Software Engineering in der Praxis

Praktische Übungen

Funktionales Testen

David Föhrweiser Marc Spisländer

Lehrstuhl für Software Engineering
Friedrich-Alexander-Universität Erlangen-Nürnberg

Nachlese

Software-Metriken und Bugpatterns

- Allgemeine Betrachtungen zum Messen
- Zweck der Software-Messung
- Together-Metriken:
  - LOC
  - Halstead-Metriken
  - McCabe-Metrik
- Fehlersuche mit FindBugs

Testen von Code

Definition

Ausführung eines Programms in einer definierten Umgebung, um:

- Fehler zu finden (Programmierfehler, algorithmische Fehler, Nichterfüllung der Spezifikation, . . .)
- Zuverlässigkeitskennwerte zu ermitteln

Testen von Code

Definition

Ausführung eines Programms in einer definierten Umgebung, um:

- Fehler zu finden (Programmierfehler, algorithmische Fehler, Nichterfüllung der Spezifikation, . . .)
- Zuverlässigkeitskennwerte zu ermitteln

Tests garantieren keine Fehlerfreiheit!

Ablauf von Tests

1. Testplanung (Testkriterien, Personal, Budget, . . .)
2. Testerstellung: Auswahl der Eingabedaten (Testdaten) und eines Prüfungsverfahrens für die Ausgabedaten
3. Testdurchführung: Ausführung des Programms mit ausgewählten Eingabedaten
4. Testauswertung: Vergleich der Ausgabedaten mit den erwarteten Ergebnissen

Testarten

Tests können in zwei Kategorien eingeteilt werden:

Black-Box-Tests

- Testdaten werden aus der Spezifikation abgeleitet.
- Ziel: Übereinstimmung mit Spezifikation prüfen

Testarten

Tests können in zwei Kategorien eingeteilt werden:

Black-Box-Tests

- Testdaten werden aus der Spezifikation abgeleitet.
- Ziel: Übereinstimmung mit Spezifikation prüfen

White-Box-Tests

- Testdaten werden aus dem Quellcode abgeleitet.
- Ziel: Auswirkungen aller Code-Abschnitte identifizieren

Black-Box-Tests

- Werden auch *funktionale Tests* genannt
- Tests können bereits vor der Implementation definiert werden.
- Möglichkeiten, Testdaten abzuleiten:
  - Äquivalenzklassentest
  - Grenzwerttests
  - Error Guessing
  - Zufallsdaten

Black-Box-Tests

Äquivalenzklassentest Teile die Eingabemenge in Klassen, so dass alle Eingaben einer Klasse »ähnliches Ausgabeverhalten hervorrufen« sollen. Wähle dann aus jeder Klasse Repräsentanten als Testdaten.

Black-Box-Tests

Äquivalenzklassentest Teile die Eingabemenge in Klassen, so dass alle Eingaben einer Klasse »ähnliches Ausgabeverhalten hervorrufen« sollen. Wähle dann aus jeder Klasse Repräsentanten als Testdaten.

Grenzwerttests Wähle Testdaten am Rande der Äquivalenzklassen (Voraussetzung: Es ex. Ordnung auf der Menge der Eingabedaten).

Black-Box-Tests

Äquivalenzklassentest Teile die Eingabemenge in Klassen, so dass alle Eingaben einer Klasse »ähnliches Ausgabeverhalten hervorrufen« sollen. Wähle dann aus jeder Klasse Repräsentanten als Testdaten.

Grenzwerttests Wähle Testdaten am Rande der Äquivalenzklassen (Voraussetzung: Es ex. Ordnung auf der Menge der Eingabedaten).

Error Guessing Spezifikationsbezogener Fehlererwartungstest (aus Erfahrung)

Black-Box-Tests

Äquivalenzklassentest Teile die Eingabemenge in Klassen, so dass alle Eingaben einer Klasse »ähnliches Ausgabeverhalten hervorrufen« sollen. Wähle dann aus jeder Klasse Repräsentanten als Testdaten.

Grenzwerttests Wähle Testdaten am Rande der Äquivalenzklassen (Voraussetzung: Es ex. Ordnung auf der Menge der Eingabedaten).

Error Guessing Spezifikationsbezogener Fehlererwartungstest (aus Erfahrung)

Zufallsdaten Wähle Testdaten nach statistischen Verteilungen.

White-Box-Tests

- Werden auch *strukturelle Tests* genannt

White-Box-Tests

- Werden auch *strukturelle Tests* genannt
- Kriterien, für die Erzeugung der Testdaten:
  - Kontrollflussabdeckung (Pfadüberdeckung, Anweisungsüberdeckung, Verzweigungsüberdeckung)

White-Box-Tests

- Werden auch *strukturelle Tests* genannt
- Kriterien, für die Erzeugung der Testdaten:
  - Kontrollflussabdeckung (Pfadüberdeckung, Anweisungsüberdeckung, Verzweigungsüberdeckung)
  - Datenflussabdeckung

White-Box-Tests

- Werden auch *strukturelle Tests* genannt
- Kriterien, für die Erzeugung der Testdaten:
  - Kontrollflussabdeckung (Pfadüberdeckung, Anweisungsüberdeckung, Verzweigungsüberdeckung)
  - Datenflussabdeckung
- Tests können erst nach der Implementation definiert werden.

JUnit als Framework für funktionale Tests

- Erhältlich unter www.junit.org
- Ziele: Normierung der Testimplementierung, damit Vereinfachung der Testdurchführung
- Auf Java beschränkt

JUnit

Grobe Vorgehensweise

1. Annotiere Methoden, die Tests durchführen, mit *@Test*.

JUnit

Grobe Vorgehensweise

1. Annotiere Methoden, die Tests durchführen, mit *@Test*.
2. Formuliere innerhalb dieser Testmethoden die Bedingung für erfolgreiches Testergebnis.

JUnit

Grobe Vorgehensweise

1. Annotiere Methoden, die Tests durchführen, mit *@Test*.
2. Formuliere innerhalb dieser Testmethoden die Bedingung für erfolgreiches Testergebnis.
3. Die JUnit-Engine findet per Reflections die annotierten Testmethoden, führt sie aus und zeigt an, ob sie erfolgreich waren oder nicht.

JUnit Beispiel

Klasse *Calculator*

- **public double** sum(**double** a, **double** b)
- **public double** diff(**double** a, **double** b)
- **public double** mult(**double** a, **double** b)
- **public double** div(**double** a, **double** b)
- **public void** setMem(**double** a)
- **public double** getMem()
- **public void** clearMem()

JUnit Beispiel

Testklasse *CalculatorTest*

```
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class CalculatorTest {
  private Calculator calculator;

  @Before public void setUp() {
    calculator = new Calculator();
  }

  @Test public void testSum() {
    double a = 5.6, b = 6.5;
    assertEquals(12.1, calculator.sum(a, b), 0.1);
  }
```

JUnit Beispiel

Testklasse *CalculatorTest* (Fortsetzung)

```
    @Test(expected = IllegalArgumentException.class)
2   public void testDiv() {
      double a = 3.0;
4     double b = 0.0;

6     calculator.div(a, b);
    }
```

Tests starten

- Aus der Kommandozeile mit:
  java org.junit.runner.JUnitCore CalculatorTest
- Aus der IDE: In Eclipse, das Projekt starten mit
  »Run As.../JUnit Test«

JUnit

Online-Ressourcen

- Primärquelle: http://junit.sourceforge.net/
- Javadocs: junit.sourceforge.net/javadoc_40/index.html