Software Testen Grundlagen

180.764 Software-Qualitätssicherung

Markus Zoffi
peso@inso.tuwien.ac.at

Research Group for Industrial Software (INSO)
https://www.inso.tuwien.ac.at

Inhalt

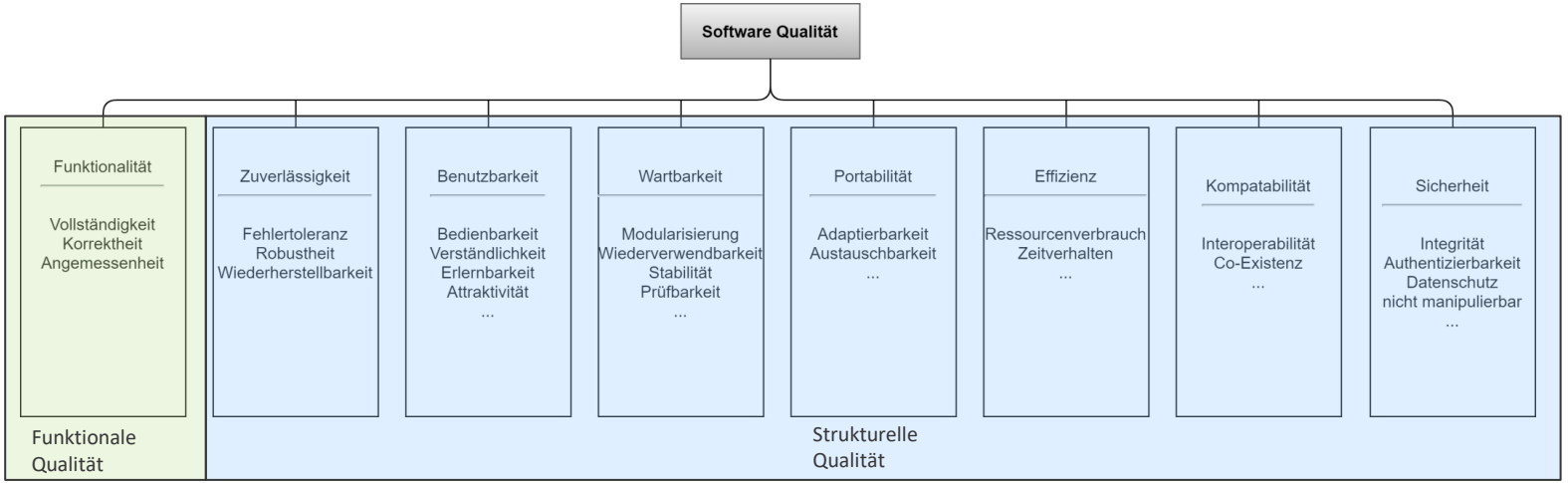
Software Testen Grundlagen

Testentwurfsverfahren

- *Black Box Methoden*
- *White Box Methoden*
- *Erfahrungsbasiertes Testen*

Recap: Definition – Software Qualität

- Was ist Software Qualität?
- Viele unterschiedliche Definitionen
- ISO/IEC 25010 beschreibt Software Qualität als ein Set an Qualitätsfaktoren:



Definition

- Motivation
  - Testen ist das Ausführen eines Programms, mit der Absicht, Fehler zu finden

- Verifikation & Validierung (V&V)
  - Überprüft, ob ein Software System der Spezifikation entspricht und den vorgesehenen Zweck erfüllt
    - Verifikation
      - „Bauen wir das Produkt richtig?“
      - „Entspricht das Produkt der Spezifikation?“
    - Validierung
      - „Bauen wir das richtige Produkt?“
      - „Ist das Produkt für seinen Einsatzzweck geeignet?

Ziele

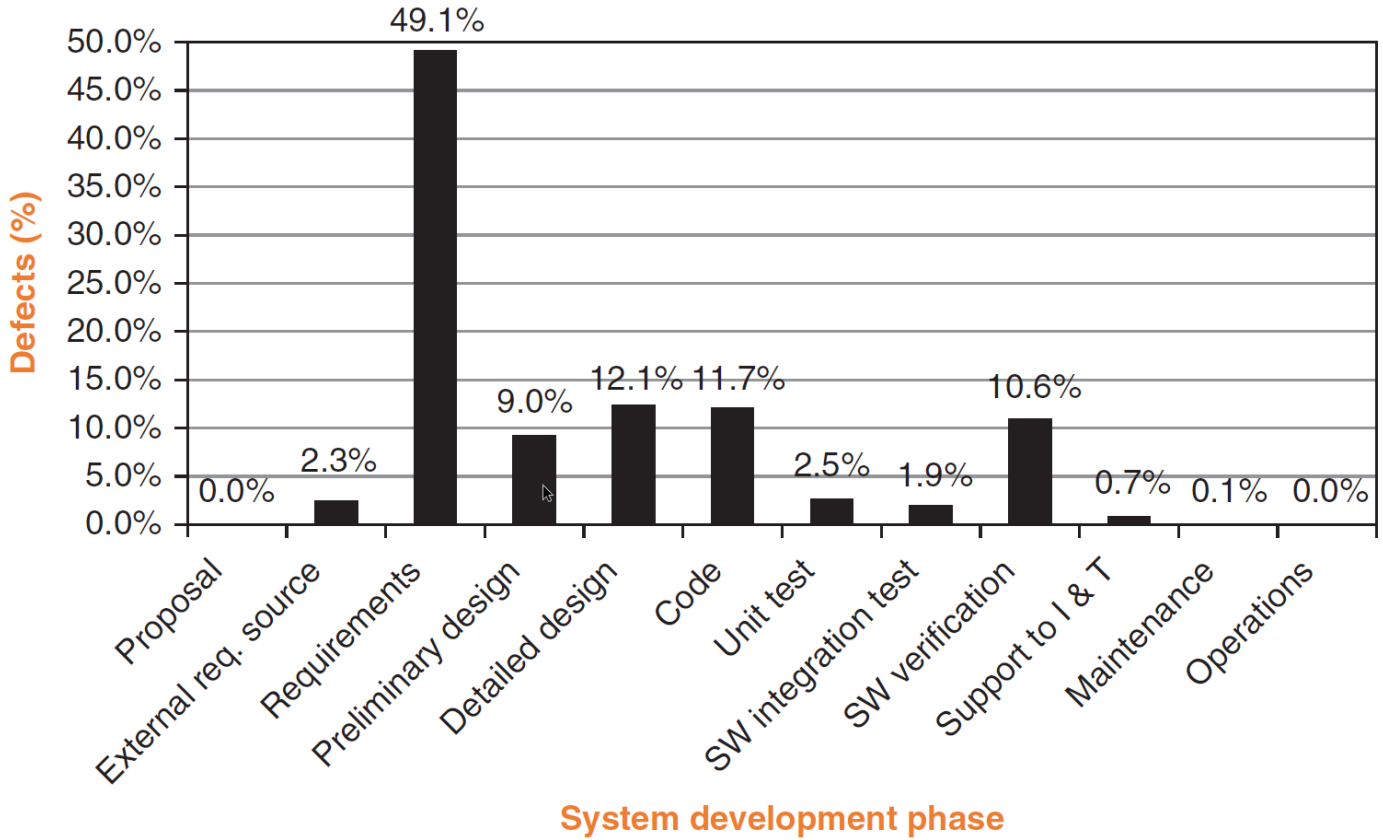
- Testen ist das Ausführen eines Programms, mit der Absicht:
  - Fehler zu finden
  - Fehler zu verhindern
  - Vertrauen/Zuversicht bzgl. der Qualität der Software zu erhalten
  - Informationen für weitere Entscheidungen bereitzustellen

- Testen ist ein konstruktiver Prozess
  - „Zerstört“ nicht das Produkt
  - Ein gefundener Fehler ist keine Kritik
  - Kommunikation muss konstruktiv bleiben
  - Während des Testens gefundene Fehler sparen Zeit und Geld

Potentielle Fehlerquellen

- Testen ist nur ein Teil der Qualitätssicherung
- ISTQB Testprinzipien zeigen:
  - Vollständiges Testen ist nicht möglich
  - Testen zeigt die Anwesenheit von Fehlern, nicht deren Abwesenheit
- Fehler in den Anforderungen
  - Mangelhafte Anforderungen /Spezifikationen
- Fehler im Design
  - Designfehler als Konsequenz ungenauer Anforderungen
  - Falsch interpretierte Anforderungen
  - Fehlende Erfahrung im Umgang mit Technologie
- Fehlerhafte Testdaten

Potentielle Fehlerquellen



-Selby P. and Selby R. W. Measurement-driven systems engineering using six sigma techniques to improve software defect detection. 2007.

Grundsätze des Testens

- ISTQB Testprinzipien
  - #1 Testen zeigt Fehler auf
    - Beweist aber nicht, dass keine Fehler mehr vorhanden sind
  - #2 Vollständiges Testen ist nicht möglich
    - Sämtliche Kombinationen von Eingaben, Vorbedingungen und Zuständen zu testen ist nicht umsetzbar/praktikabel
    - Stattdessen wird der Testfokus mittels Risiken und Prioritäten festgelegt
  - #3 Frühzeitig Testen
    - Testaktivitäten sollten so früh wie möglich begonnen werden
    - Frühzeitig gefundene Fehler sind einfach und günstig zu beheben

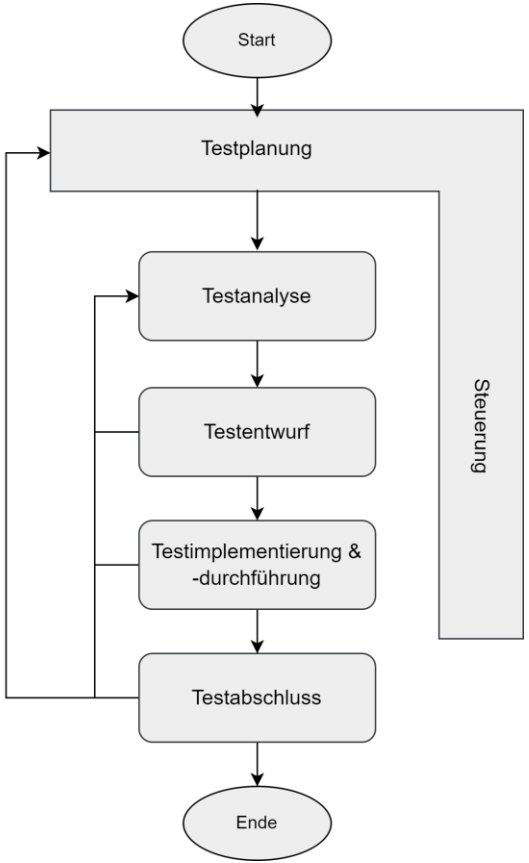
Grundsätze des Testens

- ISTQB Testprinzipien
  - #4 Fehlerhäufungen beachten
    - Fehler sind nicht gleichmäßig verteilt. Werden in einem Modul einige Fehler gefunden, sind dort meist weitere zu erwarten
  - #5 Veränderung statt Wiederholung
    - Das wiederholte Ausführen von Testfällen wird keine neuen Fehler identifizieren
  - #6 Testen ist kontextabhängig
  - #7 Trugschluss: Fehlerlose Systeme sind brauchbar
    - Fehler zu finden und zu beheben garantiert nicht, dass das Produkt den Anforderungen und Bedürfnissen des Benutzers entspricht

Testaktivitäten

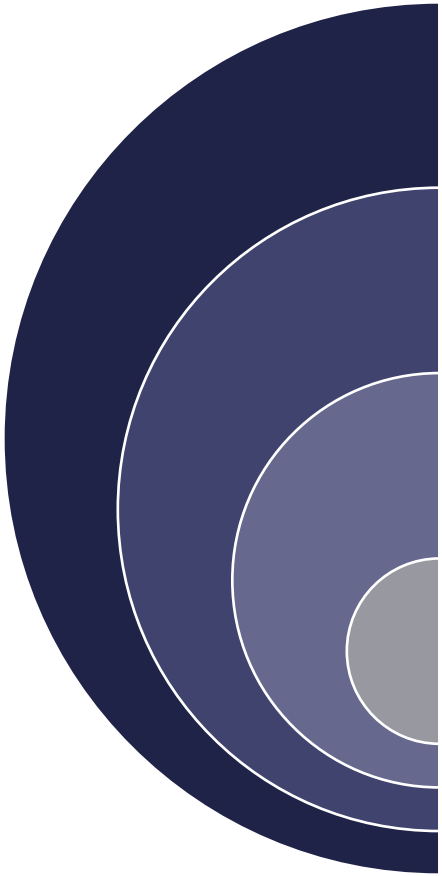
- Testaktivitäten existieren vor und nach der Testausführung
- Testaktivitäten umfassen
  - Testplanung
  - Auswahl der Testbedingungen
  - Testüberwachung / -steuerung
  - Entwurf / Ausführung von Testfällen
  - Überprüfen der Ergebnisse
  - Bewertung von Ausstiegskriterien
  - Reporting über den Testprozess und den Status des zu testenden Systems
  - Abschlussaktivitäten nach einer Testphase

ISTQB fundamentaler Test Prozess



- Testen ist ein Prozess, keine einmalige Aktivität
- Testausführung ist der sichtbarste Teil des Testens
- Um effektiv und effizient zu sein, müssen auch Tests geplant, vorbereitet und ausgewertet werden
- Aktivitäten können überlappend/gleichzeitig stattfinden
- Der Testprozess muss an die Anforderungen des Projekts angepasst werden

Teststufen



Akzeptanztests

Systemtests

Integrationstests

Komponententests

Teststufen – Komponententests

- Auch Modul- oder Unit-Tests
- Testen einer einzelnen Komponente auf korrekte Funktionalität
- Testen einer Komponente isoliert vom Rest des Systems
- Erfordert Zugang zum Source Code
- Von Entwicklern erstellt und ausgeführt

Teststufen – Integrationstests

- Testen mit dem Ziel, Fehlerzustände in den Schnittstellen und im Zusammenspiel zwischen integrierten Komponenten aufzudecken

- Zeigt Fehler in Interfaces und Interaktionen zwischen Modulen auf
- Fokus auf Integration mehrerer Komponenten
- Von Entwicklern erstellt und ausgeführt

Teststufen – Komponenten vs. Integrationstests

Warum sind funktionierende Komponententests nicht ausreichend?

Teststufen – Systemtests

- Testen eines **integrierten Systems**, um sicherzustellen, dass es spezifizierte Anforderungen erfüllt.

- Testumgebung sollte ein **Abbild der Produktivumgebung** sein, um das Risiko umgebungsspezifischer Fehler zu minimieren.

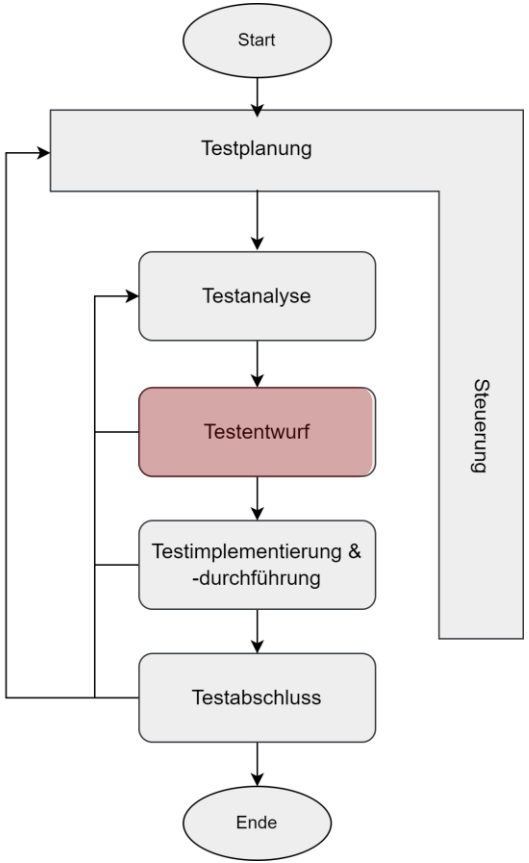
- Von unabhängigem Testteam ausgeführt

Teststufen – Akzeptanztests

- Auch Abnahmetest oder Kundenakzeptanztest
- Testet, ob das System bereit für den **produktiven Einsatz** ist und den **Abnahmekriterien des Kunden** entspricht
- Dient nicht dazu, Fehler im System zu finden
- Vom Kunden/Benutzer durchgeführt

Testentwurfsverfahren: Black Box Methoden

ISTQB fundamentaler Test Prozess



- Testentwurf
  - "Wie wird getestet?"
  - Entwurf und Priorisierung von abstrakten Testfällen
  - Identifikation von Testdaten
  - … mittels Testentwurfsverfahren

So wenig Testfälle wie möglich
So viele Testfälle wie nötig

Black Box Tests

- Basieren auf Spezifikation
- Kein Wissen über innere Struktur vorhanden
- Daten-getrieben
  - Variationen von Eingabeparametern
  - Überprüfung des Ergebnisses mit einem erwarteten Wert
- Anforderungsüberdeckung

- Methoden
  - Äquivalenzklassen
  - Grenzwerte
  - Entscheidungstabellen
  - Anwendungsfall-basiertes Testen
  - Zustandsbasiertes Testen
  - …

Äquivalenzklassen

- Einteilung möglicher Eingabewerte in Klassen/Partitionen (Äquivalenzklassen)
- Alle Werte einer Klasse/Partition führen zu demselben erwarteten Ergebnis
- Äquivalenzklassen für gültige und ungültige Werte
- Jeder Wert kann nur zu einer Äquivalenzklasse gehören

- Es ist i.d.R. ausreichend, einen repräsentativen Wert pro Partition auszuwählen
- Verwendung in Kombination mit der Grenzwertanalyse

Äquivalenzklassen Beispiel

- Kundenanforderung:
  - Ein Paketlieferdienst erweitert seine Dienstleistungen und möchte zukünftig auch Briefsendungen zustellen. Aus diesem Anlass werden die allgemeinen Zustellkonditionen überarbeitet.
  - Pakete erhalten eine allgemeine Zustellgebühr von 5€.
    Bei Briefsendungen, die weniger als 500g wiegen, sowie bei allen Briefsendungen, deren Ziel im Inland liegt, belaufen sich die Zustellgebühren auf 0,20€ pro Brief.
  - Alle anderen Briefe erhalten eine Zustellgebühr von 0,50€.

- Definieren Sie alle Äquivalenzklassen und geben Sie mögliche Testfälle an.

Äquivalenzklassen Beispiel

Klasse 1: Typ
- A1 Paket
- A2 Brief

Klasse 2: Gewicht
- B1 Gewicht < 500g
- B2 Gewicht >= 500g

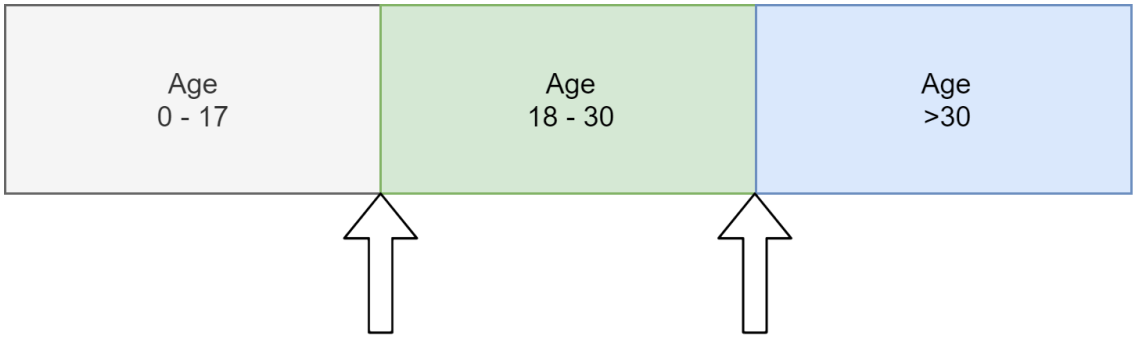
Klasse 3: Zustellungsort
- C1: Inland
- C2: Ausland

Paket				Brief			
Gewicht < 500g		Gewicht >= 500g		Gewicht < 500g		Gewicht >= 500g	
Inland	Ausland	Inland	Ausland	Inland	Ausland	Inland	Ausland
5€	5€	5€	5€	0,20€	0,20€	0,20€	0,50€

- Mögliche Testfälle:
  - T1: input { Paket (A1), 501g(B2), Inland(C1)} erw. Ergebnis {Gebühr: 5€}
  - T2: input { Paket (A1), 499g(B1), Ausland(C2)} erw. Ergebnis {Gebühr: 5€}
  - T3: input { Brief(A2), 500g(B2), Ausland(C2)} erw. Ergebnis {Gebühr: 0,50€}
  - …

Grenzwertanalyse

- Erweiterung der Äquivalenzklassenbildung
- Jeweils das Minimum und Maximum einer Äquivalenzklasse sind die Grenzwerte



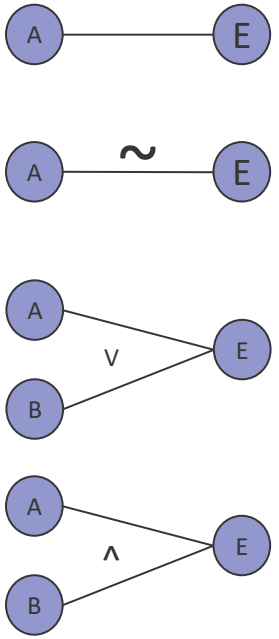
- Grenzwerte: [17] [18, 30] [31]
  2(oder mehr) Grenzwertanalyse: [16, 17] [18, 30] [31, 32]
- Das Verhalten an den Grenzen von Äquivalenzbereichen ist mit größerer Wahrscheinlichkeit falsch als das Verhalten innerhalb der Bereiche.

Entscheidungsbasierte Tests

- Gute Möglichkeit komplexe Regeln abzudecken
- Abhängigkeiten zwischen Eingabewerten werden bei Grenzwertanalysen oder Äquivalenzklassen nicht berücksichtigt
- Vollständiges Testen ist nicht möglich
  - Entscheidungstabellen sollen helfen, die Menge aller möglichen Kombinationen zu reduzieren
- Ursache-Wirkungs-Graph
  - grafische Darstellung, die Zusammenhänge/Kausalitäten zwischen Eingaben (Ursachen) und Ausgaben (Wirkung) darstellt
- Entscheidungstabellen
  - Tabelle, die zur Darstellung von Bedingungen (Eingaben) und den daraus resultierenden Aktionen (Ausgaben) verwendet wird

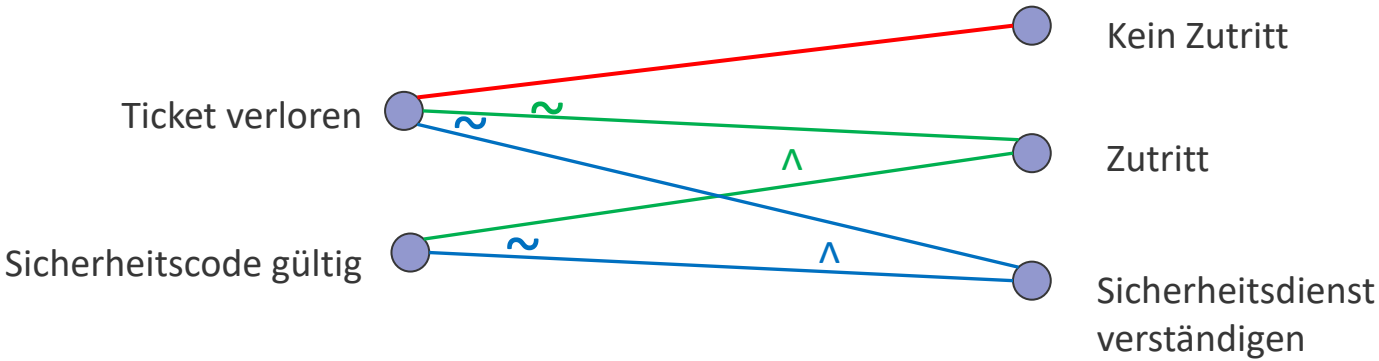
Ursache-Wirkung Graph

- Elemente
  - Auswirkung
    - Wenn A, dann E
  - Negierung
    - Wenn **nicht** A, dann E
  - Oder
    - Wenn A **oder** B, dann E
  - Und
    - Wenn A **und** B, dann E



Ursache-Wirkung Graph Beispiel

- Teilnehmer einer Konferenz bekommen nur mit einem Ticket mit gültigem Sicherheitscode Zutritt.
- Wenn der Sicherheitscode ungültig ist, muss der Sicherheitsdienst verständigt werden.
- Teilnehmer ohne Ticket erhalten keinen Zutritt.



Entscheidungstabelle Beispiel

	1	2	3
Ticket verloren	N	N	Y
Sicherheitscode gültig	Y	N	
Zutritt	Y		
Sicherheitsdienst		Y	
Kein Zutritt			Y

Entscheidungstabelle Beispiel

- Ein Buch-Verlag gibt folgenden Rabatt:
  - Kunde: “Einzelhandel”: bei einer Bestellmenge von 6 oder mehr Büchern gibt es 25% Rabatt, andernfalls keinen Rabatt
  - Kunde: “Bücherei”: Rabatt nach Bestellmenge
    - 6-19 → 5%
    - 20-49 → 10%
    - 50+ → 15%

- Erstellen Sie die Entscheidungstabelle
- Elemenieren Sie redundante Entscheidungen, sofern möglich

Entscheidungstabelle: Beispiel

	test case	1	2	3	4	5	6
Bedingung	Kundentyp	Einzelhandel	Einzelhandel	Bücherei	Bücherei	Bücherei	Bücherei
	Bestellmenge	5	6	5	19	49	51
Aktion	Rabatt	0%	25%	0%	5%	10%	15%

	test case	1	2	3	4	5	6
Bedingung	Kundentyp	Einzelhandel	Einzelhandel	Bücherei	Bücherei	Bücherei	Bücherei
	Menge >5	N	Y	N	Y	Y	Y
	Menge >= 20			N	N	Y	Y
	Menge >= 50			N	N	N	Y
Aktion	Kein Rabatt	Y		Y			
	5% Rabatt				Y		
	10% Rabatt					Y	
	15% Rabatt						Y
	25% Rabatt		Y				

Entscheidungstabelle: Beispiel

	test case	1	2	3	4	5	6
Bedingung	Kundentyp	Einzelhandel	Einzelhandel	Bücherei	Bücherei	Bücherei	Bücherei
	Menge >5	N	Y	N	Y	Y	Y
	Menge >= 20			N	N	Y	Y
	Menge >= 50			N	N	N	Y
Aktion	Kein Rabatt	Y		Y			
	5% Rabatt				Y		
	10% Rabatt					Y	
	15% Rabatt						Y
	25% Rabatt		Y				

Welche Testfälle sind redundant?

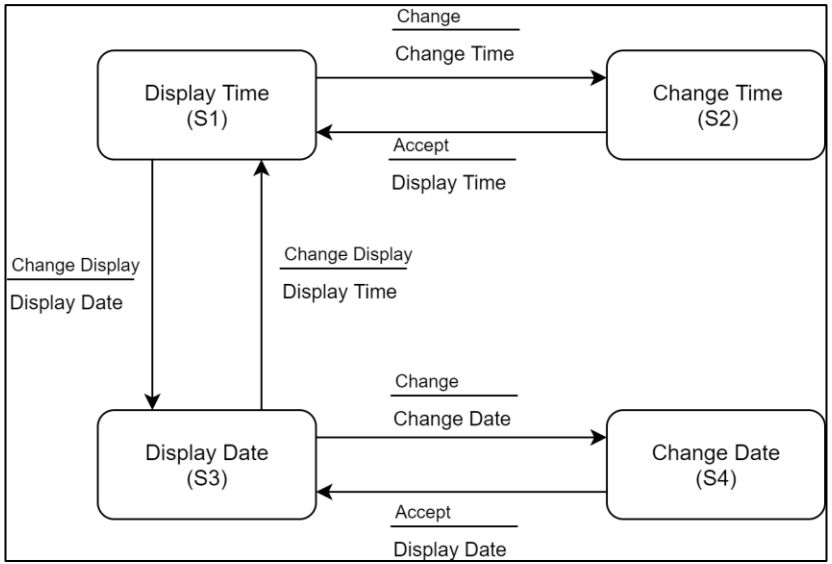
Testfall 1 und 3 haben idente Bedingungen → redundant

Der Kundentyp ist bei einer Bestellmenge < 5 Büchern egal

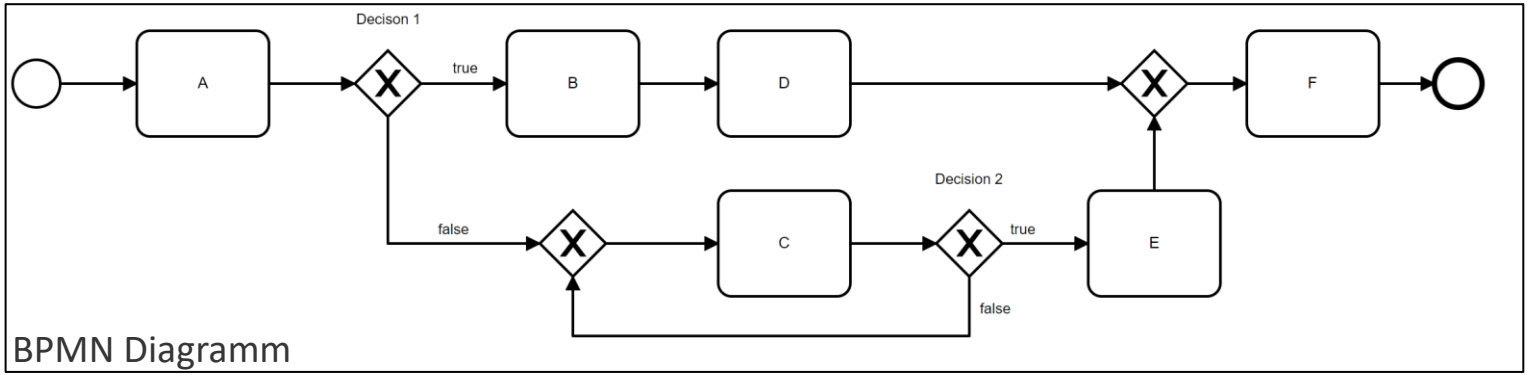
Zustandsübergangstest

- Darstellung des Verhaltens eines Systems durch Zustände
- Zeigt i.d.R. nur gültige Übergänge
- Jeder Pfad entspricht einem Testfall
- Verwendet für:
  - Menübasierte Anwendungen
  - Prozessabläufe (z.B. BPMN)

Zustandsübergangstest



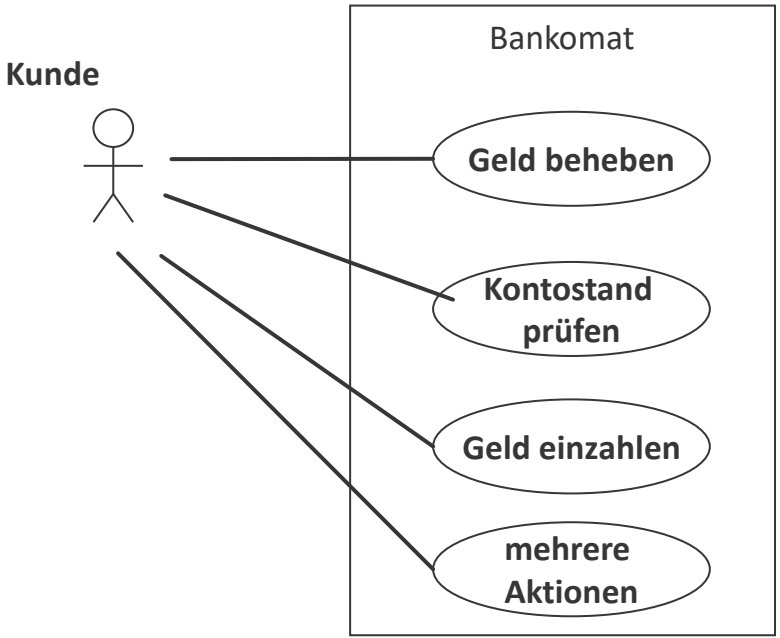
Zustandsdiagramm



BPMN Diagramm

Anwendungsfallbasiertes Testen

- Testfälle werden aus Anwendungsfällen abgeleitet
- Jeder Anwendungsfall beschreibt ein Szenario der Systeminteraktion
- Anwendungsfälle enthalten:
  - Vorbedingungen
  - Erwartete Ergebnisse
  - Nachbedingungen
- Abdeckung
  abgedeckte AF / Anzahl AF



Testentwurfsverfahren:
White Box Methoden

(Kontrollflussorientierte Testverfahren)

Kontrollflussorientierte Testverfahren

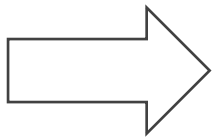
- Auch White Box / Überdeckungstests
- Es handelt sich um strukturorientierte Testmethoden
  - Basieren auf Analyse des Quellcodes
  - Orientieren sich am Kontrollflussgraphen des Programms
  - Relevant auf Unit Test Ebene
  - Definieren keine Regeln für die Erzeugung von Testdaten
  - Dienen der Ermittlung der Testabdeckung

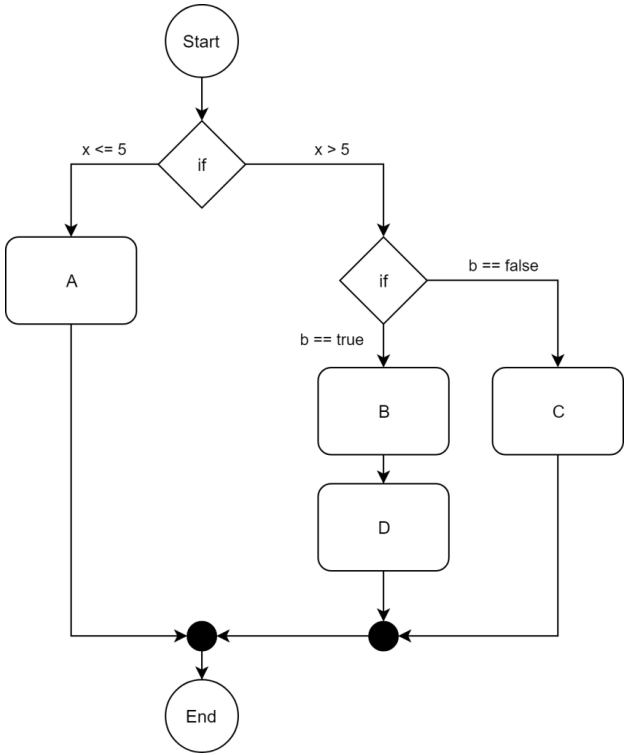
- Testarten
  - Anweisungsüberdeckung
  - Zweigüberdeckungstest
  - Pfadüberdeckungstest
  - Bedingungsüberdeckungstest

Kontrollflussgraph

- Gerichteter Graph
- Beschreibt Kontrollfluss eines Programms
- Dient zur Programmoptimierung

```
if(x <= 5) {
    A();
}
else {
    if(b) {
        B();
        D();
    }
    else {
        C();
    }
}
```





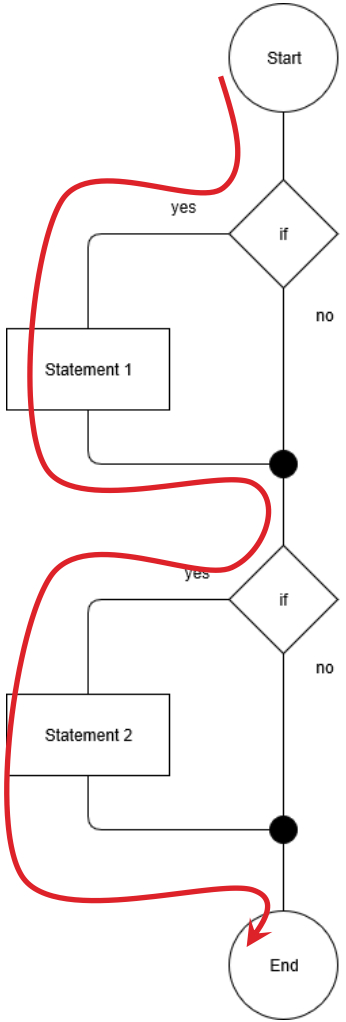
Anweisungsüberdeckung

- Statement Coverage
- Ziel: jede Anweisung muss mind. einmal durchlaufen werden
- Vollständige Anweisungsüberdeckung liegt vor, wenn sämtliche Anweisungen mindestens einmal durchlaufen werden
  - Zeigt ob toter Code existiert
    - Anweisungen, die niemals durchlaufen werden können

- Zu schwaches Kriterium für sinnvolle Testdurchführung

Anweisungsüberdeckung

- Beispiel
  - Anweisungsüberdeckung
  - jede Anweisung
    mind. ein mal durchlaufen
  - 1 Testfall



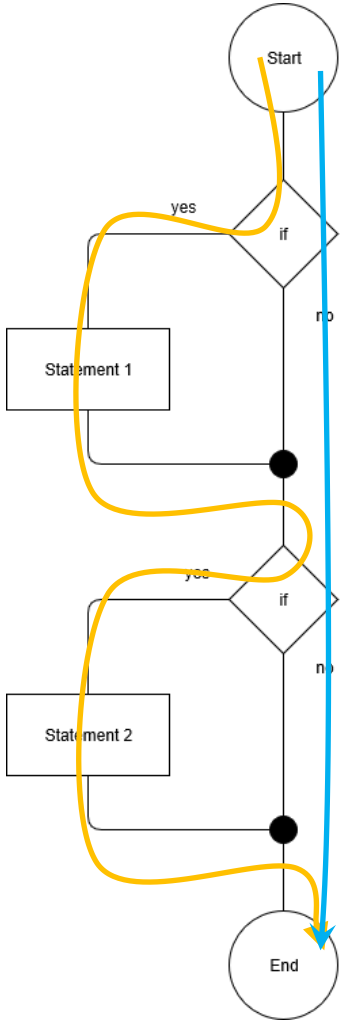
Zweigüberdeckung

- Branch Coverage
- Umfasst Anweisungsüberdeckung vollständig
- Ziel: Jede Kante muss mind. einmal durchlaufen werden
  - Zeigt nicht ausführbare Programmzweige auf
  - Hilft oft durchlaufene Programmteile gezielt zu optimieren

- Im Gegensatz zum Anweisungsüberdeckungstest muss jede Entscheidung mindestens einmal true und false annehmen
- Problematik:
  - Unzureichender Test von Schleifen
  - Komplexe Logik in Statements wird nicht berücksichtigt

Zweigüberdeckung

- Beispiel
  - Zweigüberdeckung
  - Abdeckung aller Kanten
  - 2 Testfälle

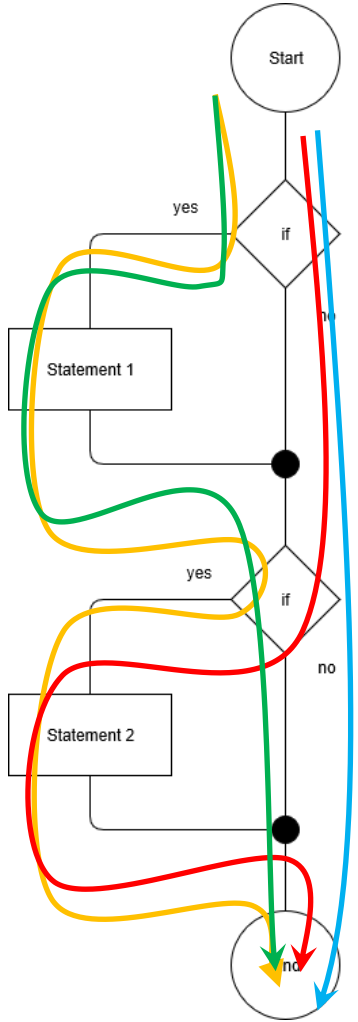


Pfadüberdeckung

- Path Coverage
- Ziel: Abdeckung aller Pfade von Start- bis Endknoten
- Für komplexe Softwaremodule meist nicht durchführbar
  - unendlich hohe Anzahl von Pfaden
  - Schleifen bieten extrem viele Pfade

Pfadüberdeckung

- Beispiele
  - Pfadüberdeckung
  - Abdeckung aller Pfade
  - 4 Testfälle



Bedingungsüberdeckung

- Condition Coverage
- Ziel: Überprüfung zusammengesetzter Entscheidungen, Teilentscheidungen

- Einfacher Bedingungsüberdeckungstest
  - Fordert den Test aller atomaren Teilentscheidungen gegen true und false
- Mehrfach-Bedingungsüberdeckungstest
  - Fordert den Test aller Wahrheitswertekombinationen der atomaren Teilentscheidungen
  - Sehr aufwändig

Bedingungsüberdeckung

- Entscheidung: `(((a == 0) || (b <5)) && ((x < 6) || (y == 0)))`
- Im Wesentlichen: `((A || B) && (C || D))`

- Sind a, b, x und y unabhängig, können A, B, C, D unabhängig voneinander true, oder false werden

- Beispiel
  - Einfacher Bedingungsüberdeckungstest
  - Testfall 1 A = false, B = false, C = false, D = false
  - Testfall 2 A = true, B = true, C = true, D = true

Bedingungsüberdeckung

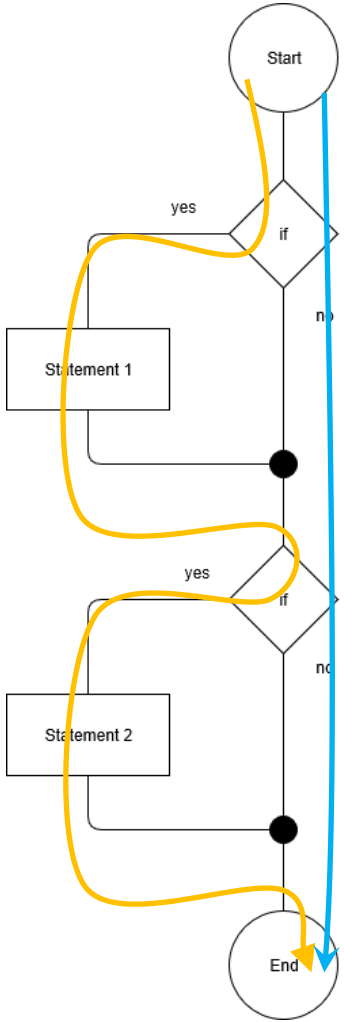
- Entscheidung: `(((a == 0) || (b <5)) && ((x < 6) || (y == 0)))`
- Im wesentlichen: `((A || B) && (C || D))`

- Sind a, b, x und y unabhängig, können A, B, C, D unabhängig voneinander true, oder false werden

- Beispiel
  - Mehrfach-Bedingungsüberdeckungstest
  - 16 Testfälle
    - 16 Wahrheitskombinationen (2^4)

Bedingungsüberdeckung

- Beispiele
  - Bedingungsüberdeckung
  - Einfacher Bedingungsüberdeckungstest
  - alle atomaren Teilentscheidungen true und false
  - Einfacher Bedingungsüberdeckungstest
    - 2 Testfälle

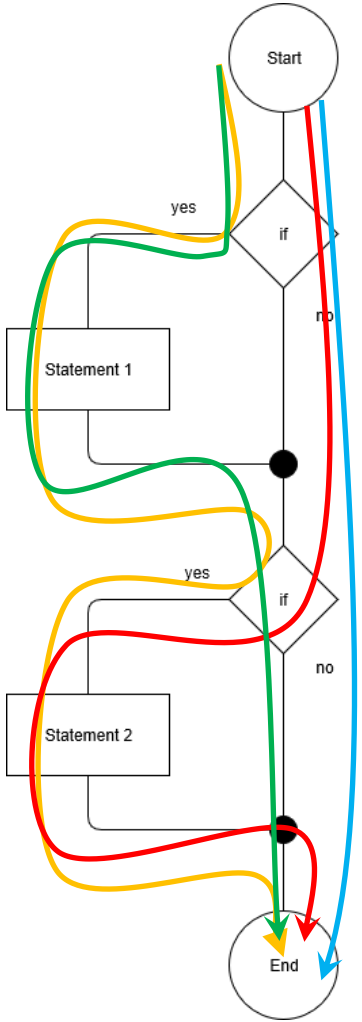


Bedingungsüberdeckung

- Beispiele
  - Bedingungsüberdeckung
  - Mehrfach-Bedingungsüberdeckungstest

  - alle Wahrheitswertekombinationen der atomaren Teilentscheidungen

  - Mehrfach-Bedingungsüberdeckungstest
    - 4 Testfälle



Überdeckung

- Wie viel % Coverage ist genug?

80%

82%

82,5%

- Quantitative Metrik vs. qualitative Methodik
- Coverage allein ist kein ausreichendes Kriterium

Erfahrungsbasiertes Testen

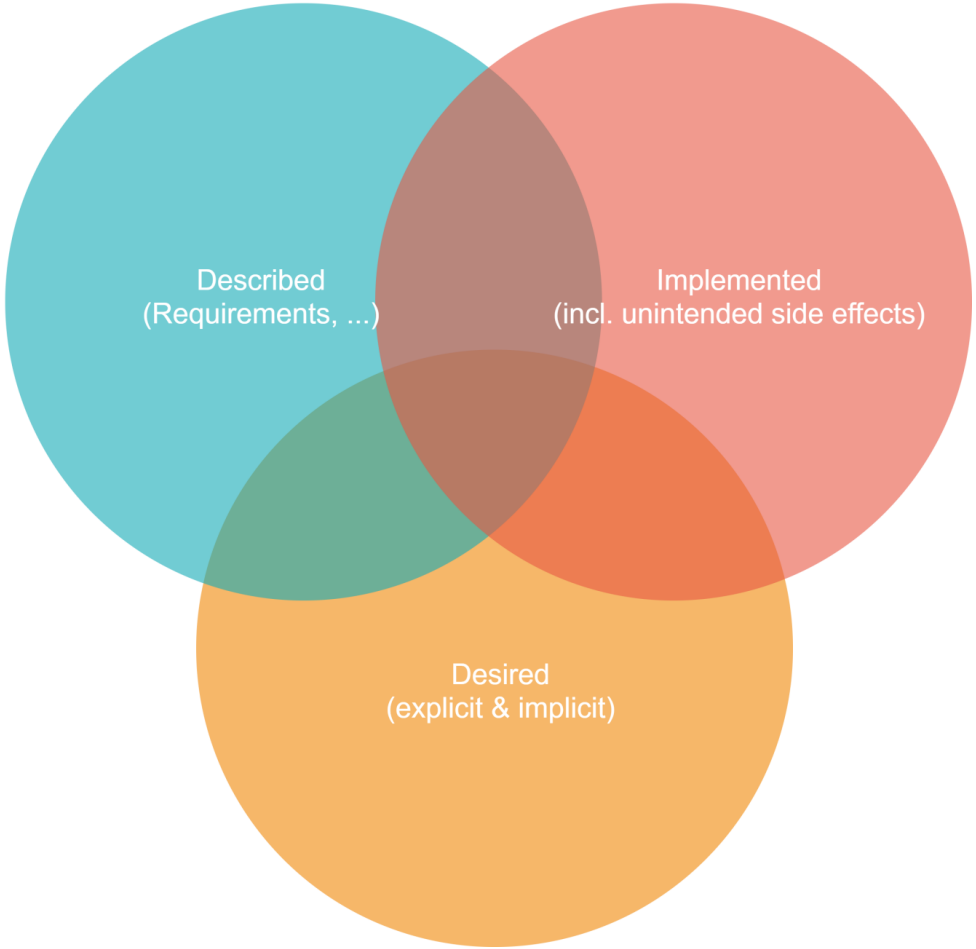
Erfahrungsbasiertes Testen

- Auch **Exploratives Testen**
- können mit verschiedenen Testentwurfsverfahren kombiniert werden

- Testfälle basieren auf Intuition und Erfahrung des Testers
  - Wo könnte es Probleme geben?
  - Wo waren Fehler in der Vergangenheit? (zB in anderen Projekten)
  - Welche Komponenten sind unter Zeitdruck entstanden?
  - …

- Erfahrungsbasiertes Testen muss dokumentiert werden

Erfahrungsbasiertes Testen



Fragen?

Referenzen

- T. Grechenig et al; Softwaretechnik Mit Fallbeispielen aus realen Entwicklungsprojekten, 2010
- ISTQB Foundation Level Syllabus
- A. Spillner and T. Linz; Basiswissen Softwaretest. 5. Aufl. dpunkt Verlag, 2012.
- D. Graham et al; Foundations of Software Testing: ISTQB Certification. Cengage Learning EMEA, 2008.
- IEEE Std 610.12-1990 (R2002), IEEE Standard Glossary of Software Engineering Terminology IEEE, 1990.
- A. Spillner et al; Praxiswissen Softwaretest – Testmanagement: Aus- und Weiterbildung zum Certified Tester, 2014