Test-Driven Design: Ein einfaches Beispiel

Martin Wirsing

in Zusammenarbeit mit Moritz Hammer und Axel Rauschmayer

Ziele

 Veranschaulichung der Technik des Test-Driven Design am Beispiel eines Programms zur Berechnung der Quersumme

Lernen JUnit für Test-Driven Design einzusetzen

Test-gesteuerter Entwurf

Neue Software-Enwurfstechniken stellen das Testen vor das Implementieren des Programms:

Externe Programming, Test-first Programming, Agile Software Development

- Schritte des Test-gesteuerten Entwurfs:
 - Erstelle UML-Diagramm
 - Entwerfe einen Test für eine Methode

Kurze Iterationen, in denen abwechselnd Code und Test geschrieben wird.

- 3. Schreibe möglichst einfachen Code, bis der Test nicht mehr fehlschlägt
- 4. Wiederhole 2. und 3. bis alle Methoden des Klassendiagramms implementiert sind.
- Dabei wird häufig der Code (und manchmal der Test) restrukturiert ("Refactoring").
 Jedes Mal werden alle Tests durchgeführt, um sicher zu stellen, dass die Coderestrukturierung nicht zu Fehlern im "alten" Code geführt hat.

UML Diagramm für Quersumme

Quersumme

int berechneQuersumme (int zahl)

Dieses UML-Diagramm spiegelt eine Entwurfsentscheidung wider; eine andere Möglichkeit wäre ein Attribut "zahl" und int berechneQuersumme() Berechnet Quersumme von zahl,

Das Gerüst der Testklasse Quersumme

```
import junit.framework.TestCase;
public class QuersummeTest extends TestCase
   public QuersummeTest()
                                                      Kurze Iterationen, in
       super();
                                                      denen abwechselnd
                                                        Code und Test
                                                       geschrieben wird.
< Testmethoden>
   public static void main(String args[])
       junit.textui.TestRunner.run(QuersummeTest.class);
```

Ein erster Test für Zahlen kleiner als 10

```
public void testQuersummeKleiner10()
{
    Quersumme o1 = new Quersumme();
    Quersumme o2 = new Quersumme();
    assertEquals(0, o1.berechneQuersumme(0));
    assertEquals(9, o2.berechneQuersumme(9));
}
```

Tests für die "Grenzwerte" 0 und 9.

Ein möglichst einfaches Programm für den 1. Test

```
public class Quersumme
    public Quersumme(){ }
    public int berechneQuersumme(int zahl)
         return zahl;
                                                                                        _ | _ | ×
                                     JU JUnit
                                    JUnit
                                     Test class name:
                                     sum.QuersummeTest
                                                                                        Run
                                     Reload classes every run
                                                                                       Ju
        Gibt zahl als Wert
                                                       X Errors: 0
                                                                      X Failures: 0
                                     Runs: 1/1
       zurück; dies ist ok für
                                     Results:
        0 \le zahl < 10!
                                                                                        Run
Erwartungsgemäß
erfüllt das einfache
Programm diesen Test!
                                       X Failures
                                                 📌 Test Hierarchy
```

Ein zweiter Test für Zahlen größer gleich 10

```
public void testQuersummeGroesserGleich10()
Quersumme o1 = new Quersumme();
Ouersumme o2 = new Ouersumme();
assertEquals(1, o1.berechneQuersumme(10));
assertEquals(11, o2.berechneQuersumme(29)
                                                     Grenzwert für die
                                                    Fallunterscheidung
                                                    "GroesserGleich10"
                                      Ein beliebiger Wert
                                        größer als 10
```

Der zweite Test für Zahlen größer gleich 10 schlägt fehl

JU JUnit _ | 🗆 | × **JUnit** Test class name: sum.QuersummeTest Run Reload classes every run Natürlich meldet Ju JUnit Fehler. X Errors: 0 X Failures: 1 Runs: 2/2 Das Programm Results: muss geeignet testQuersummeGroesserGleich10(sum.QuersummeTest):expect Run restrukturiert werden. testQuersummeGroesserGleich10(sum.QuersummeTest);expected;<1> but was;<10> iunit.framework.AssertionFailedError; expected:<1> but was:<10> at sum.QuersummeTest.testQuersummeGroesserGleich10(Quersu at sun.reflect.NativeMethodAccessorImpl.invokeO(Native Method) at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccess at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMeth M. Wirsing: Spezifikation und Test

Refactoring: Ein möglichst einfaches Programm für beide Tests

Berechnet die
Quersumme rekursiv.
Was ist mit
negativen
Zahlen??

Ein Test für negative Zahlen

```
public void testQuersummeKleiner0()
Quersumme o1 = new Quersumme();
Ouersumme o2 = new Ouersumme();
assertEquals(-9, o1.berechneQuersumme(-9));
assertEquals(-11, o2.berechneQuersumme(-
                                                     Grenzwert für die
                                                     Fallunterscheidung
                                                    "GroesserGleich10"
                                      Ein beliebiger Wert
                                        größer als 10
```

Der Test für negative Zahlen schlägt fehl!

JU JUnit **JUnit** Test class name: sum.QuersummeTest Run Reload classes every run Natürlich meldet JUnit Fehler. X Errors: 0 X Failures: 1 Runs: 3/3 Das Programm Results: muss geeignet X testQuersummeKleiner0(sum.QuersummeTest):expected:<-11> restrukturiert Run werden. X Failures 📌 Test Hierarchy junit.framework.AssertionFailedError: expected:<-11> but was:<-29> at sum.QuersummeTest.testQuersummeKleiner0(QuersummeTest at sun.reflect.NativeMethodAccessorImpl.invokeO(Native Method) at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccess M. Wirsing: Spezifikation und Test

Refactoring: Ein erstes Programm für alle Tests

```
public class Quersumme
public Quersumme(){ }
public int berechneQuersumme(int zahl)
    if (zahl<10 & zahl>0) return zahl;
    else return berechneQuersume(zahl/10)+zahl%10;
                                                 JU JUnit
  JUnit
                                                           Programmierfehler bei
   Test class name:
                                                                 zahl = 0.
   sum.QuersummeTest
                                                  Run
   Reload classes every run
                                                 Ju
                   X Errors: 1
                                 X Failures: 0
  Runs: 3/3
   Results:
   testQuersummeKleiner0(sum.QuersummeTest)
                                                  Run
M
```

Verbesserung: Ein Programm für alle Tests

```
public class Quersumme
    public Quersumme(){ }
    public int berechneQuersumme(int zahl)
         if (zahl<10 & zahl>=0) return zahl;
         else return berechneQuersumme(zahl/10)+zahl%10;
                                  JU JUnit
                                                                                   _ | _ | ×
                                  JUnit
                                   Test class name:
                                   sum.QuersummeTest
                                                                                    Run
                                   Reload classes every run
Jetzt laufen alle Tests!
                                                                  X Failures: 0
                                                    X Errors: 0
                                   Runs: 3/3
                                   Results:
                                                                                    Run
  M. Wirsing: Spezifikation und Test
```

Refactoring 1: Algebraische Umformung

```
public int berechneQuersumme(int zahl)
       if (zahl<10 & zahl>=0) return zahl;
       else return berechneOuersumme(zahl/10)+zahl%10;
                                 Da für 0<=zahl<10 gilt: zahl =
                   if (zahl==0)zahl else berQs(zahl/10)+zahl%10
ist äquivalent zu
   public int berechneQuersumme(int zahl)
                                                   Auch hier laufen alle
                                                         Tests!
       if (zahl==0) return zahl;
       else return berechneQuersumme(zahl/10)+zahl%10;
```

Refactoring 2: Ein Schema zur Transformation linearer Rekursion in while-Programme

```
R f(T x)
    if (B(x)) return A(x);
    else return f(E(x)) o C(x);
                                              Falls • kommutativ und
                                                    assoziativ ist
Ist äquivalent zu
R f(T x)
   R r = A(x);
                                                  Beispiel Quersumme:
                                                       entspricht +
   while (!(B(x)))
                                                  A(x)
                                                       entspricht 0
       r = r \circ C(x); x = E(x);
                                                  B(x)
                                                       entspricht x==0
                                                  C(x)
                                                       entspricht x%10
                                                       entspricht x/10
                                                  E(x)
   return r;
```

M. Wirsing: Spezifikation und Test

Refactoring 2: Ein iteratives Programm für alle Tests

```
public class Quersumme
    public Quersumme(){ }
    public int berechneQuersumme(int zahl)
         int qs = 0;
                                              JU JUnit
        while (zahl!=0)
                                              JUnit
                                               Test class name:
            qs = qs + zahl % 10;
                                               sum.QuersummeTest
             zahl = zahl /10;
                                               Reload classes every run
        return qs
                          Berechnet die
                        Quersumme iterativ.
                                                               X Errors: 0
                                                                              X Failures: 0
                                              Runs: 3/3
                         Erfüllt alle
                                              Results:
                             Tests!
   M. Wirsing: Spezifikation und Test
```

Zusammenfassung

Beim Test-gesteuerten Entwurf werden zuerst die Tests entworfen und dann die Programme geschrieben.

- Programme werden immer wieder restrukturiert (Refactoring). Durch automatisches Testen ist dies gut machbar.
- Formale Umformungstechniken (algebraische Umformungen, Transformationstechniken) können sehr gut zum Refactoring eingesetzt werden.
- Eine wichtige Umformung ist die Transformation linear rekursiver Programme in While-Programme.