Kontrollflußorientierte Testverfahren

Alexander Steinhöfer, Christian Grafe WS0607 / VInf5

Gliederung

1. **Definition und Theorie**
2. **Kontrollflussgraphen**
3. **Arten**
    1. **Anweisungsüberdeckungstest (C_0)**
    2. **Zweigüberdeckungstest (C_1)**
    3. **Pfadüberdeckungstest (C_2)**
    4. **Bedingungsüberdeckungstest (C_3)**
4. **Vergleich / Fazit**
5. **Diagramme und Daten zum Übungsprojekt „Onlineshop Amazan.de"**

1. Definition und Theorie

Die kontrollflussorientierten Testtechniken gehören zu der Gruppe der strukturorientierten Testmethoden, daher besitzen diese deren Stärken und Schwächen.
Kontrollflussorientierte Testmethoden geben, wie alle strukturorientierten Testmethoden, keine Regel für die Erzeugung von Testfällen an. Aus diesem Grund ist es zwingend notwendig auf eine entsprechende Abdeckung der Struktur zu achten.
Kontrollflussorientierte Testmethoden betrachten die Struktur des Quellcodes.

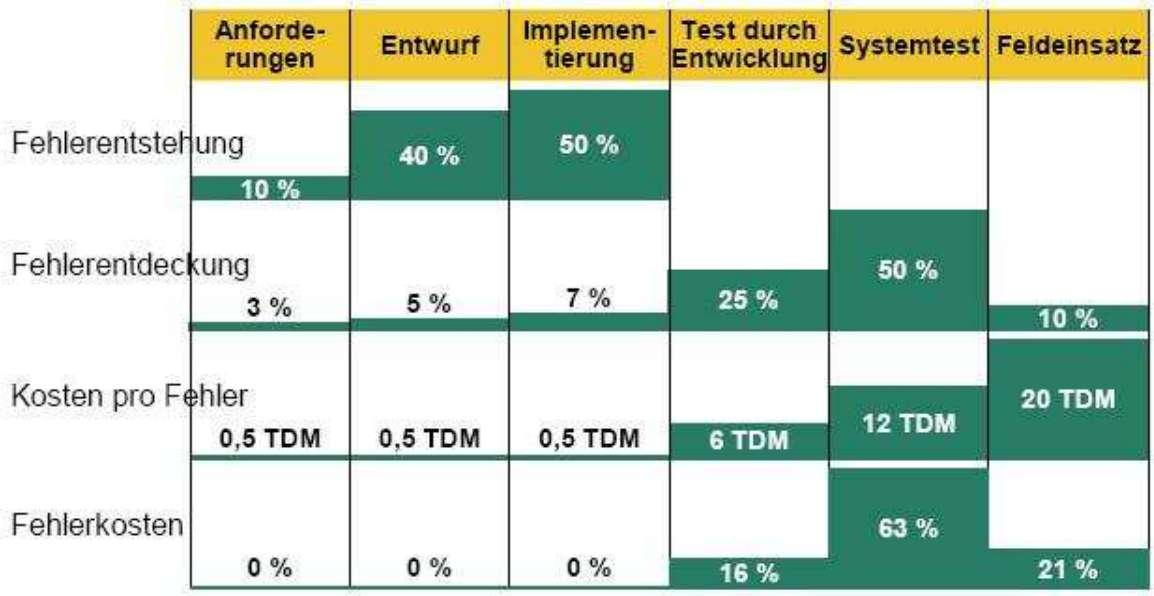
Daher handelt es sich bei diesen Tests um sogenannte White-Box-Testverfahren.

Bei White-Box-Tests handelt es sich um verschiedene Methoden von Software-Tests, bei der die Tests mit Kenntnissen über die innere Funktionsweise des zu testenden Systems entwickelt werden.

Im Gegensatz zum Black-Box-Test ist für diesen Test also ein Blick in den Quellcode gestattet, d.h. es wird direkt am Code geprüft wo mögliche Fehler liegen.

Bei allen gleich vorgestellten Testarten wird der Quellcode benötigt und zu überprüfen ob die Spezifikation eingehalten wurde.

<u>Warum man überhaupt Testen sollte, zeigt diese Grafik überdeutlich anhand der Kostenersparnis:</u>



2. Kontrollflussgraphen

Programmablaufplan (PAP) ist ein Ablaufdiagramm für ein Computerprogramm, das auch als Flussdiagramm (engl. flowchart) oder Programmstrukturplan bezeichnet wird. Die Symbole für Programmablaufpläne sind in der DIN 66001 genormt. Dort werden auch Symbole für Datenflusspläne definiert. Flussdiagramme werden oft unabhängig von Computerprogrammen zur Darstellung von Prozessen und Tätigkeiten eingesetzt. Im Bereich der Softwareerstellung werden sie nur noch selten verwendet: Programmcode moderner Programmiersprachen bietet ähnlichen Abstraktionsgrad, ist jedoch einfacher zu erstellen und in der Regel sehr viel einfacher zu verändern als ein Ablaufdiagramm.

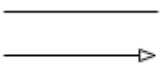
Das Konzept der Programmablaufpläne stammt ebenso wie das Nassi-Shneiderman-Diagramm (Struktogramm) aus den 1960er-Jahren und bildet einen linearen Programmfluß ab. Für die Abbildung Objektorientierter Programmkonzepte haben sich beide Konzepte als ungeeignet erwiesen. Als Konsequenz wurde die Unified Modelling Language (UML) für Objektorientierte Programmkonzepte entwickelt.

Elemente

Hauptsächlich werden die folgenden Elemente verwendet:

- Oval: Start, Ende, weitere Grenzpunkte

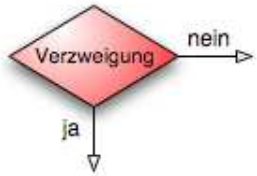
- Pfeil, Linie: Verbindung zum nächstfolgenden Element



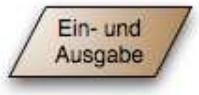
- Rechteck: Operation

- Rechteck mit doppelten, vertikalen Linien: Unterprogramm

- Raute: Verzweigung

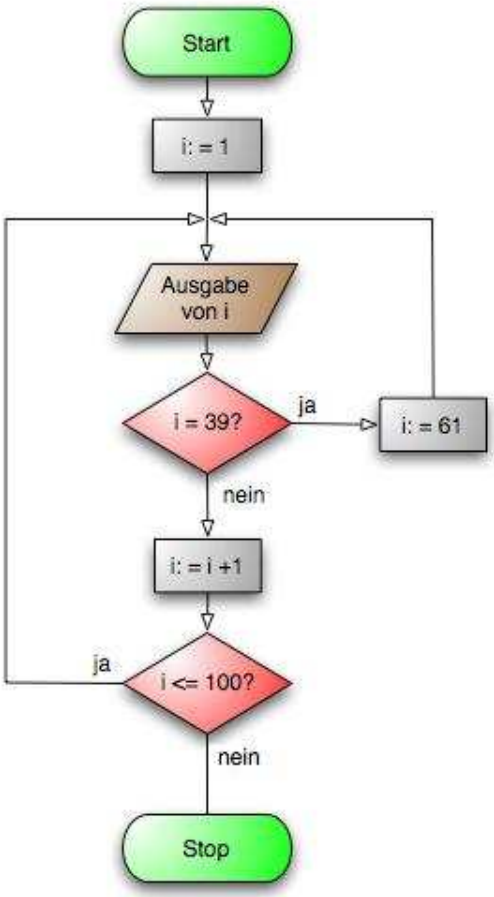


- Parallelogramm: Ein- und Ausgabe

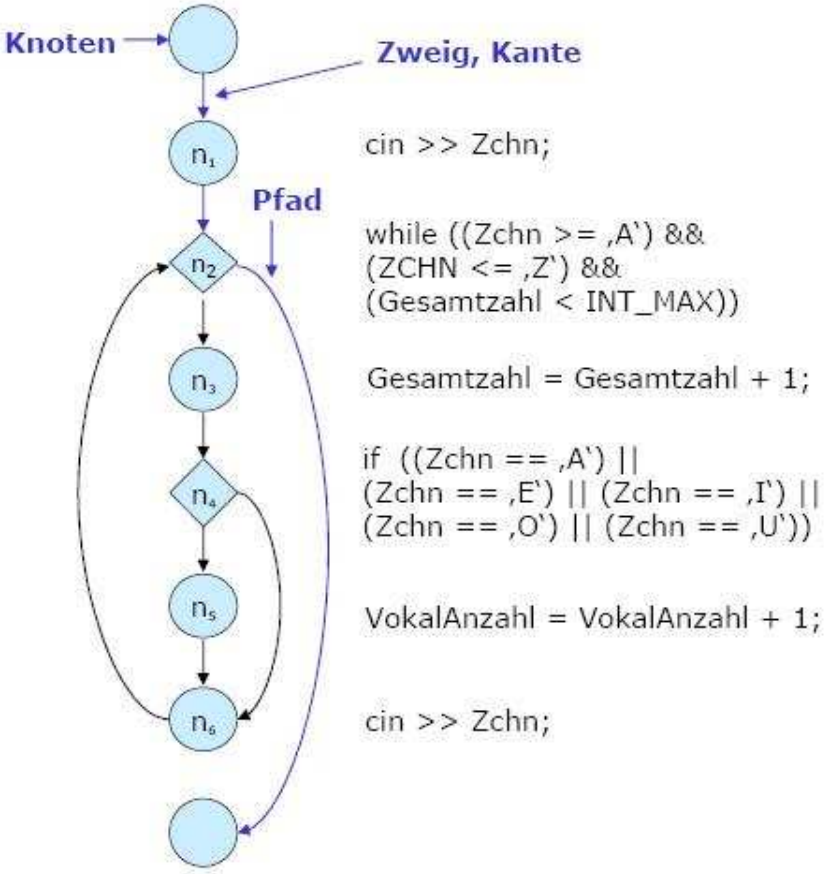


Beispiele:

Die untenstehende Abbildung zeigt eine einfache Zählschleife von 1 bis 101. Die Zählvariable i wird vor Beginn der Schleife auf ihren Startwert i=1 gesetzt. Danach wird die erste Anweisung der Schleife, das Ausgeben der Variable i, ausgeführt. Die nachfolgende zweite Anweisung ist eine einseitige Auswahl, die prüft ob i den Wert 39 besitzt. Falls dies der Fall ist, wird i auf den Wert 61 gesetzt und die Schleife beginnt mit dem nächsten Durchlauf. Falls i nicht 39 ist, wird i in der nachfolgenden Anweisung um eins erhöht und anschließend geprüft ob die Schleifenabbruchbedingung i>100 erreicht ist. Falls nicht erfolgt ein nochmaliger Schleifendurchlauf.



Weiteres Beispiel:



Arten

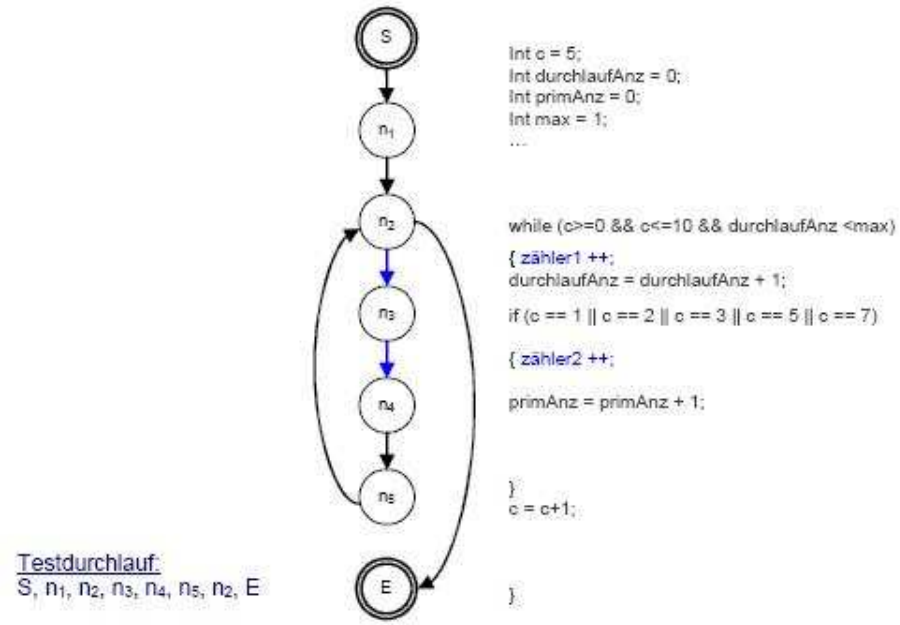
Auf den folgenden Seiten finden sie eine Übersicht über die unterschiedlichen Arten von Überdeckungstests.

Hinweis:
Das C in den Namen der einzelnen Tests steht für "Coverage" was so viel heißt wie die Abdeckung oder die Gesamtheit der ausgewerteten Informationen.

C0. Anweisungsüberdeckungstest (Statement Coverage)

Allgemein

Anweisungsüberdeckungstests, auch C_0-Test genannt, sind die am einfachsten anwendbaren kontrollflussorientierten Testmethoden.
Mit den Anweisungsüberdeckungstests wird sichergestellt, dass kein "toter Code", Anweisungen die niemals durchlaufen werden, im Programm existiert. Dies ist ein notwendiges Kriterium um sicherzugehen, dass jede Anweisung auf Fehler untersucht wird.
Der Sinn des Anweisungsüberdeckungstests ist die mindestens einmalige Ausführung aller Anweisungen in einem Programm.
Ist dies gewährleistet spricht man von einer völligen Anweisungsüberdeckung. Wie in der unten stehenden Abbildung zu erkennen ist, werden alle Anweisungen
mindestens einmal ausgeführt, wenn die While-Schleife einmal durchlaufen wird. Unser Testpfad enthält zwar alle Knoten, aber nicht alle Kanten.
Die Kante (n3,n5) wird im optionalem Else-Teil nicht ausgeführt. Genau dieser Fall wird im Zweigüberdeckungstest betrachtet.
Durch die Einführung der Zähler kann beim automatisierten Testen kontrolliert werden, ob jede Anweisung ausgeführt wurde, indem man sich die
Werte nachdem Testdurchlauf ausgeben lässt.
Anweisungsüberdeckungstests werden selten als Haupttestwerkzeug in einem Vollständigkeitstest eingesetzt, denn dafür sind sie i. d. R. zu schwach.



Beispiel: Anweisungsüberdeckung

Metrik

Der Anweisungsüberdeckungsgrad bestimmt sich wie folgt:

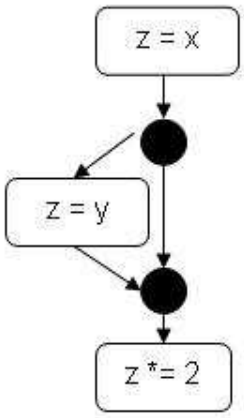
$$c_{Anweisung} = \frac{\text{Anzahl der ausgeführten Anweisungen}}{\text{Gesamtanzahl der Anweisungen}}$$

Beispiel

Gegeben ist folgender Quellcode:

```
/* z wird das doppelte des größeren Werts von x oder y zugewiesen */
int z = x;
if (y > x)
  z = y;
z *= 2;
```

Darstellung als Kontrollflußgraph:



In diesem Fall genügt ein einziger Testfall um eine 100%ige Anweisungsüberdeckung zu erreichen:

z.B**. x = 0, y = 2**

Es sind beliebig viele passende Testfälle denkbar, solange y größer als x ist

(ansonsten wird `z = y` nicht ausgeführt, und die Anweisungüberdeckung ist <100%).

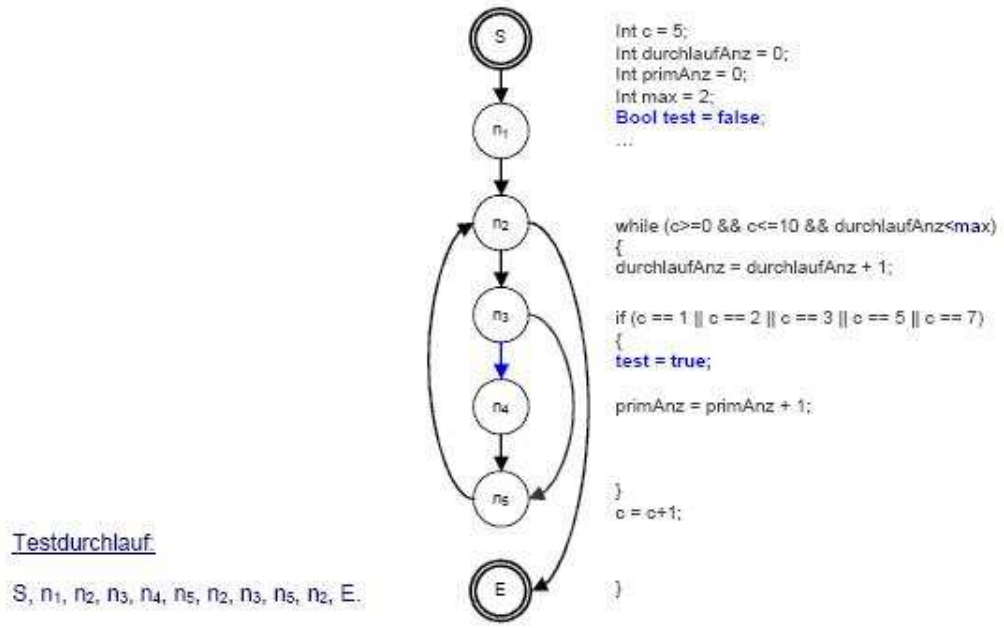
C1. Zweigüberdeckungstest (Statement / Branch Coverage)

Der Zweigüberdeckungstest (C_1–Test) umfasst den Anweisungsüberdeckungstest vollständig.
Für den C_1–Test müssen strengere Kriterien erfüllt werden als beim Anweisungsüberdeckungstest.
Im Bereich des kontrollflussorientierten Testens wird der Zweigüberdeckungstest als Minimalkriterium angewendet.
Mit Hilfe des Zweigüberdeckungstests lassen sich nicht ausführbare Programmzweige aufspüren
Anhand dessen kann man dann Softwareteile, die oft durchlaufen werden gezielt optimieren.

Analog zum Anweisungsüberdeckungstest wird, um die Codeabdeckung messbar zu machen, der Code in unten stehender Abbildung, durch eine boolesche Hilfsvariable test instrumentiert.

Im Gegensatz zum Anweisungsüberdeckungstest durchläuft der Zweigüberdeckungstest alle Zweige.
Der Zweigüberdeckungstest wird auch
Entscheidungsüberdeckungstest genannt, da die Hilfsvariable mindestens einmal mit dem Wert true und false durchlaufen werden muss.
In diesem Fall muss die While-Schleife mindestens zweimal durchlaufen werden. Mit dem Durchlaufen der Zweige wird auch sichergestellt, dass
jeder Knoten (Anweisung) mindestens einmal ausgeführt wird.
Somit wird auch das Kriterium für den Anweisungsüberdeckungstest erfüllt.

Daher subsumiert der Zweigüberdeckungstest den Anweisungsüberdeckungstest.



Beispiel: Zweigüberdeckung

Schwierig ist es für den Zweigüberdeckungstest Testfälle zu generieren, wo Betriebssystemzustände oder Dateikonstellationen
getestet werden müssen. Weiterhin ist diese Technik des Testens zum Testen von 'Schleifen' und zusammengesetzter
Entscheidungen nicht geeignet, da weder Kombinationen von Zweigen, noch kompliziert aufgebaute Entscheidungen in Betracht
gezogen werden können. Hierfür müssen Erweiterungen herangezogen werden.

Weitaus problematischer erweist sich das Zweigüberdeckungsmaß. In dem Fall, dass alle Knoten gleich bewertet sind, verzichtet man auf die

Betrachtung der Abhängigkeiten untereinander. Dadurch entsteht kein linearer Zusammenhang zwischen der erreichten Überdeckungsrate

und dem Verhältnis zwischen der Anzahl der dazu benötigten Testfälle und der eigentlichen Anzahl der Testfälle, die für die 100 prozentige

Zweigüberdeckung notwendig sind. Um den Zweigüberdeckungstest zu verbessern, wird ein Zweig, der abhängig von einem anderen Zweig ist,

nicht weiter berücksichtigt. Die Zweige, die nicht abhängig sind, werden als primitiv bezeichnet.

Daher ergibt sich für das Überdeckungsmaß:

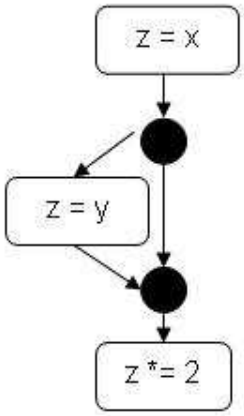
$$c_{primitiv} = \frac{\text{Anzahl der ausgeführten primitiven Zweige}}{\text{Anzahl aller primitiven Zweige}}$$

Beispiel

Gegeben ist folgender Quellcode:

```
/* z wird das doppelte des größeren Werts von x oder y zugewiesen */

int z = x;

if (y > x)

  z = y;

z *= 2;
```

Darstellung als Kontrollflußgraph:



Im Gegensatz zum Anweisungsüberdeckungstest sind nun mehr als ein Testfall notwendig, um eine 100%ige Zweigüberdeckung zu erreichen,

da sowohl der Fall für den durchlaufenen If-Zweig, als auch der Fall für den nicht-durchlaufenen If-Zweig überprüft werden muss:

Testfall 1: x = 0, y = 2

Testfall 2: x = 2, y = 0

Wie auch im Anweisungsüberdeckungstest sind verschiedene Testfälle möglich, die das geforderte Kriterium erfüllen.

Nach Ausführung stellt sich heraus, dass das Ergebnis bei beiden Testfällen der Spezifikation entspricht und der Test somit bestanden ist.

C2. Pfadüberdeckungstest

Pfadüberdeckungstest (engl. Path Coverage), stellenweise auch C2-Test genannt, werden im Kontrollflussgraphen die möglichen Pfade vom Startknoten bis zum Endknoten betrachtet.

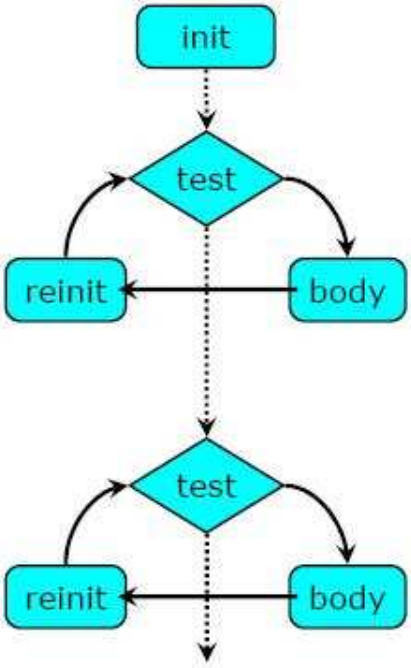
Übersicht

- C2a - vollständiger Pfadüberdeckungstest
- C2b - Boundary-Interior Pfadüberdeckungstest
- C2c - Strukturierter Pfadüberdeckungstest
- Vorteil
- Nachteil

C2a - vollständiger Pfadüberdeckungstest

Es werden alle möglichen Pfade getestet. Problem: Bei Programmen mit Schleifen sind das unendlich viele Pfade.

Problem des Wachstums der Anzahl der Pfade



(1) Zahl der Testfälle von while-Schleifen ist nicht vorab bekannt oder nicht beschränkt

(2) Zahl der Testfälle konsekutiver Schleifen ist multiplikativ

(3) Schleife mit Verzweigung im Körper: Ist N Schranke für Zahl der Schleifendurchläufe, so sind im worst case 2^N Testfälle erforderlich (exponentielles Wachstum)

C2b - Boundary-Interior Pfadüberdeckungstest

Im Prinzip wie der C2a-Test, nur dass nun die Schleifendurchläufe auf <=2 reduziert werden.

Für jede Schleife gibt es 2 Gruppen von Pfaden:

<u>Boundary-Test:</u>

- Jede Schleife wird keinmal betreten.
- Jede Schleife wird genau einmal betreten und alle Pfade in dem Schleifenkörper werden einmal abgearbeitet.

Interior-Test:

- Das Schleifeninnere gilt als getestet, wenn alle Pfade, die bei zweimaligem Durchlaufen möglich sind, abgearbeitet wurden.

C2c - Strukturierter Pfadüberdeckungstest

Im Prinzip wie der C2b-Test, nur dass nun die Anzahl der Schleifendurchläufe auf eine vorgegebene natürliche Zahl n reduziert wird.

Vorteil

- Hohe Fehlererkennungsrate

Nachteil

- nicht ansprechbare Pfade auf Grund von Bedingungen

C3. Bedingungsüberdeckungstest

Der Bedingungsüberdeckungstest (engl. Condition Coverage) gehört zu einer Gruppe von Überdeckungstests, die zum Testen von Software verwendet werden.

Das Problem der bisherigen Überdeckungstests (C1-Test, C2-Test) ist, dass zusammengesetzte, hierarchische Bedingungen nicht ausreichend getestet werden.

Übersicht

C3a - Einfachbedingungsüberdeckungstest

Jede atomare Bedingung einer Bedingung muss einmal mit true und einmal mit false getestet werden. Beispiel:

```
boolean a,b;
if(a || b) {...}
```
Testfall 1 wäre a=false und b=false. Testfall 2 wäre a=true und b=true.

C3b - Mehrfachbedingungsüberdeckungstest

Dieser Test betrachtet alle atomaren Bedingungen einer Bedingung. Wenn n atomare Bedingungen in der Bedingung stehen, dann werden 2^n Kombinationen gebildet. Das heißt für das obige Beispiel, dass 4 Testfälle gebildet werden.

C3c - minimaler Mehrfachbedingungsüberdeckungstest

Diese Version erstellt mehr Testfälle als C3a und weniger als C3b, indem jede Bedingung (atomar und zusammengestellt) zu true und zu false evaluiert wird. Die logische Struktur wird hierbei berücksichtigt und der C1-Test (Zweigüberdeckungstest) ist vollständig in diesem Test enthalten. Ein weiterer Punkt ist, dass der C3c-Test berechenbar ist.

Nachteil

- Unvollständige Auswertung einer Bedingung durch eine Programmiersprache mit sog. short circuit evaluation wie z. B. C/C++, Java, C#.

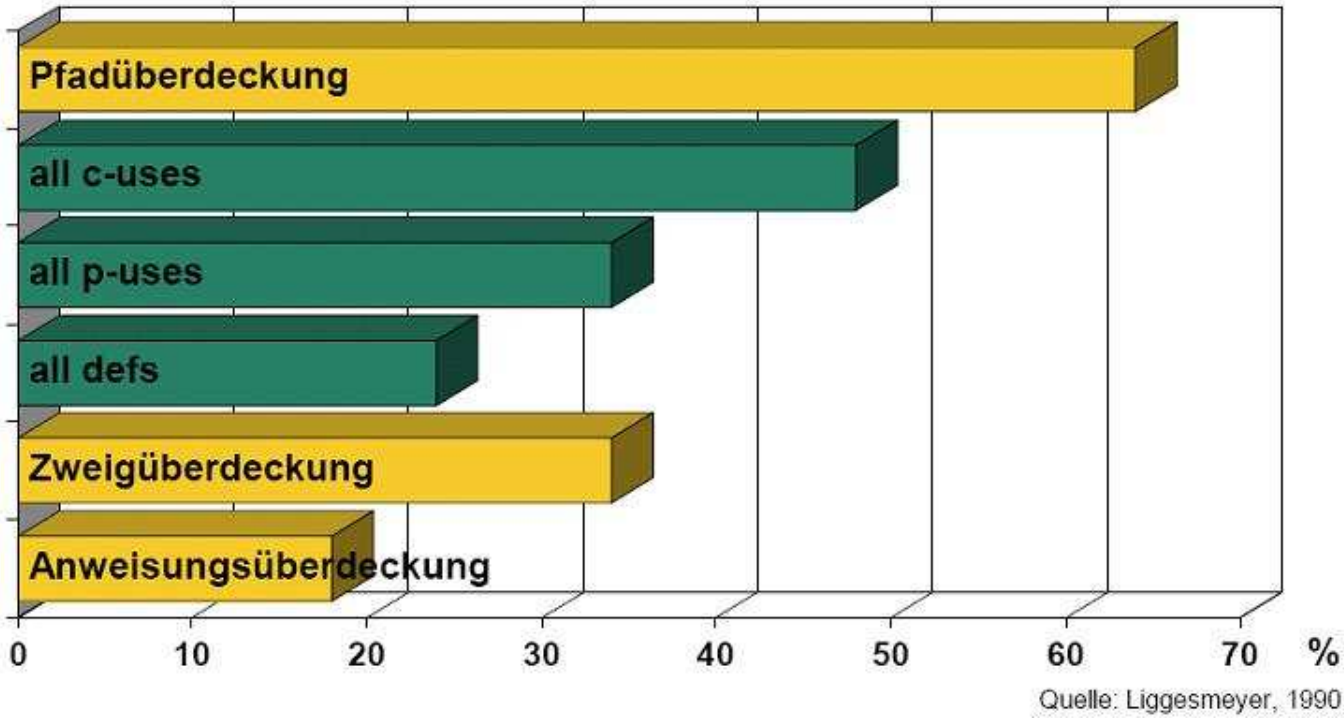
Beispiel:

```
if (a && b)  {...} else { lese b aus }
```
Wenn a false ist, dann ist die Belegung der Variable b egal.
z. B. a=false und b=null, dann passiert ein Fehler im else-Zweig

Zusammenfassung

	Kurz-name	**erfüllte Bediungung**	**Durchführ-barkeit**
Anweisungsüberdeckungstest	C0	jede Anweisung wird mindestens einmal ausgeführt	relativ einfach
Zweigüberdeckungstest	C1	jede Kante im KFG wird mindestens einmal durchlaufen	realistische Mindestanforderung, vertretbarer Aufwand
Pfadüberdeckungstest	C2		
Vollständig	C2a	Alle möglichen Pfade werden durchlaufen	unmöglich bei Schleifen
Boundary-Interior	C2b	wie C2a, Schleifen werden jedoch nach speziellen Regeln durchlaufen	aufwändig
Strukturiert	C2c	wie C2b, Schleifen werden jedoch genau *n-mal* durchlaufen	aufwändig
Bedingungsüberdeckungstest	C3		
Einfachbedingung	C3a	jede atomare Bedingung wird einmal mit true und false getestet	aufwändig
Mehrfachbedingung	C3b	jede true/false Kombination der atomaren Bedingungen wird getestet	sehr hoher Aufwand
Minimale Mehrfachbedinung	C3c	jede atomare Bedingung und die Gesamtbedingung wird mit true und false getestet	hoher Aufwand

Vergleich bzw. Fazit



Die Qualität eines Tests hängt entscheidend vom gewählten Test ab: Wurde nur nach C0 mit Überdeckungsgrad 100% getestet so ist dies trotzdem kein verlässlicher Indikator für eine fehlerfreie Software.

Wurde hingegen mit C2a auf 100% getestet würde dies ein gutes Kriterium für eine fehlerfreie bzw. -arme Software darstellen. Leider ist dieser Test in der Praxis kaum durchzuführen.

Die zweite wichtige Größe ist natürlich der Überdeckungsgrad. Dieser ist aber nur bei Verwendung des gleichen Tests untereinander vergleichbar.

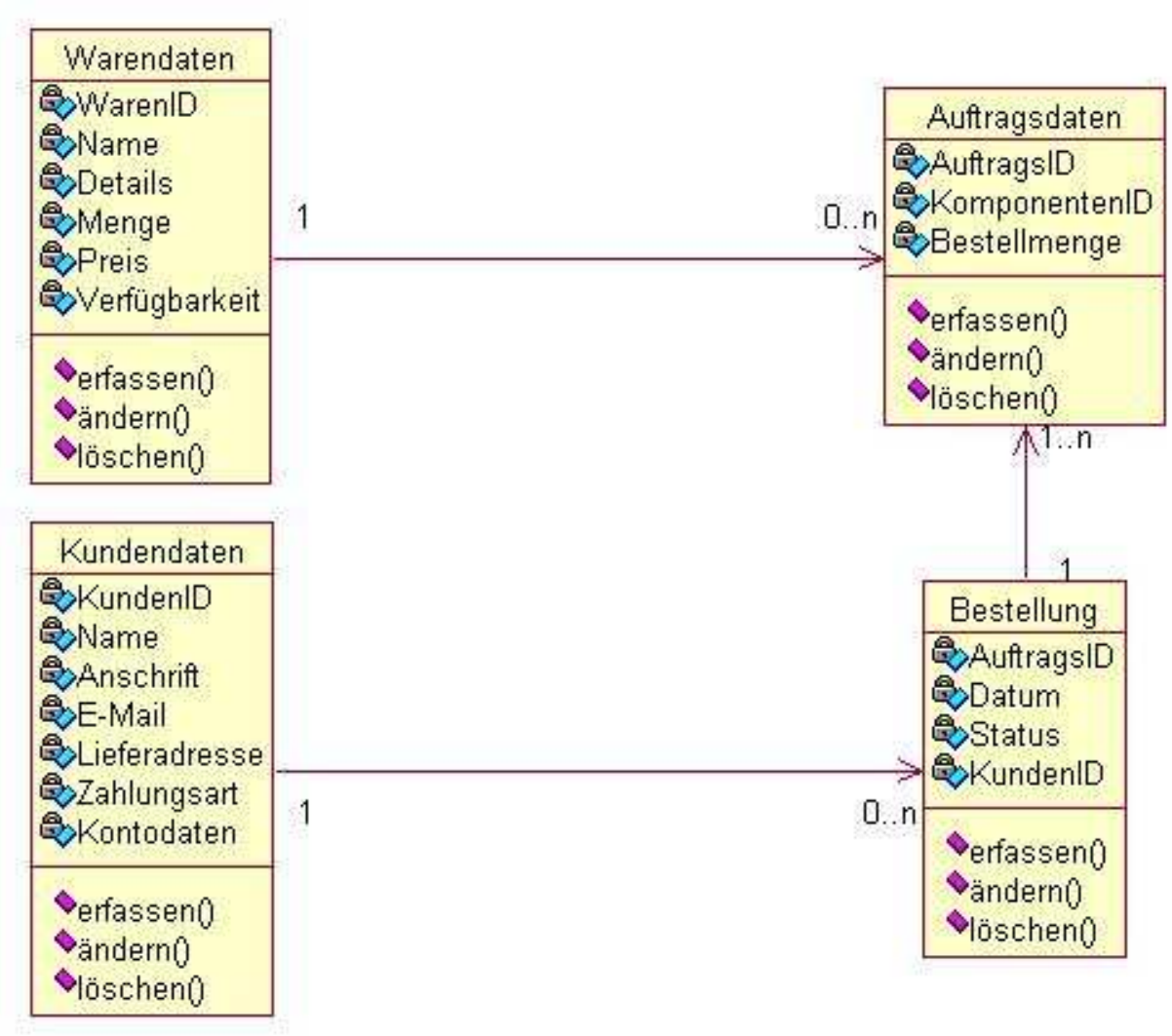
Bei einem hohen Überdeckungsgrad werden mehr Fehler gefunden als bei einem niedrigen.

Diagramme und Daten zum Übungsprojekt „Onlineshop Amazan.de“

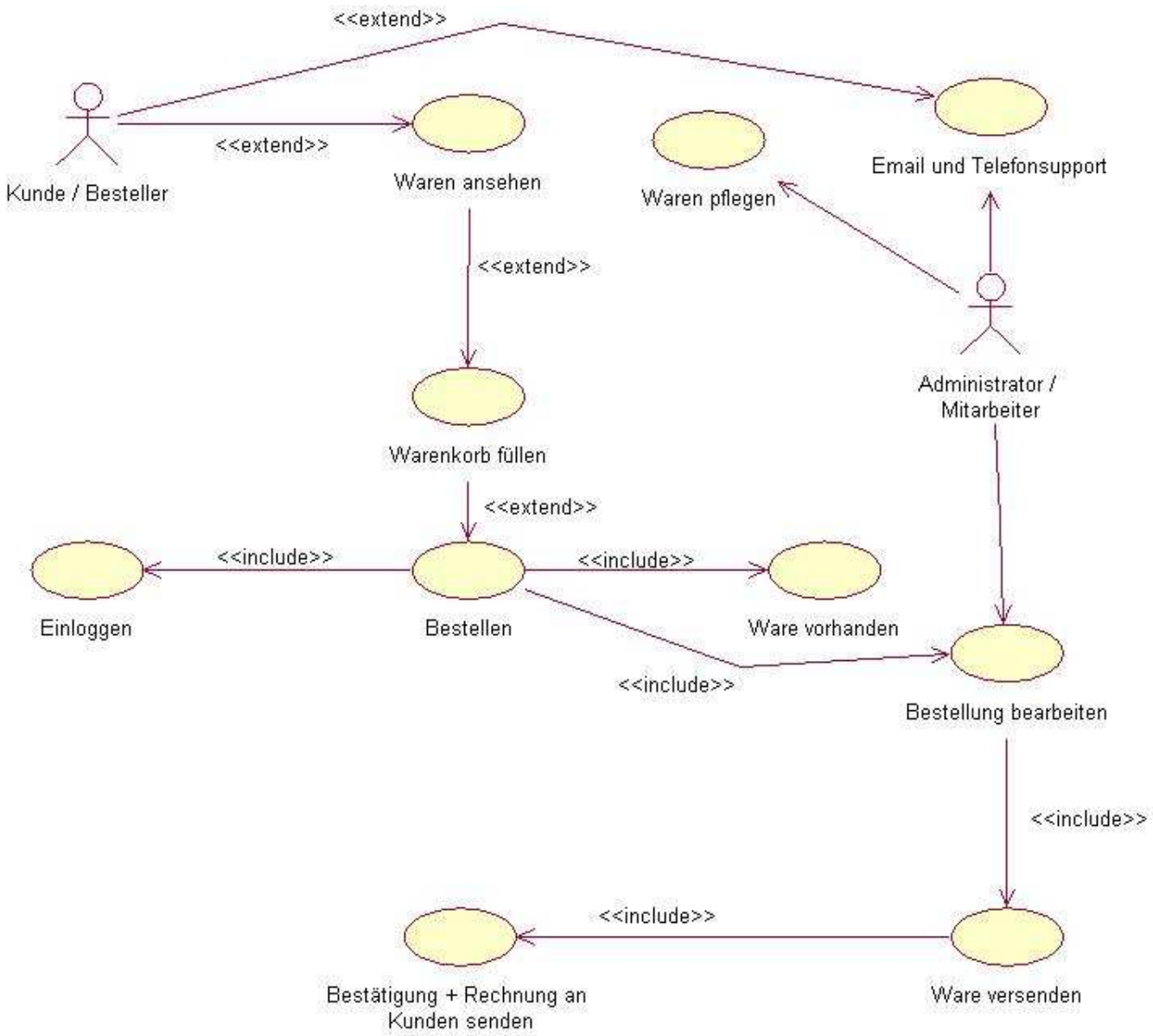
Gliederung

- **Klassendiagramm**
- **Use Case Diagramm**
- **State Activity Diagramm für den Vorgang „Bestellen“**
- **Sequenzdiagramm**
- **Paketdiagramm**
- **Ausarbeitung des Shops in PHP**
- **Aus Klassendiagramm generierte PHP Klassen**
- **Function Point Verfahren**

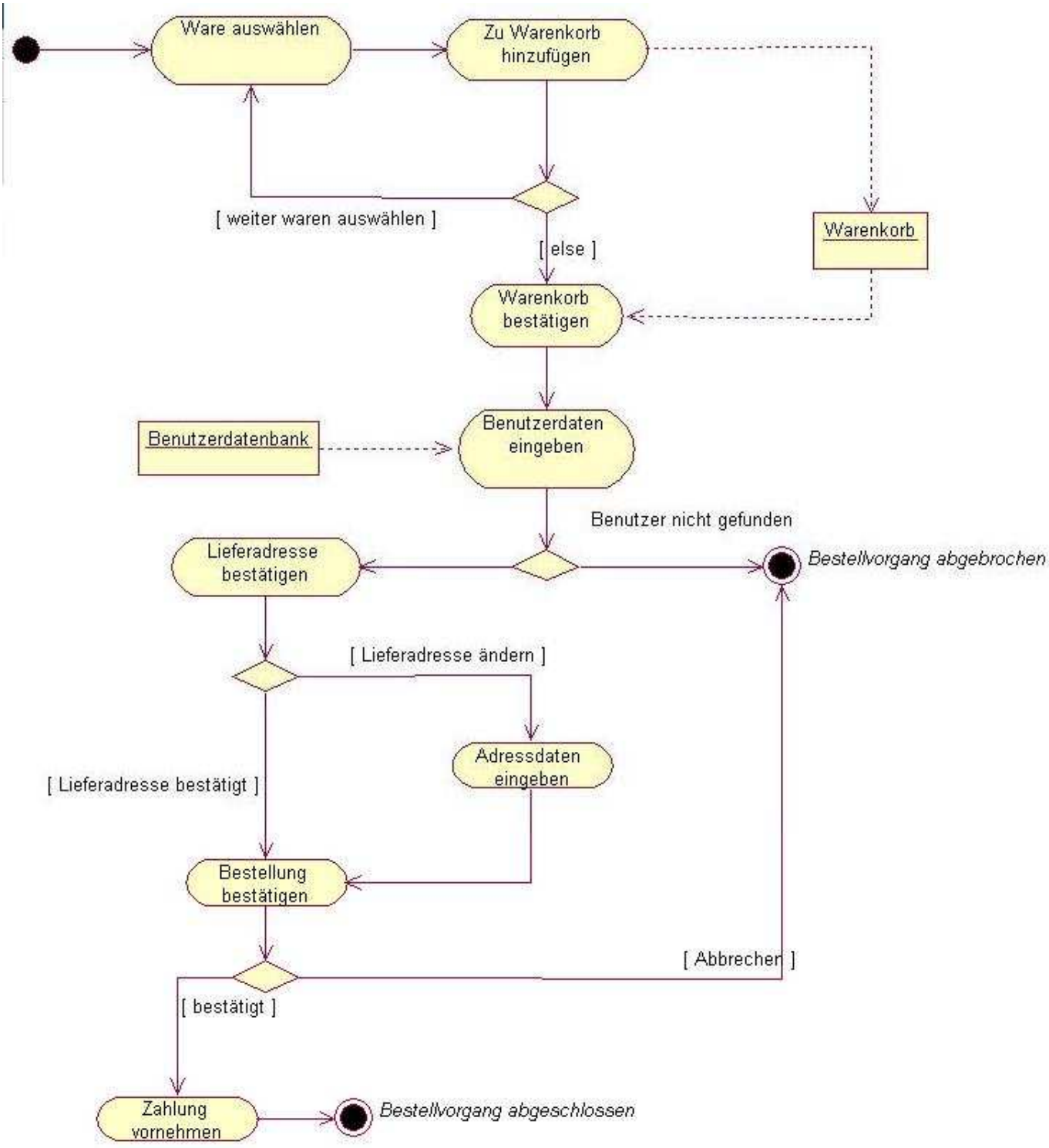
Klassendiagramm



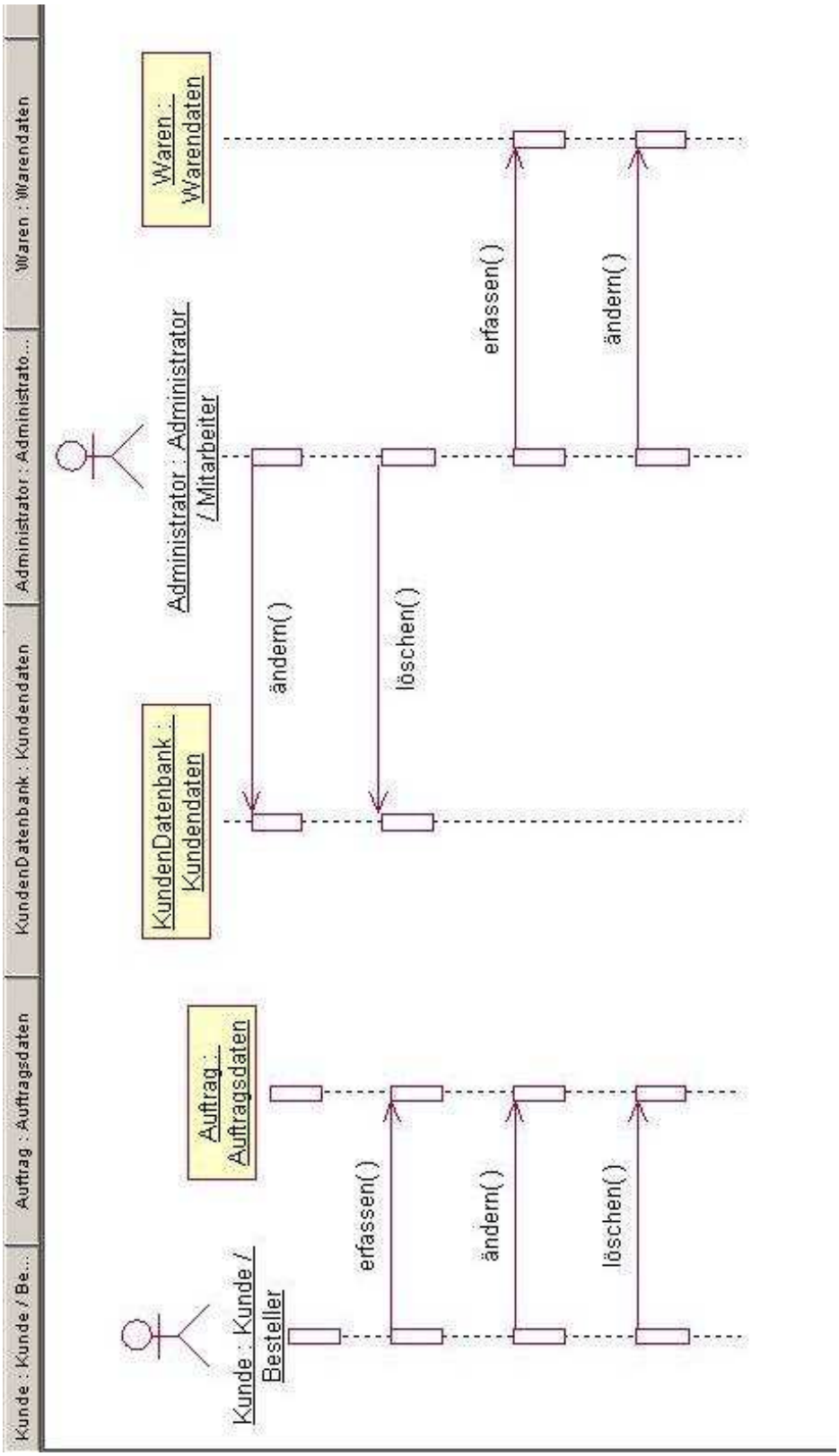
Use Case Diagramm



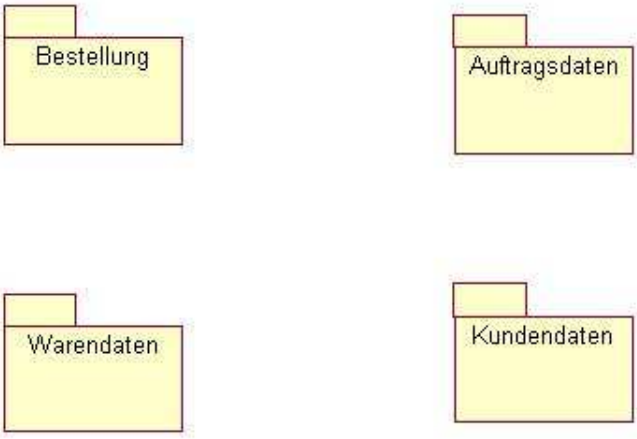
State Activity Diagramm für den Vorgang „Bestellen“



Sequenzdiagramm

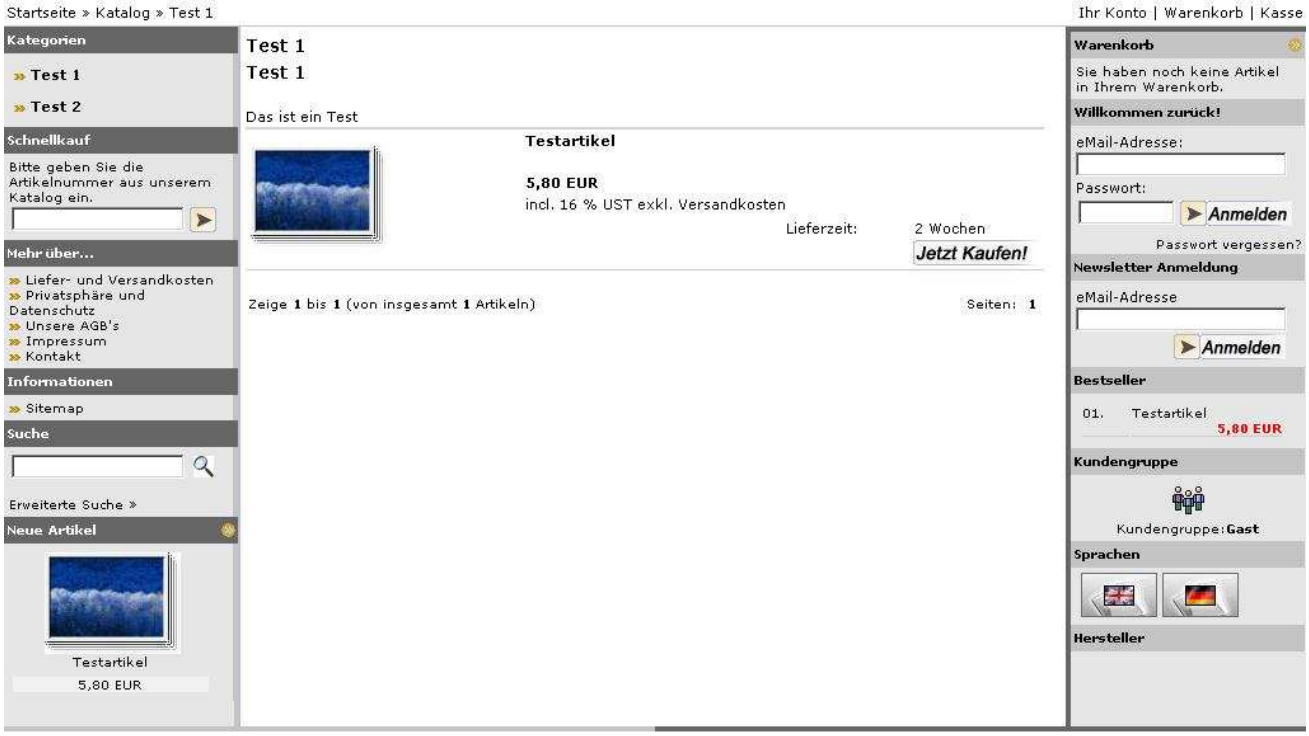


Paketdiagramm



Ausarbeitung des Shops in PHP

Oberfläche die auf dem Apache Webserver lief

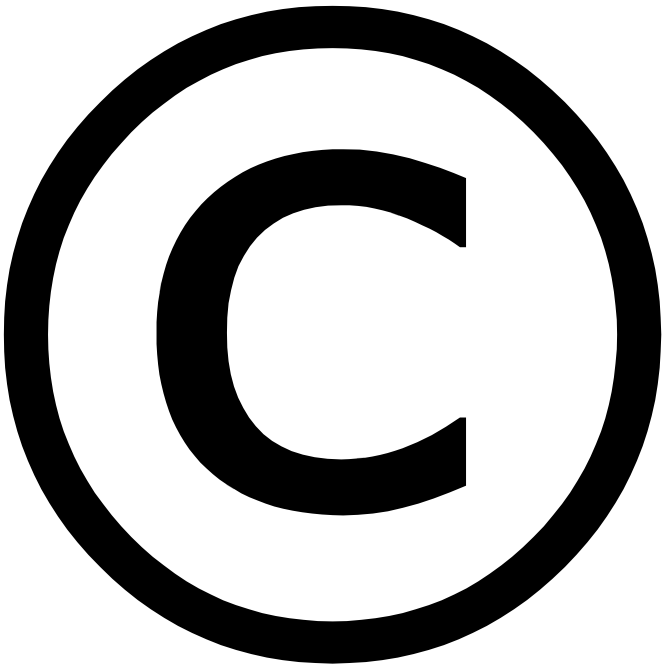


Aus Klassendiagramm generierte PHP Klassen

```
class Warendaten {
	var $WarenID;
	var $Name;
	var $Details;
	var $Menge;
	var $Preis;
	var $Verfuegbarkeit;

	public function erfassen()
	{

	}

	public function aendern()
	{

	}

	public function loeschen()
	{

	}
}


class Kundedaten {
	var $KundenID;
	var $Name;
	var $Anschrift;
	var $E_Mail;
	var $Lieferadresse;
	var $Zahlungsart;
	var $Kontodaten;

	public function erfassen()
	{

	}

	public function aendern()
	{

	}

	public function loeschen()
	{

	}
}
class Auftragsdaten {
	var $AuftragsID;
	var $KomponentenID
```

```
      var $Bestellmenge;

      public function erfassen()
      {

      }
      public function aendern()
      {

      }
      public function loeschen()
      {

      }
}

class Bestellung {
      var $AuftragsID;
      var $Datum;
      var $Status;
      var $KundenID;

      public function erfassen()
      {

      }

      public function aendern()
      {

      }

      public function loeschen()
      {

      }
}
```

Function Point - Methode								
			Tabelle:					
Kategorie	**Anzahl**	**Klassifizierung**	**Gewichtung**	**Zeilensumme**				
Eingabedaten	1	Einfach	3	3				
		Mittel	4	0				
	3	Komplex	6	18				
Abfragen	1	Einfach	3	3				
	2	Mittel	4	8				
	1	Komplex	6	6				
Ausgaben	1	Einfach	4	4				
	3	Mittel	5	15				
	1	Komplex	7	7				
Datenbestände	2	Einfach	7	14				
	1	Mittel	10	10				
		Komplex	15	0				
Referenzdaten	1	Einfach	5	5				
	1	Mittel	7	7				
		Komplex	10	0				
Summe			E1	100		Unbewertete Gesamtpunkte (100%)		
Einflußfaktoren		1 Verflechtung mit anderen Anwendungssystemen (0-5)		4				
(ändern den Function Point-Wert um +/- 30%)		2 Dezentrale Daten, dezentrale Verarbeitung (0-5)		0				
		3 Transaktionsrate (0-5)		5				
		4 Verarbeitungslogik						
		A Rechenoperationen (0-10)		3				
		B Kontrollverfahren (0-5)		4				
		C Ausnahmeregelungen (0-10)		2				
		D Logik (0-5)		2				
		5 Wiederverwendbarkeit (0-5)		3				
		6 Datenbestandskonvertierungen (0-5)		1				
		7 Anpaßbarkeit (0-5)		2				
Summe der 7 Einflüsse			E2	26				
Faktor Einflußbewertung = (E2/100) + 0,7			E3	0,96				
Bewertete Function Points: E1 * E3				96		Bewertete Gesamtpunkte (70% - 130%)		
Aufwand nach IBM-Tabelle (interpoliert)			E4	6	MM			
Optimale Entwicklungsdauer (2.5 * MM^s, mit s=0.35 (Online-Systeme))			E5	4,680507509	Monate			
Durchschnittliche Größe des Mitarbeiterteams (E4/E5)				1,281912269				



Christian Grafe
Alexander Steinhöfer

FH Hof
Wintersemester 06 / 07