Vorlesung „Softwaretechnik"

Testende Verfahren: Anmerkungen

Dr. Lutz Prechelt

Begriffe: Fehler, Defekt, Versagen

- Klares Denken verlangt, diese Begriffe genau auseinander zu halten
- Fehler (error):
  Falschtun eines Ingenieurs zur Entwicklungszeit
- Defekt (defect, „bug"):
  Strukturelles Falschsein eines Entwicklungsprodukts
- Versagen (failure, fault):
  Falsches Verhalten eines Systems zur Laufzeit

- Anmerkung: „falsch" kann etwas nach verschiedenen Maßstäben sein
  - gemäß einer eindeutigen Spezifikation
  - gemäß nicht eindeutig formulierter Erwartungen
    - Die evtl. nicht einmal bewusst sind

Begriffe: Fehler, Defekt, Versagen (2)

- Fehler und Versagen sind also Ereignisse,
  Defekte sind Zustände

- Fehler führen evtl. (aber nicht zwingend) zu Defekten
  Defekte führen evtl. (aber nicht zwingend) zu Versagen

- Die Schwere eines Defekts ist unabhängig von der Schwere des zugrunde liegenden Fehlers
- Die Schwere eines Versagens ist unabhängig von der Schwere des zugrunde liegenden Defekts
- Die Häufigkeit eines Versagens ist unabhängig von der Schwere des zugrunde liegenden Defekts

Fehlertoleranz, defensives Programmieren

- Ein Prozess ist fehlertolerant, wenn Fehler möglichst oft nicht zu Defekten im Produkt führen
- Ein Produkt ist fehlertolerant (eigentlich: defekttolerant), wenn Defekte möglichst oft nicht zu Versagen des Programms oder zu Defekten in Folgeprodukten führen
  - Auch Anforderungen, Entwürfe etc. Können defekttolerant sein
  - Robustheit: Defekttoleranz nach innen und nach aussen
    - Machmal auch: Versagen erlaubt, muss aber lokalisierbar sein
- Defensives Programmieren
  - Versucht, möglichst hohe Defekttoleranz zu erreichen
  - Oder zumindest ein leichtes und baldiges Aufdecken der Defekte (auch genannt Testbarkeit)
    - Z.B. durch ausführbare Zusicherungen im Programm
  - Auch das geht ähnlich für Anforderungen und Entwürfe

Begriff: Testfall

- Bei zustandslosen Programmen (immer gleiche Ausgabe bei gleicher Eingabe) ist Abgrenzung von Testfällen eindeutig
  - Jeder Programmaufruf ist ein Testfall
- Bei zustandsbehafteten Programmen hängen die Elemente von Folgen von Eingaben zusammen
  - Sorgfältig klären, was die Aufgabe eines Testfalles ist
  - Hilfsbegriff: Testkontext (Zustand des Programms)
  - Testfall für beliebigen Kontext
  - Testfall für spezifischen Kontext
  - Testfallgruppe für bestimmte Kontextabfolge
- Entwurfsregel: Vermeide unnötige Zustandsbehaftung

Anweisungs-/Zweigabdeckung Bedingungsüberdeckung

Anweisungsabdeckung, Zweigabdeckung:

- In der Praxis oft gerade bei korrektem Programm nicht erreichbar:
  - Z.B. unmöglicher else-Zweig einer else-if-Kette mit `assert false;` abgesichert
- Deshalb sind die Kriterien eher als Richtlinien zu verstehen

Mehrfache Bedingungsüberdeckung:

- Selten voll erreichbar, weil sich mit ODER verknüpfte Bedingungen oft gegenseitig ausschliessen

Kontrollflussorientierte Verfahren

Sind ziemlich problematisch:

- Abdeckung der Kriterien in der Praxis sehr schwierig zu ermitteln
  - Ohne Werkzeugunterstützung hoffnungslos
  - Noch schlimmer bei statisch nicht mehr zu bestimmendem Kontrollfluss in objektorientierten Programmen
- Kriterien selbst sind in der Praxis dubios bzw. Schwierig auszulegen
  - Z.B. für Felder und Behälterdatentypen

Strukturtest vs. Funktionstest

- Strukturtest ist nützlich für algorithmisch komplexe Programmteile
- Andernfalls ist aber Funktionstest oft erfolgversprechender
  - Da er nicht nur falsche, sondern auch fehlende Implementierungen aufdeckt
  - Ausserdem sind die Testfälle mehr am Problem orientiert und finden tendenziell relevantere Defekte

Strukturtest und Spezifikationsdokument

- Balzert über die Spezifikation: „Im Idealfall liegt sie in formaler Form vor."
- Stimmt nur, wenn die Spezifikation sorgfältig validiert wurde
- Andernfalls (also meistens) sollte man nicht nur nach Spezifikation, sondern zusätzlich nach gesundem Menschenverstand testen
  - Insbesondere Lücken findet man oft beim Nachdenken über Äquivalenzklassen etc.

Besonderheit bei objektorientierten Programmen

- Kontrollfluss bei OO nicht mehr direkt aus Quelltext erkennbar:
  - Ein Aufruf myobj.mymeth() kann auf viele verschiedene Methodenrümpfe mymeth() verweisen
  - Je nachdem, zu welcher Unterklasse myobj gerade gehört.
- Konsequenz: Nach Einführung einer neuen Unterklasse müssen alle Programmteile neu getestet werden, die Variablen der Oberklasse und evtl. Instanzen dieser Unterklasse verwenden
- Kann im Einzelfall furchtbar kompliziert werden
- Nur durch disziplinierten Entwurfsstil zu vermeiden:
  - Entwurf per Vertrag (Design by contract)
  - Gründlicher Modultest gegen diesen Vertrag

Danke!