andrena OBJECTS

Experts in agile software engineering

TESTGENERIERUNG IN JAVA

EIN ÜBERBLICK

Stefan Mandel, Urs Metz

LEGACY CODE

- große, gewachsene Code-Basis
- wichtige Geschäftslogik
- keine Tests

… UND WIE ES DAS SCHICKSAL WILL

- notwendige Änderungen
- in mehreren Schichten
- mit harter Deadline

... UND WAS JETZT?

- Test schreiben
  - "wir haben doch keine Zeit!"
- Änderungen ohne Tests durchführen?
  - "was da alles kaputt gehen kann?!"
- Tests automatisch generieren?
  - "das geht doch gar nicht!"

IST DAS VIELLEICHT DOCH MÖGLICH?

KRITERIEN

- Zielgerichtet: Der Testfall ist aus dem Test ersichtlich
- Vollständig: Jeder Testfall hat minimal einen Test
- Redundanzfrei: Jeder Testfall hat maximal einen Test
- Relevant: Jeder Testfall kann in der Realität auftreten
- Kurz: Der Test ist kurz
- Explizit: Testeingaben, Testergebnisse und Testaktionen sind erkennbar
- Anpassbar: Der Test lässt sich anpassen, wenn Anforderungen oder Design sich ändern
- Effizient: Lohnt sich der Konfigurationsaufwand

HERAUSFORDERUNGEN BEI AUTOMATISCHER TESTGENERIERUNG

- Zustand
  - this-Object
  - Argumente von Methoden
  - Rückgabewerte von Methoden
  - statische Variablen
  - Exceptions
- Interaktionen
  - Input von außen (Standardeingabe, Dateisystem, Zufall)
  - Output nach außen (Dateisystem, Webservice-Aufrufe)

CODE-BEISPIEL

CONNECT4

RANDOOP

RANDOM SELECTION

- Generiere
  - unterschiedliche Methodenaufrufe
  - mit zufälligen Daten
- Prüfe
  - Ergebnisse der Methodenaufrufe
  - aufgetretene Exceptions

DEMO

GENERIERUNG OHNE INTERAKTION

- Start durch Skriptaufruf, danach keine Interaktion nötig

```
java -Xmx3000m -classpath "target/classes;lib/randoop-all-4.0.3.jar" ra
```

- terminiert in Abhängigkeit verschiedener konfigurierbarer Kriterien
  - z. B. des gegebenen time budget
- Batch-Lauf über Nacht möglich
- erwartbarer Aufwand:
  - Erste Ergebnisse: gering
  - Optimierung der Ergebnisse: nur begrenzt möglich

TESTQUALITÄT (1)

- Länge der Tests schwankt stark (zwischen einigen wenigen und mehreren hundert Zeilen)
- unrealistische Testsfälle

```
Grid grid0 = null;
Color color1 = Color.YELLOW;
Game game2 = new Game(grid0, color1);
try {
    game2.drop((int) (short) -1);
    Assert.fail("Expected exception of type java.lang.NullPointerExcept
} catch (NullPointerException e) {
}
// ...
```

TESTQUALITÄT (2)

- komplexe Methoden werden nur sehr oberflächlich getestet

```
Grid grid2 = new Grid((int) (byte) 0, (int) (short) 0);
// ...
Color color6 = Color.YELLOW;
// ...
boolean boolean8 = grid2.isDifferenceDiagonalComplete(1, color6);
// ...
Assert.assertTrue("" + boolean8 + "' != '" + false + "'", boolean8 ==
```

TESTQUALITÄT (3)

- Testfälle oft stark verwoben

```
Grid grid2 = new Grid((int) (byte) 0, (int) (short) 0);
Color color5 = Color.YELLOW;
boolean boolean7 = grid2.isSumDiagonalComplete((-1), color5);
Grid grid12 = new Grid((int) (byte) 0, (int) (short) 0);
//...
boolean boolean17 = grid12.isSumDiagonalComplete((-1), color15);
Color color19 = Color.RED;
boolean boolean20 = grid12.isColumnComplete(1, color19);
Grid grid24 = new Grid((int) (byte) 0, (int) (short) 0);
Color color27 = Color.YELLOW;
//...
boolean boolean29 = grid24.isSumDiagonalComplete((-1), color27);
```

ERKENNUNG VON ZUSTANDSÄNDERUNGEN

- Nur Methodenergebnisse und Exceptions
- Änderungen an Argumenten und this nur indirekt
- Asserts nur auf primitive Werte
- Keine vollwertige IO-Simulation aber IO-Replacements

ERWEITERBARKEIT

- Kein Plugin/Extension-System
- Build erfordert manuelle Schritte

BEWERTUNG

Kriterium	
Zielgerichtet	-
Vollständig	-
Redundanzfrei	-
Relevant	-
Kurz	o
Explizit	+
Anpassbar	o
Effizient	o

EVOSUITE

SEARCH BASED TEST GENERATION

- Generiere
  - zufälligen Sequenzen
- Mutiere
  - generierte Sequenzen
- Kombiniere
  - generierte Sequenzen
- Prüfe
  - Ergebnisse der Methodenaufrufe wenn vorhanden
  - aufgetretene Exceptions
- Minimiere
  - Sequenz, sodass nur relevante Aufrufe stehen bleiben
- Filtere
  - Tests, unter Berücksichtigung des Optimierungsziels

DEMO

GENERIERUNG OHNE INTERAKTION

- Start durch Skriptaufruf, danach keine Interaktion nötig

```
java -jar lib/evosuite-1.0.6.jar -generateTests -projectCP target/class
```

- terminiert in Abhängigkeit vom gegebenen time budget (z.B. über Nacht)
- viele Konfigurationsmöglichkeiten: evosuite-parameters.txt
- erwartbarer Aufwand:
  - Erste Ergebnisse: gering
  - Optimierung der Ergebnisse: sehr hoch

ERKENNUNG VON ZUSTANDSVERÄNDERUNGEN

- beschränkt auf Rückgabewerte und Exceptions
- Methodenargumente und this nur indirekt über Methoden (z.B. Getter)
- limitierte IO-Simulation, z.B. Standardeingabe:

```
    SystemInUtil.addInputLine("9rUh ");
```

- sinnvoller Input eher die Ausnahme

TESTQUALITÄT

- Tests im Allgemeinen kurz und lesbar, aber oft
- unsystematisch:

```
Color color0 = Color.YELLOW;
Color color1 = color0.getOpponent();
Color color2 = color1.getOpponent();
assertNotSame(color2, color1);
```

- irreführend:

```
boolean boolean0 = grid0.isSumDiagonalComplete(0, color0);
//...
boolean boolean1 = grid0.isColumnComplete(0, color2);
assertTrue(boolean1 == boolean0);
```

SCHLECHTE ERWEITERBARKEIT

- Es gibt keine dokumentierten Erweiterungens-Schnittstellen
- Die Erweiterung im Source-Code ist schwierig
- Architektur ist komplex (Client, Server, Process-Spawning)

BEWERTUNG

Kriterium	
Zielgerichtet	-
Vollständig	o
Redundanzfrei	+
Relevant	-
Kurz	+
Explizit	o
Anpassbar	o
Effizient	+

TESTRECORDER

CAPTURE AND REPLAY

- Zeichne auf
  - reale Interaktion mit ausgewählter Methode
- Spiele ab
  - Herstellung des Zustands vor der Interaktion
  - Interaktion
- Prüfe
  - Zustand nach der Interaktion

DEMO

BASIERT AUF USER-INTERAKTION

- Die Anwendung muss interaktiv mit (sinnvollen) Testdaten befeuert werden
  - Aufwand für Konfiguration: gering
  - Aufwand für manuelle Tests: angemessen
  - Aufwand für Optimierung der Testergebnisse: gering
- Gezielte Tests von Business-Cases möglich

JEDER TEST IST EIN REALISTISCHES SZENARIO

```
//Arrange
Grid grid1 = new Grid(7, 7);
Color color1 = Color.RED;

//Act
grid1.drop(1, color1);

//Assert
//...
```

GEGLIEDERTE UND SORGFÄLTIGE TESTS

- Jeder Test gliedert sich in die Phasen
  - Arrange
  - Act
  - Assert
- Jeder Test berücksichtigt Änderungen auf
  - this-Objekt
  - Argumenten
  - Ergebnissen
  - Exceptions
  - Globalen Variablen

MÄCHTIGE ASSERTS

```
//Act
boolean boolean1 = grid2.isDifferenceDiagonalComplete(0, color1);

//Assert
assertThat(boolean1, equalTo(false));
```

```
assertThat("expected no change, but was:", grid2, new GenericMatche
    int cols = 7;
    Matcher<?> fields = arrayContaining(Color[].class,
        arrayContaining(Color.class, null, null, null, null, null,
        arrayContaining(Color.class, sameInstance(Color.RED), sameI
        //...
        arrayContaining(Color.class, null, null, null, null, null,
```

KOMPLEXE SETUPS

```
//Arrange
Color[] colorArray1 = new Color[]{null, null, null, null, null, nul
Color[] colorArray2 = new Color[]{Color.RED, Color.RED, Color.RED,
Color[] colorArray3 = new Color[]{Color.YELLOW, Color.YELLOW, Color
//...
Color[][] colorArrayArray1 = new Color[][]{colorArray1, colorArray2
Grid grid2 = new GenericObject() {
    int cols = 7;
    Color[][] fields = colorArrayArray1;
    int rows = 7;
}.as(Grid.class);
Color color1 = Color.RED;
```

FLEXIBLE INPUT/OUTPUT-SIMULATION

- Annotationen für Input/Output an beliebige Methoden
  - `@Input` - als Input-Interaktion aufzeichnen (Stubbing)
  - `@Output` - als Output-Interaktion aufzeichnen (Verifying)
- Annotationen für Facaden/Dependencies/Mocks an beliebigen Feldern
  - `@Excluded` - nicht aufzeichnen
  - `@Facade` - nur Interaktionen aufzeichnen

ERWEITERBARKEIT DURCH PLUGIN-SYSTEM

- Konfiguration der Aufnahme
- Zur Generierung von Tests
- Zur Serialisierung/Unterstützung neuer Datentypen
- Zur Setup-Generierung
- Zur Assert-Generierung

BEWERTUNG

Kriterium	
Zielgerichtet	+
Vollständig	+
Redundanzfrei	-
Relevant	+
Kurz	-
Explizit	+
Anpassbar	+
Effizient	+

WANN SOLL ICH WAS EINSETZEN?

Selbst einen Testgenerator bauen

Randoop (simples Prinzip)
Testrecorder (Plugin-System)

Schnell und unkompliziert rudimentäre Testabdeckung

Evosuite

Realistische, wiederverwendbare Tests generieren

Testrecorder

Komplexe Testszenarien überdecken

Testrecorder

UND WAS IST AUF DER GRÜNE WIESE?

Tests zusammen mit Produktivcode schreiben:

- schnelleres (Design-)Feedback
- testbarer Code ist meistens besserer Code
- automatisch generierte Tests bestenfalls ausreichend

VIELEN DANK