Softwaretechnik und Programmierparadigmen

VL11: Testen

Prof. Dr. Sabine Glesner
FG Programmierung eingebetteter Systeme
Technische Universität Berlin

Qualitätssicherung

- Konstruktive Maßnahmen:
  - Qualität wird in das System hineinkonstruiert
  - Z.B. durch Prozessmodelle, Spezifikationsmethoden, spezielle Sprachen
- Analytische Maßnahmen:
  - Qualität einer Implementierung wird geprüft
  - Z.B. durch Inspektion, Simulation, Testen, formale Verifikation

Analytische Maßnahmen

- Formale Verifikation
  - Z.B. Model Checking (vergangene Vorlesung: MC jABC) oder formaler Korrektheitsbeweis
  - Vorteile: theoretisch ideal, kann Korrektheit beweisen
  - Nachteile: benötigt vollständige formale Anforderungsspezifikation, Komplexitätsproblem, fehlende Automatisierung

Analytische Maßnahmen

- Testen:
- „Program testing can be used to show the presence of bugs, but never to show their absence.“

  (Edsgar W. Dijkstra, 1970)
- Nur Fehlererkennung, keine Garantie für Korrektheit
- Systematisch Testen
- Möglichst vollständig testen

Testplanung

- Wer testet:
  - Entwickler oder unabhängige Tester
- Was wird getestet?
  - Testbare Teile der Anforderungen oder der Spezifikation identifizieren
- Wann wird getestet?
  - Stellen im Entwicklungsprozess
- Wie wird getestet?
  - Tools, Testarten, Bewertungskriterien

Testbare Anforderungen

- Testorakel:
  Was ist das erwartete Ergebnis für den Testfall?
  - Kann aus Spezifikation bzw. Anforderungen abgeleitet werden
- Problem: Quantifizierung von Anforderungen:

  nicht: „akzeptable Antwortzeiten sind wichtig“

  sondern: „Antwortzeit höchstens 5 Sekunden,
  in 80% der Fälle kleiner als 3 Sekunden“

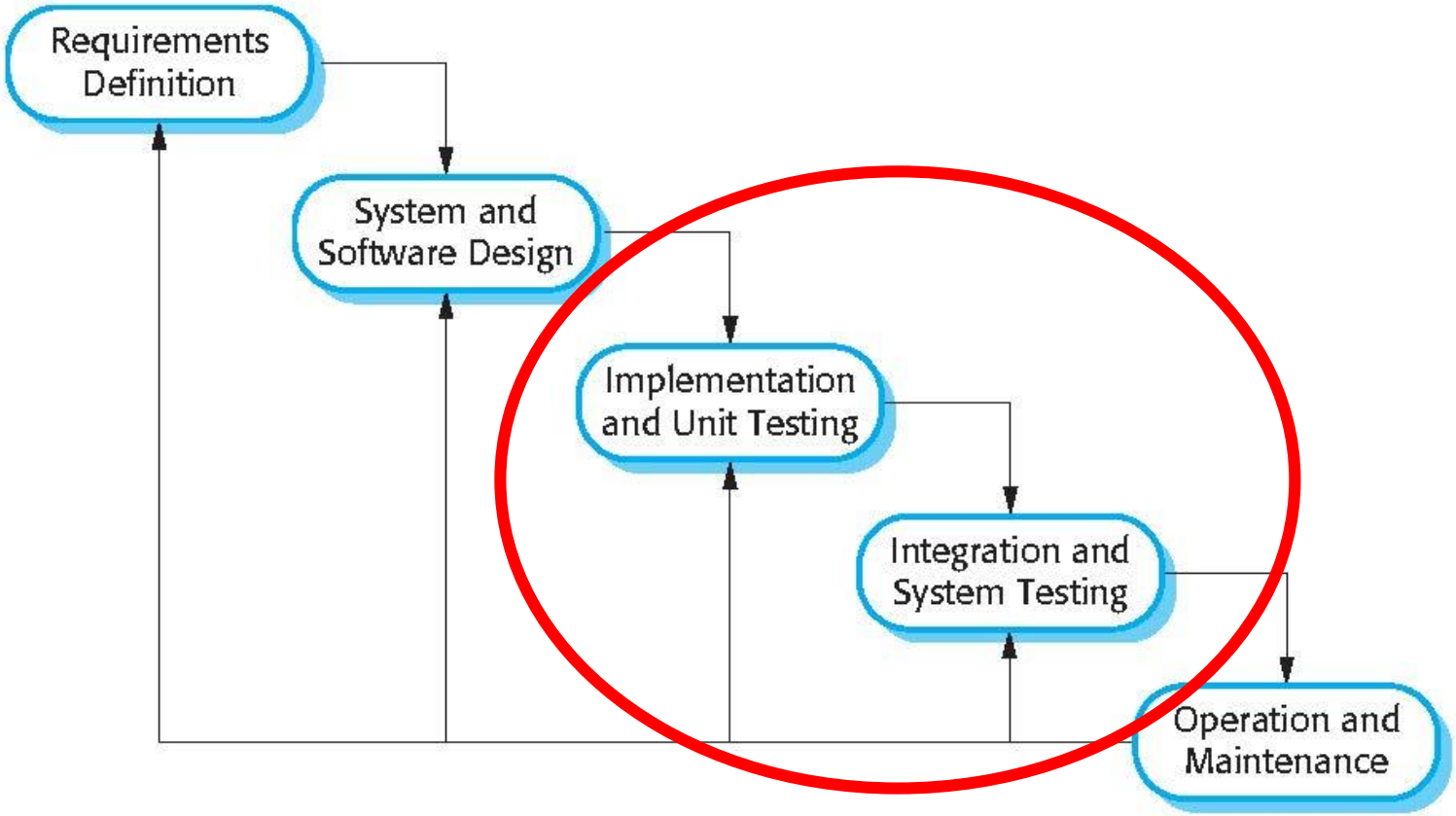
Übersicht

- Testen im Entwicklungsprozess
  - Modultests
  - Integrationstests
  - Systemtests
- Testauswahl
- Strukturorientierte Tests (White-Box Tests)
- Funktionsorientierte Tests (Black-Box Tests)
- Testbericht
- Standards

Übersicht

- Testen im Entwicklungsprozess
  - Modultests
  - Integrationstests
  - Systemtests
- Testauswahl
- Strukturorientierte Tests (White-Box Tests)
- Funktionsorientierte Tests (Black-Box Tests)
- Testbericht
- Standards

Testen im Entwicklungsprozess (Beispiel Wasserfallmodell)



Modultests (Unit Testing)

- Ziele: Aufdeckung von falschen Berechnungen, fehlenden Randfällen, prüfen der Robustheit gegenüber falscher Anwendung, Verständlichkeit von Fehlermeldungen, …

- Erstmaliger Test einzelner Komponenten
- Jeder Baustein wird unabhängig getestet!
  - keine externen Einflüsse oder Wechselwirkungen
  - klare Fehlereingrenzung
  - gute Fehlerlokalisierung

Integrationstests

- Ziel: Prüfung der Interaktion zwischen Modulen über Schnittstellen

- Kontrollflussorientierter Integrationstest
  - jede Funktion die ein Modul nach außen zur Verfügung stellt sollte mindestens einmal aufgerufen werden
- Datenflussorientierter Integrationstest
  - Übergabeparameter und globale Variablen werden betrachtet
  - Überprüfung des (globalen) Datenflusses

Systemtests

- Black-Box-Sicht auf das System unter Test
- Funktionstest: Überprüfung aller in der Spezifikation geforderten Systemeigenschaften
- Lasttest: Wie verhält sich das System bei maximaler Last?
- Stresstest: Wie verhält sich das System bei Überlast?
- Robustheitstest: Wie verhält sich das System bei Ausfällen o. anormalen Umgebungsbedingungen?
- Regressionstests: Wiederholung von Testfällen bei Versionsentwicklung

Übersicht

- Testen im Entwicklungsprozess
  - Modultests
  - Integrationstests
  - Systemtests
- Testauswahl
- Strukturorientierte Tests (White-Box Tests)
- Funktionsorientierte Tests (Black-Box Tests)
- Testbericht
- Standards

Testauswahl

- Problem: erschöpfende (*exhaustive*) Tests meist nicht möglich (kombinatorische Explosion der Eingaben)

Frage: Welche Eingaben sollen getestet werden?

Testauswahl

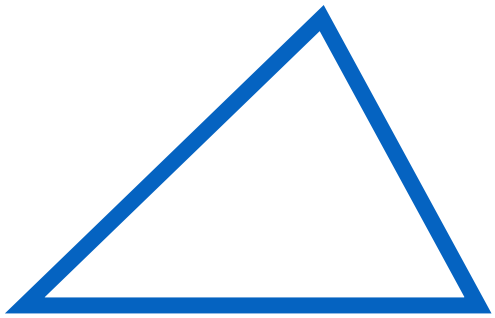
- Statischer Test:
  Test ohne Programmausführung
  - Review
  - Statische Code-Analysen
- Dynamischer Test:
  Test mit Programmausführung
  - Strukturorientierter Test

Testauswahl – Dynamische Tests

- Funktionsorientierter Test:
  Auswahl nach Eigenschaften der Eingabe oder der Spezifikation
  - Test typischer Anwendungsfälle (Nutzungsprofile)
  - gezieltes Testen von Rand- und Sonderfällen
- Strukturorientierter Test:
  Auswahl nach dem Aufbau des Moduls
  ⇨ Ziel: hoher Überdeckungsgrad auf der Struktur
  - Kontrollflussüberdeckung und Datenflussüberdeckung

Ein klassisches Beispiel (Myers)

- Funktion mit drei int-Eingabewerten (x,y,z), die als Längen von Dreiecksseiten interpretiert werden Berechnet, ob das Dreieck gleichseitig, gleichschenklig oder ungleichseitig ist



- Schreibt für diese Funktion Testfälle auf!

Auswertung (1)

1. Habt ihr einen Testfall für ein zulässiges gleichseitiges Dreieck?
2. Habt ihr einen Testfall für ein zulässiges gleichschenkliges Dreieck?
   (Ein Testfall mit 2,2,4 zählt nicht.)
3. Habt ihr einen Testfall für ein zulässiges ungleichseitiges Dreieck?
   (Beachtet, dass Testfälle mit 1,2,3 und 2,5,10 keine Ja-Antwort garantieren, da kein Dreieck mit solchen Seiten existiert.)
4. Habt ihr wenigstens drei Testfälle für zulässige, gleichschenklige Dreiecke, wobei ihr alle drei Permutationen der beiden gleichen Seiten berücksichtigt habt?
   (z.B. 3,3,4; 3,4,3; 4,3,3)

Auswertung (2)

5. Habt ihr einen Testfall, bei dem eine Seite gleich Null ist?
6. Habt ihr einen Testfall, bei dem eine Seite einen negativen Wert hat?
7. Habt ihr einen Testfall mit drei ganzzahligen Werten, in dem die Summe zweier Zahlen gleich der dritten ist?
   (D.h., wenn das Programm 1,2,3 als ungleichseitiges Dreieck akzeptiert, so enthält es einen Fehler.)
8. Habt ihr wenigstens drei Testfälle für Punkt 7, wobei ihr alle drei Permutationen für die Länge jeweils einer Seite als Summe der beiden anderen Seiten berücksichtigt habt?
   (Z.B. 1,2,3; 1,3,2; 3,1,2.)

Auswertung (3)

9. Habt ihr einen Testfall mit drei ganzzahligen Werten größer Null, bei dem die Summe aus zwei Zahlen kleiner als die dritte ist? (Z.B. 1,2,4 oder 12,15,30)
10. Habt ihr wenigstens drei Testfälle für Punkt 9, wobei ihr alle drei Permutationen berücksichtigt habt? (Z.B. 1,2,4; 1,4,2; 4,1,2.)
11. Habt ihr einen Testfall, in dem alle drei Seiten gleich Null sind? (D.h. 0,0,0)
12. Habt ihr wenigstens einen Testfall mit nichtganzzahligen Werten?
13. Habt ihr wenigstens einen Testfall, in dem ihr eine falsche Anzahl von Werten angebt ? (Z.B. zwei statt drei ganzzahlige Werte)?

Zusatzpunkte

14. Habt ihr zusätzlich zu jedem Eingangswert in allen Testfällen die erwarteten Ausgabewerte angegeben?
15. Habt ihr Tests mit maximalen Werten?
16. Habt ihr Tests auf Zahlbereichs-Überlaufbehandlung?
17. Habt ihr Tests mit unzulässigen Eingabezeichenfolgen?

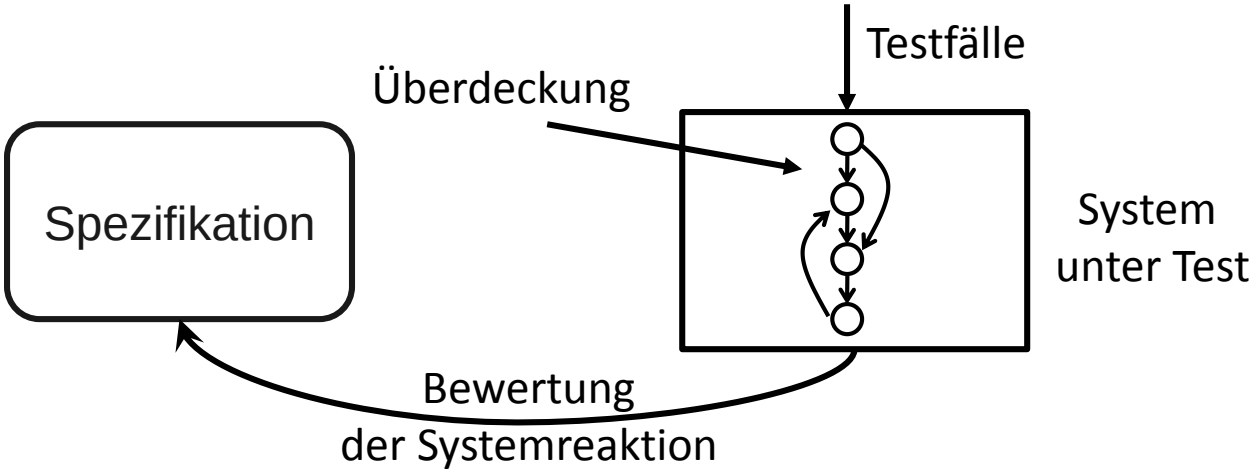
Zusatzpunkte

- Jede mit “ja” beantwortete Frage gibt einen Punkt
- Durchschnittswerte erfahrener Programmierer: 7-8 Punkte
- Selbst für dieses sehr kleine Beispiel haben die meisten Entwickler Probleme, Tests mit hoher Überdeckung anzugeben

Übersicht

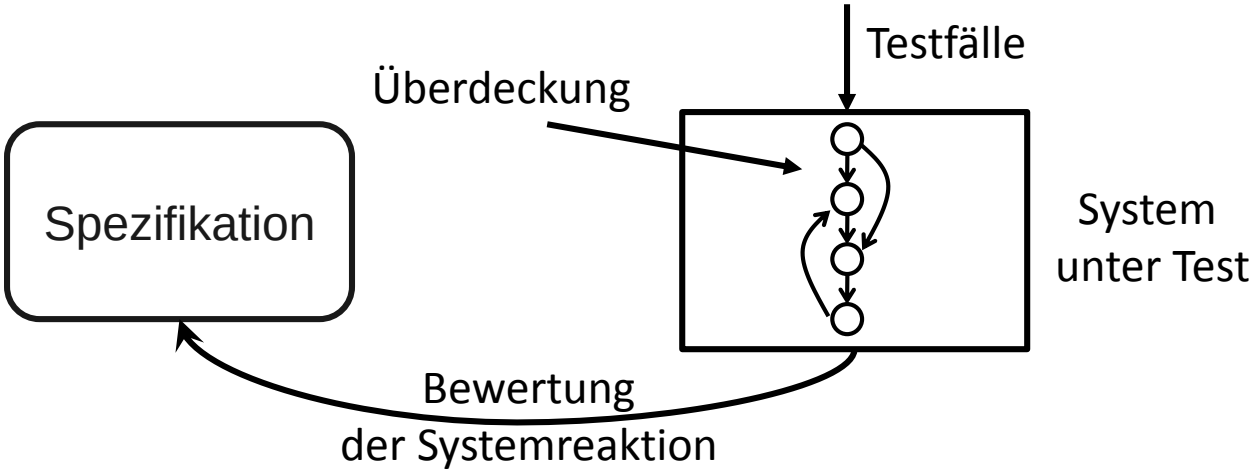
- Testen im Entwicklungsprozess
  - Modultests
  - Integrationstests
  - Systemtests
- Testauswahl
- Strukturorientierte Tests (White-Box Tests)
- Funktionsorientierte Tests (Black-Box Tests)
- Testbericht
- Standards

Strukturorientierter Test



- Auch White-Box-Tests genannt
  - Interner Aufbau muss vorliegen
  - Testfälle können erst nach der Implementierung aufgestellt werden
  - Testfälle ergeben sich aus der Spezifikation und Struktur der Software

Strukturorientierter Test



- Methode zur Testauswahl:
  - Erzielung eines maximalen Überdeckungsgrades
  - ⇨ hinzufügen von Testfällen, die den Überdeckungsgrad erhöhen
  - Keine garantierte Abdeckung der Spezifikation
  - Das Fehlen von Programmcode wird nur zufällig erkannt

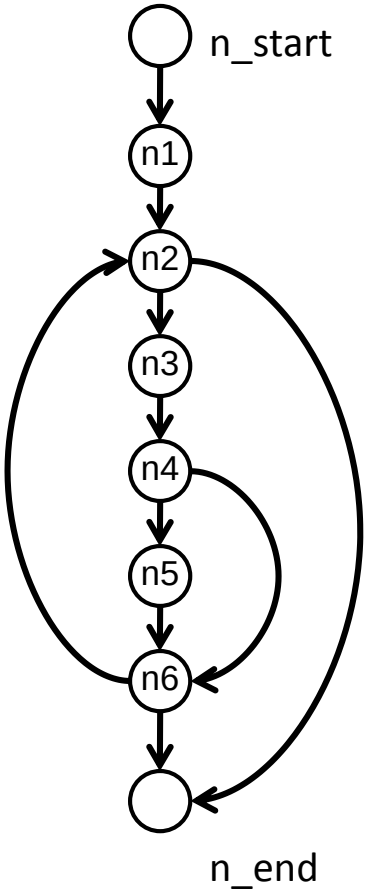
Überdeckungsmaße

- Allgemein:
  $$\text{Überdeckung} = \frac{\text{Anzahl überdeckter Merkmale}}{\text{Anzahl vorhandener Merkmale}}$$
- Datenflussüberdeckung:
  - Jedes definition-use pair wurde einmal ausgeführt
- Kontrollflussüberdeckung
  - Anweisungsüberdeckung: jede Anweisung (Knoten im Kontrollflussgraph) wurde einmal ausgeführt
  - Zweigüberdeckung: jeder Zweig (Kanten im Kontrollflussgraph) wurde einmal ausgeführt
  - Pfadüberdeckung: jeder Pfad im Kontrollflussgraphen wurde einmal ausgeführt
  - Bedingungsüberdeckung

Beispielfunktion: Zeichen zählen

```
void ZaehleZchn(int& VokalAnzahl, int& Gesamtzahl)
{
  char Zchn;
  cin >> Zchn;
  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;
  }
}
```

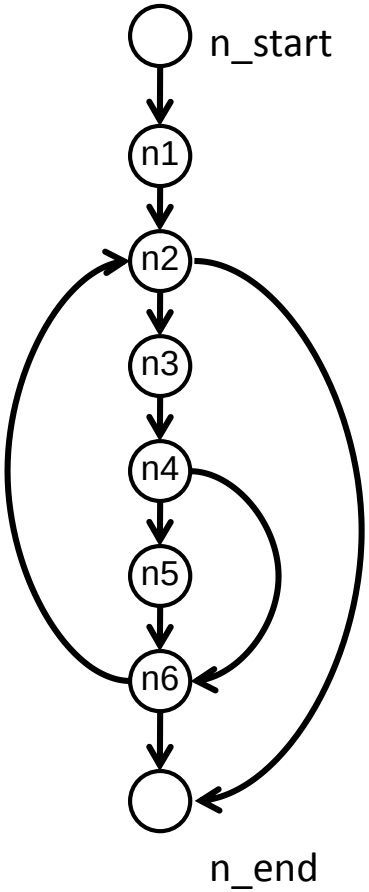
Kontrollflussgraph




```
cin >> Zchn;

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;
  }
}
```

Einfache Überdeckungskriterien

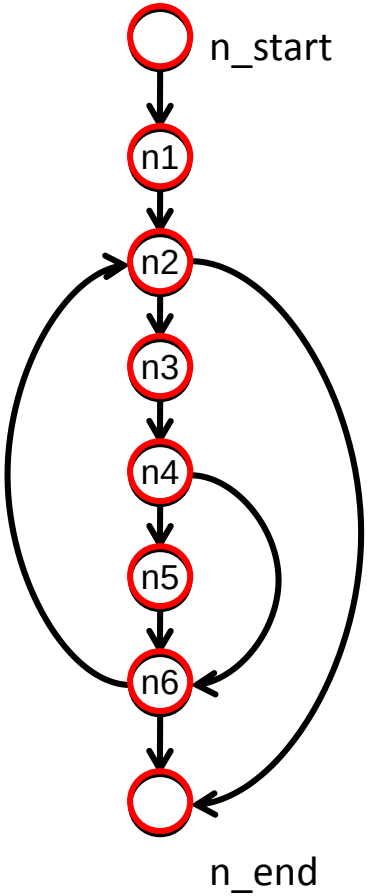


Eingabe: A1 → Anweisungsüberdeckung?

```
cin >> Zchn;

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;
  }
}
```

Einfache Überdeckungskriterien

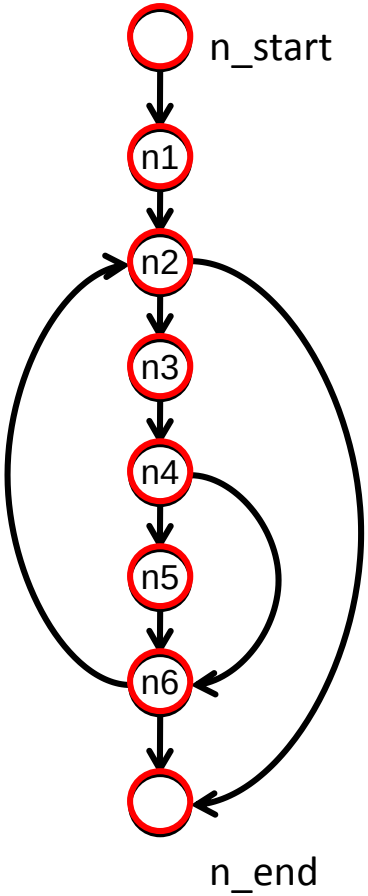


Eingabe: A1 → Anweisungsüberdeckung?

```
cin >> Zchn;

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;
  }
}
```

Einfache Überdeckungskriterien

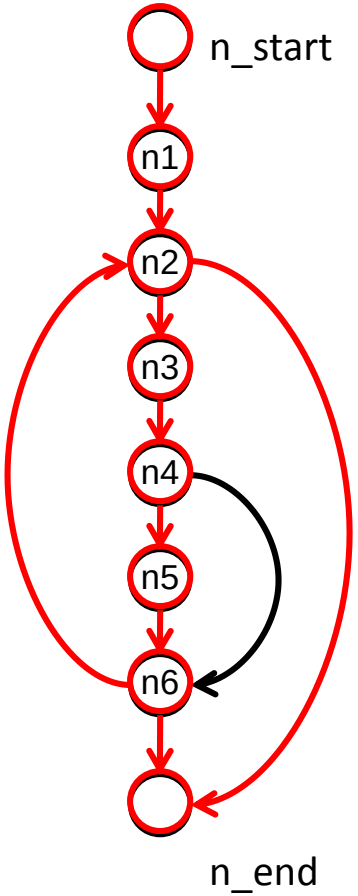


Eingabe: A1 → Zweigüberdeckung?

```
cin >> Zchn;

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;
  }
}
```

Einfache Überdeckungskriterien

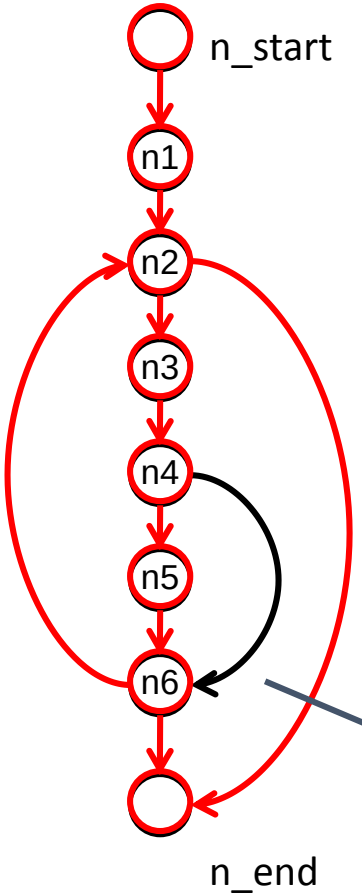


Eingabe: A1 → Zweigüberdeckung?

```
cin >> Zchn;

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;
  }
}
```

Einfache Überdeckungskriterien



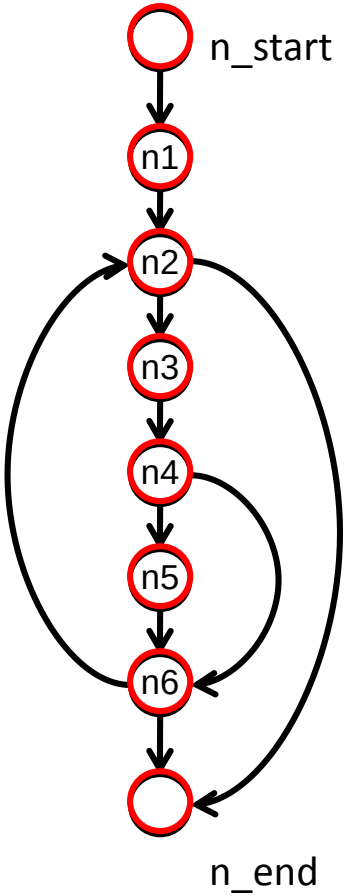
Eingabe: A1 → Zweigüberdeckung?

```
cin >> Zchn;

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;
  }
}
```

wird nicht ausgeführt

Einfache Überdeckungskriterien

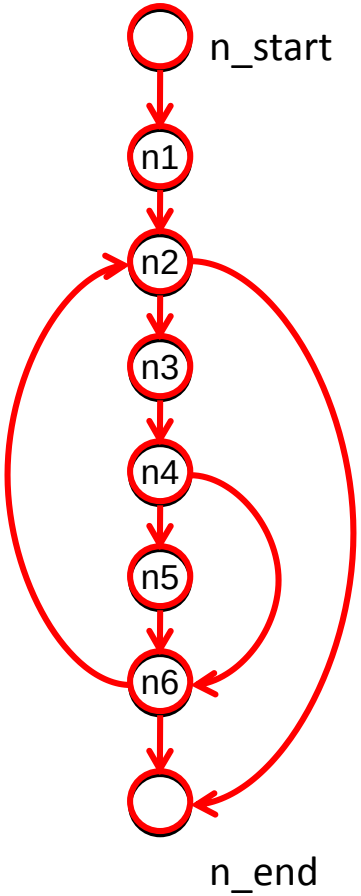


Eingabe: AB1 → Zweigüberdeckung?

```
cin >> Zchn;

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;
  }
}
```

Einfache Überdeckungskriterien



Eingabe: AB1 → Zweigüberdeckung?

```
cin >> Zchn;

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;
  }
}
```

Anweisungs- und Zweigüberdeckung

- Anweisungsüberdeckung
  - Notwendiges, aber nicht hinreichendes Testkriterium
  - Kann Code finden, der nicht ausführbar ist
  - Als eigenständiges Testverfahren nicht geeignet
  - Fehleridentifizierungsquote: 18%
- Zweigüberdeckung
  - Gilt als *das minimale Testkriterium*
  - Kann nicht ausführbare Programmzweige finden
  - Kann häufig durchlaufene Programmzweige finden (Optimierung)
  - Fehleridentifikationsquote: 34%

Pfadüberdeckung

- Falls 32767 der größte Integer-Wert ist, hat die Funktion ZaehleZchn 232768-1 Pfade
- Das sind etwa 1,41 * 109864 Pfade
- pro Sekunde ein Testfall: 4,5 * 109853 Jahre
- Zum Vergleich: Das Alter der Erde wird mit etwas mehr als 4,5 ·109Jahren angegeben

➢ Wichtigste Variante: strukturierter Pfadtest

  - Jede Schleife wird mindestens k-mal durchlaufen
  - Fehleridentifikationsquote: um 65%

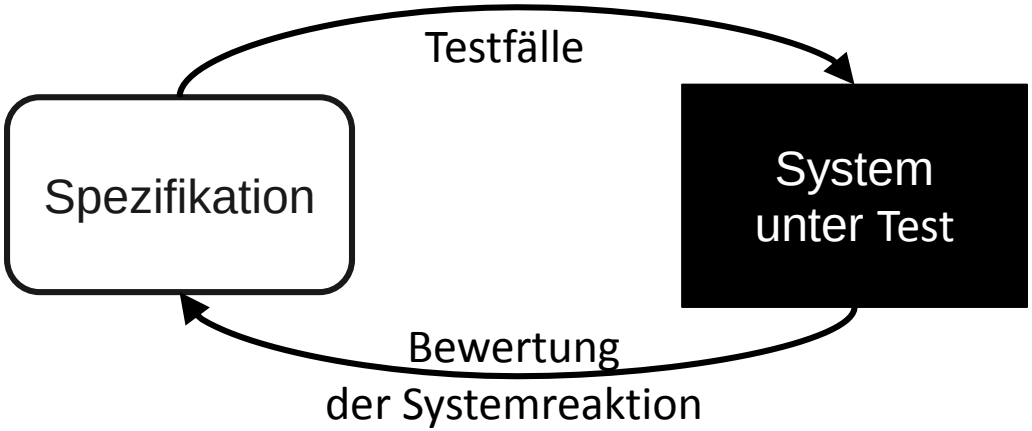
Bedingungsüberdeckung

- Einfache Bedingungsüberdeckung
  - jede atomare Teilentscheidung muss einmal *true* und *false* sein
- Mehrfache Bedingungsüberdeckung
  - jede mögliche Kombination atomarer Teilentscheidungen
- Minimale Mehrfach-Bedingungsüberdeckung:
  - Jede atomare Bedingung muss einmal *true* und *false* sein
  - Die Gesamt-Bedingung muss wenigstens einmal *true* und wenigstens einmal *false* werden
- Modifizierte Bedingungs-/Entscheidungsüberdeckung:
  - Die Testfälle müssen demonstrieren, dass jede atomare Teilentscheidung den Wahrheitswert der Gesamtentscheidung unabhängig von den anderen Teilentscheidungen beeinflussen kann

Übersicht

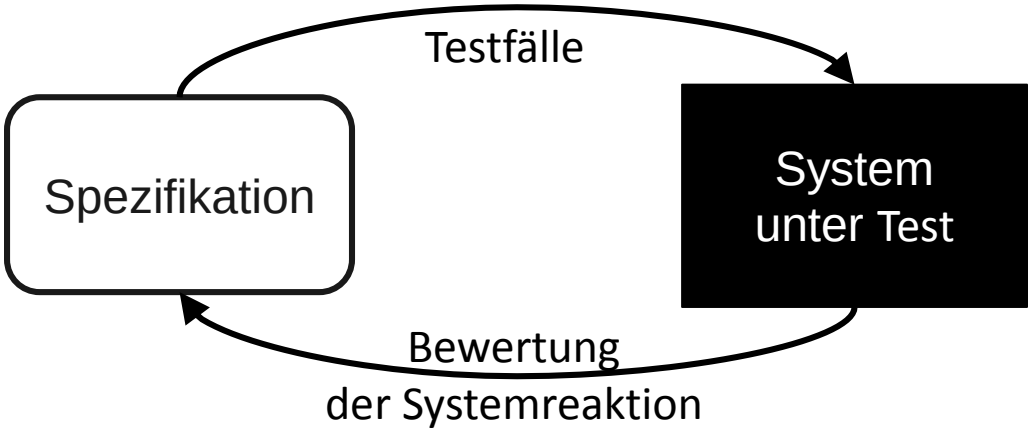
- Testen im Entwicklungsprozess
  - Modultests
  - Integrationstests
  - Systemtests
- Testauswahl
- Strukturorientierte Tests (White-Box Tests)
- Funktionsorientierte Tests (Black-Box Tests)
- Testbericht
- Standards

Funktionsorientierter Test



- Auch Black-Box-Tests genannt
  - Interner Aufbau muss nicht vorliegen
  - Testfälle können schon vor der Implementierung aufgestellt werden
  - Testfälle ergeben sich aus der Spezifikation der Software

Funktionsorientierter Test



- Methoden zur Testauswahl:
  - Äquivalenzklassenbildung
    - Auswahl “repräsentativer Daten”
    - z.B. durch Datenanalysen, Timing-Analysen, Pre/Post Analysen, Pfadbedingungen, …
  - Grenzwertanalyse
    - Wertebereiche, min/max Werte
  - Entscheidungstabellen und Klassifikationsbäume

Äquivalenzklassenbildung

- 1. Schritt: Partitionierung des Eingabedatenraumes in eine endliche Zahl von Äquivalenzklassen (bezüglich des vermuteten Ausfallverhaltens)
  - im Beispiel: „drei gleiche Eingaben größer Null“
- 2. Schritt: Auswahl der Testfälle anhand je eines Repräsentanten der Äquivalenzklasse
  - im Beispiel: (2,2,2)

Äquivalenzklassenbildung

- Wertebereiche für Ein- und Ausgabewerte betrachten
- Für jeden Wert ergeben sich gültige und ungültige Klassen
  - Werte innerhalb und außerhalb des Wertebereichs
  - Eingaben, die unterschiedlich verarbeitet werden
  - Ausgaben, die unterschiedlich berechnet werden
- Äquivalenzklassen sollten weiter aufgespalten werden, falls Grund zur Annahme besteht, dass nicht alle Elemente gleich behandelt werden

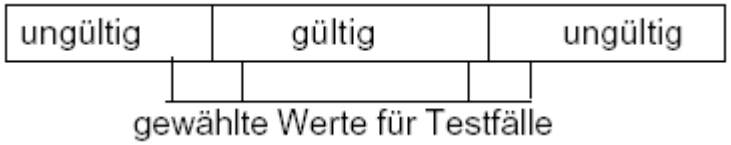
  Beispiel: Eingabebereich $1 \leq \text{Wert} \leq 99$

  gültige Äquivalenzklasse: [1,99]

  ungültige Äquivalenzklassen: < 1 und > 99

Grenzwertanalyse

- Gezielte Betrachtung der Grenzen von Wertebereichen
- Im Unterschied zur Äquivalenzklassenbildung wird kein beliebiger Repräsentant der Klassen als Testfall ausgewählt, sondern Repräsentanten an den „Rändern“ der Klassen



- In Verzweigungen und Schleifen gibt es oft Grenzwerte, für die die Bedingung gerade noch zutrifft (oder gerade nicht mehr)
- Rückwärtsanalyse, um Eingabedaten zu erhalten die diese Grenzwerte erreichen ⇨ Pfadbedingungen

Entscheidungstabelle

- Betrachtung aller möglichen Kombinationen von Eingaben (bzw. Repräsentanten von Äquivalenzklassen)

Bedingungen	Erstkunde?	N	N	N	N	J	J	J	J
	Bestellwert > 1000	N	N	J	J	N	N	J	J
	Privatkunde?	N	J	N	J	N	J	N	J
Aktion	Zahlung per Rechnung möglich	J	J	J	J	J	J	J	N

- Problem: exponentielles Wachstum der Anzahl an Spalten
- Lösung: optimierte Entscheidungstabelle

Bedingungen	Erstkunde?	N	-	-	J
	Bestellwert > 1000	-	-	N	J
	Privatkunde?	-	N	-	J
Aktion	Zahlung per Rechnung möglich	J	J	J	N

Übersicht

- Testen im Entwicklungsprozess
  - Modultests
  - Integrationstests
  - Systemtests
- Testauswahl
- Strukturorientierte Tests (White-Box Tests)
- Funktionsorientierte Tests (Black-Box Tests)
- Testbericht
- Standards

Testbericht

- Testprotokoll
  - Testfälle, Ergebnisse und Abweichungen von erwarteten Ergebnissen
- Testvorfallbericht
  - Beschreibt gefundene Fehler mit Ein- und Ausgaben
- Testergebnisbericht
  - Zusammenfassung der Tests inklusive Ergebnisse und Bewertung nach zuvor festgelegten Testkriterien

Übersicht

- Testen im Entwicklungsprozess
  - Modultests
  - Integrationstests
  - Systemtests
- Testauswahl
- Strukturorientierte Tests (White-Box Tests)
- Funktionsorientierte Tests (Black-Box Tests)
- Testbericht
- Standards

Standards und Zertifizierung

- ISO/IEC/IEEE 29119:2013:
  - Standard für Testen von Software
  - Veröffentlicht September 2013
  - Ersetzt eine Reihe von existierenden Standards über Softwaretests (z.B. IEEE 829)
- ISTQB Zertifizierung
  - International Software Testing Qualifications Board
  - Bietet Zertifizierungen für Softwaretester an

Zusammenfassung

- Testen ist eine analytische Maßnahmen zur Qualitätssicherung
- Testplanung: Wer mach was wann und wie?
- Testen ist Bestandteil des Entwicklungsprozesses
- Modultests für Komponenten
- Integrationstests für Zusammenspiel
- Systemtests aus Black-Box-Sicht
- Testauswahl: funktions- oder strukturorientiert
- Strukturorientiert:
  - Überdeckungsmaße
    - Datenfluss, Kontrollfluss (Anweisung, Zweig, Pfad, Bedingung)
- Funktionsorientiert:
  - Äquivalenzklassen, Grenzwertanalyse, Entscheidungstabelle
- Testbericht
- Standards und Zertifizierungen

Quellen

- http://www.istqb.org/, International Software Testing Qualifications Board
- Peter Liggesmeyer, ***Software-Qualität***

  Testen, Analysieren und Verifizieren von Software

  Spektrum, 2002
- Glenfield J. Myers ***The Art of Software Testing***
  Wiley, 1979.