Software Engineering II (IB)
Testen von Software / Modultests

Prof. Dr. Oliver Braun

Fakultät für Informatik und Mathematik
Hochschule München

Letzte Änderung: 16.05.2017 21:17

Inhaltsverzeichnis

Programm-Tests

- Tests sollen zeigen, dass ein Programm das tut was es tun soll
- sowie Programmfehler vor der Benutzung aufdecken
- beim Testen nutzen wir künstliche Daten
- wir prüfen die Ergebnisse von Testläufen auf Fehler, Anomalien oder Informationen über die nicht-funktionalen Attribute des Programms
- Testen kann nur Fehler finden, nicht ihre Abwesenheit
- Testen ist Teil eines breiteren Prozesses der Softwareverifikation und -validierung

Ziele des Testens

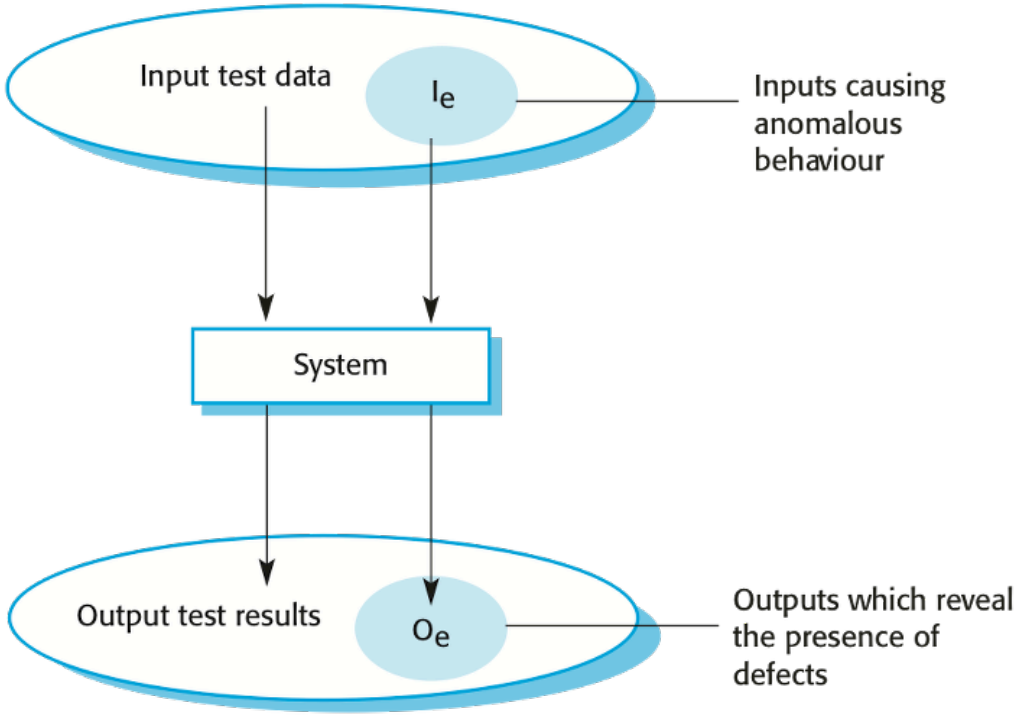
1. Dem Entwickler und dem Kunden zeigen, dass die Software die Anforderungen erfüllt.

   ⇒ **Validierungstests**

2. Situationen aufspüren, in denen sich die Software falsch, unerwünscht oder nicht spezifikationsgemäß verhält.

   ⇒ **Fehlertests**

Eingabe-Ausgabe-Modell von Programmtests



Eingabe-Ausgabe-Modell (Quelle: I. Sommerville: Software Engineering)

Keine Fehlerfreiheit durch Tests

Edsger Dijkstra et al., 1972:

> Tests können nur die Anwesenheit von Fehlern aufzeigen, nicht ihre Abwesenheit.

Verifikation vs. Validierung

Verifikation Erstellen wir das Produkt richtig?

Die Software muss die Spezifikation erfüllen!

Validierung Erstellen wir das richtige Produkt?

Die Software muss bieten was der Benutzer benötigt.

Ziele von Verifikation & Validierung

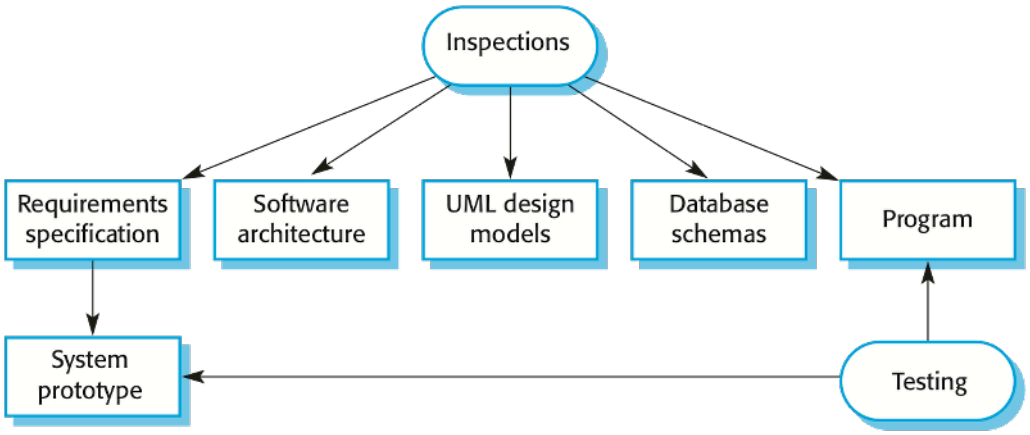
Softwarezweck Je kritischer eine Software, desto wichtiger die Zuverlässigkeit.

Erwartungen der Benutzer Benutzer manchmal geringe (z.B.neue Sw), manchmal hohe Erwartungen (z.B. spätere Versionen).
Marktsituation So schnell wie möglich auf den Markt?

Softwareinspektionen und Reviews

- Softwareinspektionen und Reviews analysieren und prüfen
  - Systemanforderungen
  - Entwurfsmodelle
  - Quellcode
  - Systemtests
- statische "V&V"-Techniken, bei denen die Software nicht ausgeführt wird

Inspektionen und Tests



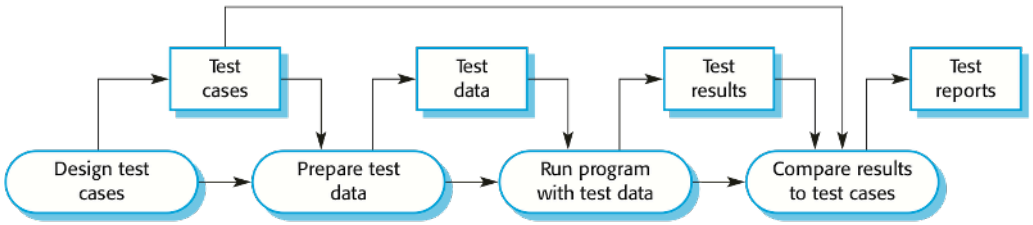
Inspektionen und Tests (Quelle: I. Sommerville: Software Engineering)

Vorteile von Inspektionen

1. Beim Testen können Fehler andere Fehler verdecken.
2. Unvollständige Systemversionen lassen sich ohne Zusatzkosten inspizieren.
3. Über die Fehlersuche hinaus kann der Inspektionsprozess weitgehende qualitative Programmeigenschaften betrachten, z.B.

- Übereinstimmung mit Standards
- Portierbarkeit
- Wartungsfreundlichkeit
- Ineffinzienzen
- unpassende Algorithmen
- schlechter Programmierstil

Modell des Softwaretestprozesses



Modell des Softwaretestprozesses (Quelle: I. Sommerville: Software Engineering)

Testphasen

1. Entwicklertests (*development testing*)
2. Freigabetests (*release testing*)
3. Benutzertests (*user testing*)

Entwicklertests

1. Modultests (*unit testing*)
2. Komponententests (*component testing*)
3. Systemtests (*system testing*)

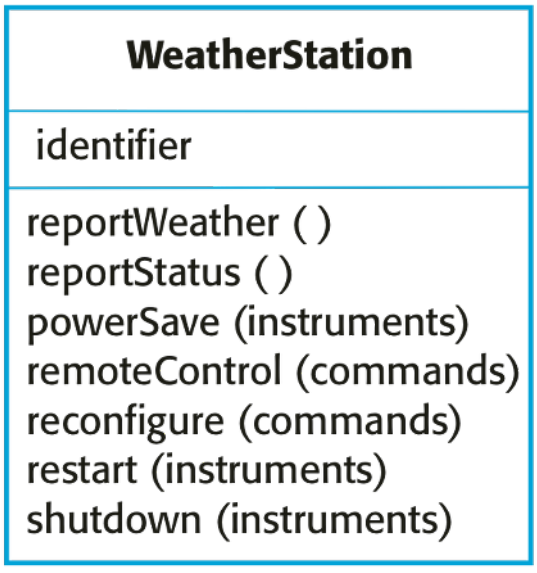
Modultests

- individuelle Komponenten in Isolation testen
- in erster Linie Fehlertests
- Module/Units können sein:
  - Funktionen, Methoden
  - Objektklassen
  - zusammengesetzte Komponenten

Objektklassentests

- Tests so gestalten, dass sie
  - alle definierten Operationen testen;
  - alle Attributwerte setzen und abfragen;
  - das Objekt in alle möglichen Zustände versetzen.
- Generalisierung und Vererbung macht das Testen komplexer, denn auch alle vererbten Member müssen in der Subklasse getestet werden

Beispiel: Eine Wetterstation



Klasse WeatherStation (Quelle: I. Sommerville: Software Engineering)

Wetterstation Tests

- Testfälle für alle Methoden
- Zustandsmodell um Folgen von Zustandsübergängen zu identizifieren
- Beispiele:
  - Shutdown → Running→ Shutdown
  - Configuring → Running → Testing → Transmitting → Running
  - Running → Collecting → Running → Summarizing → Transmitting → Running

Automatisierung

- Modul-Tests sollten immer automatisiert werden
- Testframeworks nutzen
- Testframeworks bieten oft auch graphische Darstellung von Testergebnissen o.ä.

Bestandteile Automatisierter Test

1. Aufbau: Initialisierung, Eingaben und Ausgaben festlegen
2. Aufruf: zu testende Methoden oder Objekte
3. Bewertung: Ergebnis des Aufrufs mit dem erwarteten Ergebnis

Auswahl der Testfälle für Modultests

- Testfälle sollen zeigen, dass bei korrekter Verwendung die zu testenden Komponenten, tun was sie sollen.
- Falls die Komponente Fehler enthält, sollen diese durch die Testfälle entdeckt werden.
- Zwei Arten von Testfällen:
  1. Normale Operationen abbilden.
  2. Ungewöhnliche Eingaben wo üblicherweise Probleme entstehen.

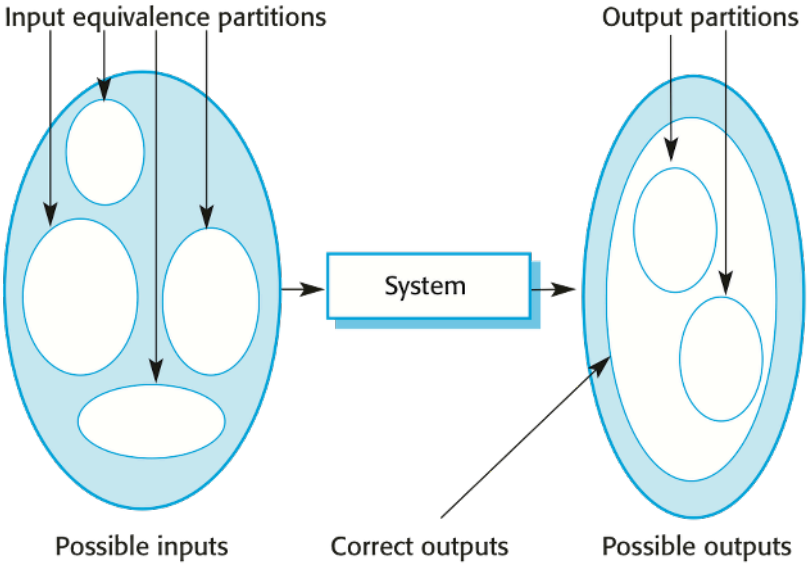
Strategien um Testfälle auszuwählen

1. Klassenbasierte Tests

- Gruppen von Eingaben identifizieren, die gemeinsame Merkmale haben und auf die gleiche Art und Weise verarbeitet werden.
- Tests aus jeder dieser Gruppen auswählen.

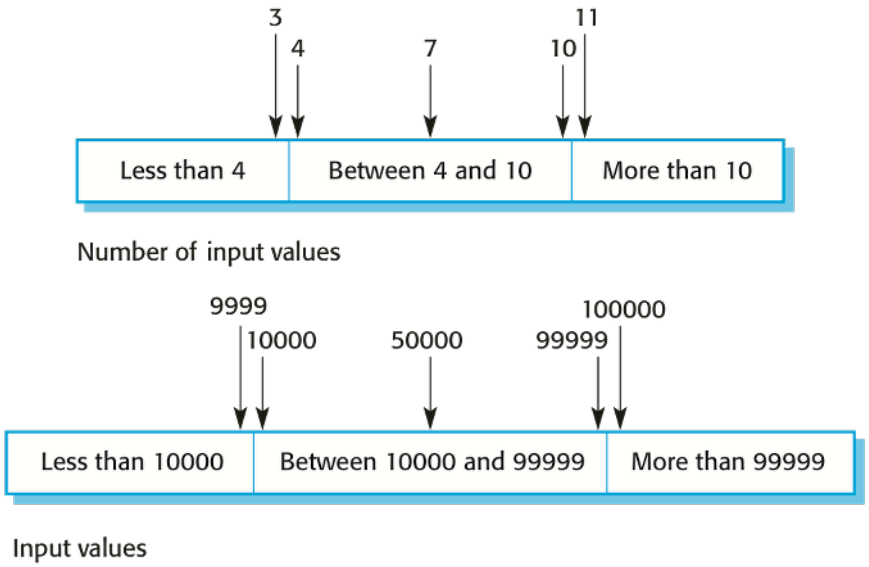
2. Richtlinienbasierte Tests
   - Richtlinien zur Auswahl von Tests
   - Richtlinien spiegeln vorausgegangene Erfahrungen wider

Äquivalenzklassen



Einteilung in Äquivalenzklassen (Quelle: I. Sommerville: Software Engineering)

Äquivalenzklassen



Äquivalenzklassen (Quelle: I. Sommerville: Software Engineering)

Black-/Whitebox-Testen

Blackbox-Testen nur Spezifikation für die Klassenbildung
Whitebox-Testen zusätzlich Quellcode nutzen

Richtlinien für Folgen, Listen, ...

- Folgen mit nur einem Wert
- Folgen mit verschiedener Größe
- leere Folgen
- weitere Tests ableiten, die die jeweils ersten, mittleren und letzten Elemente verarbeiten

Gebräuchliche Richtlinien

- Wählen Sie Eingaben, die das System zur Ausgabe aller Fehlermeldungen zwingen.
- Entwerfen Sie Eingaben, die den Eingabepuffer zum Überlaufen bringen.
- Wiederholen Sie dieselben Eingaben bzw. Eingabefolgen mehrere Male.
- Erzwingen Sie die Erzeugung ungültiger Ausgaben.
- Erzwingen Sie, dass die Berechnungsergebnisse zu groß oder zu klein ausfallen.

Pfadabdeckungstest

- Teststrategie mit Ziel:
  - Testen jedes unabhängigen Ausführungspfads eines Programms oder einer Komponentte
- jede bedingte Anweisung muss mit beiden Wahrheitswerten ausgeführt werden