 0.0: -Infinity
0.0 durch 0.0: NaN
max: 2147483647
min: -2147483648
max+1: -2147483648
min+1: -2147483647
max-1: 2147483646
min-1: 2147483647
Exception in thread "main" java.lang.ArithmeticException:
  / by zero
  at wertebereiche.Analyse.main(Analyse.java:26)
```

Spezifikationsausschnitt:

- Zu entwickeln ist eine Klasse `Mitarbeit`, wobei jedes `Mitarbeit`objekt
 - (Anf 1) einen änderbaren Nachnamen mit mindestens zwei Zeichen hat
 - (Anf 2) eine Informationssammlung mit maximal drei Fachgebieten hat, die ergänzt und gelöscht werden können
- Aufbau einer Beziehung zwischen Methoden und Anforderungen



zweites Beispiel (2/4) - Äquivalenzklassen

`setNachname(String nachname)`

Äquivalenzklassen mit Grenzwerten

- Ä1: String mit mindestens zwei Zeichen (gültig)
- Ä2: String mit einem Zeichen (ungültig)
- Ä3: null-Referenz (ungültig) → abgeleitet aus OO-Erfahrung

Test- nummer	1	2	3
geprüft	Ä1	Ä2U	Ä3
nachname	„Me“	„X“	null
Ergebnis	geändert	Abbruch	Abbruch

zweites Beispiel (3/4) - Tests

```
@Test
public void testSetNachname1(){
    Mitarbeit m = new Mitarbeit("Ute","Mai");
    m.setNachname("Me");
    Assertions.assertTrue(m.getNachname().equals("Me"));
}

@Test
public void testSetNachname2(){
    Mitarbeit m = new Mitarbeit("Ute","Mai");
    Assertions.assertThrows(IllegalArgumentException.class
        , () -> m.setNachname("X"));
}

@Test(expected=IllegalArgumentException.class) // nur JUnit4
public void testSetNachname3(){
    Mitarbeit m = new Mitarbeit("Ute","Mai");
    m.setNachname(null);
}
```

getNachname()

- Methode ohne Parameter
- generell bisher Eingabeäquivalenzklassen betrachtet; gibt auch Ausgabeäquivalenzklassen
- im Beispiel nur eine Klasse, da beliebige Strings Ergebnis (null nicht möglich)

@Test

```
public void testGetNachname(){  
    Mitarbeit m = new Mitarbeit("Ute","Mai");  
    Assertions.assertEquals("Mai", m.getNachname());  
}
```

- genauer: es gibt doch „Eingabeparameter“ im Testszenario; Werte der Exemplarvariablen

- Äquivalenzklassenbildung ist ein zentrales Verfahren, um systematisch Tests aufzubauen
- für Methoden von Objekten spielt neben Ein- und Ausgaben der interne Zustand häufig eine wichtige Rolle (erst wenn Methode x ausgeführt wurde, dann kann Methode y sinnvoll ausgeführt werden)
- Konsequenterweise muss man sich also mit dem Ein-/Ausgabeverhalten pro möglichem Objektzustand beschäftigen (was noch extrem aufwändiger sein kann)
- das bisher vorgestellte Verfahren kann in der reinen Form nur für gedächtnislose Objekte genutzt werden

Spezifikation:

- In einem Bestellsystem wird für jeden Kunden im Objekt einer Klasse Zuverlässigkeit festgehalten, wie er bezüglich seines Zahlverhaltens eingestuft wird. Diese Einstufung wird durch die folgende Aufzählung beschrieben.

```
public enum Bezahlstatus {  
    STANDARD, GEPRUEFT, KRITISCH;  
}
```

- Die Klasse Zuverlässigkeit soll eine Methode anbieten, mit der geprüft werden soll, ob eine Bestellung über eine bestimmte Summe ohne eine weitere Liquiditätsprüfung erlaubt werden soll. Die Bestellung soll für geprüfte Kunden immer und für kritische Kunden nie ohne zusätzliche Prüfung möglich sein. Für sonstige Kunden muss eine Prüfung ab einer Bestellsumme von 500€ erfolgen.

Äquivalenzklassen und Objekte (2/5)

```
public class Zuverlaessigkeit { // zu testende Klasse
    private Bezahlstatus status;

    public void setStatus(Bezahlstatus status){
        this.status=status;
    }

    public boolean einkaufssummePruefen(int wert){
        switch(status){
            case GEPRUEFT:{
                return true;
            }
            case STANDARD:{
                return wert<500;
            }
        }
        return false;
    }
}
```

Äquivalenzklassen und Objekte (3/5)

- erste Überlegung: nur Parameter **wert** zu testen, Ä1:[wert<500] und Ä2:[wert>=500] (zwei Tests)
- zweite Überlegung: Objektzustand als weiteren Parameter berücksichtigen; ergibt drei weitere Klassen Ä3, Ä4, Ä5:

Testnummer	1	2	3
geprüfte Äquivalenzklassen	Ä10 Ä3	Ä2U Ä4	(Ä10) Ä5
wert	499	500	499
status	STANDARD	GEPRUEFT	KRITISCH
Ergebnis	true	true	false

Äquivalenzklassen und Objekte (4/5)

```
public class ZuverlaessigkeitTest {  
  
    private Zuverlaessigkeit zvl;  
  
    @BeforeEach  
    protected void setUp() throws Exception {  
        this.zvl= new Zuverlaessigkeit();  
    }  
  
    @Test  
    public void testGeprueft1(){  
        this.zvl.setStatus(Bezahlstatus.GEPRUEFT);  
        Assertions.assertTrue(this.zvl.einkaufssummePruefen(499));  
    }  
  
    @Test  
    public void testGeprueft2(){  
        this.zvl.setStatus(Bezahlstatus.GEPRUEFT);  
        Assertions.assertTrue(this.zvl.einkaufssummePruefen(500));  
    }  
}
```

Äquivalenzklassen und Objekte (5/5)

```
@Test
public void testKritisch1(){
    this.zvl.setStatus(Bezahlstatus.KRITISCH);
    Assertions.assertTrue(!this.zvl.einkaufssummePruefen(499));
}
```

```
@Test
public void testKritisch2(){
    this.zvl.setStatus(Bezahlstatus.KRITISCH);
    Assertions.assertTrue(!this.zvl.einkaufssummePruefen(500));
}
```

```
@Test
public void testStandard1(){
    this.zvl.setStatus(Bezahlstatus.STANDARD);
    Assertions.assertTrue(this.zvl.einkaufssummePruefen(499));
}
```

```
// ... public void testStandard2(){
}
```

- bisher Äquivalenzklassen für Variablen (Parameter + Exemplarklassen individuell) [ok, ist Testpflicht]
- konsequent müssen Kombinationen von Äquivalenzklassen betrachtet werden
- Variante 1: Alle Kombinationen betrachten (leider nur selten machbar) (d. h. sechs Tests)
- Variante 2 (intensives [fehlerträchtiges?] Studium der Spezifikation): Neue Äquivalenzklassen durch Kombination von alten Äquivalenzklassen betrachten:

[status==GEPRUEFT]

[status==STANDARD && wert==499]

[status==STANDARD && wert==500] (fehlt bisher)

[status==KRITISCH]

- Spezifikation: Schreibe eine Methode `max()`, der drei Integer-Werte übergeben werden, die den größten Wert dieser Werte zurück gibt

```
public class Maxi {  
    public static int max(int x, int y, int z){  
        int max = 0;  
        if (x>z) max = x;  
        if (y>x) max = y;  
        if (z>y) max = z;  
        return max;  
    }  
}
```

- nullter Klassenansatz: jeder Parameter darf beliebige Werte annehmen, jeweils eine Klasse für `x`, `y`, `z`
- erster Klassenansatz: es gibt keine Ausnahmefälle, also drei Ergebnisse : Maximum an erster, zweiter oder dritter Stelle

Schwierige Äquivalenzklassenbildung (2/5)

```
public class MaxiTest {  
    public void testErstesMax(){  
        Assert.assertTrue("Maximum an erster Stelle",  
            7 == Maxi.max(7,5,4));  
    }  
    public void testZweitesMax(){  
        Assert.assertTrue("Maximum an zweiter Stelle",  
            7 == Maxi.max(5,7,4));  
    }  
    public void testDrittesMax(){  
        Assert.assertTrue("Maximum an dritter Stelle",  
            7 == Maxi.max(4,5,7));  
    }  
}
```

- Alle Tests laufen erfolgreich!
- allerdings war die Klassenbildung zu ungenau (nicht disjunkt),
- nächster Versuch mit den „Klassen“ (Permutationen der Reihenfolge):
 - $x \geq y \geq z$
 - $x \geq z \geq y$
 - $z \geq x \geq y$
 - $z \geq y \geq x$
 - $y \geq x \geq z$
 - $y \geq z \geq x$
- [Frage: sind dies Äquivalenzklassen ??]

- sechs Testfälle, Ausschnitt:

```
public void testXYZ(){
    Assert.assertTrue("X>=Y>=Z", 7 == Maxi.max(7,5,4));
}
public void testXZY(){
    Assert.assertTrue("X>=Z>=Y", 7 == Maxi.max(7,4,5));
}
public void testYXZ(){
    Assert.assertTrue("Y>=X>=Z", 7 == Maxi.max(5,7,4));
}
```

- Der Fall $X \geq Z \geq Y$ offenbart, dass das Verfahren nicht funktioniert
- Allerdings ist die Äquivalenzklassenwahl nicht sauber (nicht disjunkt)

- Saubere Äquivalenzklassen sind:
 - $x > y = z$ $y = z > x$
 - $y > x = z$ $x = z > y$
 - $z > y = x$ $y = x > z$
 - $z > y > x$ $z > x > y$
 - $y > z > x$ $y > x > z$
 - $x > z > y$ $x > y > z$
 - $x = y = z$
- Bei Grenzwertanalyse muss Übergang zwischen untersuchten Klassen betrachtet werden, d.h. wenn zwei oder drei Argumente gleich sind (letzter Fall liefert auch Fehler)
- Aus Spezifikation nicht ableitbar, ggfs. aus der Erfahrung mit Mathematik zu ergänzen, ist Untersuchung negativer Zahlen

- Äquivalenzklassen zentrales Hilfsmittel bei der Testerstellung; können auf Programmcodeebene, aber auch bei Abnahmen genutzt werden
- Grundregel: Zu jedem Stück Spezifikation sollte man Tests mit Äquivalenzklassen schreiben können
- typisch ist, dass nicht alle Kombinationen von elementaren Äquivalenzklassen betrachtet werden können (was in kritischen Fällen anzustreben sein sollte)
- Ansatz: Betrachte mehrere Variablen zusammen; trotzdem kombinatorische Explosion möglich
- hilfreich wäre ein Maß, wie weit man beim Testen bereits ist (→ Überdeckungsmaße)

4. Überdeckungstests



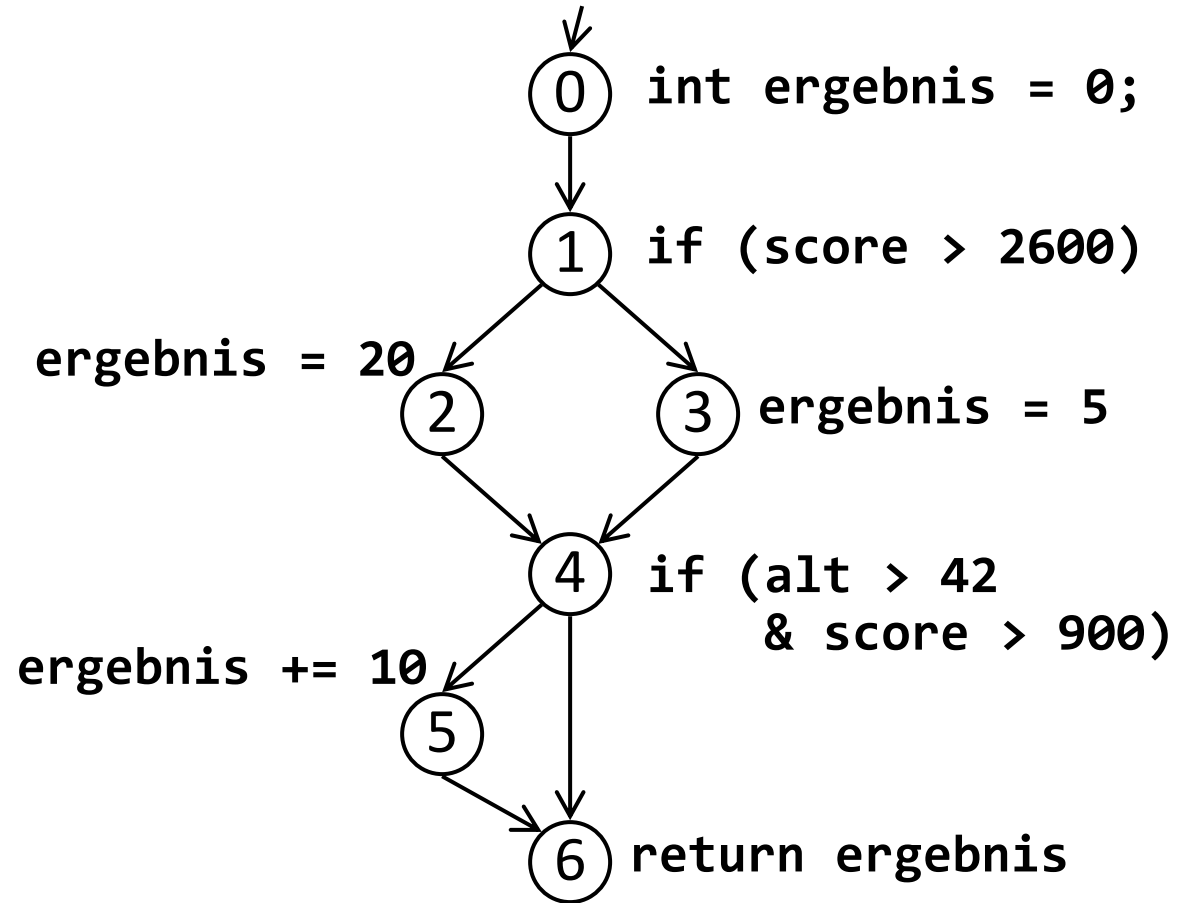
Video 5

- Kontrollflussgraph
- Anweisungsüberdeckung
- Zweigüberdeckung
- einfache Bedingungsüberdeckung
- minimale Mehrfachüberdeckung
- Herausforderung Polymorphie
- Automatische Überdeckungsberechnung
- datenflussabhängige Fehler
- Datenflussgraph

- Zu entwickeln ist eine Methode mit der der Bonus eines Kunden berechnet wird. Hierbei wird der interne Score des Kunden und das Alter des Kunden berücksichtigt. Liegt der Score über 2600 wird der Ausgangswert des Bonus mit 20, sonst mit 5 festgelegt. Liegt das Alter des Kunden über 42 und der Score über 900, wird der Bonus um 10 erhöht.

Methode mit zugehörigem Kontrollflussgraph

```
public int bonus(  
    int score,  
    int alt){  
    int ergebnis = 0;  
    if (score > 2600){  
        ergebnis = 20;  
    } else {  
        ergebnis = 5;  
    }  
    if (alt > 42  
        & score > 900){  
        ergebnis += 10;  
    }  
    return ergebnis;  
}
```



eines Programms P ist ein gerichteter Graph

$$\text{KFG}(P) =_{\text{def}} G = (V, E, V_{\text{Start}}, V_{\text{Ziel}})$$

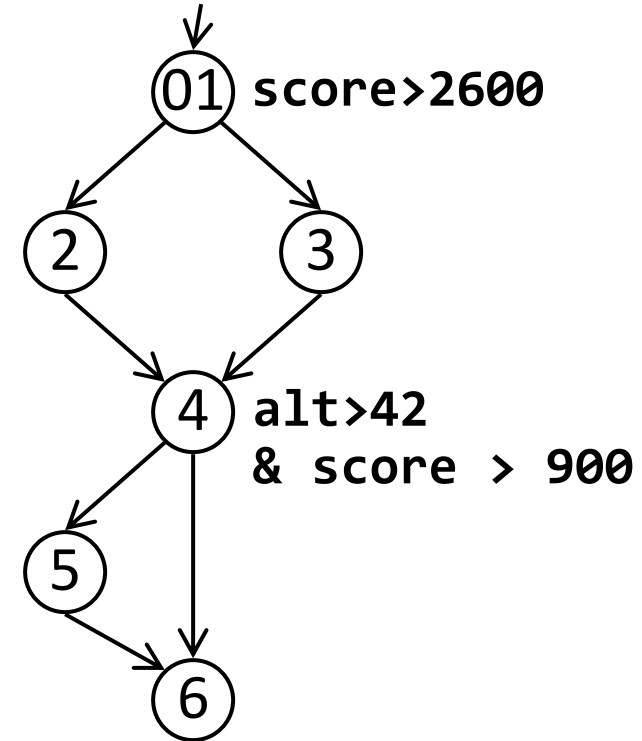
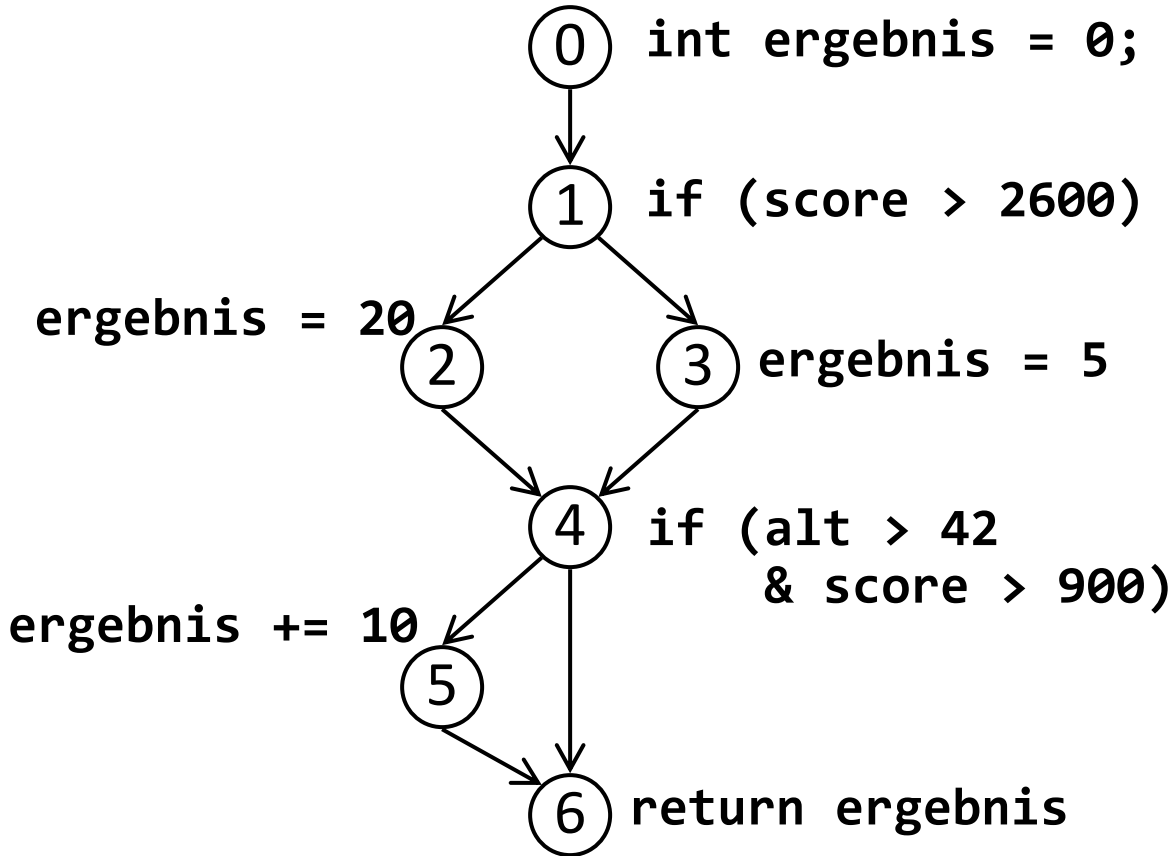
- V : Menge der Knoten (Anweisungen des Programms)
- E ist Teilmenge von $V \times V$: Menge der Kanten
(Nachfolgerrelation bezüglich der Ausführung des Programms)
- $V_{\text{Start}}, V_{\text{Ziel}}$ aus V : Ausgewählte Knoten für Start, Ende des Programms
(V_{Ziel} kann auch eine Menge von Knoten sein)

Wunsch: Graph sollte unabhängig von der Formatierung sein!

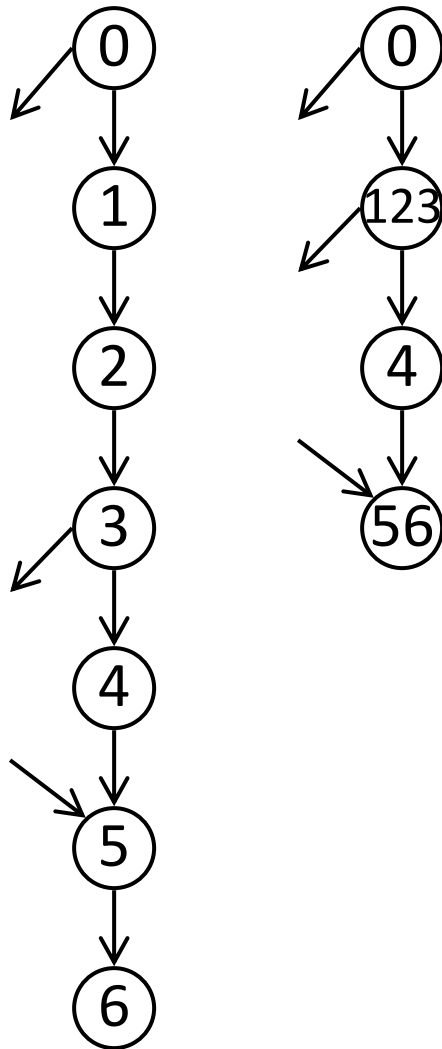
folgende Regeln, mit denen mehrere Knoten k_1, k_2, \dots, k_n , die nacheinander durchlaufen werden können, also $k_1 \rightarrow k_2 \rightarrow \dots \rightarrow k_n$, zu einem Knoten verschmolzen werden.

- Die Knotenfolge wird bei jedem Durchlauf immer nur über k_1 betreten, es gibt außer den genannten Kanten keine weiteren Kanten, die in k_2, \dots, k_n enden.
- Die Knotenfolge wird bei jedem Durchlauf immer nur über k_n verlassen, es gibt außer den genannten Kanten keine weiteren Kanten, die in k_1, \dots, k_{n-1} beginnen.
- Die Knotenfolge ist maximal bezüglich a) und b).

Beispiele für Normalisierung (1/2)



Beispiele für Normalisierung (2/2)



- 0 und 123 nicht verschmelzbar, da 0 anders als nach 123 verlassen werden kann
- 123 und 4 nicht verschmelzbar, da 123 anders als nach 4 verlassen werden kann
- 4 und 56 nicht verschmelzbar, da 56 anders als über 4 betreten werden kann

- Ein *vollständiger Pfad* ist eine Folge von verbundenen Knoten (über Kanten) im KFG, die mit V_{start} beginnt, und mit V_{ziel} endet
- Die möglichen Ausführungsreihenfolgen des Programms sind eine Teilmenge der vollständigen Pfade
- Wunsch: Durchlauf „repräsentativer“ vollständiger Pfade beim Test
- Überdeckung aller vollständigen Pfade ist im allgemeinen nicht ausführbar
- Ansatz: Verschiedene Approximationsstufen (Anweisungsüberdeckungstest, ..., Mehrfach-Bedingungsüberdeckungstest) für die Menge der vollständigen Pfade bei Auswahl der Testdurchläufe wählen

Anweisungsüberdeckung (C0)

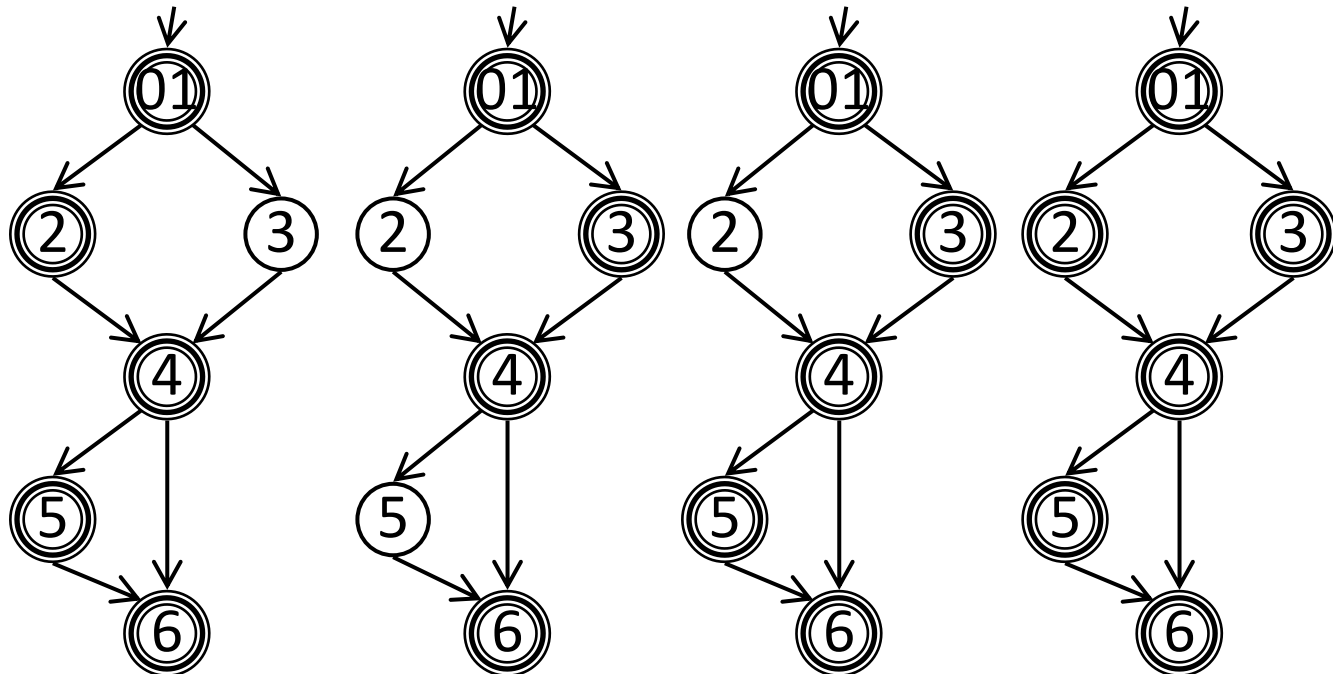
- Ziel: alle Anweisungen des Programms durch Wahl geeigneter Testdaten mindestens einmal ausführen, alle Knoten des KFG mindestens einmal besuchen.

- Testmaß $C0 = \frac{\text{Anzahl der ausgeführten Knoten}}{|M|}$

- weiterer Name: Knotenüberdeckung
- Ziel $C0=1$ (= 100%)
- typischerweise wird eine Menge von Testfällen benötigt, um $C0=1$ zu erreichen
- praktisch $C0=1$ oft schwierig (z. B. wg. Exceptions)
- Schwierigkeiten können auf schlechte Programmierung hindeuten

Beispiele für C0-Überdeckungen

Test	T1	T2	T3	T1+T3
score	2601	900	2600	
alt	43	43	88	
ergebnis	30	5	15	

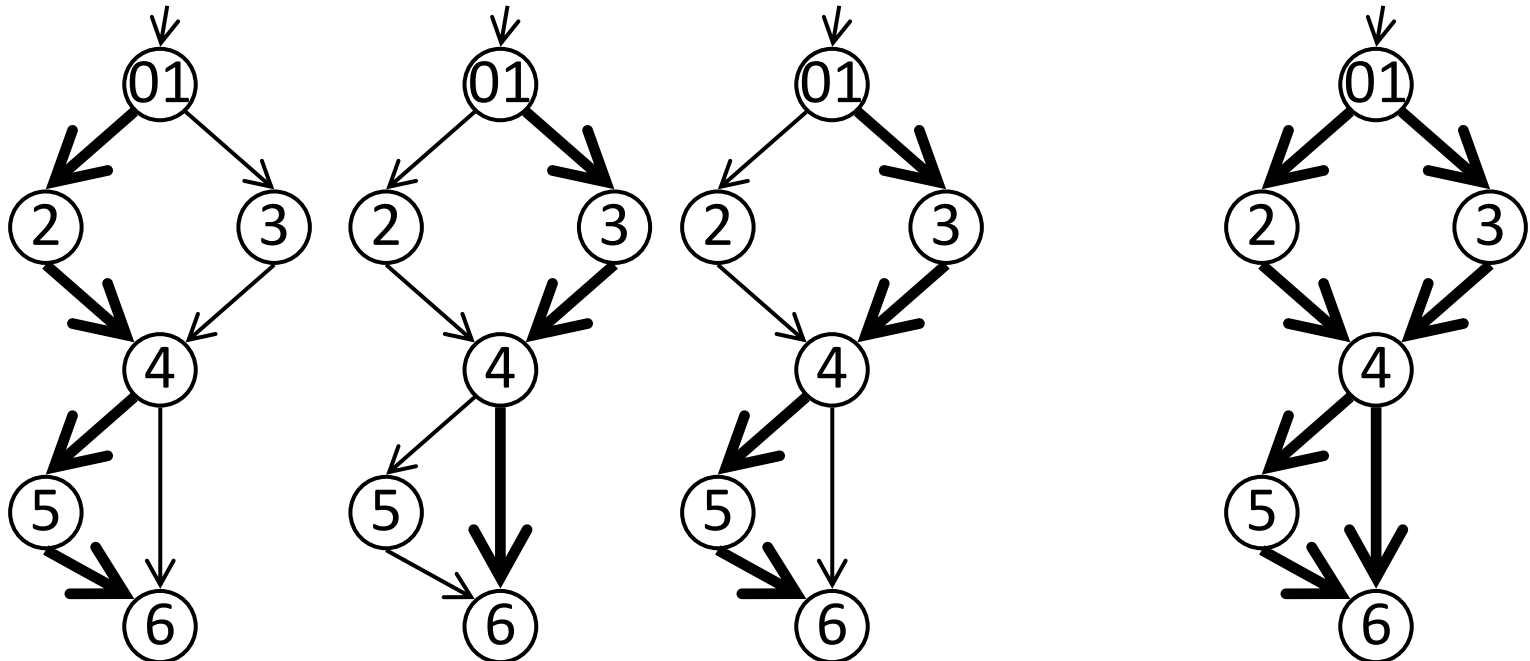


C0	5/6	4/6	5/6	6/6
-----------	-----	-----	-----	-----

- Ziel :alle Kanten des KFG überdecken, d.h. alle Pfade des Programms einmal durchlaufen
- Testmaß $C1 = \frac{\text{Anzahl der durchlaufenen Kanten}}{|E|}$
- weiterer Name: Zweigüberdeckung
- Ziel $C1=1$ (= 100%)
- typischerweise wird eine Menge von Testfällen benötigt, um $C1=1$ zu erreichen
- $C1=1$ impliziert $C0=1$ (nicht andersherum)

Beispiele für C1-Überdeckungen

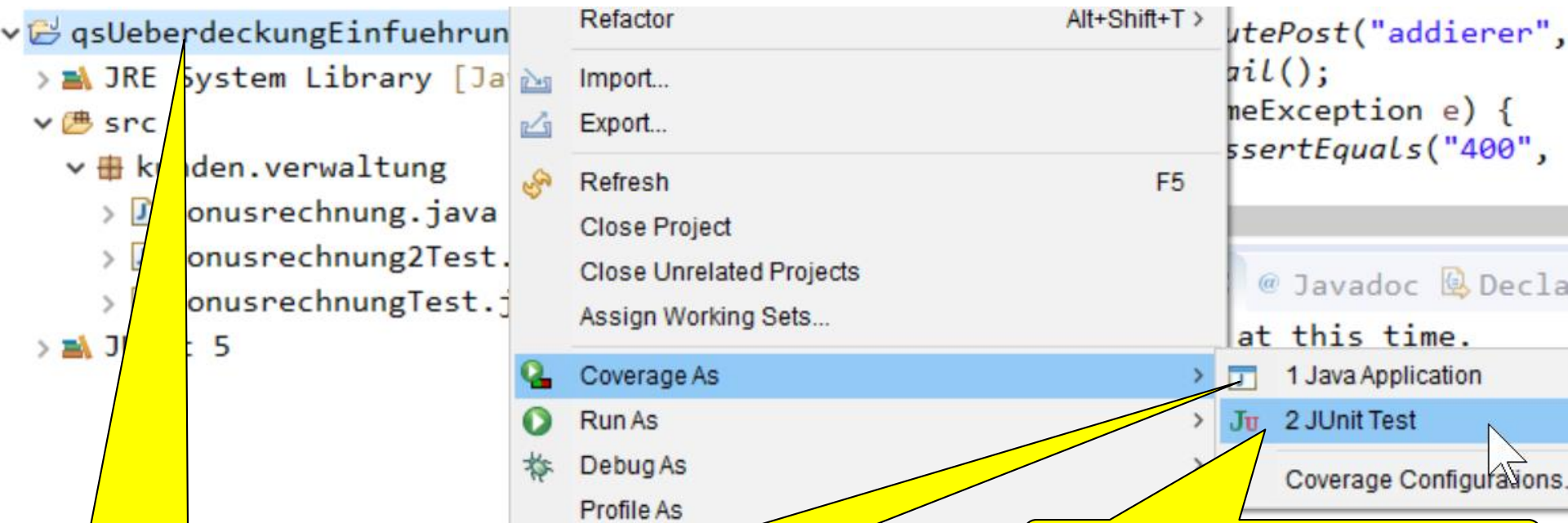
Test	T1	T2	T3	T1+T3	T1+T2
score	2601	900	2600		
alt	43	43	88		
ergebnis	30	5	15		



C0	4/7	3/7	4/7	6/7	7/7
----	-----	-----	-----	-----	-----

- Vorteile der Anweisungsüberdeckung:
 - einfach
 - geringe Anzahl von Eingabedaten
 - nicht ausführbare Programmteile werden erkannt
- großer Nachteil der Anweisungsüberdeckung:
 - Logische Aspekte werden nicht überprüft
- deshalb: Zweigüberdeckungstest gilt als Minimalkriterium im Bereich des dynamischen Softwaretests,
 - schließt den Anweisungsüberdeckungstest ein,
 - fordert die Ausführung aller Zweige eines KFG,
 - jede Entscheidung mindestens einmal wahr und falsch
- Nachteile der Zweigüberdeckung:
 - Fehlende Zweige werden nicht automatisch entdeckt
 - Kombinationen von Zweigen sind unzureichend geprüft
 - Komplexe Bedingungen werden nicht analysiert

Beispiel: Nutzung von CodeCoverage (1/2)



Rechtsklick auf Projekt

Wenn Überdeckung nur bei Ausführung (also ohne Tests) gemessen werden soll

Testüberdeckungsmessung







führt immer alle Testklassen aus, die auf "Test" enden

Beispiel: Nutzung von CodeCoverage (2/2)

qsUeberdeckungEinfuehrung > src > kunden.verwaltung > Bonusrechnung >

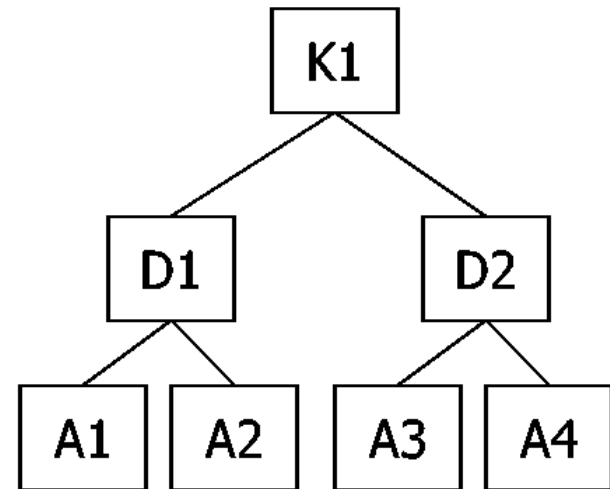
```
3 public class Bonusrechnung {
4
5     public int bonus(int score, int alt) {
6         int ergebnis = 0;
7         if (score > 2600) {
8             ergebnis = 20;
9         } else {
10            ergebnis = 5;
11        }
12        if (alt > 42 & score > 900 ) {
13            ergebnis += 10;
14        }
15    }
16 }
```

Problems Console Javadoc Declaration Search Coverage Call Hierarchy

Element	Coverage	Covered Branches	Missed...	Tot...
qsUeberdeckungEinfuehrung	 70,0 %	14	6	20
src	 70,0 %	14	6	20
kunden.verwaltung	 70,0 %	14	6	20
BonusrechnungTest.java	 50,0 %	5	5	10
Bonusrechnung2Test.java	 50,0 %	1	1	2
Bonusrechnung.java	 100,0 %	8	0	8

Bedingungsüberdeckungstest

- Ziel: Teste gezielt Bedingungen in Schleifen und Auswahlkonstrukten
- Bedingungen sind Prädikate
 - A1,..., A4 atomar
 - z.B. $(x==1)$ [auch $!(x==1)$]
 - zusammengesetzt
 - Konjunktion K
 - Disjunktion D
 - $((x==1) || (x==2)) \&\& ((y==3) || (y==4))$ hat 7 Teilprädikate: $x==1, x==2, y==3, y==4, (x==1) || (x==2), (y==3) || (y==4), ((x==1) || (x==2)) \&\& ((y==3) || (y==4))$
- Hinweis: Unterschied zwischen $||$ und $\&\&$ sowie $|$ und $\&$ (`if(true || 5/0==0)` läuft, `if(true | 5/0==0)` läuft nicht)



Einfache Bedingungsüberdeckung (C2)

- Ziel: alle atomaren Prädikate einmal TRUE, einmal FALSE
- Testmaß:
$$C2 = \frac{|wahre\ Atome| + |falsche\ Atome|}{2 * |alle\ Atome|}$$
- $|alle\ Atome|$ = Anzahl aller Atome,
- $|wahre\ Atome|$ = Anzahl aller Atome, die nach true ausgewertet werden (analog $|falsche\ Atome|$)

Beispiele für C2-Überdeckungen

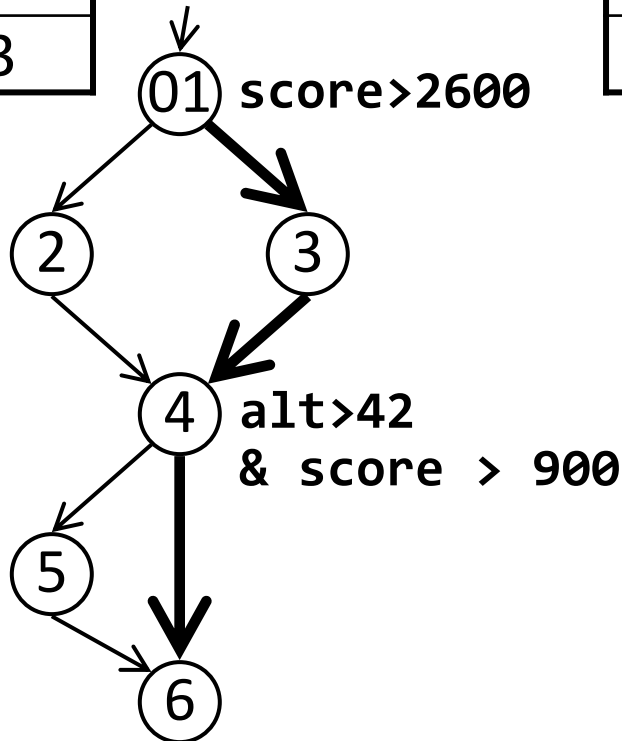
Test	T1	T2	T3	T1+T2	T4	T2+T4
score	2601	900	2600		2601	
alt	43	43	88		42	
ergebnis	30	5	15		20	
score > 2600	t	f	f	t f	t	f t
alt > 42	t	t	t	t	f	t f
score > 900	t	f	t	t f	t	f t
C2	3/6	3/6	3/6	5/6	3/6	6/6

- T1+T2 ist 100% C1-Überdeckung, aber nicht C2-Überdeckung
- aus C1-Überdeckung folgt nicht C2-Überdeckung

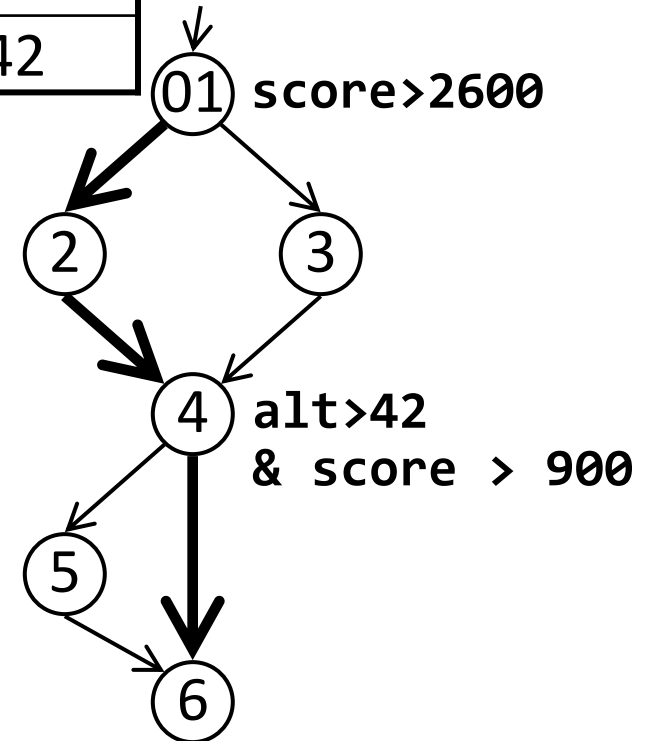
Analyse von T2+T4 bzgl. C1



Test	T2
score	900
alt	43



Test	T4
score	2601
alt	42



- aus C2-Überdeckung muss nicht C1-Überdeckung folgen (!)

Kein Zusammenhang zwischen C1 und C2

- Die Nummerierung ist historisch gewachsen
- betrachte `if (a || b)`
 - `a=true, b=false` `a=false, b=true`
 - garantiert vollständige C2-Überdeckung
 - else-Zweig wird nicht durchlaufen, kein C1 oder C0
 - `a=true, b=false` `a=false, b=false`
 - if- und else-Zweig wird durchlaufen, damit C1 und C0
 - keine vollständige C2-Überdeckung, da `b=true` fehlt

Coverage misst auch C2-Überdeckung

```
public int mach11(int x, int y, int z){
    int erg;
    if (x>0 && y>0 && z>0){
        erg = 1;
    } else {
        erg = -1;
    }
    return erg;
}
```

2 of 6 branches missed.

```
@Test
public void test11(){
    Assertions.assertEquals(1, b.mach11(1, 1, 1));
}
@Test
public void test13(){
    Assertions.assertEquals(-1, b.mach11(0, 0, 0));
}
```

- Achtung Kurzschlussauswertung bei zweitem Testfall

JaCoCo ohne Kurzschlussauswertung

```
public int mach1(int x, int y, int z){
    int erg;
    if (x>0 & y>0 & z>0){
        erg = 1;
    } else {
        erg = -1;
    }
    return erg;
}
```

1 of 8 branches missed.

```
@Test
public void test1(){
    Assertions.assertEquals(-1, b.mach1(0, 1, 0));
}
@Test
public void test3(){
    Assertions.assertEquals(-1, b.mach1(1, 0, 1));
}
```

- zu beachten, jeder Ausdruck einmal false und true, aber Gesamtausdruck nicht (deshalb zwei Möglichkeiten mehr)

Minimale Mehrfachbedingungsüberdeckung (C3)

- Ziel: alle Prädikate und Teil-Prädikate einmal TRUE, einmal FALSE
- Testmaß:

$$C3 = \frac{|wahre\ Teilprädikate| + |falsche\ Teilprädikate|}{2 * |alle\ Teilprädikate|}$$

- Da immer auch gesamter Boolescher Ausdruck betrachtet, folgt aus 100% C3- immer 100% C1-Überdeckung

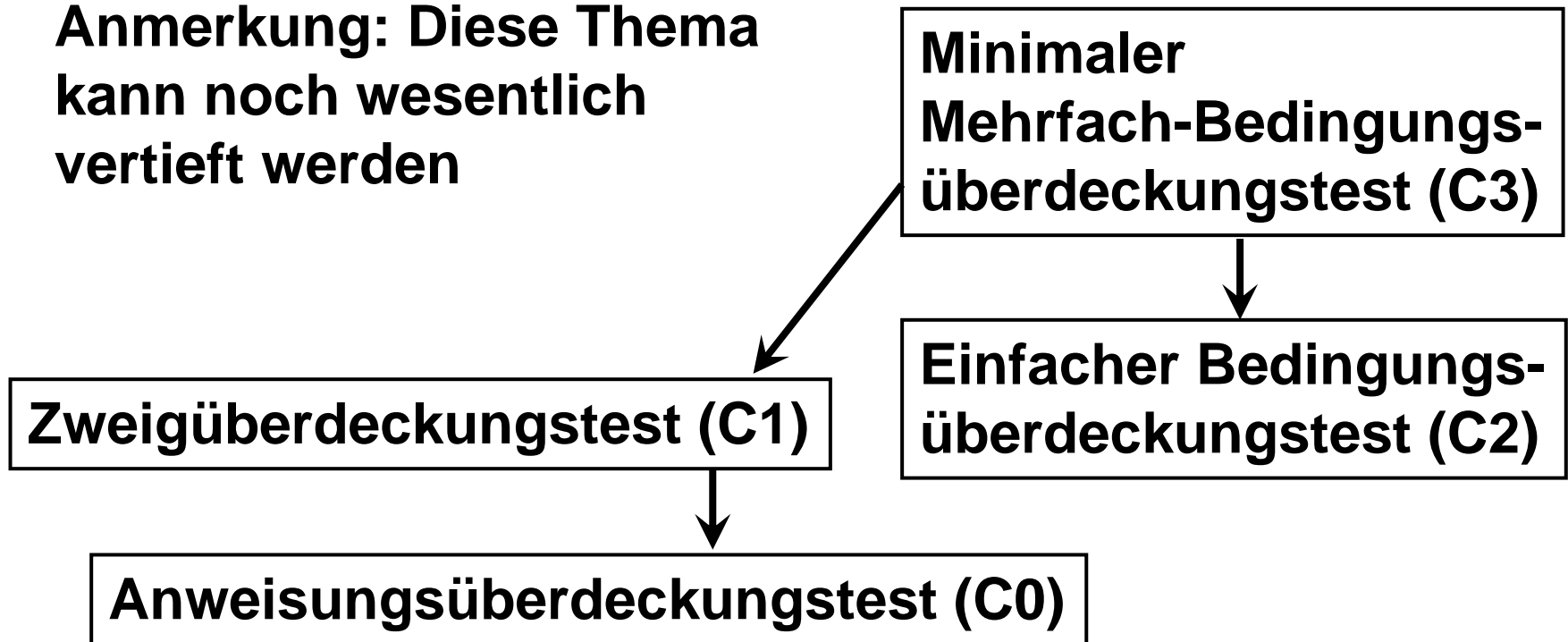
Beispiele für C3-Überdeckungen

Test	T1	T2	T3	T4	T2+T4	T1+T2+T4
score	2601	900	2600	2601		
alt	43	43	88	42		
ergebnis	30	5	15	20		
score > 2600	t	f	f	t	f t	t f
alt > 42	t	t	t	f	t f	t f
score > 900	t	f	t	t	f t	t f
alt > 42 && score >900	t	f	t	f	f	t f

C3	4/8	4/8	4/8	4/8	7/8	8/8
-----------	-----	-----	-----	-----	-----	-----

- Es geht auch mit zwei Testfällen!

Anmerkung: Diese Thema kann noch wesentlich vertieft werden



↓ vollständige Überdeckung des einen bedeutet vollständige Überdeckung des anderen

(Auch) C3 findet nicht alle Probleme

- Annahme: Methode mach liefert nur positive Ergebnisse

```
public int mach(int y, int z) {  
    if (y==0 & z==0) {  
        // t t  
        // f f  
        return 1;  
    } else {  
        return y*z;  
    }  
}
```

@Test

```
public void testTrueTrue(){  
    Assert.assertTrue(new Bsp().mach(0,0) > 0);  
}
```

@Test

```
public void testFalseFalse(){  
    Assert.assertTrue(new Bsp().mach(1,1) > 0);  
}
```

Abschlussbeispiel Ci-Überdeckungen

```
public int max(int x,  
               int y,  
               int z){
```

```
    int max = 0;
```

```
    if (x>z) {
```

```
        max = x;
```

```
    }
```

```
    if (y>x) {
```

```
        max = y;
```

```
    }
```

```
    if (z>y) {
```

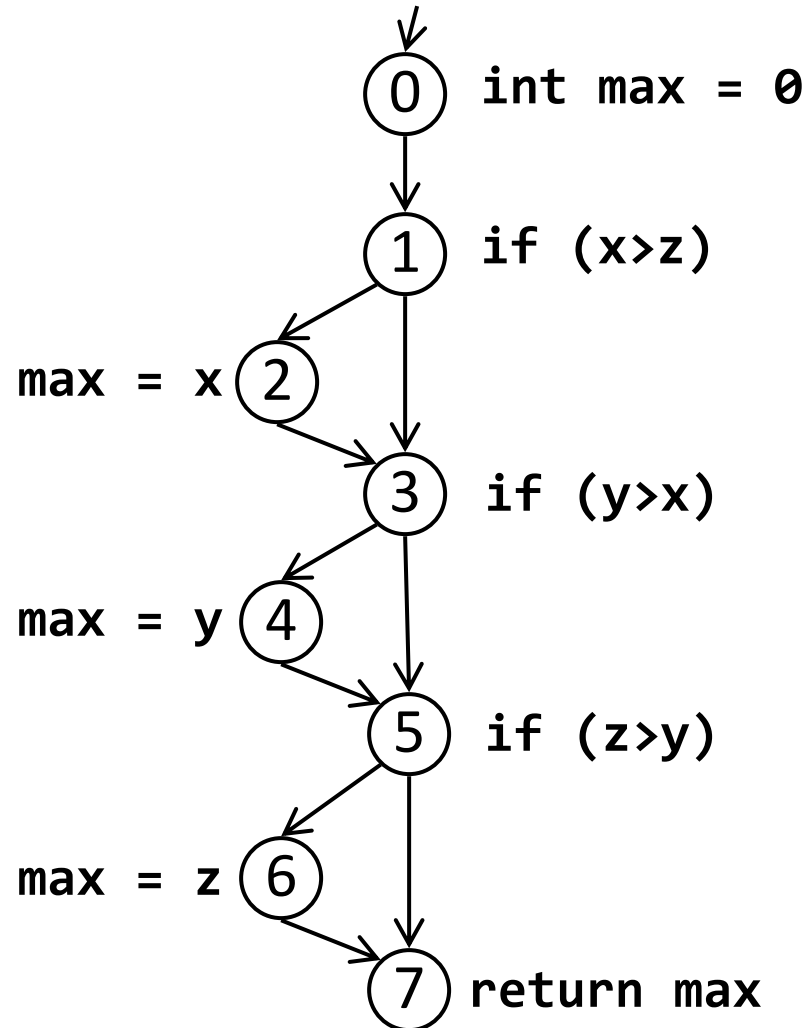
```
        max = z;
```

```
    }
```

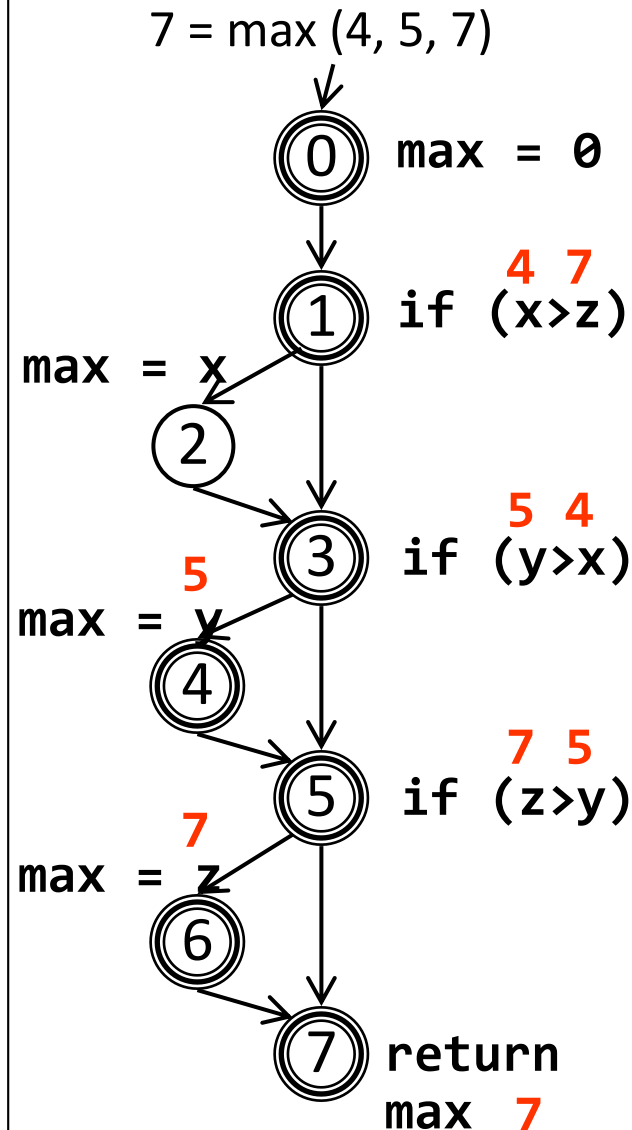
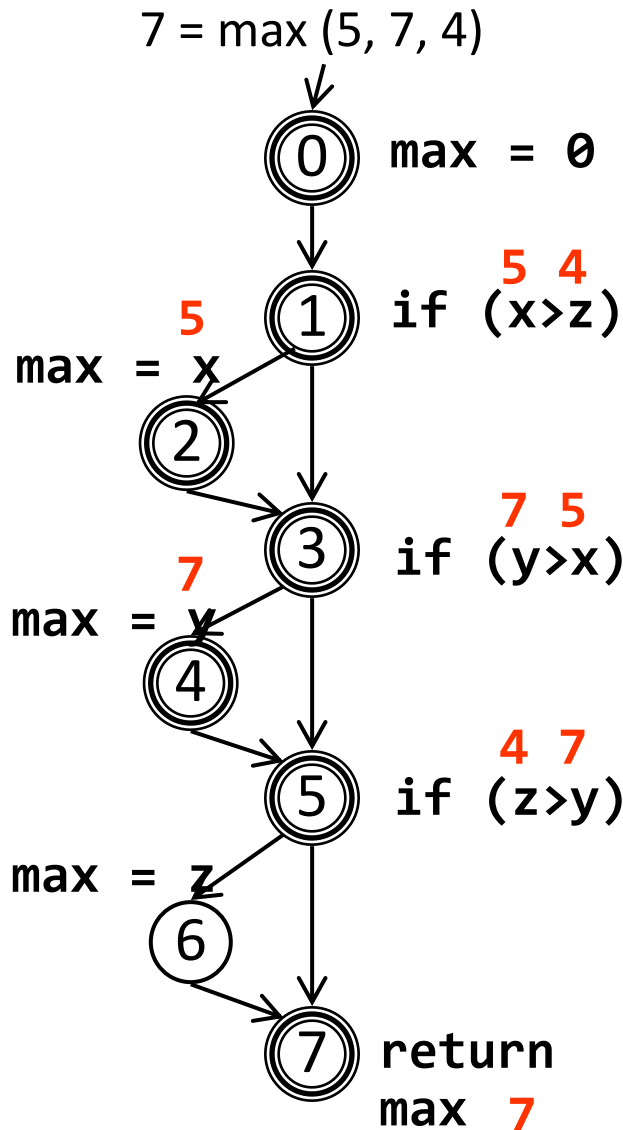
```
    return max;
```

```
 }
```

- Erinnerung: Suche Maximum von drei ganzen Zahlen



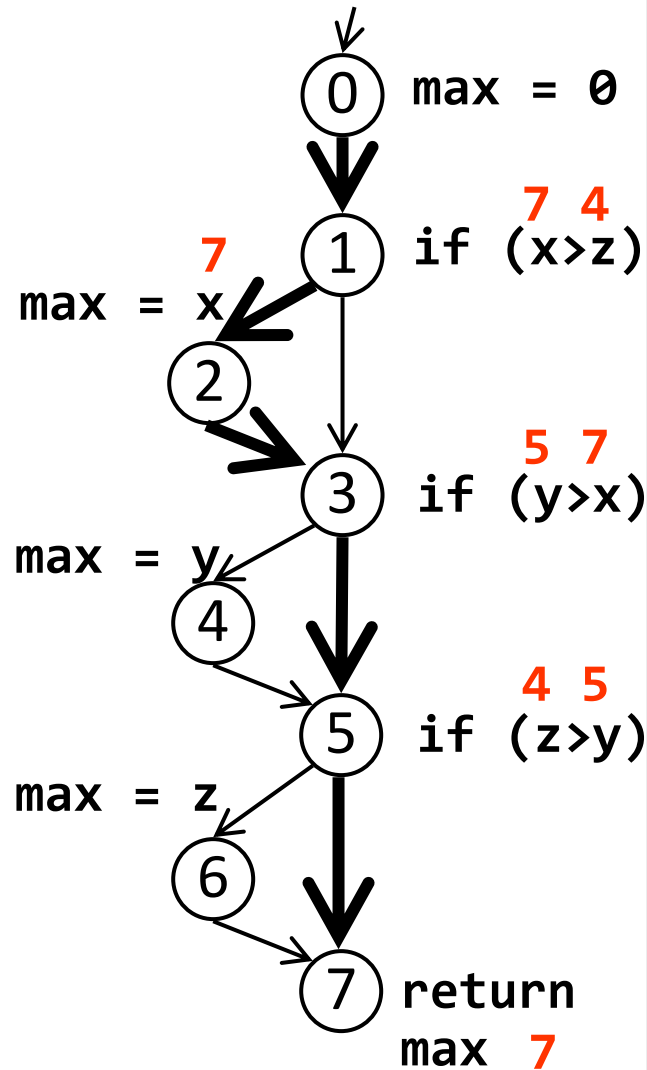
Anweisungsüberdeckung - jeder Knoten einmal



Zweigüberdeckung - jede Kante einmal

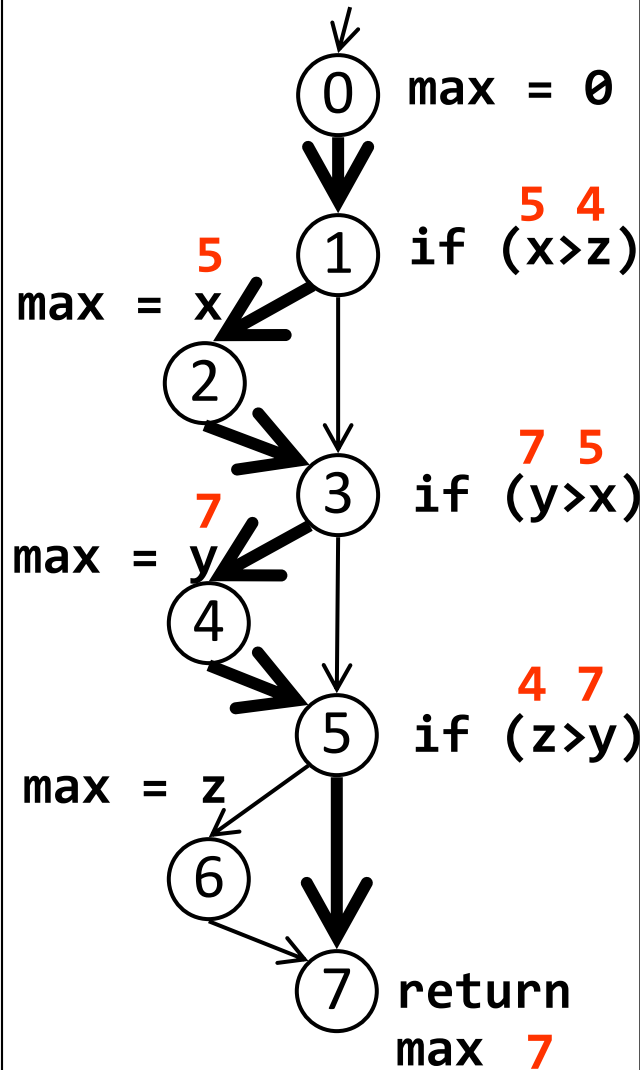


7 = max (7, 5, 4)



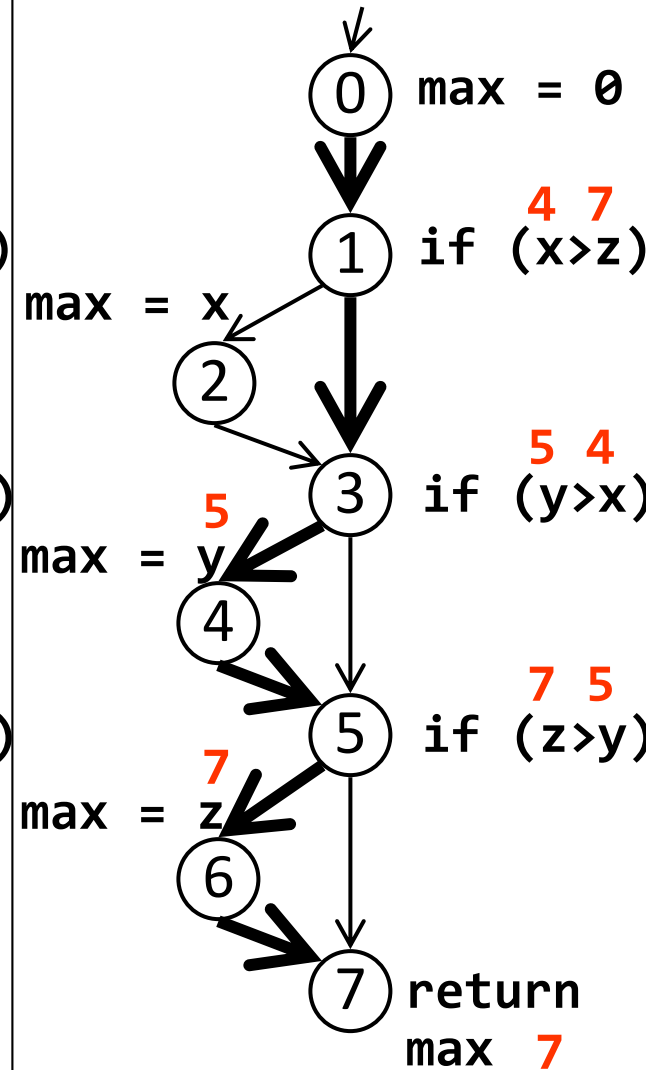
Software-Qualität

7 = max (5, 7, 4)



Stephan Kleuker

7 = max (4, 5, 7)



184

Fehler kann gefunden werden

- Kenntnisse über Äquivalenzklassen !

$x > y = z$ $y = z > x$ $y > x = z$ $x = z > y$ $z > y = x$ $y = x > z$

$z > y > x$ $z > x > y$ $y > z > x$ $y > x > z$ $x > z > y$ $x > y > z$ $x = y = z$

- Kenntnisse über Datenflussanalyse finden Fehler!
- benötigt große Programmiererfahrung
- benötigt sehr strukturiertes Denken
- benötigt Forderung nach intensiven Tests (auch DO-178C)
- benötigt Werkzeuge zur Automatisierung
- benötigt Zeit und Geld
- benötigt Management mit Software-Verstand

100% sinnlose Überdeckung?

- schnelle Überdeckung durch Test von get- und set-Methoden
- Beispiele zeigen deutlich, dass bei Überdeckung nicht beachtet wird, was in Assertions steht
- Assertions könnten weggelassen werden
- nur mit Kenntnis von Äquivalenzklassen und Grenzwerten können sinnvolle Assertions entstehen

Fazit: Äquivalenzklassen und Überdeckungen

- letztes Beispiel zeigt deutlich, dass man trotz systematischer Analyse Fehler übersehen kann !!!
- sinnvolles Vorgehen:
 1. Testfälle aus typischen Verhalten ableiten
 2. Testfälle aus Ausnahmesituationen ableiten
 3. Äquivalenzklassen überlegen und kritische Fälle systematisch analysieren
 4. Überdeckung messen, bei niedrigen Prozentzahlen zunächst das „warum“ ergründen, dann ggfls. Testfälle für Überdeckungen konstruieren

Nie, nie nur als zentrales Testziel 90 % - Überdeckung angeben, da dann Orientierung an Nutzerprozessen verloren geht (Überdeckung **notwendig, aber nicht hinreichend)**

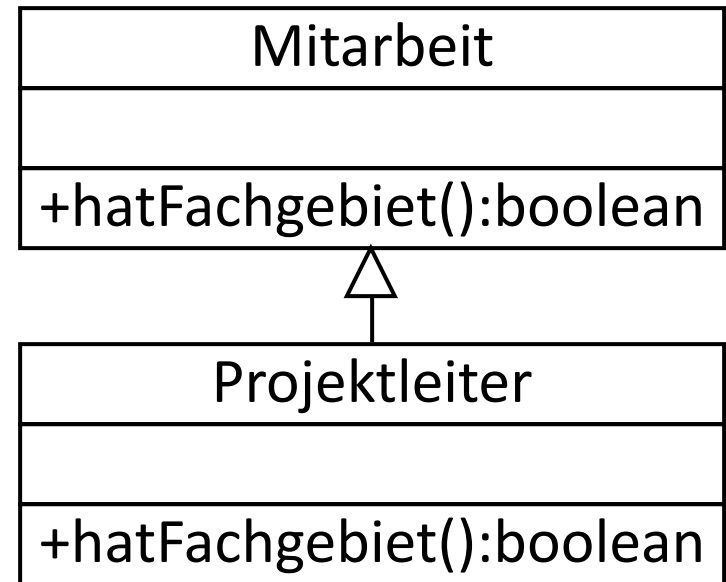
„Heimliche“ Verzweigungen (1/2)

- Vererbung ist eine mächtige Möglichkeit Doppelimplementierungen zu vermeiden
- Vererbung erlaubt dynamische Polymorphie (zur Laufzeit wird erst bestimmt, welche Methode aufgerufen wird)
- Prüfung zur Laufzeit entspricht heimlichen if mit Überprüfung durch `instanceof`
- Generell macht Polymorphie Entwicklung einfacher, wartbarer und flexibler; ist aber schwieriger zu testen als ohne
- Bei Testfallerstellung immer über durch Polymorphie entstehende Variationsmöglichkeiten nachdenken

„Heimliche“ Verzweigungen (2/2)

```
public int anzahl(Fachgebiet f){
    int ergebnis = 0;
    for(Mitarbeit m:mitarbeiten)
        if(m.hatFachgebiet(f))
            ergebnis++;
    return ergebnis;
}
```

```
public int anzahl(Fachgebiet f){
    int ergebnis = 0;
    for(Mitarbeit m:mitarbeiten)
        if(m instanceof Projektleiter){
            if (((Projektleiter)m).hatFachgebiet(f))
                ergebnis++;
        }
        else
            if(m.hatFachgebiet(f))
                ergebnis++;
    return ergebnis;
}
```



- Ohne eine Tool-Unterstützung ist es sehr aufwändig und fehlerträchtig solche Überprüfungen zu machen
- Für andere Sprachen als Java (und C#) sind Coverage-Werkzeuge vorhanden, die man bzgl. der zu untersuchenden Überdeckung einstellen kann
- Für Java gibt es Programme, die Anweisungsüberdeckungstest und teilweise Zweigüberdeckungstest (EclEmma, CodeCover, Covertura, JCoverage, Hansel für JUnit, Clover) unterstützen
- Wie muss so ein Werkzeug für Java arbeiten? Grundsätzlich muss der Code (Byte-Code) so modifiziert werden, dass Informationen über Abläufe gesammelt werden können
- Dieser Ansatz soll an einem einfachen Beispiel verdeutlicht werden (wobei hier direkt der Java-Code modifiziert wird)

Datenflussabhängige Fehler

```
public class Problem {  
    public static int mach(boolean a, boolean b){  
        int x=0;  
        if (a)  
            x=2;  
        else  
            x=3;  
        if (b)  
            return(6/(x-3));  
        else  
            return(6/(x-2));  
    }  
}
```

Beispielausführungen sorgen für
C1- und C3- Überdeckung
mach(true,false),
mach(false,true) führen aber zum
Abbruch

```
public static void main(String[] args) {  
    System.out.println(mach(true,true));  
    System.out.println(mach(false,false));  
}
```

-6
6

Ein Datenflussgraph ist ein erweiterter und markierter Kontrollflussgraph

- Kontrollflussgraph wird um einen neuen Startknoten *in* erweitert, der mit dem alten Startknoten verbunden ist
- Findet im Knoten die Definition einer Variablen x oder eine Zuweisung an x statt, wird dieser mit $def(x)$ markiert
- Enthält der Aufruf einer Methode eine Variable d , so wird der Knoten *in* mit $def(d)$ markiert
- Wird Variable y zur Berechnung in einer Zuweisung genutzt, wird Knoten mit $c-use(y)$ [computational use] markiert
- Wird in einem Knoten eine Variable z zur Berechnung in einer Fallunterscheidung genutzt, so werden die ausgehenden Kanten mit $p-use(z)$ [predicative use] markiert

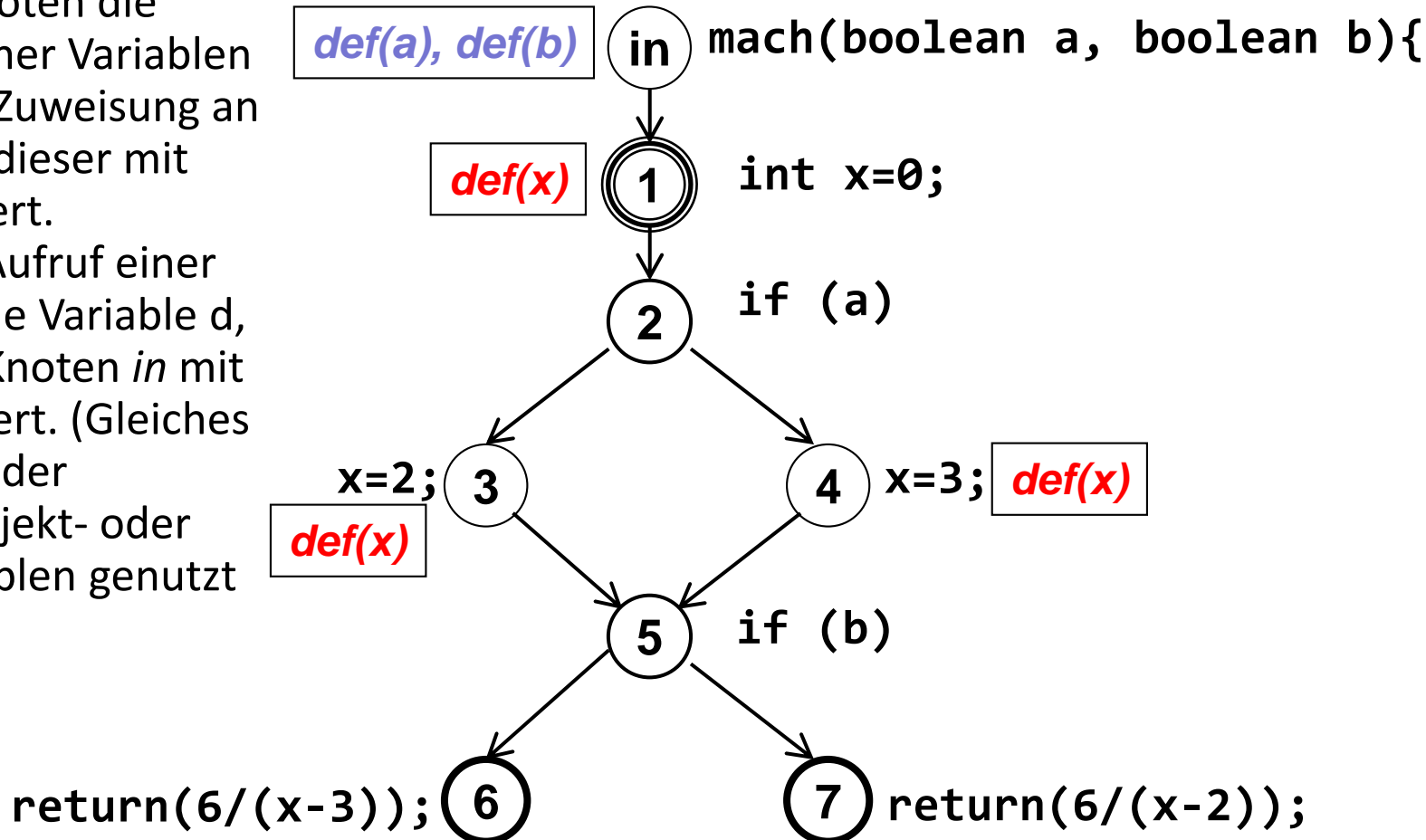
Hinweis: Kanten und Knoten erhalten Mengen von Markierungen

- **int x=0**
-> Knoten mit def(x) markiert
- **int y=x+1**
-> Knoten mit def(y) und c-use(x) markiert
- **z=f(a,b,c)**
-> Knoten mit def(z) und c-use(a), c-use(b) und c-use(c) markiert
- **i++**
-> Knoten wird mit def(i) und c-use(i) markiert
- **if (a>b)**
-> alle ausgehenden Kanten werden mit p-use(a) und p-use(b) markiert (in den meisten Definitionen wird auf eine Knotenmarkierung mit c-use(a) und c-use(b) verzichtet)

Beispiel: Erweiterter Kontrollflussgraph (1/3)

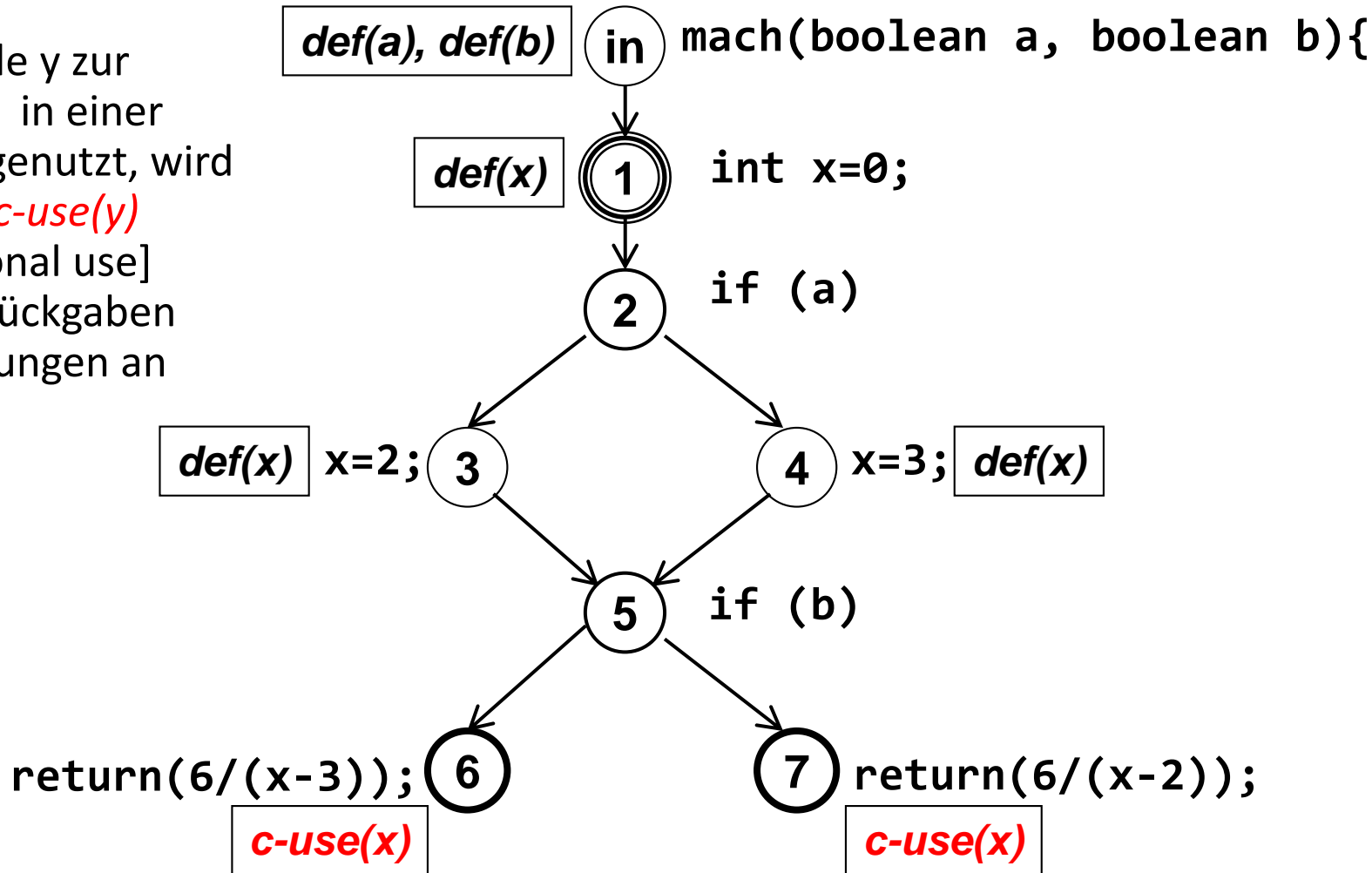
Findet im Knoten die Definition einer Variablen x oder eine Zuweisung an x statt, wird dieser mit $def(x)$ markiert.

Enthält der Aufruf einer Methode eine Variable d , so wird der Knoten in mit $def(d)$ markiert. (Gleiches gilt, wenn in der Methode Objekt- oder Klassenvariablen genutzt werden.)



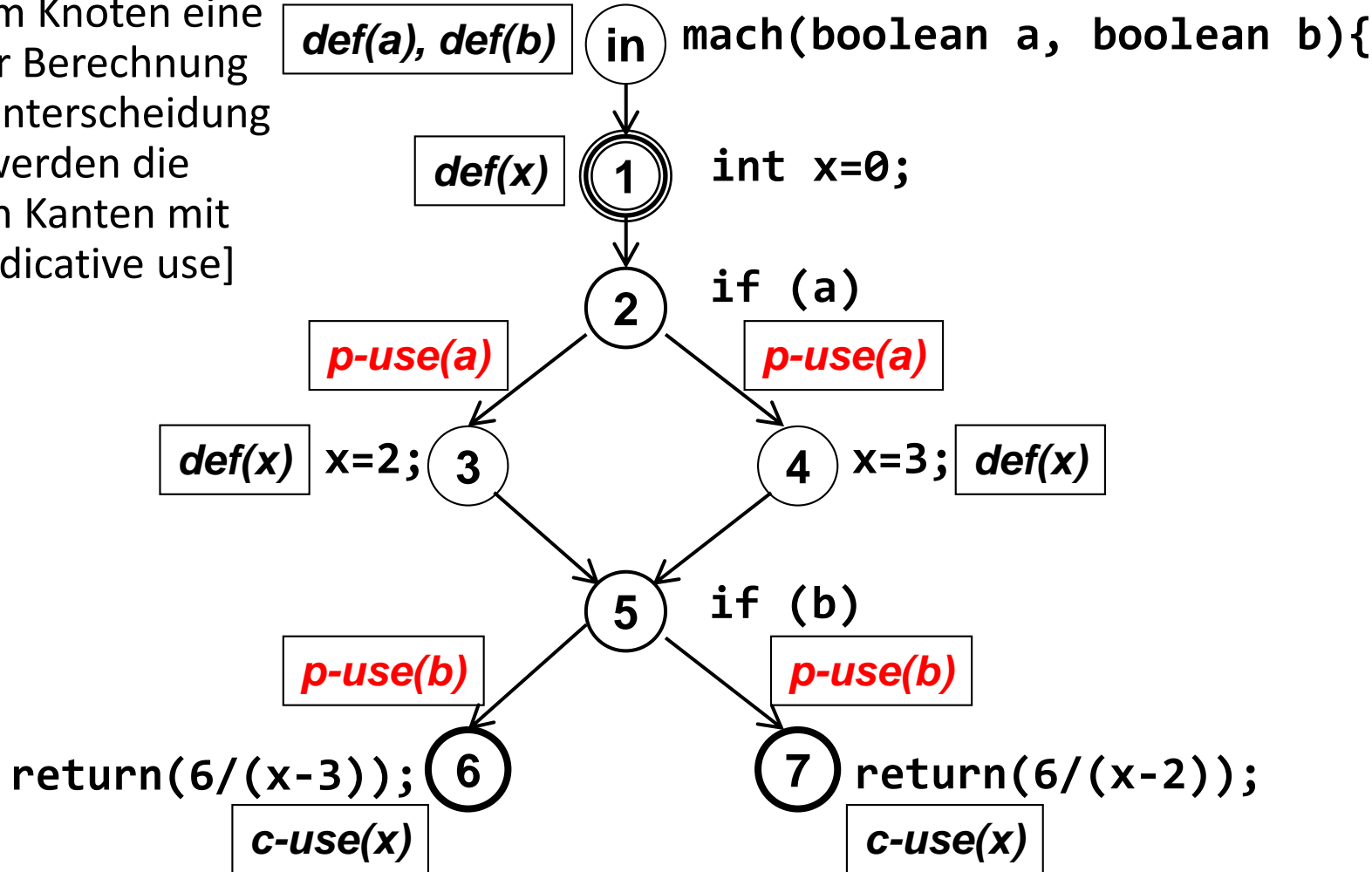
Beispiel: Erweiterter Kontrollflussgraph (2/3)

Wird Variable y zur Berechnung in einer Zuweisung genutzt, wird Knoten mit $c\text{-use}(y)$ [computational use] markiert (Rückgaben sind Zuweisungen an Aufrufer)



Beispiel: Erweiterter Kontrollflussgraph (3/3)

Wird in einem Knoten eine Variable z zur Berechnung in einer Fallunterscheidung genutzt, so werden die ausgehenden Kanten mit $p\text{-use}(z)$ [predicative use] markiert



Definitionsfreier Pfad bezüglich x

- Ein Pfad $p = (n_1, \dots, n_m)$, der nach n_1 keine Definition einer Variablen x in n_2, \dots, n_{m-1} enthält, heißt definitionsfrei bezüglich x

Anmerkung: Wenn in n_1 eine Definition von x stattfindet, hat diese Auswirkungen auf Nutzungen von x in Berechnungen auf diesem des Pfad

Anmerkung 2: Grundsätzlich dient diese und die folgenden Definitionen zur Untersuchung der Frage, an welchen Stellen haben Veränderungen von Variablen („Definitionen“) Auswirkungen

$dcu(x,ni)$

- Für einen Knoten ni und eine Variable x , wobei ni mit $def(x)$ markiert ist, ist $dcu(x,ni)$ die Menge aller Knoten nj , für die nj mit $c-use(x)$ markiert ist und für die *ein* definitionsfreier Pfad bezüglich x vom Knoten ni zum Knoten nj existiert

anschaulicher: in $dcu(x,ni)$ sind alle Knoten, die x nutzen und für die x seit ni eventuell nicht geändert wurde (zeigt den Auswirkungsbereich von $def(x)$)

$dpu(x,ni)$

- ist die Menge aller Kanten (nj, nk) , wobei ni mit $def(x)$ markiert ist, die mit $p-use(x)$ markiert sind und für die ein definitionsfreier Pfad bezüglich x von ni nach nj existiert

Hintergrund: Es sollen Testvollständigkeitskriterien auf der Basis von Variablenzugriffen definiert werden

Beispiel



$dcu(a, in) = \{\}$

$dcu(b, in) = \{\}$

$dpu(a, in) = \{(2, 3), (2, 4)\}$

$dpu(b, in) = \{(5, 6), (5, 7)\}$

$dcu(x, 1) = \{\}$

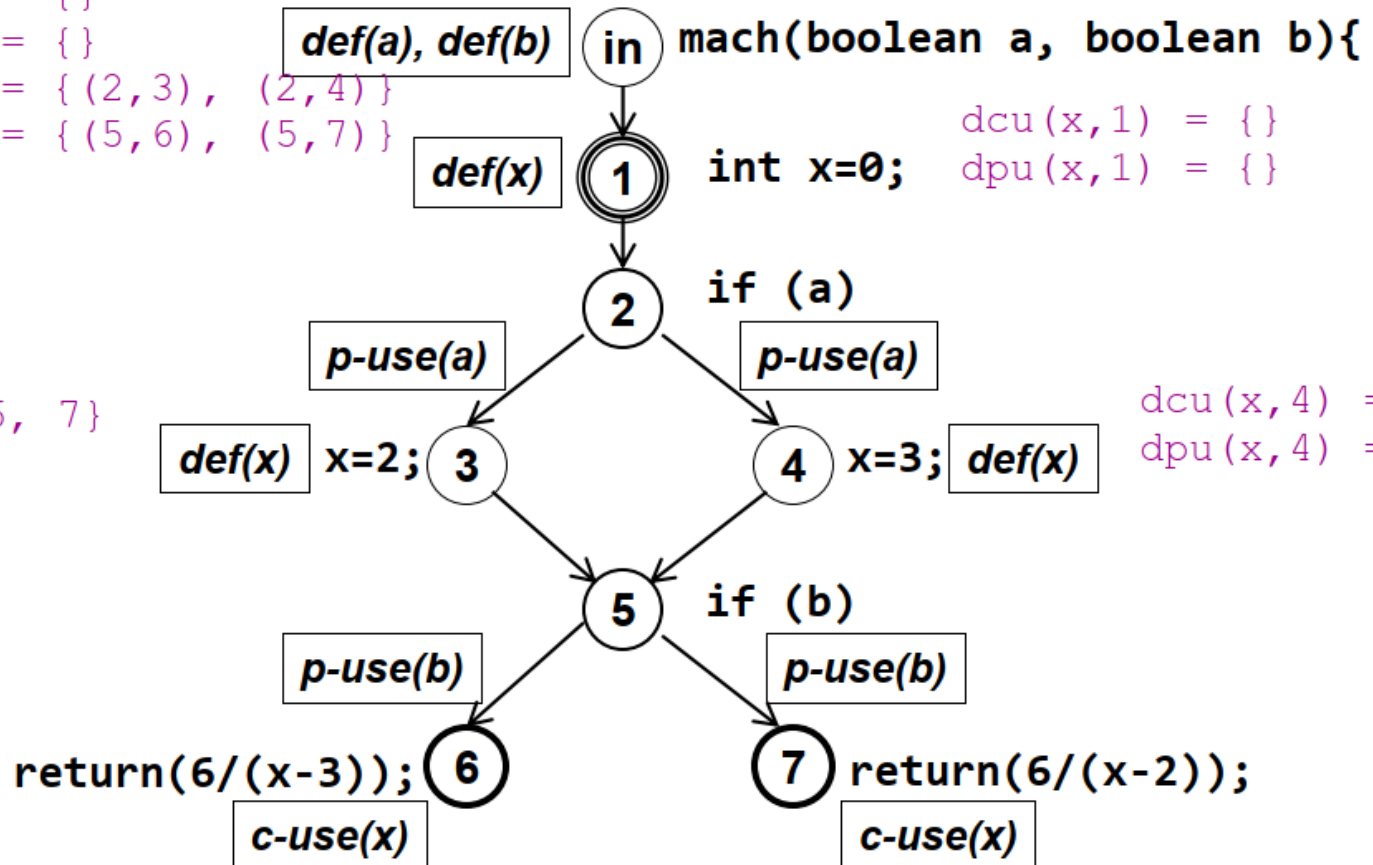
$dpu(x, 1) = \{\}$

$dcu(x, 3) = \{6, 7\}$

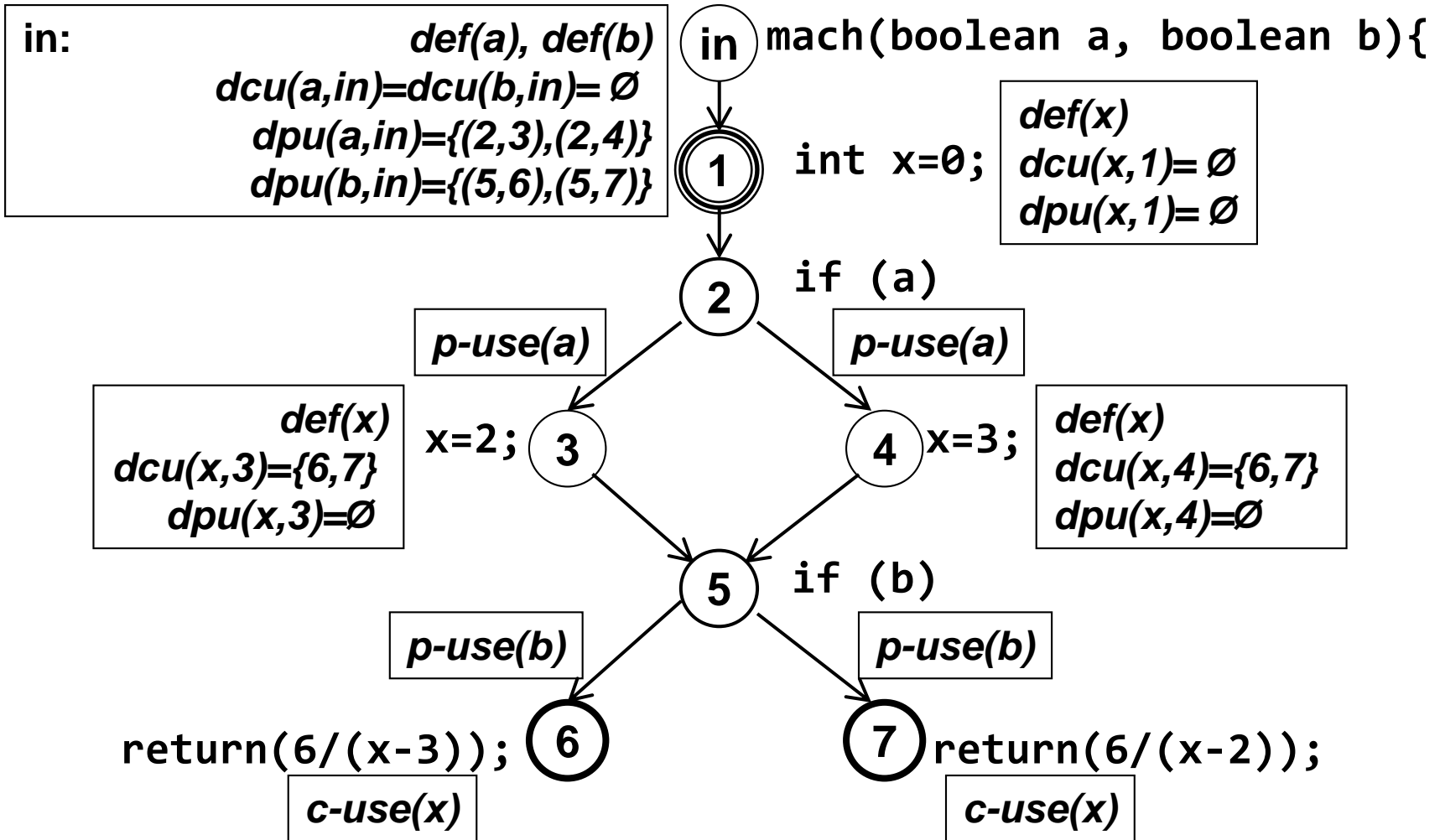
$dpu(x, 3) = \{\}$

$dcu(x, 4) = \{6, 7\}$

$dpu(x, 4) = \{\}$



Beispiel: dcu und dpu-Markierungen



Anmerkung: Markierungen werden auch von Compilern zur Optimierung genutzt

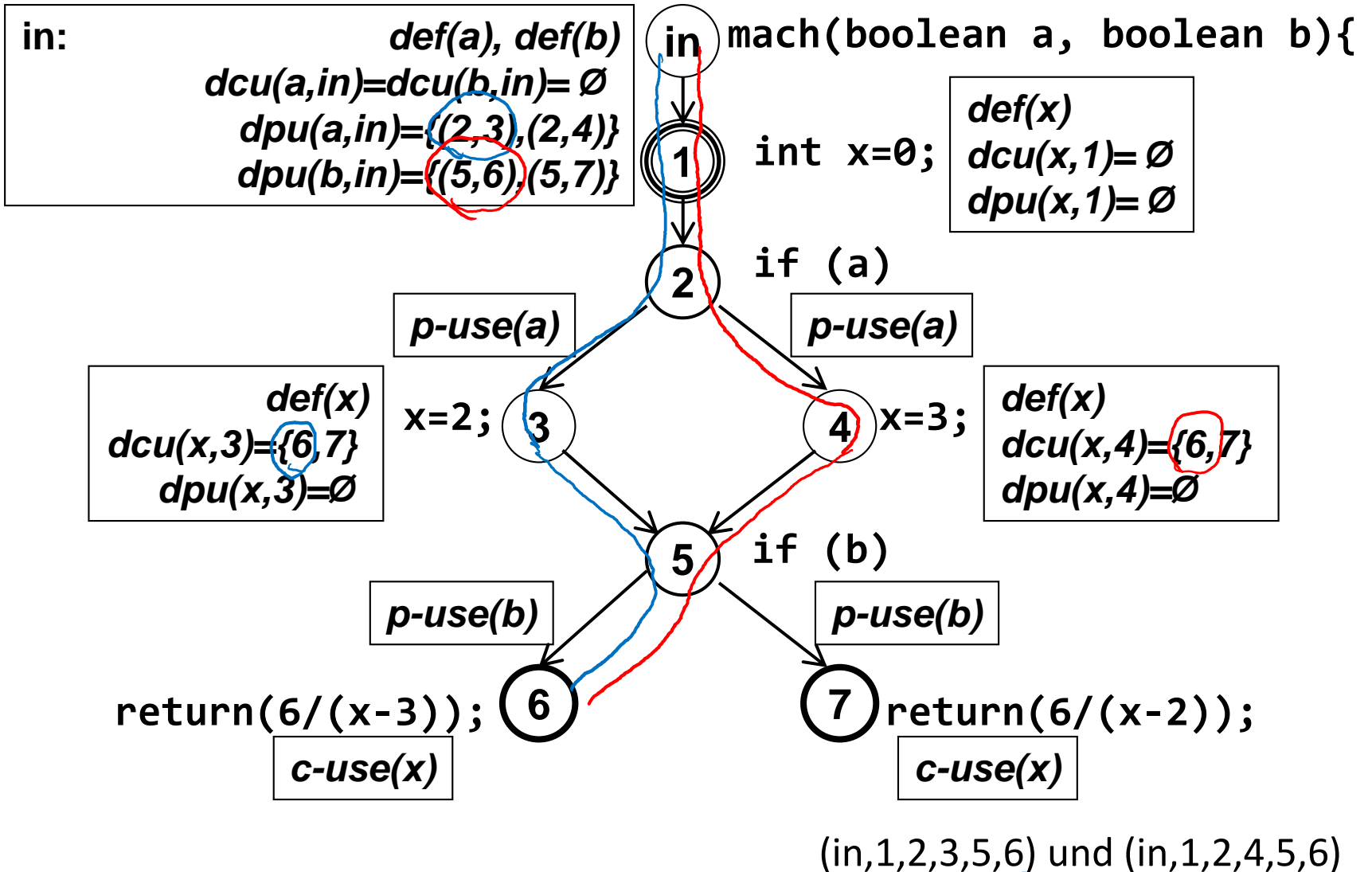
all defs-Kriterium

- jede Wertzuweisung soll mindestens einmal benutzt werden, d.h. für jeden Knoten n_i und jede Variable x , für die n_i mit $\text{def}(x)$ markiert ist, soll ein definitionsfreier Pfad zu einem Knoten aus $\text{dcu}(x, n_i)$ oder einer Kante aus $\text{dpu}(x, n_i)$ führen, insofern mindestens eine Menge nicht leer ist

(heißt: jede „Definition“ wird mindestens einmal genutzt)

- Beispiel: Eine Menge von Tests, die die Pfade $(in, 1, 2, 3, 5, 6)$ und $(in, 1, 2, 4, 5, 6)$ durchlaufen, erfüllt das all defs-Kriterium [ohne das Problem zu finden]
- all defs-Kriterium, subsumiert weder Zweig- noch Anweisungsüberdeckung
- all defs-Kriterium findet nebenbei überflüssige Zuweisungen an Variablen x , wenn $\text{dcu}(x, n_i) = \text{dpu}(x, n_i) = \emptyset$ gilt

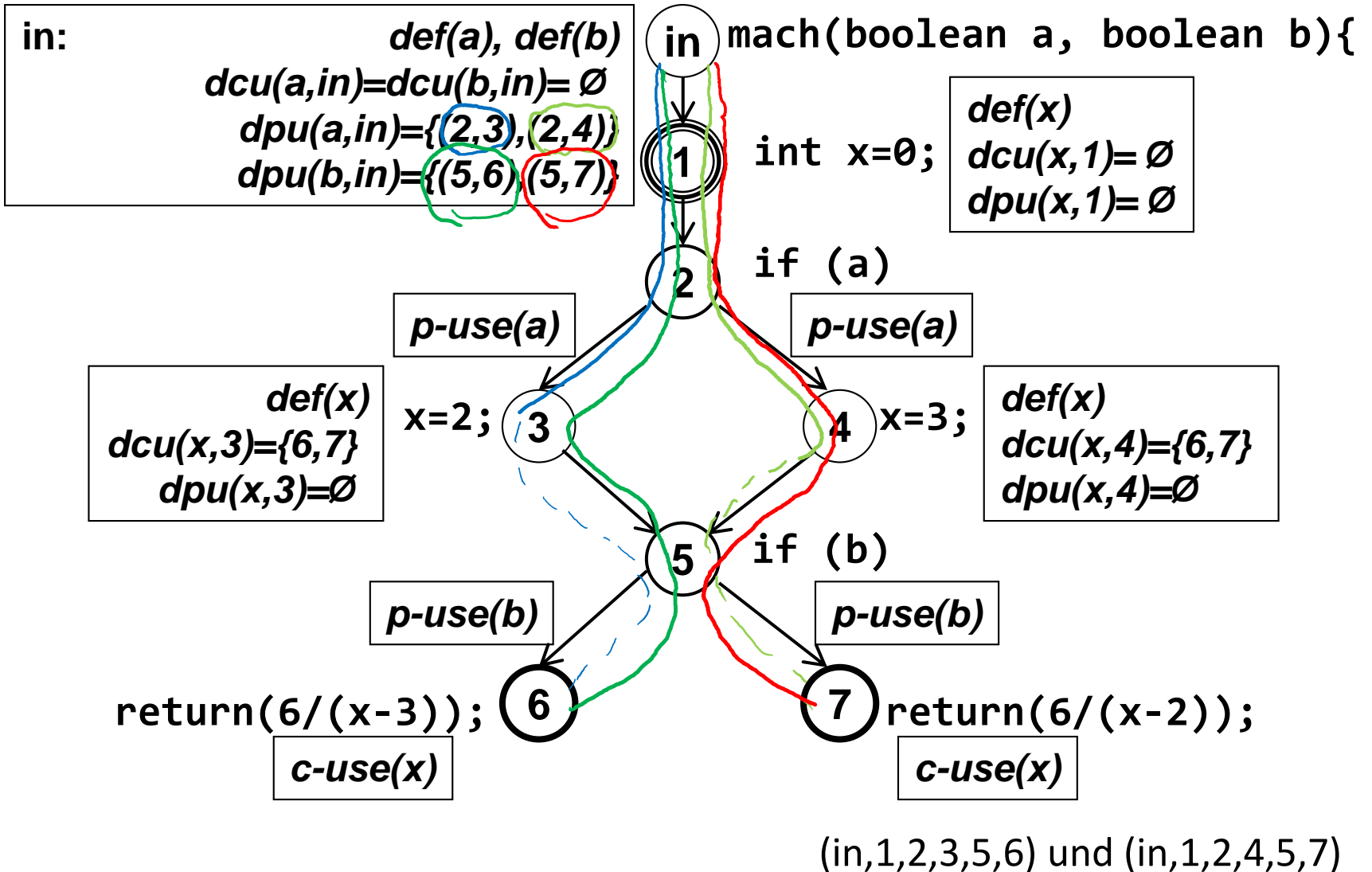
all defs: Jede def einmal genutzt



all p-uses-Kriterium

- für jeden Knoten n_i und jede Variable x , wobei n_i mit $\text{def}(x)$ markiert ist, muss ein definitionsfreier Pfad bezüglich x zu *allen* Elementen aus $\text{dpu}(x, n_i)$ in der Menge der getesteten Pfade enthalten sein
- Beispiel: Eine Menge von Tests, die die Pfade $(\text{in}, 1, 2, 3, 5, 6)$ und $(\text{in}, 1, 2, 4, 5, 7)$ durchlaufen, erfüllt das all p-uses-Kriterium [ohne das Problem zu finden]
- all p-uses-Kriterium, subsumiert die Zweig- und damit auch die Anweisungsüberdeckung

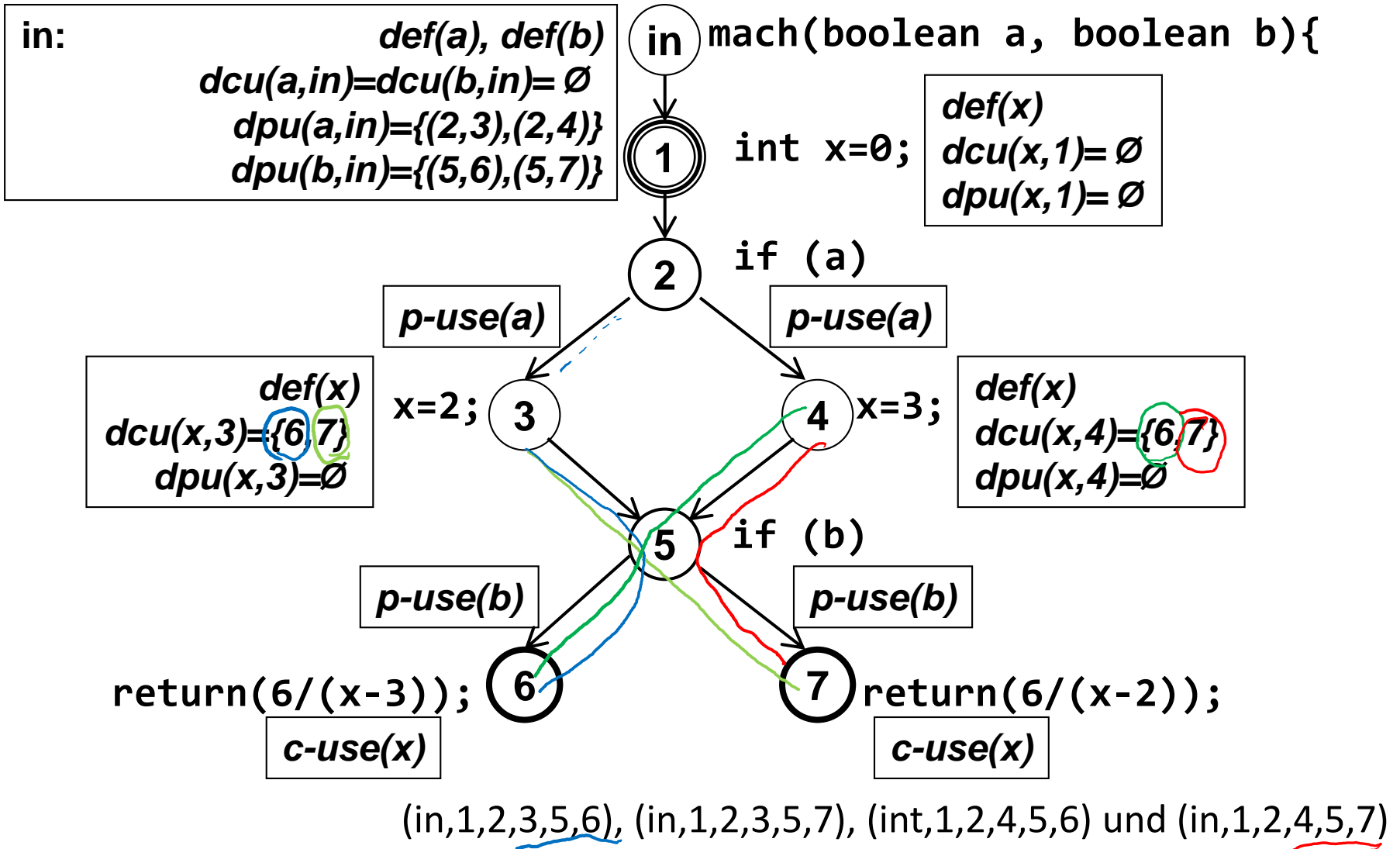
all p-uses: Alle p-use genutzt



all c-uses-Kriterium

- für jeden Knoten n_i und jede Variable x , wobei n_i mit $\text{def}(x)$ markiert ist, muss ein definitionsfreier Pfad bezüglich x zu *allen* Elementen aus $\text{dcu}(x, n_i)$ in der Menge der getesteten Pfade enthalten sein
- Beispiel: Eine Menge von Tests, die die Pfade $(\text{in}, 1, 2, 3, 5, 6)$, $(\text{in}, 1, 2, 3, 5, 7)$, $(\text{in}, 1, 2, 4, 5, 6)$ und $(\text{in}, 1, 2, 4, 5, 7)$ durchlaufen, erfüllt das all p-uses-Kriterium [und findet das Problem]
- all c-uses-Kriterium, subsumiert weder Zweig- noch Anweisungsüberdeckung

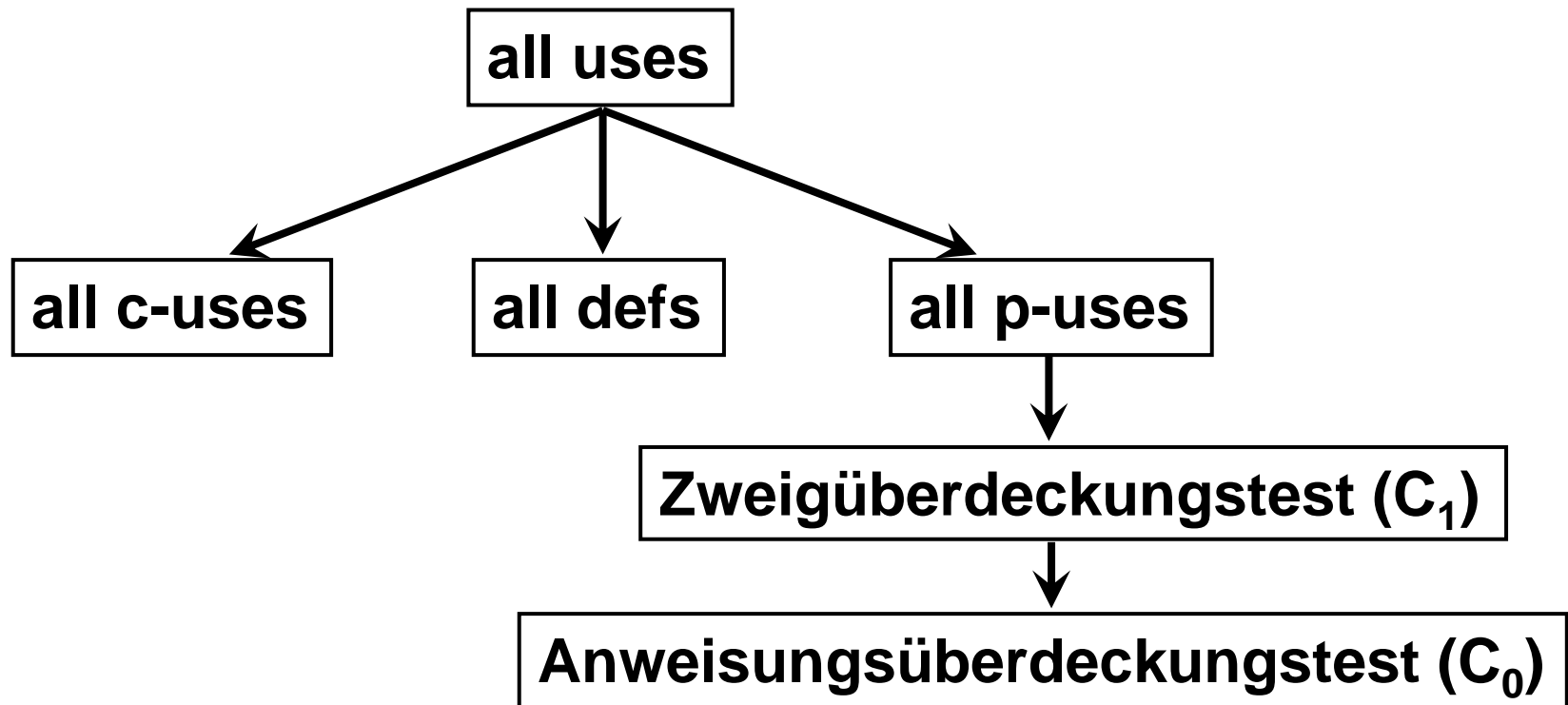
all c-uses: Alle c-use genutzt



all uses-Kriterium

- für jeden Knoten n_i und jede Variable x , wobei n_i mit $\text{def}(x)$ markiert ist, muss ein definitionsfreier Pfad bezüglich x zu *allen* Elementen aus $\text{dcu}(x, n_i)$ und $\text{dpu}(x, n_i)$ in der Menge der getesteten Pfade enthalten sein
- „all uses = all c-uses + all p-uses“
- Beispiel: Eine Menge von Tests, die die Pfade $(\text{in}, 1, 2, 3, 5, 6)$, $(\text{in}, 1, 2, 3, 5, 7)$, $(\text{in}, 1, 2, 4, 5, 6)$ und $(\text{in}, 1, 2, 4, 5, 7)$ durchlaufen, erfüllt das all uses-Kriterium [und findet das Problem]
- all uses-Kriterium, subsumiert die Zweig- und damit auch die Anweisungsüberdeckung

Testhierarchie



Anmerkung: es gibt weitere Überdeckungsansätze, die auf diesen Überlegungen basieren

Video 6

- Idee gute Tests sollten garantiert Fehler im Code finden
- Ansatz: Nehme den zu testenden Code und ändere ihn ab (Mutation), gute Testfälle sollten einen Fehler melden.
- Beispiele für Mutationen (Miniausschitt):
 - negiere Boolesche Bedingung
 - ersetze + durch -, <= durch <, >= durch >
 - ersetze int-Werte durch 0
- Vorteil: präzises Nachdenken über Assertions
- Nachteil: sehr, sehr zeitaufwändig (Grenze für Anzahl setzen)
- Nachteil: „überlebende Mutation“ muss auf kein Problem hindeuten (muss immer geprüft werden)
- Beispielwerkzeug Pitest: <https://pitest.org/>

weiterer Ansatz: Property Based Testing

- Ansatz: Spezifiziere Eigenschaften mit Parametern, die vom Computer überprüft werden
- Eigenschaft kann z. B. als Boolesche Methode spezifizierbar
- genauer: Computer rät für Parameter unterschiedliche Werte, führt Methode aus, meldet Fehler, wenn Ergebnis false
- zum Raten der Parameter werden intern Generatoren genutzt, die auch Extremwerte berücksichtigen
- bei Fehlern kann zur Optimierung nach einfacherem Beispiel gesucht werden, z. B. kürzerem String

- Beispielumsetzung: jqwik <https://jqwik.net/>
- (gleich nur Minibeispiel, bietet sehr viel mehr)
- wieder Frage des Aufwands

Property Based Testing (1/2) – Beispiel

- geforderte Eigenschaft, `summe()` nie größer 42

```
public class ErstesBeispielTest {  
  
    @Property(tries = 30) // default 1000  
    public boolean kleiner42(  
        @ForAll @IntRange(min = -50, max = 20) int anInt1  
        , @ForAll @IntRange(min = -50, max = 22) int anInt2) {  
        return this.summe(anInt1, anInt2) < 42;  
    }  
  
    public int summe(int s1, int s2) {  
        return s1 + s2;  
    }  
} // minimal benötigt
```

- kann mit JUnit 5 laufen (geht auch ohne)

Property Based Testing (2/2) – Ergebnis

ErstesBeispielTest:kleiner42 = org.opentest4j.AssertionFailedError:
Property [ErstesBeispielTest:kleiner42] failed with sample {0=20, 1=22}

tries = 28
checks = 28
generation = RANDOMIZED
after-failure = SAMPLE_FIRST
previous seed
when-fixed-seed = ALLOW
edge-cases#mode = MIXIN
edge-cases#total = 36
edge-cases#tried = 6
seed = 7377882088781133675

-----jqwik-----
of calls to property
of not rejected calls
parameters are randomly generated
try previously failed sample, then

fixing the random seed is allowed
edge cases are mixed in
of all combined edge cases
of edge cases tried in current run
random seed to reproduce generated values

Sample

arg0: 20
arg1: 22

Property Based Testing (1/2) – Beispiel2

```
@Property(tries = 30) // tries = 3 findet Fehler evtl. nicht
public boolean maxCheck(
    @ForAll int i1, @ForAll int i2, @ForAll int i3) {
    int max = this.max(i1, i2, i3);
    return max >= i1 && max >= i2 && max >= i3;
}
```

```
private int max(int x, int y, int z) { // Variante
    int erg = z;
    if (x > z) {
        erg = x;
    }
    if (y > x) {
        erg = y;
    }
    if (z > y) {
        erg = z;
    }
    return erg;
}
```

Property Based Testing (2/2) – Ergebnis2

ErstesBeispielTest:maxCheck = org.opentest4j.AssertionFailedError:
Property [ErstesBeispielTest:maxCheck] failed with sample {0=2, 1=0, 2=1}

tries = 3
checks = 3
edge-cases#total = 36
edge-cases#tried = 0
seed = 1105666002029128743

-----jqwik-----
of calls to property
of not rejected calls
of all combined edge cases
of edge cases tried in current run
random seed to reproduce generated values

Shrunk Sample (6 steps)

arg0: 2
arg1: 0
arg2: 1

Original Sample

arg0: 13857
arg1: -737077082
arg2: 40

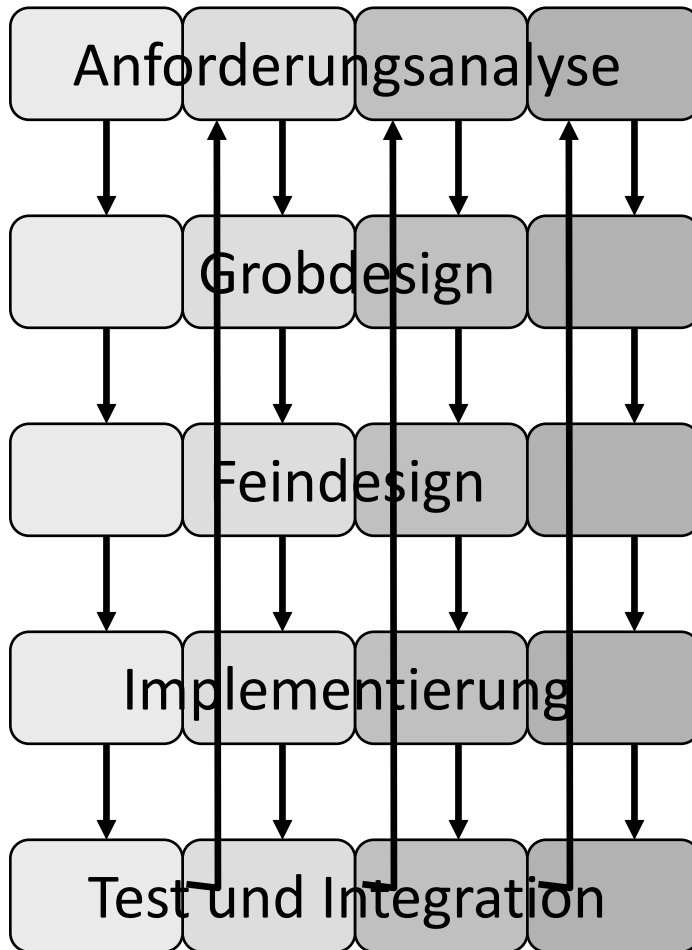
- Zertifizierung Federal Aviation Administration (FAA), Software für Luftverkehrssysteme durch Standard DO-178C für requirement-based Tests and Code Coverage Analyse
- DO-178C-Levels orientieren sich an den Konsequenzen möglicher Softwarefehler: katastrophal (Level A), gefährlich/schwerwiegend (Level B), erheblich (Level C), geringfügig (Level D) bzw. keine Auswirkungen (Level E)
- Je nach DO-178C-Level wird der 100%-ige Nachweis folgender Testabdeckungen (Code Coverages) verlangt:
- DO-178C Level A:
 - Modified Condition Decision Coverage (MC/DC)
 - Branch/Decision Coverage
 - Statement Coverage
- DO-178C Level B:
 - Branch/Decision Coverage
 - Statement Coverage

5. Vorgehensmodelle und Testen



- Vorgehensmodelle (Konzept)
- Testarten
- Testgetriebene Entwicklung
- Behaviour Driven Development (BDD) mit Cucumber

Erinnerung: Skizze eines Vorgehensmodells



Bsp.: vier Inkremente

Software-Qualität

Merkmale:

Projekt in kleine Teilschritte zerlegt,
n+1-ter Schritt kann Probleme des n-
ten Schritts lösen

Vorteile:

- dynamische Reaktion auf Risiken
- Teilergebnisse mit Kunden diskutierbar

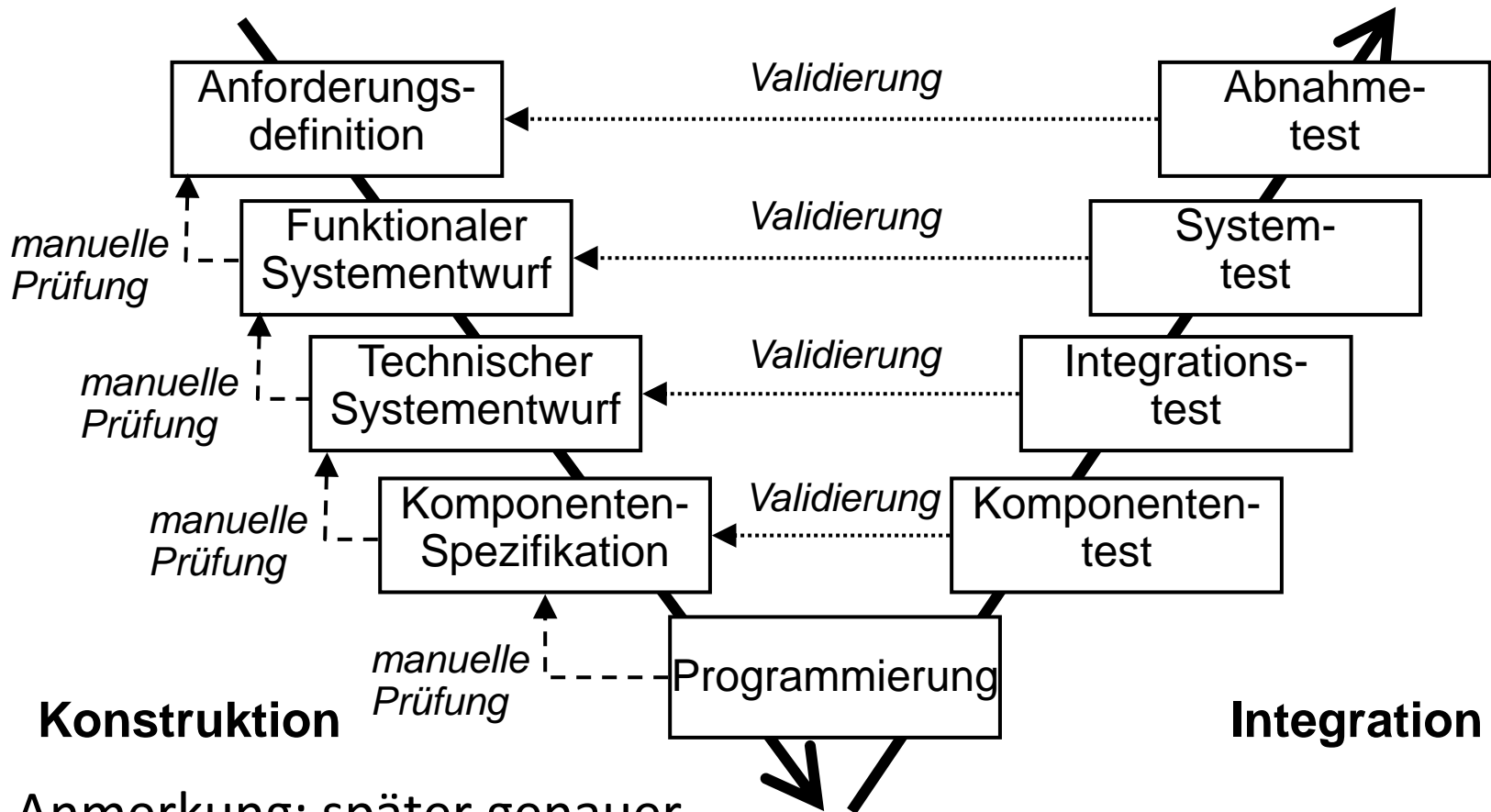
mögliche Nachteile:

- schwierige Projektplanung
- schwierige Vertragssituation
- Kunde erwartet zu schnell Endergebnis

- generelle Thematiken überall identisch:
Verstehen der Aufgabenstellung, Umsetzung, Überprüfung
- Unterschiede, wann, wie oft, welche Korrekturmöglichkeiten
- Unterschiede, wie Phasen/Schritte bearbeitet werden
- Unterschiede, welche Dokumentationsart genutzt wird

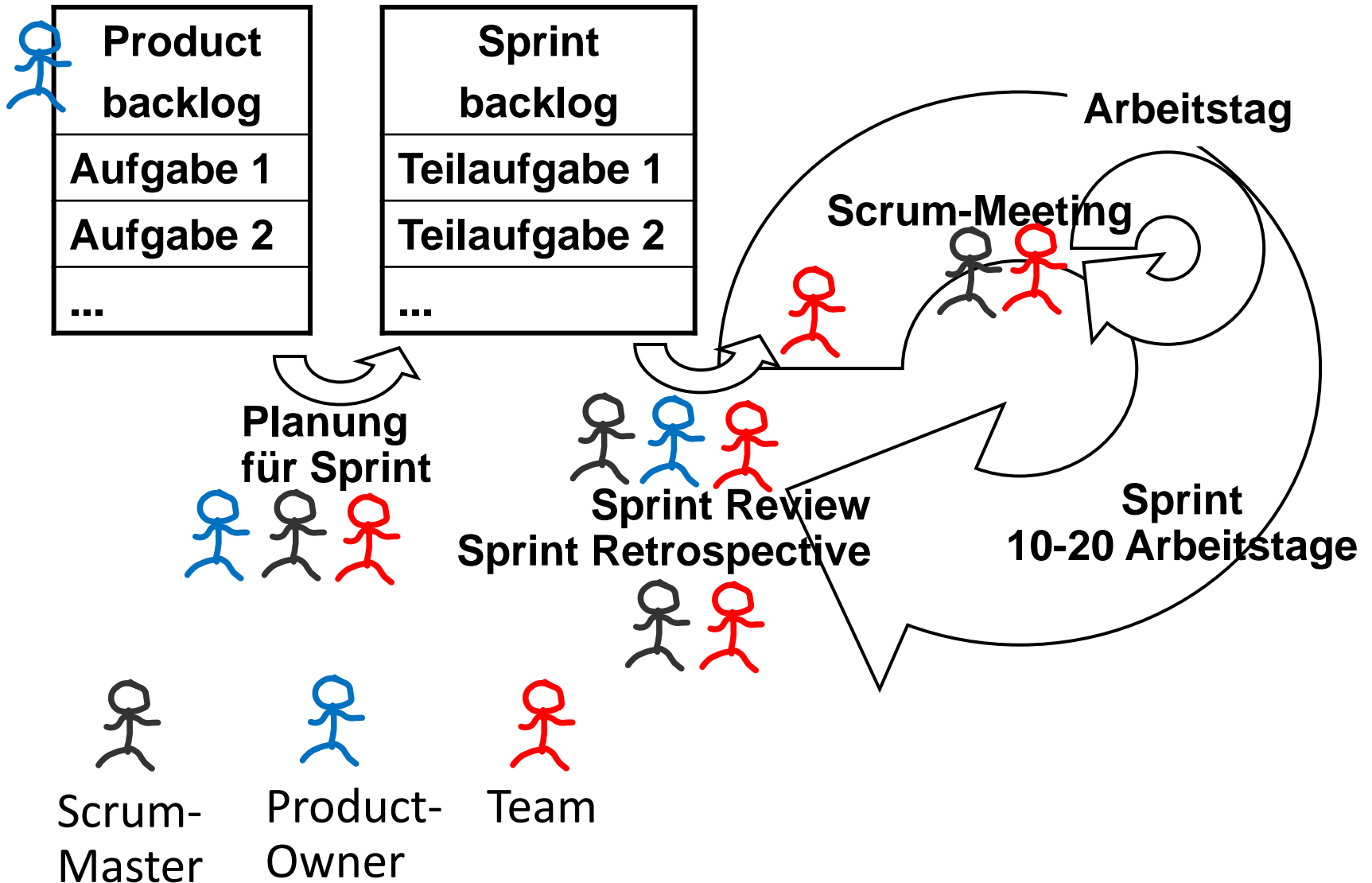
- Üblich: Testfallspezifikationen entstehen parallel zur Entwicklung
 - Beschreibung, was das System machen soll -> Testfälle die das Systemverhalten überprüfen
 - Beschreibung, was die Klasse machen soll -> Testfälle die das Klassenverhalten überprüfen
- offen: erst Tests entwickeln oder erst Code

Zusammenspiel von Test und Entwicklung (V-Modell)

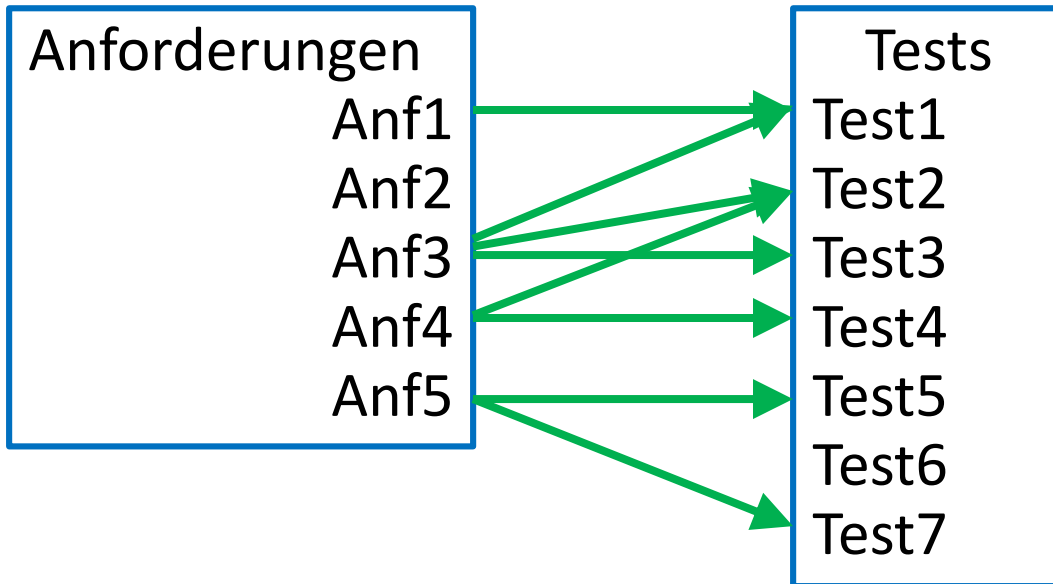


- Anmerkung: später genauer
- Anmerkung: Begriff „Komponente“ missverständlich, da für „Unit“ (= Klasse, Modul) und „Component“ (= SW-Komponente) genutzt

Agile Methoden – Beispiel Scrum



- jeder Test basiert auf einer dokumentierten Anforderung
- jede Anforderung ist so zu schreiben, dass sie testbar ist
- zentrale Nutzung von Testverwaltungswerkzeugen in Software-Entwicklungsumgebungen
- zentraler Begriff: Tracing (Nachverfolgung)
- Werkzeug beantwortet unter anderem Fragen:
 - warum gibt es diesen Test? (Antwort 1:n Anforderungen)
 - wo wird diese Anforderung getestet (Antwort: 1:n Tests)
 - suche alle Tests, die zu keinen Anforderungen gehören
 - suche alle Anforderungen, zu denen es keine Tests gibt



	T1	T2	T3	T4	T5	T6	T7
A1	100						
A2							
A3	30	20	50				
A4		20		80			
A5					40		40

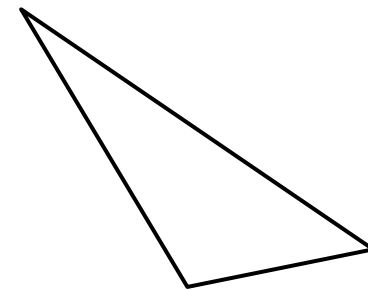
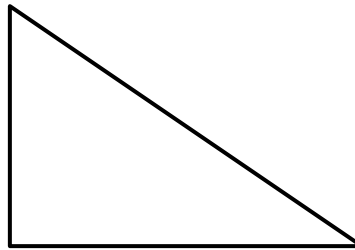
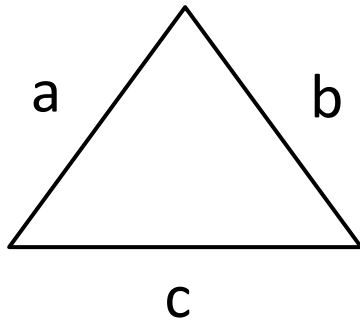
- Tracing kann weitere Ebenen haben (z. B. vom Use Case zur textuellen Anforderung)
- Tracing zwischen Anforderungen und Quellcode sowie Quellcode und Tests sinnvoll

Testgetriebene Entwicklung (TDD)

- Entwicklungsansatz zur Programmierung
- vor der eigentlichen Programmierung werden Tests geschrieben, die zu entwickelndes Teilprogramm prüfen
- Ergebnis: JUnit-Tests für anstehende Programmier-Teilaufgabe
- dann wird Funktionalität inkrementell entwickelt
- nach jedem Inkrement wird getestet; die Anzahl erfolgreicher Testfälle wächst von null bis zu 100%
- Vorteil: reiner Fokus darauf, was gemacht werden soll; nicht auf die Umsetzung
- Vorteil: spätere Testerstellung wäre sonst zu stark von Entwicklung beeinflusst
- Anmerkung: hilfreich ist Programmierung gegen Interfaces, da so Tests frühzeitig lauffähig

Mini-Beispiel (1/6): informelle Spezifikation

- gesucht ist eine Methode, die drei Integer-Werte übergeben bekommt und ausgibt, ob sich aus den mit den Werten gegebenen Seitenlängen ein Dreieck konstruieren lässt



Mini-Beispiel (2/6): formellere Spezifikation

```
package business;
```

```
public interface AbstractFormpruefer {
```

```
    public boolean istDreieckMoeglich(int a, int b, int c);
```

```
}
```

Mini-Beispiel (3/6): Tests (1/2)

```
public class FormprueferTest {  
    private AbstractFormpruefer fp;  
    @BeforeEach  
    public void setUp() {  
        //TODO  
    }  
    @Test  
    public void gleichseitig(){  
        Assertions.assertTrue(fp.istDreieckMoeglich(4, 4, 7));  
    }  
    @Test  
    public void gleichschenkelig(){  
        Assertions.assertTrue(fp.istDreieckMoeglich(4, 4, 4));  
    }  
    @Test  
    public void beliebigKorrektesDreieck(){  
        Assertions.assertTrue(fp.istDreieckMoeglich(4, 6, 9));  
    }  
}
```

Mini-Beispiel (4/6): Tests (2/2)

```
@Test
public void illegalerErsterParameter(){
    Assertions.assertFalse(fp.istDreieckMoeglich(0, 4, 4));
}

@Test
public void illegalerZweiterParameter(){
    Assertions.assertFalse(fp.istDreieckMoeglich(4, 0, 4));
}

@Test
public void illegalerDritterParameter(){
    Assertions.assertFalse(fp.istDreieckMoeglich(4, 4, 0));
}

@Test
public void nichtKonstruierbaresDreieck(){
    Assertions.assertFalse(fp.istDreieckMoeglich(5, 12, 7));
}
}
```

Mini-Beispiel (5/6): erstes Inkrement

- Ziel nur Tests lauffähig bekommen

```
public class Formpruefer implements AbstractFormpruefer{
```

```
    @Override
```

```
    public boolean istDreieckMoeglich(int a, int b, int c) {  
        throw new UnsupportedOperationException(  
            "Not supported yet.");  
    }
```

```
}
```

```
}
```

```
@BeforeEach // in FormprueferTest
```

```
public void setUp() {  
    this.fp = new Formpruefer();  
}
```

```
}
```

Tests passed: 0.00 %

No test passed, 7 tests caused an error.(0.165 s)

  test.business.FormprueferTest Failed

Mini-Beispiel (6/6): zweites Inkrement

@Override

```
public boolean istDreieckMoeglich(int a, int b, int c) {  
    return (a + b > c) && (a + c > b) && (b + c > a);  
}
```

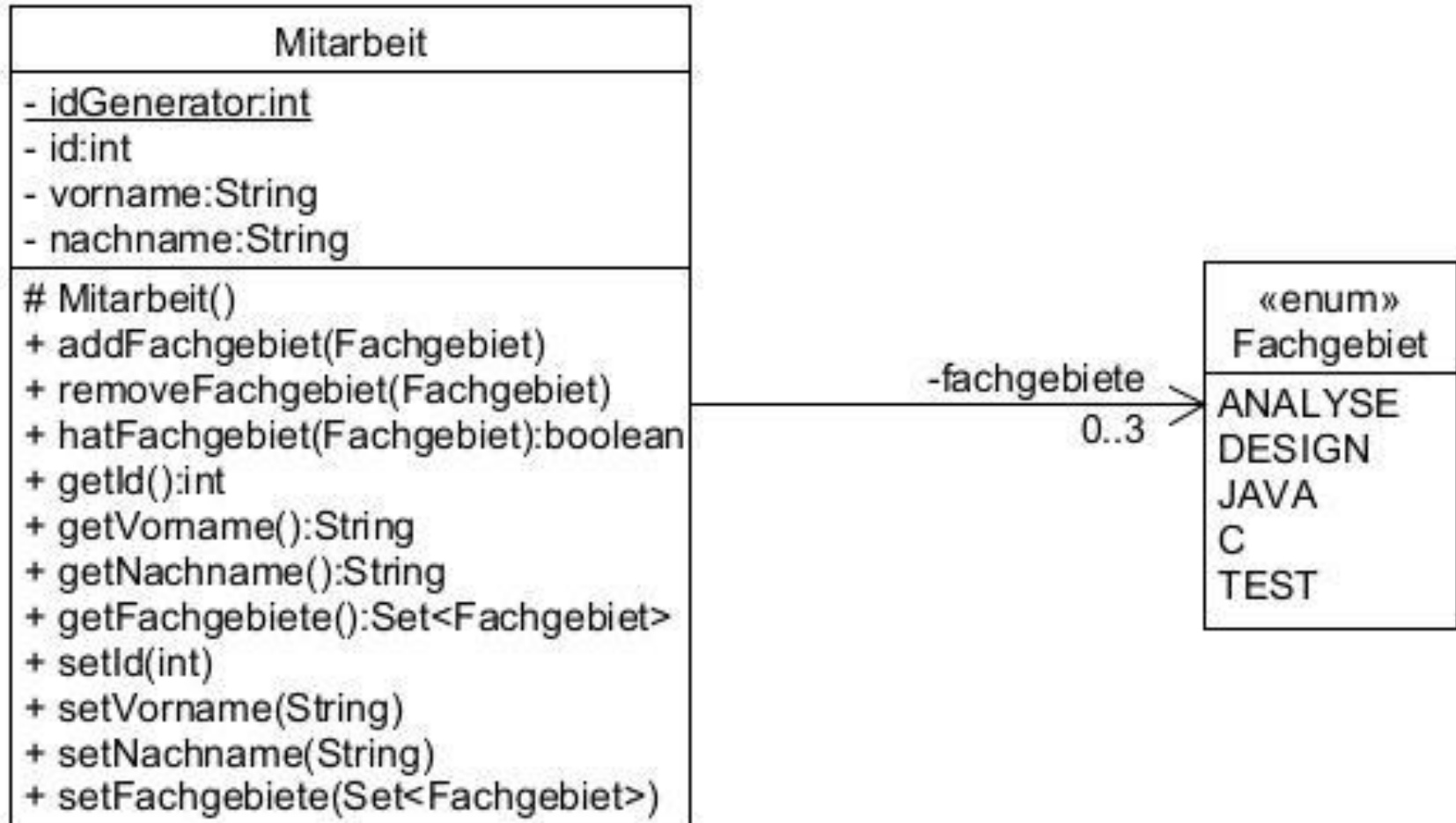
Tests passed: 100.00 %

All 7 tests passed.(0.121 s)

- test.business.FormprueferTest passed
 - gleichseitig passed (0.008 s)
 - illegalerDritterParameter passed (0.0 s)
 - illegalerErsterParameter passed (0.001 s)
 - beliebigKorrektedreieck passed (0.0 s)
 - nichtKonstruierbaresDreieck passed (0.0 s)
 - illegalerZweiterParameter passed (0.0 s)
 - gleichschenkelig passed (0.0 s)

- hier: Überraschung, keine explizite Prüfung auf negative Parameter notwendig (insofern Tests ok)

Erinnerung: Einführendes Beispiel



Ergänzung: Hilfsklasse Objekterzeugung (1/4)

```
public class MitarbeiterBuilder {  
    private int id;  
    private String vorname = "Eva"; //Default-Wert  
    private String nachname = "Mustermann";  
    private Set<Fachgebiet> fachgebiete = new HashSet<>();  
  
    public MitarbeiterBuilder() {}  
  
    public static MitarbeiterBuilder createBuilder(){  
        return new MitarbeiterBuilder();  
    }  
  
    public MitarbeiterBuilder vorname(String vorname) {  
        this.vorname = vorname;  
        return this;  
    }  
  
    public MitarbeiterBuilder nachname(String nachname) {  
        this.nachname = nachname;  
        return this;  
    }  
}
```

Ergänzung: Hilfsklasse Objekterzeugung (2/4)



```
public MitarbeiterBuilder id(int id){  
    this.id = id;  
    return this;  
}
```

```
public MitarbeiterBuilder addFachgebiet(Fachgebiet f){  
    this.fachgebiete.add(f);  
    return this;  
}
```

```
public Mitarbeiter build() {  
    Mitarbeiter erg = new Mitarbeiter();  
    erg.setId(this.id);  
    erg.setVorname(this.vorname);  
    erg.setNachname(this.nachname);  
    erg.setFachgebiete(this.fachgebiete);  
    return erg;  
}
```

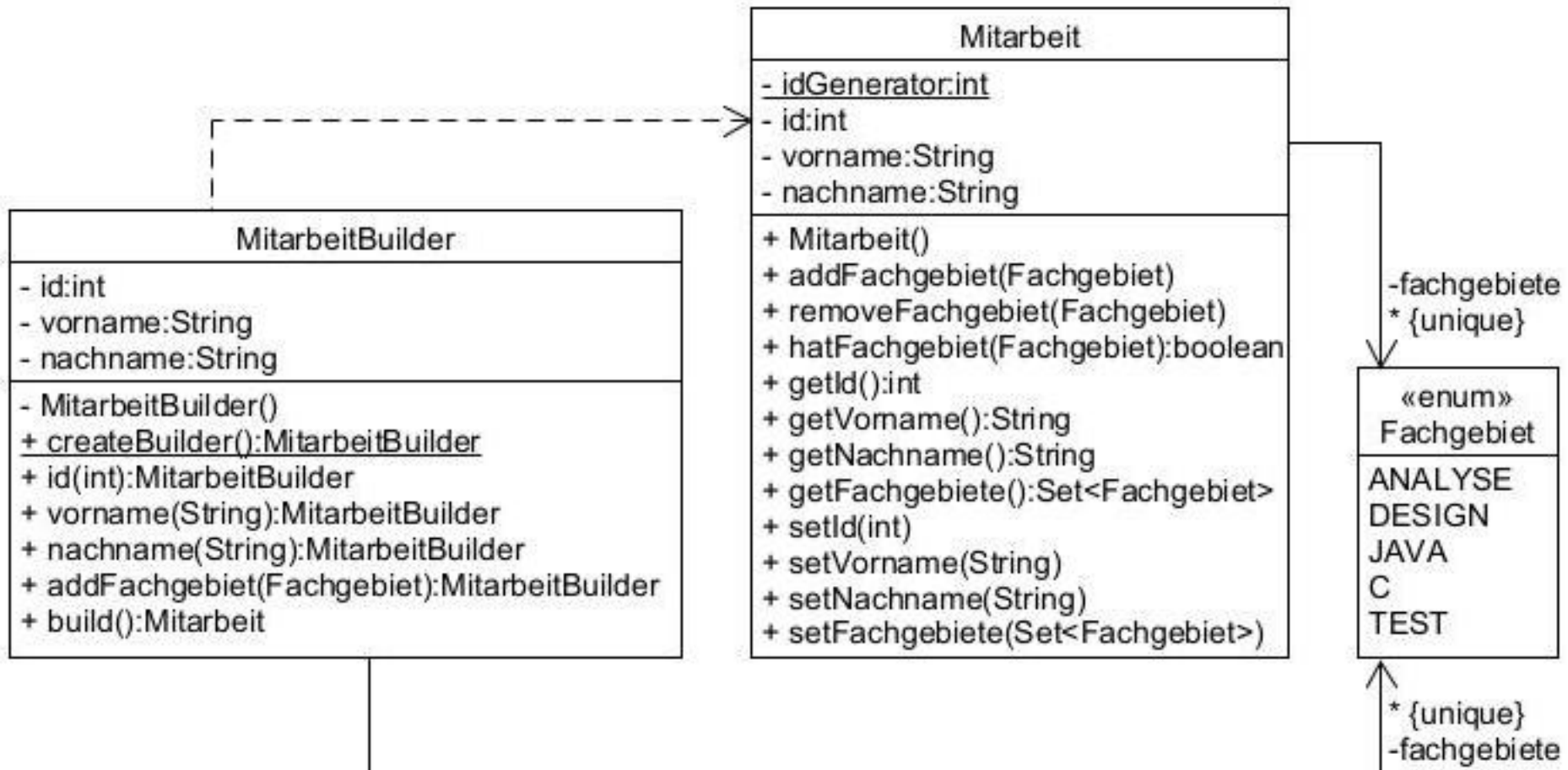
```
}
```


- Beispielnutzung

```
Mitarbeit tmp = MitarbeiterBuilder
    .createBuilder()
    .vorname("Murat")
    .nachname("Meier")
    .mitFachgebiet(Fachgebiet.C)
    .mitFachgebiet(Fachgebiet.JAVA)
    .build();
```

- generell zur Erzeugung von Objekten nutzbar
- durch Fluent-Programming (Method Chaining) besser lesbar

Ergänzung: Hilfsklasse Objekterzeugung (4/4)



- ursprünglich aus TDD abgeleitete Entwicklungsmethode, initiiert von Dan North 2003
- generell Fokus auf Kommunikation zwischen allen Stakeholdern (relevanten Personen) eines SW-Projekts
- Ausschnitt auf Vorgehensweise
 - Fokus auf zentralen Aufgaben des Systems (Features)
 - Jedes Feature wird durch typische Verhaltensweisen (Scenario) beschrieben
 - Jedes Szenario wird mit einzelnen Schritten (Steps) beschrieben
 - Die Beschreibung erfolgt in (strukturiert) natürlicher Sprache; alle Stakeholder können lesen und schreiben
 - Erstellung von Szenarien dient u. a. dem Finden von offenen Problemen, die zuerst gelöst werden müssen
 - Fokus auf Features mit höchstem Stakeholder-Nutzen

- Features sind verwandt mit Use Cases, Szenarien mit Abläufen eines zugehörigen Aktivitätsdiagramms
- strukturierte natürliche Sprache wird von Werkzeugen unterstützt -> Beschreibungen werden als Abnahmetests formal umgesetzt

Werkzeuge:

- Cucumber (<https://cucumber.io/>)
- JBehave (<http://jbehave.org/>)

Literatur:

- S. Rose, M. Wynne, A. Hellesøy, The Cucumber For Java Book, The Pragmatic Programmers, LLC., Dallas, Raleigh (USA), 2015

BDD – Nutzungshinweis für folgende Folien

- Folgende Folien stellen Nutzungsmöglichkeiten von Cucumber in den Mittelpunkt; typischerweise sollte ein Werkzeug nicht im Mittelpunkt eines Vorgehensmodells stehen
- Cucumber erstellt Systemtests; da hier genutzte Beispiele für Verständlichkeit minimal gewählt, entsteht enge Verwandtschaft zu Unit-Tests
- aber: Cucumber unabhängig von BDD zur Testentwicklung
- Feature (Funktionalität) kann Use Case sein
- Scenario (Szenario) entspricht einem Testfall



Beispiel-Features (1/3) – in .feature - Datei

#language: de

Funktionalität: Mitarbeit anlegen

Als einfache nutzende Person

Um eine Mitarbeit anzulegen

Möchte ich nur Vor- und Nachnamen eingeben

So deutsche Begriffe „Wenn
Dann Gegeben Und Aber“
nutzbar sonst „When Then
Given And But“

Szenario: Korrektes Anlegen mit vollständigen Daten

Eine Mitarbeit mit korrektem Vor- und Nachnamen wird angelegt

Wenn Ich als Vorname "Edna" und als Nachname "Meier" eingebe

Dann erhalte ich eine Mitarbeit mit Mitarbeiternummer

Und dem Vornamen "Edna"

Und dem Nachnamen "Meier"

Und mit 0 Fachgebieten

Step

Beispiel-Features (2/3)

Szenario: Korrektes Anlegen mit fehlendem Vornamen

Eine Mitarbeit mit fehlenden Vor- und korrekten Nachnamen wird angelegt

Wenn Ich den Vornamen weglasse und als Nachname "Meier" eingebe

Dann erhalte ich eine Mitarbeit mit Mitarbeiternummer

Und ohne Vornamen

Und dem Nachnamen "Meier"

Und mit 0 Fachgebieten

Szenario: Zu kurzer Nachname

Eine Mitarbeit mit korrektem Vor- aber zu kurzem Nachnamen wird angelegt

Wenn Ich als Vorname "Edna" und als zu kurzen Nachname "X" eingebe

Dann erhalte ich einen Fehler

Szenario: Fehlender Nachname

Eine Mitarbeit mit korrektem Vor- und fehlendem Nachnamen wird angelegt

Wenn Ich als Vorname "Edna" und keinen Nachnamen eingebe

Dann erhalte ich einen Fehler

Szenario: Eindeutige Mitarbeiternummer

Beim Anlegen von zwei Mitarbeitern haben diese unterschiedliche Nummern

Wenn Ich den Vornamen weglasse und als Nachname "Meier" eingebe

Und Ich als zweites Vorname "Edna" und als Nachname "Meier" eingebe

Dann erhalte ich Mitarbeitern mit unterschiedlichen Nummern


```
FachgebieteBearbeiten.feature  MitarbeiterVerwaltungsfeature.feature X
1 #language: de
2 Funktionalität: Mitarbeit anlegen
3   Als einfache nutzende Person
4   Um eine Mitarbeit anzulegen
5   Möchte ich nur Vor- und Nachnamen eingeben
6
7
8   Szenario: Korrektes Anlegen mit vollständigen Daten
9     Eine Mitarbeit mit korrektem Vor- und Nachnamen wird angelegt
10    Wenn Ich als Vorname "Edna" und als Nachname "Meier" eingebe
11    Dann erhalte ich eine Mitarbeit mit Mitarbeiternummer
12    Und dem Vornamen "Edna"
13    Und dem Nachnamen "Meier"
14    Und mit 0 Fachgebieten
```

- Schlüsselworte und Konstanten sind markiert
- Randnotizen weisen auf Implementierung hin (später)
- geht generell auch alles in anderen Sprachen

Video 7

– Jedem Step wird eine Methode zugeordnet

- Jedes Szenario wird einzeln nacheinander abgearbeitet, d. h. jeder Step ausgeführt; scheitert er, dann das Szenario

```
@When("^Ich als Vorname \"(.*)\" und als Nachname \"(.*)\" eingebe$")
public void ich_als_Vorname_und_als_Nachname_eingebe(String arg1
    , String arg2) throws Throwable {
    this.mitarbeit = new Mitarbeiter(arg1, arg2);
}
```

Aufbau:

- Annotation (<regulärer Ausdruck>)
- im regulären Ausdruck stehen in **runden Klammern** reguläre Ausdrücke, die zu einzelnen Parametern gehören, die Methode übergeben werden
- mit ^ und \$ Anfang und Ende des Texts gekennzeichnet (optional)
- in neueren Cucumber-Versionen auch

```
@When("Ich als Vorname {string} und als Nachname {string} eingebe")
```

Cucumber – module-info.java

- Cucumber wird als externes System gesehen, deshalb muss Paket mit Step Definitions exportiert werden

```
module qsCucumber {
```

```
  exports step;
```

```
  requires org.junit.jupiter.api;
```

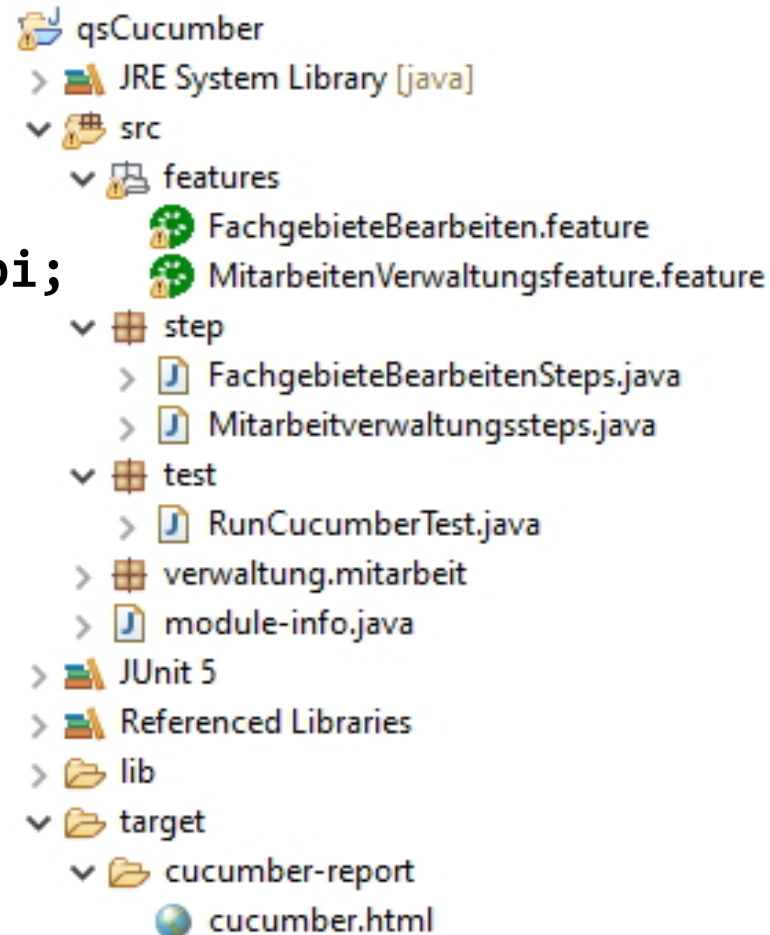
```
  requires org.junit.platform.suite.api;
```

```
  requires io.cucumber.java;
```

```
  requires io.cucumber.core;
```

```
}
```

- sonst nur Fehlermeldung, dass keine Tests gefunden werden



```
import org.junit.platform.suite.api.ConfigurationParameter;
import org.junit.platform.suite.api.IncludeEngines;
import org.junit.platform.suite.api.Suite;
import static io.cucumber.core.options.Constants.*;
@Suite
@IncludeEngines({"cucumber"}) // "junit-jupiter" ergänzbar
@ConfigurationParameter(key = FEATURES_PROPERTY_NAME
    , value = "src/features")
@ConfigurationParameter(key = GLUE_PROPERTY_NAME
    , value = "step")
@ConfigurationParameter(key = PLUGIN_PROPERTY_NAME, value =
    "pretty, html:target/cucumber-report/cucumber.html")
public class RunCucumberTest {
}
```

- Plugin hier zur Formatierung der Ausgabe in HTML
- Feature-Dateien können in mehreren Ordnern liegen
- glue-Parameter für Liste von Paketnamen mit Step Definitions



file:///F:/workspaces/eclipseWS/qsCucumber/target/cucumber-report/cucumber.html

Funktionalität: Mitarbeit anlegen

Als einfache nutzende Person

Um eine Mitarbeit anzulegen

Möchte ich nur Vor- und Nachnamen eingeben

Szenario: Korrektes Anlegen mit vollständigen Daten

Eine Mitarbeit mit korrektem Vor- und Nachnamen wird angelegt

- ✓ **Wenn** Ich als Vorname "Edna" und als Nachname "Meier" eingebe
- ✓ **Dann** erhalte ich eine Mitarbeit mit Mitarbeiternummer
- ✓ **Und** dem Vornamen "Edna"
- ✓ **Und** dem Nachnamen "Meier"
- ✓ **Und** mit 0 Fachgebieten

Szenario: Korrektes Anlegen mit fehlendem Vornamen

- Markierungen zeigen, ob ausgeführt, ob erfolgreich
- verschiedenste Ausgaben (Konsole, JSON, ...) möglich

- „normaler“ Java-Code
- Exemplarvariablen werden genutzt, um Informationen über Steps hinaus zu merken
- verschiedene Ansätze für bestimmte Anforderungen, wie geforderte Ausnahmen (unüblich bei Abnahmetests)
- Annotationen @Given, @When, @Then, @And, @But nur zur Erhöhung der Lesbarkeit unterschieden; theoretisch immer nur eine nutzbar
- genutzte reguläre Ausdrücke müssen eindeutige Zuordnung haben
- Klammern führen zu Parametern der Methode; soll dies nicht sein, steht ?: als erstes in der Klammer, z. B. Punk(?:t|te)
- Methodennamen frei wählbar (s. später: von Cucumber am Anfang generierbar, Binnenmajuskeln auch möglich)

Cucumber – Ausschnitt Step Definitions (1/3)

```
public class Mitarbeiterverwaltungssteps {  
  
    private Mitarbeiter mitarbeit;  
    private Mitarbeiter mitarbeit2;  
    private boolean exception = false;  
  
    @When("^Ich als Vorname \"(.*)\" und als Nachname \"(.*)\" "  
        + "eingebe$")  
    public void ich_als_Vorname_und_als_Nachname_eingebe(  
        String arg1, String arg2) throws Throwable {  
        this.mitarbeit = new Mitarbeiter(arg1, arg2);  
    }  
  
    @Then("^erhalte ich eine Mitarbeiter mit Mitarbeiternummer$")  
    public void erhalte_ich_eine_Mitarbeiter_mit_Mitarbeit_snr()  
        throws Throwable {  
        Assertions.assertNotNull(this.mitarbeit);  
    }  
}
```

Cucumber – Ausschnitt Step Definitions (2/3)

```
@Then("^dem Vornamen \"(.*)\"$")
public void dem_Vornamen(String arg1) throws Throwable {
    Assertions.assertEquals(arg1, this.mitarbeit.getVorname());
}
```

```
@Then("^dem Nachnamen \"(.*)\"$")
public void dem_Nachnamen(String arg1) throws Throwable {
    Assertions.assertEquals(arg1, this.mitarbeit.getNachname());
}
```

```
@Then("^mit (\\d+) Fachgebieten$")
public void mit_Fachgebieten(int arg1) throws Throwable {
    Assertions.assertTrue(
        arg1 == this.mitarbeit.getFachgebiete().size());
}
// auch moeglich
// @Then("mit {int} Fachgebieten")
```


Cucumber – Ausschnitt Step Definitions (3/3)

```
@When("^Ich als Vorname \"(.*)\" und keinen Nachnamen"
      + " eingebe$")
public void ich_als_Vorname_und_keinen_Nachnamen_eingebe(
    String arg1) throws Throwable {
    try {
        this.mitarbeit = new Mitarbeiter(arg1, null);
    } catch (IllegalArgumentException e) {
        this.exception = true;
    }
}

@Then("^erhalte ich einen Fehler$")
public void erhalte_ich_einen_Fehler() throws Throwable {
    Assert.assertTrue(exception);
    this.exception = false;
}
```

Möglichkeiten in Feature-Dateien



- Feature: steht am Anfang, dient der Anschauung, geht nicht in die Implementierung ein (aber Dokumentation)
- Scenario: kann (soll) mit informeller Erklärung beginnen, nutzt folgende Möglichkeiten, die in Implementierung einfließen
- @Given: Beschreibt die Ausgangssituation
- @When: Beschreibt die Voraussetzung
- @Then: Beschreibt das erwartete Ergebnis
- @And: wird zur Bedingungsverknüpfung benutzt
- Sequenzen @When @Then @When @Then nutzbar
- Background: beschreibt Ausgangssituation, die alle Szenarien teilen sollen (Given)
- Scenario Outline mit Examples: Sammlung von Szenarien, in die Beispieldaten übernommen werden

Beispiel für Background

Feature: Fachgebiete von Mitarbeitern bearbeiten

As a einfacher Nutzer

In order um Fachgebiete von Mitarbeitern hinzufügen oder löschen

I want möchte ich Fachgebiete hinzufügen und löschen können,
wobei jede Mitarbeit maximal drei Fähigkeiten hat

Hinweis: Für Initialisierung geprüft, dass am Anfang noch
kein Fachgebiet vorliegt

Background:

Given seien die folgenden Mitarbeitern mit folgenden Fachgebieten:

Vorname	Nachname	ANALYSE	DESIGN	JAVA	C	TEST
Edna	Meier	X	X	X		
Jelena	Kaiser				X	X
Murat	Mutlu					X
Heinz	Müller					

Umsetzung des Background (1/2)



```
private Map<String, Mitarbeit> mitarbeiten; // Nachname eindeutig
@Given("^Gegeben seien die folgenden Mitarbeiten"
      + " mit folgenden Fachgebieten:$")
public void GegebenSeienDieFolgendenMitarbeiten(DataTable arg1)
      throws Throwable {
    this.mitarbeiten = new HashMap<>();
    List< Map<String, String>> werte
        = arg1.asMaps(String.class, String.class);
    for (int i = 0; i < werte.size(); i++) {
        Mitarbeit tmp = MitarbeitBuilder.createBuilder()
            .vorname(werte.get(i).get("Vorname"))
            .nachname(werte.get(i).get("Nachname"))
            .build();
        for (Fachgebiet f: Fachgebiet.values()) {
            if (werte.get(i).get(f.toString()) != null
                && werte.get(i).get(f.toString()).equals("X")) {
                tmp.addFachgebiet(f);
            }
        }
    }
}
```

Umsetzung des Background (2/2)

```
        this.mitarbeiten.put(tmp.getNachname(), tmp);
    }
}
```

DataTable abhängig von der Form in verschiedene Typen übersetzbar:

List<List<String>>

List<Map<String, String>>

Map<String, String>


Map<String, List<String>>

Map<String, Map<String, String>>

Ausschnitt: <https://cucumber.io/docs/cucumber/api/?lang=java>



Datatable – asMaps: List<Map<String,String>>

Vorname	Nachname	ANALYSE	DESIGN	JAVA	C	TEST
Edna	Meier	X	X	X		
Jelena	Kaiser				X	X
Murat	Mutlu					X
Heinz	Müller					

- Listenelement: [(Vorname,Edna), (Nachname,Meier), (ANALYSE,X), (DESIGN,X), (JAVA,X), (C,null), (TEST,)]

Map<String,String>
- Listenelement: [(Vorname,Jelena), (Nachname,Kaiser), (ANALYSE,null), (DESIGN,null), (JAVA,null), (C,X), (TEST,X)]
konfigurierbar als leerer String "" (Standard: null)
- Listenelement: [(Vorname,Murat), (Nachname,Mutlu), (ANALYSE,null), (DESIGN,null), (JAVA,null), (C,null), (TEST,X)]
- Listenelement: [(Vorname,Heinz), (Nachname,Müller), (ANALYSE,null), (DESIGN,null), (JAVA,null), (C,null), (TEST,null)]

Datatable – asLists: List<List<T>>

Vorname	Nachname	ANALYSE	DESIGN	JAVA	C	TEST
Edna	Meier	X	X	X		
Jelena	Kaiser				X	X
Murat	Mutlu					X
Heinz	Müller					

1. Listenelement: [Vorname, Nachname, ANALYSE, DESIGN, JAVA, C, TEST]
List<String> mit 7 Elementen 
2. Listenelement: [Edna, Meier, X, X, X, null, null]
konfigurierbar als leerer String "" 
3. Listenelement: [Jelena, Kaiser , null, null, null, X, X]
4. Listenelement: [Murat, Mutlu , null, null , null, null, X]
5. Listenelement: [Heinz, Müller , null, null , null, null, null]

Beispiel: Scenario Outline

Scenario Outline: Einfaches erfolgreiches Hinzufügen

When Ich das Fachgebiet <Fachgebiet> der Mitarbeit mit Namen <Nachname> hinzufüge

Then Hat die Mitarbeit die alten Fachgebiete und zusätzlich das Fachgebiet <Fachgebiet>

Examples:

 Nachname	 Fachgebiet	
 Kaiser	 JAVA	
 Mutlu	 ANALYSE	
 Müller	 JAVA	

Umsetzung Scenario Outline (1/2)

```
private Mitarbeit aktuell;  
private Set<Fachgebiet> vorher;  
  
@When("^Ich das Fachgebiet (.*) der Mitarbeit"  
      + " mit Namen (.*) hinzufüge$")  
public void ich_das_Fachgebiet_der_Mitarbeit(String arg1  
      , String arg2) throws Throwable {  
    this.aktuell = this.mitarbeiten.get(arg2);  
    this.vorher = new HashSet<>();  
    for(Fachgebiet f: this.aktuell.getFachgebiete()){  
        this.vorher.add(f);  
    } // unabhängiges Objekt, keine Referenz  
    this.aktuell.addFachgebiet(Fachgebiet.valueOf(arg1));  
}
```

Umsetzung Scenario Outline (2/2)

```
private Mitarbeit aktuell;  
private Set<Fachgebiet> vorher;
```

```
@Then("^Hat die Mitarbeit die alten Fachgebiete"  
      +" und zusätzlich das Fachgebiet (.*)$")  
public void hat_Mitarbeit_alte_Fachgebiete_und_Fachgebiet  
      (String arg1) throws Throwable {  
    this.vorher.add(Fachgebiet.valueOf(arg1));  
    Assertions.assertEquals(this.vorher  
        , this.aktuell.getFachgebiete()  
        , "Fachgebiet nicht korrekt ergaenzt")  
}
```

Grundlage:

- ✓ **Gegeben seien** die folgenden Mitarbeitenden mit folgenden Fachgebieten:

Vorname	Nachname	ANALYSE	DESIGN	JAVA	C	TEST
Edna	Meier	X	X	X		
Jelena	Kaiser				X	X
Murat	Mutlu					X
Heinz	Müller					

Szenariogrundriss: Einfaches erfolgreiches Hinzufügen

- ✓ **Wenn** Ich das Fachgebiet <Fachgebiet> der Mitarbeit mit Namen <Nachname> hinzufüge
- ✓ **Dann** Hat die Mitarbeit die alten Fachgebiete und zusätzlich das Fachgebiet <Fachgebiet>

Beispiele:

	Nachname	Fachgebiet
✓	Kaiser	JAVA
✓	Mutlu	ANALYSE
✓	Müller	JAVA

Feature: Nur zur Veranschaulichung von Möglichkeiten

mehrere Annotationen (Charakterisierungen, Tags) erlaubt

auch gesamtes Feature so tag-bar

@L2 @L3 @L5

Scenario: Nr1

When nichts passiert

Then passiert nichts

@L3 @L2 @L4

Scenario: Nr2

When nichts passiert

Then passiert nichts

```
public class DummySteps {  
  
    @Before // korrekt: import io.cucumber.java.Before;  
    public void beforeScenario1(){  
        System.out.println("before1");  
    }  
  
    @After // import io.cucumber.java.After;  
    public void afterScenario1(){  
        System.out.println("after1");  
    }  
  
    @Before("@L5")  
    public void beforeScenario2(){  
        System.out.println("before2");  
    }  
}
```

```
@After(order = 1, value = "@L2 and not @L4")
public void afterScenario2(){
    System.out.println("after2");
}
```

```
@When("^nichts passiert$")
public void nichts_passiert throws Throwable {
}
```

```
@Then("^passiert nichts$")
public void passiert_nichts throws Throwable {
}
```

```
}
```

```
@Suite
@IncludeEngines({"cucumber"})
//@IncludeTags("L3 & L2 & !L4")
@IncludeTags("L3 | L2 | L1")
@ConfigurationParameter(
    key = FEATURES_PROPERTY_NAME
    , value = "src/features")
@ConfigurationParameter(
    key = GLUE_PROPERTY_NAME
    , value = "step")
@ConfigurationParameter(
    key = PLUGIN_PROPERTY_NAME
    , value = "pretty"
    + ", html:target/cucumber-report"
    + "/cucumber.html")
public class StartCucumberTestV2 {
}
```

```
@L2 @L3 @L5
Scenario: Nr1
before2
before1
nix
passiert
after1
after2

@L3 @L2 @L4
Scenario: Nr2
before1
nix
passiert
after1
```

Cucumber weiter: eigene Parametertypen (1/4)

```
public class Punkt {
    private int x;
    private int y;

    public Punkt(){ }

    public int getX() { return x;}
    public void setX(int x) { this.x = x; }

    public int getY() { return y; }
    public void setY(int y) { this.y = y; }

    @Override
    public String toString() {
        return "Punkt{" + "x=" + x + ", y=" + y + '}';
    }
}
```


Cucumber weiter: eigene Parametertypen (2/4)

Feature: Nur zur Veranschaulichung der Objekt-Erzeugung

Scenario: Punkte beschreiben Parallele zur Y-Achse

When der erster Punkt (2,2) ist

And der zweite Punkt (2,7) ist

Then liegt die Gerade parallel zur Y-Achse

Scenario: Punkte beschreiben keine Parallele zur Y-Achse

When der erster Punkt (2,2) ist

And der zweite Punkt (3,7) ist

Then liegt die Gerade nicht parallel zur Y-Achse

Cucumber weiter: eigene Parametertypen (3/4)

```
public class PunkteSteps {
    private Punkt p1;
    private Punkt p2;

    @ParameterType("\\(\\d+, \\d+\\)")
    public Punkt punkt(String txt) {
        Punkt ergebnis = new Punkt();
        String[] werte = txt.split(",");
        try {
            ergebnis.setX(Integer.parseInt(werte[0].substring(1)));
            ergebnis.setY(Integer.parseInt(werte[1].substring(0
                , werte[1].length() - 1)));
        } catch (Exception e) {
            throw new IllegalArgumentException(
                "Beide Punkt-Koordinaten muessen int-Werte "
                + "sein: " + Arrays.asList(werte));
        }
        return ergebnis;
    }
}
```

Software-Qualität

Cucumber weiter: eigene Parametertypen (4/4)

```
@When("der erster Punkt {punkt} ist")
public void der_erster_Punkt(Punkt p1) throws Throwable {
    this.p1 = p1;
}

@And("der zweite Punkt {punkt} ist")
public void der_zweite_Punkt_ist(Punkt p2) throws Throwable {
    this.p2 = p2;
}

@Then("^liegt die Gerade (.*)parallel zur Y-Achse$")
public void liegt_die_Gerade_parallel_zur_Y_Achse(String check)
    throws Throwable {
    if (check.equals("nicht ")){
        Assertions.assertTrue(p1.getX() != p2.getX());
    } else {
        Assertions.assertTrue(p1.getX() == p2.getX());
    }
}
}
```

Beispiel: Entwicklung mit Cucumber (1/9)

- Mit Stakeholdern wird zentrales Feature und dann zugehörige Szenarien diskutiert

Feature: Auswertung nach Kniffelregeln

As a Spieler

In order um meinen Wurf berechnen zu lassen

**I want die Berechnungsmöglichkeit zu wählen
und die passenden Punkte zu berechnen**

Scenario: Berechnung nur Einer

When ich 2,3,1,4,5 geworfen habe

And nur 1er auswerten lasse

Then erhalte ich 1 Punkt

Scenario: Berechnung nur Dreier

When ich 3,3,1,3,5 geworfen habe

And nur 3er auswerten lasse

Then erhalte ich 9 Punkte

Beispiel: Entwicklung mit Cucumber (2/9)

- Start von Cucumber

```
@Suite
```

```
@IncludeEngines({"cucumber"})
```

```
@ConfigurationParameter(key = FEATURES_PROPERTY_NAME  
    , value = "src/features")
```

```
@ConfigurationParameter(key = GLUE_PROPERTY_NAME  
    , value = "step")
```

```
@ConfigurationParameter(key = PLUGIN_PROPERTY_NAME, value  
    = "pretty, html:target/cucumber-report/cucumber.html")
```

```
public class StartTest {}
```

Ausgabe zeigt, dass kein Feature und kein Step life

Scenario: Berechnung nur Einer

- 🔍 **When** ich 2,3,1,4,5 geworfen habe
- 🔍 **And** nur 1er auswerten lasse
- 🔍 **Then** erhalte ich 1 Punkt

Beispiel: Entwicklung mit Cucumber (3/9)

- Testausgabe generiert Vorschlag für fehlende Implementierung
`io.cucumber.junit.platform.engine.UndefinedStepException: The step 'ich 2,3,1,4,5
geworfen habe' and 2 other step(s) are undefined.
You can implement these steps using the snippet(s) below:`

```
@When("ich {double} geworfen habe")
public void ich_geworfen_habe(Double double1) {
    // Write code here that turns the phrase above into concrete actions
    throw new io.cucumber.java.PendingException();
}

@When("nur 1er auswerten lasse")
public void nur_1er_auswerten_lasse() {
    // Write code here that turns the phrase above into concrete actions
    throw new io.cucumber.java.PendingException();
}

@Then("erhalte ich {int} Punkt")
public void erhalte_ich_punkt(Integer int1) {
    // Write code here that turns the phrase above into concrete actions
    throw new io.cucumber.java.PendingException();
}
```

Beispiel: Entwicklung mit Cucumber (4/9)

- Code wird in beliebige Java-Klasse im Testordner „step“, z. B. KniffelSteps.java eingebaut, Cucumber gestartet
- Ausgabe markiert erstes auftretendes Problem

Scenario: Berechnung nur Einer

ⓘ **When** ich 2,3,1,4,5 geworfen habe

```
io.cucumber.java.PendingException: TODO: implement me
    at qsAufgabeKniffelEinstieg/step.KniffelSteps.ich_geworfen_habe(KniffelSteps.java:22)
    at *.ich 2,3,1,4,5 geworfen habe(file:///F:/workspaces/eclipseWS/qsAufgabeKniffelEins
```

ⓘ **And** nur 1er auswerten lasse

ⓘ **Then** erhalte ich 1 Punkt

Beispiel: Entwicklung mit Cucumber (5/9)

- Software-Entwurf, z. B. erst Interface oder Mock-Klasse
package business;

```
import java.util.List;
```

```
public interface AuswerterInterface {
```

```
    /** Die Punktzahl berechnet sich aus der Anzahl, wie oft  
     * der wert in den gewuerfelten Werten vorkommt mal dem  
     * wert; also alsGleiche(3,[2,3,3,6,3]) = 9  
     *  
     * @param wuerfel Augenzahlen der fuenf Wuerfel  
     * @return Summe der Augen der Wuerfel mit dem Wert wert  
     */
```

```
    public int alsGleiche(int wert, List<Integer> wuerfel);
```

```
}
```


Beispiel: Entwicklung mit Cucumber (6/9)

```
public class KniffelSteps { // Step-Definitionen ausimplementieren

    @ParameterType("\\d+, \\d+, \\d+, \\d+, \\d+")
    public List<Integer> wuerfel(String txt) {
        System.out.println("Mein Converter2 bekommt: " + txt);
        List<Integer> ergebnis = new ArrayList<Integer>();
        String[] werte = txt.split(",");
        try {
            for(int i = 0; i < werte.length; i++) {
                ergebnis.add(Integer
                    .parseInt(werte[i]));
            }
        } catch (Exception e) {
            throw new IllegalArgumentException(
                "Wuerfe nicht korrekt angegeben: " + Arrays.asList(werte));
        }
        return ergebnis;
    }
}
```

Beispiel: Entwicklung mit Cucumber (7/9)

```
private AuswerterInterface auswerter;  
private List<Integer> wuerfel;  
private int punkte;
```

```
@When("ich {wuerfel} geworfen habe")  
public void ich_geworfen_habe(List<Integer> wuerfel) throws Throwable {  
    this.wuerfel = wuerfel;  
}
```

```
@When("^nur (\\d+)er auswerten lasse$")  
public void nur_er_auswerten_lasse(int wert) throws Throwable {  
    this.punkte = this.auswerter.alsGleiche(wert, this.wuerfel);  
}
```

```
@Then("^erhalte ich (\\d+) Punk(?:t|te)$")  
public void erhalte_ich_Punkt(int arg1) throws Throwable {  
    Assertions.assertEquals(arg1, this.punkte);  
}
```

Beispiel: Entwicklung mit Cucumber (8/9)

- nach (ersten) Inkrement kann es Fehler geben

Scenario: Berechnung nur Einer

- ✓ **When** ich 2,3,1,4,5 geworfen habe
- ✗ **And** nur 1er auswerten lasse

```
java.lang.NullPointerException: Cannot invoke  
"business.AuswerterInterface.alsGleiche(int, java.util.List)" because  
"this.auswerter" is null  
    at  
    qsAufgabeKniffelEinstieg/step.KniffelSteps.nur_er_auswerten_lasse(KniffelSteps.  
        at *.nur 1er auswerten lasse(file:///F:/workspaces/eclipseWS  
/qsAufgabeKniffelEinstieg/src/features/Kniffelauswertung.feature:9)
```

- **Then** erhalte ich 1 Punkt

Beispiel: Entwicklung mit Cucumber (9/9)

- inkrementelle Weiterentwicklung des Codes

```
public class Auswerter implements AuswerterInterface {
    @Override
    public int alsGleiche(int wert, List<Integer> wuerfel) {
        return wuerfel.stream().filter(w -> w == wert)
            .collect(Collectors.summingInt(Integer::intValue));
    }
}
```

```
@Before // in KniffelSteps.java
public void setUp() {
    this.auswerter = new Auswerter();
}
```

Scenario: Berechnung nur Einer

- ✓ **When** ich 2,3,1,4,5 geworfen habe
- ✓ **And** nur 1er auswerten lasse
- ✓ **Then** erhalte ich 1 Punkt

- neben Test von entwickelnden Personen werden Integrations- und Systemtests von anderen (QS-)Personen erstellt
- wichtig ist immer, dass auch typisches Verhalten getestet wird
- diese Tests können auch von Endnutzenden bzw. Fachabteilungen erstellt werden
- Beispiel: Was für Eigenschaften hat ein typischer Versicherungsnehmer; welche Extremfälle konnten beobachtet werden
- Fachabteilungen haben aber keine Personen zur Programmierung deshalb Suche nach einfacheren Eingabemöglichkeiten
- Ansatz von Fit (<http://fit.c2.com/>): nutze Word und darin Tabellen, speichere Dokument zur Verarbeitung in html ab
- FitNesse (<http://www.fitnessse.org/>) basiert auf der Idee

Fit (1/4): Word-Dokument (.doc oder .docx-Format)

Testfallspezifikationen

Mit den ersten Testfällen wird für einen neuen Mitarbeiter, der die Gebiete gebiet1, gebiet2 und gebiet3 bereits beherrscht, überprüft, wie die Reaktion beim Hinzufügen des Gebiets gebietNeu aussehen soll.

de MITest1				
gebiet1	gebiet2	gebiet3	gebietNeu	hinzu()
Java	C	Test	C	ok
Java	C	Test	Analyse	fehler
Java	Java	Java	Java	ok

Testklassenname, Parameter, Testmethodenname(n) in neuen Spalten; zunächst nur int, boolean, String möglich; Rest selbst zusammensetzen

Fit (2/4): Tests (Ausschnitt)

```
public class MITest1 extends ColumnFixture{
    public String gebiet1; // Namen aus Dokument (public)
    public String gebiet2;
    public String gebiet3;
    public String gebietNeu;

    public String hinzu(){
        Mitarbeit ma = new Mitarbeit();
        try {
            ma.addFachgebiet(Fachgebiet.fachgebietErstellen(gebiet1));
            ma.addFachgebiet(Fachgebiet.fachgebietErstellen(gebiet2));
            ma.addFachgebiet(Fachgebiet.fachgebietErstellen(gebiet3));
        } catch (IllegalArgumentException e) {
            throw new IllegalArgumentException("Verfruehte Ausnahme");
        }
        try {
            ma.addFachgebiet(Fachgebiet.fachgebietErstellen(gebietNeu));
        } catch (IllegalArgumentException e) {
            return "fehler";
        }
        return "ok";
    }
}
```

Fit (3/4): Aufruf

The screenshot displays the IDE interface for configuring a Java application. The main window is titled "Properties for TestBuchFitMini" and shows the "Run/Debug Settings" tab. The configuration is for a launch configuration named "Fit Start".

- Name:** Fit Start
- Main class:** fit.FileRunner
- Project:** TestBuchFitMini
- Include system libraries when sea** (unchecked)
- Include inhe** (unchecked)
- Stop in main** (unchecked)

The "Program arguments" field is set to: `TestfallbeschreibungFit.htm output.html`

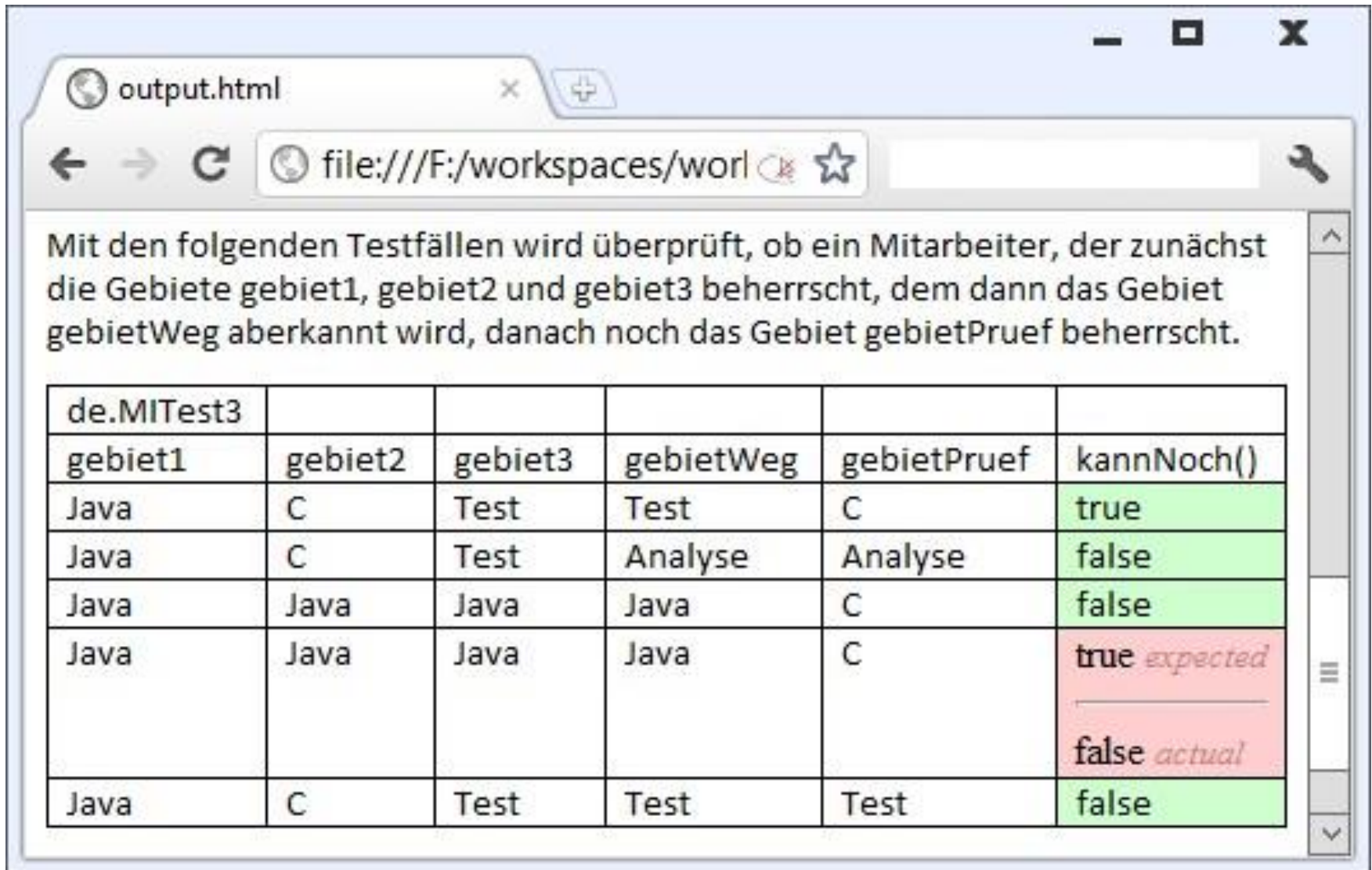
The IDE also shows a project explorer on the right with the following structure:

- TestBuchFitMini
 - src
 - de
 - MITest1.java
 - MITest2.java
 - MITest3.java
 - verwaltung.mitarbeiter
 - Fachgebiet.java
 - Mitarbeiter.java
 - Referenced Libraries
 - JRE System Library [zulu11.3
 - lib
 - output.html
 - TestfallbeschreibungFit.docx
 - TestfallbeschreibungFit.htm

At the bottom, the console shows the output of the application:

```
<terminated> Fit Start [Java Appl.  
10 right, 1 wrong, 0 ignored, 0 exceptions
```

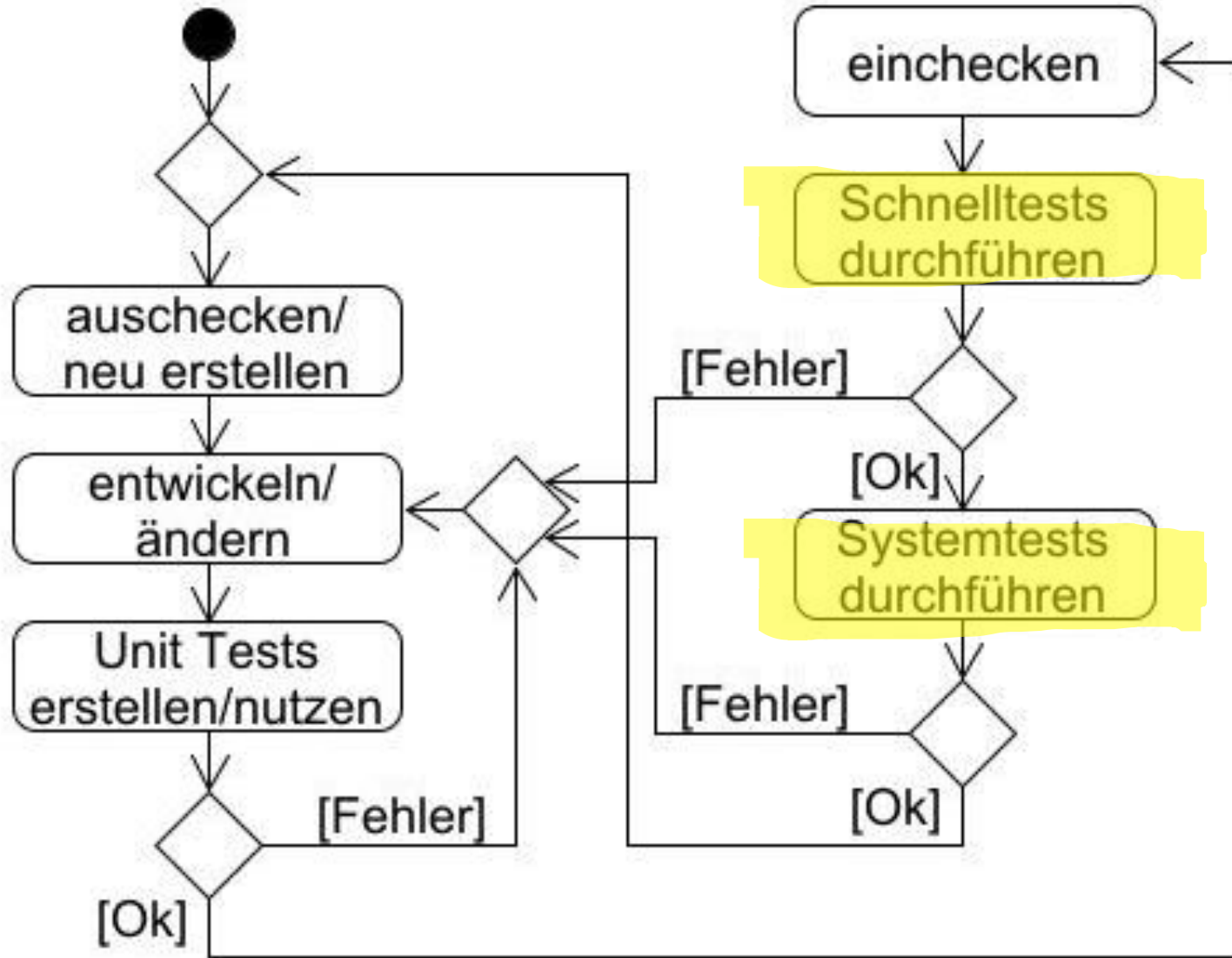

Fit (4/4): Ausgabe output.html



Mit den folgenden Testfällen wird überprüft, ob ein Mitarbeiter, der zunächst die Gebiete gebiet1, gebiet2 und gebiet3 beherrscht, dem dann das Gebiet gebietWeg aberkannt wird, danach noch das Gebiet gebietPruef beherrscht.

de.MITest3	gebiet1	gebiet2	gebiet3	gebietWeg	gebietPruef	kannNoch()
Java	C	Test	Test	C		true
Java	C	Test	Analyse	Analyse		false
Java	Java	Java	Java	C		false
Java	Java	Java	Java	C		true <i>expected</i> false <i>actual</i>
Java	C	Test	Test	Test		false

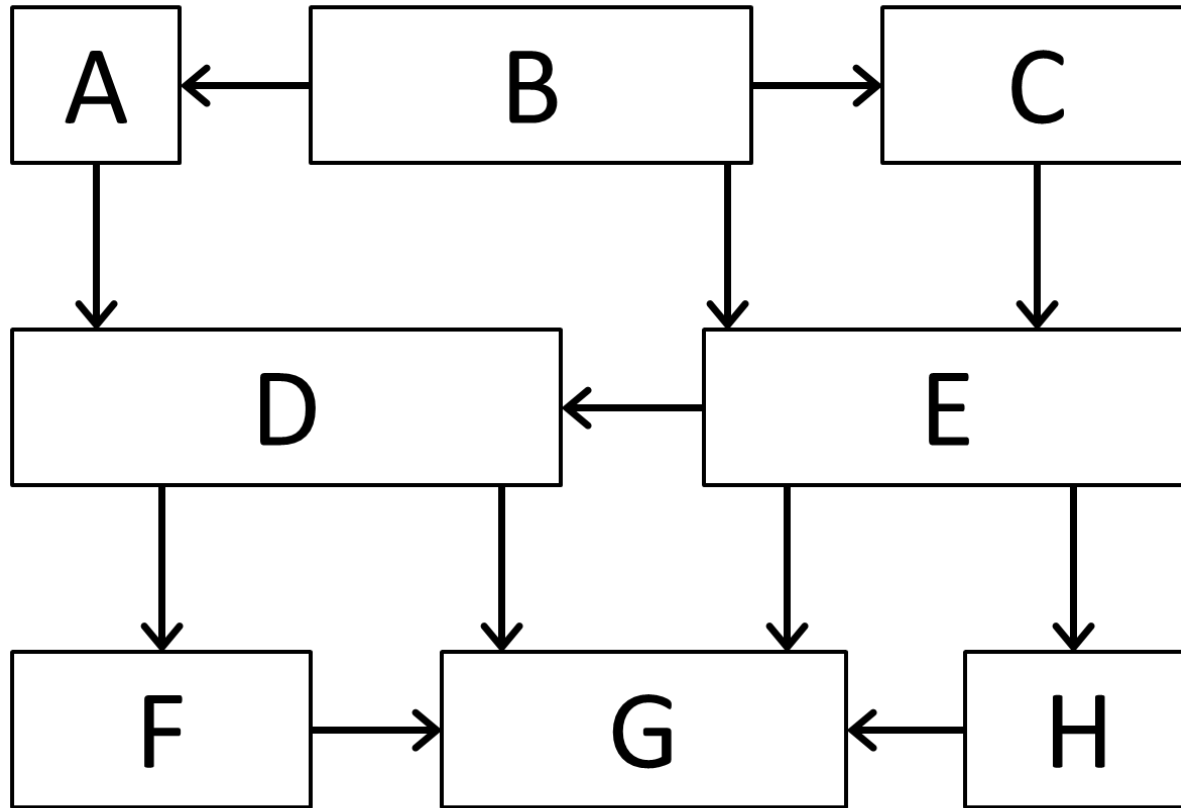
erweiterter Entwicklungsprozess



- Was wann testen
 - Wie werden Klassen testbar
 - Entwicklung von Mocks mit der Hand
 - Einführung in Mockito
 - Wann ist Mocking-Werkzeug sinnvoll
-
- Literatur: S. Freeman, N. Pryce, Growing Object-Oriented Software, Guided by Tests, Addison-Wesley (USA), 2010

Wann was testen (1/2)

- Abhängigkeiten beachten:



- sinnvoll: G F D A H E C B G F H D E A C B

- weitere Faktoren
 - Kritikalität einer Komponente
 - häufig benutzte Komponente
 - in welcher Reihenfolge fertig gestellt
 - Oberflächen, erst bei nur noch wenig erwarteten Änderungen
- Um unabhängiger von der Fertigstellung anderer zu werden:
 - Programmierung immer gegen Schnittstellen
 - Schnittstellen selbst minimal passend realisieren (Mocking)

Erinnerung: Bedeutung von Schnittstellen

- Schnittstellen sind zentrales Element des *Design by Contract*
- vorgegebene Aufgabe: Implementiere mir folgende Funktionalität ... beschrieben durch
 - Vorbedingung
 - Signatur <Sichtbarkeit> <Methodenname>(<Parameter>)...
 - Nachbedingung
- Entwicklung realisiert OO-Programm (Details sind frei)
- Entwicklung garantiert, dass Schnittstelle (oder Fassade) gewünschte Funktionalität liefert
- generell bei Vererbungen und Implementierungen die am wenigsten spezielle benötigte Klasse nutzen; deshalb
List<Projektaufgaben> aufgaben und nicht
ArrayList<Projektaufgaben> aufgaben im Code

- Bei Objekten mit internen Zuständen ist der Test von außen mit JUnit sehr schwierig
- es ist oftmals hilfreich, zusätzliche Methoden zu implementieren, die das Testen einfacher machen

```
public ... getInternerZustand(){ return "..." }
```
- Häufiger reicht get nicht aus, es muss Methoden (set) geben, mit denen man ein Objekt von außen in einen gewünschten Zustand versetzen kann
- Im Quellcode sollen Testmethoden eindeutig von anderen Methoden getrennt sein, damit sie ggfls. automatisch gelöscht werden können
- Bei komplexeren Klassen sollte man Teile der Entwicklung bereits testen, hierzu müssen die aufgerufenen Methoden zumindest in minimaler Form implementiert werden

- Bis jetzt wurde nur eine Klasse betrachtet, die keine Assoziation zu anderen zu testenden Klassen hat, diese Klassen werden elementare Klassen genannt
- Grundsätzlich sollte man versuchen, zunächst elementare Klassen und dann Klassen, die auf diesen aufbauen, zu testen
- Da es in Entwicklung nicht garantiert werden kann, wann Klassen vorliegen, muss man sich dann mit Trick behelfen
- Mock: benötigte Klasse soweit selbst implementieren, dass man die eigene Klasse testen kann
- Grundregel: Mock so primitiv wie möglich zu halten
- Liegt die Klasse vor, die man temporär durch den Mock prüfen wollte, müssen Tests mit realer Klasse wiederholt werden

- Klasse mit den notwendigen Methoden implementiert, die alle die leere Implementierung oder die Rückgabe eines Dummy-Werts enthalten (auch Stub genannt)

```
public void setParameter(int parameter){}
public int getParameter() { return 0;}
```

- Implementierung wird ergänzt, dass wir unsere Klasse testen können (möglichst einfache Fallunterscheidungen, man geht von Korrektheit der anderen Klasse aus)
- Es gibt Werkzeuge, die die einfachst möglichen Mocks automatisch generieren, die können dann ergänzt werden
- neben den Tests entsteht ein zusätzlicher Codieraufwand für Mocks, in größeren (erfolgreichen) Projekten kann der Anteil des Testcodes am Gesamtcode in Abhängigkeit von der Komplexität des Systems zwischen 30% und 70% liegen!

Beispiel für Mock-Erstellung (1/5)

Video 8

```
public class Buchung {
    public static LogDatei logging;
    ...
    public synchronized void abbuchen(int id, Konto konto,
                                       int betrag) throws BuchungsException{
        if(konto.istLiquide(betrag)){
            konto.abbuchen(betrag);
            logging.schreiben(id + " bearbeitet");
        }
        else{
            logging.schreiben(id + " abgebrochen, insolvent");
            throw new BuchungsException("insolvent");
        }
    }
    ...
}
```

- Zum Test der Methode abbuchen werden Test-Mocks der Klassen Konto und LogDatei benötigt, Ziel ist es möglichst einfache Mocks zu schreiben

Beispiel für Mock-Erstellung (2/5)

- Mock für LogDatei (wahrscheinlich) einfach

```
public class LogDatei { // Mock für Buchung
    public void schreiben(String s){}
}
```

- Für Konto verschiedene Varianten denkbar, wichtig dass `istLiquide()` true und false zurück geben kann

```
public class Konto { //Mock für Buchung, später schöner
    public boolean istLiquide(int betrag){
        return betrag < 1000;
    }
}
```

```
    public void abbuchen(int betrag){}
```

```
}
```

- In dieser Variante muss sich der Tester den Schwellenwert zur Prüfung merken (Mocks müssen auch getestet werden)

Beispiel für Mock-Erstellung (3/5)

```
import junit.framework.TestCase; // Junit 3.8
public class BuchungTest extends TestCase {
    private Konto konto;
    private Buchung buchung;

    protected void setUp() throws Exception {
        Buchung.logging = new LogDatei();
        this.buchung = new Buchung();
        this.konto = new Konto();
    }

    protected void tearDown() throws Exception {
        // logging schließen}
    }

    public void testErfolreicheBuchung(){
        try {
            this.buchung.abbuchen(42, this.konto, 100);
        } catch (BuchungsException e) {
            fail();
        }
    }
}
```

Beispiel für Mock-Erstellung (4/5)

```
public void testErfolgloseBuchung(){  
    try {  
        this.buchung.abbuchen(42, this.konto, 2000);  
        fail();  
    } catch (BuchungsException e) {  
    } }...
```

- Tests des Mocks gehören in eine eigene Testklasse, da sie später bei einer Testwiederholung für eine vollständige Implementierung nicht wiederholt werden
- Werden kompliziertere Mock-Eigenschaften verlangt, sollte die Klasse generell spezielle Konstruktoren oder Methoden zum Setzen des internen Zustands beinhalten

Beispiel für Mock-Erstellung (5/5)

- Minimalziel der Testüberdeckung erreicht

```
BuchungTest.java Buchung.java ×
1 package mockme;
2
3 public class Buchung {
4
5     public static LogDatei logging;
6
7     public synchronized void abbuchen(int id, Konto konto, int betrag)
8         throws BuchungsException {
9         if (konto.istLiquide(betrag)) {
10            konto.abbuchen(betrag);
11            logging.schreiben(id + " bearbeitet");
12        } else {
13            logging.schreiben(id + " abgebrochen, insolvent");
14            throw new BuchungsException("insolvent");
15        }
16    }
17
18 }
```

- vorheriges Beispiel: durch „return betrag<1000“ wird Testwiederverwendung erschwert, Variante Dummy-Konto

```
public class Konto { //Mock für Buchung, später schöner
    private int kontonr;
    public static final int DUMMYKONTONR = 42;
    public Konto (int kontonr){
        this.kontonr = kontonr;
    }
    public boolean istLiquide(int betrag){
        return kontonr != DUMMYKONTONR;
    }
    public void abbuchen(int betrag){}
}
```
- Dummy-Konto könnte fest vereinbart in realen Daten stehen -> Tests bleiben später nutzbar (aber Auswertungen über alle Konten kritisch)

- festes Dummy-Konto kann aber Probleme in realen Anwendungen machen (statistische Auswertungen)
- man benötigt Testsystem neben realen System
- Grundregel: nie auf Systemen testen, die in aktuellen betrieblichen Prozessen genutzt werden
- Testsystem und reales System sollten sich fachlich und inhaltlich wenig (gar nicht) unterscheiden
- Problem Datenschutz: Mutieren von Vor- und Nachnamen reicht nicht aus
- Problem Datenalterung: Testdaten können zu alt werden, z. B. bei Berücksichtigung des Rentenalters
- Problem Datenvarianten: enthalten Testdatensätze alle realen Variationen (-> Testabdeckung)

- Mockito basierte auf EasyMock (auch Alternative), hat dann eigenen Weg eingeschlagen (<http://mockito.org/>)
- Alternative JMock (<http://www.jmock.org/>)
- Vor Werkzeugauswahl immer Kriterien überlegen, z. B.
 - wirklich benötigter Funktionsumfang
 - unterstützte Technologien
 - Lizenz, Kosten
 - Größe des Entwicklungsteams
 - Dokumentation, Support
 - ...
- Kriterien individuell im Projekt gewichten

Beispiel: Buchung, Konto, Logging (1/4)

```
import org.mockito.Mockito;
```

```
...
```

```
@BeforeEach  
public void setUp() throws Exception {  
    this.buchung = new Buchung();  
}
```

```
@Test  
public void testIstLiquide1(){  
    final Konto k = Mockito.mock(Konto.class);  
    Buchung.logging = Mockito.mock(LogDatei.class);  
    try {  
        this.buchung.abbuchen(0, k, BETRAG1);  
        Assertions.fail("fehlender Abbruch");  
    } catch (BuchungsException e) {  
        Mockito.verify(k).istLiquide(BETRAG1);  
        Mockito.verify(Buchung.logging).schreiben(  
            "0 abgebrochen, insolvent");  
    }  
}
```

Direkte Mock-
Erstellung;
Objekte direkt
nutzbar, hier noch
keine besonderen
Rückgabewerte

Interface oder
Klasse Konto
muss existieren

Prüfung, ob
Methoden so
aufgerufen

Beispiel: Buchung, Konto, Logging (2/4)

```
@Test
```

```
public void testIstLiquide3(){  
    Konto k = Mockito.mock(Konto.class);  
    Buchung.logging = Mockito.mock(LogDatei.class);  
  
    Mockito.when(k.istLiquide(BETRAG1)).thenReturn(true);  
  
    try {  
        this.buchung.abbuchen(0, k, BETRAG1);  
    } catch (BuchungsException e) {  
        Assertions.fail("nicht erwarteter Abbruch");  
    }  
    Mockito.verify(k).istLiquide(BETRAG1);  
    Mockito.verify(k).abbuchen(BETRAG1);  
    Mockito.verify(Buchung.logging).schreiben("0 bearbeitet");  
}
```

Spezifikation des Rückgabewerts (Default false)

Beispiel: Buchung, Konto, Logging (3/4)

@Test

```
public void testIstLiquide4(){
    final Konto k = Mockito.mock(Konto.class);
    Buchung.logging = Mockito.mock(LogDatei.class);
    Mockito.when(k.istLiquide(BETRAG1))
        .thenReturn(true).thenReturn(false);
    try {
        this.buchung.abbuchen(0, k, BETRAG1);
    } catch (BuchungsException e) {
        Assertions.fail("nicht erwarteter Abbruch");
    }
    try {
        this.buchung.abbuchen(0, k, BETRAG1);
        Assertions.fail("fehlender Abbruch");
    } catch (BuchungsException e) {
        Mockito.verify(k, Mockito.times(2)).istLiquide(BETRAG1);
        Mockito.verify(k).abbuchen(ArgumentMatchers.anyInt());
        Mockito.verify(Buchung.logging, Mockito.times(2))
            .schreiben(Mockito.anyString());
    }
}
```

mehrere
Ergebnisse
nacheinander

geforderter
mehrfacher
Methodenaufruf

Mockito-Matcher

Beispiel: Buchung, Konto, Logging (4/4)

@Test

```
public void testIstLiquide5(){
    Konto k = Mockito.mock(Konto.class);
    Buchung.logging = Mockito.mock(LogDatei.class);
    Mockito.when(k.istLiquide(AdditionalMatchers.gt(42)))
        .thenThrow(new NumberFormatException());
    try {
        this.buchung.abbuchen(0, k, 100);
        Assertions.fail("fehlender Abbruch");
    } catch (NumberFormatException e) {
        Mockito.verify(k).istLiquide(AdditionalMatchers.gt(42));
    } catch (BuchungsException e) {
        Assertions.fail("falsche Exception");
    }
}
```

Menge von
Matchers zur
Verfügung

Werfen von
Exceptions

Forderungen für Mock-Methodenaufrufe

```
Mockito.verify(k,Mockito.times(2)).istLiquide(BETRAG1);
```

Methode	so oft darf die folgende Methode aufgerufen werden
<code>times(1)</code>	genau einmal; default, kann weggelassen werden
<code>times(n)</code>	genau n-Mal
<code>atLeast(n)</code>	mindestens n-Mal
<code>atMost(n)</code>	höchstens n-Mal
<code>never(k)</code>	niemals

- Methoden ohne Anzahlangabe können beliebig oft aufgerufen werden
- mit `Mockito.verifyZeroInteractions(mock1, ...)` geprüft, dass angegebene Mocks nicht genutzt

- Grundsätzlich kann man Mocks selber schreiben
 - Vorteil: Testcode und Mock-Klassen können getrennt sein; Tests können bei realen Klassen wiederverwendbar sein
- Mockito erlaubt Testerstellung und Erwartung an Mock-Klassen (bzw. reale Klassen eng) zu verknüpfen
- typische Einsatzszenarien, neben fehlenden Klassen:
 - reales Objekt liefert nicht deterministische Ergebnis liefert (Uhrzeit, Sensoren)
 - Ausnahmeverhalten schwer zu erzeugen (Netzwerkprobleme)
 - reale Methode langsam und Ergebnisdetails irrelevant (Datenbankaufruf)

Beispiel: DB-Mock (1/7)

- Aufgabe: DB, auf die vereinfachend nur lesend zugegriffen wird, mocken
- warum: benötigte keine DB-Lizenz, garantiert gleiche Werte ohne aufwändiges „reset“, kein Zeitverlust durch Verbindungsaufbau
- Ansatz: Statement-Interface von JDBC mocken
- Mock in eigene Klasse zur Wiederverwendung auslagern
- Hinweis: Zur Vereinfachung des Beispiels wird SQL nicht ganz sauber eingesetzt

Beispiel: DB-Mock (2/7): Erinnerung JDBC

Datenbankverbindung
herstellen

```
class DriverManager  
Connection con=  
    DriverManager.getConnection(...);  
Statement stmt=  
    con.createStatement();
```

Datenbankanfrage

```
ResultSet rs =  
stmt.executeQuery(...);
```

Ergebnisse
verarbeiten

```
rs.next();  
int n = rs.getInt("KNr");
```

Verbindung zur DB
schließen

```
con.close();
```

Beispiel: DB-Mock (3/7): Programm (1/2)

- Hinweis: Programm auf Testbarkeit ausgelegt; einfaches Einfügen eines Statement-Objekts

```
package db;
```

```
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;
```

```
public class Statistik {  
    private Statement statement;
```

```
    public Statistik(){  
    }  
}
```

```
    public void setStatement(Statement statement){  
        this.statement = statement;  
    }  
}
```

```
module qsMockitoDatenbank {  
    exports db;  
    exports spy;  
    requires java.sql;  
    requires org.mockito;  
    requires  
        org.junit.jupiter.api;  
}
```

Beispiel: DB-Mock (4/7): Programm (2/2)

```
public double studiDurchschnitt(String name)
    throws SQLException{
    int anzahl = 0;
    int summe = 0;
    ResultSet rs= this.statement.executeQuery(
        "SELECT * FROM Noten WHERE Studi ='" + name + "'");
    while (rs.next()) {
        anzahl++;
        summe += rs.getInt(3);
    }
    if (anzahl == 0)
        return 0d;
    else
        return summe/ (anzahl * 100d);
} // wäre natürlich mit AVG in SQL viel einfacher
```

Beispiel: DB-Mock (5/7): Mock-Aufbau (1/2)

```
package noten;
```

```
import java.sql.ResultSet;
```

```
import java.sql.Statement;
```

```
import org.mockito.Mockito;
```

```
public class DBMock {
```

```
    public Statement dbErstellen() throws Exception {  
        final Statement st = context.mock(Statement.class);  
        final String[][] pruefungen = { // Beispieltabelle  
            { "Ute", "Prog1", "400" },  
            { "Uwe", "Prog1", "230" },  
            { "Ute", "Prog2", "170" } };  
    }
```

Beispiel: DB-Mock (6/7): Mock-Aufbau (2/2)

```
ResultSet r1 = Mockito.mock(ResultSet.class);
Mockito.when(r1.next()).thenReturn(true, true, false);
Mockito.when(r1.getInt(3)).thenReturn(
    Integer.parseInt(pruefungen[0][2])
    , Integer.parseInt(pruefungen[2][2]));
Mockito.when(st
    .executeQuery("SELECT * FROM Noten WHERE Studi ='Ute'"))
    .thenReturn(r1);
}
```

- man beachte: bei der gewählten Form ist z. B. die Platzierung der Leerzeichen im String wichtig, die in SQL irrelevant sind (ggfls. eigene Matcher schreiben)
- da nur ein Argument kann auf `Matcher.eq(...)` verzichtet werden

Beispiel: DB-Mock (7/7): Tests (Ausschnitt)

```
public class StatistikTest {  
  
    private Statement db;  
    private Statistik s;  
  
    @BeforeEach  
    public void setUp() throws Exception{  
        this.db = new DBMock().dbErstellen();  
        this.s = new Statistik();  
        this.s.setStatement(db);  
    }  
  
    @Test  
    public void testSchnittUte() throws SQLException {  
        Assertions.assertTrue(2.85 ==  
                                this.s.studiDurchschnitt("Ute"));  
    }  
}
```

Reaktion auf Parameter in Mocks (1/2)

```
public class NurStubbingBeispielTest {  
    private Statement stmt;  
  
    @Before  
    public void setup() throws SQLException {  
        this.stmt = Mockito.mock(Statement.class);  
        Mockito.when(this.stmt.execute(Mockito.anyString()))  
            .thenAnswer( new Answer() {  
                @Override  
                public Object answer(InvocationOnMock inv)  
                    throws Throwable {  
                    System.out.println(  
                        Arrays.asList(inv.getArguments()));  
                    System.out.println(inv.getMethod());  
                    System.out.println(inv.getMock());  
                    String tmp = inv.getArgument(0);  
                    return tmp.length() > 6;  
                }  
            }  
    }  
}
```


Reaktion auf Parameter in Mocks (2/2)

```
@Test
public void stubbingTest() throws SQLException {
    System.out.println(stmt.execute("SELECT *"));
    System.out.println(stmt.execute("INSERT"));
}
```

```
[SELECT *]
public abstract boolean java.sql.Statement.execute(java.lang.String)
throws java.sql.SQLException
Mock for Statement, hashCode: 20066205
true
[INSERT]
public abstract boolean java.sql.Statement.execute(java.lang.String)
throws java.sql.SQLException
Mock for Statement, hashCode: 20066205
false
```

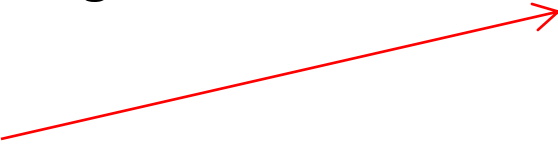
- Achtung, dies ist machbar, ist aber Indikator der Überspezifikation von Mocks (schwer wartbare Tests)

Eigene Matcher (1/2)

```
public class MeinStringMatcher
    implements ArgumentMatcher<String>{

    @Override
    public boolean matches(String arg) {
        System.out.println(arg);
        return arg.equals("INSERT");
    }

    @Override
    public String toString(){
        // Ausgabe bei Verifikationsfehlern
        return "[Quelle ist MeinStringMatcher]";
    }
}
```



- Hinweis: gibt einige Möglichkeiten Matcher mit AND, OR und NOT zu verknüpfen

Eigene Matcher (2/2)

```
public class MeinArgumentMatcherTest {
    private Statement stmt;

    @Before
    public void setup() throws SQLException {
        this.stmt = Mockito.mock(Statement.class);
        Mockito.when(stmt.execute(
            Mockito.argThat(new MeinStringMatcher())))
            .thenReturn(true); // default-Wert false
    }

    @Test
    public void stubbingTest() throws SQLException {
        System.out.println(stmt.execute("SELECT *"));
        System.out.println(stmt.execute("INSERT"));
    }
}
```

```
SELECT *
false
INSERT
true
```

Mocks über Annotationen

```
public class Mockannotationsspielerei {  
    @Mock  
    private List<String> list1;  
  
    @Mock  
    private List<String> list2;  
  
    @Test  
    public void testAnnotations() {  
        MockitoAnnotations.openMocks(this); // Mocking erlauben  
        Mockito.when(this.list1.size()).thenReturn(100);  
        this.list1.add("Hallo");  
        System.out.println(this.list1.size() + " "  
            + this.list1.get(0)  
            + " - " + this.list2.size() + " "  
            + this.list2.get(100));  
    }  
}
```

100 null - 0 null

```
@Spy
```

```
List<String> lists1 = new ArrayList<String>();
```

```
@Spy
```

```
List<String> lists2 = new ArrayList<String>();
```

```
@Test
```

```
public void testspy() {  
    Mockito.when(this.lists1.size()).thenReturn(100);  
    this.lists1.add("Hallo");  
    this.lists2.add("Hallo2");  
    System.out.println(this.lists1.size() + " "  
        + this.lists1.get(0) + " - " + this.lists2.size()  
        + " " + this.lists2.get(0));  
    Mockito.when(this.lists2.size()).thenReturn(100);  
    System.out.println(this.lists2.get(99));  
}
```

```
100 Hallo - 1 Hallo2  
IndexOutOfBoundsException
```

- Mockito bietet einige weitere Möglichkeiten, dabei immer teilweise spezielle Randbedingungen beachten
- unterstützt auch Dependency Injection, z. B. `@InjectMock`, `@Spy` und `@Mock` in Junit-Tests
- Klasse `BDDMockito` vereinfacht Einsatz in BDD
- Mockito etwas einfacher zu nutzen als JMock; insbesondere mehr "normales Java"
- Mockito und JMock haben kleine Anteile, die der andere nicht kann (man kann beide zusammen einsetzen)
- generell kritischer Werkzeugvergleich immer sinnvoll; oft auch gemeinsame Nutzung eine Lösung
- Mockito kann keine Klassenmethoden mocken, geht mit Ergänzung PowerMock <https://github.com/jayway/powermock>

Video 9

- grundsätzlich mit Mockito möglich

@Test

```
public void mainUsage() {  
    PrintStream out = Mockito.mock(PrintStream.class);  
    System.setOut(out);  
    ZuTestendesProgramm.main(new String[]{});  
    Mockito.verify(out).println(ArgumentMatchers  
        .startsWith("erster Text"));  
}
```

http://www.adam-bien.com/roller/abien/entry/testing_system_out_println_outputs

- flexibler mit spezieller Umsetzung in SystemLambda
<https://github.com/stefanbirkner/system-lambda>
- folgendes Beispiel: Programmiere eine Einkaufliste, zu der neue Elemente hinzugefügt und die ausgegeben werden kann

Erster Ansatz: Schwer testbare Lösung (1/5)

```
public class Dialog {
    private List<String> einkaeufe = new ArrayList<>();
    public String leseString(){
        try {
            return new Scanner(System.in).nextLine();
            //return in.nextLine();
        }catch (Exception e) {
        }
        return "";
    }
    public int leseInt(){
        try {
            return Integer.decode(this.leseString()).intValue();
        } catch (NumberFormatException e) {
            return 0;
        }
    }
}
```


Erster Ansatz: Schwer testbare Lösung (2/5)

```
public void start() {
    int eingabe = -1;
    while (eingabe != 0) {
        System.out.println("(0) beenden\n"
            + "(1) neuer Eintrag\n" + "(2) Liste anzeigen: ");
        eingabe = this.leseInt();
        switch (eingabe) {
            case 1: {
                System.out.print("auf Liste: ");
                String neu = this.leseString();
                this.einkaeufe.add(neu);
                break;
            }
            case 2: {
                System.out.println("auf der Liste: "
                    + this.einkaeufe);
                break;
            }
        }
    }
}
```

Erster Ansatz: Schwer testbare Lösung (3/5)

```
import java.util.Scanner;
import java.util.regex.Pattern;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import com.github.stefanbirkner.systemlambda.SystemLambda;

public class DialogTest {
    private Dialog dialog;

    @BeforeEach
    public void setUp(){
        this.dialog = new Dialog();
    }
}
```

Erster Ansatz: Schwer testbare Lösung (4/5)

```
@Test
```

```
public void einElementTestGehtNicht() throws Exception{
```

```
    String[] inputs = {"1", "Pony"};
```

```
    SystemLambda.withTextFromSystemIn(inputs);
```

```
    this.dialog.start(); // Problem
```

nächste
genutzte
Eingaben

lese Ausgaben als
normalen String

```
    String out = SystemLambda.tapSystemOutNormalized(() -> {
```

```
        SystemLambda.withTextFromSystemIn(new String[]{"2"});
```

```
        this.dialog.start();
```

```
    });
```

```
    Assertions.assertTrue(Pattern.matches("(?s).*Pony.*", out)
```

```
        , "kein Pony auf der Liste: "+ out);
```

```
}
```

Erster Ansatz: Schwer testbare Lösung (5/5)

- Testmöglichkeit, aber recht unflexibel, es müssen zuerst alle gewünschten Eingaben kombiniert werden

@Test

```
public void einElementTest() throws Exception{
    String[] inputs = {"1", "Pony", "2", "0"}; // vollstaendig
    String out = SystemLambda.tapSystemOutNormalized(() -> {
        SystemLambda.withTextFromSystemIn(inputs)
            .execute(() -> {
                this.dialog.start();
            });
    });
    Assertions.assertTrue(Pattern.matches("(?s).*Pony.*"
        , out)
        , "kein Pony auf der Liste: "+ out);
}
```

Optimierung mit Refactoring (1/4) – in Dialog

```
switch(eingabe) { // nur umgebauter zweiter Teil
    case 1: {
        this.neuerEintrag();
        break;
    }
    case 2:{
        this.ausgeben();
        break;
    } } } }

public void neuerEintrag(){
    System.out.print("auf Liste: ");
    String neu = this leseString();
    this.einkaeufe.add(neu);
}

public void ausgeben(){
    System.out.println("auf der Liste: " + this.einkaeufe);
}
```

Optimierung mit Refactoring (2/4) – in Testklasse

```
public void eingeben(String wunsch) throws Exception {
    String[] inputs = {wunsch};
    SystemLambda.withTextFromSystemIn(inputs)
        .execute(() -> {
            this.dialog.neuerEintrag();
        });
}
```

```
public String ausgabe() throws Exception {
    return SystemLambda.tapSystemOutNormalized(() -> {
        SystemLambda.withTextFromSystemIn(new String[] {"2"})
            .execute(() -> {
                this.dialog.ausgeben();
            });
    });
}
```

Optimierung mit Refactoring (3/4)

@Test

```
public void leereListeTest() throws Exception{
    String ausgabe = this.ausgabe();
    Assertions.assertTrue(Pattern.matches("(?s).*\\[\\].*"
        , ausgabe)
        , "Liste sollte leer sein: " + ausgabe);
}
```

@Test

```
public void einElementTest2() throws Exception{
    this.eingeben("Pony");
    String ausgabe = this.ausgabe();
    Assertions.assertTrue(Pattern.matches("(?s).*Pony.*"
        , ausgabe)
        , "kein Pony auf der Liste: "+ ausgabe);
}
```

@Test

```
public void mehrereElementeTest() throws Exception{
    this.eingeben("Pony");
    this.eingeben("Hase");
    this.eingeben("Pony");
    this.eingeben("Hase");
    String ausgabe = this.ausgabe();
    Assertions.assertTrue(Pattern.matches(
        "(?s).*Pony.*Pony.*", ausgabe)
        , "Liste sollte zwei Ponys haben: "+ ausgabe);
    Assertions.assertTrue(Pattern.matches(
        "(?s).*Hase.*Hase.*", ausgabe)
        , "Liste sollte zwei Hasis haben: "+ ausgabe);
}
```


- vorheriges Beispiel zeigt, dass in der Entwicklung an die Testbarkeit gedacht werden muss
- alle komplexen Berechnungen und Zugriffe auf externe Systeme in eigene Methoden auslagern
- vorheriges Beispiel zeigt, dass Tests selbst eine Architektur haben müssen
- im Beispiel: Erstellung von Hilfsmethoden mit sinnvollen Namen, so das eigentliche Tests lesbarer werden
- im nächsten Schritt kann es eigene Hilfsklassen geben
- sinnvoll auch genutzte Frameworks so zu kapseln, dass ein Austausch möglich wird (z. B. wenn JUnit eigene Funktionalität ergänzt)

- Erinnerung: Spezifikation mit Features und Szenarien
- Danach keine direkte Implementierung sondern Übergang zur Modellierung
- zunächst erste Überlegungen zur Architektur (Schichten, Modularität)
- dann Klassen ableiten, Entitäts-Klassen, auch Steuerungsklassen
- diese Klassen dann als Mocks realisieren
- Mocks sollten Szenarien erfüllen
- - > man erhält ersten Indikator, ob Design sinnvoll und umsetzbar

7. Test von WebServices



- Ideen des Systemtests
- WebServices
- JSON
- Test von WebServices

Teststufen (grober Ablauf)

entwicklungs-
intern

Klassentest

typisch: macht
Entwickelnde selbst

hier Webservice-
Test auf eigenem
Rechner

Integrationstest

sollten Entwickelnde
nicht selbst machen
(z. B. eigenes Scrum-
Team-Mitglied)

hier Webservice-
Test auf externen
Rechner / Docker

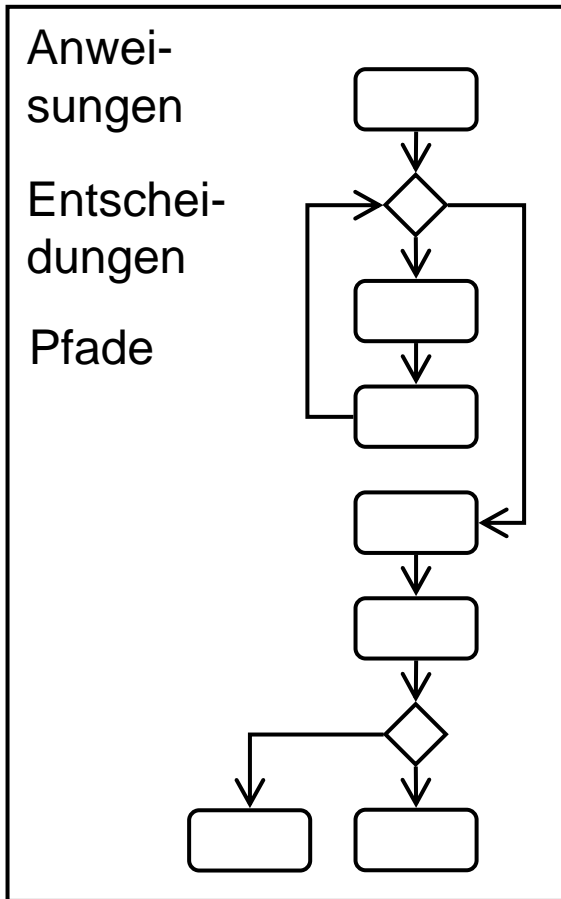
Systemtest

teilweise
identisch, Kunde
traut AN

entwicklungs-
extern (mit Kunden)

Abnahmetest

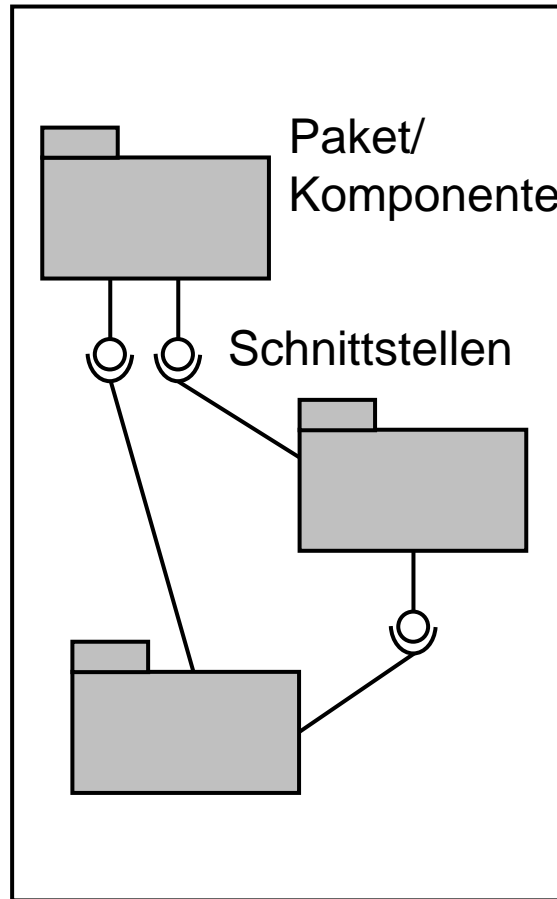
White-Box-Test



Methoden-/Klassentest

Software-Qualität

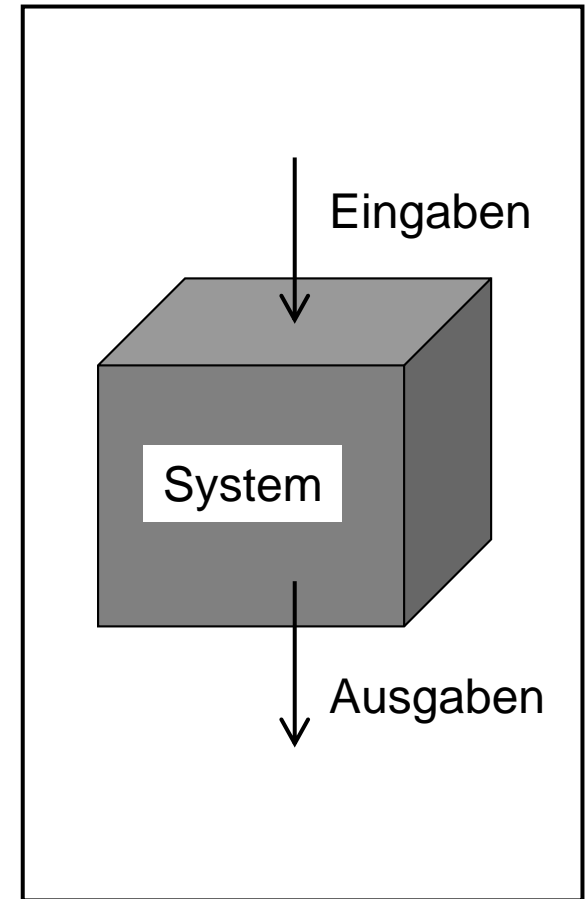
Gray-Box-Test



Integrationstest

Stephan Kleuker

Black-Box-Test



Systemtest

- über Web-Protokoll, z. B. HTTP, ausführbare Dienste, die
 - Informationen liefern
 - Berechnungen durchführen
 - Informationen speichern/ prüfen/ verwalten
- stehen über erreichbaren Rechner (virtuelle Maschine, Docker) zur Verfügung
- Sicherheitsaspekte relevant [in dieser VL nicht]
- Hier nicht: Services über SOAP, MQTT, WebSockets
- stateful (Server kennt Nutzer (z. B. User-Token), weiß, was vorher passiert ist)
- stateless (Server kennt Nutzer nicht; aber Zustandsinformationen können mitgeschickt werden)

- Representational State Transfer
- nutzt HTTP-Protokoll zur Bearbeitung von Ressourcen, die durch URLs identifiziert werden
- GET: Lesen von Informationen
- POST: Erstellen von neuen Informationen (Details im Body der Nachricht enthalten)
- PUT: Verändern vorhandener Informationen (Body-Nutzung)
- DELETE: Löschen von Informationen
- GET <http://ein.server/studi/42> (lese Studi mit Id 42)
- Parameter (42) meist in Pfadinformation, Attribute nutzbar
- Anmerkung Jakarta EE (JEE) bietet dafür mächtige, einfach nutzbare Bibliotheken , u. a. Java™ API for RESTful Web Services (JAX-RS), Java™ API for JSON Processing

- JavaScript Object Notation (<http://json.org/>)
- textuelles Austauschformat, abgeleitet aus JavaScript

```
{ "name": "Tony Stark",  
  "alter": 42,  
  "firma": { "name": "Stark Industries",  
            "ort": "New York, N.Y"  
  },  
  "freunde":["Steve Rogers", "Bruce Banner"]  
}
```
- Sammlung von
 - (Name: Wert)-Paaren
 - Arrays von Werten
- Werte können wieder aus beiden Elementen bestehen

- gibt in jeder Sprache viele Bibliotheken (Java in JEE eingebaut); hier GSON (<https://github.com/google/gson>)
- serialisierbares POJO (parameterloser Konstruktor)

```
public class Firma {  
    private String name ;  
    private String ort;  
    public Firma() {}
```

```
public class Person {  
    private String name;  
    private int alt;  
    private Firma firma;  
    private List<String> freunde;  
    public Person() {}
```

```
// bel. weitere Konstruktoren/ Methoden/ hier get, set, eq
```

```
public static void main(String[] args) {
    Person p = new Person("X", 52, new Firma("HS", "OS"),
        null);
    Gson gson = new GsonBuilder().serializeNulls().create();
    String json = gson.toJson(p);
    System.out.println(json);
    Person p2 = gson.fromJson(json, Person.class);
    System.out.println(p2 + "\n" + (p == p2)
        + "\n" + (p.equals(p2)));
}
```

```
{"name":"X","alt":52,"firma":{"name":"HS","ort":"OS"},
"freunde":null}
Person [name=X, alt=52, firma=Firma [name=HS, ort=OS],
freunde=null]
false
true
```

GSON-Nutzung (2/2)

String daten =

```
{ \"name\": \"Tony Stark\",  
  + \"alter\": 42,  
  + \"firma\": { \"name\": \"Stark Industries\",  
  + \"ort\": \"New York, N.Y\"  
  + },  
  + \"freunde\": [\"Steve Rogers\", \"Bruce Banner\", 42]  
  + }";
```

```
Person p3 = gson.fromJson(daten, Person.class);
```

```
System.out.println(p3);
```

```
}  
  
Person [name=Tony Stark, alt=0, firma=Firma  
[name=Stark Industries, ort=New York, N.Y],  
freunde=[Steve Rogers, Bruce Banner, 42]]
```

SuT (System under Test)

- Gegeben sei Webservice mit zwei Diensten
- GET <http://localhost:8080/teiler/42>
- zwei ein int-Parameter – Ergebnis ist übergebener Wert und Liste aller Teiler {"Wert":42,"Teiler":[1,2,3,6,7,14,21,42]}
- POST <http://localhost:8080/addierer>
im Body übertragener Wert {"Wert":42} wird zum Addierer addiert, Ergebnis ist neuer Addierer-Stand und übertragener letzter Wert {"Hinzu":42,"Addierer":84}
- Antwort hat immer einige Standard- und optionale Attribute
- Status (200 = OK, 201 = Created, 400 = Bad Request
404 = Not Found, ...)

<https://de.wikipedia.org/wiki/HTTP-Statuscode>

- Auswahl der Technologie um Tests durchzuführen
 - hier: in Java selbst machbar, Aufgabe: Erstellung von Hilfsklassen, um eigentliche Ausführung zu vereinfachen
 - JEE: Arquillian (Integrationstest-Framework)
 - generelles Ziel: möglichst Technologie-unabhängiges Framework, das bei Testerstellung den Fokus auf Testdaten erlaubt
- Ableitung der Testfälle
 - unabhängig von Technologie, mit bekannten Ansätzen (Äquivalenzen, Grenzwerte, etc.)
- Umsetzung der Testfälle in der passenden Umgebung (hier: JUnit)

Hilfsmethode für GET (1/2)

```
// in Klasse Execute
```

```
public static String executeGet(String path)
                                throws IOException {
    HttpURLConnection conn = null;
    try {
        URL url = new URI("http://localhost:8080/" + path).toURL();
        System.out.println(url);
        conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");
        conn.setRequestProperty("Accept", "application/json");
        if (conn.getResponseCode() != 200) { // 200 OK
            throw new RuntimeException("" + conn.getResponseCode());
        }
    }
}
```

Hilfsmethode für GET (2/2)

```
InputStreamReader in
    = new InputStreamReader(conn.getInputStream());
BufferedReader br = new BufferedReader(in);
String text = ""; // besser StringBuffer
String output;
while ((output = br.readLine()) != null) {
    text += output;
}
return text;
} finally {
    if (conn !=null) {
        conn.disconnect();
    }
}
}
```

- Services schicken JSON-Objekte als Ergebnis oder/und erwarten sie als Parameter
- ein direkter Weg: erstelle für jedes mögliche JSON-Objekt eine Klasse mit get- und set-Methoden

```
public class Teiler {  
    private int Wert; // leider gross wegen Attributsname  
    private ArrayList<Integer> Teiler;
```

```
    public Teiler() {}
```

```
    public int getWert() {  
        return Wert;  
    }
```

```
    ...
```


Beispielnutzung executeGet

```
public static void main(String[] args) {
    try {
        String json = Execute.executeGet("teiler/42");
        System.out.println(json);
        Gson gson = new GsonBuilder().serializeNulls().create();
        Teiler t = gson.fromJson(json, Teiler.class);
        System.out.println(t);
    } catch (Exception e) {
        System.out.println("e: " + e.getMessage());
    }
}
http://localhost:8080/teiler/42
{"Wert":42,"Teiler":[1,2,3,6,7,14,21,42]}
Teiler [Wert=42, Teiler=[1, 2, 3, 6, 7, 14, 21, 42]]

try {
    System.out.println(Execute.executeGet("teiler/42x"));
} catch (Exception e) {
    System.out.println("e: " + e.getMessage());
}
http://localhost:8080/teiler/42x
e: 400
}
```

Hilfsmethode für POST (1/2)

```
public static String executePost(String path, String content)
                                throws IOException {
    HttpURLConnection conn = null;
    try {
        URL url = new URI("http://localhost:8080/" + path).toURL();
        System.out.println(url);
        conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("POST");
        conn.setRequestProperty("Content-Type"
                                , "application/json; utf-8");
        conn.setRequestProperty("Accept"
                                , "application/json");
        conn.setDoOutput(true);
        OutputStream os = conn.getOutputStream();
        byte[] input = content.getBytes("utf-8");
        os.write(input, 0, input.length);
    }
}
```

Hilfsmethode für POST (2/2)

```
    if (conn.getResponseCode() != 201) { // 201 created
        throw new RuntimeException("" + conn.getResponseCode());
    }
// System.out.println("Loc:"+conn.getHeaderField("Location"));
    InputStreamReader in
        = new InputStreamReader(conn.getInputStream());
    BufferedReader br = new BufferedReader(in);
    String text = "";
    String output;
    while ((output = br.readLine()) != null) {
        text += output;
    }
    return text;
} finally {
    if (conn != null) {
        conn.disconnect();
    }
}
}
```

```
public class Wert {  
    private int Wert;
```

```
    public Wert() {}
```

```
    ...
```

```
public class Addierer {  
    private int Hinzu;  
    private int Addierer;
```

```
    public Addierer() {}
```

```
    ...
```

Beispielnutzung executePOST (1/2)

```
public static void main(String[] args) {
    try {
        Gson gson = new GsonBuilder().serializeNulls().create();
        Wert w = new Wert(42);
        System.out.println(gson.toJson(w));
        Execute.executePost("addierer", gson.toJson(w));
        String json = Execute.executePost("addierer"
                                          , gson.toJson(w));

        System.out.println(json);
        Addierer a = gson.fromJson(json, Addierer.class);
        System.out.println(a);
    } catch (Exception e) {
        System.out.println("e: " + e.getMessage());
    }
}
```

```
{"Wert":42}
http://localhost:8080/addierer
http://localhost:8080/addierer
{"Hinzu":42,"Addierer":420}
Addierer [Hinzu=42, Addierer=420]
```

Beispielnutzung executePOST (2/2)

```
try {
    System.out.println(Execute.executePost("addierer"
                                           , "{\\"Wert\\":x}"));
} catch (Exception e) {
    System.out.println("e: " + e.getMessage());
}
}
```

```
http://localhost:8080/addierer
e: 400
```

```
// module-info.path
open module qsWebServiceTest {
    exports test;
    exports entity;
    exports example;

    requires com.google.gson;
    requires org.junit.jupiter.api;
}
```

Test der WebServices (1/3)

```
public class SystemTest {  
    private Gson gson  
        = new GsonBuilder().serializeNulls().create();  
  
    @Test  
    public void testTeilerOk() throws IOException {  
        String json = Execute.executeGet("teiler/42");  
        Teiler t = this.gson.fromJson(json, Teiler.class);  
        Assertions.assertIterableEquals(  
            Arrays.asList(1,2,3,6,7,14,21,42), t.getTeiler());  
    }  
  
    @Test  
    public void testTeilerOkErgebnisLeer() throws IOException {  
        String json = Execute.executeGet("teiler/0");  
        Teiler t = this.gson.fromJson(json, Teiler.class);  
        Assertions.assertIterableEquals(  
            new ArrayList<Integer>(), t.getTeiler());  
    }  
}
```

Test der WebServices (2/3)

```
@Test
public void testTeilerNok() throws IOException {
    try {
        Execute.executeGet("teiler/101");
        Assertions.fail();
    } catch (RuntimeException e) {
        Assertions.assertEquals("400", e.getMessage());
    }
}

@Test
public void testAddiererNok() throws IOException {
    try { //Fehler: kleines w bei wert
        Execute.executePost("addierer", "{\"wert\":42}");
        Assertions.fail();
    } catch (RuntimeException e) {
        Assertions.assertEquals("400", e.getMessage());
    }
}
```



```
@Test
public void testAddiererOk() throws IOException {
    // es gibt keinen Reset-Service
    Wert w = new Wert(0);
    String json = Execute.executePost("addierer"
                                     , gson.toJson(w));
    Addierer a = gson.fromJson(json, Addierer.class);
    int start = a.getAddierer();
    json = Execute.executePost("addierer"
                              , gson.toJson( new Wert(42)));
    a = gson.fromJson(json, Addierer.class);
    Assertions.assertEquals(42, a.getHinzu());
    Assertions.assertEquals(start + 42, a.getAddierer());
}
}
```

wichtiges Testwerkzeug Postman

The screenshot displays the Postman REST client interface. At the top, the request method is set to 'POST' and the URL is 'http://localhost:8080/addierer'. The 'Body' tab is selected, showing a JSON payload:

```
{ 1: { 2: "Wert": 42 3: }
```

. Below the request, a list of links is shown, including <https://www.postman.com/>. The response section at the bottom shows a status of '201 Created' and a JSON response:

```
{ 1: { 2: "Hinzu": 42, 3: "Addierer": 462 4: }
```

- <https://www.postman.com/> (auch Chrome-Erweiterung)

wichtiges Testwerkzeug curl

- <https://curl.haxx.se/download.html>
- wesentlich mächtiger als „nur“ Command-Line

```
C:\Users\x>
C:\Users\x>curl -H "Accept: application/json" http://localhost:8080/teiler/42
{"Wert":42,"Teiler":[1,2,3,6,7,14,21,42]}
C:\Users\x>curl -X POST -H "Content-Type: application/json, Accept: applicatio
n/json" -d "{\"Wert\":42}" "http://localhost:8080/addierer"
{"Hinzu":42,"Addierer":756}
C:\Users\x>curl -X POST -H "Content-Type: application/json" -d "{\"Wert\":42}"
http://localhost:8080/addierer
{"Hinzu":42,"Addierer":798}
C:\Users\x>
```

8. Test von Software mit Nutzungsoberflächen

Video 10

- Erinnerung: Testebenen
- Ansätze zum Oberflächentest
- Kompakt: GUI-Aufbau mit Swing
- Einführung in AssertJ/FEST
- Systematische Nutzung von AssertJ/FEST
- Teststrategien
- Capture & Replay
- Vorstellung Sikuli
- Test von Web-Oberflächen mit Selenium

Teststufen (grober Ablauf)

Klassentest

typisch: macht
Entwickelnde selbst

Integrationstest

sollten Entwickelnde
nicht selbst machen
(z. B. eigenes Scrum-
Team-Mitglied)

Systemtest

teilweise
identisch, Kunde
traut AN

Abnahmetest

entwicklungs-
intern

hier GUI-
Test

entwicklungs-
extern (mit Kunden)

- Grundsätzlich sind Oberflächen gewöhnliche SW-Module und können wie diese getestet werden
- Variante 1: Direkte Ausführung von GUI-Aktionen über Testsoftware
- Allerdings: Für xUnit-Werkzeuge ist der Zugriff auf Oberflächen teilweise schwierig (xUnit muss auf Oberflächenkomponenten zugreifen und diese bedienen können, JButton in Java hat z.B. Methode `doClick()`)
- folgende Folien: AssertJ/FEST (Fixtures for Easy Software Testing, <http://code.google.com/p/fest/>), hier GUI-Anteil
- Variante 2: Das Werkzeug zeichnet alle Mausbewegungen und Tastatureingaben auf, die können dann zur Testwiederholung erneut abgespielt werden (später)

- Viele elementare GUI-Bausteine (JLabel, JButton, JSlider, JTable, JColorChooser,...)
- Bausteine werden in GUI-Komponenten zusammengefasst
- Zusammenfassung typisch in JPanel
- Jede Zusammenfassung hat LayoutManager, der Anordnung der einzelnen Komponenten berechnet
- Komponenten können in Komponenten geschachtelt werden; äußere Komponente hat wieder LayoutManager
- konsequentes Schachtel-in-Schachtel-Prinzip
- GUI-Baustein nutzt Model-Delegate-Prinzip; zu jeder Komponente gibt es Model-Objekt, das Eigenschaften speichert (JButton hat ein ButtonModel)
- GUI-Bausteine mit vielen get-/set-Methoden konfigurierbar

Achtung: Swing und Threads

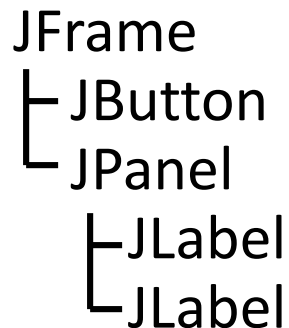
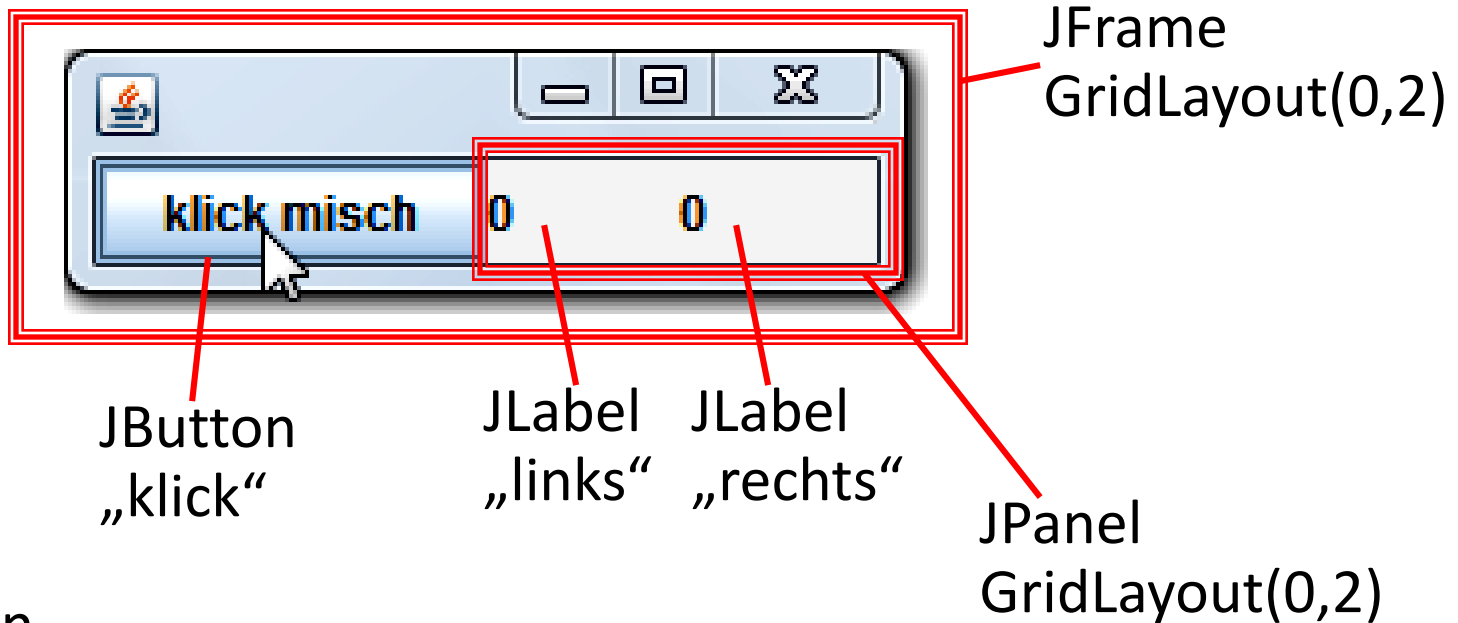
- Swing-Objekte wie JFrame-Objekte können direkt gestartet werden
- Event-Verwaltung findet in unabhängigen Prozessen statt (genauer: Threads -> Betriebssysteme)
- Problem: Komponenten reden mit noch nicht erzeugten Komponenten (-> NullPointerException)

```
javax.swing.SwingUtilities.invokeLater(new Runnable() {  
    @Override  
    public void run() {  
        new MeineGUI();  
    }  
});
```

- Anmerkung: Im Gegensatz zu JavaFX bleiben Swing und AWT Teile der Java SE Spezifikation (mind. 09/2026) [danach Bibliothek?]

Minibeispiel: Klickzähler (1/3)

- Zählt Links- und Rechtsklicks des Knopfes
- Für Test notwendig (bzw. sehr sinnvoll), GUI-Elemente haben Namen



GUI immer hierarchisch organisiert

Minibeispiel: Klickzähler (2/3)

```
public class Klicker extends JFrame implements MouseListener{
    private JButton klick = new JButton("klick misch");
    private JLabel links = new JLabel("0");
    private JLabel rechts = new JLabel("0");

    public Klicker(){
        super.setLayout(new GridLayout(0, 2));
        super.setName("klicker");
        JPanel tmp = new JPanel();
        tmp.setLayout(new GridLayout(0, 2));
        tmp.add(this.links);
        this.links.setName("links");           // Name
        tmp.add(this.rechts);
        this.rechts.setName("rechts");        // Name
        super.add(this.klick);
        this.klick.setName("klick");         // Name
        this.klick.addMouseListener(this);
        super.add(tmp);
        super.setDefaultCloseOperation(EXIT_ON_CLOSE);
        super.pack();
        super.setVisible(true);
    }
}
```

Minibeispiel: Klickzähler (3/3)



```
public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        @Override public void run() {
            new Klicker();
        }
    });
}

@Override
public void mouseClicked(MouseEvent e) {
    JLabel ziel = null;
    if(e.getButton() == 1)
        ziel = this.links;
    else
        if(e.getButton() == 3)
            ziel = this.rechts;
    if(ziel != null)
        ziel.setText(""+(Integer.parseInt(ziel.getText()) + 1));
}

@Override public void mouseEntered(MouseEvent arg0) {}
@Override public void mouseExited(MouseEvent arg0) {}
@Override public void mousePressed(MouseEvent arg0) {}
@Override public void mouseReleased(MouseEvent arg0) {}
}
```

Test des Minibeispiels (1/3)

```
@ExtendWith(GUITestExtension.class)
public class KlickerTest {
    private FrameFixture gui;
    private Klicker k;

    @BeforeAll
    public static void setUpOnce() { // für Event-Verwaltung
        FailOnThreadViolationRepaintManager.install();
    }

    @BeforeEach
    public void setUp() { // für Event-Verwaltung
        GuiActionRunner.execute(() -> {
            k = new Klicker();
            return k;
        });
        this.gui = new FrameFixture(k);
        // this.gui.robot().settings().delayBetweenEvents(3000);
        // this.gui.show(); // nicht immer erforderlich
    }
}
```

Test des Minibeispiels (2/3)

@After

```
public void tearDown() {  
    this.gui.cleanUp();  
}
```

@Test

```
public void testEinLinksklick(){  
    this.gui.button("klick").click();  
    Assertions.assertEquals(this.gui.label("links").text(), "1");  
    Assertions.assertEquals(this.gui.label("rechts").text(), "0");  
}
```

@Test

```
public void testRechtsklick(){  
    this.gui.button("klick").click(MouseButton.RIGHT_BUTTON);  
    Assertions.assertEquals(this.gui.label("links").text(), "0");  
    Assertions.assertEquals(this.gui.label("rechts").text(), "1");  
}
```

Test des Minibeispiels (3/3)

```
@Test
```

```
public void testKeineLinksUndRechtsklicks(){  
    this.gui.button("klick").click(MouseButton.MIDDLE_BUTTON);  
    this.gui.button("klick").pressAndReleaseKey(  
        KeyPressInfo.keyCode(KeyEvent.VK_C));  
    Assertions.assertEquals(this.gui.label("links").text(), "0");  
    Assertions.assertEquals(this.gui.label("rechts").text(), "0");  
}
```

```
@Test
```

```
public void testVieleLinksUndRechtsklicks(){  
    this.gui.button("klick").doubleClick();  
    this.gui.button("klick").pressAndReleaseKey(  
        KeyPressInfo.keyCode(KeyEvent.VK_SPACE));  
    for(int i=0;i<10;i++)  
        this.gui.button("klick").click(MouseButton.RIGHT_BUTTON);  
    Assertions.assertEquals(this.gui.label("links").text(), "2");  
    Assertions.assertEquals(this.gui.label("rechts").text(), "10");  
}
```

- Einiger konstanter Aufwand um Eventverwaltung zu organisieren
- einfacher Zugriff auf GUI-Elemente wg. setName(.)-Werten
- FrameFixture-Objekt erlaubt Zugriff auf (fast) alle Arten von GUI-Elementen, die über Namen identifiziert werden
`gui.button("klick")`
- Jedes Fixture-Objekt bietet (fast) alle Möglichkeiten zur Bedienung an `click(MouseButton.MIDDLE_BUTTON);`
- Fixture-Objekte haben Methoden zur Eigenschaftsabfrage
`gui.label("rechts").text()`

- AssertJ ist generell Bibliothek für Java-Zusicherungen (Standard-Java, Guava, Joda Time, relationale Datenbanken, Swing):
<https://assertj.github.io/doc/#assertj-overview>
- Swing: <https://assertj.github.io/doc/#assertj-swing>
- basiert auf FEST - seit 2013 nicht weiterentwickelt; funktioniert aber stabil (Apache License 2.0) , allerdings nur bis JUnit 4
- AssertJ in JUnit5 nutzbar, benötigt Starterklasse, wird beschrieben in <https://stackoverflow.com/questions/70631626/assertj-swing-and-junit-5-support>

```
open module qsGUIEinstieg { // module-info.java
    exports assertj5;
    exports main;
    requires java.desktop;
    requires org.junit.jupiter.api;
    requires assertj.swing.junit; // unstable derived from the
    requires assertj.swing;      // module's filename
}
```


AssertJ erkennt Probleme: Schreipfehler

```
gui.button("klack").click(MouseButton.RIGHT_BUTTON);
```

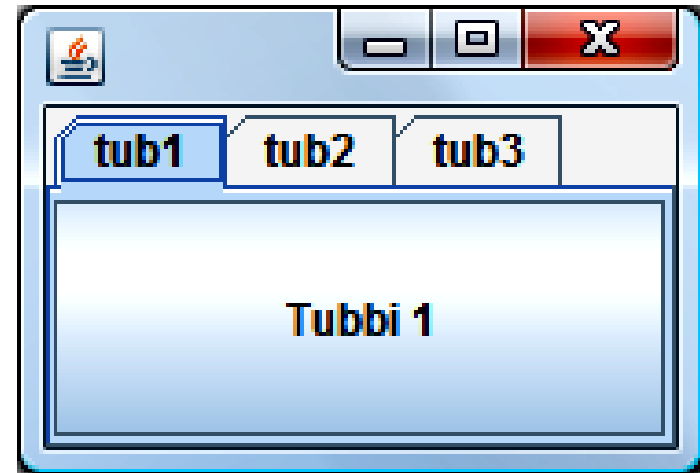
```
org.assertj.swing.exception.ComponentLookupException: Unable to find  
component using matcher org.assertj.swing.core.NameMatcher[name='klack',  
type=javax.swing.JButton, requireShowing=true].
```

Component hierarchy:

```
main.Klicker[name='klicker', title='', enabled=true, visible=true,  
showing=true]  
  javax.swing.JRootPane[]  
    javax.swing.JPanel[name='null.glassPane']  
      javax.swing.JLayeredPane[]  
        javax.swing.JPanel[name='null.contentPane']  
          javax.swing.JButton[name='klick', text='klick misch',  
enabled=true, visible=true, showing=true]  
        javax.swing.JPanel[name=null]  
          javax.swing.JLabel[name='links', text='0', enabled=true,  
visible=true, showing=true]  
          javax.swing.JLabel[name='rechts', text='0', enabled=true,  
visible=true, showing=true]
```

AssertJ erkennt Probleme: nicht sichtbarer Knopf (1/3)

```
public class Tapps extends JFrame {  
    public Tapps(){  
        JTabbedPane jt = new JTabbedPane();  
        jt.setName("tabbed");  
        for(int i=1;i<=3;i++){  
            JButton jb= new JButton("Tubbi "+i);  
            jb.setName("tub"+i);  
            jt.add(jb);  
            add(jt);  
        }  
        setSize(200, 100);  
        setVisible(true);  
    }  
    // main mit invokeLater  
}
```



AssertJ erkennt Probleme: nicht sichtbarer Knopf (2/3)

```
@ExtendWith(GUITestExtension.class)
public class TappsTest {
    private FrameFixture gui;

    @BeforeClass
    public static void setUpOnce() {
        FailOnThreadViolationRepaintManager.install();
    }

    @Before public void setUp() {
        Tapps t = GuiActionRunner.execute(new GuiQuery<Tapps>() {
            protected Tapps executeInEDT() {
                return new Tapps();
            }
        });
        this.gui = new FrameFixture(t);
        this.gui.tabbedPane("tabbed").selectTab("tub1");
    }

    @After public void tearDown() { this.gui.cleanUp(); }
```

garantiert auch,
dass GUI aktiv ist

AssertJ erkennt Probleme: nicht sichtbarer Knopf (3/3)

```
@Test public void testTabwechsel(){
    this.gui.tabbedPane("tabbed").selectTab("tub2");
    this.gui.button("tub2").rightClick();
}
```

```
@Test
public void testTabwechsel2(){
    this.gui.tabbedPane("tabbed").selectTab("tub3");
    this.gui.button("tub2").rightClick();
}
```

```
org.assertj.swing.exception.ComponentLookupException:
  Unable to find component using matcher
  org.assertj.swing.core.NameMatcher[name='tub2',
  type=javax.swing.JButton, requireShowing=true].
```

```
Component hierarchy: ...
```

```
javax.swing.JButton[name='tub2', text='Tubbi 2',
  enabled=true, visible=false, showing=false]
```

```
...
```

- Zu entwickeln ist ein kleines Programm zur Berechnung von Tarifen im Nahverkehr. Grundlage der Entwicklung ist folgende Spezifikation zur Tarifberechnung, die insgesamt vier Parameter berücksichtigt. Der erste Parameter steht für die Anzahl der zu fahrenden Zonen, der zweite für das Alter, der dritte für die Uhrzeit (die Stunde) und der vierte bestimmt, ob es sich um einen Betriebszugehörigen handelt oder nicht. Der Preis wird wie folgt berechnet: Jede Zone kostet 130, Personen unter 14 und über 64 zahlen 40 weniger, von 9-14.59 Uhr ist der Preis um die Hälfte reduziert, Betriebszugehörige (und deren Angehörige) zahlen 150 weniger, allerdings wird der 9-14.59 Uhr-Tarif nicht zusätzlich berücksichtigt. Es soll aber immer der günstigste Preis (entweder/oder) ausgegeben werden. Der minimale Ticketpreis ist mit 30 festgelegt.

Spezifikation – Tarifrechner (2/2)



x Zonen	$130 * x$
Alter < 14 oder Alter > 64	- 40
9-14.59 Uhr, nicht zum Betrieb	- 50%
nicht 9-14.59 Uhr, zum Betrieb	-150
9-14.59, zum Betrieb	- 50% oder -150
Minimum: 30	

Beispiel: ein 13-jähriger „Betriebszugehöriger“

fährt 3 Zonen um 10 Uhr

Alternative 1: $((130*3) - 40) / 2 = 175$

Alternative 2: $(130*3) - 40 - 150 = 200$

Der Preis beträgt 175.

Spezifikation – Äquivalenzklassen

Nummer	1	2	3	4	5	6
Klassen	(0) (1o) (4o) (7)	(0) (2u) (5u) (8)	(0) (2o) (5o) (7)	(0) (3u) (6u) (8)	(9)	„Bauchgefühl“
Zonen	1	2	1	2	„Hai“	3
Alter	13	14	64	65	64	13
Uhrzeit	8	9	14	15	15	8
Betrieb	true	false	true	false	false	true
Ergebnis	30	130	30	220	130	175

Zonen: (0) Zahl (9) keine Zahl (= Default-Wert 1)

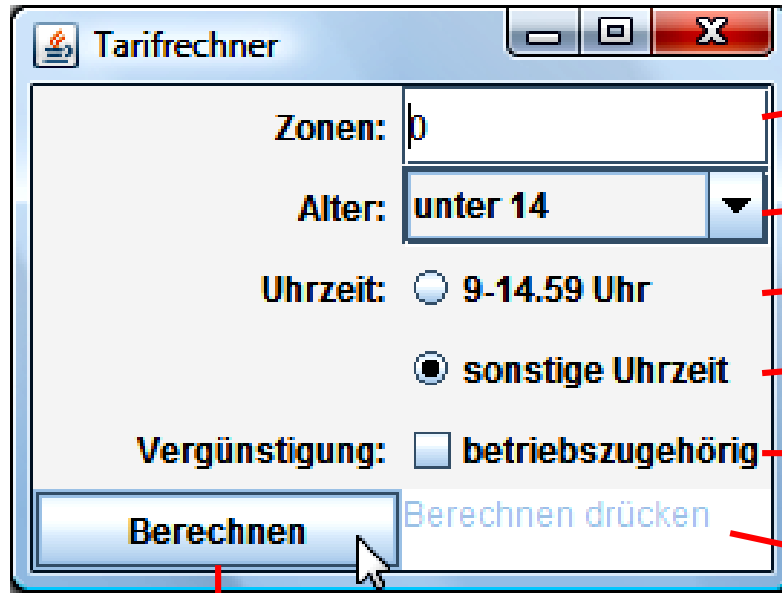
Alter: (1) $alter < 14$ (2) $alter \geq 14 \ \&\& \ alter \leq 64$ (3) $alter > 64$

Uhrzeit: (4) $uhr < 9$ (5) $uhr \geq 9 \ \&\& \ uhr < 15$ (6) $uhr \geq 15$

Betriebszugehörig: (7) Ja (8) Nein

Beispiel systematische AssertJ-Nutzung (1/7)

- GUI planen; festlegen der GUI-Elemente und ihrer Namen
- Beispiel Tarifzonen-Berechnung



JTextField „zonen“

JComboBox „alter“

JRadioButton „billigUhr“

JRadioButton „teuerUhr“

JCheckBox „zumBetrieb“

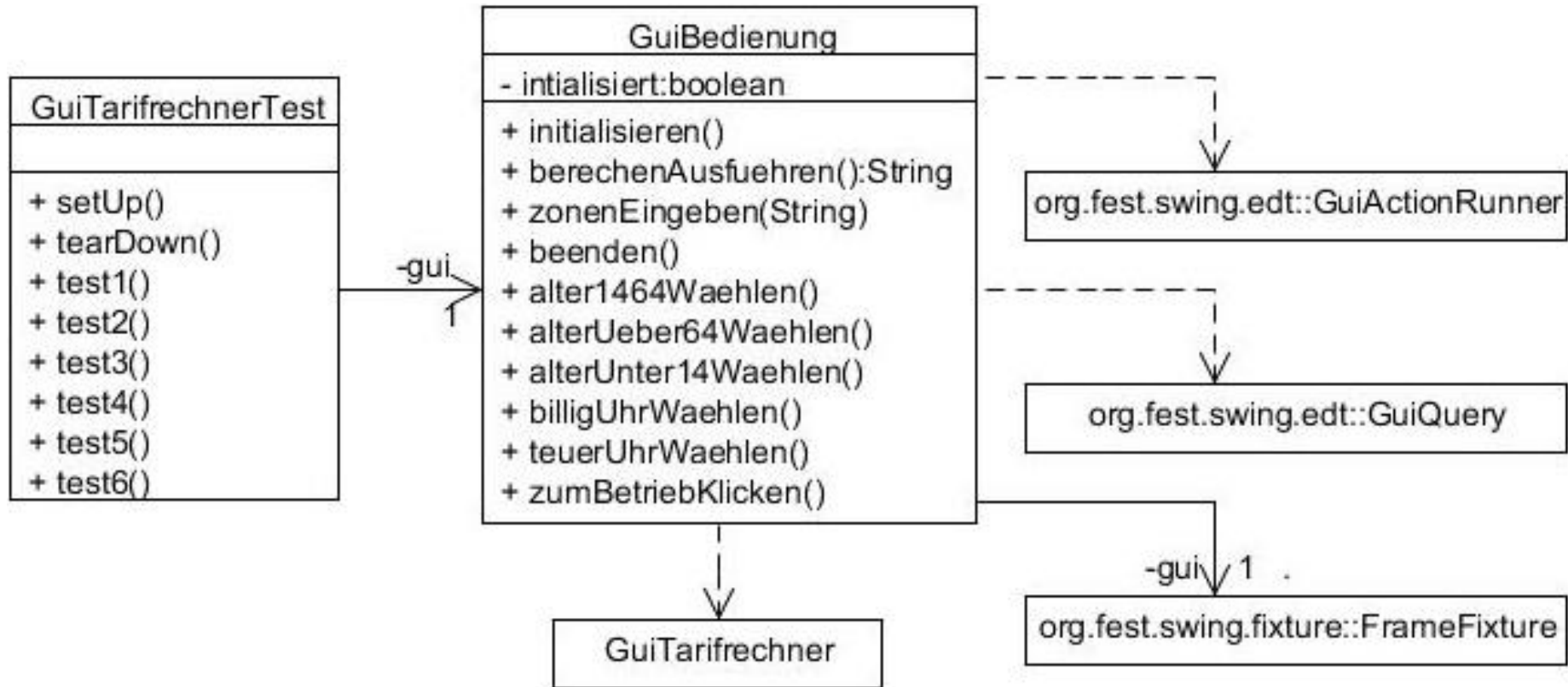
JTextArea „preis“

JButton „berechnen“

- Personen, die Tests schreiben, müssen weitere Implementierungsdetails nicht kennen (black box)

Beispiel systematische AssertJ-Nutzung (2/7)

- Testarchitektur (schnelle Reaktion auf Änderungen)



Beispiel systematische AssertJ-Nutzung (3/7)

```
public class GUIBedienung {
    private FrameFixture gui;
    private static boolean initialisiert = false;

    public void initialisieren() {
        if(!initialisiert){
            FailOnThreadViolationRepaintManager.install();
            this.initialisiert = true;
        }
        GuiTarifrechner gtf = GuiActionRunner
            .execute(new GuiQuery<GuiTarifrechner>() {
                protected GuiTarifrechner executeInEDT() {
                    return new GuiTarifrechner();
                }
            });
        this.gui = new FrameFixture(gtf);
    }
}
```

Beispiel systematische AssertJ-Nutzung (4/7)

```
public void beenden() {
    this.gui.cleanUp();
}

public void zonenEingeben(String eingabe){
    this.gui.textBox("zonen").deleteText();
    this.gui.textBox("zonen").enterText(eingabe);
}

public String berechnenAusfuehren(){
    this.gui.button("berechnen").click();
    return this.gui.textBox("preis").text();
}

public void alterUnter14Waehlen(){
    this.gui.comboBox("alter").selectItem("unter 14");
}

public void alterUeber64Waehlen(){
    this.gui.comboBox("alter").selectItem("\u00fcber 64");
}
}
Software-Qualität                Stephan Kleuker                379
```

Beispiel systematische AssertJ-Nutzung (5/7)

```
public void alter1464Waehlen(){  
    this.gui.comboBox("alter").selectItem("14-64");  
}
```

```
public void billigUhrWaehlen(){  
    this.gui.radioButton("billigUhr").click();  
}
```

```
public void teuerUhrWaehlen(){  
    this.gui.radioButton("teuerUhr").click();  
}
```

```
public void zumBetriebKlicken(){  
    this.gui.checkBox("zumBetrieb").click();  
}
```

Beispiel systematische AssertJ-Nutzung (6/7)

```
// in GuiTarifrechnerTest
private GUIBedienung gui; // in setUp() initialisieren
@Test
public void test1() {
    this.gui.zonenEingeben("1");
    this.gui.alterUnter14Waehlen();
    this.gui.teuerUhrWaehlen();
    this.gui.zumBetriebKlicken();
    Assertions.assertTrue(this.gui.berechnenAusfuehren()
        .equals("30 Cent"));
}
@Test
public void test2() {
    this.gui.zonenEingeben("2");
    this.gui.alter1464Waehlen();
    this.gui.billigUhrWaehlen();
    Assertions.assertTrue(this.gui.berechnenAusfuehren()
        .equals("130 Cent"));
}
```

Beispiel systematische AssertJ-Nutzung (7/7)

```
@Test
public void test3() {
    this.gui.zonenEingeben("1");
    this.gui.alter1464Waehlen();
    this.gui.billigUhrWaehlen();
    this.gui.zumBetriebKlicken();
    Assertions.assertTrue(this.gui.berechnenAusfuehren()
        .equals("30 Cent"));
}
```

```
@Test
public void test4() {
    this.gui.zonenEingeben("2");
    this.gui.alterUeber64Waehlen();
    this.gui.teuerUhrWaehlen();
    Assertions.assertTrue(this.gui.berechnenAusfuehren()
        .equals("220 Cent"));
}
```

- entwickelnde Personen verpflichtet setName() zu nutzen (Programm läuft ohne; gibt auch in AssertJ andere Methoden zur Suche nach GUI-Elementen im Swing-Baum)
- Auslagerung der GUI-Nutzung in eigene Klasse sinnvoll
- Nur wenige Testersteller müssen so AssertJ kennen
- Bei mehreren Masken sollten Tests Masken zugeordnet werden können

- Achtung, im Beispiel werden praktisch alle Tests über die Oberfläche durchgeführt; schöner:
 - teste Business-Logik getrennt
 - teste nur Zugriffsmöglichkeiten über die Oberfläche

Generelle Frage nach der Teststrategie

- bottom-up
 - zunächst elementare Klassen, dann Klassen die darauf aufbauen
 - nur Zusammenspiel der getesteten Objekte testen
 - bringt sehr hohe Überdeckung, aber sehr hohen Aufwand
- top-down
 - Tests von oben, typisch über Oberfläche ausführen
 - benötigt stabiles GUI für Testwiederholung
 - vermeintlich weniger Aufwand, weniger Expertise
 - häufig sehr geringe Testabdeckung

- middle-out
 - Tests setzen auf kleinen Gruppen von Klassen auf
 - GUI-Tests testen Verbindungen zu Gruppen
 - erbt anteilig Vor- und Nachteile der anderen Ansätze
- Variante
 - zunächst Tests über GUI durchführen und Überdeckung messen
 - Klassen die unter xx% überdeckt werden, dann in Richtung Überdeckungsgrenze genauer analysieren
- generell hängt Test-Strategie von Projektart, vorhandenen Werkzeugen und vorhandenem Know-how ab

GUI-Elementsuche auch ohne setName

- Generell können „Matcher“ eingesetzt werden, mit denen die nutzende GUI-Komponente bestimmt werden kann

@Test

```
public void testEineZone2() {
    gui.textBox(JTextComponentMatcher
        .withName("zonen").andText("0")).deleteText();
    gui.textBox("zonen").enterText("1");
    GenericTypeMatcher<JButton> textMatcher =
        new GenericTypeMatcher<JButton>(JButton.class) {
            @Override protected boolean isMatching(JButton button) {
                return "Berechnen".equals(button.getText());
            }
        };
    gui.button(textMatcher).focus().click();
    Assert.assertEquals(gui.textBox("preis").text(), "90 Cent");
}
```

Video 11

- statt Objekt direkt zu nutzen, kann auch Programmstart über main genutzt und Fenster später gesucht werden

```
public static void main(final String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
            Klicker k= new Klicker();
            if (args.length > 0)
                k.setTitle(args[0]);
        }
    });
}
```

Start von Programmen über main (2/2)

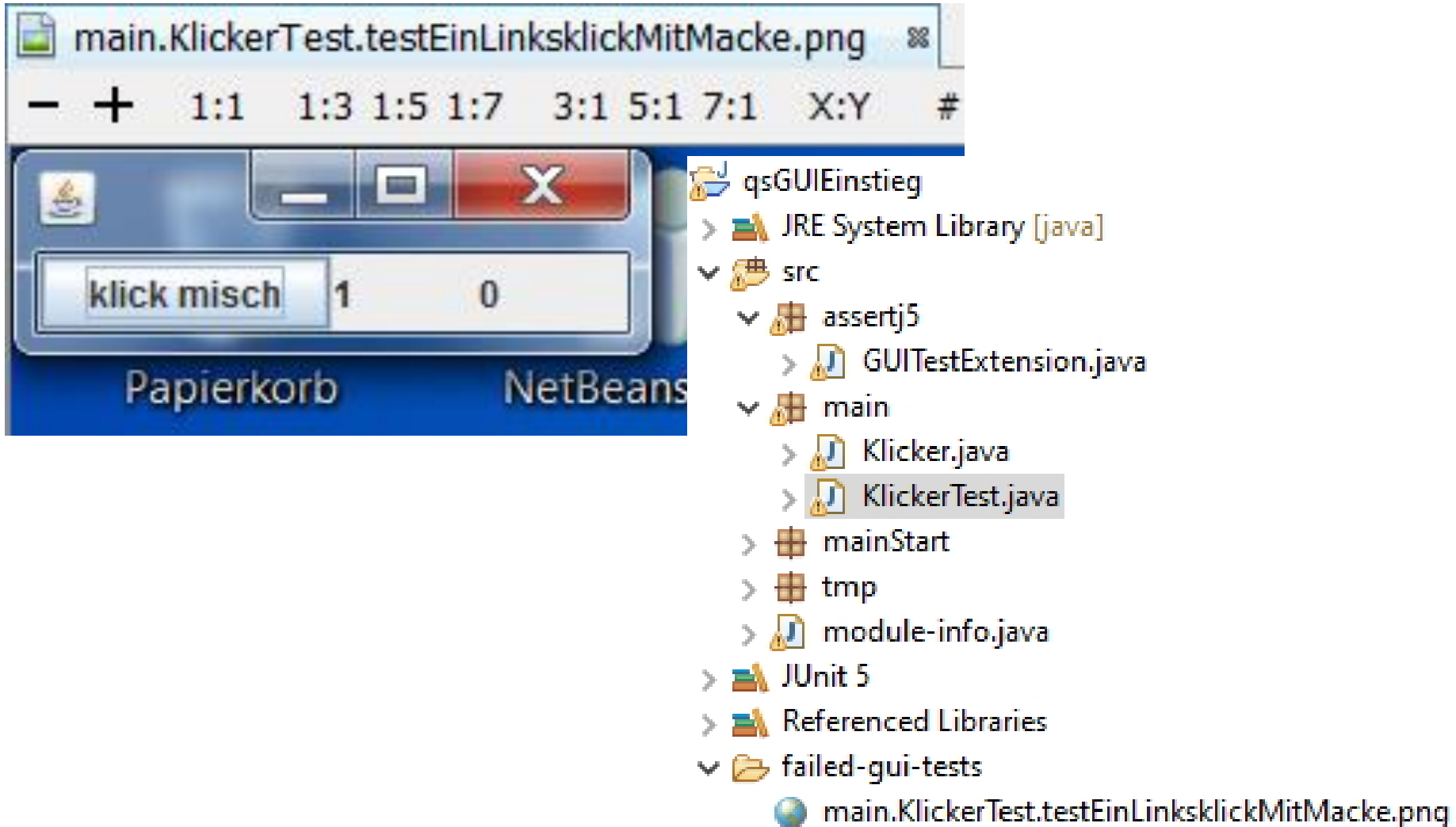
```
@Test
public void testEinLinksklick2(){
    ApplicationLauncher.application("klicker.Klicker")
        .withArgs("Hai", "Wo").start();
    FrameFixture frame = WindowFinder.findFrame(
        new GenericTypeMatcher<JFrame>(JFrame.class) {
            protected boolean isMatching(JFrame frame) {
                return "Hai".equals(frame.getTitle())
                    && frame.isShowing();
            }
        })
        .using(gui.robot);
    frame.focus().button("klick").click();
    Assert.assertEquals(frame.label("links").text(), "1");
    Assert.assertEquals(frame.label("rechts").text(), "0");
}
```

```
@ExtendWith(GUITestExtension.class)
public class KlickerTest {
    // wie sonst auch

    @GUITest @Test
    public void testEinLinksklickMitMacke(){
        gui.button("klick").click();
        Assert.assertEquals(gui.label("links").text(), "1");
        Assert.assertEquals(gui.label("rechts").text(), "1");
    }
}
```

Fotos von Fehlersituationen (2/2)

- Werkzeug macht Screenshot (des gesamten Bildschirms)

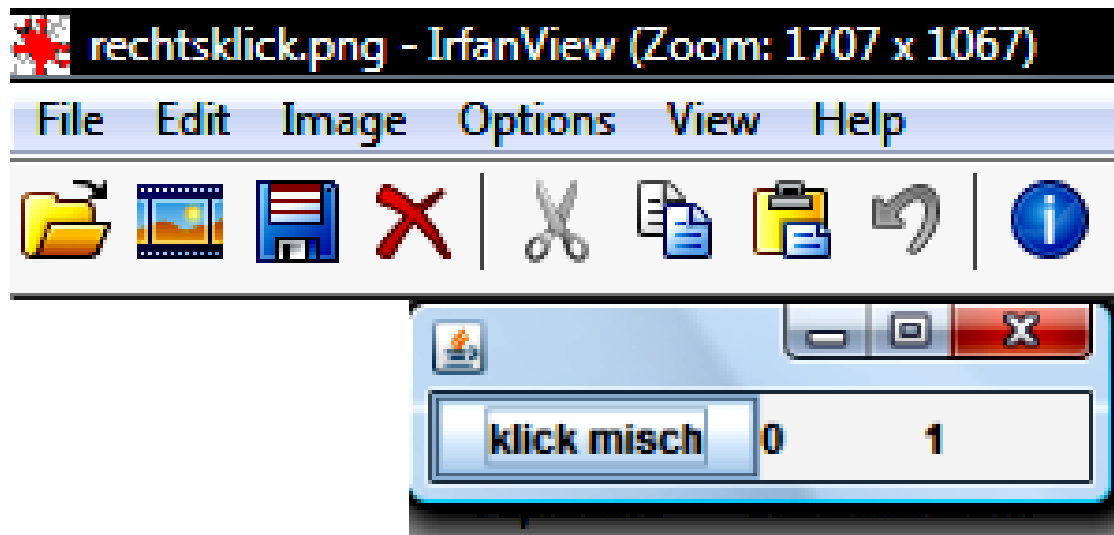


Variante: Bildschirmfotos erstellen lassen

- auch für einzelne Komponenten möglich

@Test

```
public void testMachScreenshot(){  
    ScreenshotTaker st= new ScreenshotTaker();  
    gui.button("klick").rightClick();  
    st.saveDesktopAsPng("rechtsklick.png");  
    Assert.assertEquals(gui.label("rechts").text(), "1");  
}
```



- Vorteile
 - relativ einfache Programmierung; Zugang sehr intuitiv
 - Tests können ohne weitere Werkzeuge in JUnit umgesetzt werden
 - leichte GUI-Änderungen (Layout) benötigen keine Teständerungen
- Nachteile
 - Entwickelnde müssen konsequent setName() nutzen (ok mit Coding-Guidelines)
 - (Testentwicklung bei Capture & Replay schneller)
 - Testende müssen Java können
 - generelles zentrales Problem: Änderungen des GUIs können zu aufwändigen Teständerungen führen

Variante 2:

- Grundidee: Das Werkzeug zeichnet alle Mausbewegungen und Tastatureingaben auf, die können dann zur Testwiederholung erneut abgespielt werden
- Typisch ist, dass der Nutzer die aufgezeichneten Skripte modifizieren kann (z. B. Test von berechneten Daten)
- Tools können teilweise auch Oberfläche lesen (Frage ob Texte richtig ausgegeben), Snapshots vergleichen
- professionelle Beispiele: Winrunner von HP (früher Mercury), VisualTest von IBM-Rational

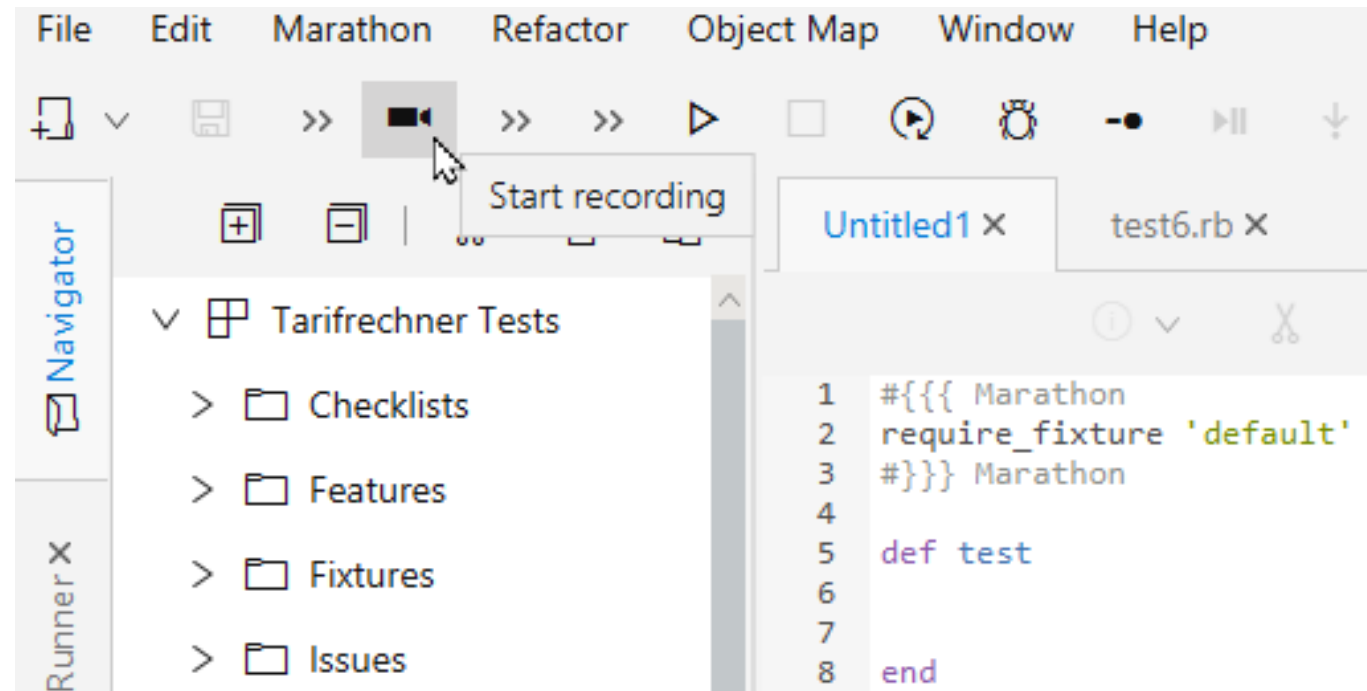
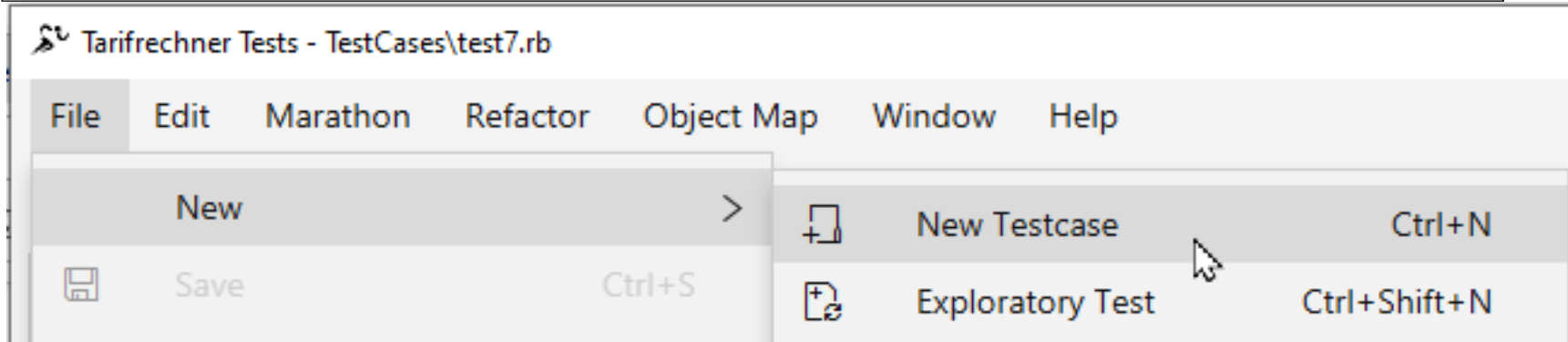
- Open Source (hier genutzt: 5.5.5.0)
<https://sourceforge.net/projects/marathonman/>
- Marathon erlaubt die Aufzeichnung und Wiedergabe von Swing- und JavaFX-basierten Oberflächen
- Aufzeichnungen erfolgen in Skriptsprache (Jython, JRuby)
- Zusicherungen und weitere Analysen werden ebenfalls in dieser Skriptsprache ergänzt
- Werkzeug erlaubt Erstellung einfacher Zusicherungen ohne Skriptsprachenkenntnisse
- gibt kommerzielle Version MarathonITE

- Error läuft nur mit Java 8

```
Error: You need to use Java version >= 1.8.0_112 and Below 9.0 (Expected: <= 1.8.0_999 Actual: 21.0.1)
marathon [options...] [project-directory [test...]]
```



Marathon-Beispiel (1/6) – Start einer Aufnahme



Marathon-Beispiel (2/6) - Aufnahme

The screenshot displays two windows side-by-side. The left window, titled 'Tarifrechner', contains the following form elements: 'Zonen: 5' (text input), 'Alter: über 64' (dropdown menu), 'Uhrzeit: 9-14.59 Uhr' (radio button), and 'sonstige Uhrzeit' (radio button). Below these is a checkbox for 'Vergünstigung: betriebszugehörig' and a 'Berechnen' button. The right window, titled 'Marathon Control Center', features a code editor with the following script:

```
with_window("Tarifrechner") {  
  select("Alter", "5")  
  select("alter", "über 64")  
  select("9-14.59 Uhr", "true")  
}
```

Below the code editor is a toolbar with several icons. A mouse cursor is hovering over a square icon, which has triggered a tooltip that says 'Stop recording'. An orange arrow points from this tooltip down towards the text below the screenshot.

Aufnahme immer hier beenden
Alle Klicks und Eingaben rechts werden links aufgezeichnet

Marathon-Beispiel (3/6) – resultierendes Skript

- Tests in Ruby, weitere Programmierung so möglich
- Wichtige Befehle

```
assert_p('<Name des GUI-Elements>', '<Property>'  
        , '<erwarteter Wert>')  
wait_p("preis", "Text"  
        , "175 Cent")
```

Property aus

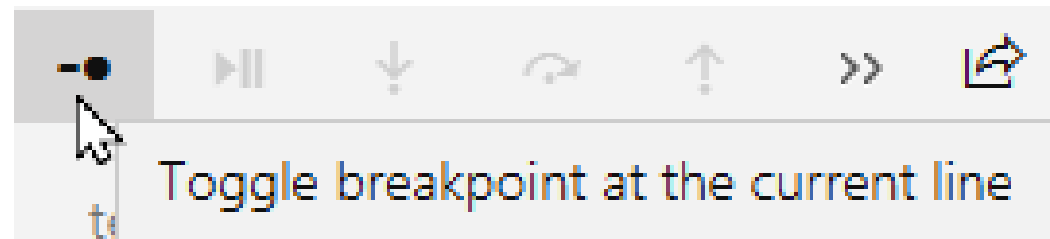
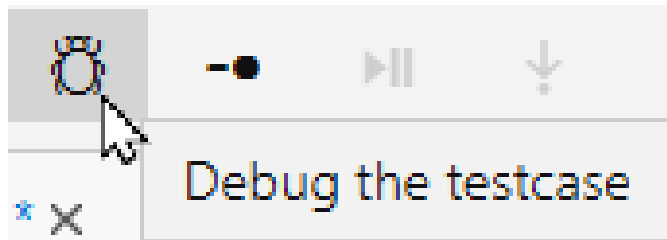
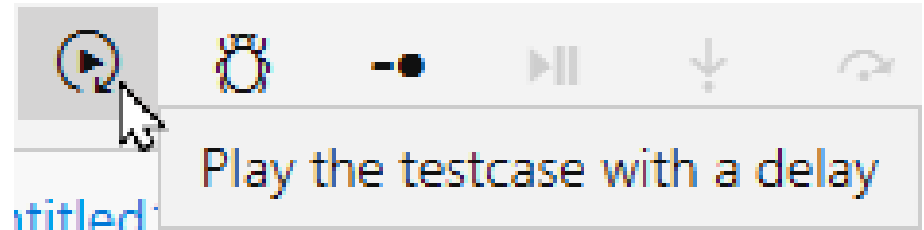
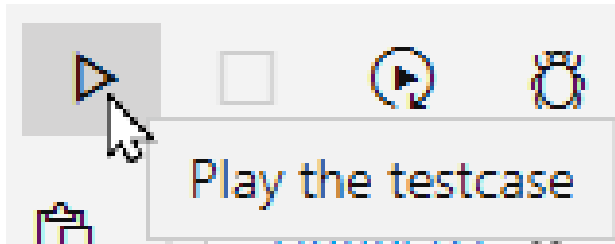
Java-Bean (mit

get und set

bearbeitbar)

```
erstesBeispielFolien24.rb* x  test6.rb x  
1  #{{{ Marathon  
2  require_fixture 'default'  
3  #}}} Marathon  
4  
5  severity("normal")  
6  
7  def test  
8    with_window("Tarifrechner") {  
9      select("Alter", "5")  
10     select("alter", "über 64")  
11     select("9-14.59 Uhr", "true")  
12     click("Berechnen")  
13   }  
14 end
```

Marathon-Beispiel (4/6) – Ausführen des Skripts



1 error

Message	File	Location
Assertion failed for property: Texton component = ({... expected = `175 Cent` actual = `305 Cent`	F:\workspaces\ekli...	line 13 in funcion

< >

Results Record & Playback Log Output

Marathon-Beispiel (5/6) – Aufnahme mit Zusicherung

The screenshot displays three overlapping windows from a software testing tool:

- Tarifrechner:** A form with fields for 'Zonen: 03', 'Alter: über 64', 'Uhrzeit: 9-14.59 Uhr' (selected), and 'Vergünstigung: betriebszugehörig' (checked). A 'Berechnen' button is visible, and the result '175 Cent' is shown below it.
- Marathon Control Center:** A code editor containing the following script:

```
select("alter", "über 64")
select("9-14.59 Uhr", "true")
select("betriebszugehörig", "true")
click("Berechnen")
}
```
- Component Inspector:** A window titled '{name=preis, type=JTextArea} ()' showing a tree of assertions for the 'preis' field. The 'Text {175 Cent}' assertion is selected. Other assertions include 'Background', 'Border', 'Enabled', and 'FieldName'. Below the tree, the value '175 Cent' is displayed, and buttons for 'Insert Wait' and 'Insert Assertion' are visible.

während Aufnahme:
Strg+Rechtsklick auf
zu untersuchendes
Element

Assertion auswählen

Marathon-Beispiel (6/6) – weiteres Beispielskript

```
#{#{# Marathon
require_fixture 'default'
#}}# Marathon

name("MitAssert")
severity("normal")

def test

  with_window("Tarifrechner") {
    select("Alter", "05")
    select("alter", "über 64")
    select("9-14.59 Uhr", "true")
    select("betriebszugehörig", "true")
    click("Berechnen")
    assert_p("preis", "Text", "305 Cent")
    assert_p("alter", "Text", "über 64")
  }

end
```


- Festlegung von Test-Fixtures, die beschreiben was vor und nach Tests immer ausgeführt werden soll
- Module, Programmstücke, die in andere Skripte eingebaut werden können und so eine Strukturierung der Testfälle ermöglichen
- Nutzung eines Debuggers, um ab einem Breakpoint die Ausführung Schritt für Schritt erfolgen zu lassen und ggfls. Objektwerte zu verändern
- „Data Driven Test“, Möglichkeit zur Nutzung von Sammlung von Eingabewerten
- detaillierte Aufnahme aller Aktionen, z. B.
`click("Alter", 1, 28, 10) // Pixelpositionen`
`keystroke("Alter", "Backspace")`
`keystroke("Alter", "1")`

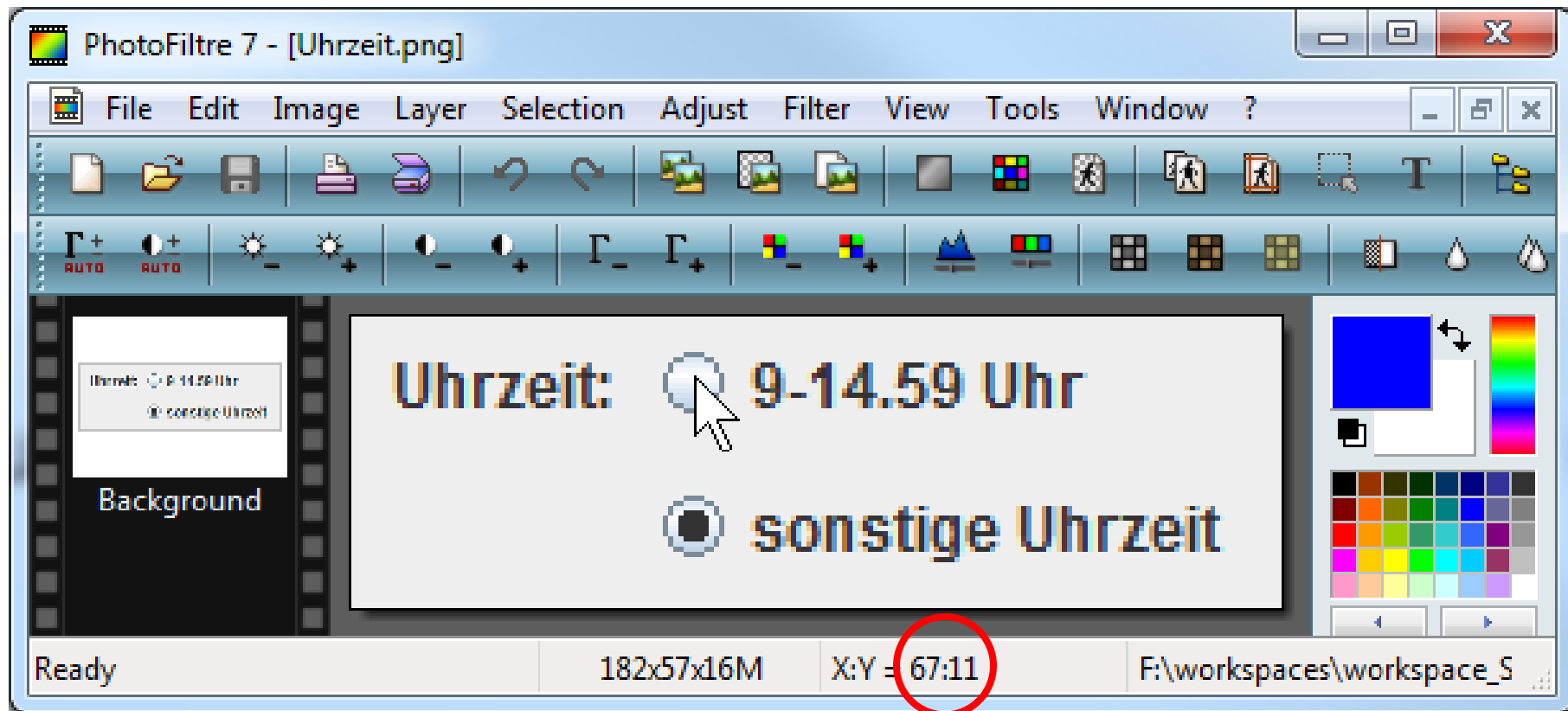
Variante: Automatisierte GUI-Steuerung

- bisherige Ansätze auf Java eingeschränkt
- generell: Capture & Replay gibt es, oft programmiersprachenabhängig, in vielen Varianten
- generell: Kosten und Nutzen bzgl. erwarteter Änderungen überlegen
- Variante: Software steuert die Bedienung anhand von Bildern
- Idee: Bildausschnitte werden zuerst fotografiert, dann bei Ausführung gesucht; Bildausschnitte können dann geklickt und über Tastatur gefüllt werden
- Testen: Prüfung, ob erwartete Bildausschnitte gefunden werden (unterstützt png und jpg)
- hier: SikuliX API (<http://www.sikulix.com/>) , auch <http://search.maven.org/#search|ga|1|g%3A%22com.sikulix%22>

- Nutzung hängt stark vom Testrechner ab (Geschwindigkeit, Auflösung, gewähltes GUI-Design, Anzahl angeschlossener Monitore)
- Ansatz: es gibt feste Testrechner, die zum Testen genutzt werden, mit klarer Konfiguration
- sinnvolle Variante: Nutzung virtueller Rechner, benötigt hier aber physikalischen Monitor
- keine durchsichtigen Fenster oder Ränder
- Prozessautomatisierung zur Testausführung (aufspielen, ausführen, Ergebnis einsammeln)
- Randbedingungen für Betriebssysteme beachten:
<http://sikulix.com/quickstart/>

Hilfsmittel

- Screencapturer, z. B. FastStone Capture (letzte freie Version)
- Bildanalyseprogramm mit Pixelangaben, z. B. Photofiltre



- ausgehend von links-oben muss relativ (67,11) geklickt werden

<http://www.portablefreeware.com/?id=775>

http://portableapps.com/apps/graphics_pictures/photofiltre_portable

- ursprünglich Sikuli, nach Entwicklungsübernahme Sikulix
- Sikulix unterstützt keine Module, d. h. in Projekten mit Modulen nicht direkt integrierbar
- kein Problem, da zum Systemtest meist andere Werkzeuge genutzt werden
- Startmöglichkeiten Sikulix:
 - direkt Javas-GUI aufrufen (dann ohne Module)
 - in eigenem Projekt aufrufen und schließen

```
App app = App.open("javaw -jar tarifrechner.jar");
app.close();
// s.type(Key.F4, Key.ALT); // Tastenkombination
```
 - über Bilder, Icon finden, doppelklicken, über Schließen-Button beenden

```
public static void main(String... arg){
    App app = App.open("javaw -jar tarifrechner.jar");
    Screen s = new Screen();
    try {
        Match m = s.find("bilder/ZoneAnklicken.png");
        System.out.println("Genauigkeit: " + m.getScore());
        Location loc = m.getCenter().right(-22).below(-18);
        s.click(loc);
        s.type("42");
        s.click("bilder/Berechnen.png");
    } catch (FindFailed ex) {
        System.out.println("Bild nicht gefunden");
    }
    app.close();
}
```

suche Bild

Klickpunkt festlegen, auch von Ecken aus möglich (getTopLeft())

klickt Mitte

<https://github.com/RaiMan/SikuliX1>

- Wenn Quellcode nicht vorliegt, kann so beliebiges Java-Programm (jar-Datei) gestartet werden

```
App app = App.open("javaw -jar tarifrechner.jar");
Screen s = new Screen();
```
- Steht der Quellcode zur Verfügung, ist auch direkter Start möglich, steht main-Methode in der Klasse main.Main, z. B.

```
main.Main.main(null);
Screen s = new Screen();
```
- bei Win10 kleinere Probleme; Meldungen der folgenden Form können ignoriert werden, wenn kein Absturz erfolgt 😊

```
[error] RobotDesktop: checkMousePosition: should be
L(960,540)@S(0)[0,0 1920x1080]
but after move is L(80,767)@S(0)[0,0 1920x1080]
Possible cause in case you did not touch the mouse while script
was running:
  Mouse actions are blocked generally or by the frontmost
application.
```

Hilfsprogramm zur Analyse (1/3)



```
import java.io.File;
import java.nio.file.DirectoryStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Iterator;
import java.util.Scanner;
import org.sikuli.script.App;
import org.sikuli.script.FindFailed;
import org.sikuli.script.Match;
import org.sikuli.script.Screen;

public class Analyse {

    private final String ORDNER ="bilder/";
```


Hilfsprogramm zur Analyse (2/3)

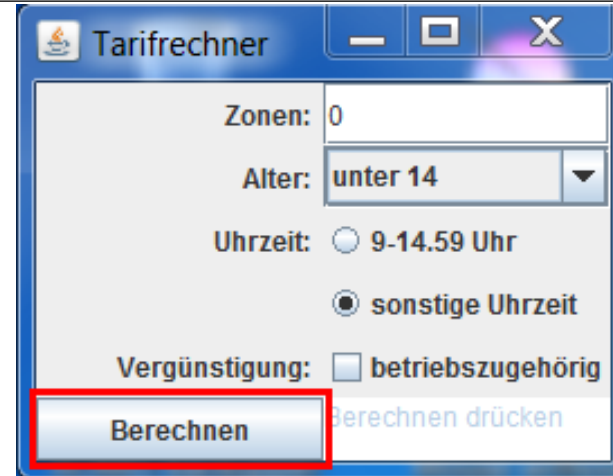
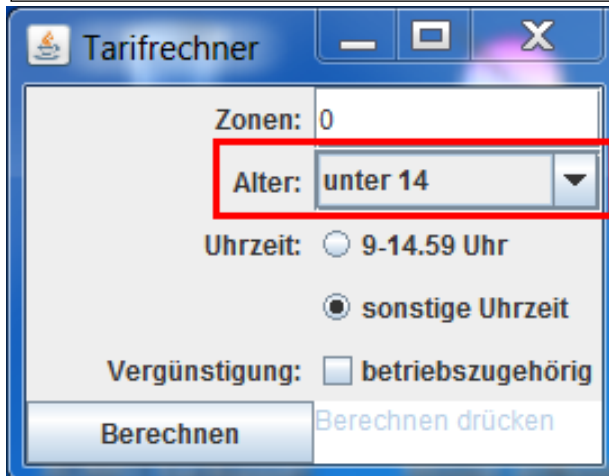
```
public void zeigeDetails(String dateiname) {
    System.out.println("suche: " + dateiname);
    Screen screen = new Screen();
    try {
        Iterator<Match> it = screen.findAll(dateiname);
        while(it.hasNext()){
            Match m = it.next();
            System.out.println(dateiname + ": " + m.getScore());
            m.highlight(4); // Rahmen drumherum 4 Sekunden
        }
        System.out.println("<Return> zum naechsten Bild");
        new Scanner(System.in).nextLine();
    } catch (FindFailed e) {
        System.out.println(dateiname + " nicht gefunden.");
    }
}
```

Hilfsprogramm zur Analyse (3/3)

```
public void alle(){
    Path dir = Paths.get(ORDNER);
    try (DirectoryStream<Path> stream = Files
        .newDirectoryStream(dir, "*.png")) {
        for (Path entry : stream) {
            this.zeigeDetails(entry.toString());
        }
    } catch (Exception e) {
        System.out.println(e);
    }
    app.close();
}

public static void main(String[] args) {
    new GuiTarifrechner(); // wenn ohne Module
    new Analyse().alle();
}
}
```

Schwellenwertanalyse (1/2)



suche: bilder\AlterAendern.png

bilder\AlterAendern.png nicht gefunden.

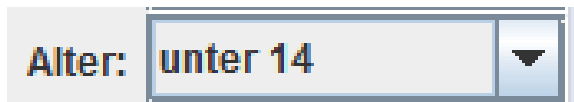
suche: bilder\AltersBox.png

bilder\AltersBox.png mit 0.9631317257881165

[log] highlight M[95,62 173x30]@S(S(0)[0,0 1920x960]) S:0,96

C:181,77 [-1 msec] for 4.0 secs

<Return> zum nächsten Bild



suche: bilder\Berechnen.png

bilder\Berechnen.png mit 0.9235213398933411

[log] highlight M[9,175 129x30]@S(S(0)[0,0 1920x960]) S:0,92

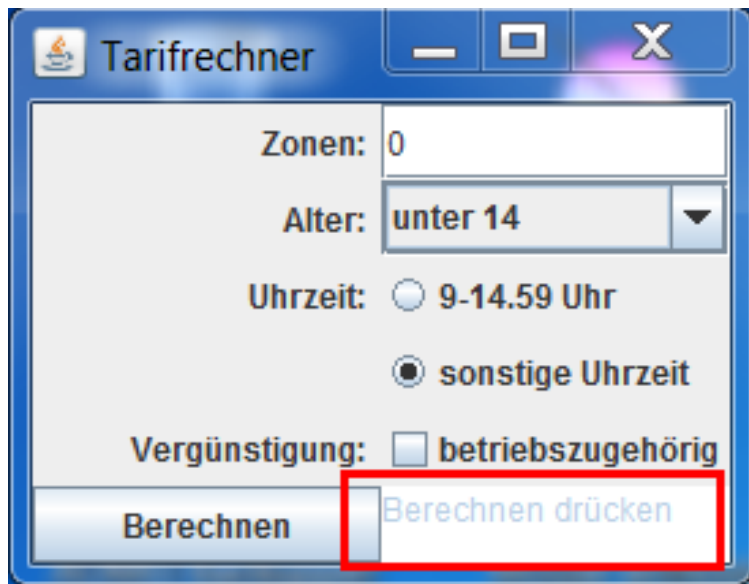
C:73,190 [-1 msec] for 4.0 secs

<Return> zum nächsten Bild



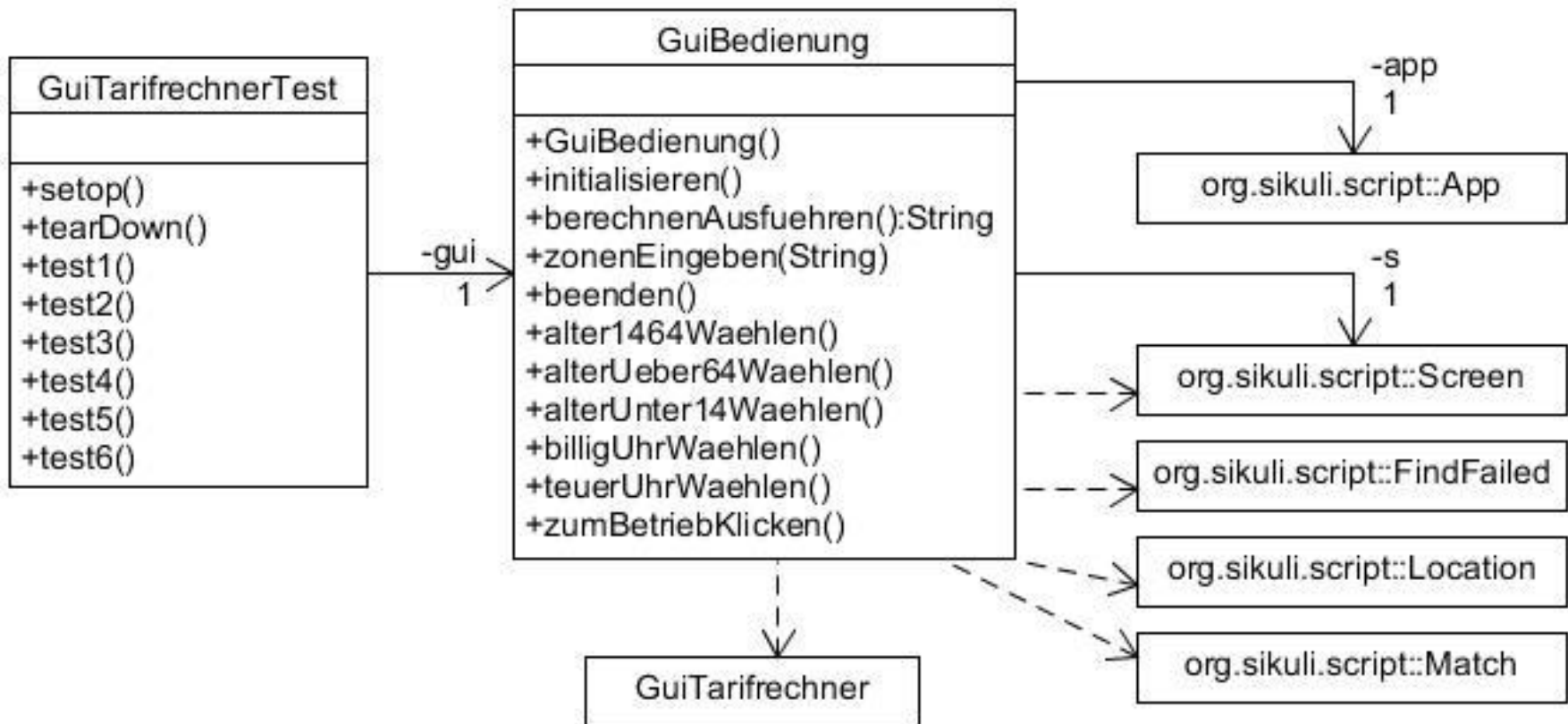
Schwellenwertanalyse (2/2)

- auch genau passende Ausschnitte liefern nicht unbedingt 1.0
- auch unpassende Ausschnitte liefern evtl. hohe Werte
- generell möglichst große, sich verändernde Flächen (hier recht klein)



```
suche: bilder\175Cent.png  
bilder\175Cent.png mit  
0.7370604276657104  
[log] highlight M[125,173  
135x31]@S(S(0)[0,0 1920x960])  
S:0,74 C:192,188 [-1 msec] for  
4.0 secs
```

Sikuli-Nutzung (1/10): Testarchitektur



Sikuli-Nutzung (2/10): benutzte Bilder

- Steuerung

Zonen: 0

Alter: unter 14

Alter: unter 14

Alter: unter 14

Uhrzeit: 9-14.59 Uhr
 sonstige Uhrzeit

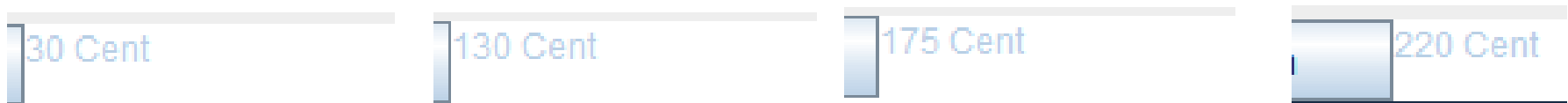
Vergünstigung: betriebszugehörig

Alter: unter 14

Uhrzeit: unter 14
über 64
14-64

Berechnen

- Ergebnisüberprüfung



Sikuli-Nutzung (3/10): zentrale Variablen

```
public class GUIBedienung {  
  
    private String verzeichnis = "bilder/";  
    private int offset = 0; // aendern bei mehreren Monitoren  
    private Screen s;  
    private App app; // nicht genutzt, nur angedeutet
```

Anmerkung: Sikulix-IDE hat ein Positionierungsproblem, wenn mehr als ein Monitor angeschlossen ist und der erste Monitor nicht der Hauptbildschirm ist



Sikuli-Nutzung (4/10): mehrere Monitore / Starten

```
public GUIBedienung() { // nur bei mehreren Monitoren
    GraphicsEnvironment ge = GraphicsEnvironment
        .getLocalGraphicsEnvironment();
    GraphicsDevice[] gs = ge.getScreenDevices();
    GraphicsConfiguration[] gc = gs[0].getConfigurations();
    System.out.println(gc[0].getBounds().getX());
    if (gc[0].getBounds().getX() > 0) {
        this.offset = (int) gc[0].getBounds().getX();
    }
}

public void initialisieren() {
    this.s = new Screen();
    this.app = App.open("javaw -jar tarifrechner.jar");
}
```


Sikuli-Nutzung (5/10): Texteingabe

```
public void beenden() {  
    this.app.close();  
}
```

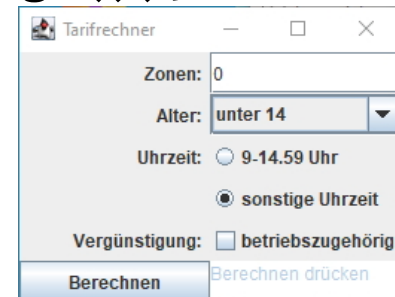
```
public void zonenEingeben(String eingabe) {  
    try {  
        Match m = s.find(verzeichnis + "ZoneAnklicken.png");  
        Location loc = m.getCenter().right(-22).below(-18);  
        this.s.click(loc);  
        this.s.type(eingabe);  
    } catch (FindFailed e) {  
        throw new IllegalArgumentException(e.getMessage());  
    }  
}
```



Sikuli-Nutzung (6/10): Drop-Down

```
public void alterUnter14Waehlen() {
    try {
        this.s.click(this.s.find(verzeichnis + "AltersBox.png")
            .getCenter().right(71).below(-2));
        this.s.click(this.s.find(verzeichnis + "AlterAendern.png")
            .getCenter().right(5).below(-6));
    } catch (FindFailed e) {
        throw new IllegalArgumentException(e.getMessage());
    }
}
```

```
public void alterUeber64Waehlen() {
    try {
        this.s.click(s.find(this.verzeichnis + "AltersBox.png")
            .getCenter().right(71).below(-2));
        this.s.click(this.s.find(verzeichnis + "AlterAendern.png")
            .getCenter().right(0).below(12));
    } catch (FindFailed e) {
        throw new IllegalArgumentException(e.getMessage());
    }
}
```



Sikuli-Nutzung (7/10): GUI-Elemente ansteuern

```
public void billigUhrWaehlen() {
    try {
        this.s.click(this.s.find(verzeichnis + "Uhrzeit.png")
            .getCenter().right(-24).below(-15));
    } catch (FindFailed e) {
        throw new IllegalArgumentException(e.getMessage());
    }
}
```

```
public void zumBetriebKlicken() {
    try {
        this.s.click(this.s
            .find(verzeichnis + "Verguenstigung.png")
            .getCenter().right(9).below(5));
    } catch (FindFailed e) {
        throw new IllegalArgumentException(e.getMessage());
    }
}
```



Sikuli-Nutzung (8/10): Ergebnis finden



```
public String berechnenAusfuehren() {
    try {
        this.s.click(verzeichnis + "Berechnen.png");
    } catch (FindFailed e) {
        throw new IllegalArgumentException(e.getMessage());
    }
    Match m = this.s.exists(verzeichnis + "175Cent.png");
    if (m != null && m.getScore() > 0.74) {
        return "175 Cent";
    }
    m = this.s.exists(verzeichnis + "130Cent.png");
    if (m != null && m.getScore() > 0.75) {
        return "130 Cent";
    }
    m = this.s.exists(verzeichnis + "220Cent.png");
    if (m != null && m.getScore() > 0.738) {
        return "220 Cent";
    }
    ... // 30 Cent
    throw new IllegalArgumentException("Unerwartetes Ergebnis");
}
```

Sikuli-Nutzung (9/10): Tests (1/2) wiederverwandt

```
// in GuiTarifrechnerTest
private GUIBedienung gui; // in setUp() initialisieren
@Test
public void test1() {
    this.gui.zonenEingeben("1");
    this.gui.alterUnter14Waehlen();
    this.gui.teuerUhrWaehlen();
    this.gui.zumBetriebKlicken();
    Assertions.assertTrue(this.gui.berechnenAusfuehren()
        .equals("30 Cent"));
}
@Test
public void test2() {
    this.gui.zonenEingeben("2");
    this.gui.alter1464Waehlen();
    this.gui.billigUhrWaehlen();
    Assertions.assertTrue(this.gui.berechnenAusfuehren()
        .equals("130 Cent"));
}
```

Sikuli-Nutzung (10/10): Tests (2/2) wiederverwandt

```
@Test
public void test3() {
    this.gui.zonenEingeben("1");
    this.gui.alter1464Waehlen();
    this.gui.billigUhrWaehlen();
    this.gui.zumBetriebKlicken();
    Assertions.assertTrue(this.gui.berechnenAusfuehren()
        .equals("30 Cent"));
}
```

```
@Test
public void test4() {
    this.gui.zonenEingeben("2");
    this.gui.alterUeber64Waehlen();
    this.gui.teuerUhrWaehlen();
    Assertions.assertTrue(this.gui.berechnenAusfuehren()
        .equals("220 Cent"));
}
```

Testergebnis

```
@BeforeEach  
public void setUp() throws Exception {  
    this.gui = new GUIBedienung();  
    this.gui.initialisieren();  
}
```

```
@AfterEach  
public void tearDown() throws Exception {  
    this.gui.beenden();  
}
```

Finished after 51,498 seconds

Runs: 6/6 ❌ Errors: 0 ❌ Failures: 0

GuiTarifrechnerTest [Runner: JUnit 5] (51,266 s)

- test1() (11,741 s)
- test2() (7,750 s)
- test3() (8,026 s)
- test4() (7,431 s)
- test5() (6,827 s)
- test6() (9,473 s)


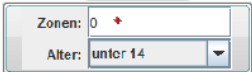
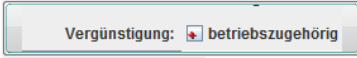

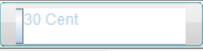
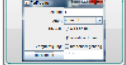
The screenshot shows a test runner interface with a green progress bar at the top. Below the progress bar, it displays 'Finished after 51,498 seconds'. The summary shows 'Runs: 6/6', 'Errors: 0', and 'Failures: 0'. The test suite 'GuiTarifrechnerTest [Runner: JUnit 5] (51,266 s)' is expanded, showing six individual test cases: test1() (11,741 s), test2() (7,750 s), test3() (8,026 s), test4() (7,431 s), test5() (6,827 s), and test6() (9,473 s). Each test case is marked with a green checkmark icon, indicating successful completion.

- Sikulix-API mit Potenzial, aber wenig Entwickelnde
- kombinierbar mit anderen Ansätzen
- Variante Sikuli-Script (Sikulix-IDE) (<http://www.sikuli.org/>)
 - ähnliches Konzept, aber mit Oberfläche und Skriptsprache, die das Suchen, Analysieren und Nutzen von Bildern vereinfacht
- typische Weiterentwicklung: Texterkennung
















Sikulix-IDE mit Sikulix-Skript








The screenshot shows the Sikulix-IDE interface. The main window displays a script for 'Test02Sikuliskript.sikulix' with the following code:

```
1 doubleClick(  )
2 click(  )
3 type('1')
4 click(  )
5 click(  )
6 exists(  )
7 click(  )
8
```



The left sidebar contains the following categories and functions:

- Finden**
 - exists()
 - find()
 - findAll()
 - wait()
 - waitVanish()
- Maus-Aktionen**
 - click()
 - doubleClick()
 - rightClick()
 - hover()
 - dragDrop( , )
- Tastatur-Aktionen**
 - type(text)
 - type( , text)
 - paste(text)
 - paste( , text)
- Ereignis-Überwachung**
 - onAppear( , handler)
 - onVanish( , handler)
 - onChange(handler)
 - observe()

Finden

```
exists(  )
find(  )
findAll(  )
wait(  )
waitVanish(  )
```

Ereignis-Überwachung

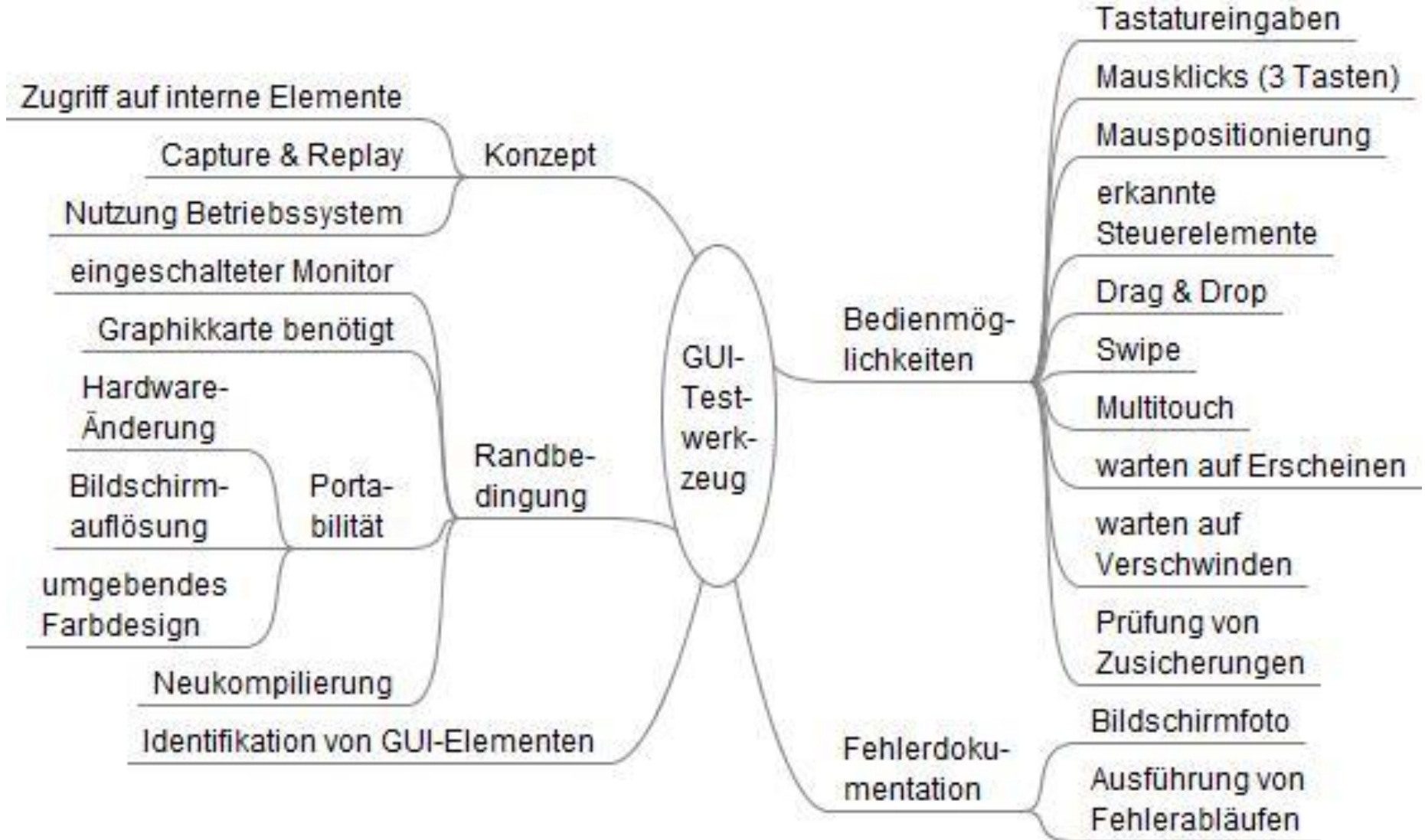
```
onAppear(  , handler )
onVanish(  , handler )
onChange( handler )
observe( )
```

- Sikulix sucht generell nur nach Bildausschnitten auf dem aktuellen Bildschirm
- dies leicht mit anderen (GUI-)Steuerungs-Testwerkzeugen kombinierbar
- Beispiel: Kombination mit AssertJ
 - AssertJ wird dann typischerweise zur Steuerung genutzt
 - Sikulix sucht dann erwartete Bildschirmausgaben
 - unterstützt z. B. den Test graphischer Editoren
 - auch Konsolenausgaben als Bild interpretierbar
- Ähnlicher Ansatz bei OpenQA (<https://openqa.opensuse.org/>), entwickelt zum Test von Installationsskripten von OpenSuSe-Linux (<https://www.opensuse.org/>), auch anderweitig verwendbar

Video 12

- GUI-Tests sind generell möglich
- GUI-Tests erst planen, wenn GUI relativ festgelegt ist
- Werkzeuge kritisch evaluieren, ob sie für Projekt bzw. typische Unternehmensaufgaben geeignet sind
- freie GUI-Werkzeuge können auf jeden Fall Einstieg in GUI-Testansätze sein
- Evaluation von kommerziellen Werkzeugen in diesem Bereich oft sinnvoll
- (wieder) können Kombinationen von Werkzeugen sinnvoll sein
- generell wird Teststrategie benötigt, was wann getestet wird (bottom up, top down, middle out)

Analysefaktoren für GUI-Testwerkzeuge



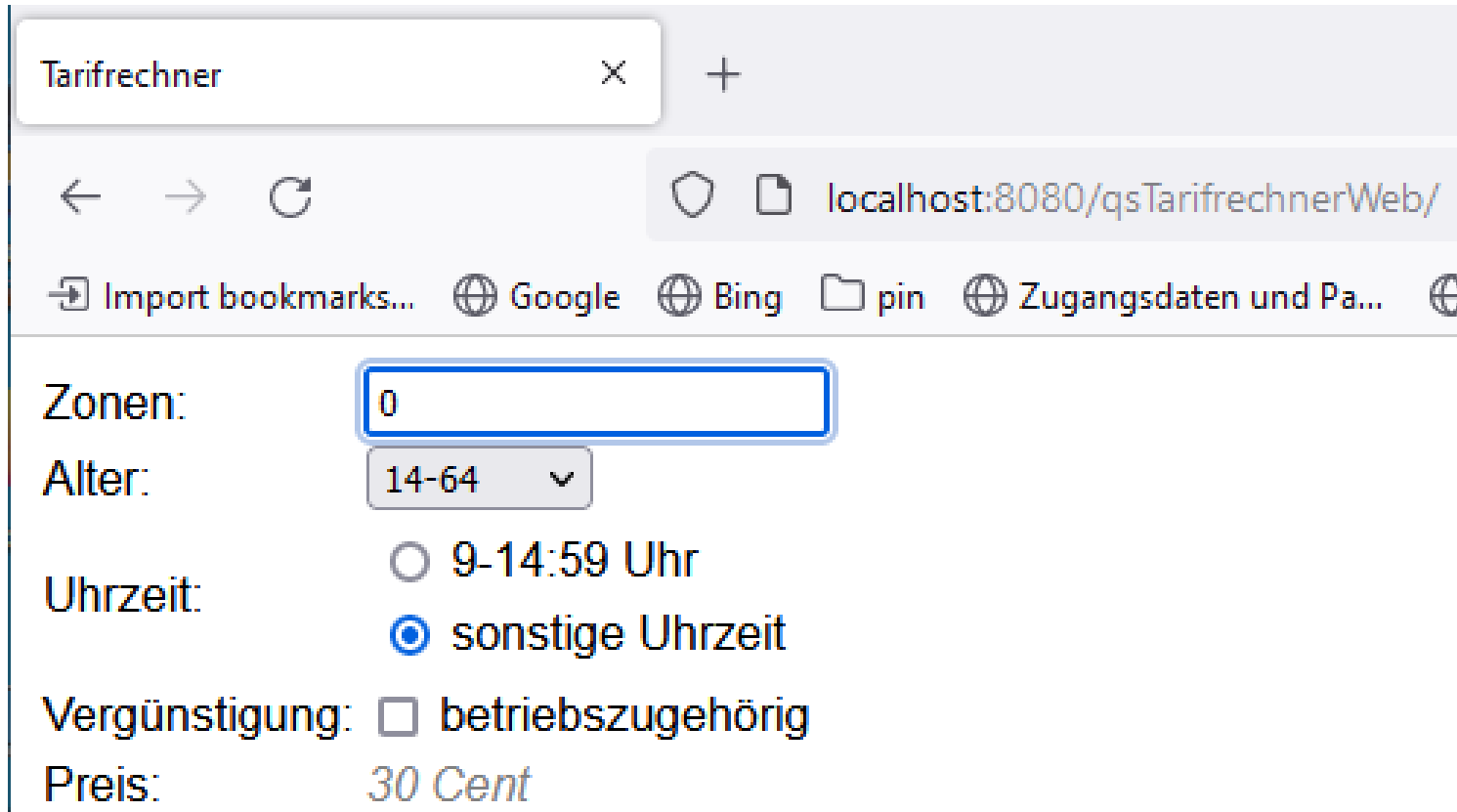
- sehr komplexe Aufgabe, da viele Äquivalenzklassen
- Browser mit unterschiedlichen Engines
- Browser mit unterschiedlichen Einstellungen (Profile):
 - Font, Fontgröße, Farben
 - Umgang mit Cache
 - Nutzung von PlugIns
 - ...
- generelles Problem mit jedem neuen Update
- pragmatischer Ansatz: Browser mit festen Basiseinstellungen, läuft auf Testrechner, der nicht automatisch aktualisiert, nutzt z. B. portable Browserversionen <https://portableapps.com/>

generelle Anmerkung zur GUI-Steuerung

- externe GUI-Steuerungsprogramme können sehr schnell reagieren
- zentrale Funktionalität ist „warten“
- warten, bis etwas auf der Oberfläche geschieht; wenn nicht, dann Exception
- keine gute Idee `Thread.sleep()` da ggfls. zu lang oder zu kurz
- `s.wait(Bild, Timeout)`
- `s.waitVanish(Bild, Timeout)`

Sikulix auch für Web-Applikationen nutzbar (1/2)

- kleine Variante, Preisausgabe ändert sich beim Tippen sofort



Tarifrechner × +

← → ↻ localhost:8080/qsTarifrechnerWeb/

Import bookmarks... Google Bing pin Zugangsdaten und Pa...

Zonen:

Alter: ▼

Uhrzeit: 9-14:59 Uhr
 sonstige Uhrzeit

Vergünstigung: betriebszugehörig

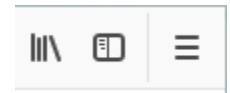
Preis: 30 Cent

- nur zwei Änderungen: neue Bilder mit anderen Pixelpositionen und Start, Browser muss über „X“ geschlossen werden

Sikulix auch für Web-Applikationen nutzbar (2/2)

```
public void initialisieren() {
    s = new Screen();
    app = App.open("FirefoxPortable/FirefoxPortable.exe -private "
        + "http://localhost:8080/qsTarifrechnerWeb/");
    try {
        Thread.sleep(3000); // unsauber, besser auf Bild warten
    } catch (InterruptedException ex) {}
}
```

```
public void beenden() {
    try {
        s.click(s.find(verzeichnis+"Terminieren2.png")
            .getTopLeft().right(86).above(23));
    } catch (FindFailed e) {
        throw new IllegalArgumentException(e.getMessage());
    }
    try {
        Thread.sleep(3000);
    } catch (InterruptedException ex) {}
}
```

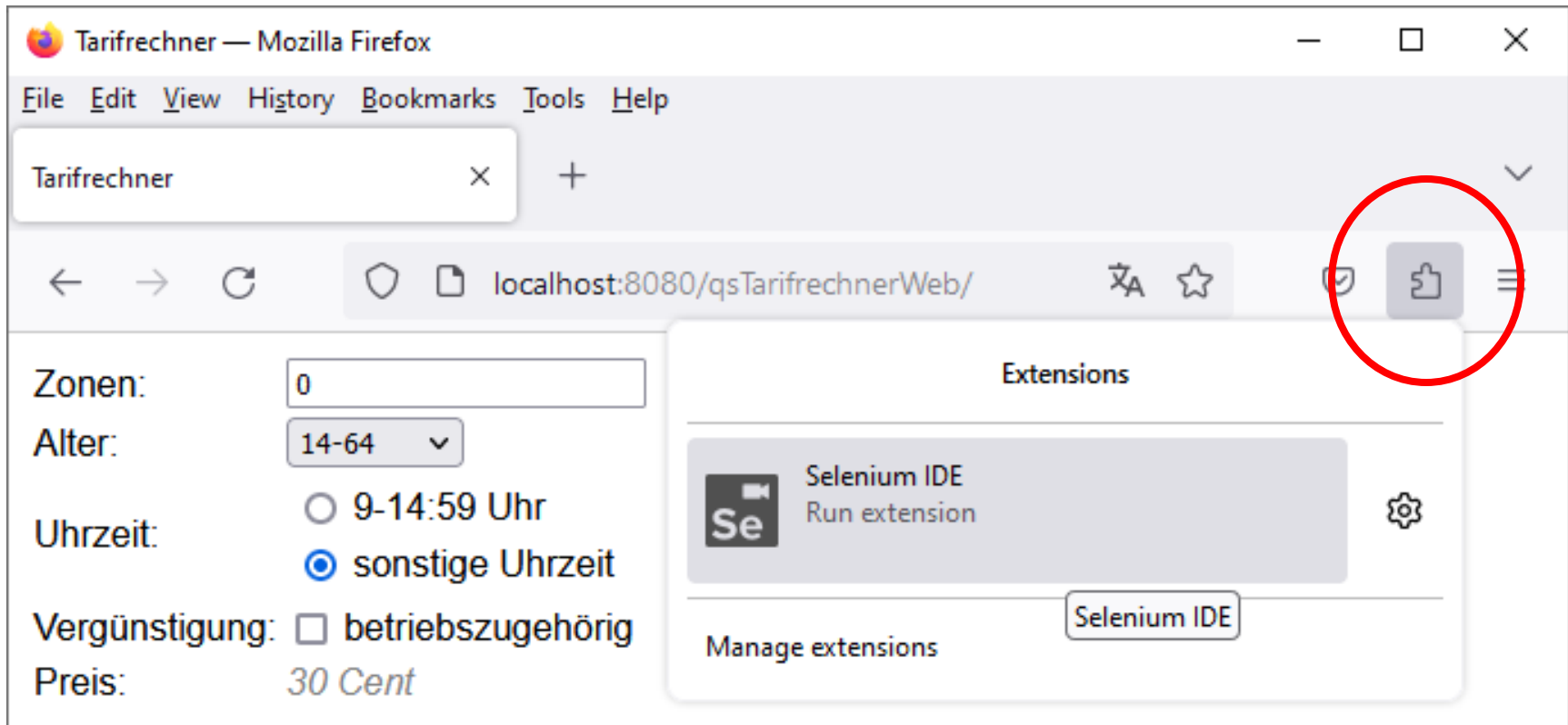


- grundsätzlich: viel aufteilbar und getrennt testbar (Beans, Persistenzanteil), klassisch mit JUnit und Mocks testbar
- verschiedene Varianten mit Test-Servern (oder speziellen Containern), die in Unit-Frameworks eingebunden werden
- Beispiele: Arquillian als Embedded Container (JBoss)
`EJBContainer.createEJBContainer();`
- Tests steuerbar über Selenium
- trotzdem gerade im reinen Enterprise-Bereich (ohne GUIs) bleibt gewisse Unsicherheit
- aktuelle Forschung: Tests verteilt auf mehreren Rechnern laufen lassen, Ergebnisse einsammeln und komponieren

- Web-Browser nutzen schwerpunktmäßig HTML zur Darstellung
- Capture & Replay-Werkzeuge, die hardgecoded Pixel und Klicks verarbeiten, eignen sich meist auch für diese Programme
- Einfaches Werkzeug für Web-Applikationen und anfänglich nur Firefox ist Selenium IDE (<http://seleniumhq.org/>)
 - erlaubt Capture & Replay von Nutzereingaben
 - ermöglicht Tests von Elementen
 - erlaubt den Export der aufgezeichneten Tests u. a. in JUnit
 - basiert auf JavaScript and Iframes
 - kommerzielle Alternative/Ergänzung: Katalon Recorder <https://katalon.com/katalon-recorder-ide>
- Programmierte Tests mit Selenium WebDriver (zu bevorzugen)

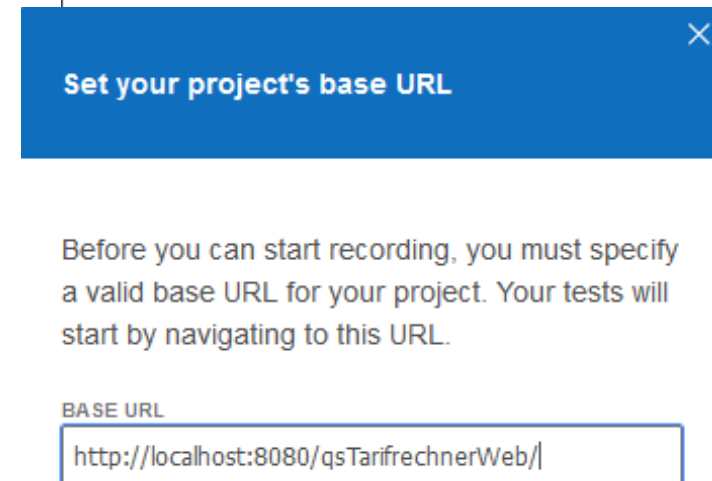
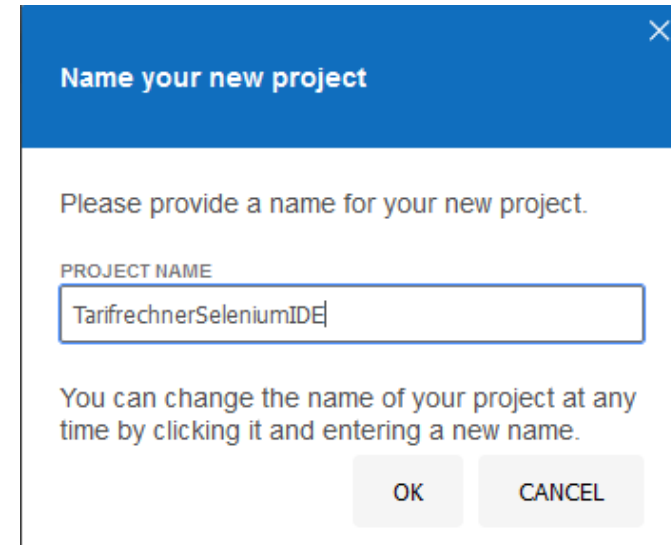
Selenium IDE – Starten

- Selenium IDE zu Firefox oder Chrome als Erweiterung hinzufügen
- über Icon Aufzeichnung starten



Selenium IDE – Projektverwaltung

- eigene Projektverwaltung; in andere Werkzeuge integrierbar
- Testsuites und Tests



- <https://www.selenium.dev/selenium-ide/>

Selenium IDE - Aufzeichnen

The screenshot shows a Mozilla Firefox browser window with the 'Tarifrechner' (Rate Calculator) application running on localhost:8080. The application has several input fields: 'Zonen' (Zones) with value '4', 'Alter' (Age) with a dropdown set to 'über 64', 'Uhrzeit' (Time) with radio buttons for '9-14:59 Uhr' and 'sonstige Uhrzeit' (selected), 'Vergünstigung' (Discount) with a checkbox for 'betriebszugehörig', and 'Preis' (Price) showing '480 Cent'. A Selenium IDE notification bar at the bottom left says 'Selenium IDE is recording'. Overlaid on the browser is the Selenium IDE extension interface. The 'Project' is 'TarifrechnerSeleniumIDE*'. The 'Tests' panel shows a table of recorded actions:

Command	Target	Value
click	id=form:zo nen	
type	id=form:zo nen	4
select	id=form:alt er	label=über 64
click	css=optio n:nth-child	

Below the table are input fields for 'Command', 'Target', 'Value', and 'Description'. The 'Log' and 'Reference' tabs are visible at the bottom of the extension window.

Selenium IDE – Tests editierbar

- nach Testende kann Testfall bearbeitet (und wieder ausgeführt) werden

The screenshot shows the Selenium IDE interface in Mozilla Firefox. The project is named 'TarifrechnerSeleniumIDE*'. A test case 'testÜber64*' is selected, and its commands are listed in a table below. The 8th command, 'click' on target 'id=form:betriebszugehoerig', is highlighted. Below the table, the details for this command are shown: Command: click, Target: id=form:betriebszugehoerig, Value: (empty), and Description: (empty). A 'Run current test Ctrl+R' tooltip is visible over the search bar.

	Command	Target	Value
1	open	/qsTarifrechnerWeb/	
2	set window size	466x503	
3	click	id=form:zonen	
4	type	id=form:zonen	4
5	select	id=form:alter	label=über 64
6	click	css=option:nth-child(3)	
7	click	id=form:uhrzeit:0	
8	click	id=form:betriebszugehoerig	

8	<i>click</i>	id=form:betriebszugehoerig	
9	<i>assert text</i>	id=form:ergebnis	230 Cent

- Befehle und damit Zusicherungen ergänzbar

Command

Target

Value

Description

8	✓ <i>click</i>	id=form:betriebszugehoerig	
9	✗ <i>assert text</i>	id=form:ergebnis	230 Cent

Command

Target

Value

Description

Runs: 1 Failures: 1

- Selenium steuert Browser von Java (C#, Python, Ruby) aus
- Installation als jar-Dateien
- flexible Möglichkeiten zum Finden von GUI-Komponenten
- ideal für Regressionstests, bei wenig sich ändernden GUIs

- in fast allen Unternehmen genutzt, die Web-Applikationen herstellen
- kontinuierliche Weiterentwicklung (nicht immer alles übertragbar, Selenium -> Selenium 2)
- Grundregel: nur automatisieren, was sinnvoll und machbar ist, Rest manuell
- <http://docs.seleniumhq.org/docs/>

- Klasse WebDriver als zentrale Steuerungsmöglichkeit
- Erzeugt neue Browser-Instanz
- Browser muss auf dem System installiert sein, nutzt keine weiteren Einstellungen des aktuellen Nutzers (leeres Profil)
- werden kontinuierlich weiterentwickelt
- (früher reichte `driver = new InternetExplorerDriver();`)
- bisheriges Angebot (unterschiedliche Qualität):
HtmlUnitDriver(), FirefoxDriver(), ChromeDriver(), EdgeDriver()
InternetExplorerDriver(),
- OperaDriver durch andere Entwickelnde, iPhoneDriver nur zusammen mit Apple-XCode-Umgebung, AndroidDriver mit Android-Entwicklungsunterstützung

Mit Entwicklern definierte (HTML-)Ids

Zonen: **form:zonen**

Alter: **form:alter**

Uhrzeit: 0-14:59 Uhr **form:uhrzeit:0**
 sonstige Uhrzeit **form:uhrzeit:1**

Vergünstigung: Betriebszugehörig **form:betriebszugehoerig**

Preis: **form:ergebnis**

Einführendes Beispiel (1/5)

```
public class Analyse {
public static void main(String[] args) throws IOException {
    File pfad = new File("");
    System.out.println(pfad.getAbsolutePath());
    System.setProperty("webdriver.gecko.driver",
        pfad.getAbsolutePath()
        + "\\lib\\geckodriver.exe");
    FirefoxProfile profile = new FirefoxProfile();
    FirefoxOptions options = new FirefoxOptions();
    options.setBinary("C:\\Program Files\\Mozilla Firefox"
        + "\\firefox.exe");
    options.setProfile(profile);
    FirefoxDriver driver = new FirefoxDriver(options);
    // driver = new HtmlUnitDriver();
    // driver = new ChromeDriver();
    driver.getCapabilities().asMap().forEach(
        (k, v) -> System.out.println(k + " : " + v));
}
```

Einführendes Beispiel (2/5) - konfigurierbar

```
F:\workspaces\eclipseWS\qsWebSeleniumTarifrechner
acceptInsecureCerts : true
browserName : firefox
browserVersion : 121.0.1
moz:accessibilityChecks : false
moz:buildID : 20240108143603
moz:debuggerAddress : 127.0.0.1:42804
moz:geckodriverVersion : 0.34.0
moz:headless : false
moz:platformVersion : 10.0
moz:processID : 3464
moz:profile :
C:\Users\Kleuker\AppData\Local\Temp\rust_mozprofileER7RtB
moz:shutdownTimeout : 60000
moz:webdriverClick : true
moz:windowless : false
pageLoadStrategy : normal
platformName : windows
proxy : Proxy()
```

Einführendes Beispiel (3/5)



```
driver.get("http://localhost:8080/qsTarifrechnerWeb/");
List<WebElement> liste
    = driver.findElements(By.tagName("input"));
for (WebElement w : liste) {
    System.out.println("  " + w.getTagName()
        + " :: " + w.getAttribute("type")
        + " :: " + w.getAttribute("name")
        + " :: " + w.getAttribute("value")
        + " :: " + w.getText()
        + " :: " + w.getLocation() + " :: " + w.isEnabled());
}
```

```
input::hidden::form::form:::(0, 0)::true
input::text::form::zonen::0:::(121, 11)::true
input::radio::form::uhrzeit::billig:::(129, 69)::true
input::radio::form::uhrzeit::teuer:::(129, 94)::true
input::checkbox::form::betriebszugehoerig::on:::(125, 122)::true
input::hidden::javax.faces.ViewState::9119616927154184678:712674
2113548250622:::(0, 0)::true
```

Einführendes Beispiel (4/5)

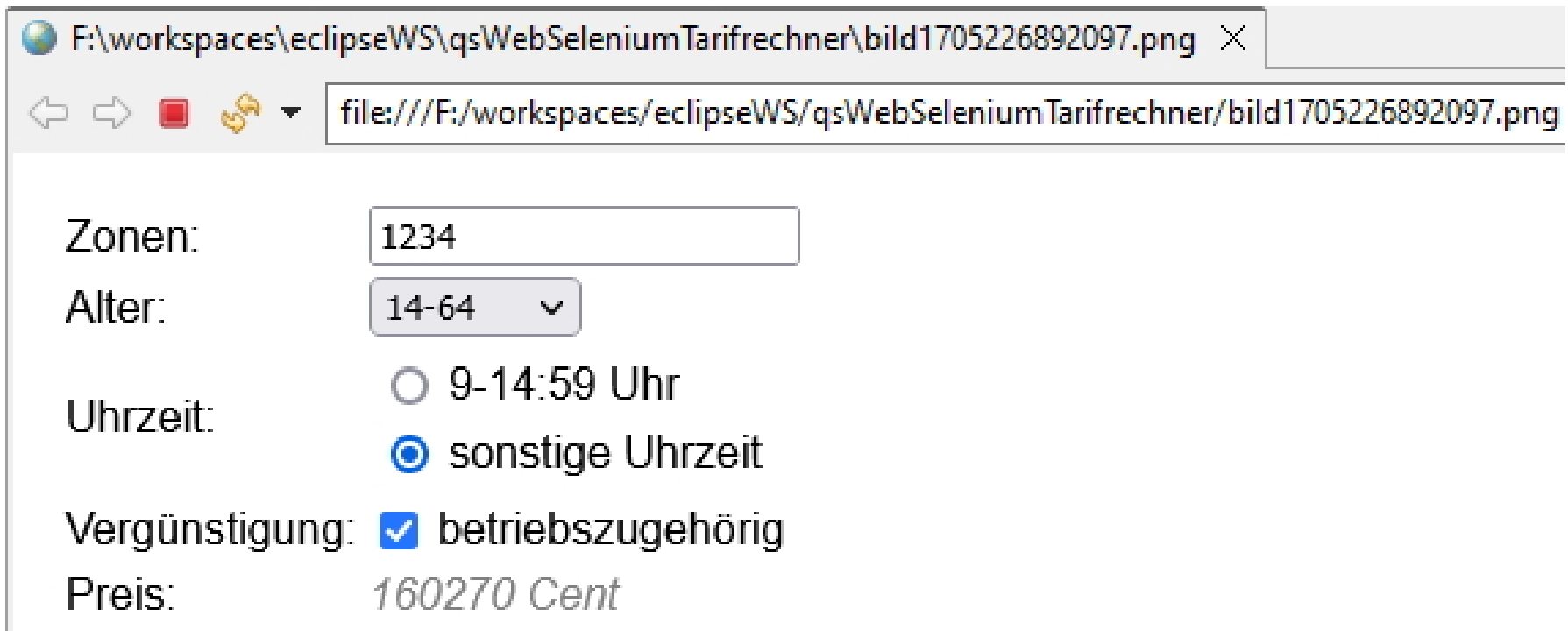
```
//Steuerung
WebElement element = driver
    .findElement(By.id("form:zonen"));
element.clear();
element.sendKeys("1234");

driver.findElement(By.id("form:betriebszugehoerig"))
    .click();

// Bildschirmfoto
File screenshot = ((TakesScreenshot) driver)
    .getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(screenshot, //Apache commons-io
    new File("bild" + new Date().getTime() + ".png"));

driver.close();
}
}
```

Einführendes Beispiel (5/5): erstellte Bilddatei



- Viele weitere Lokalisierungsmöglichkeiten

Method Summary

static By className (java.lang.String className)
static By cssSelector (java.lang.String selector)
WebElement findElement (SearchContext context)
List< WebElement > findElements (SearchContext context)
static By id (java.lang.String id)
static By linkText (java.lang.String linkText)
static By name (java.lang.String name)
static By partialLinkText (java.lang.String linkText)
static By tagName (java.lang.String name)
static By xpath (java.lang.String xpathExpression)

<https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/By.html>

Browsermöglichkeiten (1/2)

- sehr viele Bedienungsmöglichkeiten in Browsern, (fast) alle in Sikulix abgebildet
- Beispiel: Single-Page-Application, http-Adresse bleibt identisch, aber interner HTML-DOM-Tree verändert sich; WebElement-Objekte können veraltet sein (z. B. gelöscht und mit gleicher Id aktualisiert hinzugefügt)

Zonen:	<input type="text" value="1"/>	→	Zonen:	<input type="text" value="10"/>
Alter:	<input type="text" value="14-64"/>		Alter:	<input type="text" value="14-64"/>
Uhrzeit:	<input type="radio"/> 9-14:59 Uhr <input checked="" type="radio"/> sonstige Uhrzeit		Uhrzeit:	<input type="radio"/> 9-14:59 Uhr <input checked="" type="radio"/> sonstige Uhrzeit
Vergünstigung:	<input type="checkbox"/> betriebszugehörig		Vergünstigung:	<input type="checkbox"/> betriebszugehörig
Preis:	130 Cent		Preis:	1300 Cent

0 getippt

Browsermöglichkeiten (2/2)

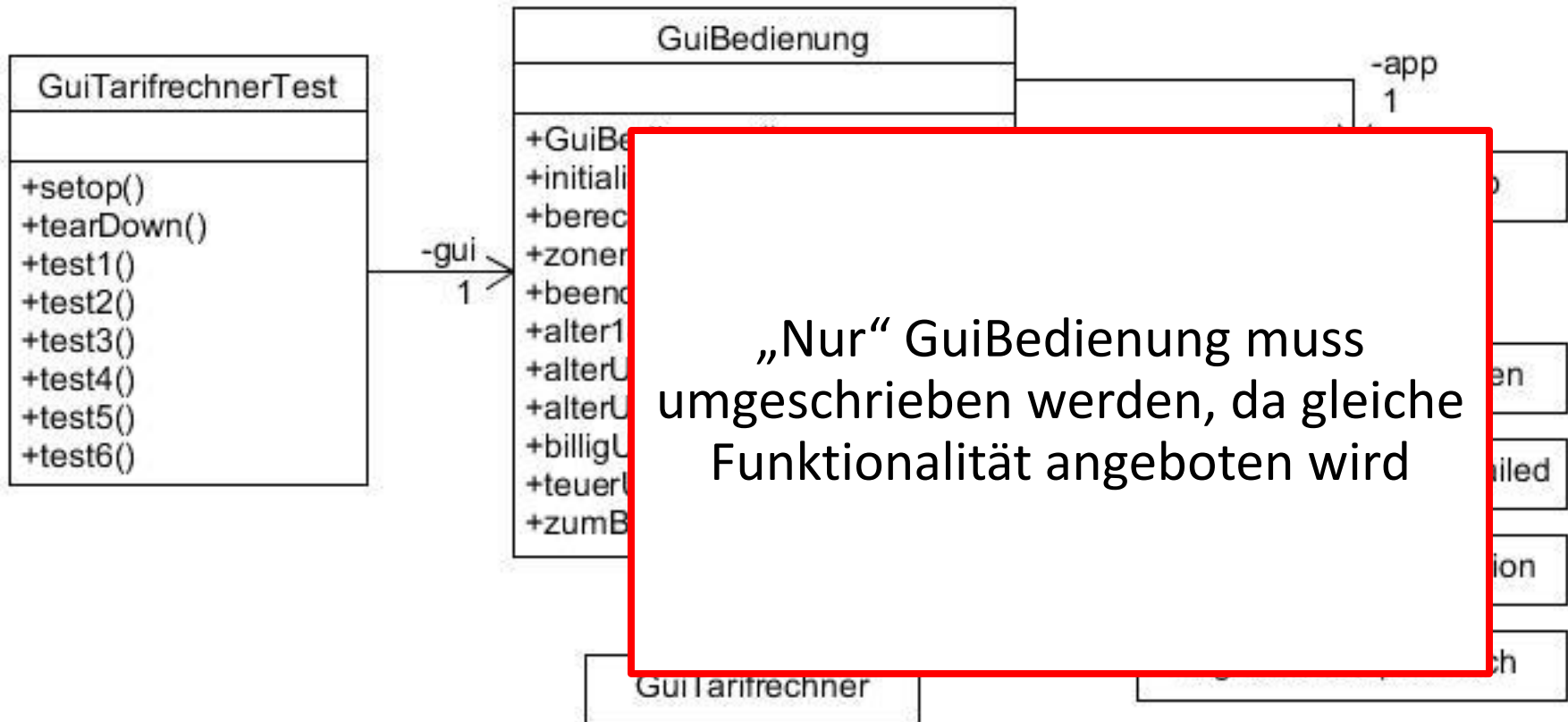
- theoretisch sehr viele Äquivalenzklassen, da Browser extrem konfigurierbar, vieles als Parameter, z. B. Nutzungsprofil

The screenshot shows a web browser window with a single tab titled 'Tarifrechner'. The address bar displays 'localhost:8080/qsTarifrechnerWeb/'. The page content is a form with the following elements:

- Zonen:** A text input field containing '1234'.
- Alter:** A dropdown menu showing '14-64'.
- Uhrzeit:** Two radio button options: '9-14:59 Uhr' (unselected) and 'sonstige Uhrzeit' (selected).
- Vergünstigung:** A checked checkbox next to the text 'betriebszugehörig'.
- Preis:** The value '160270 Cent' is displayed in a light blue font.

- startet Basisversion, Firefox zeigt mit rotem Balken potenzielles Sicherheitsrisiko (Fernbedienung)

Test der Applikation (1/7)



Test der Applikation (2/7)

```
public class GUIBedienung {  
    private WebDriver driver; // Interface (= Flexibilität)  
  
    public GUIBedienung() {  
        File pfad = new File("");  
        System.setProperty("webdriver.gecko.driver",  
            pfad.getAbsolutePath() + "\\lib\\geckodriver.exe");  
        FirefoxProfile profile = new FirefoxProfile();  
        FirefoxOptions options = new FirefoxOptions();  
        options.setBinary("C:\\Program Files\\Mozilla Firefox"  
            + "\\firefox.exe");  
        options.setProfile(profile);  
        this.driver = new FirefoxDriver(options);  
    }  
  
    public void initialisieren() {  
        this.driver  
            .get("http://localhost:8080/qsTarifrechnerWeb/");  
    }  
}
```

```
public void beenden() {
    try {
        this.driver.quit();
    } catch (Exception e){
        System.out.println("Problem: " + e);
    }
}
```

```
// nicht genutzt, aber wichtig ☺
private void warteAufSeiteMitId(String id) {
    new FluentWait<WebDriver>(driver).until(
        new ExpectedCondition<WebElement>() {
            @Override
            public WebElement apply(WebDriver d) {
                return d.findElement(By.id(id));
            }
        }
    ));
}
```

```
public String berechnenAusfuehren() {
    String erg = null;
    boolean stable = false;
    while(!stable) {
        try {
            String ergOld = erg;
            erg = driver
                .findElement(By.id("form:ergebnis")).getText();
            if (erg !=null && erg.equals(ergOld)) {
                stable = true;
            }
        } catch (Exception e) {
            System.out.println("while: " + erg);
        }
    }
    return erg;
} // vermeiden von StaleElementReferenceException
```

```
public void zonenEingeben(String eingabe) {
    WebElement element = this.driver
        .findElement(By.id("form:zonen"));
    element.clear();
    element.sendKeys(eingabe);
}
```

```
public void alterUnter14Waehlen() {
    WebElement element = this.driver
        .findElement(By.id("form:alter"));
    Select box = new Select(element);
    box.selectByVisibleText("unter 14"); // enge Kopplung
}
```

```
public void alterUeber64Waehlen() {
    WebElement element = this.driver
        .findElement(By.id("form:alter"));
    Select box = new Select(element);
    box.selectByVisibleText("über 64");
}
```

```
public void alter1464Waehlen() {
    WebElement element = this.driver
        .findElement(By.id("form:alter"));
    Select box = new Select(element);
    box.selectByVisibleText("14-64");
}
```

```
public void billigUhrWaehlen() {
    this.driver.findElement(By.id("form:uhrzeit:0")).click();
}
```


Test der Applikation (7/7)

```
public void teuerUhrWaehlen() {  
    this.driver.findElement(By.id("form:uhrzeit:1")).click();  
}
```

```
public void zumBetriebKlicken() {  
    this.driver  
        .findElement(By.id("form:betriebszugehoerig")).click();  
}
```

```
}
```

Finished after 30,471 seconds

Runs: 6/6 ❌ Errors: 0 ❌ Failures: 0

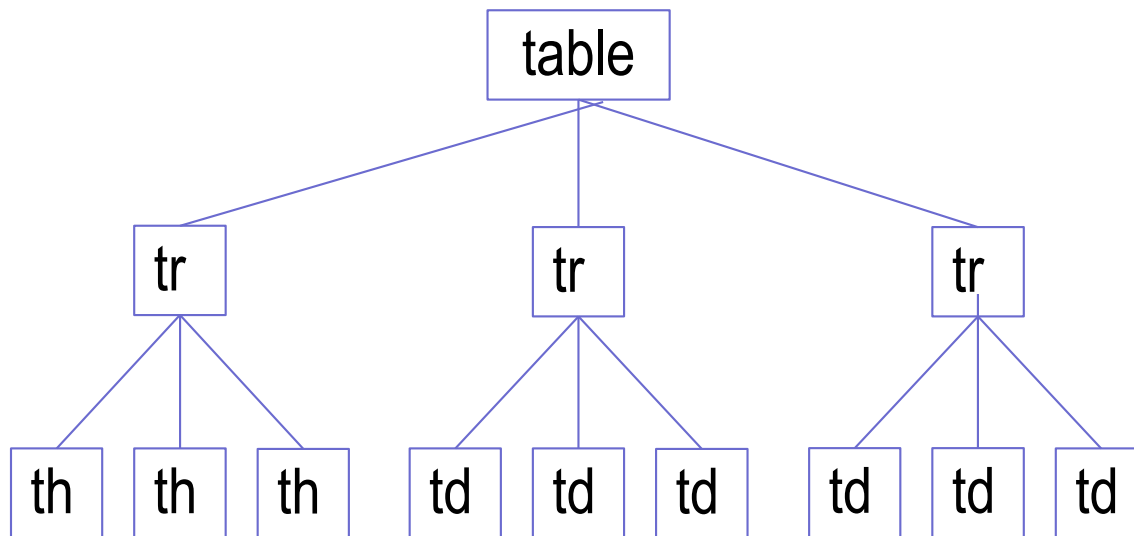
GuiTarifrechnerTest [Runner: JUnit 5] (30,261 s)

- test1() (5,969 s)
- test2() (4,777 s)
- test3() (4,864 s)
- test4() (4,940 s)
- test5() (4,629 s)
- test6() (5,080 s)

The screenshot shows a test runner interface with a green progress bar at the top. Below the progress bar, it displays the test results for 'GuiTarifrechnerTest'. The test suite is marked as successful with a green checkmark. The total execution time is 30,261 seconds. The test suite contains six individual tests, each with a green checkmark and its execution time in seconds.

Hilfreiche Suche mit XPath (1/2)

- HTML-Dokumente sind hierarchisch aufgebaut und können als Baum dargestellt werden
- gibt Varianten zur Darstellung mit `<thead>`, `<tbody>`, `<tfoot>`



- https://wiki.selfhtml.org/wiki/HTML/Tabellen/Aufbau_einer_Tabelle

```
<table>
  <tr>
    <th> Ueberschrift1 </th>
    <th> Ueberschrift2 </th>
    <th> Ueberschrift3 </th>
  </tr>
  <tr>
    <td> Zeile1_1</td>
    <td> Zeile1_2 </td>
    <td> Zeile1_3 </td>
  </tr>
  <tr>
    <td> Zeile2_1</td>
    <td> Zeile2_2 </td>
    <td> Zeile2_3 </td>
  </tr>
</table>
```

Hilfreiche Suche mit XPath (2/2) - Minieinblick

```
this.driver.findElement(  
    By.xpath("pfad"))
```

- //table : finde alle Tabellen, mit // wird beliebige Baumtiefe erreicht
- //table/tr: alle tr-Elemente, mit .size() Zeilenanzahl
- //table/tr/th: alle th-Elemente direkt unter th-Element, mit .size() Spaltenanzahl
- //table/tr[2]/td[1]: vom 2. tr-Element das erste th-Element, hat Text Zeile1_1
- //*[@id='id42']: alle Elemente mit Attribut id mit Wert 42, * für beliebiges Element
- Werkzeug: <http://xpather.com/>

```
<table>  
  <tr>  
    <th> Ueberschrift1 </th>  
    <th> Ueberschrift2 </th>  
    <th> Ueberschrift3 </th>  
  </tr>  
  <tr>  
    <td> Zeile1_1</td>  
    <td> Zeile1_2 </td>  
    <td> Zeile1_3 </td>  
  </tr>  
  <tr>  
    <td> Zeile2_1</td>  
    <td> Zeile2_2 </td>  
    <td> Zeile2_3 </td>  
  </tr>  
</table>
```

Selenium – alternative Nutzung (Link-Prüfung 1/2)

```
public static void main(String[] s) throws IOException {
    WebDriver driver = new EdgeDriver();
    driver.get("http://kleuker.iui.hs-osnabrueck.de/index.html");
    WebElement link = driver
        .findElement(By.partialLinkText("Aktuell"));
    link.click();
    System.out.println(driver.getCurrentUrl());
    List<WebElement> liste = driver.findElements(By.tagName("a"));
    for (WebElement w : liste) {
        String href = w.getAttribute("href");
        if(!href.contains("kleuker")) {
            HttpURLConnection hc = (HttpURLConnection) new URL(href)
                .openConnection();
            System.out.println(hc.getResponseCode() + " - " + href);
            hc.disconnect();
        }
    }
    driver.close();
}
```

Selenium – alternative Nutzung (Link-Prüfung 2/2)



- 301 - <http://www.archimedon.de/>
- 301 - <http://www.basecom.de/>
- 200 - <https://www.greenbone.net/>
- 400 - <http://www.hellmann.de/>
- 301 - <http://www.iscope.de/>
- 301 - <http://www.lmis.de/>
- 301 - <http://www.logentis.de/>
- 301 - <http://www.netrocks.info/>
- 301 - <http://www.infomantis.de/>
- 200 - <https://salt-and-pepper.eu/>
- 301 - <http://www.sievers-group.com/>
- 200 - <https://www.slashwhy.de/de/>
- 200 - <https://www.swo-netz.de/unternehmen/portrait.html>
- 200 - <https://www.symbic.de/karriere.php>
- 302 - <http://www.tso.de/>
- 301 - <http://www.gmh-systems.de/>
- 200 - <https://www.gs-it-solutions.com/de/>
- 301 - <http://www.sla.de/>
- 301 - <http://www.soft2tec.com/>

- Wechsel zwischen Fenstern und zwischen Frames
- Möglichkeit vorwärts und zurück zu navigieren
- Nutzung von Cookies
- Unterstützung von Drag und Drop
- Proxy-Nutzung
- Einstellung von Wartezeiten
- Warten auf das Erscheinen von HTML-Elementen (wichtig in Richtung AJAX und HTML5)
- Zusammenspiel mit Selenium IDE zur Testaufzeichnung

Achtung: Viele Einstiegsfallen

- generell gute Einarbeitungsmöglichkeit durch gute Dokumentation
- trotzdem viele kleine Fehlerquellen, die Entwicklungsprozess bremsen können
- Beispiel: Tests ziehen auf anderen Rechner um
- wichtiges Detail aus der Doku "The browser zoom level must be set to 100% so that the native mouse events can be set to the correct coordinates." (nicht erster Google-Treffer)
- teilweise weitere Browser-Einstellungen beachten
- Fazit: Testrechner nie zu anderen Zwecken nutzen, Konfiguration sichern

- Browser stellen identische Inhalte leicht verändert da
- technisch überflüssig, aber wichtig für den Zeitgeist: modische Design-Anpassungen
- Für IT-Profi: Sisyphos-Arbeit; Test mit unterschiedlichen Browsern
- Direkte Hilfsmittel:
 - Lunascape: ein Browser, bei dem man zwischen drei Maschinen umschalten kann IE (Trident) + Firefox (Gecko) + Chrome, Safari (Webkit)
 - Windows: USB-portable Browser ohne Installationsnotwendigkeit (verändern keine Einstellungen): Firefox, Chrome, Opera, ...
- evtl. auch Capture & Replay mit Selenium zum inhaltlichen Test

- Integrationsnotwendigkeit von Entwicklung und Test
Programmierregeln müssen auf Testbarkeit abgestimmt sein

- Beispiel: öffentliche get- und set-Methoden für alle Objektvariablen

- Warum: einfache Herstellung präziser Testszenarien

Werkzeugauswahl zur Entwicklung muss mit Werkzeugauswahl zum Testen abgestimmt sein

- Beispiel: GUI-Entwicklungswerkzeug soll eindeutige Identifikatoren der GUI-Elemente unterstützen

- Warum: Einfache klare Ansteuerung von GUI-Elementen über Ids in Tests möglich; GUI-Änderungen führen nur zu k(l)einen Änderungen in Testfällen

Video 13

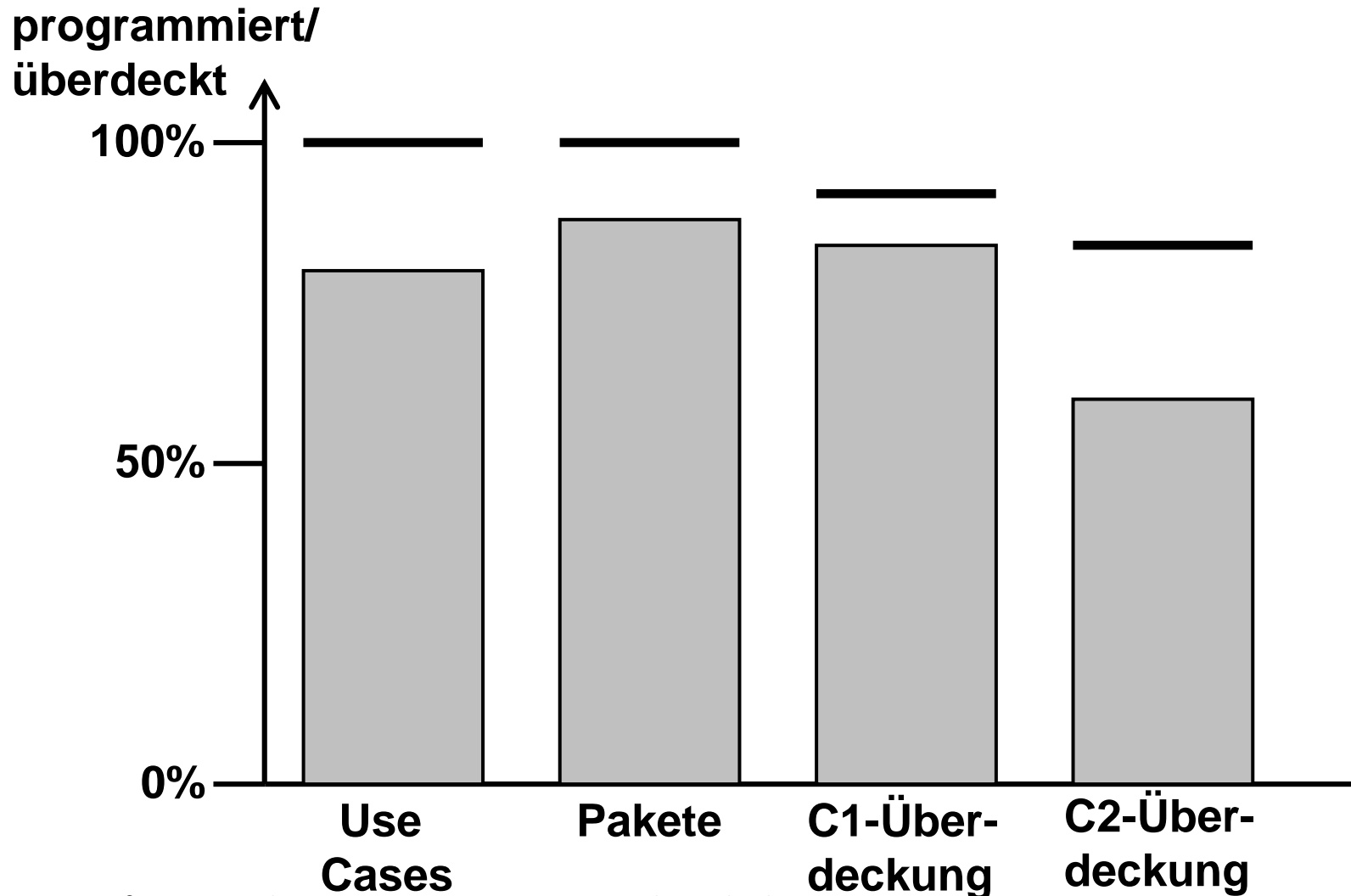
- Idee von Maßsystemen
- Halstead
- live Variables
- Variablenspanne
- McCabe-Zahl
- LCOM*

- bisherigen Prüfverfahren sind aufwändig, besteht Wunsch, schneller zu Qualitätsaussagen zu kommen
- Ansatz: Nutzung von Maßsystemen, die Zahlenwerte generieren, deren Werte Indiz für Qualität der SW sind
- Werden Maße automatisch berechnet, kann man Qualitätsforderungen stellen, dass bestimmte Maßzahlen in bestimmten Bereichen liegen

- Ähnliche Ansätze in der Projektverfolgung und Analyse der Firmengesamtlage (-> Balanced Scorecard)
-> siehe Qualitätsmanagement

- Wichtig ist, dass man weiß, dass nur Indikatoren betrachtet werden, d.h. gewonnene Aussagen müssen nachgeprüft werden
- Kritisch wird es, wenn die Entwicklung an Maßen orientiert wird
- Beispiel: Maß für das Testniveau -> Überdeckungsmaße
- Fehler: mit wenig Tests eine hohe Abdeckung bekommen
- sinnvoll: typische Testfälle schreiben, dann Überdeckung messen; bei niedrigen Werten Hintergründe analysieren und ggfls. Testfälle ergänzen

Metriken zur Ermittlung des Projektstands



Grobe Metriken (Min, Max, Schnitt, Abweichung)

- Lines of Code pro Methode (LOC)
mögliche Grundregel: maximale Zahl unter 20
- Methoden pro Klasse
Die Zahl sollte zwischen 3 und 15 liegen
- Parameter pro Methode
Die Zahl sollte unter 6 liegen
- Exemplarvariablen pro Klasse
Die Zahl sollte unter 10 liegen [Entitäten ?]
- Abhängigkeiten zwischen Klassen
Keine Klasse sollte von mehr als vier Klassen abhängen
- weitere Maßzahlen z. B. Anzahl von Klassenvariablen und Klassenmethoden

Basiert auf vier Größen

- N1: Gesamtzahl der verwendeten Operatoren (in Java z.B. +, -, *, /, ==, if, while, <Methodenname>)
- N2: Gesamtzahl der verwendeten Operanden (Variablen, Konstante)
- n1: Anzahl der genutzten unterschiedlichen Operatoren
- n2: Anzahl der genutzten unterschiedlichen Operanden
- Größe des Vokabulars: $n = n1 + n2$
- Länge der Implementation: $N = N1 + N2$

```
if (k < 2){  
    if (k > 0) x = x * k;  
}
```

unterschiedliche Operatoren: if (<) { > = * ; }

unterschiedliche Operanden: k 2 0 x

N1 = 13

N2 = 7

n1 = 10

n2 = 4

Berechnung von D (Schwierigkeiten ein Programm zu schreiben oder zu verstehen)

$$D = \frac{n1 * N2}{2 * n2}$$

- D ist Funktion vom Vokabular und der Anzahl der Operanden
- Quotient $N2 / n2$ gibt die durchschnittliche Verwendung von Operanden an
- Satz von Halstead: D beschreibt den Aufwand
 - zum Schreiben von Programmen,
 - den Aufwand bei Code- Reviews und
 - das Verstehen von Programmen bei Wartungsvorgängen.

Anmerkung: Aufbauend auf den Basismetriken gibt es weitere Metriken, die unterschiedliche Eigenschaften eines Programms erfassen

- Ansatz: Erstellung/Prüfung einer Anweisung umso schwieriger, je mehr Variablen beachtet werden müssen
- Eine Variable heißt zwischen der ersten und der letzten Nutzung lebendig

```
public static int mach(boolean a, boolean b){
    int x=0;           // a,b,x lebendig
    if (a)             // a,b,x lebendig
        x=2;          // b,x lebendig
    else
        x=3;          // b,x lebendig
    if (b)             // b,x lebendig
        return(6/(x-3)); // x lebendig
    else
        return(6/(x-2)); // x lebendig
}
```

- Interessant sind maximale und mittlere Zahl (Gesamtzahl lebendiger Variablen durch Anzahl ausführbarer Anweisungen) lebendiger Variablen

- Für Variable x: maximaler Abstand (in Code-Zeilen) zwischen ihren Nutzungen. (Zeilennummer in der x zum (i+1)-ten Mal minus Zeilennummer in der x zum i-ten Mal vorkommt.)

```
public static int mach(boolean a, boolean b){ //1
    int x=0; //2
    if (a) //3
        x=2; //4
    else
        x=3; //5
    if (b) //6
        return(6/(x-3)); //7
    else
        return(6/(x-2)); //8
}
```

a: $3-1=2$

b: $6-1=5$

x: $4-2=8-5=3$

Maximum: 5

Durchschnitt: 3.3

- Wieder durchschnittlicher und maximaler Wert interessant

1. Man konstruiere die Kontrollflussgraphen
2. Man messe die strukturellen Komplexität

Die zyklomatische Zahl $z(G)$ eines Kontrollflussgraphen G ist:

$$z(G) = e - n + 2 \text{ mit}$$

e = Anzahl der Kanten des Kontrollflussgraphen

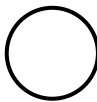
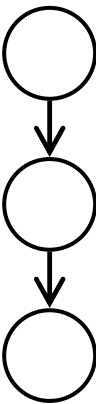
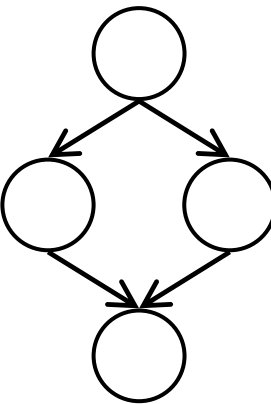
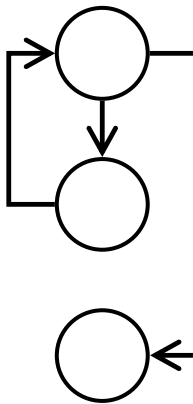
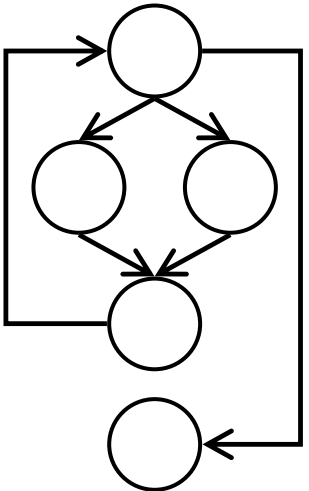
n = Anzahl der Knoten

Zyklomatische Komplexität gibt Obergrenze für die Testfallanzahl für den Zweigüberdeckungstest an

In der Literatur wird 10 oft als maximal vertretbarer Wert genommen (für OO-Programme geringer, z. B. 5)

für Java: $\#if + \#do + \#while + \#switch-cases + 1$

Beispiele für die zyklomatische Zahl

					
Anzahl Kanten	0	2	4	3	6
Anzahl Knoten	1	3	4	3	5
McCabe Zahl	1	1	2	2	3

- Komplexität von Verzweigungen berücksichtigen
- gezählt werden alle Booleschen Bedingungen in `if(<Bedingung>)` und `while(<Bedingung>)`: `anzahlBedingung`
- gezählt werden alle Vorkommen von atomaren Prädikaten: `anzahlAtome`
 - z. B.: `(a || x>3) && y<4` dann `anzahlAtome=3`
- erweiterte McCabe-Zahl
$$ez(G) = z(G) + \text{anzahlAtome} - \text{anzahlBedingung}$$
- wenn nur atomare Bedingungen, z. B. `if(x>4)`, dann gilt $ez(G) = z(G)$

- kurze Methoden mit selbsterklärenden Methodennamen fördern die Programmlesbarkeit wesentlich
- lokal zu optimieren: kurze Methoden (1), wenig Parameter (2), wenige Methoden (3)
- oftmals kann man Schleifen oder if-Blöcke in Methoden auslagern (Ansatz ist Teil der Refactoring-Idee)
- (**Refactoring**: Umbau von Programmen zur Erhöhung der Lesbarkeit, Wartbarkeit und Erweiterbarkeit)
- Eclipse hat Erweiterung Metrics mit mehreren Metriken u. a. erweiterte McCabe-Zahl

Refactoring – Positives Beispiel



```
public int ref(int x, int y, int z){
    int a=0;
    if(x>0){
        a=x;
        x++;
        --y;
        a=a+y+z;
    }
    return a;
}
```

```
public int ref(int x, int y, int z){
    int a=0;
    if(x>0){
        a = mach(x, y, z);
    }
    return a;
}

private int mach(int x, int y, int z){
    int a;
    a=x;
    x++;
    --y;
    a=a+y+z;
    return a;
}
```

Refactoring in Eclipse (1/2)

```
5 public int  
6     int a=0  
7     if(x>0)  
8         a=x;  
9         x++;  
10        --y;  
11        a=a+y;  
12    }  
13    return  
14 }  
15
```

The screenshot shows the Eclipse IDE interface. On the left, a code editor displays a Java method. Lines 8 through 11 of the code are highlighted in blue. A context menu is open over this selection, listing various actions such as Cut, Copy, Paste, and Refactor. The 'Refactor' option is highlighted in blue, and a sub-menu is visible to its right, with 'Extract Method...' selected and a mouse cursor hovering over it.

Refactoring in Eclipse (2/2)

Extract Method

Method name:

Access modifier: public protected package private

Parameters:

Type	Name
int	x
int	y
int	z

Declare thrown runtime exceptions

Generate method comment

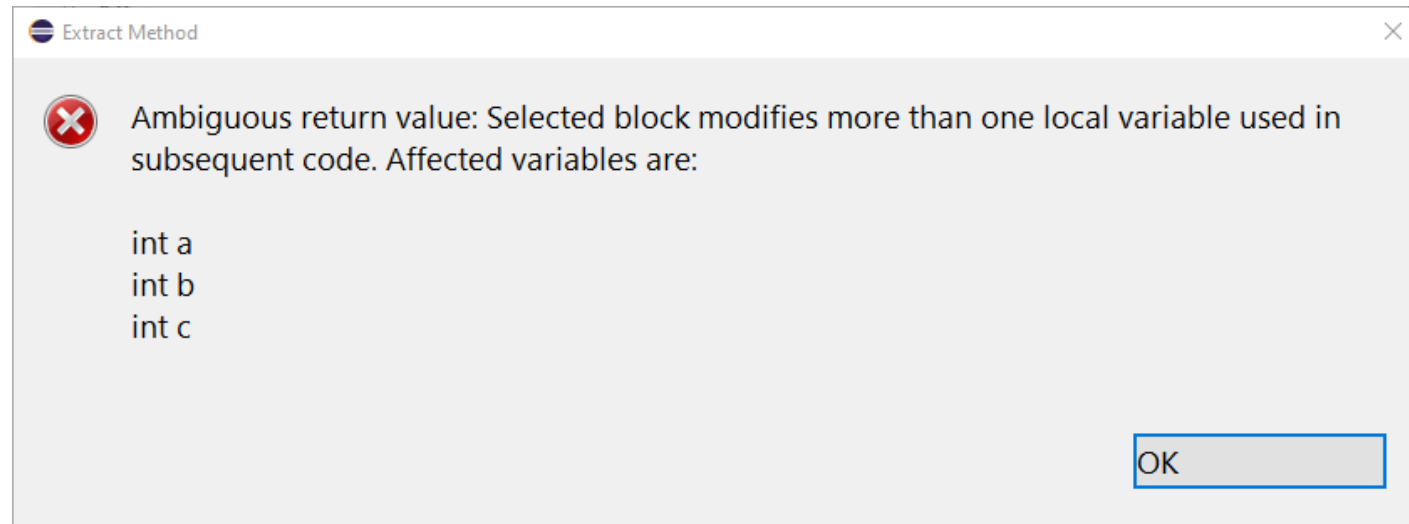
Replace additional occurrences of statements with method

Method signature preview:
public int mach(**int** x, **int** y, **int** z)

Edit...
Up
Down

Refactoring – nicht einfaches Beispiel

```
public int ref2(int x){  
    int a=0;  
    int b=0;  
    int c=0;  
    if(x>0){  
        a=x;  
        b=x;  
        c=x;  
    }  
    return a+b+c;  
}
```



Refactoring – (nicht) einfaches Beispiel in C++

```
int Rechnung::ref2(int x){
    int a=0;
    int b=0;
    int c=0;
    if(x>0){
        abcAnpassen(a,b,c,x);
    }
    return a+b+c;
}
```

```
void Rechnung::abcAnpassen(int& a, int& b, int& c, int x){
    a=x;
    b=x;
    c=x;
}
```

- Frage: Gibt es Maß für gute Objektorientierung?
- Ansatz (Indikator): Exemplarvariablen sollten in mehreren Methoden genutzt, möglichst kombiniert sein
- Hinweis: Es muss vorher festgelegt werden, ob Klassenvariablen und Klassenmethoden berücksichtigt werden sollen
- Was passiert, wenn man konsequent exzessiv OO macht und auch in den Exemplarmethoden get() und set() Methoden nutzt? Wie ist LCOM* dann rettbar?

Lack of Cohesion in Methods (LCOM*) - Berechnung

$$\text{LCOM}^* = (\text{avgNutz} - m) / (1 - m)$$

- sei var die Anzahl der Variablen
- nutzt(a) die Zahl der Methoden, die eine Exemplarvariable a der untersuchten Klasse nutzen
- sei avgNutz der Durchschnitt aller Werte für alle Exemplarvariablen (Summe nutzt(a)) / var
- sei m die Anzahl aller Methoden der untersuchten Klasse
- Ist der Wert nahe Null, handelt es sich um eine eng zusammenhängende Klasse
- Ist der Wert nahe 1, ist die Klasse schwach zusammenhängend, man sollte über eine Aufspaltung nachdenken

LCOM*-Beispiel

```
public class LCOMSpielerei {  
    private int a;  
    private int b;  
    private int c;  
  
    public void mach1(int x){  
        a=a+x;  
    }  
  
    public void mach2(int x){  
        a=a+x;  
        b=b-x;  
    }  
  
    public void mach3(int x){  
        a=a+x;  
        b=b-x;  
        c=c+x;  
    }  
}
```

```
var = 3 // Anzahl Variablen  
m = 3 // Anzahl Methoden  
nutzt(a)=3  
nutzt(b)=2  
nutzt(c)=1  
avgNutzt=(3+2+1)/3=2  
LCOM* = (2-3) / (1-3)  
        = -1/-2 = 0.5
```

für NetBeans gibt es einfaches Plugin
SimpleCodeMetrics (SCM) berechnet u.
a. McCabe-Zahl und LCOM* (= LCOM 3)

Berechnung mit SCM



```
public class LCOMSpielerei {  
    private int a;  
    private int b;  
    private int c;  
  
    public void mach1(int x){  
        a=a+x;  
    }  
  
    public void mach2(int x){  
        a=a+x;  
        b=b-x;  
    }  
  
    public void mach3(int x){  
        a=a+x;  
        b=b-x;  
        c=c+x;  
    }  
}
```

Output - Analysis of Refactoring % Test Results Usages

```
Cyclomatic complexity  
-----  
Average cyclomatic complexity: 1.0  
  
Methods with the highest cyclomatic complexity:  
  
Refactoring::mach3: 1  
Refactoring::mach2: 1  
Refactoring::mach1: 1  
  
-----  
LCOM  
-----  
  
-----  
LCOM 1: 0  
-----  
  
-----  
LCOM 2: 0.33333333333333337  
-----  
  
-----  
LCOM 3: 0.5  
-----  
  
-----  
LCOM 4: 1  
-----
```

Andere LCOM-Maße (Programm leicht geändert)

```
public class LCom {  
    private int a,b,c;  
  
    public int meth1(){  
        return a+b;  
    }  
  
    public int meth2(){  
        return b+a;  
    }  
  
    public int meth3(){  
        return c;  
    }  
}
```

LCOM 1: 1

P: Methodenpaare mit mindestens einer gemeinsamen Variable $P = \{(1,2)\}$
Q: restliche Paare $\{(1,3), (2,3)\}$
 $= |Q| - |P|$, mindestens 0

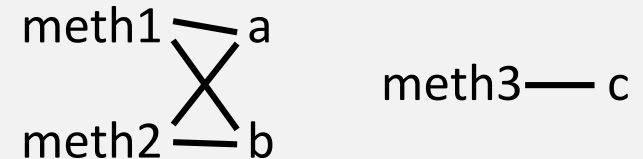
LCOM 2: 0.44444

var = 3 // Anzahl Variablen
m = 3 // Anzahl Methoden
nutzt(a)=2 nutzt(b)=2
nutzt(c)=1 sum = 2+2+1
 $= 1 - (\text{sum}/(m*\text{var}))$
 $= 1 - (5/9) = 4/9$

LCOM 3: 0.66666

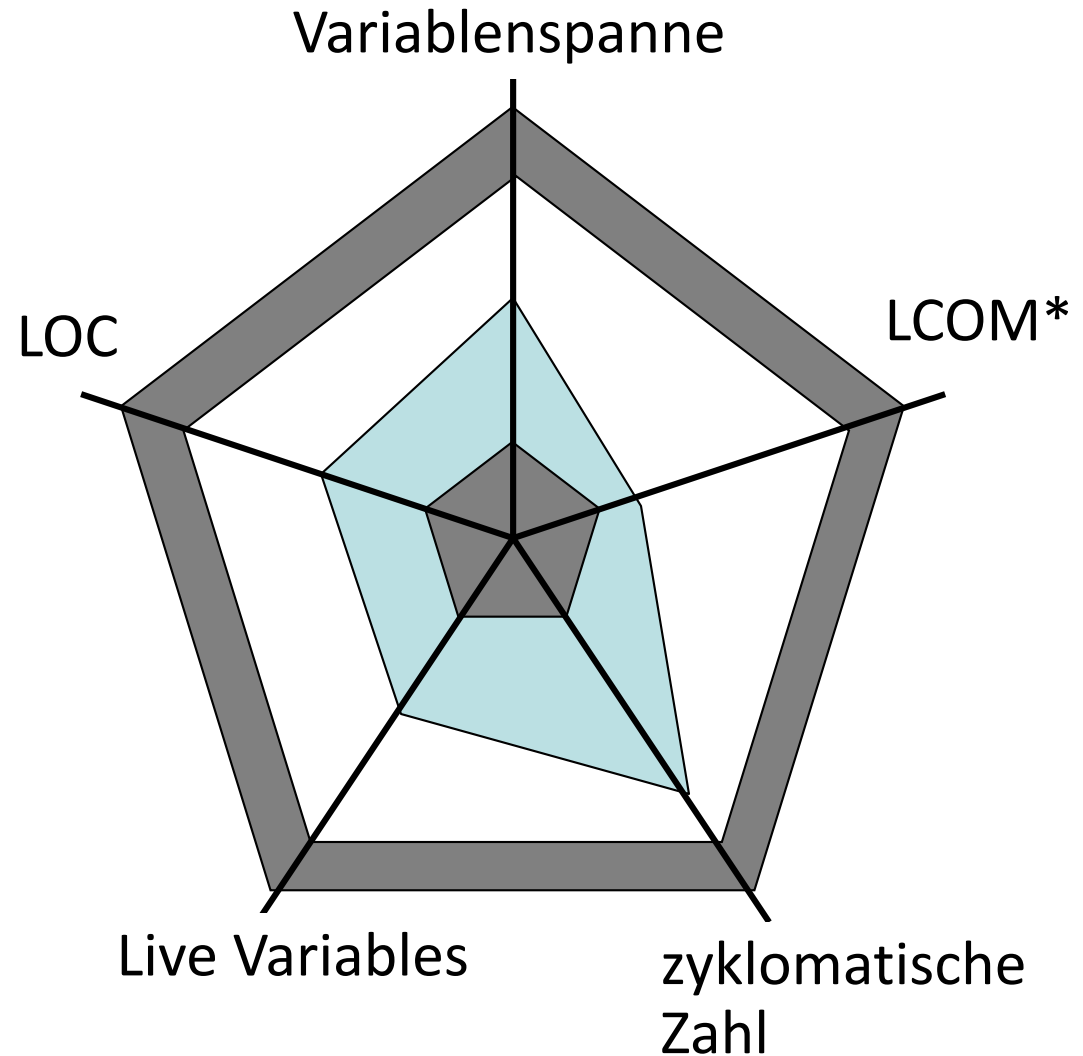
LCOM 4: 2

Anzahl getrennter Graphen bei Zugriffen



Beispiel eines Kivat-Diagramms

Maßzahlen können graphisch dargestellt werden, im Kivat-Diagramm steht jede Achse für eine Metrik, der weiße Bereich ist ok, die anderen Bereiche kritisch



10. Konstruktive Qualitätssicherung



- Idee
- Coding Guidelines
- Werkzeugeinstellungen
- weitere Maßnahmen

- die analytische Qualitätssicherung greift erst, wenn ein Produkt erstellt wurde
- interessant ist der Versuch, Qualität bereits bei der Erstellung zu beachten
- typische konstruktive Qualitätsmaßnahmen sind
 - Vorgabe der SW-Entwicklungsumgebung mit projekteigenem Werkzeughandbuch, was wann wie zu nutzen und zu lassen ist
 - Stilvorgaben für Dokumente und Programme (sogenannte Coding-Guidelines)
- Die Frage ist, wie diese Maßnahmen überprüft werden
 - Codereviews (gerade bei neuen Personen in der Entwicklung)
 - werkzeugunterstützt (nächste Folien)

- Detailliertes Beispiel: Taligent-Regeln für C++ (<http://root.cern.ch/TaligentDocs/TaligentOnline/DocumentRoot/1.0/Docs/index.html>)
- Sun hat auch Regeln für Java herausgegeben (nicht ganz so stark akzeptiert)
- z. B. Eclipse-Erweiterung Checkstyle
- Generell gibt es Regeln
 - zur Kommentierung,
 - zu Namen von Variablen und Objekten (z.B. Präfix-Regeln),
 - zum Aufbau eines Programms (am schwierigsten zu formulieren, da die Programmarchitektur betroffen ist und es nicht für alle Aspekte „die OO-Regeln“ gibt)

Beispiel-Coding-Regel

Line wrapping for `if` statements should generally use the 8-space rule, since conventional (4 space) indentation makes seeing the body difficult. For example:

```
//DON'T USE THIS INDENTATION
if ((condition1 && condition2)
    || (condition3 && condition4)
    || !(condition5 && condition6)) { //BAD WRAPS
    doSomethingAboutIt();           //MAKE THIS LINE EASY TO MISS
}
```

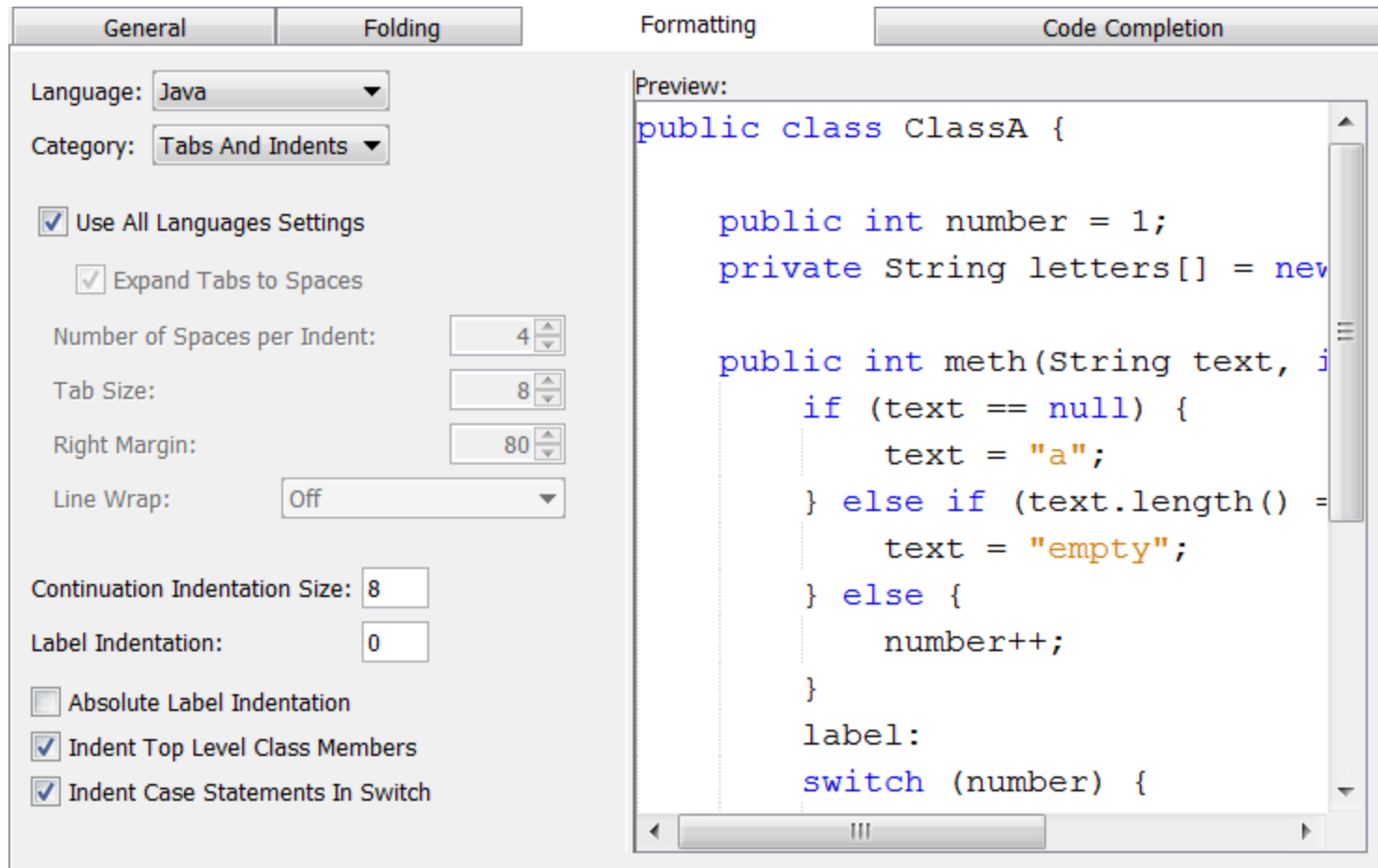
```
//USE THIS INDENTATION INSTEAD
if ((condition1 && condition2)
    || (condition3 && condition4)
    || !(condition5 && condition6)) {
    doSomethingAboutIt();
}
```

```
//OR USE THIS
if ((condition1 && condition2) || (condition3 && condition4)
    || !(condition5 && condition6)) {
    doSomethingAboutIt();
}
```

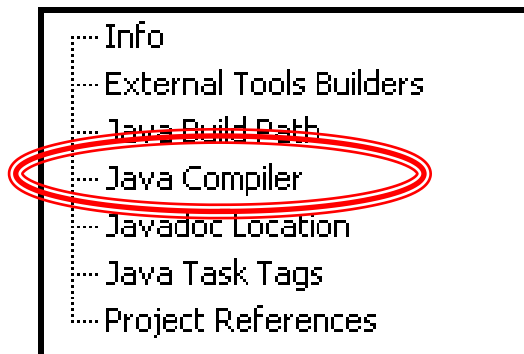
- Ausschnitt aus „Java Code Conventions“, Sun, 1997
- Inhalt soll sich nicht nur auf Formatierung beziehen

Einheitliche Werkzeugeinstellungen

- Vor Projekt einheitliche Formatierung festlegen
- Styleguide für verwendete Werkzeuge



Properties for tmp



In Eclipse kann „Schärfe“ der Syntaxprüfung eingestellt werden. Grundsätzlich sollte die schärfste Version eingestellt werden (solange es keinen wesentlichen Mehraufwand beim Beheben potenzieller Fehler gibt)

Java Compiler

- Use workspace settings
- Use project settings

Problems | Style | Compliance and Classfiles | Build Path

Select the severity level for the following problems:

Unreachable code:

Unresolvable import statements:

Unused local variables (i.e. never read):

Unused parameters (i.e. never read):

Unused imports:

Unused private types, methods or fields:

Usage of non-externalized strings:

Usage of deprecated API:

Signal use of deprecated API inside deprecated code.

- Pattern dienen zur sinnvollen Strukturierung komplexer, aber gleichartiger Systeme
- Anti-Pattern sind wiederkehrende schlechte Lösungen, die man an Strukturen erkennen kann, z. B.
 - Spaghetti-Code, viele if, while und repeat-Schleifen gemischt, intensive Nutzung der Möglichkeiten mit break, früher: goto
 - Cut-and-Paste-Programmierung: „was oben funktionierte, funktioniert hier auch“
 - allmächtige Klassen, kennen jedes Objekt, sitzen als Spinne im Klassendiagramm, immer „gute“ Quelle für Erweiterungen
 - Rucksack-Programmierung: bei vergessenem Sonderfall in allen betroffenen Methoden
 - if (Sonderfall){ Reaktion } else { altes Programm}
- Literatur (z. B.): W. J. Brown, R. C. Malveau, H. W. McCormick III, T. J. Mowbray, AntiPatterns, Wiley, 1998

hierzu gehören einige Maßnahmen des proaktiven Risikomanagements

- Berücksichtigung von Standards
- richtiges Personal mit Erfahrungen und potenziellen Fähigkeiten finden (evtl. Coaching organisieren)
- frühzeitig Ausbildungen durchführen (niedriger Truckfaktor)
- frühzeitig passende Werkzeuge finden (Nutzungsregeln)
- Vorgehensmodell mit Reaktionsmöglichkeiten bei Problemen (generell: gelebtes flexibles Prozessmodell)
- Unabhängigkeit der Qualitätssicherung
- Erfahrungen im Anwendungsgebiet

11. Performance und Speicherauslastung



- Java-Parameter mit Performance-Einfluss
- Versteckte Speicherlecks
- Direkte Zeitmessung in Java
- Konzept von Performance-Messwerkzeugen
- Netbeans-Profiler

- Lasttests
- Apache JMeter

Typische Probleme (1/2)



- Programme zur Zeit- und Speichermessung beeinflussen Laufzeit und verbrauchten Speicher
- Gerade bei Laufzeitbetrachtungen können durch langsamere Abläufe neue Effekte entstehen
- Testszenario muss in realistischer Zeit messbar sein
- generell sollten auf Testrechner wenig oder einfach bzgl. Speicher- und Rechenzeitverbrauch zu kalkulierende Programme laufen (virtuelle Maschinen, Docker Images)
- oftmals ist mehrmalige Messwiederholung sinnvoll
- generell: Tests müssen wiederholbar sein
- immer zwischen Startphase (Programm wird geladen, Speicher allokiert, in Java erste Klassen geladen) und eingeschwungener Phase (keine „überraschenden“ Ressourcennutzungen) unterscheiden

Typische Probleme (2/2)



- alle potenziellen Bottlenecks vorher analysieren, eventuell mit Alternativen Messungen wiederholen (wenig Speicher, langsame Festplatte, langsame Netzverbindung, langsamer nicht kontrollierbarer externer Service, kein Load-Balancer, ...)
- bei Nutzung von Zufallswerten muss man Werte entweder speichern oder Versuche häufig wiederholen
- Java VM kann recht flexibel bzgl. Speicher gestartet werden; entspricht ggfls. Optimierungsaufgabe für Applikation
- man kann Strategie für Java VM Garbage Collector ändern
- ...

Video 14

Direkte Parameter für die Java VM:

- Xbatch** Disables background compilation so that compilation of all methods proceeds as a foreground task until completed.
- Xdebug** Start with the debugger enabled.
- Xnoclassgc** Disable class garbage collection.
- Xincgc** Enable the incremental garbage collector.
- Xmsn** Specify the initial size, in bytes, of the memory allocation pool. (-Xms6144k -Xms6m)
- Xmxn** Specify the maximum size, in bytes, of the memory allocation pool. (-Xmx81920k -Xmx80m)
- Xssn** Set thread stack size.

<http://download.oracle.com/javase/8/docs/technotes/tools/windows/java.html>

Direkte Parameter für den Java-Compiler:

- g Generate all debugging information, including local variables.
- verbose This includes information about each class loaded and each source file compiled.
- verbose:class Display info about each class loaded.
- verbose:gc Report on each garbage collection event.
- target *version* Generate class files that target a specified version of the VM. (**1.1 1.2 1.3 1.4 1.5** (also **5**) **1.6** (also **6**) ...)
- Xlint Enable all recommended warnings. (Passt hier nicht hin, ist aber wichtig für QS 😊)

<http://download.oracle.com/javase/8/docs/technotes/tools/windows/javac.html>

<https://docs.oracle.com/en/java/javase/21/>

Verstecktes Speicherleck (1/3)

```
package boese;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class MachAuf {

    public List<FileWriter> fwl = new ArrayList<FileWriter>();

    public void steuern() throws IOException {
        int anzahl = 0;
        while (anzahl != 42) {
            System.out.print("Anzahl: ");
            anzahl = Eingabe leseInt();
            System.out.print("Datei: ");
            oeffnen(Eingabe leseString(), anzahl);
        }
    }
}
```

Verstecktes Speicherleck (2/3)

```
public void oeffnen (String name, int anzahl)
                    throws IOException {
    for (int i = 0; i < anzahl; i++){
        FileWriter fw = new FileWriter(
            new File(".\\bah\\"+name + i + ".dof"));
        fw.write(42);
        fwl.add(fw);
    }
}
```

```
public static void main(String[] arg) throws IOException {
    new MachAuf().steuern();
}
}
```

// Verzeichnis .\bah muss vorher existieren

Verstecktes Speicherleck (3/3)

Programm-
start

Name	Benutzername	CPU	CPU-Zeit	Arbeitsspeicher ...	Handles
javaw.exe	x	00	00:00:00	3.804 K	99

Anzahl: 200
Datei: Hai

Name	Benutzername	CPU	CPU-Zeit	Arbeitsspeicher ...	Handles
javaw.exe	x	00	00:00:00	6.088 K	299

Anzahl: 550
Datei: Bingbong

Name	Benutzername	CPU	CPU-Zeit	Arbeitsspeicher ...	Handles
javaw.exe	x	00	00:00:02	11.904 K	849

Anzahl: 1200
Datei: Ibah

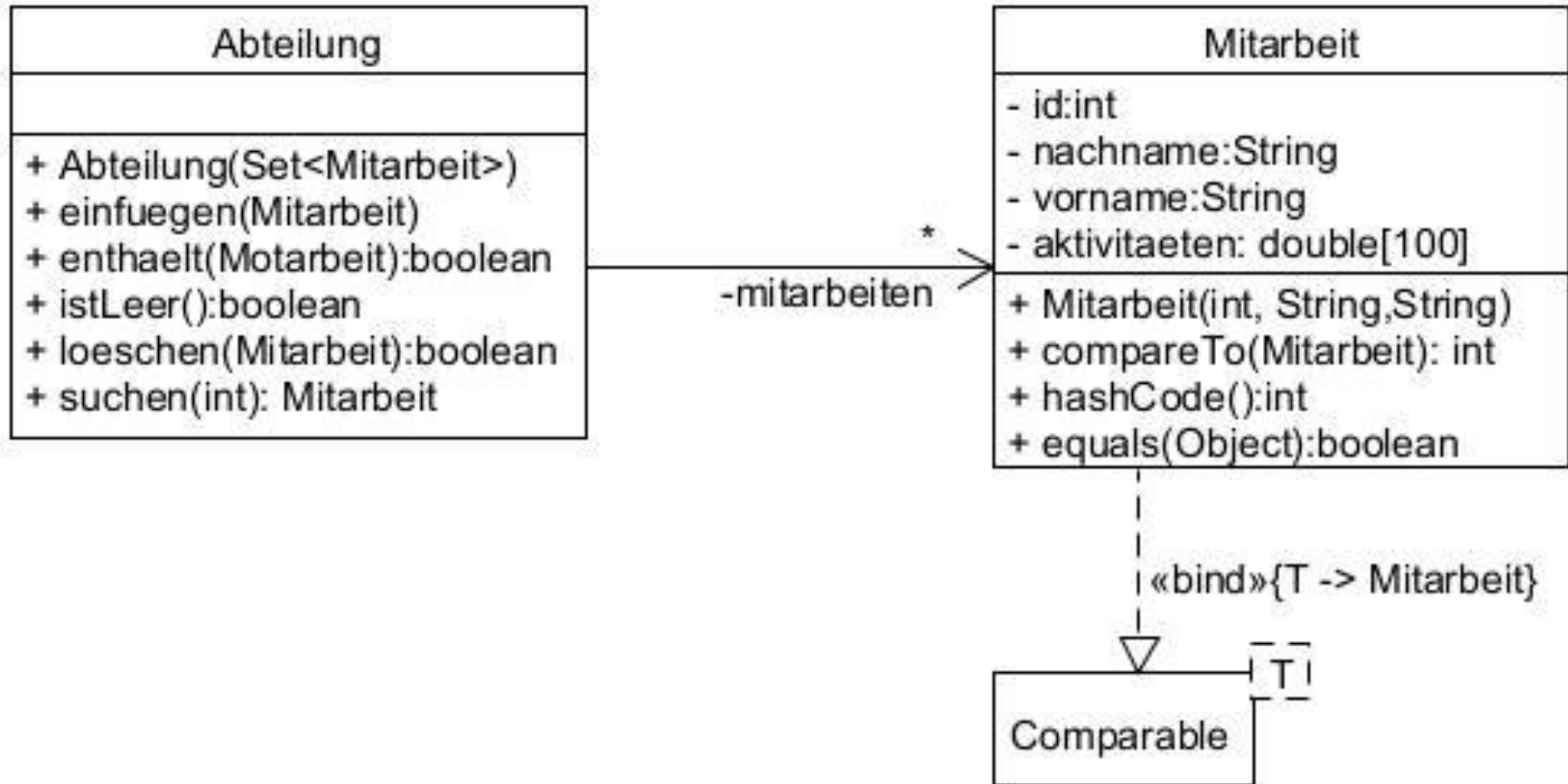
Name	Benutzername	CPU	CPU-Zeit	Arbeitsspeicher ...	Handles
javaw.exe	x	00	00:00:06	23.076 K	2.049

Name	Größe	Geändert	Typ
..		23.11.2010 1...	Dateiordner
Bingbong0.dof	0 KB	23.11.2010 1...	DOF-Datei

1/1950 Objekt(e) markiert 0 KB 23.11.2010 15:42:56 A (Frei 54,31 GB)

Zeitmessung selbst gestrickt (1/9)

- Szenario: Abteilung mit mehreren Mitarbeitern
- Frage nach passenden Typ für mitarbeiten



Zeitmessung selbst gestrickt (2/9) - Ausschnitte

```
public class Mitarbeiter implements Comparable<Mitarbeiter> {
    private int id;
    private String vorname;
    private String nachname;
    private double[] aktivitaeten = new double[100];

    public Mitarbeiter(int id, String vorname, String nachname) {
        this.id = id;
        this.vorname = vorname;
        this.nachname = nachname;
        for (int i = 0; i < Mitarbeiter.GROESSE; i++) {
            this.aktivitaeten[i] = Math.random();
        }
    }

    @Override
    public int compareTo(Mitarbeiter other) {
        return this.id - other.getId();
    }
}

// ... Software-Qualität
}
```

Zeitmessung selbst gestrickt (3/9) – Abteilung 1/2

```
public class Abteilung {  
    private Set<Mitarbeit> mitarbeiten;  
    public Abteilung(Set<Mitarbeit> mitarbeiten) {  
        this.mitarbeiten = mitarbeiten;  
    }  
  
    public void einfuegen(Mitarbeit m) {  
        this.mitarbeiten.add(m);  
    }  
  
    public Mitarbeit suchen(int minr) { // vollstaendig  
        Mitarbeit mi = null;           // durchiterieren  
        for (Mitarbeit m : this.mitarbeiten) {  
            if (m.getId() == minr) {  
                mi = m;  
            }  
        }  
        return mi;  
    }  
}
```

```
public boolean enthaelt(Mitarbeit m) {  
    return this.mitarbeiten.contains(m);  
}  
  
public boolean loeschen(Mitarbeit m) {  
    return this.mitarbeiten.remove(m);  
}  
}
```

Zeitmessung selbst gestrickt (5/9) - Testszenario

```
public class PerformanceAnalyse {

    public static int ANZAHL = 10000;
    private List<Mitarbeit> mitarbeiten = new ArrayList<>();
    private List<Integer> einfuegen = new ArrayList<>();
    private List<Integer> loeschen = new ArrayList<>();
    private List<Integer> suchen = new ArrayList<>();
    private List<Set<Mitarbeit>> testobjekte = new ArrayList<>();

    public PerformanceAnalyse() {
        this.testobjekte.add(new HashSet<>());
        this.testobjekte.add(new LinkedHashSet<>());
        this.testobjekte.add(new TreeSet<>());
        this.testobjekte.add(new CopyOnWriteArraySet<>());
        this.testobjekte.add(new ConcurrentSkipListSet<>());
        for (int i = 0; i < ANZAHL; i++) {
            this.mitarbeiten.add(new Mitarbeit(i, i+ "vor", i + "nach"));
        }

        ArrayList<Integer> nummern = new ArrayList<>(); // 5000 Nummern
        for (int i = 0; i < ANZAHL / 2; i++) {
            nummern.add(i * 2);
        }
    }
}
```

Zeitmessung selbst gestrickt (6/9) - Testszenario

```
while (!nummern.isEmpty()) {
    Integer wahl = nummern.get((int) (Math.random() * nummern.size()));
    this.einfuegen.add(wahl); // 5000 einzufuegende Mitarbeiter
    nummern.remove(wahl);
}

for (int i = 0; i < ANZAHL; i++) { // 10000 Nummern
    nummern.add(i);
}

while (!nummern.isEmpty()) {
    Integer wahl = nummern.get((int) (Math.random() * nummern.size()));
    this.loeschen.add(wahl); // 10000 Mitarbeiter in zufälliger Folge
    nummern.remove(wahl);
}

for (int i = 0; i < ANZAHL * 10; i++) { // 100000 zu löschende MA
    this.loeschen.add(0, (int) (Math.random() * ANZAHL));
}

for (int i = 0; i < ANZAHL * 10; i++) { // 100000 MA suchen
    this.suchen.add((int) (Math.random() * ANZAHL));
}

analyse();
}
    Software-Qualität
```

Zeitmessung selbst gestrickt (7/9) - Testszenario



```
private void loeschen(Abteilung abteilung) {
    for (int i : this.loeschen) {
        abteilung.loeschen(this.mitarbeiten.get(i));
    }
}

private void suchen(Abteilung abteilung) {
    for (int i : this.suchen) {
        abteilung.enthaelt(this.mitarbeiten.get(i));
    }
}

private void iterieren(Abteilung abteilung) {
    for (int i : this.suchen) {
        abteilung.suchen(i);
    }
}

private void einfuegen(Abteilung abteilung) {
    for (int anz = 0; anz < ANZAHL / 100; anz++) {
        for (int i : this.einfuegen) {
            abteilung.einfuegen(this.mitarbeiten.get(i));
        }
    }
}
}
Software-Qualität
```


Zeitmessung selbst gestrickt (8/9) - Testszenario



```
public void analyse() {
    long zeit;
    for (int i = 0; i < this.testobjekte.size(); i++) {
        Set<Mitarbeit> testobjekt = this.testobjekte.get(i);
        System.out.println("Analyse von: " + testobjekt.getClass());
        Abteilung abteilung = new Abteilung(testobjekt);
        zeit = System.currentTimeMillis();
        this.einfuegen(abteilung);
        System.out.println("    einfuegen:\t"
            + (System.currentTimeMillis() - zeit) + " ms");
        zeit = System.currentTimeMillis();
        this.iterieren(abteilung);
        System.out.println("    iterieren:\t"
            + (System.currentTimeMillis() - zeit) + " ms");
        zeit = System.currentTimeMillis();
        this.suchen(abteilung);
        System.out.println("    suchen:\t"
            + (System.currentTimeMillis() - zeit) + " ms");
        zeit = System.currentTimeMillis();
        this.loeschen(abteilung);
        System.out.println("    loeschen:\t"
            + (System.currentTimeMillis() - zeit) + " ms");
    }
}
```

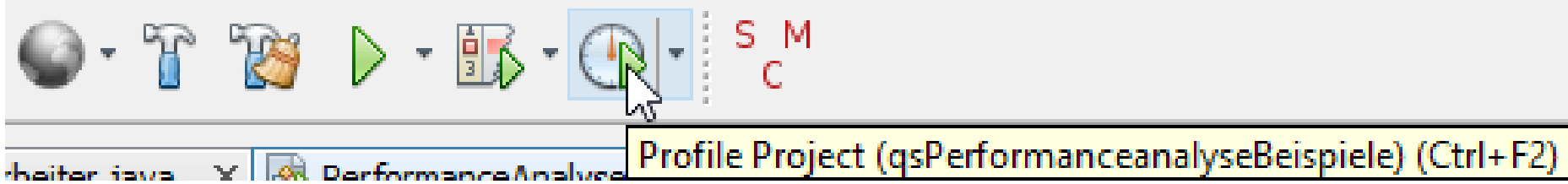
Zeitmessung selbst gestrickt (9/9) - Ergebnisse

Klasse	einfüegen	iterieren	suchen	loeschen
HashSet	210	12421	48	50
	206	12109	47	50
LinkedHashSet	241	6038	46	67
	230	5919	43	56
TreeSet	128	13875	28	17
	135	13508	31	20
CopyOnWriteArraySet	4659	8697	1787	155
	4576	8678	1715	145
ConcurrentSkipListSet	232	10526	46	30
	231	9857	47	31

in Millisekunden

- ähnlich zum letzten Beispiel wird Java-Code erweitert
- `java -agentlib:hprof` (hprof als Beispiel)
- Erweiterungen melden Informationen an Messerwerkzeug, welches protokolliert
- Meldungen sollen erlauben, das Verhalten des Messwerkzeugs heraus zu rechnen, genauer:
 - Java hat Java Virtual Machine Tool Interface (JVMTI)
 - ermöglicht als Aufrufargument einen *Java Agent*
 - *Java Agent* ist spezielle Klasse
 - aufgerufen bevor irgendwas passiert (vor main)
 - Java Agent kann Filter installieren; bekommt mit, wenn Klassen geladen werden und kann diese verändern
- <http://docs.oracle.com/javase/8/docs/technotes/guides/jvmti/>
- (Ansatz vergleichbar mit Aspect-oriented Programming)






Beispiel: Netbeans Profiler (1/6)



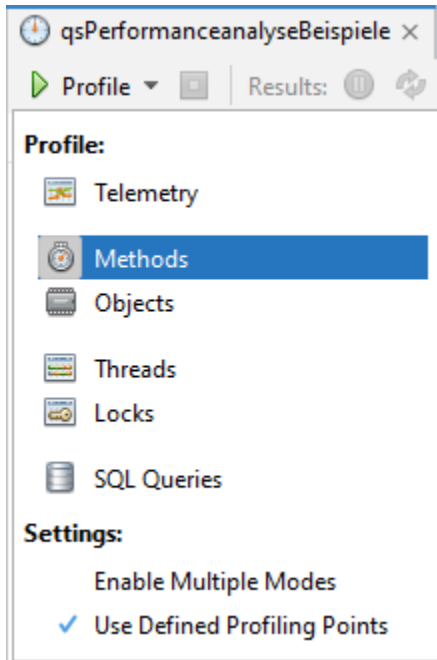
...ava PerformanceAnalyse.java x qsPerformanceanalyseBeispiele x

Configure Session ▾

Configure and Start Profiling

1. Click the [Configure Session](#) button in toolbar and select the desired profiling mode:
 -  **Telemetry** Monitor CPU and Memory usage, number of threads and loaded classes
 -  **Methods** Profile method execution times and invocation counts, including call trees
 -  **Objects** Profile size and count of allocated objects, including allocation paths
 -  **Threads** Monitor thread states and times
 -  **Locks** Collect lock contention statistics
2. Click the **Profile** button in toolbar once the session is configured to start profiling.
3. Use the Profile **dropdown arrow** to change profiling settings for the session.

Beispiel: Netbeans Profiler (2/6)



siehe z. B.

<https://www.youtube.com/watch?v=DI4EFkzqCCg>

C:\kleukersSEU\java\bin\java.exe

-agentpath:C:/tmp/netbeans/profiler/lib/installed/jdk16/windows-amd64/profilerinterface.dll

-Xbootclasspath/a:C:\tmp\netbeans\profiler\lib\jfluid-server.jar;C:\tmp\netbeans\profiler\lib\jfluid-server-15.jar

org.netbeans.lib.profiler.server.ProfilerServer
C:/tmp/netbeans/profiler/lib/installed/jdk16/windows-amd64 5141 10 ____Profiler+Calibration+Run____

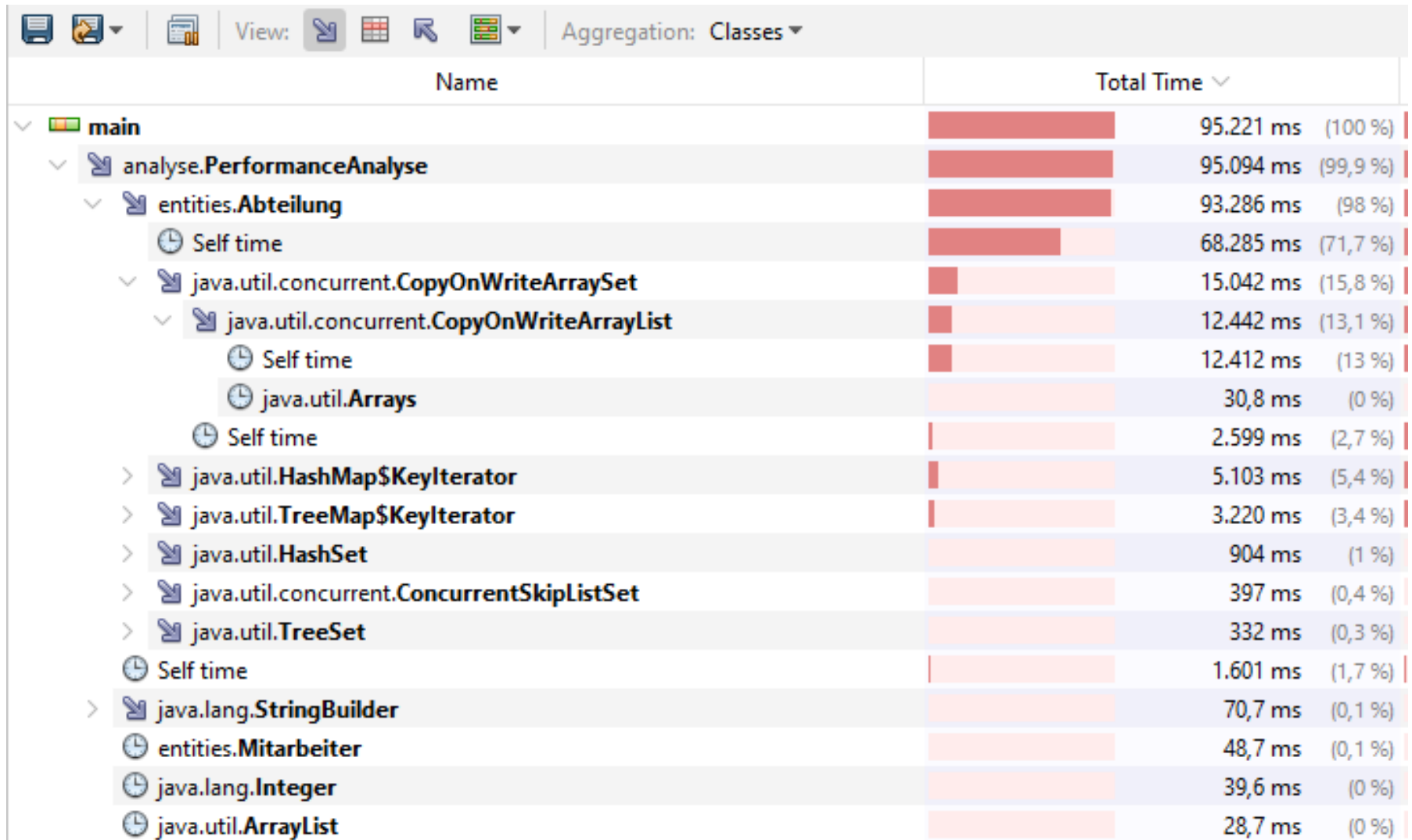


Beispiel: Netbeans Profiler (3/6) - Methods

Name	Total Time	(CPU)
main	95.094 ms	(100 %)
main	95.221 ms	(100 %)
analyse.PerformanceAnalyse	95.094 ms	(99,9 %)
analyse.PerformanceAnalyse	95.094 ms	(100 %)
analyse.PerformanceAnalyse.entities.Abteilung	93.286 ms	(98 %)
analyse.PerformanceAnalyse.entities.Abteilung Self time	68.285 ms	(71,7 %)
analyse.PerformanceAnalyse.entities.Abteilung Self time	76.640 ms	(80,6 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.concurrent.CopyOnWriteArraySet	15.042 ms	(15,8 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.concurrent.CopyOnWriteArraySet	76.371 ms	(80,3 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.concurrent.CopyOnWriteArraySet.java.util.concurrent.CopyOnWriteArrayList	12.442 ms	(13,1 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.concurrent.CopyOnWriteArraySet.java.util.concurrent.CopyOnWriteArrayList Self time	67.820 ms	(71,3 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.concurrent.CopyOnWriteArraySet.java.util.concurrent.CopyOnWriteArrayList Self time	12.412 ms	(13 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.concurrent.CopyOnWriteArraySet.java.util.concurrent.CopyOnWriteArrayList Self time	5.103 ms	(5,4 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.concurrent.CopyOnWriteArraySet.java.util.concurrent.CopyOnWriteArrayList Self time	30,8 ms	(0 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.concurrent.CopyOnWriteArraySet.java.util.concurrent.CopyOnWriteArrayList Self time	5.103 ms	(5,4 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.concurrent.CopyOnWriteArraySet.java.util.concurrent.CopyOnWriteArrayList Self time	2.599 ms	(2,7 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.concurrent.CopyOnWriteArraySet.java.util.concurrent.CopyOnWriteArrayList Self time	0,0 ms	(0 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.HashMap\$KeyIterator	5.103 ms	(5,4 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.HashMap\$KeyIterator	3.220 ms	(3,4 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.HashMap\$KeyIterator	3.220 ms	(3,4 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.HashMap\$KeyIterator	3.220 ms	(3,4 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.HashMap\$KeyIterator	904 ms	(1 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.HashMap\$KeyIterator	3.206 ms	(3,4 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.concurrent.ConcurrentSkipListSet	397 ms	(0,4 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.concurrent.ConcurrentSkipListSet	14,6 ms	(0 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.concurrent.ConcurrentSkipListSet	332 ms	(0,3 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.concurrent.ConcurrentSkipListSet	0,0 ms	(0 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.concurrent.ConcurrentSkipListSet	1.601 ms	(1,7 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.util.concurrent.ConcurrentSkipListSet	100 ms	(0,1 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.lang.StringBuilder	70,7 ms	(0,1 %)
analyse.PerformanceAnalyse.entities.Abteilung.java.lang.StringBuilder	84,4 ms	(0,1 %)
analyse.PerformanceAnalyse.entities.Abteilung.entities.Mitarbeiter	48,7 ms	(0,1 %)
analyse.PerformanceAnalyse.entities.Abteilung.entities.Mitarbeiter	15,8 ms	(0 %)
analyse.PerformanceAnalyse.entities.Abteilung.entities.Mitarbeiter	48,7 ms	(0,1 %)
analyse.PerformanceAnalyse.entities.Abteilung.entities.Mitarbeiter	39,6 ms	(0 %)
analyse.PerformanceAnalyse.entities.Abteilung.entities.Mitarbeiter	39,6 ms	(0 %)
analyse.PerformanceAnalyse.entities.Abteilung.entities.Mitarbeiter	28,7 ms	(0 %)
analyse.PerformanceAnalyse.entities.Abteilung.entities.Mitarbeiter	28,7 ms	(0 %)

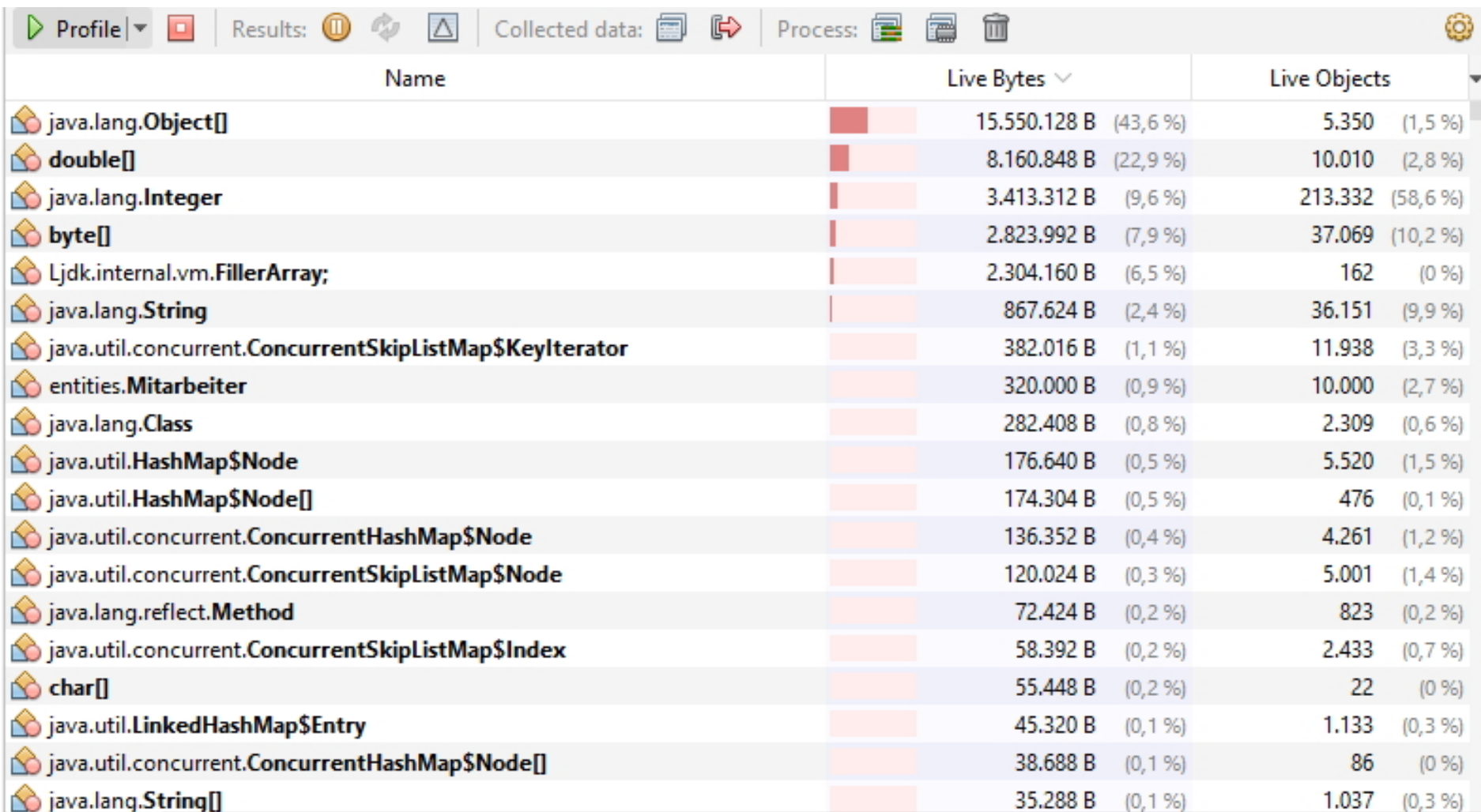
typischerweise analysiert man Zeiten selbst geschriebener Methoden;
man erkennt auch den Zeitverbrauch gegebener Klassen

Beispiel: Netbeans Profiler (4/6) - Classes



Name	Total Time
main	95.221 ms (100 %)
analyse.PerformanceAnalyse	95.094 ms (99,9 %)
entities.Abteilung	93.286 ms (98 %)
Self time	68.285 ms (71,7 %)
java.util.concurrent.CopyOnWriteArraySet	15.042 ms (15,8 %)
java.util.concurrent.CopyOnWriteArrayList	12.442 ms (13,1 %)
Self time	12.412 ms (13 %)
java.util.Arrays	30,8 ms (0 %)
Self time	2.599 ms (2,7 %)
java.util.HashMap\$KeyIterator	5.103 ms (5,4 %)
java.util.TreeMap\$KeyIterator	3.220 ms (3,4 %)
java.util.HashSet	904 ms (1 %)
java.util.concurrent.ConcurrentSkipListSet	397 ms (0,4 %)
java.util.TreeSet	332 ms (0,3 %)
Self time	1.601 ms (1,7 %)
java.lang.StringBuilder	70,7 ms (0,1 %)
entities.Mitarbeiter	48,7 ms (0,1 %)
java.lang.Integer	39,6 ms (0 %)
java.util.ArrayList	28,7 ms (0 %)

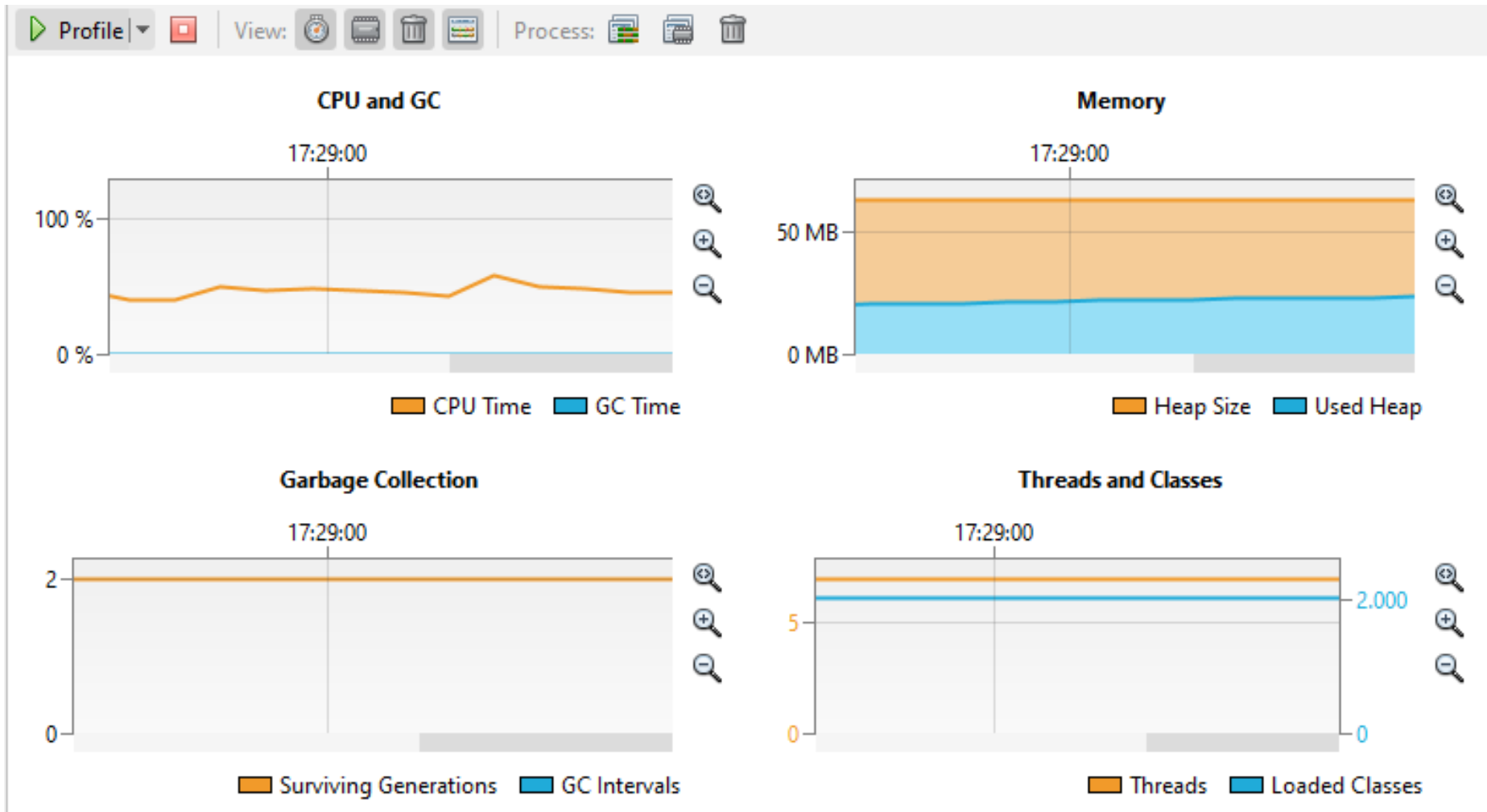
Beispiel: Netbeans Profiler (5/6) - Speicherverbrauch



The screenshot shows the NetBeans Profiler's memory usage table. The table has three main columns: 'Name', 'Live Bytes', and 'Live Objects'. Each row represents a different memory object, with its name, size in bytes, percentage of total memory, and the number of live objects. The 'Live Bytes' column includes a small red bar chart and a percentage. The 'Live Objects' column includes a small blue bar chart and a percentage. The table is sorted by 'Live Bytes' in descending order.

Name	Live Bytes	Live Objects
java.lang.Object[]	15.550.128 B (43,6 %)	5.350 (1,5 %)
double[]	8.160.848 B (22,9 %)	10.010 (2,8 %)
java.lang.Integer	3.413.312 B (9,6 %)	213.332 (58,6 %)
byte[]	2.823.992 B (7,9 %)	37.069 (10,2 %)
Ljdk.internal.vm.FillerArray;	2.304.160 B (6,5 %)	162 (0 %)
java.lang.String	867.624 B (2,4 %)	36.151 (9,9 %)
java.util.concurrent.ConcurrentSkipListMap\$KeyIterator	382.016 B (1,1 %)	11.938 (3,3 %)
entities.Mitarbeiter	320.000 B (0,9 %)	10.000 (2,7 %)
java.lang.Class	282.408 B (0,8 %)	2.309 (0,6 %)
java.util.HashMap\$Node	176.640 B (0,5 %)	5.520 (1,5 %)
java.util.HashMap\$Node[]	174.304 B (0,5 %)	476 (0,1 %)
java.util.concurrent.ConcurrentHashMap\$Node	136.352 B (0,4 %)	4.261 (1,2 %)
java.util.concurrent.ConcurrentSkipListMap\$Node	120.024 B (0,3 %)	5.001 (1,4 %)
java.lang.reflect.Method	72.424 B (0,2 %)	823 (0,2 %)
java.util.concurrent.ConcurrentSkipListMap\$Index	58.392 B (0,2 %)	2.433 (0,7 %)
char[]	55.448 B (0,2 %)	22 (0 %)
java.util.LinkedHashMap\$Entry	45.320 B (0,1 %)	1.133 (0,3 %)
java.util.concurrent.ConcurrentHashMap\$Node[]	38.688 B (0,1 %)	86 (0 %)
java.lang.String[]	35.288 B (0,1 %)	1.037 (0,3 %)

Beispiel: Netbeans Profiler (6/6) -Telemetry



auch besonders für Web-Applikationen interessant (beobachtbar)

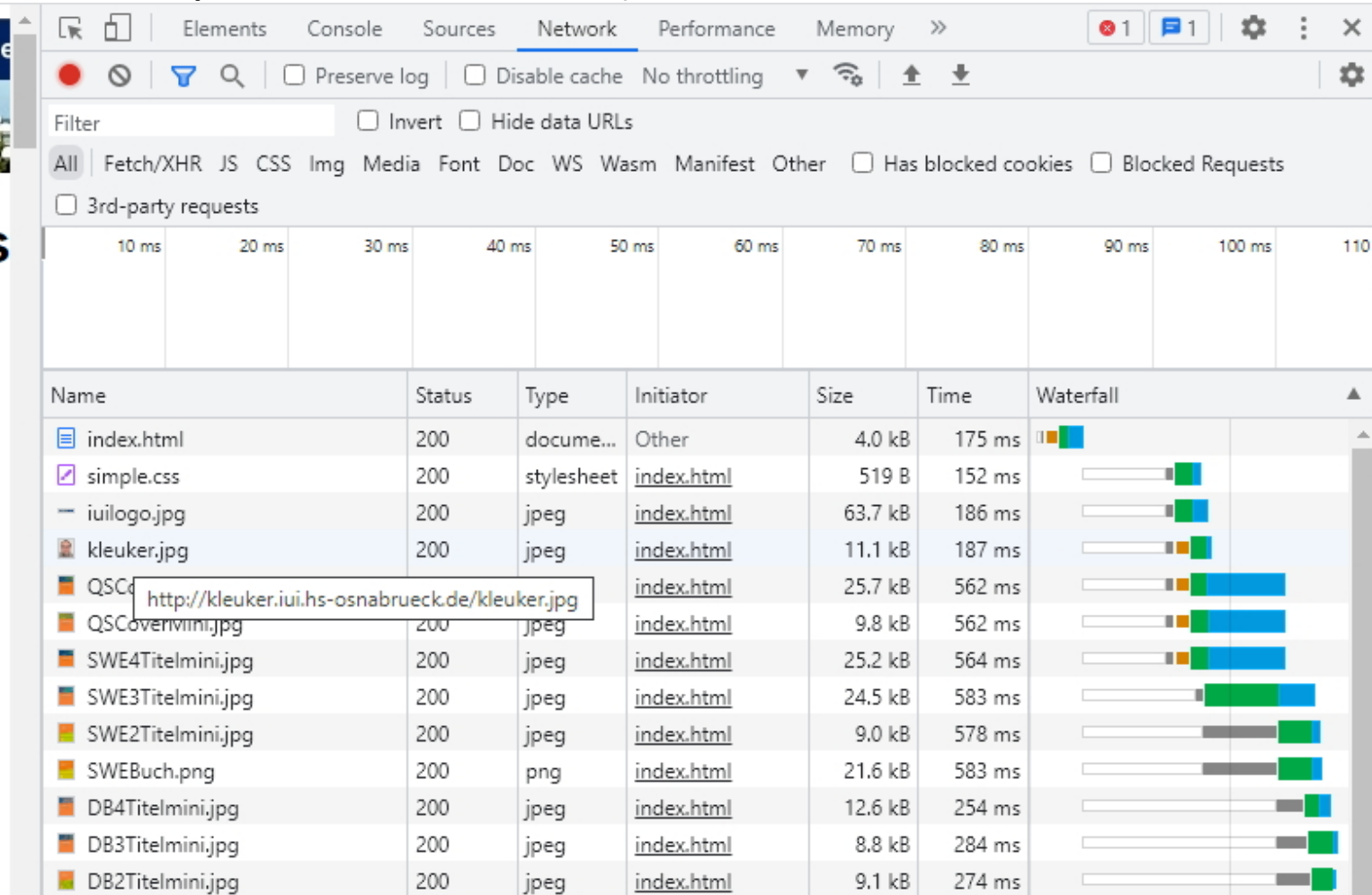
Performancemessung von Web-Applikationen

- viele Werkzeuge, die für jeden Schritt einer Seite (laden von Bildern, JavaScript-Bibliotheken, ...) z. B. direkt im Browser

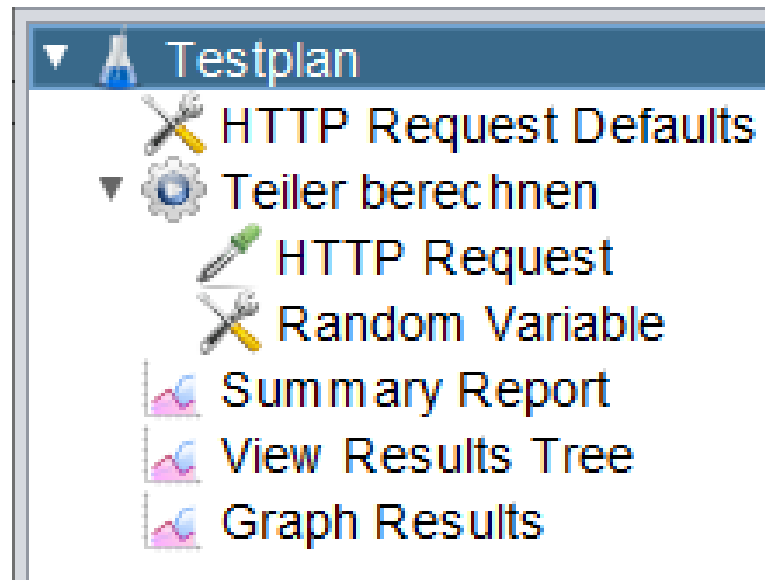
Fakultät Ingenieurwissen

Informations
Prof. Dr.
Stephan
Kleuker

Hochschule
Osnabrück
Informatik-
Studiengänge
Lehrstuhl für
Software-



- Apache JMeter
- Möglichkeit einzelne Web-Aufrufe zu spezifizieren
- die Aufrufe können dann zu einem Testplan kombiniert werden
- man kann dann festlegen, wie oft die Aufrufe zum Server geschickt werden; die Antwortzeiten werden gemessen



Ausblick: Lastmessung mit Apache JMeter (1/5)

Default-Einstellung des Servers, localhost:8080

HTTP Request Defaults

Name:

Comments:

Web Server

Protocol [http]: Server Name or IP: Port Number:

HTTP Request

Path: Content encoding:

Send Parameters With the Request:

Name:	Value	URL Encod...	Content-Type	Include Equal...
-------	-------	--------------	--------------	------------------

Ausblick: Lastmessung mit Apache JMeter (2/5)

- 400 mal aufrufen, mit Startzeit 1 Sekunde

Thread Group

Name:

Comments:

Action to be taken after a Sampler error

Continue Start Next Thread Loop Stop Thread Stop Test Stop Test Now

Thread Properties

Number of Threads (users):

Ramp-up period (seconds):

Loop Count: Infinite

Same user on each iteration

Delay Thread creation until needed

Specify Thread lifetime

Ausblick: Lastmessung mit Apache JMeter (3/5)

Aufruf eines Web-Services zur Teilerberechnung
GET localhost:8080/teiler/\${Nummer}
Nummer ist Zufallszahl

The screenshot shows the configuration for an HTTP Request in Apache JMeter. The 'Name' field is set to 'HTTP Request'. Below it, there are 'Basic' and 'Advanced' tabs. The 'Web Server' section contains fields for 'Protocol [http]', 'Server Name or IP', and 'Port Number'. The 'HTTP Request' section shows the method set to 'GET' and the path as '/teiler/\${Nummer}'. There are checkboxes for 'Redirect Automatically', 'Follow Redirects', 'Use KeepAlive', 'Use multipart/form-data', and 'Browser-compatible headers'. Below this are tabs for 'Parameters', 'Body Data', and 'Files Upload'. At the bottom, a table header for 'Send Parameters With the Request' is visible, with columns for 'Name', 'Value', 'URL Encod...', 'Content-Type', and 'Include Equal...'.

Send Parameters With the Request:				
Name:	Value	URL Encod...	Content-Type	Include Equal.

Zufallsvariable

The screenshot shows the 'Random Variable' configuration dialog in Apache JMeter. It is divided into several sections:

- Random Variable**:
 - Name: Random Variable
 - Comments: (empty)
- Output variable**:
 - Variable Name: Nummer
 - Output Format: (empty)
- Configure the Random generator**:
 - Minimum Value: 1
 - Maximum Value: 20000
 - Seed for Random function: (empty)
- Options**:
 - Per Thread(User)?: False

Hinweis: Besteht keine direkte Zugriffsmöglichkeit, wird bei Web-Applikationen der Proxy von JMeter genutzt, der im Browser eingestellt wird, dessen Aktionen dann von JMeter aufgezeichnet und für Tests genutzt werden

Ausblick: Lastmessung mit Apache JMeter (5/5)

Ergebnisdarstellung

Summary Report

Name:

Comments:

Write results to file / Read from file

Filename

Label	# Samples	Average	Min	Max	Std. Dev.	Error %
HTTP Request	400	463	1	781	270.32	15.25%
TOTAL	400	463	1	781	270.32	15.25%

Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
151.5/sec	80.31	15.98	542.
151.5/sec	80.31	15.98	542.

Label	# Samp...	Ave...	Min	Max	Std. ...	Error ...
HTTP ...	300	487	40	691	136.65	0.00%
TOTAL	300	487	40	691	136.65	0.00%

Text

- ✓ HTTP Request
- ✓ HTTP Request
- ✓ HTTP Request
- ✓ HTTP Request
- ✓ HTTP Request

Sampler result | Request | Response data

Response Body | Response headers

```
{"Wert":7494,"Teiler":[1,2,3,6,1249,2498,3747,7494]}
```


- Definition des Testszenarios ist hier sehr komplexe Aufgabe
- Testergebnisse können von vielen kleinen Parametern (Objektgrößen, Systemeinstellungen) abhängen
- Kleine Änderungen können große Effekte haben
- Performance- und Speicherverbrauchsmessung oft nicht ganz exakt durchführbar
- JMeter-Empfehlung: Tests mit GUI erstellen, dann ohne GUI laufen lassen
- Zentrale Frage: welche Methode wird wie oft aufgerufen und verbraucht wieviel Zeit
- Zentrale Frage: Welche Objekte verbrauchen wieviel Speicherplatz
- Werkzeugunterstützung ist vorhanden

12. Testautomatisierung



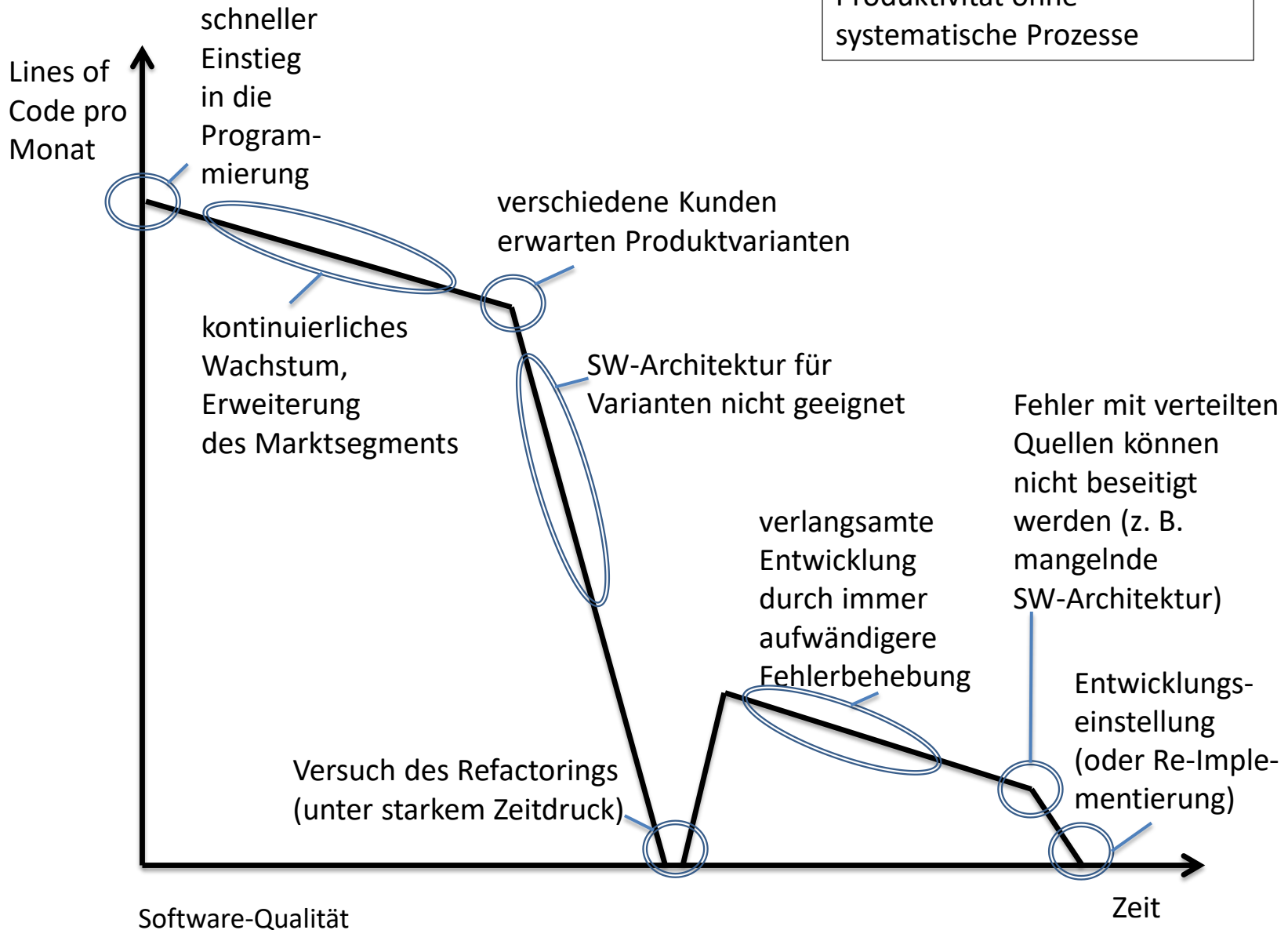
- Automatisierungsidee
- Beispiele für Werkzeuge
- Build-Server
- Idee von Build-Werkzeugen
- Einführung in Ant
- Tests aus Ant starten

- klassische Testansätze
 - Entwicklung einer Testspezifikation (Vorbedingung, Ausführung, erwartete Ergebnisse)
 - manuelle Testausführung
 - manuelle Erfassung und Auswertung der Testergebnisse
- erste Automatisierungsstufe
 - Werkzeuge wie JUnit, Marathon erlauben die automatische Testausführung und Protokollierung (teilweise Auswertung)
 - Werkzeuge müssen einzeln gestartet werden
- zweite Automatisierungsstufe
 - mehrere Werkzeuge laufen nacheinander / zusammen
 - Ergebnisse werden zentral protokolliert

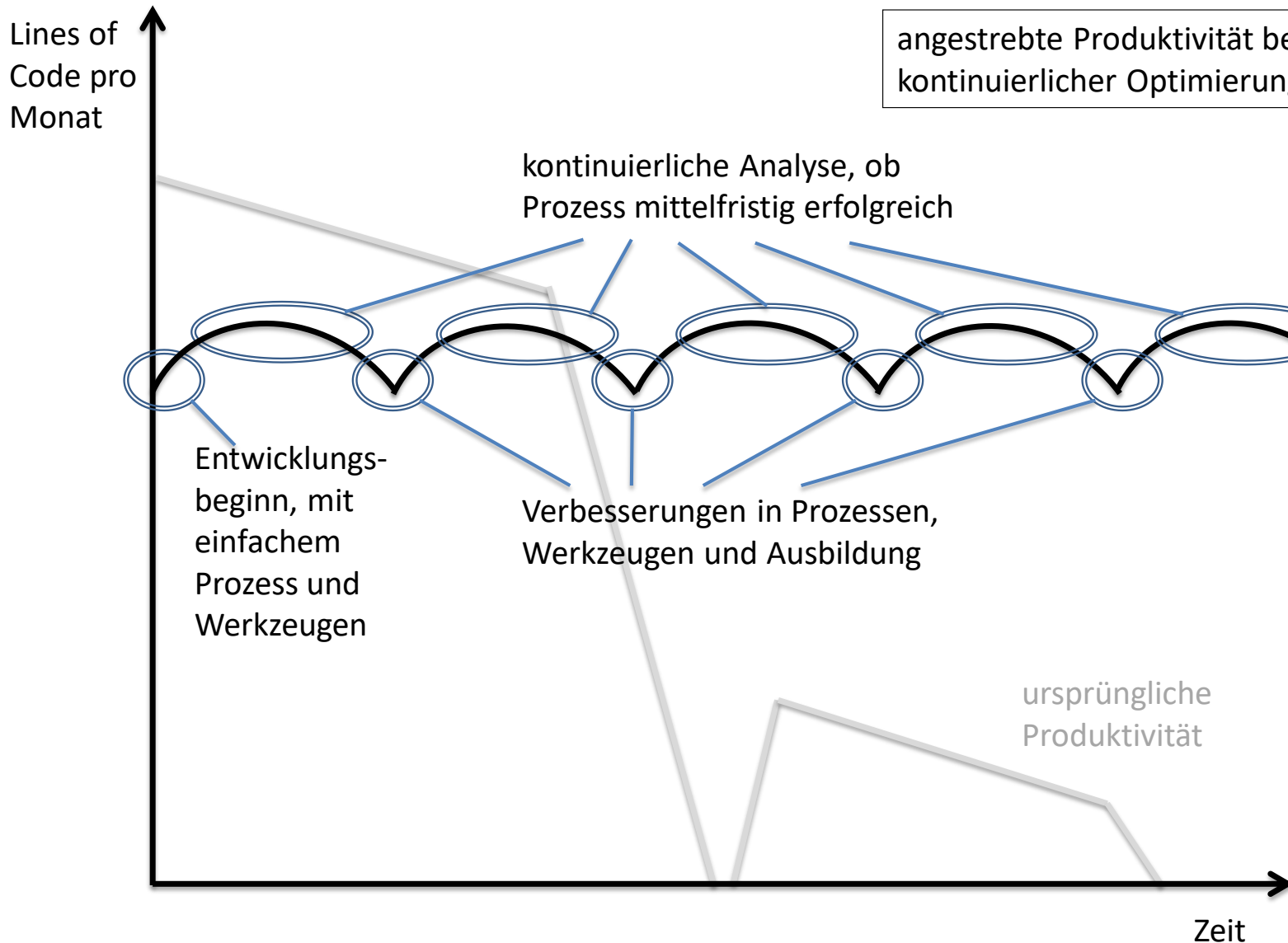
Warum Automatisierung? Gefahr



Produktivität ohne
systematische Prozesse



Warum Automatisierung? Optimierter Prozess



- Konzeption und Implementierung der Automatisierung:
 - Unit-Tests
 - GUI-Tests
 - Messung der Codeüberdeckung
 - Statische Codeanalyse
 - Softwaremetrik
- Integrierbarkeit in das bisherige Verfahren

Fallstudie: Werkzeugauswahl nach Analyse (1/2)

JUnit

Unit Test Results. Designed for use with [JUnit](#) and [Ant](#).

Class test..JUnitUtilTest

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
JUnitUtilTest	3	0	0	0.397	2012-02-09T08:03:55	asc23

Tests

Name	Status	Type	Time(s)
test	Success		0.277
test2	Success		0.003
test3	Success		0.003

QF-Test (kommerziell)

File Bearbeiten Ansicht Einfügen Operationen Aufnahme Wiedergabe Debugger Clients Extras Hilfe

*Testsuite.qft x admileoTests.qft x

Testsuite "Testsuite.qft"

- Testfallsatz: admileo checks
 - Testfall: admileo healthcheck
 - Prozeduraufruf: admileo starten()
 - Prozeduraufruf: Integrations-Testsystem Login()
 - Sequenz: Warten auf bedienbare Oberfläche
 - Try: check zeit login bis hauptfenster bedienbar
 - Prozeduraufruf: Warten auf STE Button\$(wartezeitLoginBisBedienbar)
 - Catch: TestException: Ladezeit zu lang
 - Try
 - Prozeduraufruf: Warten auf STE Button\$(wartezeitLoginBisBedienbar2)
 - Catch: TestException: STE Button nicht gefunden
 - Programm Beenden [admileo produktiv]
 - Prozeduraufruf: TestfallAbbrechen()
 - Prozeduraufruf: STE Status-check()
 - Programm Beenden [admileo produktiv]

Prozeduraufruf

Name der Prozedur
auswahl Integrationsystem
Variable für Rückgabewert

Lokale Variable
 Variablen Definition

Name

Id

Verzögerung vorher (ms) Verzögerung

OK Abbrechen

JaCoCo

admileo > Server > de.archimedon.emps.server.dataModel.beans > ProjektSettingsBean

ProjektSettingsBean

Element	Missed Instructions	Cov	Missed Branches	Cov	Missed	Cov	Missed	Lines	Missed	Methods
static [..]	100%	n/a	100%	0	1	0	28	0	1	
getIsTerminUeberwachtIndex()	100%	100%	100%	0	2	0	3	0	1	
getIsTerminAgzUeberwachtIndex()	100%	100%	100%	0	2	0	3	0	1	
getIsKostenUeberwachtIndex()	100%	100%	100%	0	2	0	3	0	1	
getIsPlanUeberwachtIndex()	100%	100%	100%	0	2	0	3	0	1	
getIsPlanAgzUeberwachtIndex()	100%	100%	100%	0	2	0	3	0	1	
getIsEinstellungenGesperrtIndex()	100%	100%	100%	0	2	0	3	0	1	
getIsUeberwachungGesperrtIndex()	100%	100%	100%	0	2	0	3	0	1	
getIsMeldestrategieGesperrtIndex()	100%	100%	100%	0	2	0	3	0	1	
getIsHLimitGesperrtIndex()	100%	100%	100%	0	2	0	3	0	1	
getIsHLimitIndex()	100%	100%	100%	0	2	0	3	0	1	
getIsTerminPlanUeberwachtIndex()	100%	100%	100%	0	2	0	3	0	1	
getSchwellenwertTerminPlanIndex()	100%	100%	100%	0	2	0	3	0	1	
getVerzoegerungTerminPlanIndex()	100%	100%	100%	0	2	0	3	0	1	
getIsTerminPlanGesperrtIndex()	100%	100%	100%	0	2	0	3	0	1	
getMeldestrategieIdIndex()	100%	100%	100%	0	2	0	3	0	1	
getWorkflowProjektabschlussIdIndex()	100%	100%	100%	0	2	0	3	0	1	
getWorkflowAnfrageIdIndex()	100%	100%	100%	0	2	0	3	0	1	
getWorkflowMehrfachIdIndex()	100%	100%	100%	0	2	0	3	0	1	
getWorkflowAnfrageIdIndex()	100%	100%	100%	0	2	0	3	0	1	

Sonar

Home admileo Configuration Log In Search

Dashboard Version 3.0.3 - 27. Jan 2012 10:53 - Time changes...

Components

Violations Drilldown

Time Machine

Clouds

Design

Hotspots

Libraries

sonar

Lines of code: 1.134.950

Classes: 9.359

Violations: 76.166

Rules compliance: 87,8%

Package tangle index: 28,4%

Dependencies to cut: 456 between packages

Complexity: 2,9 /method, 30,3 /class, 34,7 /file

Response for Class: 1,2 /class, 13,6% files having LCOM4>1

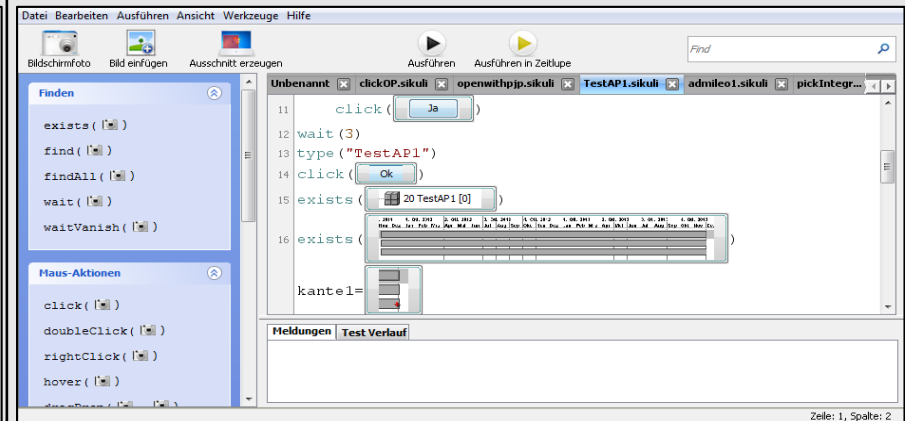
Jenkins



The screenshot shows the Jenkins dashboard with a table of build records. The table has columns for status (S), weather icon (W), name, last success, last failure, and last duration. The builds listed are GUI-Tests1, GUI-Tests2, sonar, Unit Tests, and Version 3.0.3 bauen.

S	W	Name	Letzter Erfolg	Letzter Fehlschlag	Letzte Dauer
●	☀	GUI-Tests1	34 Minuten (#60)	23 Stunden (#54)	39 Sekunden
●	☀	GUI-Tests2	34 Minuten (#2)	Unbekannt	4 Sekunden
●	☀	sonar	33 Minuten (#48)	2 Stunden 44 Minuten (#40)	1 Minute 26 Sekunden
●	☀	Unit Tests	34 Minuten (#38)	2 Stunden 3 Minuten (#34)	7,5 Sekunden
●	☁	Version 3.0.3 bauen	1 Stunde 41 Minuten (#52)	4 Tage 0 Stunden (#49)	38 Minuten

Sikulix (Capture & Replay)

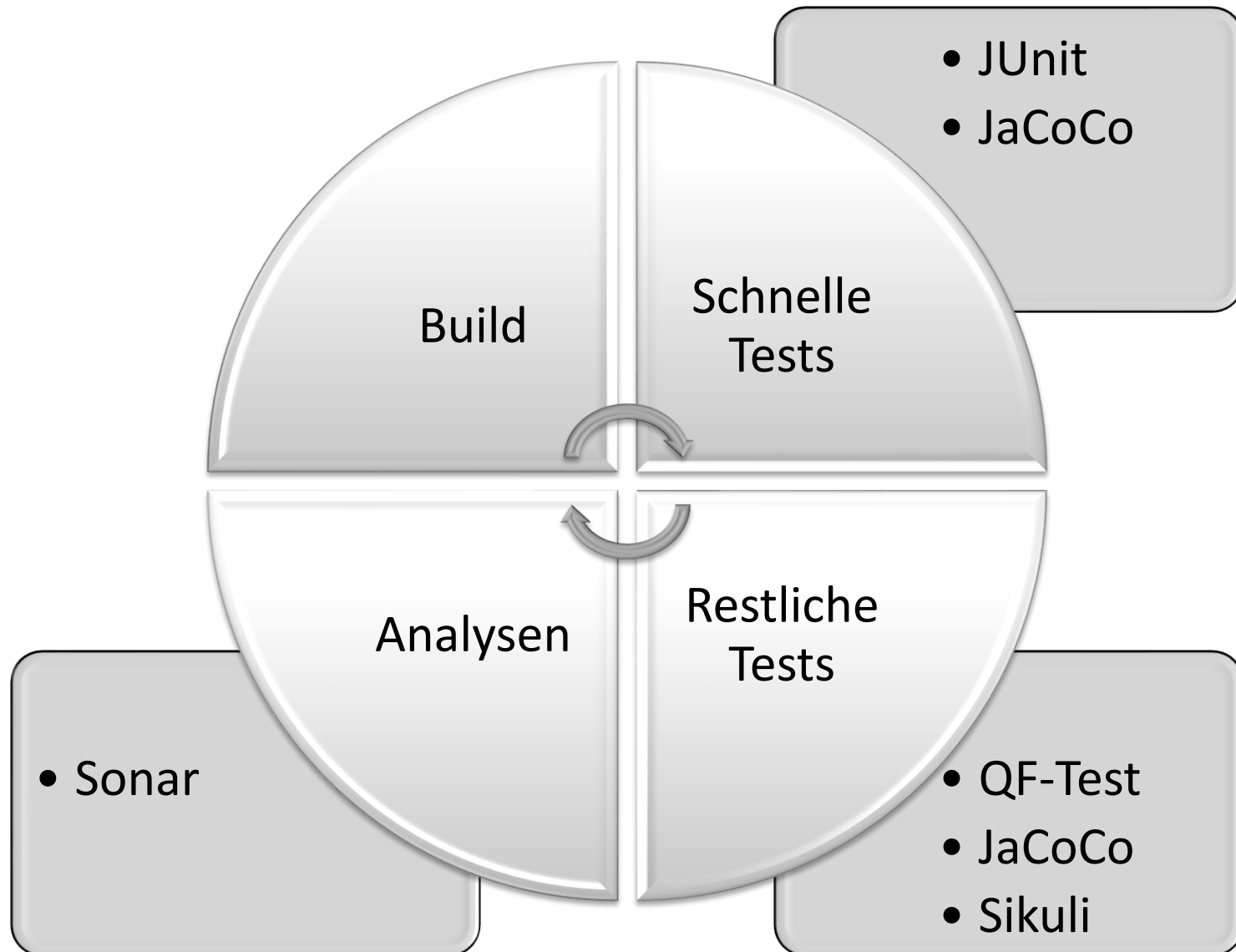


The screenshot shows the Sikulix interface with a test script. The script contains actions such as click, wait, type, find, findAll, wait, and waitVanish. A dialog box is visible over the script, and the interface includes a menu bar and a toolbar.

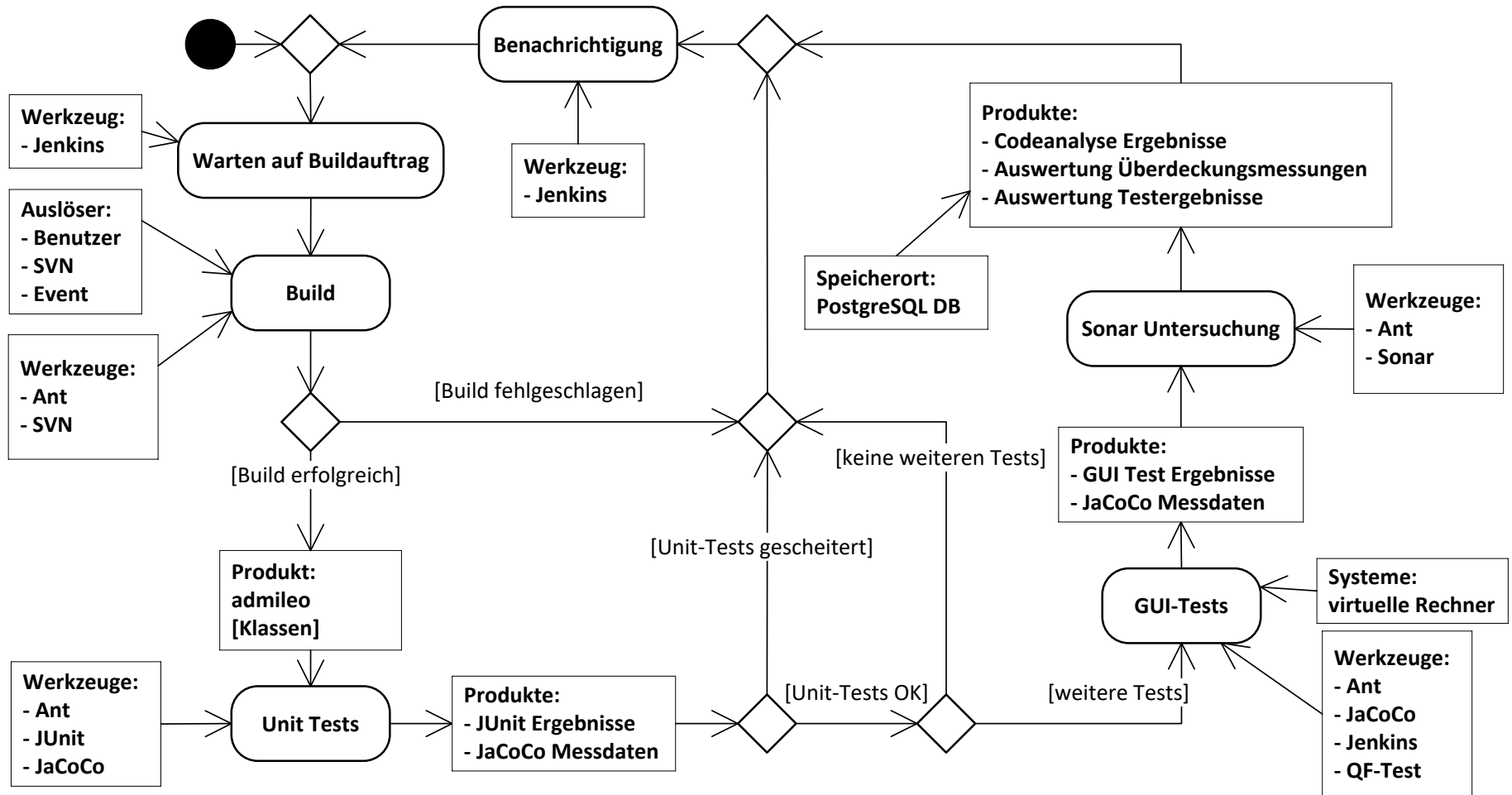
```
11 click( [Ja] )
12 wait( 3 )
13 type( "TestAP1" )
14 click( [Ok] )
15 exists( [20 TestAP1 [0]] )
16 exists( [ ] )
kante1=
```

Weitere Werkzeuge:

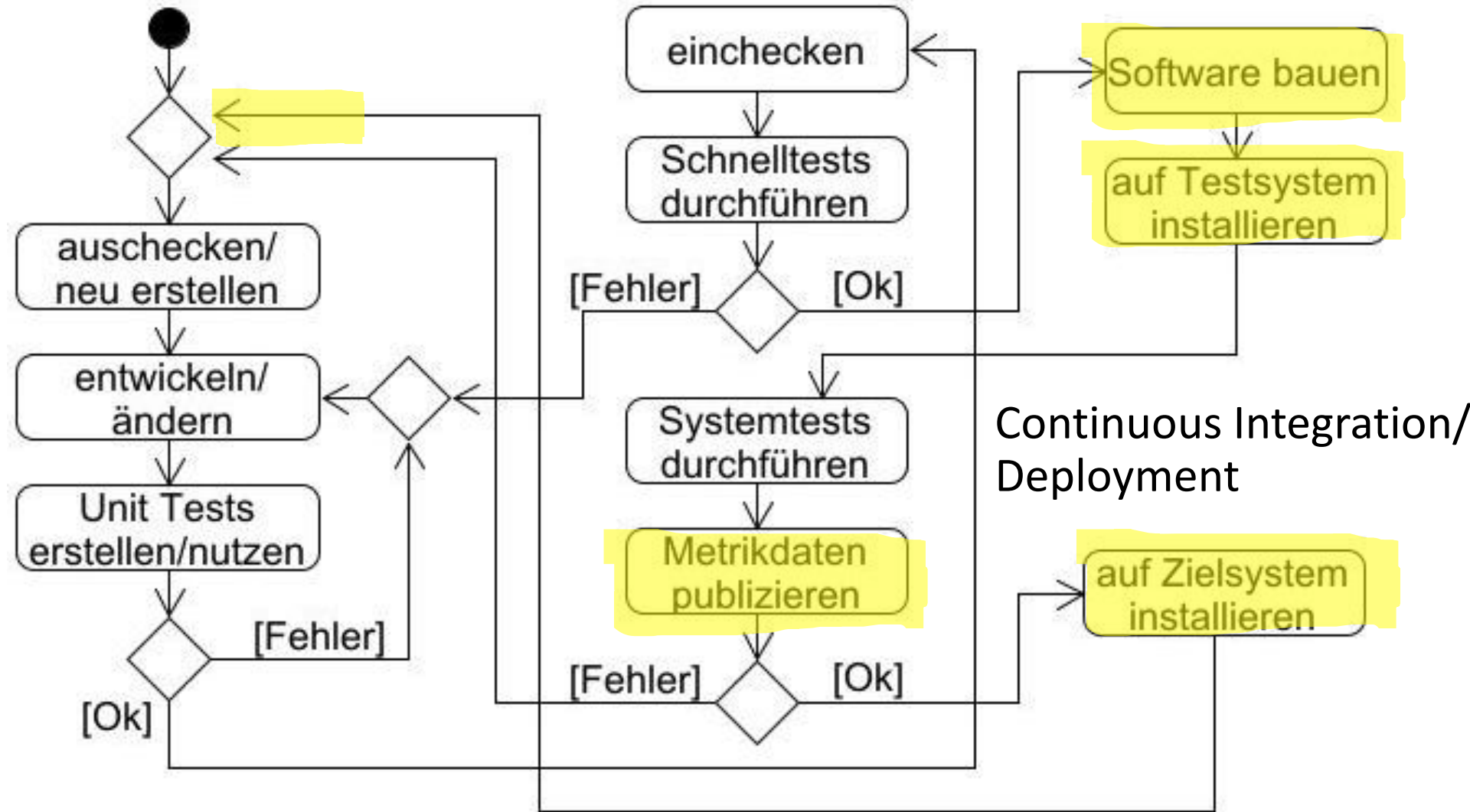
- Ant
- Hyper-V
- Jython
- PostgreSQL
- Tomcat



Fallstudie: Gesamtablauf



erweiterter Entwicklungsprozess



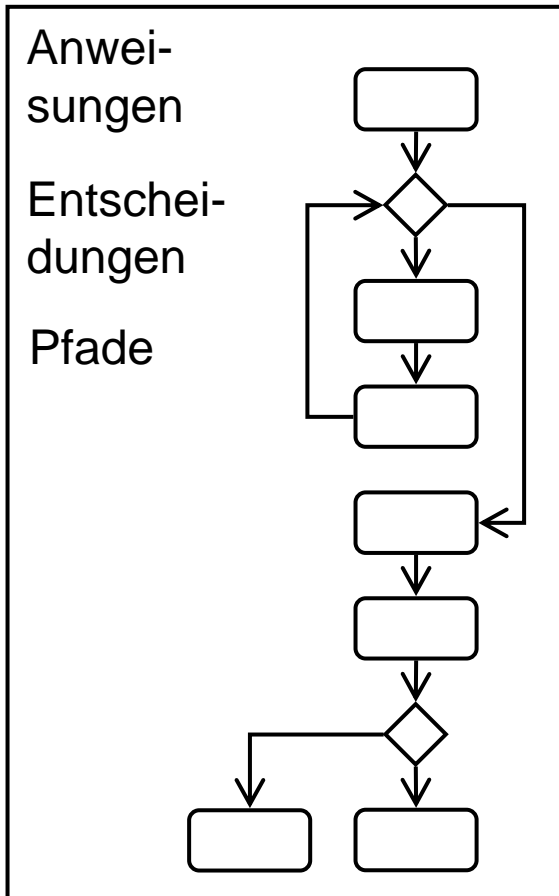
13. Organisation des QS-Prozesses in IT-Projekten

- Teststufen
- Regressionstest
- Testverfahren nach ANSI/IEEE-829
- Organisation der QS

siehe auch:

- H. M. Sneed, M. Winter, Testen objektorientierter Software, Hanser, München Wien
- A. Spillner, T. Roßner, T. Linz, Praxiswissen Softwaretest, ab 2. Auflage, dpunkt Verlag, Heidelberg

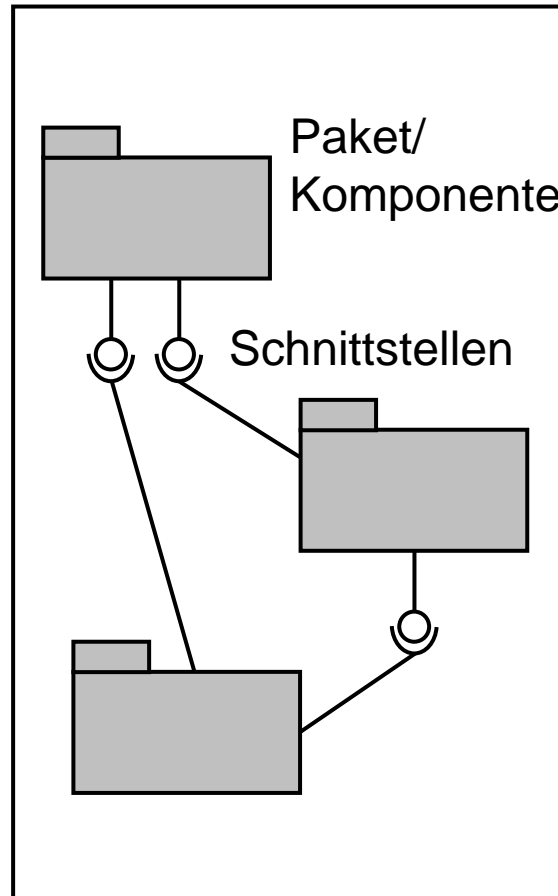
White-Box-Test



Methoden-/Klassentest

Software-Qualität

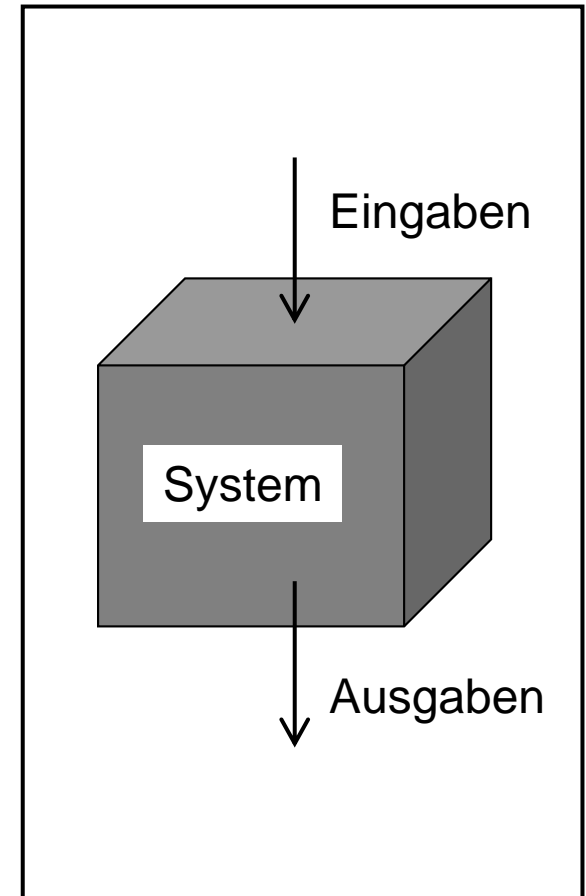
Gray-Box-Test



Integrationstest

Stephan Kleuker

Black-Box-Test

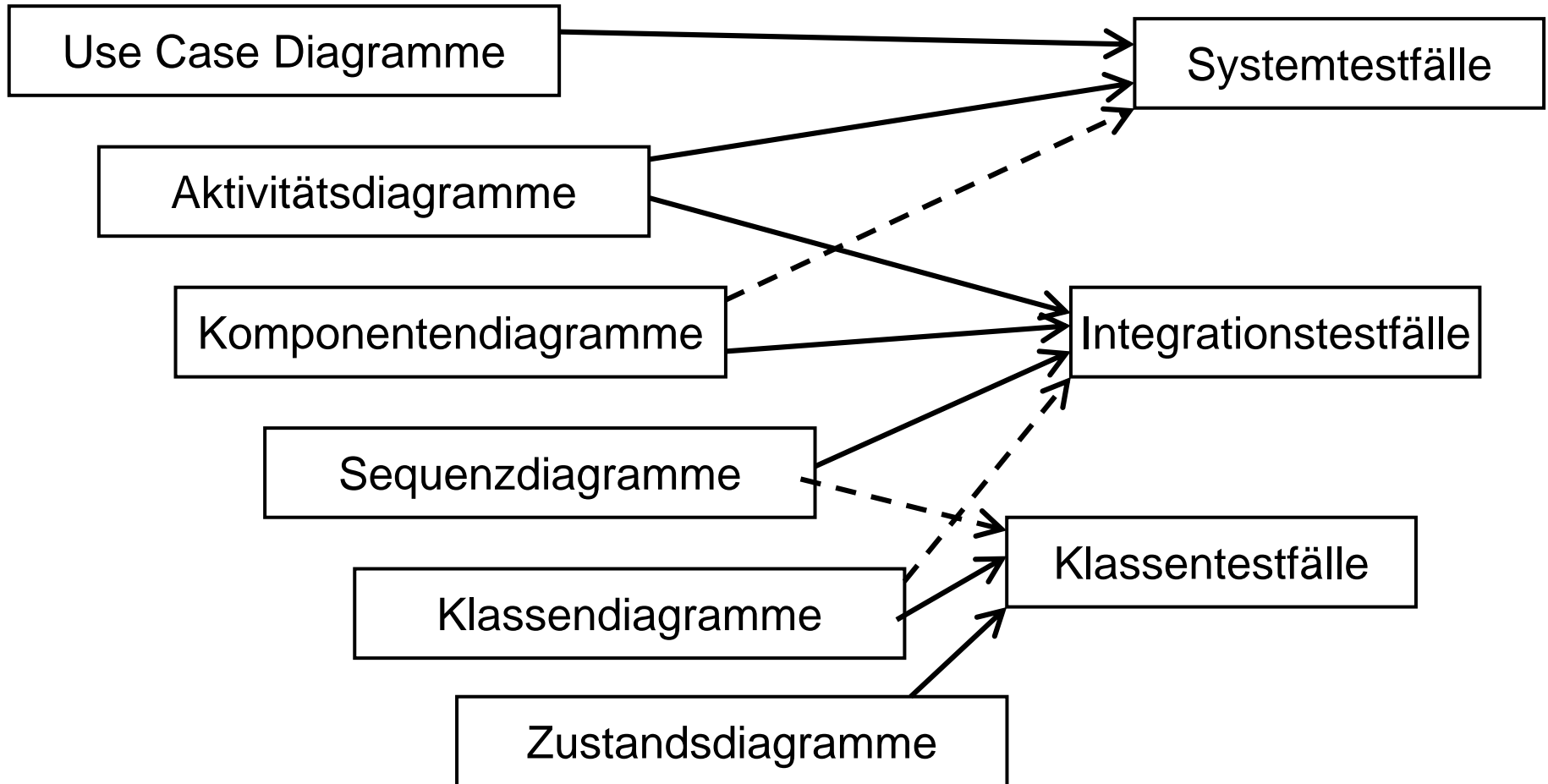


Systemtest

Testfälle und die UML (Erinnerung Tracing)

Entwicklung in der UML

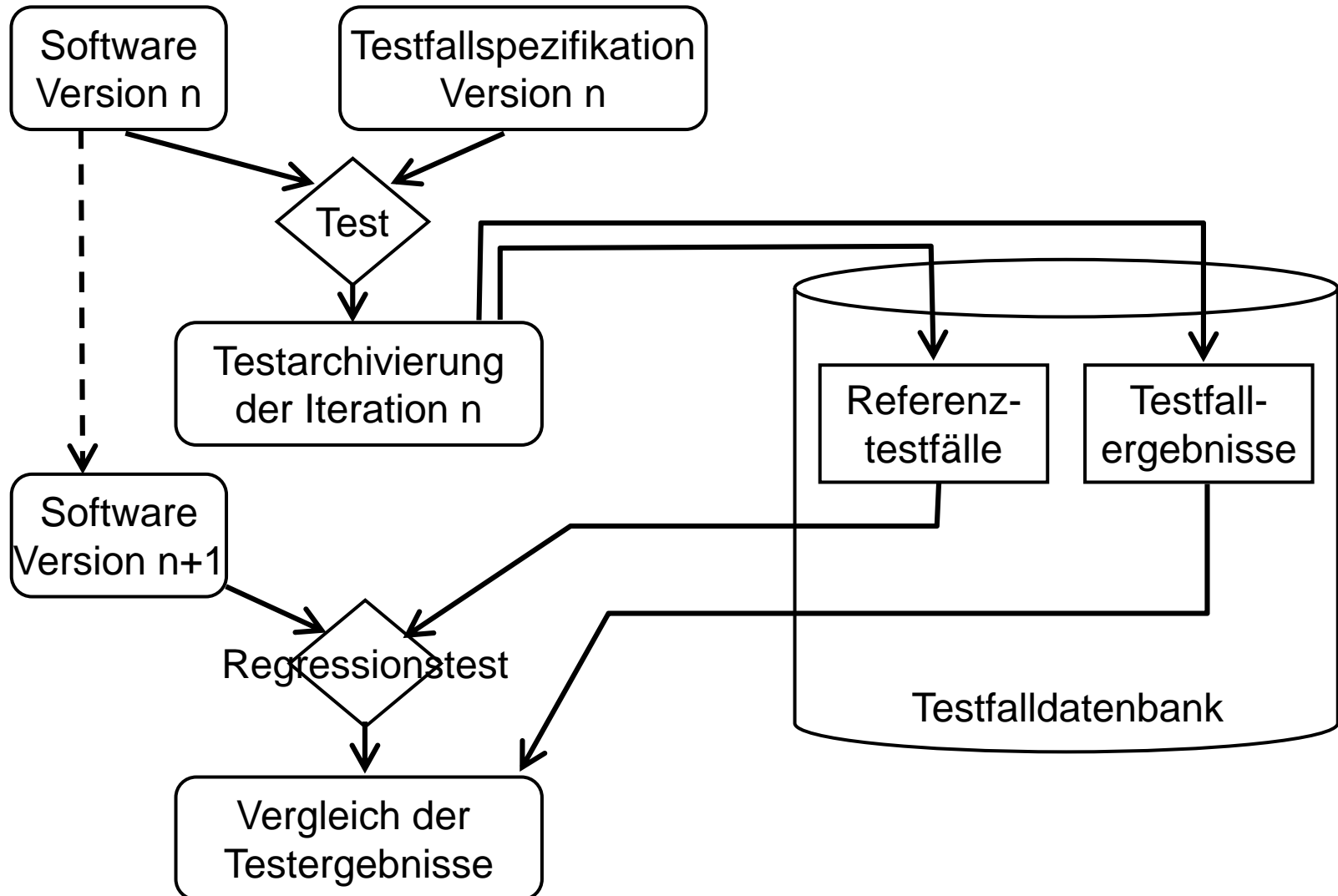
Testen



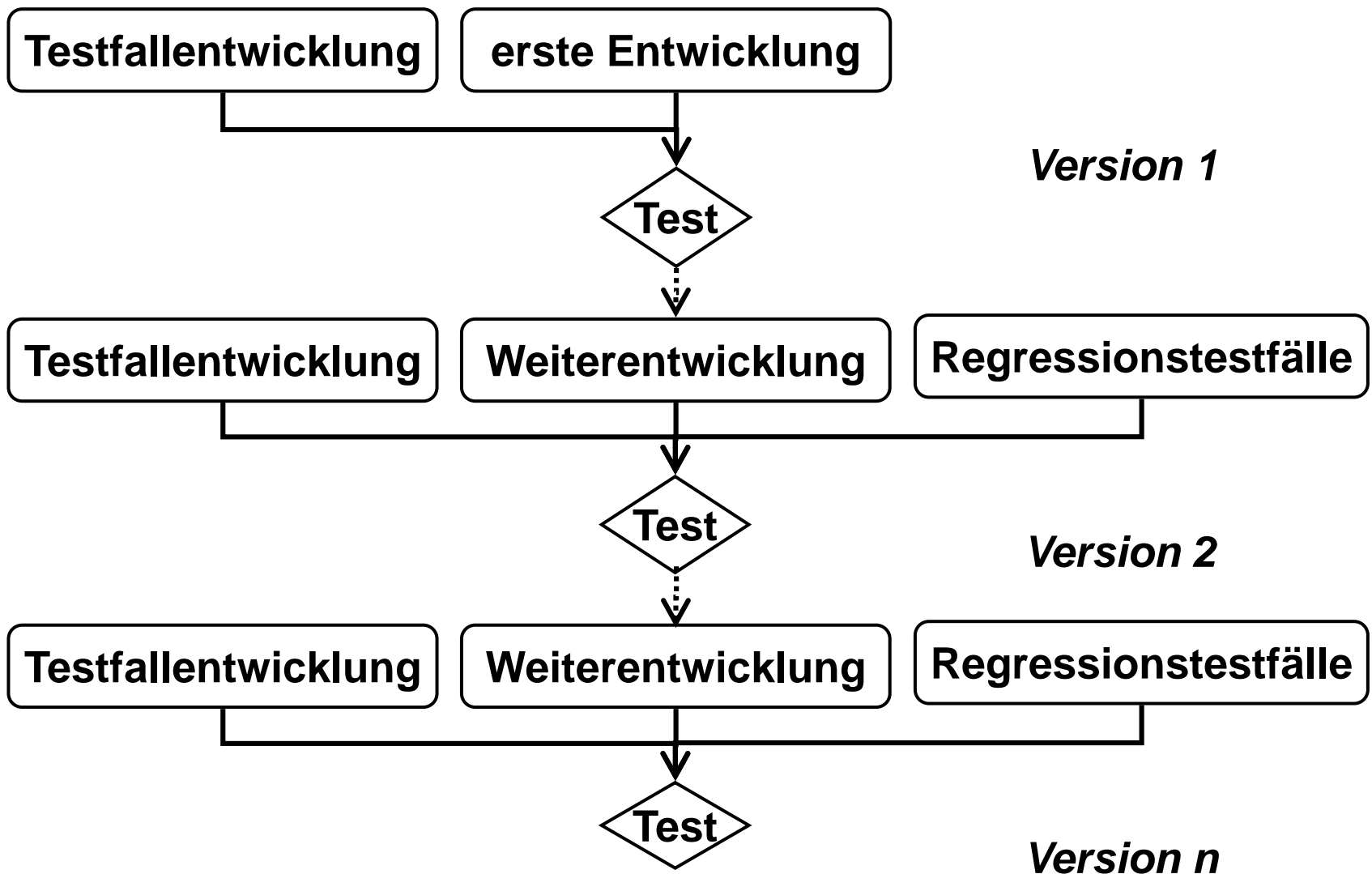
- Änderungen an bereits freigegebenen Modulen sind notwendig
- Gibt es Auswirkungen auf die alten Testergebnisse?
- Wenn ja, welche?
- Wiederholbarkeit der Tests
- Wiederherstellung der Testdaten

- Der Testprozess muss automatisierbar sein
- Testfälle müssen gruppiert werden können, damit man sie wegen der untersuchten Funktionalität (oder auch Testdauer) gezielt einsetzen kann

Prinzip des Regressionstests



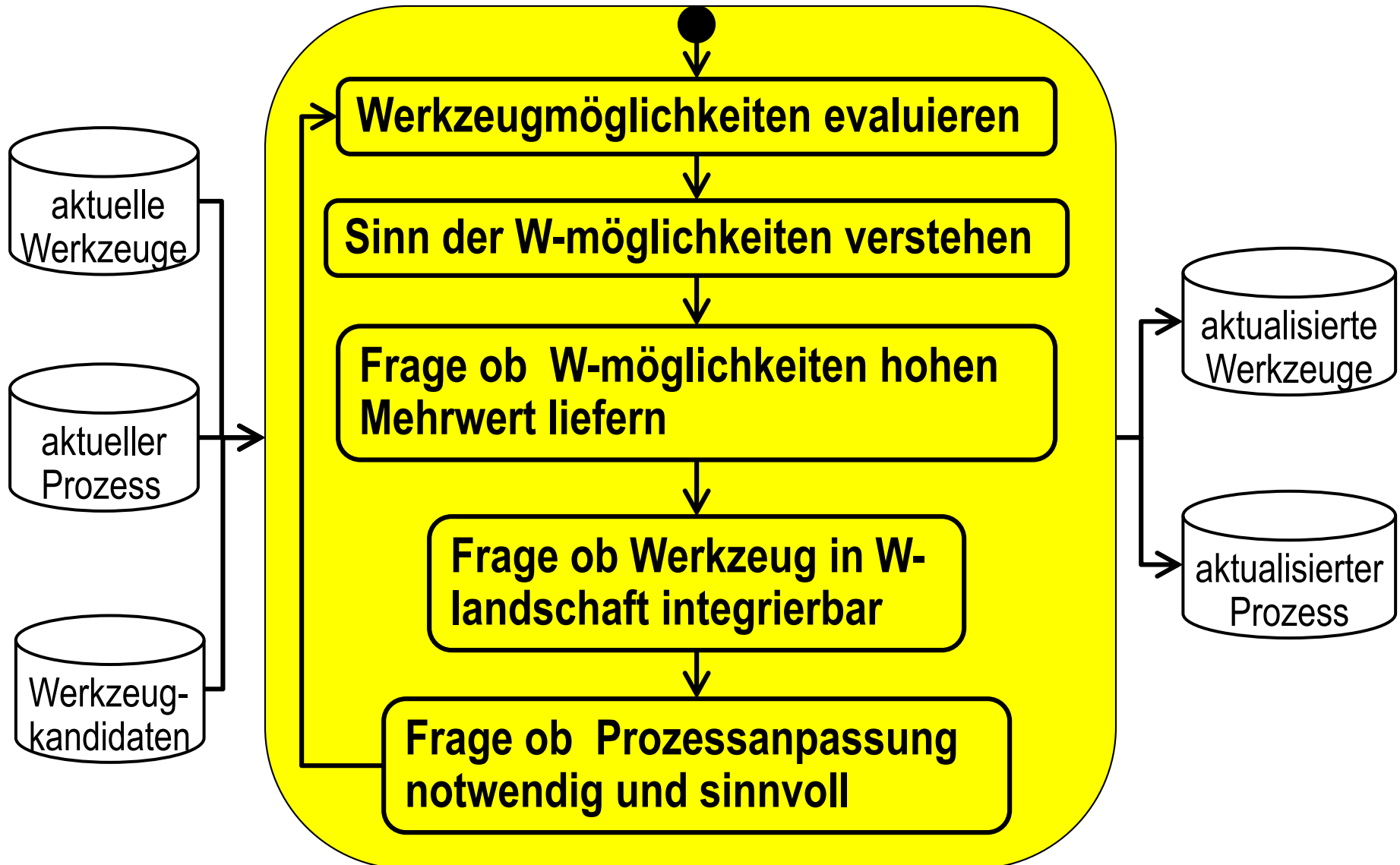
Regressionstests im Entwicklungszyklus



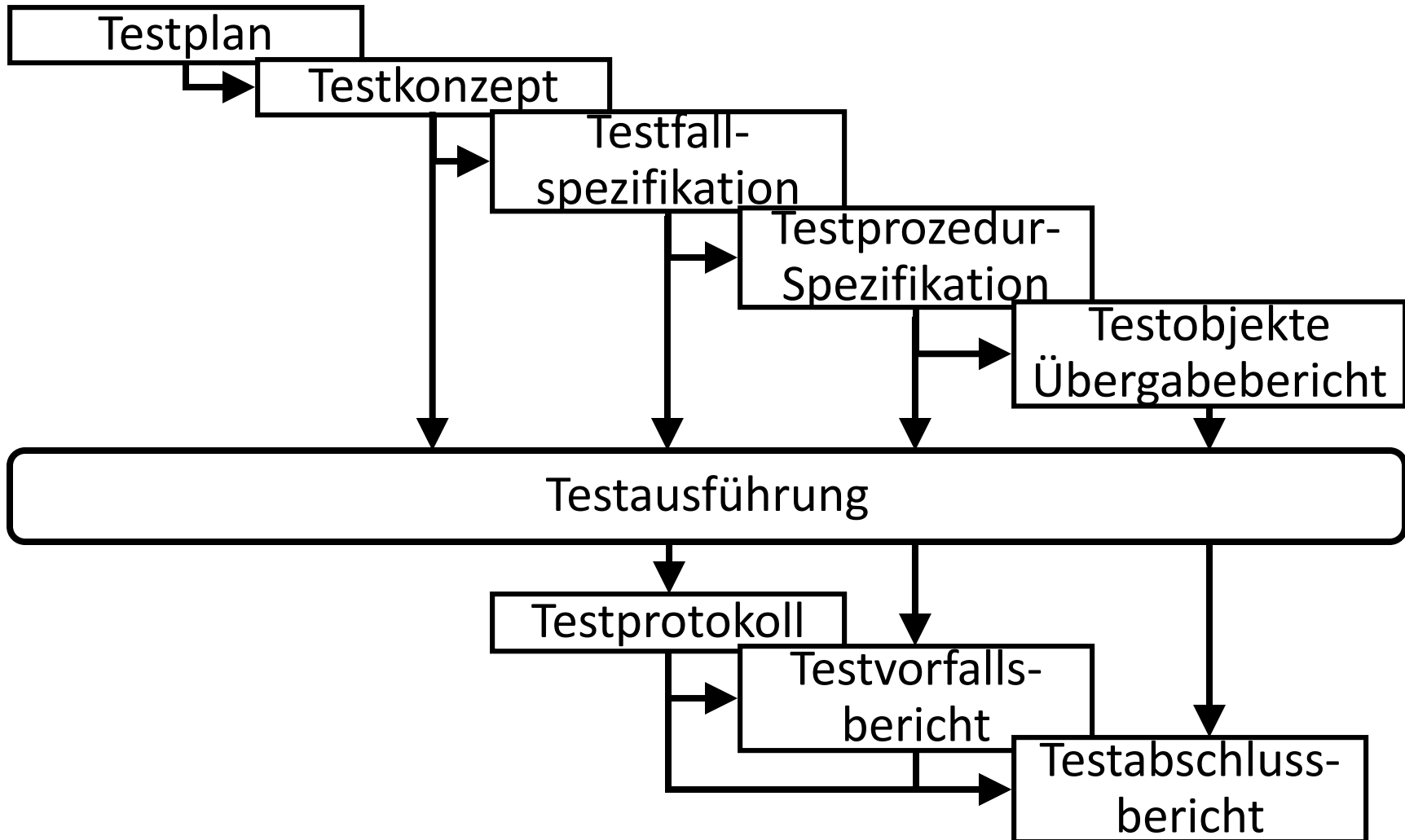
- Der Test ist geteilt in Änderungstest (White-Box) und Regressionstest (Black-Box)
- Änderungstest von Entwickelnden, schreiben die Testfälle fort
- Regressionstest von unabhängiger Testgruppe mit den alten plus neuen Testfällen durchgeführt
- Testgruppe ist für Pflege und Fortschreibung der Systemtestfälle verantwortlich

- Geforderte Performance
 - Durchsatz bzw. Transaktionsrate
 - Antwortzeiten
- Skalierbarkeit
 - Anzahl Endnutzender
 - Datenvolumen
 - Geografische Verteilung
- Zugriffskonflikte konkurrierender Nutzungen
- Entspricht dem Zeitraum nach der Inbetriebnahme
- Simulation von
 - Anzahl Endnutzender,
 - Transaktionsrate , ...
 - Über einen signifikanten Zeitraum (mehrere Stunden)

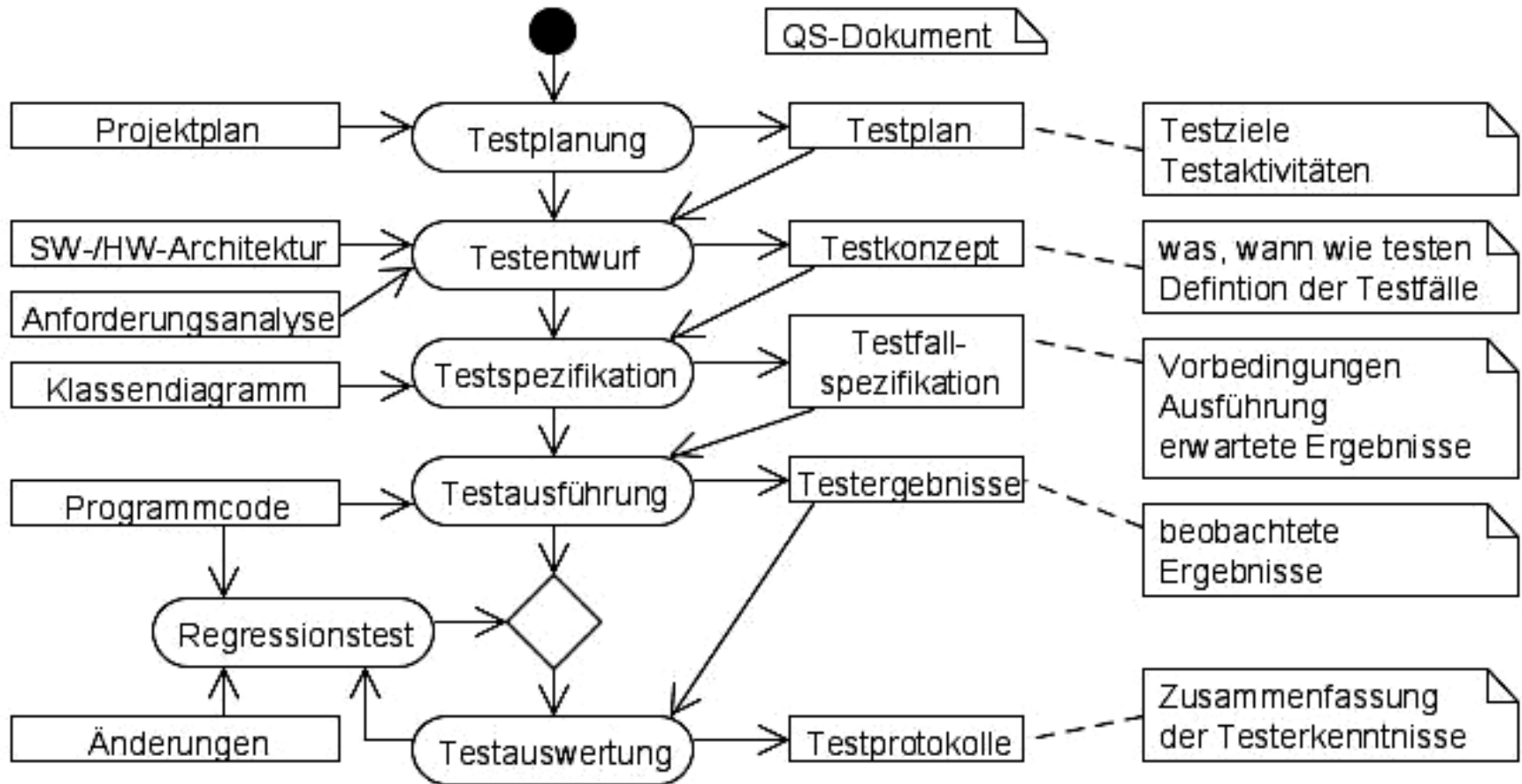
Iterativ inkrementelle Werkzeugauswahl

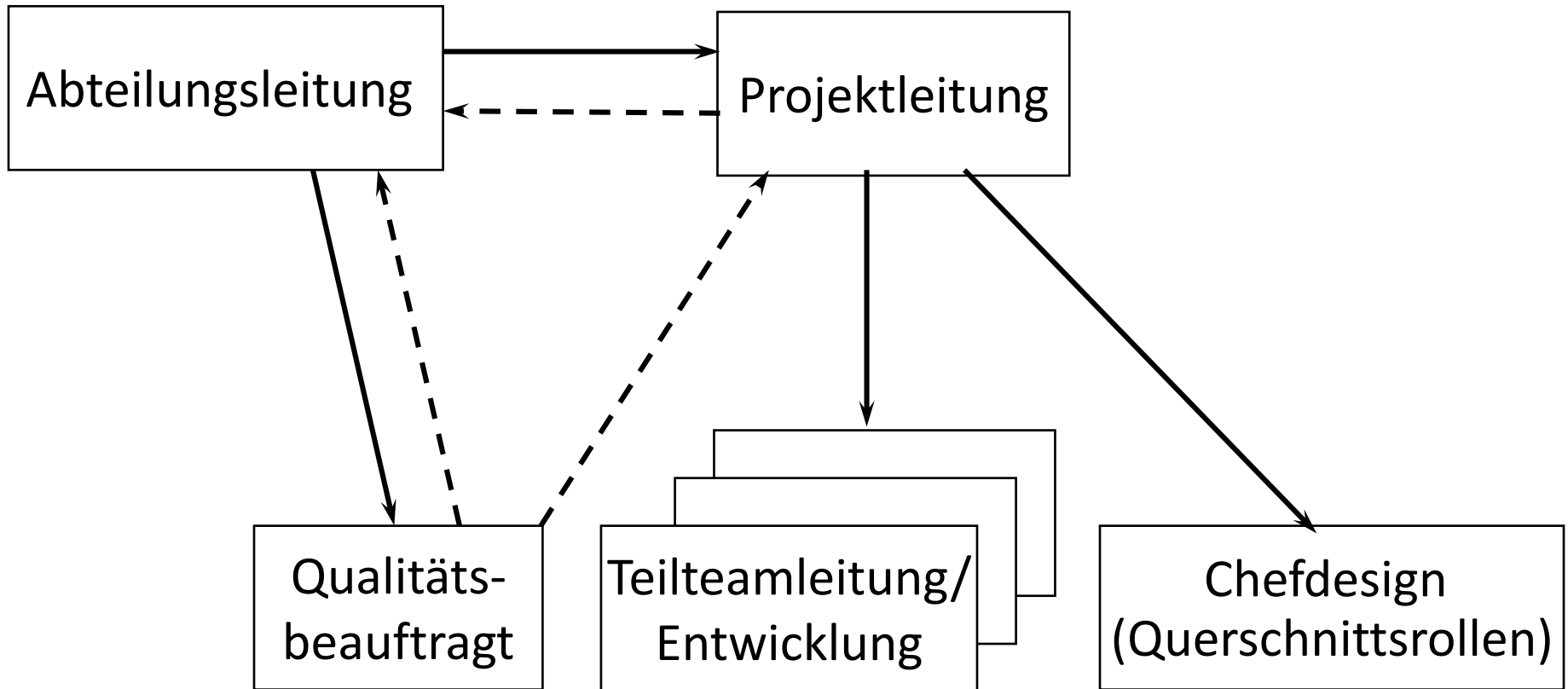


Testverfahren nach ANSI/IEEE-829



Dokumentation der Qualitätssicherung (Tests)





informiert
→
beauftragt
→

Anmerkung: Q-Sicht, Qualitätssicherung
nicht der Projektleitung unterstellt

- bekannt: Standards sind wichtiges Hilfsmittel der QS
- konsequent: Standards für das Testen, genauer die generellen Testprozesse (also nicht Werkzeuge)
- Ergebnis: ISTQB – International Software Testing Qualifications Board,
- definiert Vokabeln
- bietet Schulungen an
- Schulungen auf verschiedenen Leveln und für verschiedene Testarten und Testprozesse (z. B. agiles Testen)
- deutsche Information unter GTB (German Testing Board)
<https://www.german-testing-board.info/>

Zertifizierungen fachlich und nach Schweregrad (Erfahrungsgrad) aufbauend organisiert, z. B.

- Core Foundation: Certified Tester Foundation Level (Fundament)
- Core Advanced: Test Analyst, Test Manager
- Specialist: Agile Test Leadership at Scale
- Agile: Agile Tester
- Expert Level Process: Implementing Test Process Improvement
- Expert Level Test Management: Strategic Test Management