


# Softwaretechnik

## Kapitel 3




1. Wieso Software Engineering?
2. Vom Einzelkämpfer zum Großprojekt
- 3. Anforderungen und Test: Basis des Projekts**
4. Entwurf: Strukturen und nicht-funktionale Eigenschaften
5. Entwürfe notieren mit UML: Modelle im SE
6. Design Patterns: Entwurfserfahrungen nutzen
7. Management: Technik und Projektmanagement

**Systematische Softwareentwicklung**

Leibniz Universität Hannover SWT 2015/16 76

# Softwaretechnik

## Inhalt von Kapitel 3



### Systematische Softwareentwicklung


#### 3. Anforderungen und Test: Basis des Projekts

- Aufgaben und Herausforderungen
- Objekt-Orientierte Analyse (OOA)
- Abläufe erfassen mit Use Cases
- Testen – schon jetzt


Leibniz Universität Hannover SWT 2015/16 77

## Professionelle Anforderungsanalyse

**naiv**



**professionell**



Ist doch ganz einfach:  
Der Kunde beschreibt,  
was er möchte,  
und wir bauen das!

So einfach ist es leider nicht.  
Da lauern überall Fallen  
und Missverständnisse.  
Aber wenn man sie kennt,  
kann man damit umgehen.



Kurt Schneider Leibniz Universität Hannover SWT 2015/16 78

## „Kunde beschreibt was er möchte“

- Wenn jemand Software will, kommt er zu uns
- er muss eben genau sagen, was er will
- aber wir kennen uns ja auch aus, vielleicht sogar besser als so mancher Kunde

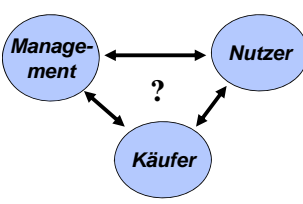
- Vielleicht braucht er vor allem bessere Geschäftsprozesse?
- Man muss schon zum Kunden gehen, und so genau kann der das gar nicht sagen
- Anfängerfehler! In seinem Fach ist der Kunde König und kennt sich aus

und überhaupt ...

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 79

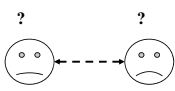
## Wer ist der Kunde?



Kurt Schneider Leibniz Universität Hannover SWT 2015/16 80

## „Symmetry of Ignorance“ beachten nach Gerhard Fischer

Unverständnis zwischen Informatikern und Kunden führt sehr oft zu vielen Missverständnissen und zu Frustration



Unverständnis auf beiden Seiten („symmetrisch“)

- Informatiker wissen nicht
  - wie das Geschäft läuft
  - was Kunden wollen, brauchen
  - was sie meinen
  - was das Problem ist
  - dass sie es nicht wissen

Hüten Sie sich davor zu glauben, Sie wüssten sowieso, was der Kunde will

- Kunden/Fach-Leute wissen nicht
  - was Software leisten kann
  - was Informatiker können
  - wovon sie reden (UML)
  - welche Lösungen möglich wären
  - dass die Informatiker sie auch nicht verstehen

Hinterfragen Sie scheinbare „Selbstverständlichkeiten“

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 81

## Anforderungserhebung als Kommunikationsaufgabe

Kommunikation = Sprechen und Verstehen

Beispiel  
„Sie müssen sich natürlich an alle üblichen Sicherheitsstandards halten; wir versuchen das auch stets.“

Beispiel  
(Sie wissen hoffentlich, welche Standards das sind?)  
(Traue ich Ihnen kaum zu, sonst müsste ich das nicht sagen)  
(Ich bin hier der Boss – deshalb dieser Ton)  
(Wir schaffen es selten, aber wir geben es nicht zu)

Auf allen Ebenen

- Antworten
- Beobachtungen
- Warnungen
- Feintuning

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 82

## Naives Loswerkeln rächt sich

- Stellen Sie früh fest, wer Anforderungen hat
- Versetzen Sie sich in seine/ihre Lage, so gut es geht
- Beschäftigen Sie sich mit dem Arbeitsgebiet des Kunden
- Nehmen Sie nichts für „selbstverständlich“, fragen Sie nach
- Glauben Sie nicht, Sie wüssten schon alles
- Kommunizieren Sie bewusst auf allen Ebenen
- Prüfen Sie, was Sie gefunden haben und wiederholen Sie

Betreiben Sie systematisch Requirements Engineering (folgt →)

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 83

## „Requirements Engineering“

**Erinnerung – wieder einmal Engineering:**  
„a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of **Requirements**“

**Konsequenzen auf den Umgang mit Anforderungen**

- **Kostendenken**
- **Qualitätsbewußtsein**
  - nicht nur subjektiv definiert
- **Anwendung von Normen und Regeln**
  - keine Künstler
- **Baugruppen und Wiederverwendung**
  - statt „not invented here“
- **Probleme durch Zerlegung lösen**
  - „divide et impera“

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 84

## Aufgaben im Requirements Engineering

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 85

## Anforderungen suchen: Elicitation

- „Herauskitzeln“ von Anforderungen
- Grober Ablauf
  1. Stakeholder herausfinden: wer ist betroffen?
  2. Umfeld des Systems erheben: Altsysteme, Schnittstellen, Doku.
  3. Sprechen mit Stakeholdern
    - „Sprechen“: Interviews, Workshops, Fragebögen
    - Beobachtung und Aufzeichnung
    - Regelrechte Elicitation-Techniken für „tacit knowledge“
- „Rohanforderungen“ müssen danach veredelt werden
- Beispiel Bankomat
  - Stakeholder: Kunden; die Bank; DB-Admin; Geldbefüller
  - Schnittstellen: zur DB, zu Schalterauszahlung u. Money-Mgmt.
  - Sprechen: Interview mit Schalterbeamten, Kundenfragebogen

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 86

## Interpretation von Anforderungen

- Die Anforderungen werden inhaltlich bearbeitet
  - Dazu ist Interpretation (=Zusprechen von Bedeutung) nötig
  - Diese Interpretation kann falsch sein!
- Spreu vom Weizen trennen
  - Was ist essenziell, was Nebensache
  - Was ist wirklich gefordert, was nur Erläuterung
- Ordnen und Sortieren, Strukturieren
  - Ähnliche Anforderungen zusammen
- Detaillieren und Konkretisieren
  - Formulierungen schärfen (erfordert Interpretation)
  - Prüfbar machen
- Beispiel Bankomat
  - Identifikation: Stornierung ist vor „ok“-Button gefordert, danach nicht mehr
  - Sortieren: Anford. an die Bedienung; an die Geschwindigkeit; an die Funktion
  - Konkretisieren: Jede Kontostandprüfung muss in 5 s. fertig und bestätigt sein

1 Identifikation

2 Strukturierung
 

- Verfeinern
- Verschmelzen
- Gruppieren

3 Konkretisierung

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 87

## SE Anforderungen verhandeln: Negotiation

1. Abhängigkeiten identifizieren
  - Eine Anforderung erzwingt andere.
    - Beispiel: Online-Überweisung => Alpha-Tastatur => Sicherheitsanf.
  - Anforderungen widersprechen sich
2. Widersprüchliche Anforderungen identifizieren
  - Bsp.: Mächtiges Werkzeug oder billiges Hilfsmittel?
  - Bsp.: Für Laien oder für Experten optimiert?
3. Konflikt ist üblich: Verhandlungsprozess, Moderation nötig
  - Es gibt nicht eine objektive Wahrheit, sondern viele Interessen
4. Inkonsistenzen auflösen – oder auch nicht!
  - Entscheidungsprozess
  - Selten eine *technische* Entscheidung
  - Mit gewissen Widersprüchen kann man leben

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 88

## SE Dokumentation

- Anforderungen zu hören reicht nicht
- Anforderungen müssen fixiert werden
  - Und trotzdem ändern sie sich!
  - Aber nur bei dokumentierten merkt man es
- Anforderungen dürfen sich ändern
  - Dann muss man nachdokumentieren
  - Und bezahlen muss das der Urheber (meistens)
- **Sehr wichtig: *mehr* dokumentieren als reine Anforderungen**
  - Gesprächspartner, Rollen, Ziele
  - Hintergründe und Randbedingungen
  - Annahmen
  - Absichten, Rivalitäten, alte Fehlversuche

- 1 Fixieren
- 2 Zerlegen in Einzelanforderungen
- 3 Mit Attributen beschreiben
- 4 Anforderungen verbinden

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 89

## SE Verbesserte Noteneinsicht Beispiel

- Stellen Sie sich vor, es gibt eine Idee:
  - Noten sollen schon vor Einsichtnahme zu sehen sein
  - Aber noch nicht freigegeben – kann sich ja noch ändern!
- Daran hängen viele weitere Anforderungen
  - Von Studierenden
  - Von Lehrenden
  - Von Fachschaft
  - Von Prüfungsamt
- Werden erhoben wie oben beschrieben (Elicitation, Int., ...)

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 90

## SE Dokumentation (Auszug)

[R01] Studierende haben Zugang über Name und PIN  
*Entscheidungstabelle, in Workshop abgesegnet (Negot.-WS)*

[R02] PIN fünfstellig, nicht mit Matrikelnummer korreliert  
*Matrikelnummer steht öffentlich auf Studentenausweis (Dekan)  
Fünfstellige PINs ausreichend (Theorie-Vertreter)*

[R03] Alle Verbindungen zum Notenserver sind verschlüsselt  
*Fordern Dekanat, Prüfungsamt, Dozenten. Studierende implizit.*

[R04] Zugriff nur auf die eigenen Noten möglich  
*Kein online-Gesamtaushang, damit keine externen Robots (anonyme)  
Notenprofilauswertungen vornehmen können*

[R05] Statistik ebenfalls nur dort sichtbar, nicht offen  
*Siehe [R04]*

[R06] Alle eigenen Noten des laufenden Prüfungszeitraums zusammen bei einem Login zu sehen  
*Studenten wollen sich weder mehrfach einloggen noch Klausurnamen merken*

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 91

## SE Vielfältige Beziehungen von Anforderungen

*manuell kaum zu überblicken*

```

graph LR
    PF[Parameter-formate] -- "verwendet Format" --> P[Parameter]
    P -- "definiert in" --> A[Anforderungen]
    A -- "verwendet" --> P
    A -- "definiert in" --> S[Schnittstellen]
    S -- "verwendet" --> A
    A -- "verwendet Begriff" --> G[Glossar]
    A -- "verfeinert" --> G
    RB[Randbedingungen] -- "wird eingeschränkt durch" --> A
  
```

- Es gibt noch mehr Beziehungsarten
- Es gibt weitere Dokumente
- Anforderungen werden verfeinert: Dokument->Satz
- Man kann Regeln (über Beziehungen) definieren
- ...
- Und so weiter!

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 92

## SE Validierung und Verifikation

- Begriffe
  - Validierung := „sind die dokumentierten auch die wirklichen Anforderungen?“
    - Egal, was bisher aufgeschrieben wurde
  - Verifikation := „stimmen Anf. mit zuvor dokumentierten überein?“
    - Egal, ob der Kunde das (noch) wirklich will

**Validierung: Inhaltliche Prüfung**  
*Ist der Entwurf das, was der Kunde will?*

**Verifikation: Formale Prüfung**  
*Erfüllt der Entwurf die Spezifikation?*

- Prüfungstechniken
  - Validierung:** Befragung, Interview, Nachprüfung, Konfrontation
  - Verifikation:** Formale Verfahren, Konsistenzprüfungen, Erreichbarkeit

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 93

## Requirements Management

- Die Verwaltung der Anforderungen
- Leicht handhabbar ablegen
  - Werkzeug (wie Doors, RequisitePro)
- Änderungen und Versionen im Griff behalten
  - Besonders interessant: auch die Gründe für Änderungen erfassen und aufbewahren!
- Änderungen weitergeben, Verfolgbarkeit in beide Richtungen (Tracing)
  - Das ist sehr schwer!
  - Schon seit Jahrzehnten ungelöst
  - Hier helfen Werkzeuge

**Req. Management**

```

graph TD
    RM[Req. Management] --> AM[Änderungsmanagement]
    RM --> VT[Verfolgbarkeit Tracing]
    AM --> AW[Änderungswünsche]
    AM --> VA[Versionen von Anforderungen Historie]
    AM --> PA[Propagieren der Änderung -> Tracing]
    VT --> FA[Festhalten der Annahmen, Quellen von Anforderungen pre-tracing]
    VT --> AS[Auswirkungen von Änderung im System post-tracing]
            
```

**Verfolgbarkeit (Tracing)**

Festhalten der Annahmen, Quellen von Anforderungen (pre-tracing)

Auswirkungen von Änderung im System (post-tracing)

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16 94

## Klassifikation von Anforderungen

- Funktionale Anforderungen an das Produkt**
  - Was die SW tun soll
- Nicht-funktionale Anforderungen an das Produkt**
  - In welcher Weise die SW es tun soll
    - Qualitätsanforderungen
      - Flexibilität, Portabilität, Wartbarkeit, Schnelligkeit usw.
    - Andere nicht-funktionale Anforderungen
      - Mengengerüst
- Prozessanforderungen**
  - Welchen Abläufen und Vorgaben das Projekt folgen soll
- Projektanforderungen**
  - Zeit- und Geldbudget des Projekts

Constraints

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16 95

## Constraints

- Produkt-Anforderung beschreiben, welche Funktionen oder Eigenschaften die Software haben soll.
- Prozess- und Projektanforderungen legen fest, mit welchen Mitteln und auf welchem Wege das Projekt ablaufen soll.
- Constraints schränken dagegen den Lösungsraum ein.
  - „MVC muss verwendet werden“
  - „Das Play!-Framework darf nicht verwendet werden“
  - „Unsere hausinterne Bibliothek ist vorzuziehen“
  - „Zur Planung muss MS Project verwendet werden“
- Es gibt Überschneidungen; Abgrenzung ist schwierig

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16 96

## Klassifikation

(Fiktives Beispiel; Auszug)

Id	Anforderung	Kategorie
[R01]	Studierende haben Zugang über Name und PIN <i>Entscheidungstabelle, in Workshop abgesegnet (Negot.-WS)</i>	Funktional
[R02]	PIN fünfstellig, nicht mit Matrikelnummer korreliert <i>Matrikelnummer steht öffentlich auf Studentenausweis (Dekan) Fünfstellige PINs ausreichend (Theorie-Vertreter)</i>	Sicherheit Qualitätsanford.
[R03]	Alle Verbindungen zum Notenserver sind verschlüsselt <i>Fordern Dekanat, Prüfungsamt, Dozenten. Studierende implizit.</i>	Sicherheit Qualitätsanford.
[R04]	Zugriff nur auf die eigenen Noten möglich <i>Kein online-Gesamtausgang, damit keine externen Robots (anonyme) Notenprofilauswertungen vornehmen können</i>	Funktional
[R05]	Dozent kann Jahresstatistik anzeigen <i>Genauer: NUR Dozent kann das (siehe R04)</i>	Funktional
[R06]	Noten aller Prüfungen einer Person zusammen sichtbar <i>Studenten wollen sich weder mehrfach einloggen noch Klausurnamen merken</i>	Funktional

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16 97

## Klassifikation

(Fiktives Beispiel; Auszug, fortgesetzt)

Id	Anforderung	Kategorie
[R42]	Der jetzige Arbeitsablauf von Dozenten und Prüfungsamt ist zu beachten. Mehraufwand ist zu vermeiden ...	Funktional (Geschäftsproz.)
[R61]	Die technische Schnittstelle zu den HIS-Programmen steht nicht zur Disposition und muss beachtet werden.	Constraint
[R82]	Das Programm soll bis zum Sommersemester operativ laufen	Projektanford.
[R89]	Alle Studierenden der Fakultät sollen dann alle neuen Klausurnoten einsehen können.	Mengengerüst
[R90]	Eine Zugriffsrate von 200 Personen pro Tag, bis zu 20 gleichzeitig muss verkraftet werden und darf nicht zu Antwortzeiten von über 10 sek. führen	Mengengerüst Qualität/Geschw. unklar def.

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16 98

## Übersicht: Techniken um Anforderungen

- Bisher:**
  - Wieso sind Anforderungen so wichtig?
  - Wo sind die Schwierigkeiten?
  - Aufgaben und Forschungsthemen im RE
- Weiter mit Techniken zum RE:**
  - Denken in Objekten: Objekt-Orientierte Analyse
  - Abläufe im Zentrum: Use Cases
  - Testen und Anforderungen gehören zusammen

Elicitation  
Interpretation  
Negotiation  
Documentation  
Validation  
Tracing

### Anforderungen und Testen hängen eng zusammen

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16 99

# Softwaretechnik

## Inhalt von Kapitel 3

### Systematische Softwareentwicklung

#### 3. Anforderungen und Test: Basis des Projekts

- Aufgaben und Herausforderungen
- Objekt-Orientierte Analyse (OOA)
- Abläufe erfassen mit Use Cases
- Testen – schon jetzt

Leibniz Universität Hannover SWT 2015/16 100

## Neue Sichtweise: OOA (objekt-orientierte Analyse)

- Die Welt besteht aus Objekten (Dingen, Agenten)
- Objekte sind unterscheidbar
- Sie schicken sich Nachrichten und interagieren auf diese Weise

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 101

## Methode im Zusammenhang Objekt-Orientierte Analyse

Erinnerung:

- es geht hier um die *oo Analyse eines Systems und seines Umfeldes*

Beschreibung Schritt für Schritt (=Methode)

1. Systemgrenzen festlegen
2. Objekte finden (stehen für reale Dinge, Konzepte etc.)
3. Objekte sortieren und klassifizieren (Domain Concepts)
4. Interaktion und Abhängigkeiten ermitteln

Schritte 1-4 werden zwei Mal absolviert:

- Zunächst IST-Zustand ermitteln
- Dann den Soll-Zustand

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 102

## Beispiel: OOA für das Computerspiel

Zu Anfang: Idee im Kopf

Kurt Schneider Blast thru Special Edition, rockSolid Software ca. 2002 SWT 2015/16 103

## 1. Schritt: Systemgrenzen festlegen

Wie viele Levels?

und so weiter...

Highscore?

Musik?

Ladbare Elemente?

Internet-Option?

Spielstandspeicher?

Spieler-erweiterbar?

Kurt Schneider Blast thru Special Edition, rockSolid Software ca. 2002 SWT 2015/16 104

## 2. Schritt: Objekte finden

### Soll-Analyse

„Entwickeln Sie ein Computerspiel, bei dem man einen Schläger (unten im Bild) lenkt.“

Ein Ball wird ins Bild geschossen, er trifft auf Hindernisse (Steine, Bomben) und zerstört diese. Das gibt Punkte. An anderen prallt er ab.

Wenn der Ball am Schläger vorbei unten aus dem Bild läuft, ist er weg. Dann kommt der nächste Ball...“

Objekte suchen:  
Substantive unterstreichen

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 105

**Objekte festlegen**  
Danach kommen Entwurf u. OO Programmierung

**DANN**  
3. Identifizierte Objekte klassifizieren und Klassen ableiten  
4. Beziehung und Interaktion festlegen  
... weiter mit UML

Kurt Schneider      Blast thru Special Edition, rockSolid Software ca. 2002      SWT 2015/16 106

**Inhalt von Kapitel 3**

**Systematische Softwareentwicklung**

**3. Anforderungen und Test: Basis des Projekts**

- Aufgaben und Herausforderungen
- Objekt-Orientierte Analyse (OOA)
- Abläufe erfassen mit Use Cases
- Testen – schon jetzt

Softwaretechnik

Leibniz Universität Hannover      SWT 2015/16 107

**Wichtig: Use Cases**

- Definition**  
Beschreibung einer Interaktion zwischen (min.) einem Akteur und dem System, durch das ein Ziel des Hauptakteurs erreicht wird.
- Beispiel Bankomat (Kern eines Use Case)**

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16 108

**Wozu Use Cases?**

- Beschreiben das Systemverhalten**
  - Das ganze, extern wahrnehmbare Systemverhalten
  - Aber nicht mehr als das (keine Interna, keine Qualitätsanforderungen)
- Meist von Entwicklern geschrieben**
  - In Zusammenarbeit mit Fach-Experten
  - Diskussion führt zu Reflexion, klärt vieles („elicitation“)
- Sind Basis für alle weiteren Aktivitäten**
  - Stellen Analyse-Schritt dar (Ist-Use Cases)
  - Repräsentieren Requirements (Soll- Use Cases)
  - Design muss Use Cases ermöglichen
  - Testfälle können aus Use Cases abgeleitet werden
    - Hinweis: Einfacher Use Case ist nahe an Testfall – ist aber keiner
    - Use Case: „Klasse“ von Abläufen. Testfall: ein ganz konkreter
  - Als „Contract“ (Vertrag) dienen sie beim Abnahmetest

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16 109

**Formate für Use Cases**

- Im Prinzip ist jedes Format möglich - wenn es hilft
- In der Literatur dominieren die „graphischen UML-Modelle“

- Aber Achtung!**
  - Diagramme zeigen nur den Überblick
    - Welche use cases gibt es?
    - In welcher Beziehung stehen Actors und use cases?
  - Sie zeigen nicht: den eigentlichen „Kern des Use Cases“
    - die Abfolge der Schritte, Vor- u. Nachbedingung etc.

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16 110

**Use Case-Diagramme**

**Allgemein**

- Aktoren mit Namen Name
  - Rechteck für Systeme
  - Stereotype <actor>
  - Strichmännchen sonst
- Use Case mit Namen UC name
  - <Actor> nimmt teil an <UC>
- Ausgeklammerter Teil-Use Case
- Optionale Erweiterung
- Systemgrenze (Rahmen)


**Beispiel**

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16 111



## Formate für Use Cases

- Die Interaktionen können in vielen Notationen beschrieben werden
  - Ablaufdiagramm
  - Struktogramme
  - Petri-Netze
  - Pseudo-Code
  - Echter Code
- Wichtig: ihre Aufgabe ist Kommunikation unter Menschen
- Daher eignet sich für viele Use Cases besonders: **Text**
- Aber den sollte man strukturieren



Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16      112

## Stile textbasierter Use Cases

nach Alistair Cockburn / (meine deutschen Entsprechungen)

- Narrative / Geschichtchen** (noch kein Use Case)
 

„Peter braucht schnell Geld fürs Kino. Er betritt den Bankomatenraum und steckt seine ec-Karte ein. Dadurch wird er identifiziert und durch seine PIN authentifiziert. Er gibt den Betrag ein, erhält Geld und Karte.“
- Brief / Kurzfassung**
  - Ein Absatz: Zusammenfassung, oft des Haupt-Erfolgsszenarios.
- Casual / Informell**
  - Mehrere informell geschriebene Absätze, darin mehrere Abläufe.
- Fully dressed / Detailliert**
  - Am ausgefeiltesten.
  - In Tabellenform sind alle Schritte und Varianten detailliert aufgeführt. Zusatzangaben wie „Voraussetzungen“, „Garantien“.

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16      113

## Elemente von Use Case Beschreibungen

- Akteure / Actors**
  - außerhalb der Systemgrenzen
  - Interagiert mit dem System im Rahmen des Use Case
  - Hauptakteur (Primary Actor): der, dessen Ziel erreicht werden soll
- Stakeholder**
  - Jemand mit Interesse am Use Case - muss nicht daran teilnehmen
  - Wer ist das? Fertigkeiten, Aufgaben, Hintergrund? -> Separate Tabelle
- Systemgrenzen / Scope**
  - Was gehört noch zum modellierten System, was nicht mehr?
- Erfolgsfall und Misserfolgsfall / success and failure**
  - Wenn der gewünschte Fall eintritt bzw. wenn er nicht eintritt
- Garantie**
  - Was man auch im Misserfolgsfall noch sicherstellen kann

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16      114

## Formular für einen vollen Use Case

Use Case Nr. <nr>	<Name: kurzer, aktiver Satz>		
Umfeld	<Wo, unter welchen Umständen ausgeführt>		
Systemgrenzen	<Scope: was gehört dazu, was nicht?>		
Ebene	<Überblick, Aufgabe oder Teilfunktion>		
Hauptakteur	<Kurzbeschreibung dessen, der Ziel hat>		
Stakeholder u. Interesse	Stakeholder	Interesse	
	<erste>	<ihre Interessen>	
	<zweiter>	<seine Interessen>	
Voraussetzung	<wovon kann man bei Beginn ausgehen?>		
Garantien	<was in jedem Fall gewährleistet ist>		
Erfolgsfall	<Zustand bei erfolgreicher Beendigung>		
Auslöser	<wann der Use Cases startet>		
Beschreibung	Schritt	Aktion	
	1	<von Auslöser bis Aufräumen>	
	2	<z.B.: „zurück in Ausgangsz.“>	
Erweiterungen	1a	WENN ... DANN <and. Use Case>	
Technologie ...	<Variationen durch Technik>		

in Anlehnung an Cockburn, A.: Writing Effective Use Cases. Addison-Wesley, 2001

**Szenario := Ein Ablauf in einem Use Case**

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16      115

## Drei Ebenen

Am wichtigsten ist die „Sehhöhe“ der Aufgaben

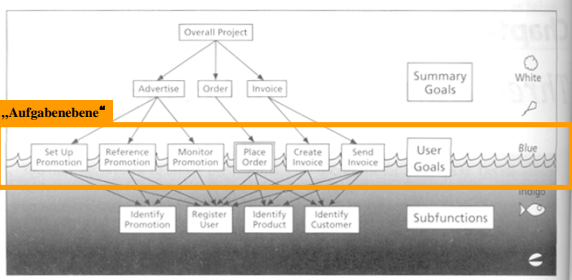


Figure 5.1 Use case levels. The use case set reveals a hierarchy of goals—the ever-unfolding story.

Kurt Schneider      aus Cockburn, A.: Writing Effective Use Cases. Addison-Wesley, 2001      SWT 2015/16      116

## Beispiel: Bankomat

### UC Geld abheben (Teil 1)

UC 1	Geld abheben
Umfeld	Bankautomat im Freien oder im Foyer
Systemgrenzen	Kontoverwaltung und Bankomaten-HW existieren bereits
Ebene	Hauptebene
Hauptakteure	Kunde (dieser oder einer anderen Bank)
Stakeholder u. Interessen	Kunde möchte schnell und sicher Geld erhalten Bank möchte Personalaufwand reduzieren
Voraussetzungen	Kunde hat gültige Karte Bankomat läuft und hat Verbindung zum Banksystem
Garantie	Es wird entweder (Geld ausgezahlt und abgebucht) oder (weder ausgezahlt noch abgebucht)
Erfolgsfall	Der Kunde hat den gewünschten Geldbetrag erhalten; dieser Betrag wurde von seinem Kont
Auslöser	Kunde steckt Karte in den Leser
Beschreibung	<ol style="list-style-type: none"> <li>Kunde steckt Karte in den Leser</li> <li>System fragt nach PIN</li> <li>Kunde gibt PIN ein</li> <li>System fragt nach gewünschter Aktion</li> <li>Kunde wählt "Abhebung" aus</li> <li>System bietet Beträge an</li> <li>Kunde gibt Wunschbetrag an</li> <li>System prüft Verfügbarkeit und zahlt Betrag aus</li> <li>System schließt die Sitzung, gibt Karte aus.</li> </ol>

[R27] verfügbar ist Guthaben plus Kreditrahmen

Fortsetzung folgt ...

**Der hier beschriebene UC-Ablauf soll möglich sein. Weitere Anforderungen kann man annotieren/anfügen**

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16      117

**Beispiel: Bankomat**  
UC Geld abheben (Teil 2)

<b>Beschreibung</b>	1. Kunde steckt Karte in den Leser 2. System fragt nach PIN 3. Kunde gibt PIN ein 4. System fragt nach gewünschter Aktion 5. Kunde wählt "Abhebung" aus 6. System bietet Beträge an 7. Kunde gibt Wunschbetrag an 8. System prüft Verfügbarkeit und zahlt Betrag aus 9. System schließt die Sitzung, gibt Karte aus.
<b>Erweiterungen</b>	3a. WENN keine gültige PIN eingegeben, DANN zurück zu 2 (nach drei Fehlversuchen) 8a. WENN Wunschbetrag nicht verfügbar ist DANN 8a.1 bietet das System den höchsten verfügbaren Betrag an 8a.2 Kunde bestätigt, dass er ihn akzeptiert 8a.2a: Falls nicht akzeptiert, beende Sitzung (9). 8a.3 System zahlt Betrag aus, weiter bei 9
<b>Technologie</b>	Bei Geräten mit Irissensor ist alternativ zur PIN-Eingabe auch ein Iris-Scan möglich

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16      118

**Beispiel 2: UC Überweisung tätigen**  
Teil 1

<b>UC 2</b>	<b>Überweisung tätigen</b>
<b>Umfeld</b>	Bankautomat mit Alpha-Tastatur im Foyer
<b>Systemgrenzen</b>	HW und Bankensoftware werden unverändert übernommen
<b>Ebene</b>	Hauptebene
<b>Hauptakteure</b>	Kunde dieser Bank
<b>Stakeholder u. Interessen</b>	Kunde möchte schnell und direkt vor Ort Überweisung durchführen Bank möchte Aufwand für Beleglesen sparen
<b>Voraussetzungen</b>	Kunde hat ein überweisungsfähiges Konto bei dieser Bank Bankomat läuft und hat Verbindung zum Bankensystem Kunde hat sich über seine Karte und PIN identifiziert und authentifiziert
<b>Garantie</b>	Am Ende der Sitzung wird ummissverständlich angezeigt, ob die Transaktion erfolgreich war
<b>Erfolgfall</b>	Die Überweisung ist abgeschickt und ins normale Bankensystem übergeben
<b>Auslöser</b>	Kunde wählt Aktion "Überweisung" aus
<b>Beschreibung</b>	1. Kunde wählt Aktion "Überweisung" aus 2. System weist darauf hin, dass mTAN erforderlich ist 3. Kunde bestätigt 4. System zeigt Formular wie auf Papier an 5. Kunde füllt das Formular aus 6. System verschickt mTAN per SMS, fragt Kunden danach 7. Kunde gibt mTAN ein 8. System übergibt Überweisung an das Bankensystem 9. System <i>bestätigt erfolgreiche Überweisung</i> 10. System beendet die Sitzung

Fortsetzung folgt ...

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16      119

**Beispiel 2: UC Überweisung tätigen**  
Teil 2

<b>Beschreibung</b>	1. Kunde wählt Aktion "Überweisung" aus 2. System weist darauf hin, dass mTAN erforderlich ist 3. Kunde bestätigt 4. System zeigt Formular wie auf Papier an 5. Kunde füllt das Formular aus 6. System verschickt mTAN per SMS, fragt Kunden danach 7. Kunde gibt mTAN ein 8. System übergibt Überweisung an das Bankensystem 9. System <i>bestätigt erfolgreiche Überweisung</i> 10. System beendet die Sitzung
<b>Erweiterungen</b>	3a. WENN Kunde nicht bestätigt, DANN Abbruch (10) 5a. WENN Pflichtfelder nicht ausgefüllt, DANN weist System auf Fehlendes, zurück zu 5 5b. WENN Kunde abbricht, DANN beende Sitzung (10) 7a. WENN keine gültige mTAN, DANN zurück zu 6. Sperrung nach drei Fehlversuchen 9a. WENN Kunde Beleg wünscht, DANN 9a.1 System druckt Beleg, beendet Sitzung (10)
<b>Technologie</b>	In Filialen mit Fingerkuppenleser ist keine TAN nötig

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16      120

**Use Case-Template**  
hier beispielsweise für Excel; finden Sie auf StudIP

<b>UC &lt;Nr.&gt;</b>	<b>&lt;Name: kurzer, aktiver Satz, was erreicht werden soll&gt;</b>
<b>Umfeld</b>	<informell: wo und unter welchen Umständen ausgeführt>
<b>Systemgrenzen</b>	<was gehört dazu, was nicht>
<b>Hauptakteure</b>	<wer hat das Ziel und welches>
<b>Stakeholder u. Interessen</b>	<Bezeichner>      <Interessen>
<b>Voraussetzungen</b>	<davon kann man bei Beginn ausgehen>
<b>Garantie</b>	<in allen erwähnten Erfolgs- und Misserfolgssfällen sichergestellt>
<b>Erfolgfall</b>	<was im Idealfall erreicht wird>
<b>Auslöser</b>	<Wodurch wird die Interaktion gestartet?>
<b>Beschreibung</b>	1. erster Schritt ist meist der Auslöser 2. 3.
<b>Erweiterungen</b>	2a. WENN <Bedingung, um in 2 abzuweichen> DANN <Aktion in diesem Fall, evtl. mehrere Schritte: 2a.1, 2a.2
<b>Technologie</b>	3a. Besonderheit wg. Technologievarianten

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16      121

**Wie man Use Cases einsetzt**

- Akteure identifizieren**
  - Wer benutzt das System? Wozu? → Stakeholder-Profil-Table
  - Wer ist mit Ein- und Ausgaben konfrontiert? → Acteure
- Systemgrenzen festlegen**
  - Akteure sind außen, Use Cases beschreiben das Hin und Her
- Wichtigste Use Cases identifizieren**
  - Narrative, Beziehung zwischen Use Case und Actor herstellen (UML)
- Zunehmend genauer beschreiben**
  - Narrative (Geschichtchen)
  - Kurzfassung -> Informell oder Detailliert
    - Erfolgsszenarien
    - Fehlerszenarien
  - Use Cases mit dem Kunden diskutieren
  - Bei Bedarf: grafische Übersicht (UML, Beziehungen) ergänzen

Zwei Schleifen: über alle Use Cases, durch die Ebenen; mit Kunden

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16      122


**Wann ist man fertig?**

- Fertig, sobald**
  - alle Hauptakteure identifiziert wurden
  - ihre Ziele genannt haben
  - Jedes Ziel von mindestens einem Use Case abgedeckt ist.
- Und wenn alle Use Cases klar und verständlich sind:**
  - Sponsoren bestätigen („agree“), damit Abnahme machen zu können
  - Nutzer bestätigen, dass Systemverhalten zutreffend beschrieben
  - Sponsoren bestätigen, dass die Use Cases (vorerst) vollständig sind
- Natürlich ist man dann *nicht endgültig* fertig!

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16      123



## Tipps und Tricks

- Use Cases erfassen *nur funkt.* Anforderungen, die aber *alle*
- Zuerst in die Breite arbeiten, dann in die Tiefe 
- Eher zu wenig als zu viel schreiben  
– *Sonst liest es doch keiner, dann hilft es nichts*
- Es geht um das Geschriebene (UC), nicht das Gemalte (Diag.)
- Diskussion über Formalität und Stil nicht übertreiben  
– *Es kommt letztlich halt auf Klarheit an - egal, wie*
- Keine „if-Konstrukte“ in Hauptszenarios  
– *Sond. Extensions/Abzweig-Szenarien*

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16    124

## Wiederholung: Anforderungen

- Anforderungen: Bezugspunkt von SW-Entwurf, -Entwicklung, -Test
- Anforderungen sind aber nicht leicht zu bekommen
  - Sie erfordern Kommunikation mit „Fach-Leuten“
  - „Symmetry of ignorance“
  - Alle Schwierigkeiten üblicher Kommunikation eingeschlossen
- Requirements Engineering umfasst mehrere Aktivitäten
  - Elicitation: Herauskitzeln
  - Interpretation: Das Gehörte deuten
  - Negotiation: Priorisieren, Widersprüche auflösen, Verhandeln
  - Dokumentieren: Aufschreiben
  - Validation & Verification: Prüfen
  - Managen und Tracen: Verfolgen, Änderungen einarbeiten
- Wichtige RE-Techniken
  - Structured Analysis (SA): Anforderungen aus Datensicht [historisch]
  - Objekt-orientierte Analyse: Die Welt aus interagierenden Objekten
  - Use Cases: Abläufe im Zentrum

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16    125

## Softwaretechnik

### Inhalt von Kapitel 3

#### Systematische Softwareentwicklung

#### 3. Anforderungen und Test: Basis des Projekts

- Aufgaben und Herausforderungen
- Objekt-Orientierte Analyse (OOA)
- Abläufe erfassen mit Use Cases
- Testen – schon jetzt

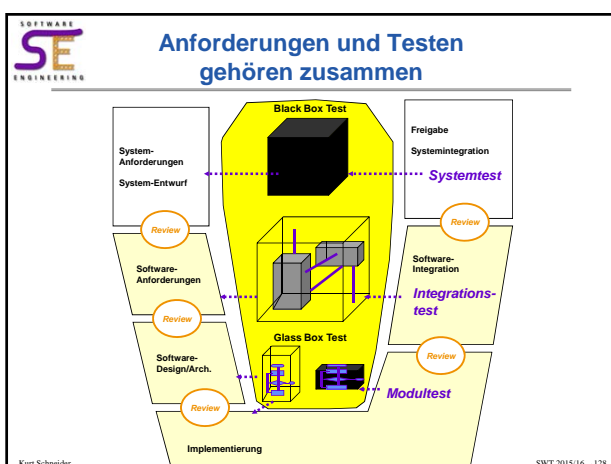
Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16    126

## Was ist Testen?

≡<sub>Def</sub> „Ausführen eines Programms mit dem Ziel, Fehler zu finden“

- Das heißt also:
  - das Programm muss ausgeführt werden
  - jeder gefundene Fehler ist ein Erfolg für den Tester
  - Für den Entwickler nur indirekt
- Testen ist also nicht:
  - Wunsch zu zeigen, dass das Programm in Ordnung ist
  - Herumprobieren-Ändern-wieder Probieren
  - ein unmittelbar konstruktiver Verbesserungsschritt
- Fazit: Testen erscheint zunächst destruktiv
  - verlangt gewisse Schadenfreude: Autor hat die nicht!
  - enthält viele kreative Elemente (was könnte schiefgehen?)

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16    127



## Beispiele für Testfälle

Einfache Fälle: mathematisch und klar

```
public int quersum (int zahl)
```

– [ R01 ] die Methode gibt die Quersumme einer zahl aus

– Testfälle: (was geht rein? → was muss dann rauskommen?)

- T01: quersum (123) → 6
- T02: quersum (1234) → 10
- T03: quersum (12345) → 15
- T04: quersum (123456) → 21
- ... und so immer weiter?

Req. Engineer:  
„Aha, keine einstellige / iterierte Quersumme, denn da wäre  
itQuer (1234) =  
itQuer (10) = 1 (< 10 !)  
Anforderung R01 ist jetzt klarer“

– [ R02 ] ein negatives Vorzeichen wird ignoriert

- T42: quersum (-529) → - 16

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16    129

### Beispiel: Kassensoftware testen

- Einzelposten gegeben
- Gesamtsumme ermitteln (YHTEENSA)
- Und Barrückgabe (TAKAISIN)
- Zu einfach?
  - Es sind aber finnische Belege!
  - Kunde ist ein finn. Supermarkt
  - Na und?
  - Dann sehen Sie mal genau hin.

### Finnland verwendet keine 1c, 2c Münzen

- Daher wird gerundet
- Auf- und Ab; das mittelt sich aus
- Idealerweise explizit ausgewiesen
  - „Pyörästys 0,01“

### Aber das scheint noch nicht alles zu sein

- Nach einiger Zeit stoßen Sie auf diesen „Testfall“:

**Was tun Sie jetzt?**

[ R02 ] Bei Kartenzahlung wird nicht gerundet

**Rationale (Begründung):**  
„Bei Kartenzahlung muss man ja keine Münzen herausgeben – da kann man exakt zahlen“

### Finnische Quittung

Beispiel aus dem echten Leben

In Finnland gibt es keine 1c und 2c-Münzen. An der Kasse im Supermarkt wird der Endbetrag daher auf- oder abgerundet.

[ R01 ] Die Summe der Einzelbeträge wird auf volle 5c mathematisch gerundet.

**Eingaben**

T01: finCash (8,42 €) → 8,40 €

T02: finCash (1.043,43 €) → 1.043,45 €

T03: finCash (18,00 €) → 18,00 €

T04: finCard (1.043,43 €) → 1.043,43 €

T05: finCash (-1,21 €) → 1,20 € Pfand zurück

[ R02 ] Bei Kartenzahlung wird nicht gerundet

**Soll/Ausgaben**

→ 8,40 €

→ 1.043,45 €

→ 18,00 €

→ 1.043,43 €

→ 1,20 € Pfand zurück

### Von der Anforderung zum Testfall

direkter Zusammenhang

**Phishing, nichtsahn**

ID	Kurzbeschreibung	Anforderung	Rationale
[R01]	Phishing verhindern	Die Karte soll mehrfach teilweise eingezogen und ausgegeben werden	Vorsatzgeräte können dann nicht mehr den Streifen lesen
[R02]	PINs haben genau vier Stellen	Akzeptiere nur PINs mit exakt vier Stellen	Internationaler Standard; auch nicht mehr Stellen
[R03]	Alle vierstelligen Zahlen (0-9) sind als PIN erlaubt	Alle vierstelligen Zahlen aus den Ziffern 0 bis 9 sind als PIN erlaubt	Einige Banken erlauben das, also muss überall diese Eingabe erlaubt sein

**Testfall: Setup**

Testfall	Setup	Soll
T01.1	Karte in Spalt geben	Wird min. zwei Mal in jede Richtung bewegt
T02.1	Beim Verändern der PIN vierstellige wählen	wird akzeptiert
T02.2	Beim Verändern der PIN fünfstellig wählen	wird zurückgewiesen
T03.1	PIN Zahl im Mittelfeld wählen (z.B. 4711)	wird akzeptiert
T03.2	Als PIN den Grenzwert 0000 wählen	wird akzeptiert

- Kann noch hinzukommen:
  - setUp() von Datenbanken, Ausgangssituation
  - Aufwändige Beschreibung der Vorbereitung/Durchführung
  - Angabe von Toleranzen

### Gute Anforderungen und Testfälle

- Gute Testfälle
  - Konkret
  - Spezifisch
  - Leicht anwendbar
  - Objektiv beurteilbar
  - Testbar
- So ausgewählt, dass sie viele Fehler finden!

- Anforderung:
  - was soll Programm tun?
- Test:
  - Tut das Programm, was der Test verlangt?
- Ideal:
  - Programm besteht alle Tests
  - ↔
  - es erfüllt die Anforderungen

### Use Case vs. Testfall ähnlich, aber nicht dasselbe

Beschreibung	<b>Schritt Aktion</b> 1 Kd drückt „Überweisung“ 2 Hinweis erscheint, dass er TAN braucht. Kd quittiert.. 3 Formular wie Papier erscheint 4 Kd füllt die Felder aus 5 Ist Kd fertig, schaltet er weiter 6 Ausgefüllte Überweisung wird gezeigt, nach TAN gefragt 7 Kd gibt (verdeckt) TAN ein 8 Sys fragt, ob Beleg gewünscht 9 Sys druckt Beleg 10 Sys verabschiedet sich, Ruhezust.	<b>Testfall</b> Ablauf gleich Aber konkrete Eingaben in Formular - Empfänger: „Bill“ - Betrag: 50 € - Kontonr.: 3312 - BLZ: 500 000 12 - Grund: „Wette“ - TAN=45312
Erweiterungen	<b>Konkrete Soll: Überweisung bestätigt (und durchgef.)</b> 2a WENN Kd keine da hat DANN beendet er die Aufgabe 5a WENN ein Feld fehlt oder falsch ist DANN fragt System nur nach diesem Feld. Weiter: 6 7a WENN ungültige TAN, DANN Fehlermeldung, zurück zu 6	
Technologie ...	In Filialen mit Fingerkuppenleser keine TAN nötig	

Kurt Schneider in Anlehnung an Cockburn, A.: Writing Effective Use Cases. Addison-Wesley, 2001 SWT 2015/16 136

### Vollständig testen: fast nie möglich

- Abschätzung:**
  - Für einen 16-bit-Parameter gibt es  $2^{16}$  mögliche Werte
  - Für drei Parameter schon  $2^{48} \approx 2,8 \cdot 10^{14}$
  - Bei 100 Tests pro Sekunde wären das über  $10^{12}$  Sekunden - fast 90.000 Jahre!
- Oft gibt es noch mehr Permutationen**
  - Viele Programme haben mehr als drei Parameter
  - Dazu kommen Sonder- und Fehlerfälle
  - Oberflächen führen zu noch mehr Varianten

**• Es reicht nicht, die *Eingaben* zu erzeugen! Auch *Sollwerte* für Test nötig!**

- Erstaunlich:** in Einzelfällen ist vollständiges Testen trotzdem möglich; und es lohnt sich! *Beispiel: Kleines Steuergerät*
  - Parameterwerte nur 0 bis 255
  - In der Regel nur ein oder zwei Parameter ( $255^2 = 65.025$ , im 10 Min.-Bereich)
  - Ergebnisse leicht ableitbar / formal spezifiziert (über Formel)

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 137

### Einige Definitionen

**Prinzip**

**Prüfling** =<sub>Def</sub> zu prüfende Software (*nicht*: deren Autor!)

**Autor** =<sub>Def</sub> Verfasser des Prüflings

**Laufzeitversuch** =<sub>Def</sub> Entwickler spielt herum, um zu sehen, „ob Programm läuft“  
selbstverständliche Voraussetzung für Test - nicht mehr

**Test** =<sub>Def</sub> Ausführung eines Programmes mit dem Ziel, Fehler zu finden  
„Test“ wird von manchen auch anders verstanden: oft Missverständnisse

**Testvorschrift:** Anweisung, wie Test abzuwickeln ist (inkl. Setup)

**Testprotokoll:** standardisiertes Formular zur Doku. über jeden gefundenen Fehler

**Sonderfall Abnahme(test)** =<sub>Def</sub> Offiziell/juristischer Akt zur Feststellung, ob Software die an sie gestellten Anforderungen erfüllt.

- Kriterium: ausschließlich die Abnahmetestfälle
- Abnahme erfolgt, falls Abnahmetestfälle ohne Fehler durchlaufen
- Hier geht es also nicht vor allem darum, Fehler zu finden

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 138

### Analytische Qualitätssicherung

wie trägt Test zu besserer SW-Qualität bei?

- Analytische Maßnahme alleine verbessert die Qualität nicht
- Sie ist aber in einen Q-Prozess eingebunden, der es tut

**Höhere Qualität, besseres Vertrauen**

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 139

### Fehler in Tests

**Hintergrund**

**Abweichung von Soll und Ist bedeutet Fehler in**

falschem	<b>Ist</b>	(„falsche Ausgabe“)
oder falschem	<b>Soll</b>	(„falsche Vorgabe“)
oder ungeeignetem	<b>Vergleich</b>	(„falscher Test“)

Nicht jede Abweichung ist daher Indiz für **Programmfehler**  
obwohl man die eigentlich sucht!

**Andere Möglichkeiten**

- Testfall war falsch **Programm korrekt, Soll falsch aus Anf. ermittelt**
- Anforderung falsch interpretiert **Soll korrekt ermittelt, Anf. missverstanden**
- Vergleich entspricht nicht den Erfordernissen  
*Toleranz, Kriterien oder Typ falsch*
- Kombination mehrerer Gründe (das ist am übelsten)  
*Problem: woher weiß man, welcher Fall vorliegt?*

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 140

### „Testbarkeit“ eines Programms

„Ist das (Teil-) Programm so gestaltet, dass Testen einfach ist?“

- Leicht mit Eingaben und Parametern zu versorgen
- Ergebnisse leicht erkennbar
- Programmcode leicht verständlich (Stil und Struktur)
- Evtl. Zwischenergebnisse „abzweigbar“
- Nicht viele Außenbeziehungen
- Wenige Einflussgrößen vorhanden (nicht viele Kombinationen)
  - Diese sind alle gut kontrollierbar
  - Damit sind Ergebnisse reproduzierbar
- Ausführung des Prog. ist technisch und organisatorisch einfach
  - Zugänglich, nötige Hardware vorhanden
  - Billig (Rechenzeit, Bediener etc.)
  - Wenig Zusatzausrüstung/Vorlaufzeit nötig

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 141

## Tätigkeiten und Dokumente beim Testen

**Vorgehensweise**

- **Test planen**
  - **Resultat: Testplan**
    - Vorgehen: wer tut wann was?
    - Testgeschirr
- **Testfälle erstellen**
  - **Resultat: Testfallbeschreibung**
    - Setup, Voraussetzungen
    - Eingaben und Sollausgaben
- **Tests nach Plan durchführen**
  - **Resultat: Testberichte**
- **Gefundene Fehler zusammenfassen**
  - **Resultat: Fehlerberichte**

**Wichtig:**  
Tests dokumentieren

Resultate: Testplan bis Fehlerbehebungsberichte

Siehe auch: Ziser, Bittl, Grechening, Köhle (2001): Software Engineering mit UML und dem Unified Process. Pearson Studium  
Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16    142

## Testplanung

**Vorgehensweise**

- **Im Prinzip: Wer führt wann welche Testfälle durch**
  - **Testmethoden** (Black-/White-Box-Test, Regressionstest usw.)
  - **Konkrete Testfälle** (Eingabe, Soll-Ausgabe)
  - **Konfiguration der Testplattform** (auch Testdaten in Datenbank)
  - **Bereitstellung des Testgeschirrs** („Hilfsgerüst für die Testfälle“)
- **Kriterien für den Start der Testdurchführung**
  - **Version für den Test freigegeben**
- **Kriterien für Testunterbrechung**
  - **Normalerweise wird bei Fehlern weitergetestet**
  - **Kriterien, wann das nicht mehr sinnvoll ist**
- **Kommunikationswege**
  - **Wer erfährt in welcher Form von Testergebnissen?**
  - **Was tun diese Rollen daraufhin (selbständig); korrigieren sie?**

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16    143

## Testfälle erstellen

**Prinzip**

- **Black-Box-Test: Aus der Spezifikation**
  - zu allem, was in Spezifikation verlangt ist (*und nur dazu!*), muss geprüft werden, ob es wirklich so funktioniert
  - Spezifikation muss dazu die **Soll-Werte** vorgeben
    - Soll ist prinzipiell **nicht** aus dem Code ableitbar!
  - SW-Verhalten muss Spezifikation genügen
    - Was „intern“ passiert, ist uns egal
  - **Teststrategie**
    - Zu jeder Anforderung mindestens einen Testfall
    - Aber auch nicht viel mehr
    - Daher Testfälle geschickt wählen



Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16    144

## Testfälle praktisch ermitteln

1. Ausgangspunkt: Anforderung
  - Beispiel: „Der Eintrittspreis ist 3€ für Ermäßigte 2€“
2. Testfall: Anforderung anwenden, Sollwert ermitteln
  - Testfall 1: Knopf „Erwachsener“ drücken. Soll-Resultat: 3€
  - Testfall 2: Knopf „Ermäßigt“ drücken. Soll-Resultat: 2€
3. Mehr und kompliziertere Anforderungen: mehr Testfälle nötig
  - **Unklarheiten ausräumen:** quersum oder itQuer?
  - **Kombinationen von Eingabeparametern**
  - **Sonderfälle:** quersum (-12)? finCash (-1,28) ?
  - **Was soll bei Falscheingaben geschehen?**
    - (Vorher abgefangen, also unmöglich)
    - Warnung
    - Fehler
    - Raten, was gemeint war

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16    145

## Test durchführen

**Ablauf und Rollen**

Tester	Testleiter	Programmierer	Change Management
● Fehler melden	Fehler bewerten Fehler zuteilen	Fehler beheben	Korrektur ausliefern
Fehler nachtesten	Fehler abschließen		

Siehe auch: Ziser, Bittl, Grechening, Köhle (2001): Software Engineering mit UML und dem Unified Process. Pearson Studium  
Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16    146

## Vor- und Nachteile von Tests

**Prinzip**


**Vorteile**

- Reproduzierbar
- Mehrfach nutzbar
  - Rechenzeit ist billig
- Umgebung wird mitgeprüft
  - Bibliotheken
  - Virtuelle Maschine
- Systemverhalten veranschaulicht
- Falls automatisiert: schnell und billig

**Nachteile**

- Bedeutung „fehlerloser“ Tests wird überschätzt („schlecht gesucht“)
- Nicht alle Eigenschaften von Software sind testbar („Lesbarkeit“ ?)
- Nicht alle Situationen reproduzierbar
- Test zeigt Fehlerursache nicht
- Falls *nicht* automatisiert: Aufwändig, insbesondere Wiederholung

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16    147



## Testebenen

von unten nach oben testen!

**Systemtest**

- Funktioniert das ganze Gerät?
- Incl. Funktions- und Leistungstest
- Wiederholung der Black Box Tests
- Regressionstest
- Ressourcen, Antwortzeit, Last, Durchsatz

**Integrationstest:**

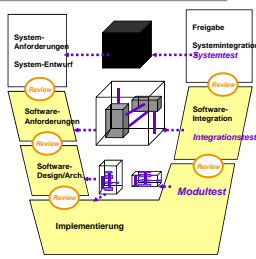
- Funktionieren Module zusammen?
- Incl. Installationstest
- Lässt sich SW installieren?

**Modul-/Unit Test:**

- Funktioniert ein Modul?


**Weitere Tests**

- Benutzbarkeit, Sicherheit, Interoperabilität, Neustart nach Absturz



Kurt Schneider
Leibniz Universität Hannover
SWT 2015/16 148

Softwaretechnik



## Kapitel 4

1. Wieso Software Engineering?
2. Vom Einzelkämpfer zum Großprojekt
- Systematische Softwareentwicklung**
3. Anforderungen und Test: Basis des Projekts
- 4. Entwurf: Strukturen und nicht-funktionale Eigenschaften**
5. Entwürfe notieren mit UML: Modelle im SE
6. Design Patterns: Entwurfserfahrungen nutzen
7. Management: Technik und Projektmanagement

Leibniz Universität Hannover
SWT 2015/16 149