Softwarequalität und -test

6. Vorlesung

„Methodischer Aufbau und Durchführung von Fach- und Abnahmetest“

www.beuth-hochschule.de
Dipl.-Inform. Thomas Ziemer

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Der gesamte Testprozess muss so strukturiert und beschrieben sein, dass jeder am Test Beteiligte weiß, **wann** er **was**, **wie** und **womit** zu testen hat. Das Ziel muss es sein, den Testprozess dergestalt zu optimieren, dass…

…der Anwender mit der Qualität des ihm zur Verfügung gestellten Softwaresystems zufrieden ist…

…und diese Qualitätsstufe in einem angemessenen Verhältnis hinsichtlich der Zeit und Kosten erreicht wird.

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Durch systematisches Erstellen von Testdaten und methodischer Durchführung der Tests werden folgende Projektziele verwirklicht:

Erhöhung der Zuverlässigkeit
durch optimierten Test aller Kombinationsmöglichkeiten des Ablaufes und deren Dateninhalte.

Erhöhung der Testeffizienz
durch eine exakte Testdokumentation; schnellere Fehlerkorrektur und Vermeidung redundanter Testläufe.

Verkürzung der Entwicklungsdauer
durch die Einsatzmöglichkeit (automatischer) Testwerkzeuge.

–

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Erhöhung der Planbarkeit
durch die Erstellung eines Testplans; Möglichkeit der exakten Planung von Terminen, Personaleinsatz und Kosten.

Reduzierung der Wartungszeit und Wartungskosten
durch die (häufige) Wiederverwendung der bereits erstellen Testfälle und Testdaten.

–

Testfälle

Entwurf der Testfälle

Der Entwurf der Testfälle ist eine wichtige Tätigkeit, da von ihm die Qualität des Tests, das Entdecken der Fehler, und damit die des gesamten, lauffähigen Systems abhängt.

Dabei werden einerseits die Testfälle so optimiert, dass ihre Zahl und die dadurch hervorgerufene Zahl der Testläufe möglichst klein bleibt, andererseits ein möglichst hoher Testabdeckungsgrad erreicht wird.

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Entwurf der Testfälle (Fortsetzung)

Testfälle zu ermitteln, zu verwalten und durchzuführen, ist aufwendig. Es kostet Zeit und Geld. Daher möchte man mit möglichst wenigen Testfällen auskommen.

Andererseits will man beim Testen möglichst viele Fehlerquellen prüfen.

Das sieht wie ein Widerspruch aus, und es ist in der Tat nicht einfach, mit wenigen Testfällen viele Fehler zu finden.

Wie soll man vorgehen?

Entwurf der Testfälle (Fortsetzung)

Szenario

Die in der Anforderungsanalyse beschriebenen Geschäftsprozesse werden weiter verfeinert, indem sie durch geeignete „Szenarien“ ergänzt werden. Ein Szenario ist ein typischer Verlauf eines Geschäftsprozesses unter der Annahme bestimmter fester Randbedingungen (Daten, Verhalten etc.).

Ein solches Szenario bildet die ideale Vorlage für einen entsprechenden Testfall. Aus diesem Grund werden heute nicht selten direkt nach der Erarbeitung der Geschäftsprozesse die davon abzuleitenden Testfälle erarbeitet, also bereits in einer frühen Phase des Projekts.

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Entwurf der Testfälle (Fortsetzung)

Black-Box-Tests aus der Spezifikation

Die Idee bei der geschäftsprozess-orientierten Testfallbestimmung ist: jede Anforderung muss getestet werden. Daraus ergibt sich: für jede Anforderung muss ein Testfall erstellt werden. Unter einer Anforderung wird dabei eine atomare, nicht mehr weiter unterteilbare Anforderung gemeint. Wenn die Spezifikation als Fließtext geschrieben wurde, muss dieser zunächst in seine Einzelanforderungen zerlegt werden.

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Entwurf der Testfälle (Fortsetzung)

Black-Box-Tests aus der Spezifikation

Die grauen Spalten rechts zeigen, dass es gelungen ist, zu den fünf Anforderungen R1 bis R5 entsprechende Testfälle T01 bis T09 zu finden. Zu jeder Anforderung gibt es mindestens einen Testfall, also ein Kreuz in jeder Spalte.

Auszug aus der Spezifikation
Funktion „Preis-nach-Altersstufe“
R1: 0 bis 10 J.: „Kind: gratis“
R2: bis 17 J.: „jugendlich: 2€“
R3: bis 110 J.: „volljährig: 3€“
R4: andere Zahlen: „Fehleingabe“ (rot)
R5: andere Zeichen (Text): nicht möglich

ID	*Eingabe*			*R1*	*R2*	*R3*	*R4*	*R5*
T01				X				
T02	14	„jugendl.: 2€“			X			
T03	18	„volljährig: 3€“	Gerade erst volljährig			X		
T04	19	„volljährig: 3€“				X		
T05	77	„volljährig: 3€“	Senioren zahlen voll!			X		
T06	130	„Fehleingabe“	Unmögliches Alter (noch…)				X	
T07	0	„Kind: gratis“	Baby unter einem Jahr	X				
T08	-1	„Fehleingabe“	Sinnlose Eingabe abfangen				X	
T09	„Kind“	(nicht möglich)	Buchstaben nicht akzeptieren					X

Entwurf der Testfälle (Fortsetzung)

Minimalforderung und Effizienzprinzip

Zu einer Liste von Anforderungen können ganz unterschiedliche Testfälle erstellt werden. Ein „guter Test“ jedoch ist eine Sammlung von Testfällen, die sowohl der Spezifikation als auch dem ökonomischen Zwang in Softwareprojekten genügt.

Minimalforderung
Jede spezifizierte Anforderung muss durch mindestens einen Testfall abgedeckt werden. Alle Testfälle zusammen sollen alle Anforderungen abdecken, also die gesamte Spezifikation.

Effizienzforderung
Dieses Prinzip fordert, möglichst wenige Testfälle zu erstellen und dabei möglichst auch mehrere Anforderungen durch einen gemeinsamen Testfall abzudecken.

Teststrategien

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Teststrategien

Mit der Teststrategie wird festgelegt, in welcher Reihenfolge einzelne Module zu einander aufbauenden Subsystemen integriert und getestet werden können. Dabei sind sowohl technische als auch organisatorische Kriterien zu berücksichtigen.

Technische Kriterien wären beispielsweise

- Art und Verwendungszweck der eingesetzten Werkzeuge
- Aufwand für die Erstellung der Testdaten
- Die für das Testen benötigte Maschinenzeit (CPU-Zeit)
- –

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Teststrategien (Fortsetzung)

Organisatorische Kriterien wären

- Entwicklungsstrategie
- Endtermin(e) für das Testen der Module
- Verfügbarkeit notwendiger Personen
- Gewählte Arbeitsaufteilung
- –

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Teststrategien (Fortsetzung)

Top-down-Strategie

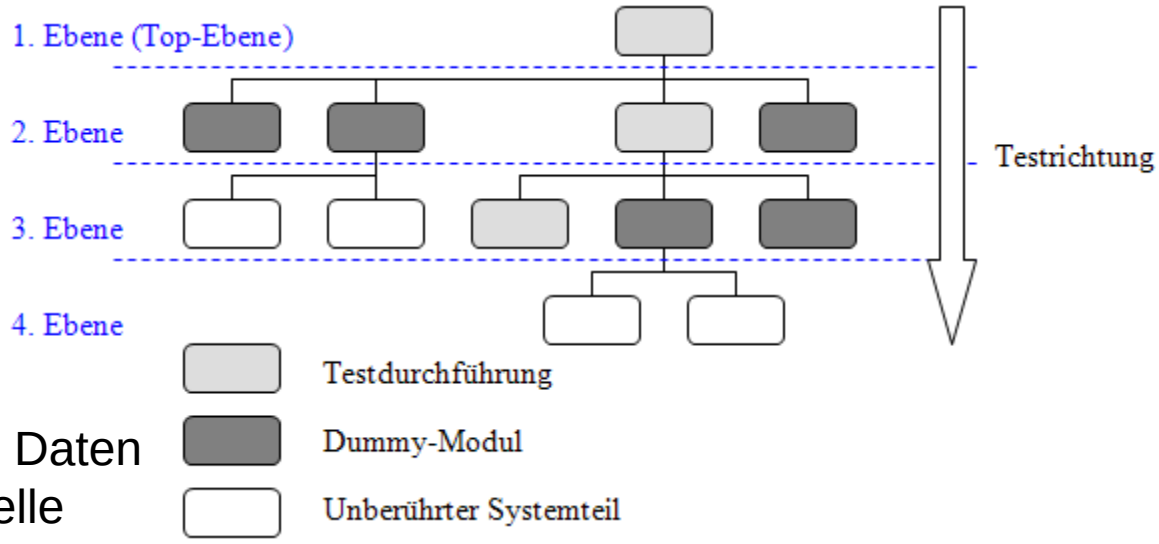
Beim Einsatz der Top-down-Strategie („von oben nach unten“) wird eine Software entlang ihrer hierarchischen Ebenen ausgetestet. Dabei wird zunächst die Steuerung erstellt und ausgetestet. Anschließend werden die jeweils erstellten Module in der Reihenfolge ihrer Fertigstellung getestet. Da zu diesem Zeitpunkt die darunter liegenden (auf**zu**rufenden) Module noch nicht vorhanden sind, werden so genannte Dummy-Module definiert, die danach sukzessive durch die echten Module ersetzt werden.
Die Top-down-Strategie eignet sich für den Test von Programmen, die sehr klar strukturiert sind, und deren Modulsteuerung immer im hierarchisch darüber liegenden Modul liegt.

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Teststrategien (Fortsetzung)

Top-down-Strategie



Als Hilfsmittel werden Dummy-Module benötigt. Ein Dummy-Modul kann sowohl Daten über eine Schnittstelle erhalten (Eingabeparameter) als auch Daten über eine definierte Schnittstelle weitergeben (Ausgabeparameter).

Diese übergebenen Daten werden jedoch nicht in diesem Modul erzeugt. Stattdessen wird eine Möglichkeit vorgesehen, welche das manuelle Eingeben der Daten (beispielsweise über den Bildschirm) gestattet. Außerdem sollte jedes Dummy-Modul über die Möglichkeit der Ausgabe seiner Modulbezeichnung verfügen.

Teststrategien (Fortsetzung)

Bottom-up-Strategie

Beim Einsatz der Bottom-up-Strategie („von unten nach oben“) werden zunächst die Verarbeitungsmodule, dann anschließend die Steuerung erstellt und ausgetestet.

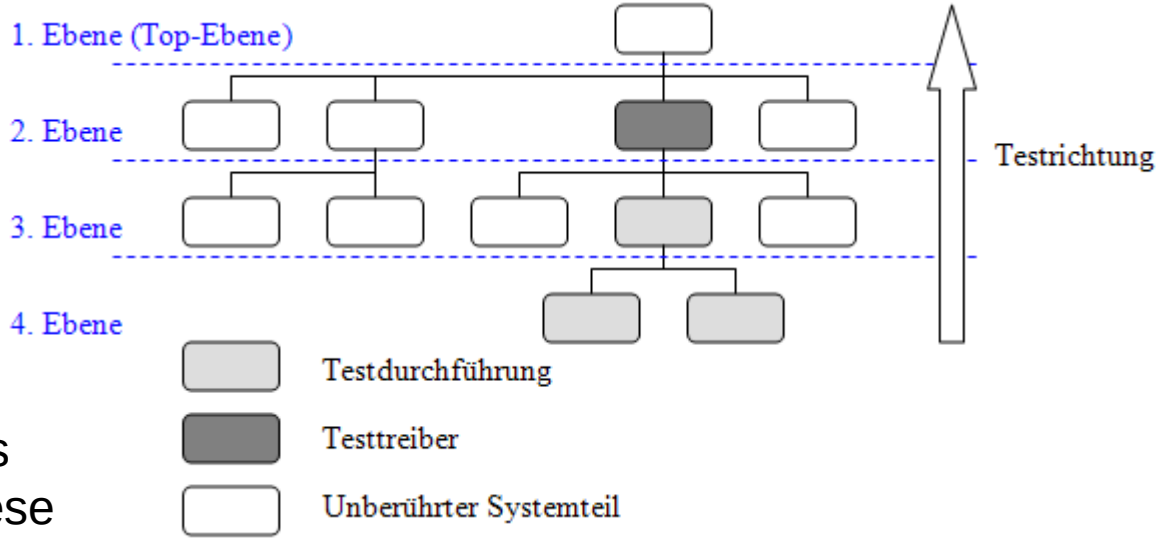
Da zu diesem Zeitpunkt die darüber liegenden (aufrufenden) Module noch nicht vorhanden sind, wird ein so genannter Testtreiber definiert, der die Übergabe und den Empfang der im Modul erzeugten Daten über die Schnittstelle ermöglicht.

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Teststrategien (Fortsetzung)

Bottom-up-Strategie



Als Hilfsmittel wird ein Testtreiber benötigt, der für die Möglichkeit der Eingabe von Daten (Eingabeparameter) über die definierte Schnittstelle an das auf**ge**rufene Modul sorgt. Diese Eingabemöglichkeit sollte dabei über den Bildschirm bestehen. Nach Verarbeitung der Daten im Modul übernimmt der Testtreiber die in der Schnittstelle zur Verfügung gestellten Ausgabedaten (Ausgabeparameter). Nach Durchführung des Tests werden sowohl die Eingabeparameter als auch die Ausgabeparameter dokumentiert.

Teststrategien (Fortsetzung)

Up-down-Strategie

Mit der Up-down-Strategie (auch „Inside-out“ oder „Hardest-first-Strategie“ genannt) wird eine Mischform beschrieben, bei der zunächst die schwierigsten Module erstellt und ausgetestet werden.

Da zu diesem Zeitpunkt unter Umständen weder die darüber liegenden (aufrufenden) noch die darunter liegenden (aufgerufenen) Modul vorhanden sind, werden sowohl Testtreiber als auch Dummy-Module benötigt.

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Teststrategien (Fortsetzung)

Strategiewahl in Abhängigkeit vom Anwendungstyp

Die folgende Tabelle zeigt einige typische Anwendungssysteme und die für sie jeweils günstigsten Teststrategien:

Typ der Anwendung	*Top-down*	*Bottom-up*
Systemsoftware (beispielsweise Betriebssystem)		√
Dateisystem		√
Software in Echtzeit, z.B. Autopilot im Flugzeug		√
Kommerzielle Anwendung, z.B. Lohnabrechnung	√	
Anwendung dominierender Mensch-Maschine-Schnittstelle	√	
Anwendung mit großen interaktiven Anteilen	√	
Spielprogramm	√	

Quelle: G. E. Thailer: Software-Test, heise-Verlag, 2002

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Testmethoden

Bei einem überschaubaren Programm, dessen Programmcode nicht im einzelnen bekannt ist, sondern nur die Programmvorgabe, könnte man Testfälle für sämtliche zulässigen Eingabedaten des Programms entwerfen. Damit würde überprüft werden, ob das Programm gemäß seiner Anforderungen das tut, was es soll.

Da jedoch auch überprüft werden muss, ob das Programm nicht etwas tut, das nicht zulässig ist, müssten auch Testfälle für sämtliche unzulässigen Eingabedaten entworfen werden.

Man kommt auf eine unendliche Anzahl von Testfällen: der erschöpfende Datentest ist demnach nicht möglich.

Äquivalenzklasse

Testmethoden (Fortsetzung)

Äquivalenzklasse

Die Zahl der Testfälle muss also dahingehend verringert werden, dass mit einem Testfall möglichst gleich eine ganze Klasse von Fehlern aufgedeckt wird (beispielsweise generell falsche Verarbeitung der Werte einer Eingabegröße vom Typ „String“).

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Testmethoden (Fortsetzung)

Äquivalenzklasse

Mit der Äquivalenzklassenmethode können Testfälle gefunden werden, die gut zu einer gegebenen Spezifikation passen. Sie ist damit eine spezielle Form des Black-Box-Tests. Das Prinzip der Äquivalenzklassenmethode basiert auf der Beobachtung, dass oft viele ähnliche Eingaben den gleichen Fehler auslösen – während viele andere Eingaben ihn alle *nicht* auslösen. Nun wird versucht, die Eingaben in „Schubladen“ (Äquivalenzklassen) einzuteilen:

Unter einem Jahr = { 0 }
Kinder Ab Eins = { 1, 2, 3, 4, 5, 6, 7, 8, 9,10 }
Jugendliche = { 11, 12, 13, 14, 15, 16, 17 }

Testmethoden (Fortsetzung)

Äquivalenzklasse

Erwachsene = { 18, 19, …, 110 } wären alle Erwachsenen, die Klasse teilen wir jedoch noch einmal auf, weil wir vermuten, mit den dreistelligen Altersangaben könnte es Probleme geben:

Zweistellige Erwachsene = { 18, 19, …, 99 }
Dreistellige Erwachsene = { 100, 101, …, 110 }

Testmethoden (Fortsetzung)

Äquivalenzklasse

Unter einer Äquivalenzklasse wird also diejenige Menge von Eingabewerten verstanden, die einen identischen Ausgabewert hervorrufen, und somit ein gleiches Programmverhalten aufzeigen sollten. Für alle Werte einer Äquivalenzklasse gilt, dass bei der Testausführung mit *einem* Repräsentanten dieser Klasse die gleiche Wirkung (in Bezug auf die entdeckte Fehlerart und -anzahl) auftritt, als wenn mit einem beliebigen anderen Wert dieser Klasse getestet wird.

Liegen die Werte innerhalb des spezifizierten Definitionsbereichs der Ein- oder Ausgaben, so handelt es sich um eine gültige Äquivalenzklasse, ansonsten um eine ungültige.

Grenzwertanalyse

Testmethoden (Fortsetzung)

Grenzwertanalyse

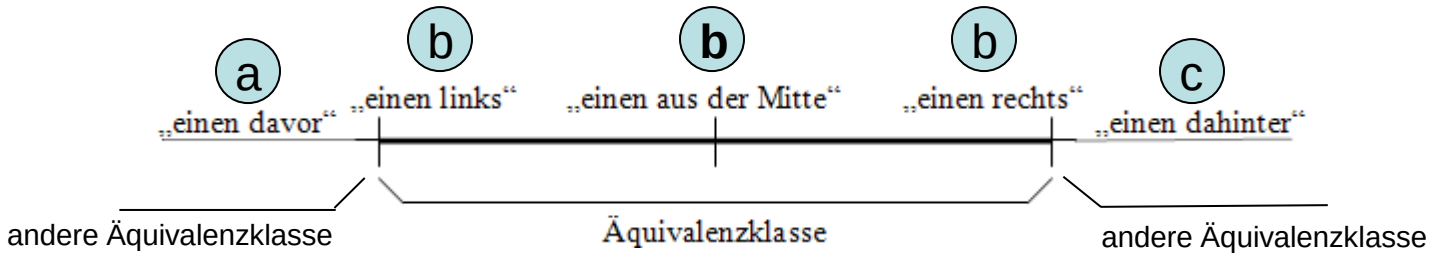
Aus der Bildung der gültigen und ungültigen Äquivalenzklassen ergeben sich automatisch Grenzwerte als die „Ränder“ der Äquivalenzklassen. Jeder Rand einer Äquivalenzklasse (sowohl des Eingaberaumes als auch des Ausgaberaumes) muss in einem Testfall auftreten. Auch alle Ausgabe-Äquivalenzklassen müssen durch entsprechende repräsentative Eingabewerte berücksichtigt werden.

Der Hintergrund dieser Methode besteht ganz einfach in der Erfahrung, dass bei Ausnahme- und Grenzwerten die Wahrscheinlichkeit eines Fehlers sehr viel größer ist als bei den so genannten Normalwerten.

Testmethoden (Fortsetzung)

Grenzwertanalyse

Als Faustregel zur Bildung der Grenzwerte gilt im allgemeinen: *„einen links, einen rechts, einen davor, einen dahinter und einen aus der Mitte“*. Das sind insgesamt fünf Testfälle, durch die eine Äquivalenzklasse abgedeckt wird:



Testmethoden (Fortsetzung)

Spezifikationsabdeckung optimieren

Es sollen alle Anforderungen einer Spezifikation abgedeckt werden. Durch die Äquivalenzklassenmethode wird noch einmal nach Eingabeparametern für jede Operation differenziert. Trotzdem soll die Zahl der Testfälle noch weiter beschränkt werden.

Dazu werden die Minimalforderung und das Effizienzprinzip verwendet: jede Äquivalenzklasse für jeden Parameter muss mindestens einmal angesprochen werden, also in mindestens einem Testfall auftauchen.

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Testmethoden (Fortsetzung)

Spezifikationsabdeckung optimieren

Durch günstige Kombination der Äquivalenzklassen aller Parameter sollten nicht viel mehr Testfälle entstehen, als man schon allein für den Parameter mit den meisten Äquivalenzklassen braucht – nämlich so viele, wie er Äquivalenzklassen hat.

Umsatz (€)	m	w	m	w	m	w	m	w	m	w
> 7.000									T5	
3.000 – 6.900			T2					T4		
100 – 2.999	T1									
< 100						T3				
	10 – 13		*14 – 17*		*18 – 29*		*30 – 60*		*über 60*	

Alter

Modultest

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Modultest

Mit Modultests (*Unit Tests*) kann schon während der Entwicklung in der Implementierungsphase immer wieder überprüft werden, ob ein Programm wie vorgesehen arbeitet. Modultests zeichnet aus:

- Saubere Definition der (zu testenden) Schnittstellen
  Modultests fördern einen modularen Aufbau der Anwendung.

- Finden von Fehlern in Modulen, die schon längst funktionieren sollten
  Minimierung des Debuggens durch Regressionstests

- Fehlersuche automatisieren und wesentlich beschleunigen
  Für jedes relevante Stück Code zunächst eine eigene Testroutine schreiben; Tests lassen sich danach beliebig oft wiederholen

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Modultest (Fortsetzung)

- Modultests untersuchen stets einen möglichst kleinen, für sich allein funktionierenden Code-Ausschnitt, etwa eine einzelne Methode oder Klasse.

- Mit ihnen können bereits die Entwickler Detailfehler aufdecken und preiswert beseitigen, bevor die Testabteilung oder gar der Kunde darauf stößt.

- Modultests werden schon länger für aufwendige Qualitätssicherung besonders sicherheitskritischer Software eingesetzt. Neu ist allerdings die Idee, dass Programmierer solche Tests bereits während der Implementierung ihres eigenen Codes einsetzen. Diese Art der Modultests wird deshalb auch Entwicklertest genannt. Bei Verwendung agiler Vorgehensmodelle implementiert der Entwickler die Tests schon vor den eigentlichen Anwendungsmethoden (*Test First*, *Test Driven Design*).

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Modultest (Fortsetzung)

- Das Schreiben von Modultests setzt ein hohes Verständnis guten Softwaredesigns voraus (hierzu später mehr).

–

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Dummy-Modul

Werden verschiedene Module ein und desselben Programms parallel entwickelt, steht keines dieser Module den anderen zu Testzwecken zur Verfügung. Getestet wird stattdessen gegen Dummy-Module, welche die gleichen Schnittstellen zur Verfügung stellen wie die Originale, und eine manuelle Prüfung der Funktionsfähigkeit der zu testenden Module erlauben (beispielsweise mittels Protokollausgaben).

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Black-Box- und Glass-Box-Test

Black-Box-Tests (auch funktionale Tests genannt) verwenden den Ansatz, nur aufgrund der offengelegten Schnittstellen eines Testmoduls dieses mit Eingaben zu versorgen und das Ergebnis auszuwerten. Die Syntax und Semantik des der implementierten Methode zugrundeliegenden Algorithmus' wird nicht interpretiert.

Wer nicht den *Test First*-Ansatz verfolgt, und stattdessen die Tests für bereits existierende Operationen implementiert, kann Testfälle jedoch auch durch Analyse des Quellcodes identifizieren. Diese Vorgehensweise nennt man Glass-Box-Test.

–

Modultest mit JUnit

Beispiel *swXercise*

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Teststruktur mit JUnit

Modultests werden in Java häufig mit *JUnit* durchgeführt. JUnit ist ein Framework zum Testen von Java-Anwendungen auf Methodenebene. JUnit wird seit vielen Jahren entwickelt – maßgeblich unter der Leitung von *Erich Gamma* (einer der *Gang of Four*).

Die grundsätzliche Struktur eines übersichtlichen JUnit-Tests wird im folgenden als konkretes Code-Beispiel anhand eines Patterns vorgestellt.

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Teststruktur mit JUnit – Beispiel *swXercise*

https://git.ziemers.de/edu/swXercise

Ein Java-Beispielprojekt für die Lehrveranstaltungen "Software Engineering 2" (SE2), "Softwarequalität und Test" (SwQT) sowie "Webentwicklung" (an der HTW).

UserViewControllerTest

Controller-Test

Teststruktur mit JUnit

Testklassen tragen immer den Namen der zu testenden Klasse, aber mit einem sprechenden Suffix. Sie befinden sich in einem Java-Package, das typischerweise den gleichen Pfad hat wie das der Produktklassen, aber mit anderer Wurzel beginnt, beispielsweise:

test.java.net.ziemers.swxercise.ui.**UserViewController****Test**

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Teststruktur mit JUnit – Der Test des Controllers

test.java.net.ziemers.swxercise.ui.**UserViewControllerTest**

```
@RunWith(MockitoJUnitRunner.class)
public class UserViewControllerTest {
    ...
    // hier kommt der Inhalt der Testklasse
}
```

Der Test läuft mit Java *SE*, nicht mit Java *JEE*. Mockito stellt uns eine ähnliche Umgebung mit *Dependency Injection* und Objekt-Verwaltung zur Verfügung.

Getestet wird hier ausschließlich der *Controller*, nicht der *UserService*.

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Teststruktur mit JUnit – Vertrauen schaffen

```
@Test
public void testJUnitFrameworkSucceeds() {
   assertTrue(true);
}
```

Sicherstellen, dass *JUnit* wirklich funktioniert.

```
/**
 * Hier sieht man eine erwartete Exception
 */
@Test(expected = AssertionError.class)
public void testJUnitFrameworkThrowsException() {
    assertTrue(false);
}
```

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Teststruktur mit JUnit – Das GWT-Pattern

```
@Test
public void testCreateUserSucceeds() {
G   given()      .userDto()
                 .method1()
                 .method2();
    bzw. doing()
W   when()       .createUser(true);
T   then()       .assertCreateUserSucceeded();
}
```

Die Aneinanderreihung von Methodenaufrufen nennt man *Method Chaining*.

Vor- und auch Nachteil:
Technische Aspekte sind in dieser Testmethode nicht zu erkennen.

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Teststruktur mit JUnit – Das GWT-Pattern

```
private UserViewControllerTest given() {
    return this;
}

private UserViewControllerTest when() {
    return this;
}

private UserViewControllerTest then() {
    return this;
}
```

Was tun *given()*, *when()* und *then()*?

Bzw. *doing()* statt *when()*, da Mockito bereits eine Methode gleichen Namens besitzt, wie wir in ein paar Folien sehen werden.

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Teststruktur mit JUnit – Testvorbereitung

```
public class UserViewControllerTest {

    private UserDto userDto;

    …

    private UserViewControllerTest userDto() {
        this.userDto = new UserDtoTestDataBuilder().build();
        return this;
    }
```

Hier werden die Voraussetzungen für den Test geschaffen: Wir benötigen für ihn ein Objekt der Klasse *UserDto*.

Die Implementierung der sehr hilfreichen Methode *UserDtoTestDataBuilder()* gucken Sie bitte selber an.

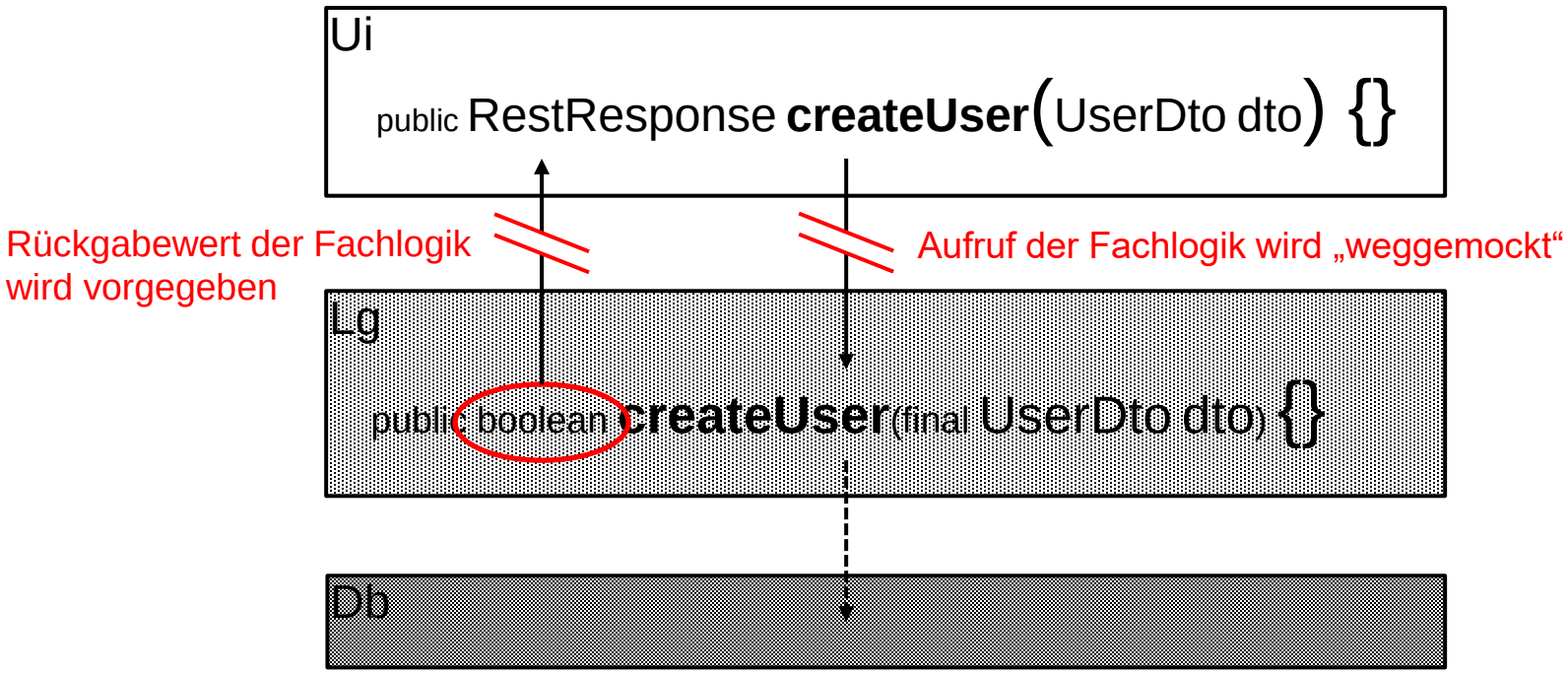
G

Method Chaining auch hier.

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Teststruktur mit JUnit – „Wegmocken“ der Fachlogik



Softwarequalität und -test

Methodischer Aufbau und Durchf…

Teststruktur mit JUnit – Testdurchführung

```
private UserDto userDto;

@Mock
private UserService userService;

@InjectMocks
private UserViewController underTest;
private  RestResponse actual;

private void createUser(final boolean result) {
   /*
    * The when().thenReturn() method chain is used to specify
    * a return value for a method call with pre-defined parameters.
    */
   when(userService.createUser(userDto)).thenReturn(result);
   actual = underTest.createUser(userDto);
}
```

@Mock und *@InjectMocks* kommen von Mockito und realisieren die Objektverwaltung.

Fachlogik Controller

```
@POST
@Path("v1/user")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
@RolesAllowed(RightState.Constants.ADMIN)
public RestResponse createUser(UserDto dto) {
   if (userService.createUser(dto)) {
      return new RestResponse();
   }
   return new RestResponse(
                  ResponseState.ALREADY_EXISTING);
}
```

main.java.net.ziemers.swxercise.ui.UserViewController

Hier wird der eigentliche Test durchgeführt.

Softwarequalität und -test

Methodischer Aufbau und Durchf…

Teststruktur mit JUnit – Testnachbereitung

```
public class UserViewControllerTest {

    private RestResponse actual;
    …

    private void assertCreateUserSucceeded() {
        final RestResponse expected = new RestResponse();
        assertEquals(expected, actual);
    }
```

```
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    RestResponse that = (RestResponse) o;
    return responseState == that.responseState;
}
```

main.java.net.ziemers.swxercise.ui.utils.RestResponse

Die Methode *equals()* prüft nur den *responseState*.

Hier werden erwartetes und reales Testergebnis verglichen.

assertEquals() liefert eine sprechende Fehlermeldung im *Java Stack Trace*.

UserServiceTest

Fachlogik-Test

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Teststruktur mit JUnit – Der Test der Fachlogik

test.java.net.ziemers.swxercise.lg.user.service.**UserServiceTest**

```
@RunWith(CdiRunner.class)
public class UserServiceTest extends JpaTestUtils {
  ...
  // hier kommt der Inhalt der Testklasse
}
```

Die Implementierung der sehr hilfreichen Klasse *JpaTestUtils* gucken Sie bitte selber an.

Der Test läuft mit Java *SE*, nicht mit Java *JEE*.
Der *CDIRunner* stellt uns eine ähnliche Umgebung mit *Dependency Injection* und Objekt-Verwaltung zur Verfügung.

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Teststruktur mit Junit – Das GWT-Pattern

test.java.net.ziemers.swxercise.lg.user.service.**UserServiceTest**

```
public class UserServiceTest extends JpaTestUtils {
    private static String USERNAME_TEST = "username_test";
    …

    @Test
    public void testCreateUserSucceeds() {
      given()    .userDto(USERNAME_TEST);
      when()     .createUser();
      then()     .assertCreateUserSucceeded();
    }
```

Der Benutzer *username_test* sollte natürlich noch nicht in der Datenbank persistiert sein.

Teststruktur mit JUnit – Testvorbereitung

```
public class UserServiceTest extends JpaTestUtils {

    private UserDto userDto;

    ...

    private UserServiceTest userDto(final String username) {
        userDto = new UserDtoTestDataBuilder() .withUsername(username) .build();
        return this;
    }
```

Die Implementierung der sehr hilfreichen Methode *UserDtoTestDataBuilder()* gucken Sie bitte selber an.

Hier werden die Voraussetzungen für den Test geschaffen: Wir benötigen für ihn wiederum ein Objekt der Klasse *UserDto*.

G

Softwarequalität und -test

Methodischer Aufbau und Durchf…

Teststruktur mit JUnit – Testdurchführung

```
public class UserServiceTest extends JpaTestUtils {
    private boolean actual;
    private UserDto userDto;
    @Inject
    private UserService underTest;
    …
    private UserServiceTest createUser() {
      txBegin();
      actual = underTest.createUser(userDto);
      txCommit();
      return this;
    }
```

@Inject kommt vom CDIRunner und realisiert die Objektverwaltung (dependency injection).

txBegin() und *txCommit()* kommen aus der Klasse *JpaTestUtils*.

```
public boolean createUser(final UserDto dto) {
    final UserDtoToEntityContext ctx = ctxService
                                    .createContext(dto);
    if (ctx.user != null) {
      mapper.map(ctx);
      return dao.save(ctx.user) != null;
    }
    return false;
}
```

main.java.net.ziemers.swxercise.lg.user.service.UserService

Softwarequalität und -test

Methodischer Aufbau und Durchf…

Teststruktur mit JUnit – Testdurchführung

```
public class UserServiceTest extends JpaTestUtils {

  …

  private UserServiceTest createUser() {
    txBegin();
    actual = underTest.createUser(userDto);
    txCommit();
    dbInitialized = false;
    return this;
  }
```

Mit *@Before* wird die Methode vor jedem einzelnen Test ausgeführt.

```
@Before
public void setUp() throws Exception {
  if (!dbInitialized) {
    cleanDb();
    initDbWith("UserServiceTestData.xml");
    dbInitialized = true;
  }
}
```

Nach jedem Test, der die Datenbank modifiziert, ist diese ggf. in ihren Urzustand zurückzuführen!

Softwarequalität und -test

Methodischer Aufbau und Durchf…

Teststruktur mit JUnit – Testnachbereitung

```
public class UserServiceTest extends JpaTestUtils {

    private static String USERNAME_TEST = "username_test";

    @Inject
    private UserDao userDao;

    …

    private void assertCreateUserSucceeded() {
        final User user = userDao.findByUsername(USERNAME_TEST);
        assertNotNull(user);
    }
```

```
public User findByUsername(final String username) {
    User user = null;
    try {
        user = (User) entityManager
                .createNamedQuery("User.findByUsername")
                .setParameter("username", username)
                .getSingleResult();
    }
    catch (Exception e) { /* nix */ }
    return user;
}
```

main.java.net.ziemers.swxercise.db.dao.user.UserDao

Wie bekommen wir heraus, ob der Test funktioniert hat?
Indem wir schauen, ob der Benutzer sich nun tatsächlich in der Datenbank befindet.

Eingabetest

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Testmethoden (Fortsetzung)

Eingabetest

Bei dieser Testmethode geht es darum,

- den Definitionsbereich von Eingabedaten,
- die richtige Syntax der Eingabedaten,
- die Fehlerhinweise der Anwendung,
- das Handling der Masken (Tab-, Return- und Esc-Taste),
- interne Plausibilitätsprüfungen der Anwendung,
- die Hilfe-Funktion und weitere F-Tasten sowie
- die Vorbelegungen in einer Maske

umfangreich zu testen.

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Testmethoden (Fortsetzung)

Nach dem heutigen Stand der Technik sind folgende Mindestanforderungen bzgl. Robustheit und Benutzerfreundlichkeit an ein Dialogsystem zu stellen:

- Keine Benutzereingabe darf das Anwendungssystem zum Absturz bringen.
- Bei ungültigen Eingaben sind aussagekräftige Fehlermeldungen zu liefern.
- Zu allen Feldern einer Maske soll eine Hilfe-Funktion zur Verfügung stehen.
- Durch Plausibilitätsprüfungen sind widersprüchliche Eingaben abzuweisen.
- Bis zur Freigabe der Maske müssen alle Eingaben korrigiert werden können.
- Maskenbezeichnungen und Feldbezeichnungen müssen konsistent sein.
- Masken müssen korrekt initialisiert (vorbelegt) sein.

–

Testmethoden (Fortsetzung)

Statistische und intuitive Testdatenauswahl

Der Sinn dieser Methode ist umstritten. Denn jeder Test muss nachvollziehbar und wiederholbar sein. Dies ist beim statistischen Test nicht immer gewährleistet. Statistische Tests geben Wahrscheinlichkeitsaussagen wieder, beispielsweise über erwartete Zuverlässigkeit, Restfehlerrate und Risiko des Testobjekts.

Testmethoden (Fortsetzung)

Das Ziel solcher Tests ist:

Die Wahrscheinlichkeit eines Software-Ausfalls soll unter eine gewisse Grenze mit einem Vertrauensniveau (Zuverlässigkeit) von *P %* liegen.

Die Zahl der Fehler, die noch vorliegen (Restfehlerrate), wird unterhalb eines bestimmten Wertes erwartet.

Die wegen eines Software-Fehlers erwarteten Kosten (Risiko) sollen unterhalb eines gegebenen Wertes mit einem Vertrauensniveau von *P %* liegen.

Testmethoden (Fortsetzung)

Test aufgrund von Erfahrungen

Dies ist ein auf die Wahrnehmungsfähigkeit des Menschen bzw. auf heuristisches Vorgehen ausgerichteter Ansatz. Es handelt sich daher nicht um eine Methode im strengen Sinn, sondern um ein in der Praxis sehr wirkungsvolles Vorgehen.

Es gibt Personen, meist sehr erfahrene Programmierer oder Systemanalytiker, die durch bloße Intuition Fehler aufspüren können. Meist trifft dies auf Fehlerkategorien zu, die immer wieder auftreten (falsche Schleifeninitialisierung, Rechnen mit Werten unterschiedlichen Typs etc.).

Dieser Ansatz eignet sich sehr gut, systematisch mit den vorgestellten Methoden erzeugte Testfälle qualitativ noch zu verbessern.

Entscheidungstabellen

Entscheidungstabellen

Während die Grenzwert- und Äquivalenzklassenanalyse vor allem Definitionsbereiche und Plausibilitäten überprüfen, lassen sich mit Hilfe von Entscheidungstabellen komplexe Testfälle zur Überprüfung der logischen Vollständigkeit aufbauen.

Entscheidungstabellen (Fortsetzung)

Grundsätzlich dient die Entscheidungstabellen-Technik (ET-Technik) dazu, fachliche Zusammenhänge übersichtlich und eindeutig darzustellen, bei denen Arbeitsabläufe von Bedingungskonstellationen abhängen. Sie helfen bei:

Dokumentation: Zur Dokumentation von Lösungswegen, Vorgehensweisen, fachlichen Sachverhalten usw. eignen sich Entscheidungstabellen sehr gut, da sie in knapper Form alle wesentlichen Gesichtspunkte klar und vollständig darstellen

Programmvorgabe: Werden Entscheidungstabellen in Fach- oder DV-Konzepten verwendet, so kann der Entwickler sie mit wenig Aufwand in sein Programm übernehmen. Verwendet man gar ET-Generatoren, lassen sich Entscheidungstabellen automatisch in eine Programmiersprache transformieren.

Entscheidungstabellen (Fortsetzung)

Test: Wenn Arbeitsabläufe von vielen Bedingungskonstellationen abhängen, tritt beim Testen häufig das Problem auf, dennoch alle möglichen Konstellationen mit Testfällen abdecken zu müssen.

Auf Basis aller möglichen Entscheidungen repräsentieren die Regeln der Entscheidungstabelle immer bereits alle logisch notwendigen Testfälle.

Entscheidungstabellen (Fortsetzung)

Voraussetzung für den Einsatz von Entscheidungstabellen ist, dass sie

vollständig (kein Fall darf fehlen),

redundanzfrei (Regeln so weit wie möglich konsolidiert) und

widerspruchsfrei (bei sich entsprechenden Bedingungen dürfen keine unterschiedliche Aktionen auftreten)

sind.

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Entscheidungstabellen (Fortsetzung)

Eine Regel legt fest, unter welchen Voraussetzungen bestimmte Maß-nahmen zu ergrei-fen sind. Die Ge-samtheit der An-zeiger einer Spalte bildet eine Regel. Die Bedingungen einer Regel sind und-verknüpft.

Bedingungsteil	**Bedingungsanzeiger**	
	Regel 1	Regel 2
Bedingung 1	J/N	...
...	...	...
Aktionsteil	**Aktionsanzeiger**	
Aktion 2	X bzw. ohne Angabe	..
...	...	...

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Entscheidungstabellen (Fortsetzung)

Ob eine Bedingung zutrifft oder nicht, wird durch Bedingungsanzeiger (*J* oder *N*) angezeigt.

Bedingungsteil	**Bedingungsanzeiger**	
	Regel 1	Regel 2
Bedingung 1	J/N	J/N
...	J/N	J/N
Aktionsteil	**Aktionsanzeiger**	
Aktion 2	X bzw. ohne Angabe	...
...	...	...

Entscheidungstabellen (Fortsetzung)

Werden als Bedingungsanzeiger nur *J* und *N* verwendet, spricht man von einer begrenzten Entscheidungstabelle. Werden auch andere Bedingungsanzeiger (*1*, *2*, *3*, *verheiratet*, *geschieden* etc.) verwendet, so handelt es sich um eine erweiterte Entscheidungstabelle.

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Entscheidungstabellen (Fortsetzung)

Der Aktionsanzeiger *X* kennzeichnet in einer Regel, ob eine Aktion ausgeführt wird oder nicht. Findet keine Aktion statt, wird dies mit einem leeren Feld gekennzeichnet.

Bedingungsteil	**Bedingungsanzeiger**	
	Regel 1	Regel 2
Bedingung 1	J/N	J/N
...	J/N	J/N
Aktionsteil	**Aktionsanzeiger**	
Aktion 2	X bzw. ohne Angabe	...
...	...	...

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Erstellung einer Entscheidungstabelle

Für die systematische Erstellung von Entscheidungstabellen muss bekannt sein, wie viele Regeln sie maximal enthalten kann. Bei einer begrenzten Entscheidungstabelle berechnet sich die maximale Regelanzahl nach der Formel:

$$\textbf{max. Regelanzahl} = 2^{\textbf{max. Bedingungsanzahl}}$$

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Erstellung einer Entscheidungstabelle (Fortsetzung)

Bei einer begrenzten Entscheidungstabelle mit vier Bedingungen beispielsweise lautet die vollständige Standardtabelle:

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16
B1	J	J	J	J	J	J	J	J	N	N	N	N	N	N	N	N
B2	J	J	J	J	N	N	N	N	J	J	J	J	N	N	N	N
B3	J	J	N	N	J	J	N	N	J	J	N	N	J	J	N	N
B4	J	N	J	N	J	N	J	N	J	N	J	N	J	N	J	N

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Komplexe Entscheidungstabelle

Durch das systematische Vorgehen beim Erstellen der Bedingungen des gegebenen Sachverhaltes entsteht beispielsweise die folgende Entscheidungstabelle:

Überholvorgang		**R1**	**R2**	**R3**	**R4**
B1	Straßenverlauf übersichtlich	J	J	N	N
B2	Überholspur frei	J	N	J	N
A1	Gasgeben und überholen	X			
A2	Hinter dem Vordermann einordnen		X	X	X

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Komplexe Entscheidungstabelle (Fortsetzung)

Nur für den Fall, dass die beiden Bedingungen *B1* und *B2* zutreffen, findet die Aktion *A1* statt. Für jede andere Bedingung hingegen die Aktion *A2*.

Überholvorgang		**R1**	**R2**	**R3**	**R4**
B1	Straßenverlauf übersichtlich	J	J	N	N
B2	Überholspur frei	J	N	J	N
A1	Gasgeben und überholen	X			
A2	Hinter dem Vordermann einordnen		X	X	X

Überholvorgang		**R1**	**R2**	**R3**	**R4**
B1	Straßenverlauf übersichtlich	J	J	N	N
B2	Überholspur frei	J	N	-	-
A1	Gasgeben und überholen	X			
A2	Hinter dem Vordermann einordnen		X	X	X

In den Fällen *R3* und *R4* ist es offensichtlich ohne Bedeutung, ob die Überholspur frei ist oder nicht. Die Bedingung *B2* ist hier irrelevant und kann gestrichen werden. Dafür wird der so genannte Irrelevanzanzeiger ‚-' verwendet. Er darf nur bei Bedingungen verwendet werden, niemals bei Aktionen:

Komplexe Entscheidungstabelle (Fortsetzung)

Es entstehen zwei identische Regeln *R3* und *R4*, die konsolidiert werden:

Überholvorgang		**R1**	**R2**	**R3**
B1	Straßenverlauf übersichtlich	J	J	N
B2	Überholspur frei	J	N	-
A1	Gasgeben und überholen	X		
A2	Hinter dem Vordermann einordnen		X	X

Eine Entscheidungstabelle mit Irrelevanzanzeiger heißt komplexe Entscheidungstabelle.

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

Prozessorientierte konsolidierte Entscheidungstabelle

Die folgenden Regeln helfen beim systematischen Herleiten konsolidierter Entscheidungstabellen:

- (1) Die erste Regel senkrecht aufschreiben und die korrekten Aktionen auslösen. In der ersten Regel darf der Bedingungsanzeiger nur auf *J* oder *irrelevant* (‚-') gesetzt werden. Der Bedingungsanzeiger *N* darf niemals gesetzt werden.
- Für jede weitere Regel gilt:
  - (2) Das niederwertigste (unterste) *J* durch ein *N* ersetzen. Die höherwertigen Bedingungsanzeiger (darüber) bleiben unverändert.
  - (3) Alle niederwertigeren Bedingungsanzeiger dürfen nur *J* oder *irrelevant* gesetzt werden.
- (4) Die letzte Regel ist erreicht, wenn alle Bedingungsanzeiger *N* oder *irrelevant* sind.

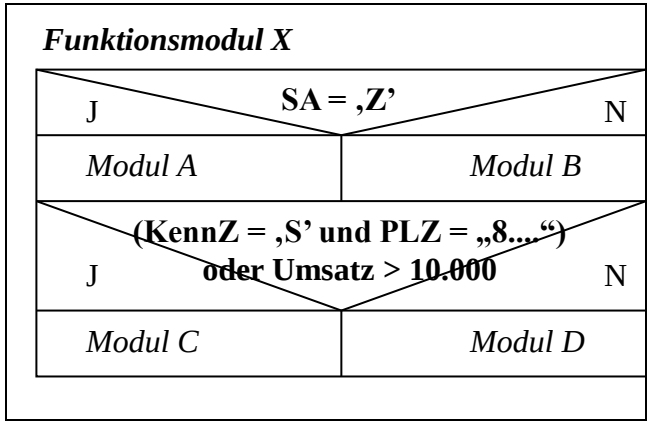
Funktionsmodul X

SA = ‚Z'	
J	N
Modul A	*Modul B*
(KennZ = ‚S' und PLZ = „8....") oder Umsatz > 10.000	
J	N
Modul C	*Modul D*

Demo!

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest



- (1) Die erste Regel senkrecht aufschreiben und die korrekten Aktionen auslösen. In der ersten Regel darf der Bedingungsanzeiger nur auf *J* oder *irrelevant* („-“) gesetzt werden. Der Bedingungsanzeiger *N* darf niemals gesetzt werden.
- Für jede weitere Regel gilt:
  - (2) Das niederwertigste (unterste) *J* durch ein *N* ersetzen. Die höherwertigen Bedingungsanzeiger (darüber) bleiben unverändert.
  - (3) Alle niederwertigeren Bedingungsanzeiger dürfen nur *J* oder *irrelevant* gesetzt werden.
- (4) Die letzte Regel ist erreicht, wenn alle Bedingungsanzeiger *N* oder *irrelevant* sind.

Softwarequalität und -test

Methodischer Aufbau und Durchführung von Fach- und Abnahmetest

---

Puh, fertig...