Softwarequalitätssicherung

Sommersemester 2003

Dr. Thomas Santen
Softwaretechnologie
TU Dresden

Muster für den Methodentest (1)

***Category-Partition*:** Testfälle für eine einzelne Methode durch funktionale Äquivalenzklassenanalyse.

Fehlermodell: Fehlverhalten wird durch best. Kombinationen von Eingabewerten und Attributwerten ausgelöst. Fehler, die durch Aufrufsequenzen entstehen oder in nicht-öffentlichen Attributen niederschlagen, werden nicht gefunden.

Combinational Function Test: Testfälle, die gezielt best. Entscheidungen in der Programmlogik auslösen; analog Bedingungsüberdeckung, aber auf Spezifikation (Nachbedingung), nicht auf Abfragen im Code.

Fehlermodell:

- falsche Wertzuweisung an Entscheidungsvariable
- falscher / fehlender Operator in Prädikat
- falscher / fehlender *default*-Fall
- fehlendes `break`
- etc.

Muster für den Methodentest (2)

Recursive Function Test: Testfälle für eine rekursiv programmierte Methode, die *Basisfall*, *Abstieg* und *Aufstieg* gezielt ansprechen.

Fehlermodell:

- Nicht-Terminieren bei verletzter Vorbedingung
- Basisfall nicht behandelt
- Fehler in algorithmischer Aufteilung in Basis-/Rekursionsfall
- falsche Bedingung für Basisfall
- fehlerhaftes Traversieren einer rekursiven Datenstruktur
- Nicht-Terminieren bei verletzter Nachbedingung im Aufstieg
- nicht-lineare Laufzeit verletzt Realzeit-Bedingung

Muster: Polymorphic Message Test

Absicht: Testentwurf für den Klienten eines polymorphen Servers, der alle Bindungsmöglichkeiten berücksichtigt.

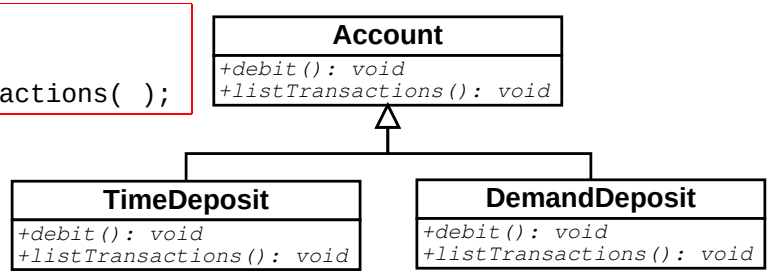
Kontext: Wenn eine Servermethode durch Vererbung überschrieben wird, dann müssen alle möglichen dynamischen Bindungen eines Methodenaufrufs in einer Klientenmethode getestet werden.

Fehlermodell:

1. Klient erfüllt nicht alle Vorbedingungen der Re-Definitionen (Server-Hierarchie ist nicht verhaltenskonform)
2. fehlerhafte / unbeabsichtigte dynamische Bindung
3. Server-Klasse wurde geändert (macht erneuten Test des Klienten notwendig)

POLYMORPHIC MESSAGE TEST – BEISPIEL (1)

```
Account* acct;
// ...
acct->listTransactions( );
```



- Aufruf von `listTransactions` wird zu Fallunterscheidung übersetzt
- statisch ist nicht zu entscheiden, welcher Fall erreicht wird
- Vorbedingungen von `debit` in `TimeDeposit` und `DemandDeposit` könnten verschärft sein (keine Konformität!)

POLYMORPHIC MESSAGE TEST – BEISPIEL (2)

```
report( ) {
   Account* accountList[MAXACCTS];

   int ix = tdIndex( );   // Berechne Index
   accountList[ix] = new TimeDeposit;
   // ...
   ix = ddIndex( );       // Berechne neuen Index
   accountList[ix] = new DemandDeposit;
   // ...
   ix = tdIndex( )        // Wähle DemandDeposit Konto
                          // Fehler: falsche Funktion
   accountList[ix]->debit( );
   // ...                                              }
```


POLYMORPHIC MESSAGE TEST – STRATEGIE

Testmodell: Erweiterter Flussgraph, der alle möglichen dynamischen Bindungen an den Server berücksichtigt.

Test-Prozedur:

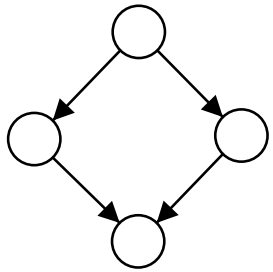
1. mögliche Bindungen für Server-Aufrufe bestimmen
2. Flussgraph an den Server-Aufrufen entsprechend expandieren
3. Zwei Knoten für jede Bindung: Verzweigung, Methoden-Aufruf
4. Schlussknoten, der Bindungsfehler zur Laufzeit repräsentiert
5. Testfallerzeugung auf Basis dieses Flussgraphen

Eingangskriterien: Small-Pop; Server muss stabil sein (evtl. mittels Stubs)

Ausgangskriterien: Zweigüberdeckung auf dem erweiterten Flussgraphen, d.h. jede mögliche Bindung wird mindestens einmal verwendet.

BEISPIEL-FLUSSGRAPH

```
void reportHistory(Account *acct) {
  if !acct->isOpen( ) {
    acct->listTransactions( );
  }
  else {
    acct->debit(amount);
  }
}
```



Erweiterter Flussgraph