



Qualitätssicherung von Software

Prof. Dr. Holger Schlingloff

Humboldt-Universität zu Berlin
und
Fraunhofer FIRST

Kapitel 2. Testverfahren

2.1 Testen im SW-Lebenszyklus

2.2 funktionsorientierter Test

- Modul- oder Komponententest
- Integrations- und Systemtests

2.3 strukturelle Tests, Überdeckungsmaße

2.4 Test spezieller Systemklassen

- Test objektorientierter Software
- Test graphischer Oberflächen
- Test eingebetteter Realzeitsysteme

2.5 automatische Testfallgenerierung

2.6 Testmanagement und –administration

Übung

Entwerfen
Sie einen
Testplan
für die
Java-
Standard-
klasse
Stack!

Java 2 Platform SE v1.3.1: Class Stack - Microsoft Internet Explorer

Adresse <http://java.sun.com/j2se/1.3/docs/api/java/util/Stack.html>

Google Web-Suche 120 blockiert Optionen

Fields inherited from class java.util.Vector

[capacityIncrement](#), [elementCount](#), [elementData](#)

Fields inherited from class java.util.AbstractList

[modCount](#)

Constructor Summary

[Stack\(\)](#)
Creates an empty Stack.

Method Summary

Boolean	empty() Tests if this stack is empty.
Object	peek() Looks at the object at the top of this stack without removing it from the stack.
Object	pop() Removes the object at the top of this stack and returns that object as the value of this function.
Object	push(Object item) Pushes an item onto the top of this stack.
int	search(Object o) Returns the 1-based position where an object is on this stack.

Methods inherited from class java.util.Vector

[add](#), [add](#), [addAll](#), [addAll](#), [addElement](#), [capacity](#), [clear](#), [clone](#), [contains](#), [containsAll](#), [copyInto](#), [elementAt](#), [elements](#), [ensureCapacity](#), [equals](#), [firstElement](#), [get](#), [hashCode](#), [indexOf](#), [indexOf](#), [insertElementAt](#), [isEmpty](#), [lastElement](#), [lastIndexOf](#), [lastIndexOf](#), [remove](#), [remove](#), [removeAll](#), [removeAllElements](#), [removeElement](#), [removeElementAt](#), [removeRange](#), [retainAll](#), [set](#), [setElementAt](#), [setSize](#), [size](#), [subList](#), [toArray](#), [toArray](#), [toString](#), [trimToSize](#)

Methods inherited from class java.util.AbstractList

[iterator](#), [listIterator](#), [listIterator](#)

Methods inherited from class java.lang.Object

[finalize](#), [getClass](#), [notify](#), [notifyAll](#), [wait](#), [wait](#), [wait](#)

Methods inherited from interface java.util.List

[iterator](#), [listIterator](#), [listIterator](#)

Internet

Testorakel und Zusicherungen

- Problematik der Beurteilung eines Testergebnisses
 - Kapselung verhindert direkten Zugriff auf Objektattribute
 - Aufhebung der Kapselung (friendly test class) verlagert das Korrektheitsproblem auf die Testumgebung
 - Kompromiss: nur lesende Zugriffe im Tester
- Zusicherungen
 - Boole'sche Ausdrücke, die Aussagen über den Objektzustand formulieren
 - Geben Vor- und Nachbedingung für jede Methode wieder
 - Zur Übersetzungszeit zu- oder abschaltbar
 - Eingriff in zeitliches Ablaufverhalten!

OO-Erweiterungen

- Parametrisierte Klassen

- Verarbeitung sollte möglichst unabhängig vom Typ der verarbeiteten Objekte sein
- Review um Abhängigkeiten aufzuzeigen
- Wahl *einer* möglichst einfachen Parameterinstanz (z.B. Stack(Int) für Stack(x))

- Abstrakte Klassen

- Test *aller* jeweiligen Instanzen
- abstrakte Klasse für den Test nur „syntaktischer Zucker“

- Vererbung

- Mehrfachvererbung ist zu vermeiden!
- saubere Klassen- Unterklassenhierarchie für Testbarkeit

Vererbung und Unterklassen

- Für den Test einer Unterklasse
 - für alle neuen Methoden
 - neue Testfälle aufstellen und ausführen
 - für alle redefinierten Methoden
 - neue Testfälle aufstellen und ausführen;
 - selbst semantisch ähnliche Programmteile müssen neuen, unterschiedlichen Tests unterzogen werden (Neuimplementierung)
 - für alle geerbten Methoden
 - alle Testfälle der Oberklasse erneut durchführen (Kontext der Unterklasse ist anders, erneuter Test notwendig)
- normalerweise wesentlich mehr ererbte als eigene Methoden! (→ Testautomatisierung erforderlich)

objektorientierte Analyse

- Beschreibung des Systems durch Klassen- und Interaktionsdiagramme
- Ableitung von Testfällen aus den Entwurfsdokumenten (nicht aus dem Code!)
- Abstrakte Datentypen (später)

OO-Integration

Integrationsstrategien für OO-Systeme

- Klassen-Integration
 - Klassen, die zu einer Komponente gehören
- Komponenten-Integration
 - Komponente: Menge von gegenseitig abhängigen Klassen, die zu einer binären, ausführbaren Code-Einheit zusammengefaßt sind. DLL's, Lib., ...
 - alle Komponenten einer Anwendungsschicht
- Schichten-Integration
 - Präsentations- / Verarbeitungs- / Zugriffs-Schicht oder Client- / Server-Schicht

OO-Integrationstest

- Im Gegensatz zu modularer Software, wo größtenteils Baumstrukturen der Benutzungsbeziehungen auftreten, sind in objekt-orientierten Programmen oft Graphen mit starken Zusammenhangskomponenten (Cluster) und zyklischen Abhängigkeiten zu beobachten.
- Objektorientierte Systeme sind eher ereignisgesteuerte Systeme. Integrationsansatz muss dies berücksichtigen.
- 5 Stufen
 - Methodentest (method test)
 - Nachrichtensequenzen (message sequences)
 - Ereignissequenzen (event sequences)
 - Ablauftest (thread testing)
 - Anwendungstest (thread interaction testing)

OO-Systemtest

- Der Systemtest betrachtet das System als schwarzen Kasten. Es ist prinzipiell unerheblich, ob das System intern objektorientiert oder konventionell aufgebaut ist
- Auswahl der Testfälle auf Basis der Entwurfsdokumente (z.B. *use cases*).
- frühzeitige Überprüfung der Modelle unter Einbeziehung des Auftraggebers

Pause!



Lotos

- algebraische Spezifikationssprache (*Language of Temporal Ordering Specification*)
- standardisiert (ISO 8807, 1989)
- viel Theorie, einige praktische Beispiele
- unterstützt objektorientierten Entwurf
- Erweiterungen / Varianten (z.B. CSP-CASL)
- Testtheorie (Forschung!)

Literatur: Kenneth J. Turner, The Formal Specification Language LOTOS: A Course for Users. <http://www2.cs.uregina.ca/~sadaouis/CS872/lotos-users.pdf>

Marie-Claude Gaudel und Perry R. James.
Testing Algebraic Data Types and Processes: A Unifying Theory.
Formal Aspects of Computing, 10(5-6), (1999) Seite 436-451

(lesen!!!)

Bestandteile:

- Abstrakter Datentyp
 - Datentypbezeichnung(en)
 - Funktionen / Operationen mit Typ
 - definierende Gleichungen
- Prozessalgebraische Verhaltensbeschreibung
 - rekursive Prozessdefinition
 - Parallelität und Synchronisation / Kommunikation

Beispiel: ADT Stack

```
type Stack is Boolean
  formalsorts Element
sorts Stack
opns
  empty : Stack - > Bool
  emptyStack : - > Stack
  push : Element, Stack - > Stack
  peek : Stack - > Element
  pop : Stack - > Stack
eqns
forall e: Element, s: Stack
  ofsort Bool
    empty(emptyStack) = true;
    empty(push(e,s)) = false;
  ofsort Element
    peek(push(e,s)) = e;
  ofsort Stack
    pop(push(e,s)) = s;
endtype (* Stack *)
```

Frage:
Erweiterung um
search?

Instantiierung

- Stack entspricht abstrakter Klasse
- Konkrete Klasse:

```
type NatStack is
    GenericStack actualizedby NaturalNumber
    using sortnames
        Nat for Element
        NatStack for Stack
endtype (* NatStack *)
```

Semantik von Termen

- Termalgebra: alle wohltypisierten Ausdrücke
- freie Algebra, („Herbrand-Universum“): keine Gleichungen, jeder Term ist ein eigener Wert
- Gleichungen implizieren Äquivalenzpartitionierung
- mehrere Möglichkeiten der Semantik
 - initiale Semantik: kleinste Äquivalenzpartitionierung der freien Algebra
(alles was nicht beweisbar gleich ist, ist ungleich)
 - lose Semantik: irgendeine Partitionierung

weitere Möglichkeiten

- bedingte Gleichungen
- Parametrisierte Typen (abstrakte Klassen)
- Überladen von Funktionen (Polymorphie)
 - z.B. Gleichheit
 - **ofsort** zur Kennzeichnung des Typs
- Renaming und Subtypisierung
 - **type B is A renamedby sortnames ... for ...**