dbUnit Framework- a JUnit Extension

Inhalt

- Einleitung
- Produkt dbUnit
- Datenbank Testing mit JUnit – Heikle Punkte
- Interfaces DbUnit
- Verschiedene Teststrategien
- Fazit: Lohnt sich der Einsatz von dbUnit?
- Links
- Fragen

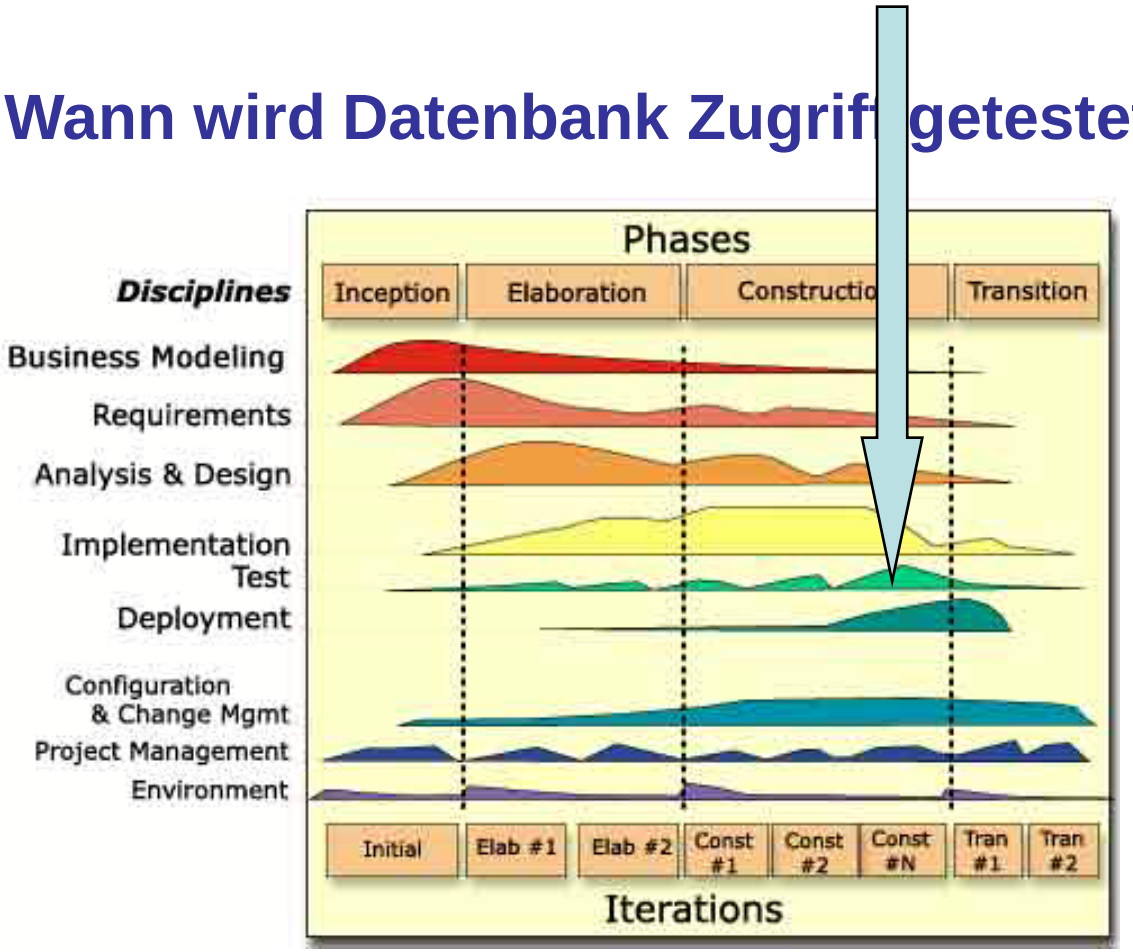
Einleitung

Warum wird Datenbank Zugriff getestet?

- Code kann immer fehlerhaft sein
- Falls Datenbank hinter Framework versteckt (z.B. Hibernate):
  - ein Framework kann falsch konfiguriert sein
  - ein Framework tut das Falsche
    (resp. Jemand lässt ihn das falsche tun)

Einleitung

Wann wird Datenbank Zugriff getestet?



Rational Unified Process

- Objektorientiertes Prozessmodell
- Metamodell für Prozessmodelle
- Gleichzeitig mit UML veröffentlicht

Grady Booch, Ivar Jacobson und James Rumbaugh

Einleitung

- Warum dbUnit
- Vorwissen
- Herangehensweise
- Ziele:
  - Möglichkeiten kennenlernen
  - kontroverse BestPractices von Anwendern kennenlernen
  - auftretende Probleme oder Grenzen erkennen

Produkt dbUnit

- Open-Source-Projekt
- sourceforge.net
- In welcher Form einbinden in Java?
  1. Java-Projekt: als jar-File (zusätzlich jars von slf4j)

     http://sourceforge.net/project/showfiles.php?group_id=47439&release_id=242511
  2. PlugIn-Projekt: PlugIn Maven

     http://mojo.codehaus.org/dbunit-maven-plugin/
  3. Auch mit Ant verwendbar

     http://dbunit.sourceforge.net/anttask.html

Datenbank Testing mit JUnit

Probleme

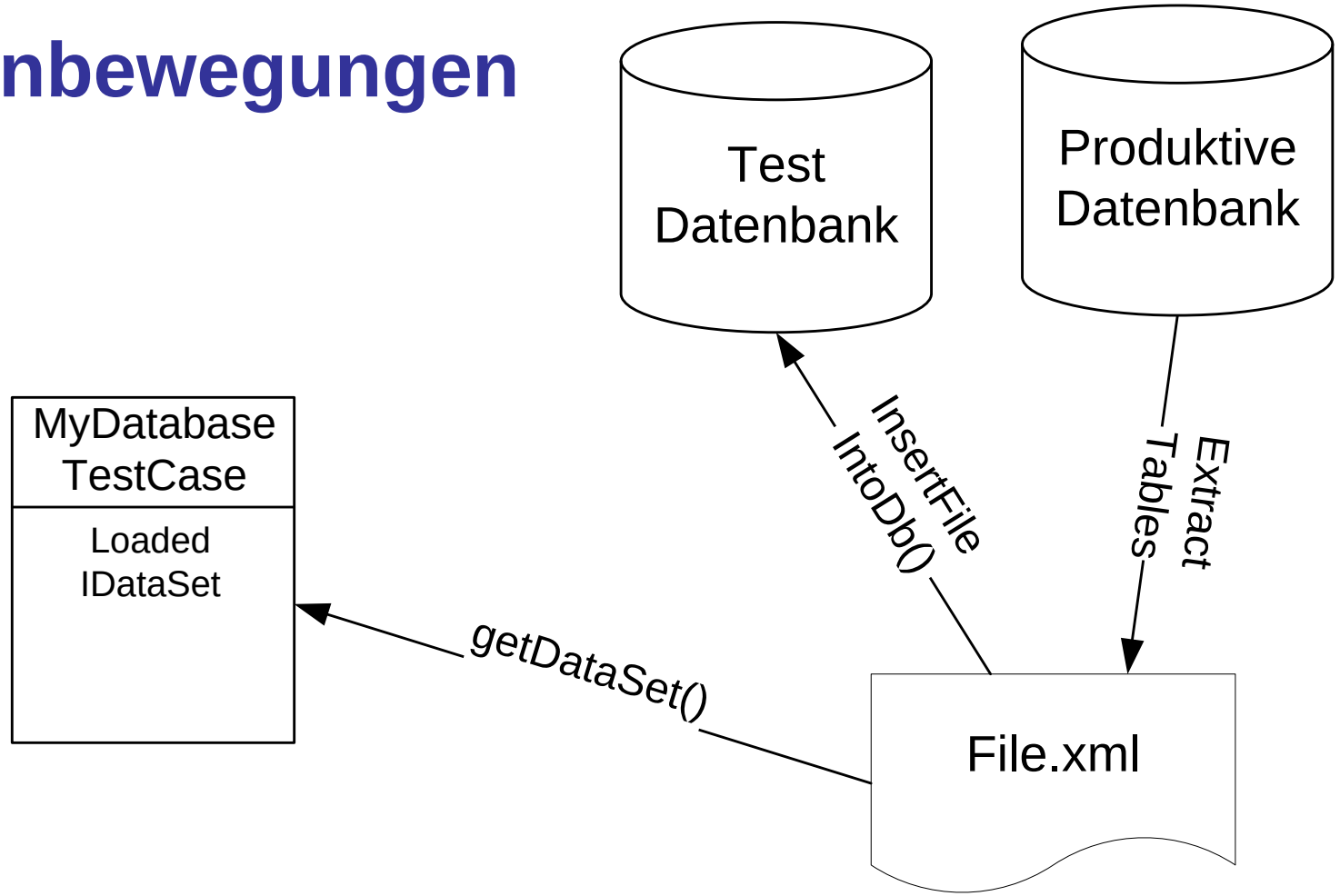
- In welchem Zustand befindet sich die Datenbank nach einem Test ?
- 1 Test (z.B. einer Suite) verlässt die Datenbank in inkonsistentem Zustand → die folgenden Tests schlagen fehl
- 1 Test braucht u.U. 5 Einträge in Tabellen
  → viel Testcode: Fehleranfälligkeit, Wartungsaufwand bei Aenderungen

Lösung dbUnit

Vorteil

- Datenbank vor jedem Test in gewünschten Zustand bringen
- Löschen von Datensätze beim SetUp → Delegiert an dbUnit
- Einfügen von Daten in Testmethode → Delegiert an dbUnit
- Durch Delegation → wenig Code, wenig Fehler, wenig Wartung

Datenbewegungen



Interfaces dbUnit

DbUnit Core Components

- IDatabaseConnection - interface repräsentiert eine DbUnit Connection zu einer Datenbank
- IDataSet - interface repräsentiert mehrere Tabellen
- DatabaseOperation – Abstrakte Klasse, repräsentiert eine Datenbank-Manipulation, die vor oder nach einem Test ausgeführt wird
- DatabaseConnection, - wrappt eine JDBC Connection
- ITable repräsentiert die Daten in einer Tabelle

Klasse DatabaseTestCase

- Konkrete Testklassen von DatabaseTestCase ableiten (Unterklasse von TestCase)
- Abstrakte Klasse
- Folgende Methoden müssen implementiert werden:

  - protected IDatabaseConnection getConnection()
  - protected IDataSet getDataSet()

Klasse DatabaseTestCase

- getConnection()

  gültige Datenbank-Connection wird geholt

```
protected IDatabaseConnection getConnection()
{
   Connection conn = null;
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   String url = "jdbc:odbc:MYSQLTESTSERVER";
   conn = DriverManager.getConnection(url, "root",
      “password");
   return new DatabaseConnection(conn);
}
```

Klasse DatabaseTestCase

- getDataSet(): von Source input.xml werden Datensätze in IDataSet der Testklasse geladen
- Aufruf durch setUp: super.SetUp(). Muss also nicht explixit aufgerufen werden.

```
protected IDataSet getDataSet()  {

    loadedDataSet = new
    FlatXmlDataSet(this.getClass().getClassLoader().
    getResourceAsStream("input.xml"));
    return loadedDataSet;
  }
```

Beispiel input.XML

```
<dataset>
    <PUBLISHER
        idPublisher = '1'
        name = 'OReilly'
        address = 'address'
        phone = '781-221-2212'/>
    <BOOK
        isbn = '0-596-00298-X'
        Publisher_idPublisher = '1'
        title = 'C++ in a Nutshell'/>
</dataset>
```

XML File Erstellen

- Von Hand
- per Code aus Datenbank: extractTables
- mit Abfrage aus Datenbank

Datenbank Operationen

- DatabaseOperation
  - UPDATE
  - INSERT
  - DELETE
  - DELETE_ALL
  - TRUNCATE
  - REFRESH
  - CLEAN_INSERT
  - NONE
- Other Operations
  - CompositeOperation
  - TransactionOperation
  - IdentityInsertOperation

Abfragen

- mit DbConnection: Daten abfragen aus Datenbank und diese mit asserts prüfen
- mit Abfrage DataSet erstellen und als FlatXMLDataSet speichern; kann später wieder in Datenbank geladen werden
- Aus Abfragen direkt Spaltenwerte abfragen ohne Iteration

Abfragen

- Beispiel 1

```
String query = "SELECT * FROM MEDIA WHERE ID= "+id;
ITable databaseData =
dbConnection.createQueryTable("EXPECTED_DATA",query);
assertEquals(1, databaseData.getRowCount());
BigDecimal foreignKey = (BigDecimal)
databaseData.getValue("FK_OTHER_ID");
assertEquals(new BigDecimal(1234)), foreignKey);
```

- Beispiel 2

```
QueryDataSet queryDataSet = new queryDataSet(getConnection());
queryDataSet.addTable(TABLE_NAME, "SELECT * FROM " +
TABLE_NAME);
Assertion.assertEquals(loadedDataSet, queryDataSet);
```

Daten vergleichen

- mit assert-Methoden Datenbank Inhalt prüfen
- dabei werden miteinander verglichen:

  a) 2 Datasets oder

  b) 2 Tabellenrepräsentationen

Beispiele

a) public static void assertEquals(IDataSet expected, IDataSet actual);

b) public static void assertEquals(ITable expected, ITable actual);

Teststrategien

dbUnitMacher

- use one database instance per developer
- Good setup don‘t need cleanUp
  - nie auf vorgehendem Test aufbauen
  - Don't be afraid to leave your trace after a test
- Use multiple small DataSets
- Perform setup of stale data once for entire test class or test suite
- Connection management strategies

Connection Management

- Bei ODBC-Connection: unbedingt UserDSN benutzen!!!
- Connection pro Testmethode
  - Setup: Connection erzeugen
  - Teardown: Connection schliessen
- Kommen in einer Testklasse viele Testmethoden vor, kann pro Testklasse eine einzige Connection verwendet werden

Teststrategien

Sun Developer

- Don’t extend DBTestCase
  - Locks you into JUnit
  - DBUnit lifecycle can make it more difficult to perform certain tests
  - Just isn’t necessary

- Simply leverage the fundamental DBUnit classes

Zu Beachten

- werden Tabellen aus einer produktiven Datenbank abgefüllt, könnten zu viele Datensätze ein Problem werden

  → mit Query arbeiten, sodass nur eine Teilmenge der Datensätze abgefüllt wird

Aussagen von Anwender

Beschreibung eines Software-Entwicklers, der mit dem ganzen Team dbUnit verwendete:

- Daten abfüllen: zu Anfang aufwändig
- Nach einer Anfangsinvestition geht es einfach, Entwickler lernen Datenbank besser kennen.

Fazit:Lohnt sich der Einsatz von dbUnit?

- Nach Anfangsinvestition scheint Aufwand kleiner als wenn Testdaten mittels Code eingefügt werden
- Handhabung recht einfach
- Keine Probleme soweit gefunden
- ??? Sehr viele Daten

Links

DbUnit Framework 2.2.3 API

http://www.dbunit.org/apidocs/index.html

Anwendungsbeispiele

http://www.ibm.com/developerworks/library/j-dbunit.html

Vortrag zum Thema dbUnit

http://developers.sun.com/learning/javaoneonline/j1sessn.jsp?sessn=TS-5859&yr=2008&track=javaee

Anhang

Verschiedene Implementation von DataSet:

- FlatXmlDataSet
- XmlDataSet
- StreamingDataSet
- DatabaseDataSet
- QueryDataSet
- DefaultDataSet
- CompositeDataSet
- FilteredDataSet
- XlsDataSet
- ReplacementDataSet

Fragen