Technische Hochschule Rosenheim

Software-Engineering-Praxis

Prof. Dr. Gerd Beneken

Kapitel 13

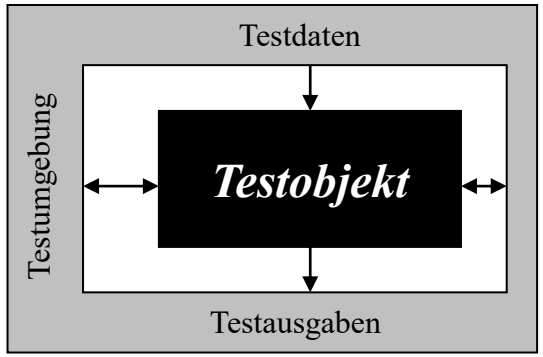
Testverfahren

Testverfahren

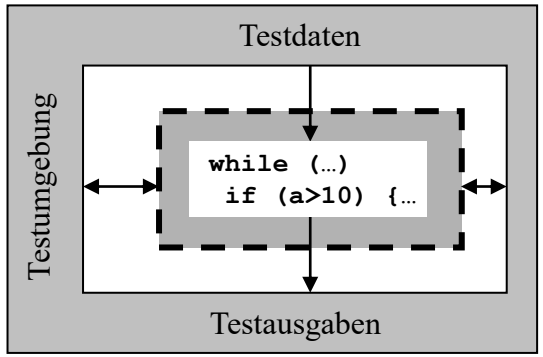
- Blackbox-Tests
- Whitebox / Glassbox-Tests
- Fehlerbasierte Tests
- Explorative / Erfahrungsbasierte Tests

Blackbox und Whitebox-Tests

- Blackbox-Test (Spezifikations-Orientiert)
  - System/Modul = Blackbox, nur Außensicht,
    Code: Nur Schnittstelle sichtbar, keine Implementierung
  - Grundlage für Tests: Spezifikation, Zufall, …
  - Beobachtung des Verhaltens von Außen



- Glassbox-Test (Struktur-Orientiert)
  - Struktur / Code sichtbar
  - Grundlage für Tests: Quelltexte, Architektur,
    Spezifikation
  - Beobachtung auch interner Zustände
  - Häufig mit Werkzeugen wie JaCoCo



Weitere Verfahren zum Testentwurf

- Fehlerbasierter Test
  - Grundlage: Fehlertaxonomien (wie z.B. aus Cem Kaners Buch)
  - Analyse bisheriger Fehler: Testfälle versuchen Fehler zu reproduzieren
  - Ziel: Typische Fehlerstellen testen, über Checkliste

- Erfahrungsbasierter Test
  - Grundlage: Erfahrungen der Anwendungsentwickler bzw. Tester
  - Testen ist eher erforschend, Tester sucht nach Fehlern
  - Wichtig: Intuition des Testers
    (auch ungewöhnliches Verhalten z.B. Lüfter springt an)
  - = ***Explorativer Test***

Technische Hochschule Rosenheim

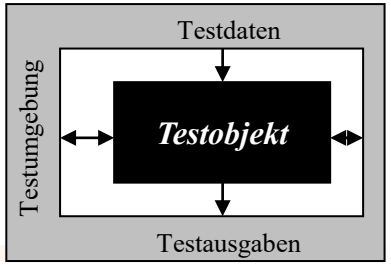
Software-Engineering-Praxis

Prof. Dr. Gerd Beneken

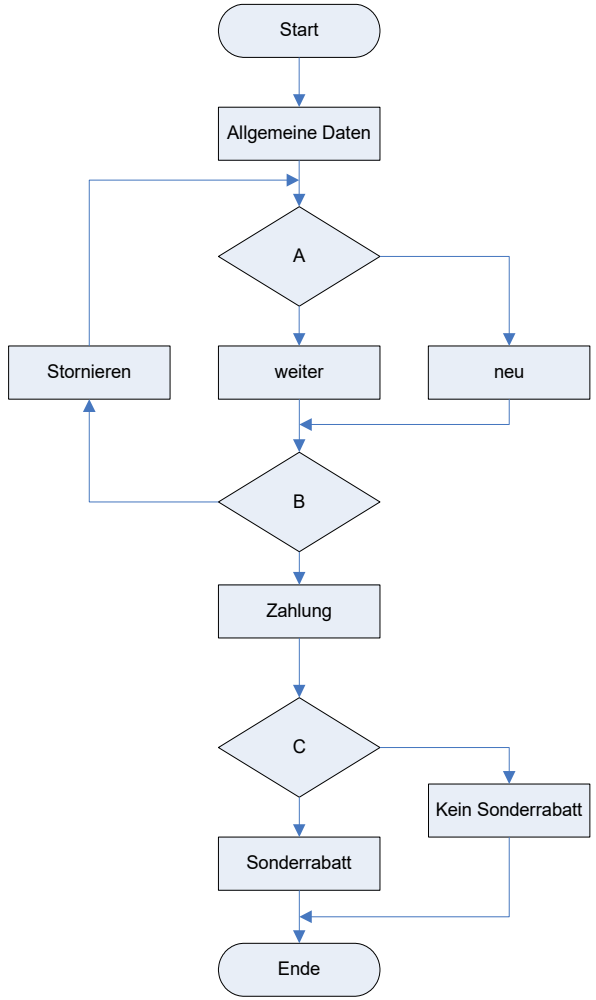
Kapitel 13.2

Backbox-Tests

Spezifikations-Orientierte Testverfahren (Black-Box-Test)



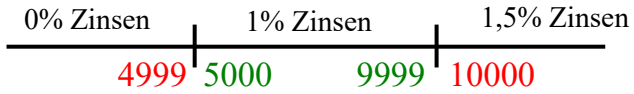
- Grundlagen: Spezifikation, Schnittstellenbeschreibungen
- Wie werden Testfälle gefunden?
  - Workflows systematisch durchlaufen
  - Anwendungsfälle systematisch durchlaufen
  - Datenmodell:
    CRUD Test (= Anlegen, Ändern, Löschen, Suchen)
  - GUI: Screenflow, Eingabefelder, Dialogzustände abarbeiten
  - Schnittstellen: Vor-, Nachbedingungen, Entscheidungstabellen
- Testverfahren
  - Grenzwertanalyse, Äquivalenzklassenbildung, Entscheidungstabellen
  - Pfadtest (= Durchlaufen von Workflows / Prozessen)
  - Zustandstest (= Durchlaufen von Zustandsautomaten)



Blackbox – Verfahren Äquivalenzklassenbildung

- Ausgangspunkt = Parameter einer Operation / Methode oder Eingabefelder in GUI
- Von ***Eingabedaten werden Äquivalenzklassen gebildet***
  - Äquivalenzklasse = System verhält sich bei allen Daten aus der Äquivalenzklasse ähnlich/gleich
  - Beispiel: Zinsen bei Festgeld Äquivalenzklassen für den angelegten Betrag: 0 bis 4999 €,
    5000 € bis 9999 €, 10000 € bis 15000 € jeweils identisch
- ***Falsche Daten bilden jeweils eigene Äquivalenzklassen***
  (z.B. zu hohe, zu niedrige Werte, Buchstaben statt Zahlen, …)
  - Beispiel für Beträge (s.o) : Negativer Betrag, Buchstaben, …
- ***Test mit nur jeweils einem Repräsentanten*** der Äquivalenzklasse
  - Bei 0..5000€ beispielsweise 2500 €

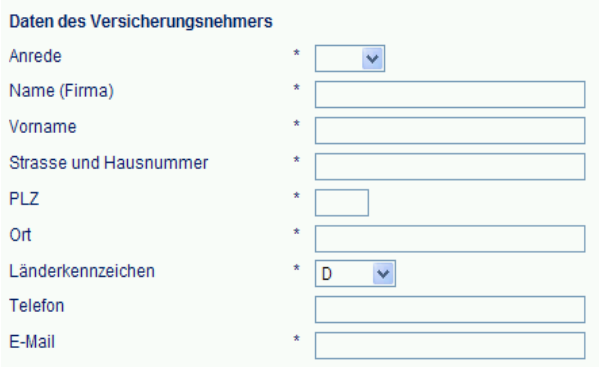
Blackbox – Verfahren Grenzwertanalyse



- Fehlerhypothese
  - Logik und Berechnungsfehler häufig an den Grenzen von Äquivalenzklassen
  - Entwickler prüft z.B. mit < statt mit <=
- Kleinsten und größten Wert testen, diesen dann um kleinstes Inkrement vergrößern und verkleinern
  - Erster Wert => Finden über ***kleinstes mögliches Inkrement***
  - ***Erster Wert links / rechts innerhalb der Äquivalenzklasse***
    (bei 5000€ - 9999€ sind das 5000 und 9999)
  - ***Erster Wert links / rechts außerhalb der Äquivalenzklasse***
    (bei 5000€ - 9999€ sind das 4999 und 10000)
- Voraussetzung: Auf Eingabemenge gibt es eine mathematische Ordnung (<= Relation)
  - Beispiele: Integer, Double, Date, (String)
  - Gegenbeispiel: Aufzählungstypen (Jugendlich, Erwachsen, Rentner), (Herr, Frau, Firma), hier ist Verfahren nicht anwendbar

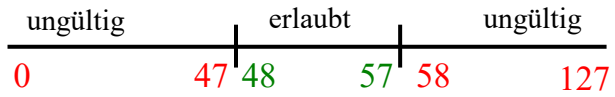
Grenzwerte: Eingabefelder für Texte

- Fehlerhypothesen
  - Pufferüberlauf bei zu langen Texten?
  - Test auf muss / kann Felder?
  - Sonderzeichen-Prüfung korrekt (z.B. über regulären Ausdruck)?
- Textlängen prüfen
  - Grenzwerte (positiv): minimale Länge / maximale Länge Eingabefeld
  - Grenzwerte (negativ): minimale Länge – 1 oder
    leere Eingabe, maximale Länge +1, möglichst langer Text
- Sonderzeichen prüfen
  - Äquivalenzklassen (positiv): Text mit erlaubten Buchstaben (a-z,ä,ü,ö A-Z,Ä,Ü,Ö,ß,à,è,ô,…), erlaubte Ziffern (0 – 9),
    erlaubte Sonderzeichen (-.;,‘`)
    - Beispiel: „Dr. Leuthäuser-de‘Souza“, „Deißenböck“
  - Äquivalenzklasse (negativ): Text mit Sonderzeichen
  - Äquivalenzklasse (negativ): Text mit Leerzeichen
  - Frage: Wie weit geht Internationalisierung (Türkisch?, Norwegisch?)
  - Grenzen jeweils zu nicht erlaubten ASCII Zeichen



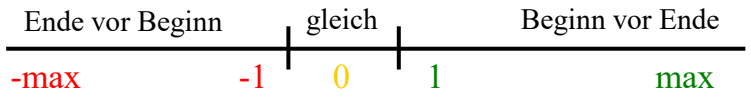
Beispiel: ASCII – Codes

Zeichen ‚/‘	= Code 47
Ziffer **0**	= Code 48
…	
Ziffer **9**	= Code 57
Zeichen ‚:‘	= Code 58



Grenzwerte: Datums- und Zeitangaben

- Datums- und Zeitangaben
  - Grenzwerte (positiv): Maximales Datum (Zukunft), Minimales Datum (Vergangenheit / heute)
  - Grenzwerte (negativ): Minimales Datum -1 Tag / Gestern, Maximales Datum + 1 Tag
  - Sonderfälle (Semantik): 29.02 in (nicht) Schaltjahren, 31.04., evtl. 00.00.0000 und 99.99.9999
  - Sonderfälle (Syntax): Buchstaben und Sonderzeichen
- Zeitintervalle
  - Grenzwerte (positiv): 0 Tage, maximale Tageszahl
  - Grenzwerte (negativ): -1 Tag, maximale Tageszahl +1



Test über Entscheidungstabellen

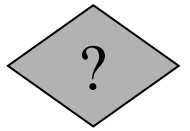
- Entscheidungstabelle
  = ***Systematische*** Betrachtung von Ursachen und Wirkungen
  - Ursache: Erfüllt / Nicht erfüllt
  - Wirkung: Tritt ein / tritt nicht ein
- Testfälle: Alle erfüllt / nicht erfüllt Kombinationen der Ursachen
- Problem: Sehr viele Testfälle
  = 2^n bei n Ursachen
- Idee: Zusammenfassung der Testfälle mit derselben Wirkung

Bedingung (Ursache)	1	2	3	4	5	6	7	8
Limit überschritten	N	J	N	J	N	J	N	J
Adresse veraltet	N	N	J	J	N	N	J	J
Karte gestohlen	N	N	N	N	J	J	J	J
Handlungen (Wirkungen)								
Limit erhöhen		J		J				
Adresse aktualisieren			J	J				
Sicherheitsdienst informieren					J	J	J	J
Zahlung erlauben	J	J	J	J				

Vereinfachung

Bedingung (Ursache)	1	2	3	4	5
Limit überschritten	N	J	N	J	-
Adresse veraltet	N	N	J	J	-
Karte gestohlen	N	N	N	N	J
Handlungen (Wirkungen)					
Limit erhöhen		J		J	
Adresse aktualisieren			J	J	
Sicherheitsdienst informieren					J
Zahlung erlauben	J	J	J	J	

Entscheidungspunkt-Test



- = Systematischer Test der Entscheidungspunkte
  (= Boolescher Ausdrücke an Verzweigungspunkten)
- Beispiel aus Rabattberechnung:

```
IF (Anzahl BÜcher > 8 OR Summe >= 250) THEN Sonderrabatt
```

- Abdeckungsarten
  - ***Bedingungsabdeckung***

Anzahl Bücher > 8	Summe >= 250 €	Ergebnis
1	0	1 (Sonderrabatt)
0	1	1 (Sonderrabatt)

  - ***Entscheidungsabdeckung***

Anzahl Bücher > 8	Summe >= 250 €	Ergebnis
0	1	1 (Sonderrabatt)
0	0	0

*) Beispiel aus: Komen et al.: T-Map Next, dpunkt, 2007

Entscheidungspunkt-Test /2

- Modifizierte Bedingungs-/Entscheidungsüberdeckung (MCDC) =Minimale Mehrfachüberdeckung

Anzahl Bücher >8	Summe >=250	Ergebnis
1	0	1 (Sonderrabatt)
0	1	1 (Sonderrabatt)
0	0	0

- Mehrfachbedingungsüberdeckung

Anzahl Bücher >8	Summe >=250	Ergebnis
1	1	1 (Sonderrabatt)
1	0	1 (Sonderrabatt)
0	1	1 (Sonderrabatt)
0	0	0

*) Beispiel aus: Komen et al.: T-Map Next, dpunkt, 2007

Entscheidungspunkt-Test
Modifizierte Bedingungs-/Entscheidungsüberdeckung (MCDC)

- Idee: Verringerung der Testfälle über MCDC
- Zusicherung der MCDC für Bedingungen A,B,C, …
  - Es gibt mindestens eine Testsituation, in der das Ergebnis Wahr ist, weil die Bedingung A Wahr ist
  - Es gibt mindestens eine Testsituation, in der das Ergebnis Falsch ist, weil die Bedingung A falsch ist
  - Dies gilt ebenso für alle anderen Teil-Bedingungen B,C, …
- Finden der MCDC
  1. Tabelle erstellen, mit drei Spalten: Bedingung, 1,0
  2. Für jede der n-Bedingungen eine Zeile (also n Zeilen)
  3. Füllen der Tabellenzellen mit n Punkten
  4. Diagonal wird die Spalte 1 mit 1 und die Spalte 0 mit 0en gefüllt
  5. Verbleibende Punkte werden durch die jeweils ***neutralen Elemente*** des Teilausdrucks ersetzt
  6. Doppelte Elemente werden entfernt

Entscheidungspunkt-Test /2
Modifizierte Bedingungs-/Entscheidungsüberdeckung

- Schritt 4

R = A OR B	1	0
A= Anzahl Bücher > 8	1 .	0 .
B=Summe >=250 €	. 1	. 0

- Schritt 5

R = A OR B	1	0
A= Anzahl Bücher > 8	1 0	0 0
B=Summe >=250 €	0 1	0 0

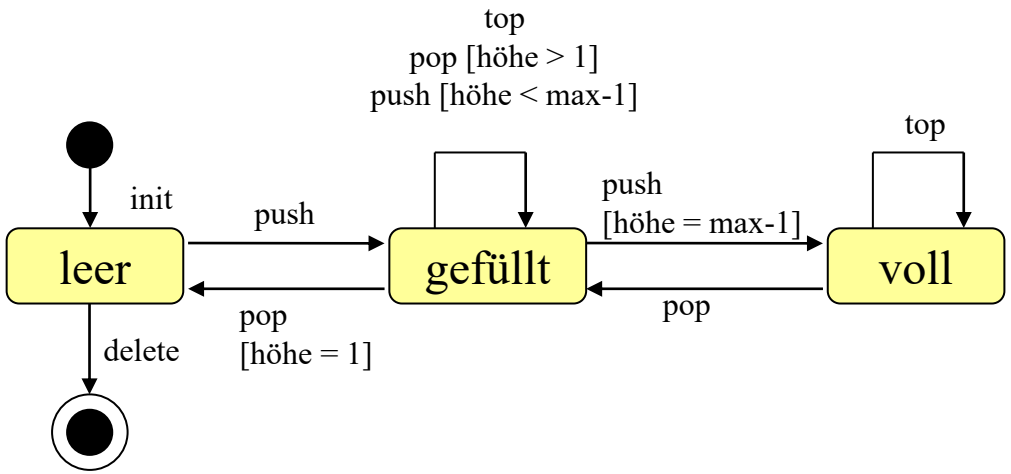
- Schritt 6

R = A OR B	1	0
A= Anzahl Bücher > 8	1 0	0 0
B=Summe >=250 €	0 1	~~0 0~~

*) Beispiel aus: Komen et al.: T-Map Next, dpunkt, 2007

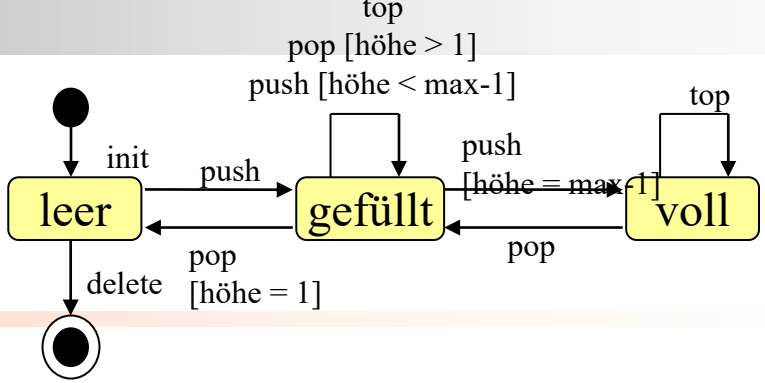
Zustandsbezogener Test

- Software ändert durch Testfälle ***ihren Zustand***
  - Dialogzustand (z.B. dirty und clean)
  - Zustand von Verträgen und anderen Daten (z.B. in Bearbeitung, aktiv, inaktiv, …)
- Idee des Zustandsbezogenen Tests
  - Definition / Analyse der Systemzustände, z.B. als Endlicher Automat
  - ***Systematisches Durchlaufen des Automaten*** über Testfälle
- Standardbeispiel: Stack*)

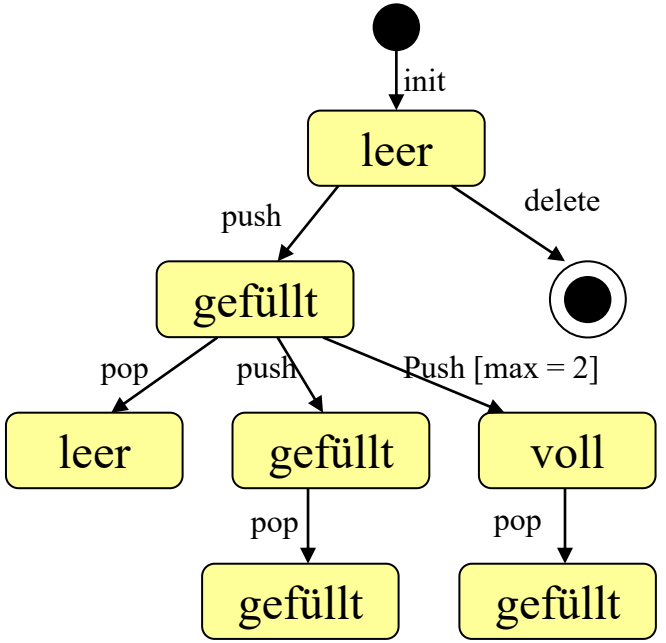


*) Beispiel aus: Spillner, Linz: Basiswissen Softwaretest, dpunkt, 2010

Zustandsbezogener Test



- Varianten des Zustandsbezogenen Tests
  1. Jeder Zustand muss einmal erreicht worden sein
  2. Alle Zustände durchlaufen und alle Funktionen aufgerufen
  3. Alle Zustandsübergänge durchlaufen
- Idee zu 3. : Übergangsbaum
  1. Wurzel = Startzustand
  2. Für jeden Übergang vom Startzustand ein Unterbaum
  3. Für jeden Übergang der Unterbäume weitere Unterunterbäume
  4. Punkt 3 solange fortsetzen, bis
     - Blattzustand wurde bereits unterwegs von der Wurzel aus erreicht, oder
     - Blattzustand ist Endzustand.
- Testfall = Pfad von Wurzel zu jedem Blatt des Baumes



Hier: 4 Testfälle

*) Beispiel aus: Spillner, Linz: Basiswissen Softwaretest, dpunkt, 2010

Technische Hochschule Rosenheim

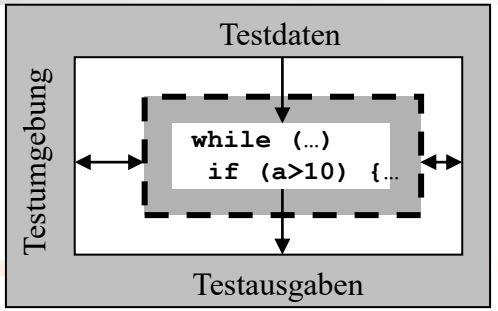
Software-Engineering-Praxis

Prof. Dr. Gerd Beneken

Kapitel 13.3

Glassbox-Tests

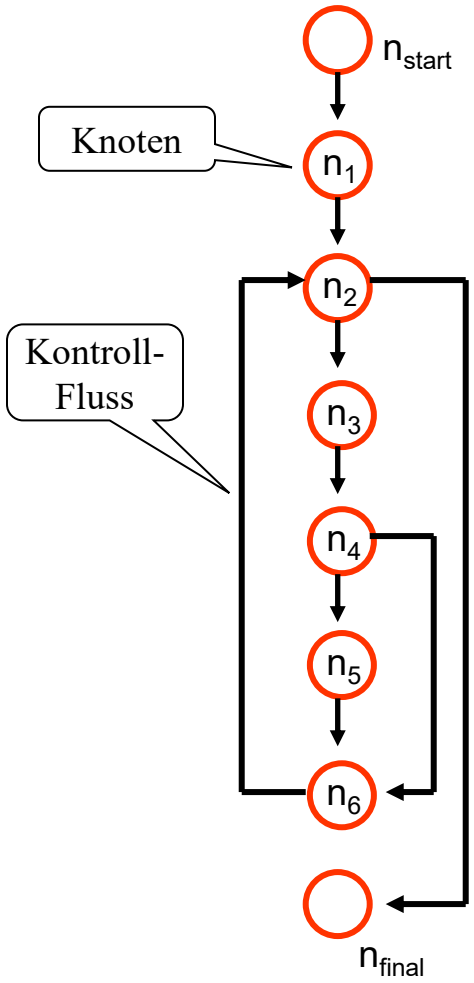
Strukturorientierte Testverfahren (White-Box / Glass-Box)



- Grundlage: Quelltexte, Schnittstellenbeschreibungen
- Wie werden Testfälle gefunden?
  - Aktiv: ***Code lesen*** und Testfälle finden, die bestimmte Überdeckungen erfüllen
  - Passiv: Andere Testverfahren verwenden und ***mit Werkzeug Überdeckung*** ermitteln
- Überdeckungsarten, z.B.
  - Anweisungsüberdeckung (jede Anweisung durchlaufen)
  - Zweigüberdeckung (bei if, switch, while, for sind alle Zweige durchlaufen)

```
public int berechneGGT
     (int zahlA, int zahlB) {
  if (zahlA > 0 && zahlB > 0) {
     while (zahlA != zahlB) {
        while (zahlA > zahlB) {
           zahlA -= zahlB;
        }
        while (zahlB > zahlA) {
           zahlB -= zahlA;
        }
     }
   } else {
      zahlA = 0;
   }
   return zahlA;
}
```

Anweisungs- und Zweigüberdeckung



Programmstück in *C++*

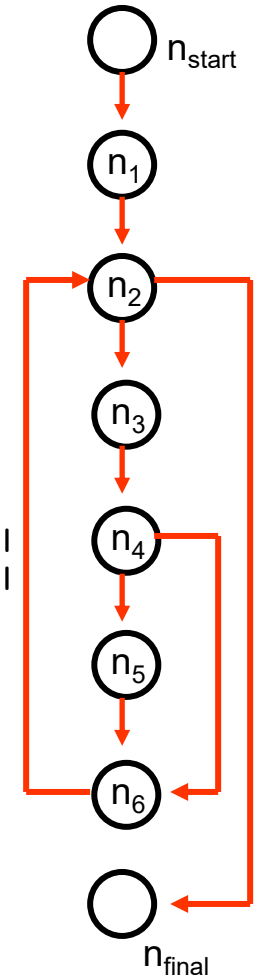
```
cin >> Zchn;  // Zeichen einlesen

while ((Zchn >= 'A') && (Zchn <= 'Z')
      && (Gesamtzahl < INT_MAX)) {

  Gesamtzahl = Gesamtzahl + 1;

  if ((Zchn == 'A') || (Zchn == 'E') ||
      (Zchn == 'I') || (Zchn == 'O') ||
      (Zchn == 'U'))
    VokalAnzahl = VokalAnzahl + 1;

  cin >> Zchn; // Zeichen einlesen
}
```



*) aus Balzert, Lehrbuch der SW-Technik, Bd.2, 1. Auflage, 1998

Strukturorientierte Testverfahren
Whitebox - Verfahren

Strukturorientierte Testverfahren streben an:

- ***Anweisung***süberdeckung
- ***Zweig***überdeckung
- ***Bedingung***süberdeckung
- ***Pfad***überdeckung

Grundlage für Überdeckungsmessung ist ***Kontrollflussgraph*****:**

- Gerichteter Graph, zur Darstellung der Kontrollstruktur von Programmen
- Anweisungen werden als Knoten und
- Programmverzweigungen als Kanten modelliert.

Anwendung

- Code Lesen und dann Aufrufparameter „entwerfen“, so dass Überdeckung erreicht wir
- Andere Testverfahren durchführen und Testüberdeckung messen

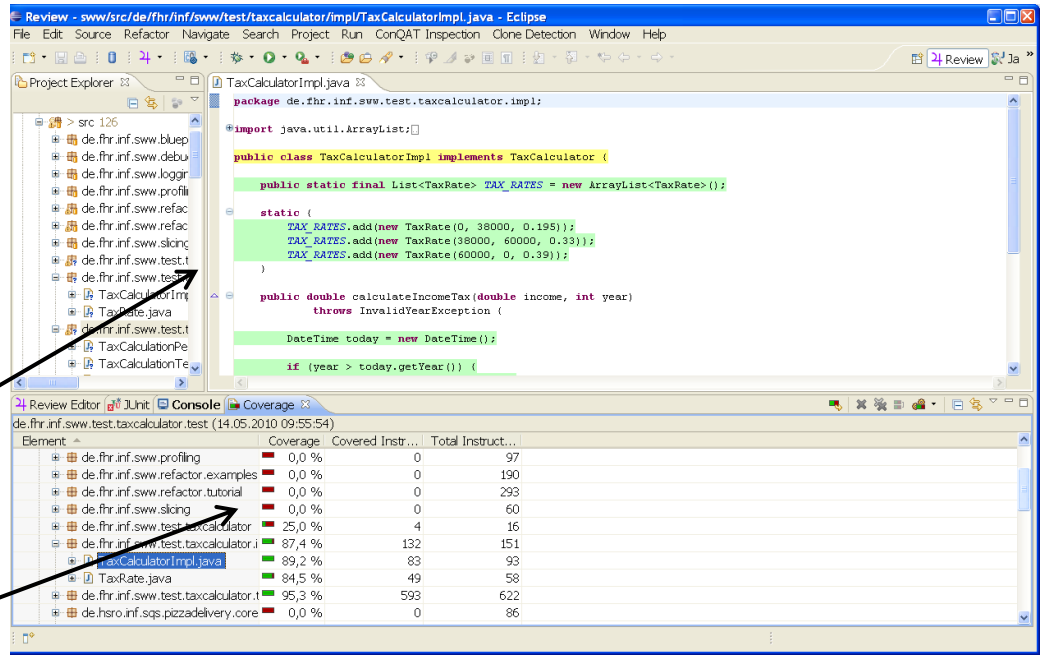
Anweisungs- und Zweigüberdeckung Nachmessen mit Code Coverage

- Idee: Verwendung eines ***Coverage Tools***
  - Java: ECL-EMMA (Eclipse), Cobertura (in der CI-Pipeline!!)
- Testfälle mindestens: Anweisungsüberdeckung wichtiger Bereiche anstreben
- Im Nightly-Build: Trend der Überdeckung beobachten

Farben:
- nicht getestet
- teilweise getestet
- vollständig getestet

Hier: Quelltext vollständig getestet

Grad der Anweisungsüberdeckung



Technische Hochschule Rosenheim

Software-Engineering-Praxis

Prof. Dr. Gerd Beneken

Kapitel 13.4

Exploratives Testen

Prüfen vs. Erforschen

Testen mit definierten Testfällen

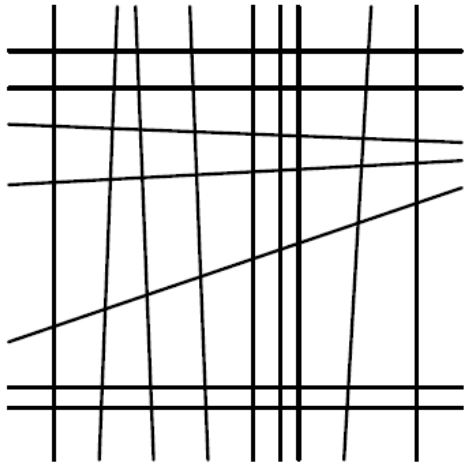
Automatisierter Test

Prüfen
(= Sicherheitsnetz)

Suchendes Testen über die Erfahrung des Testers

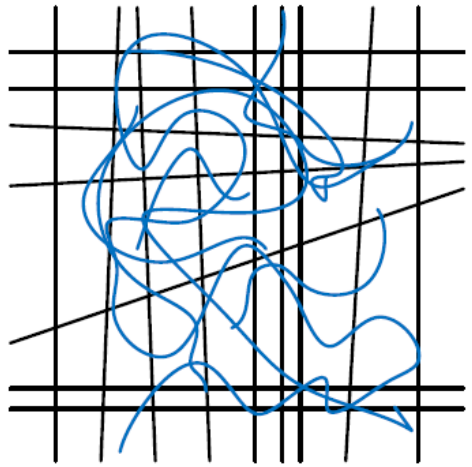
Erforschen
(was fällt durchs Sicherheitsnetz?)

Vergleich beider Verfahren



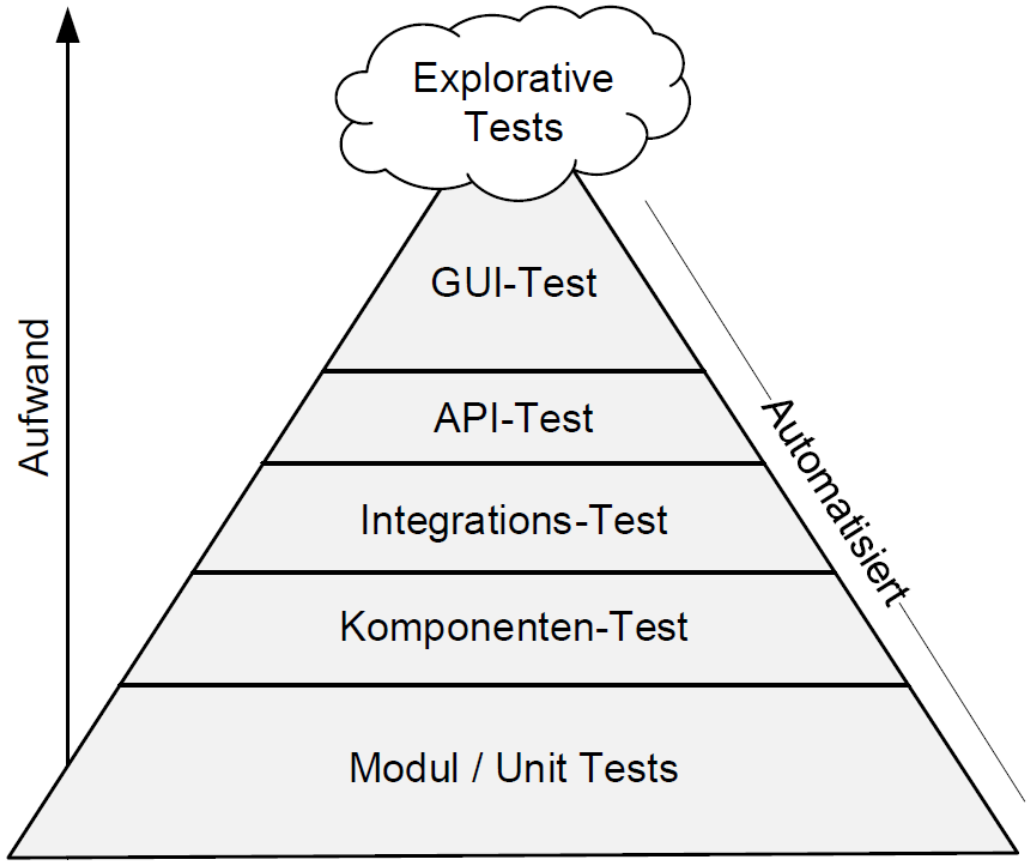
Prüfendes Testen mit automatisierten Tests = Sicherheitsnetz

Forschendes Testen mit explorativen Tests



Kombination beider Verfahren

Testpyramide nach Cockburn



Sprüche

"Exploratory testing involves ***simultaneously learning, planning, running tests, and reporting / troubleshooting results***."

[Cem Kaner]

"Exploratory testing is an ***interactive process*** of concurrent product exploration, test design and test execution."

" To the extent that the next test we do is influenced by the result of the last test we did, we are doing exploratory testing."

[James Bach]

Erfahrungsbasierte (Explorative) Testverfahren = Erforschen

- Grundlage: Erfahrung des Testers / der Anwendungsentwickler
- Wie werden Testfälle gefunden?
  - Nach Risiko: ***Raten, wo Fehler sein könnten***
  - Start: z.B. mit Funktionen, die in früheren Versionen fehleranfällig waren
  - Planen, Entwerfen und Durchführen von Tests erfolgt gleichzeitig (Während Tester das Produkt kennenlernt)
- Anwendungsbereich
  - Für erste Stichproben, ob Software funktioniert
  - Wenn wenig Zeit zum Testen verfügbar
- Probleme
  - ***Tests kaum wiederholbar***, da kaum dokumentiert (Problem für Debugging!)
  - Überdeckung mit Tests kaum sicherzustellen
  - Abhängig von Erfahrung der Anwendungsentwickler / Tester
- Wichtig: ***Schreiben Sie mit***, wenn Sie explorativ vorgehen!

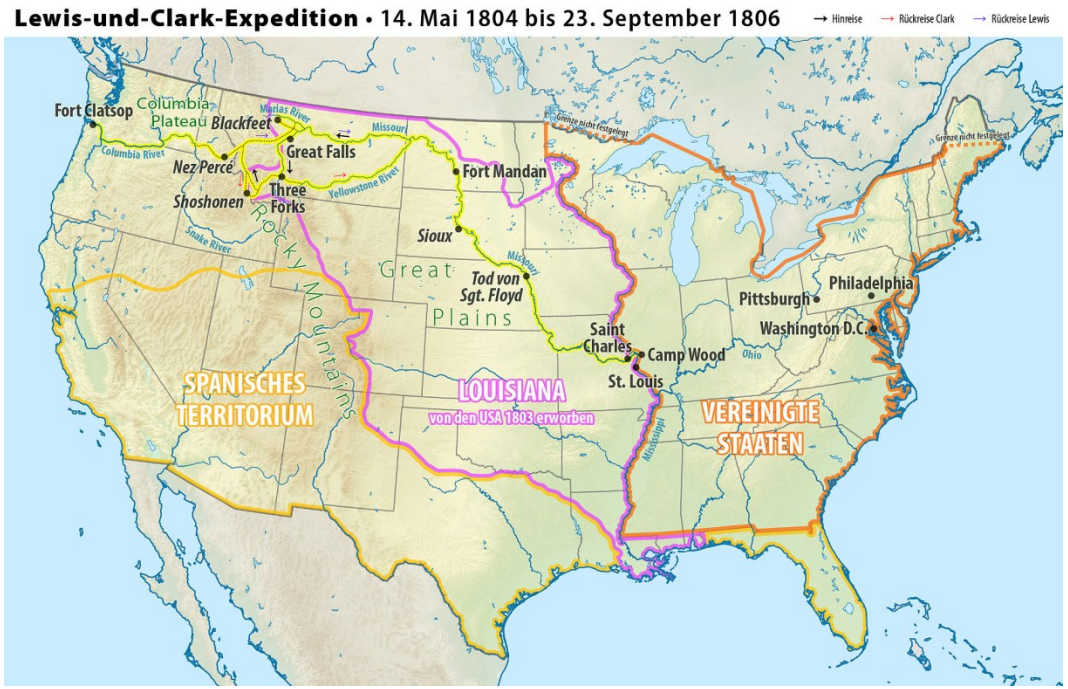
Beispiele für Heuristiken: Zählbare Dinge variieren

- Zahl der Benutzer, Zahl der Zeilen einer Datei, Zahl der Rückgabewerte einer Query, Zahl der Kontakte bei Facebook

- Test: Null / Eins /Viele
  - Bei Abfragen: Kein Treffer, genau ein Treffer, viele Treffer
- Test: Zu Viele
  - Überschreitung der erlaubten Höchstgrenze: Zu viele Kontakte, zu viele Verbindungen,
- Test: Zu Wenige
  - Mindestzahl unterschreiten: Leere Datei, Leer String, kein Drucker installiert, kein Kontakt bei Facebook, zu wenig Sonderausstattung im Auto

Verzetteln verhindern: Charter (Forschungsplan)

Im Auftrag von Thomas Jefferson: Expedition von Lewis und Clark entlang des Missouri, 1804



Quelle: Von Maximilian Dörrbecker (Chumwa) - Eigenes Werk, using File:Usa_edcp_relief_location_map.png by Uwe Dedering (Wikipedia) CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=40566017

Auftrag (***Charter***)

- ***Wo*** forschen?
  - Entlang des Missouri
- ***Ausstattung***
  - Boote, Zelte, Messinstrumente, Waffen, …
- ***Gesuchte Information***
  - Handelswege: direkte und zweckmäßige Wasserwege für den Handel

Vorlage für Charter

```
Erforschen von
<Ziel>
mit <Hilfsmittel>
um <Information>
zu finden.
```

Idee: ***Gezieltes erforschen*** von Eigenschaften der Software, keine planlose Suche

- ***Ziel***: Bereich eingrenzen, zu erforschende Funktion / zu erforschende Eigenschaft des Systems
- ***Hilfsmittel***: Werkzeuge (z.B. Profiler, Monitoring-Tool, Konsole, …), Datensätze, Techniken, Konfigurationen, Hardware …
- ***Informationen***: Wonach genau wird gesucht? Sicherheit? Geschwindigkeit? Korrektheit? Robustheit? Konsistenz? … Welches Risiko wird erforscht?

Beispiele

- Erforschen der Aktualisierung von Profilen (GUI)
  - mit Code Injection (SQL-Inj., XSS, …)
  - um Schwachstellen in der Sicherheit zu entdecken

  gut
- Erforschen der Aktualisierung von Profilen
  - mit verschiedenen Authentifizierungsmethoden,
  - um Überraschungen zu finden

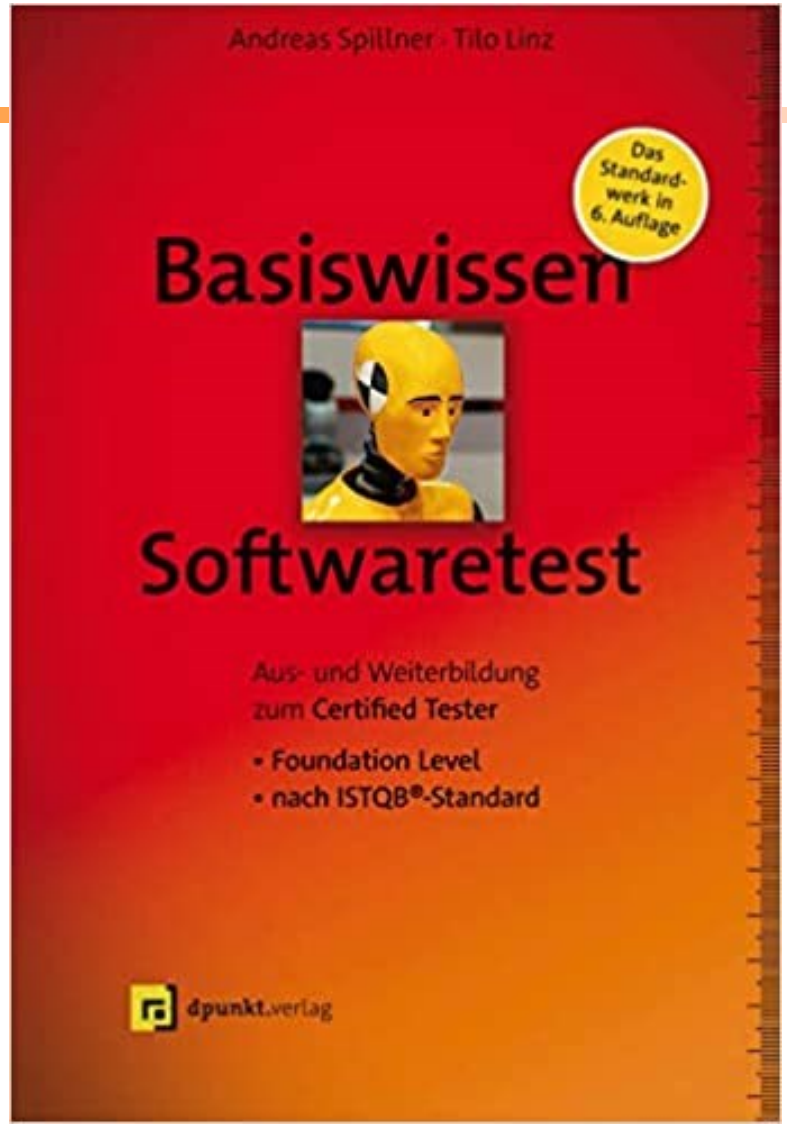
- Erforschen der Änderung des Nachnamens
  - mit dem Wert O´Beneken
  - um festzustellen, ob die Funktion zum Ändern des Profils Namen mit Apostroph verarbeitet werden kann

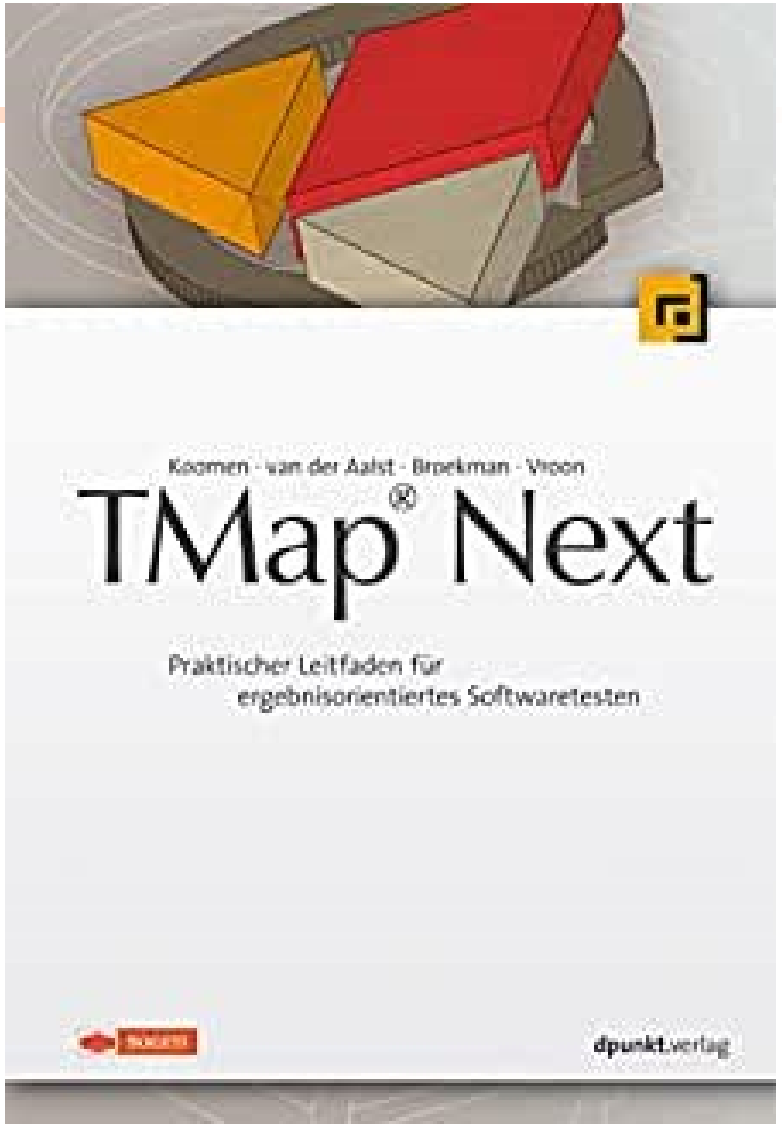
  zu speziell
- Erforschen der Systemsicherheit
  - mit allen möglichen Hacking-Programmen,
  - um alle Sicherheitslücken zu finden

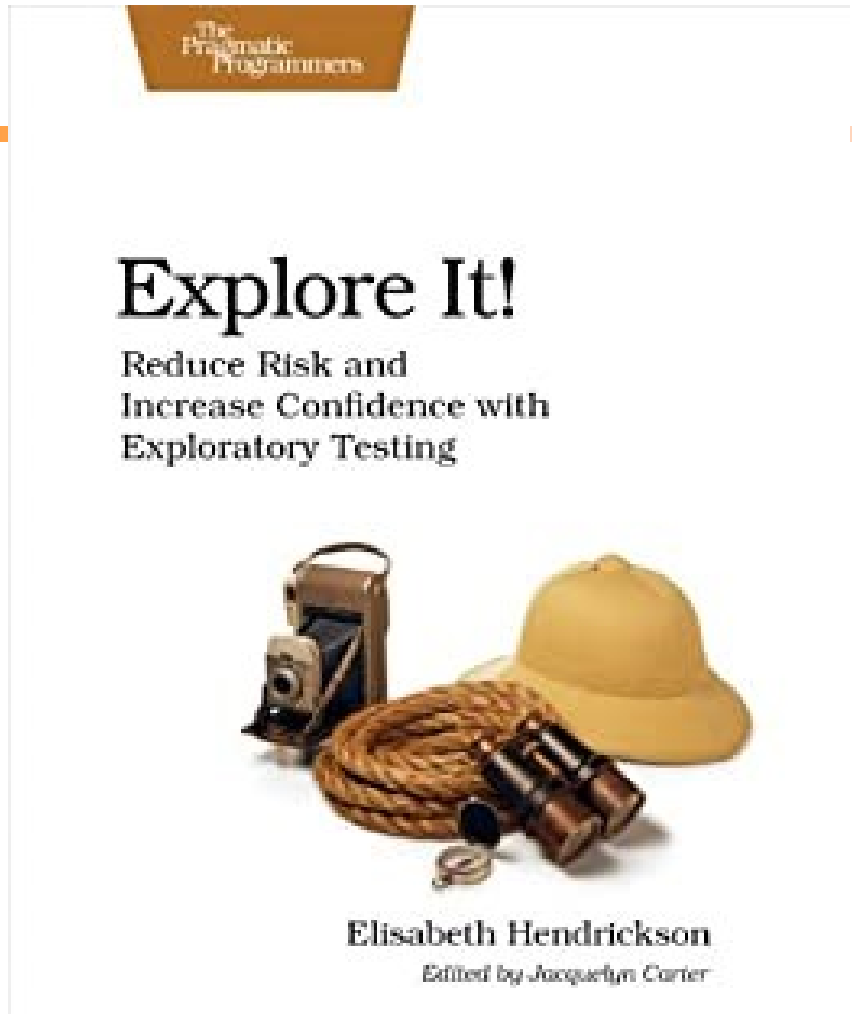
  zu allgemein

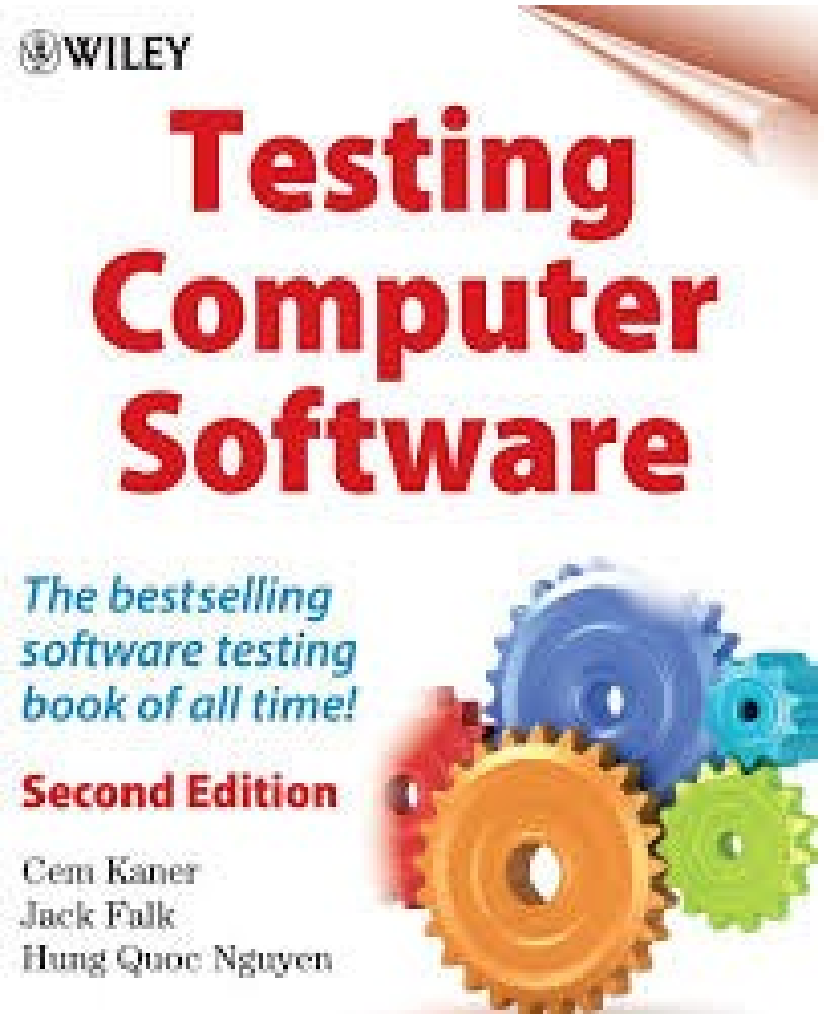
Charter finden

- Ableiten aus den ***Anforderungen*** (speziell aus ***Diskussionen*** zu Anforderungen während der Verfeinerung)
  - Beim Definieren von Charters werden idR. neue Anforderungen entdeckt
  - Beim Durchlaufen der Charters werden idR. neue Anforderungen gefunden (da Anforderungen/Spezifikation selten vollständige Verhaltensbeschreibungen)
- ***Implizite Erwartungen*** (vgl. Kano) wie Verlässlichkeit, Konsistenz, Robustheit
- ***Horrorszenarios*** brainstormen


Andreas Spillner · Tilo Linz
Das Standard-werk in 6. Auflage
Basiswissen
Softwaretest
Aus- und Weiterbildung zum Certified Tester
• Foundation Level
• nach ISTQB®-Standard
dpunkt.verlag


Koomen · van der Aalst · Broekman · Vroon
TMap® Next
Praktischer Leitfaden für ergebnisorientiertes Softwaretesten
dpunkt.verlag


The Pragmatic Programmers
Explore It!
Reduce Risk and
Increase Confidence with
Exploratory Testing
Elisabeth Hendrickson
Edited by Jacquelyn Carter


WILEY
Testing
Computer
Software
The bestselling
software testing
book of all time!
Second Edition
Cem Kaner
Jack Falk
Hung Quoc Nguyen