
Max Riesemann

10.05.2005



ex|Xcellent
solutions

Testdaten Up To Date

„Automatische Aktualisierung von Testdaten“

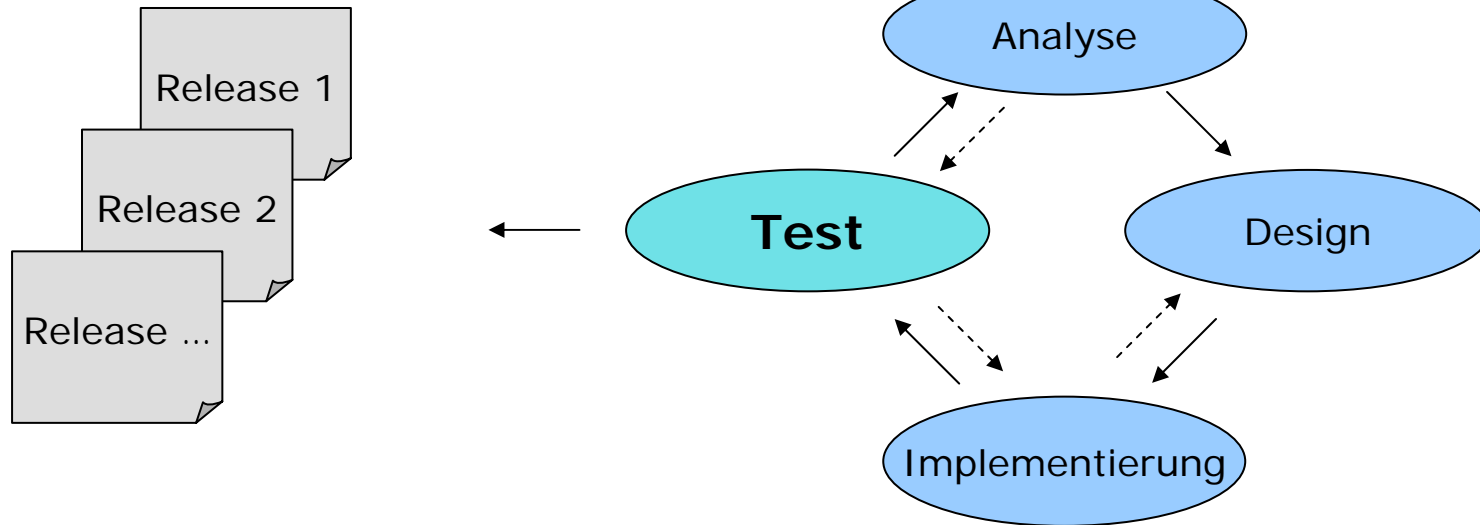


- x| Testen im Softwareentwicklungsprozess
 - | Testdaten sinnvoll erzeugen, intelligent pflegen

- x| Erzeugung von Testdaten
- x| Strategien zur Pflege der Testdaten
 - | Migration von persistenten Testdaten
 - | Generierung von Referenzdaten aus Geschäfts-Objekten

- x| Fazit
- x| Fragen & Diskussion

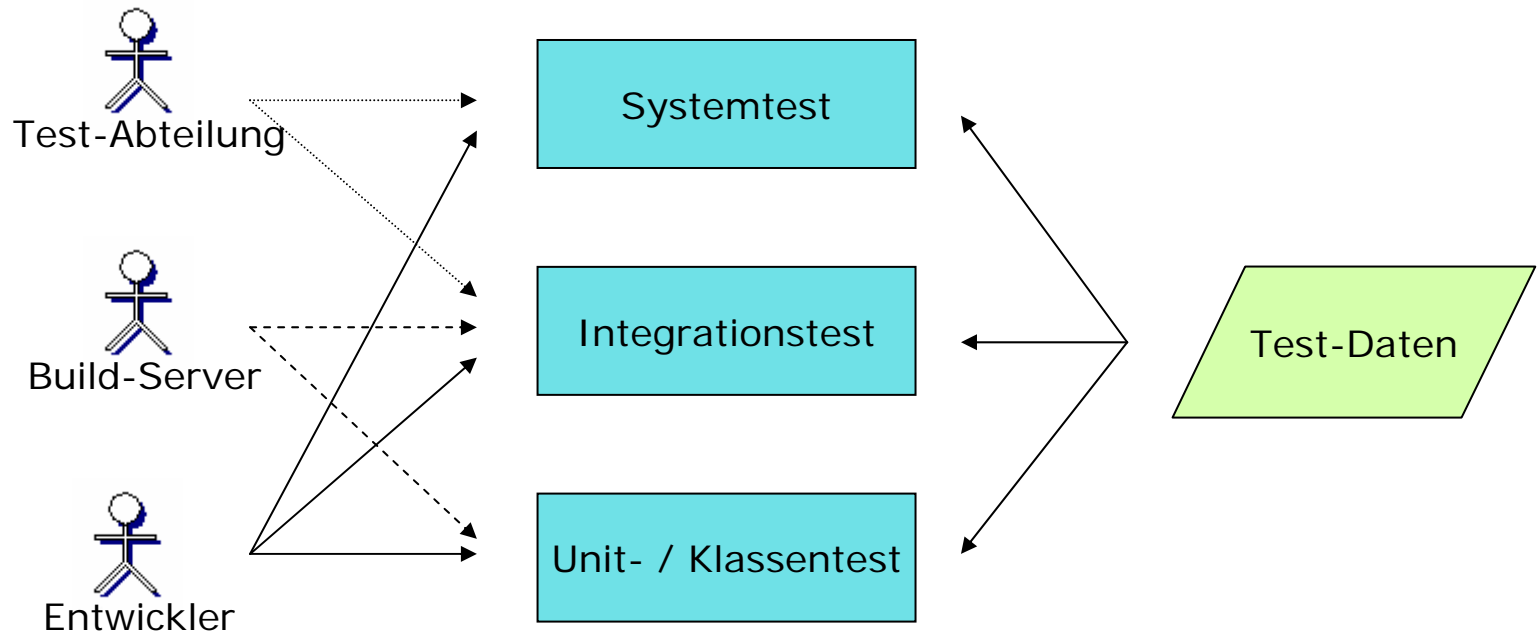
Testen im Software-Entwicklungsprozess



x| Grundsätzliche Fragestellungen zum Thema Test

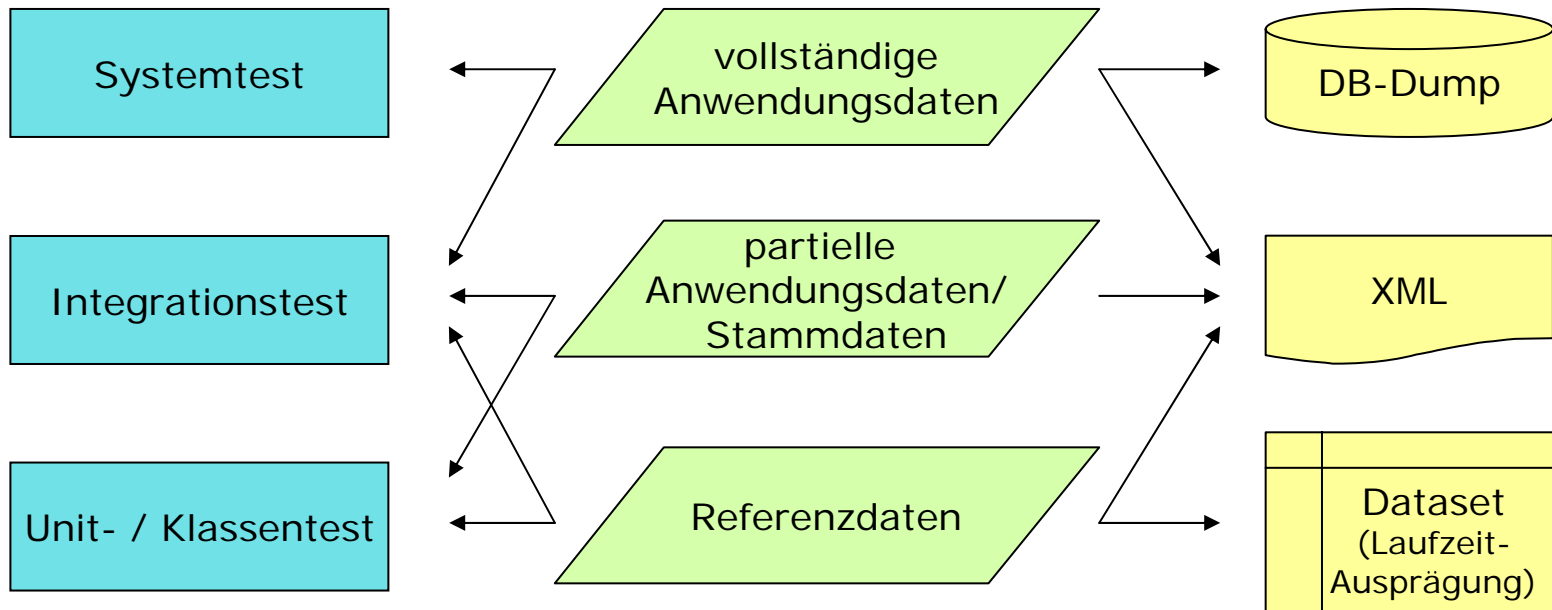
- | Wie leite ich Testcases aus den funktionalen Anforderungen ab?
- | Wie verwalte ich Testcases mit meinem Prozess?
- | Wie können Testabläufe automatisiert werden?
- | Wie werden die Testergebnisse ausgewertet?
- | Wie verwalte ich zugehörige Testdaten?

- x| Tests erfolgen auf unterschiedlichen Ebenen und werden von unterschiedlichen Rollen durchgeführt.



- x| Aussagekräftige Testdaten werden auf jedem Level benötigt.

- x| Testdaten werden in unterschiedlicher Granularität benötigt, je nach Art des zugrundeliegenden Tests.



- x| Testdaten können in unterschiedlichen Ausprägungen gehalten werden.

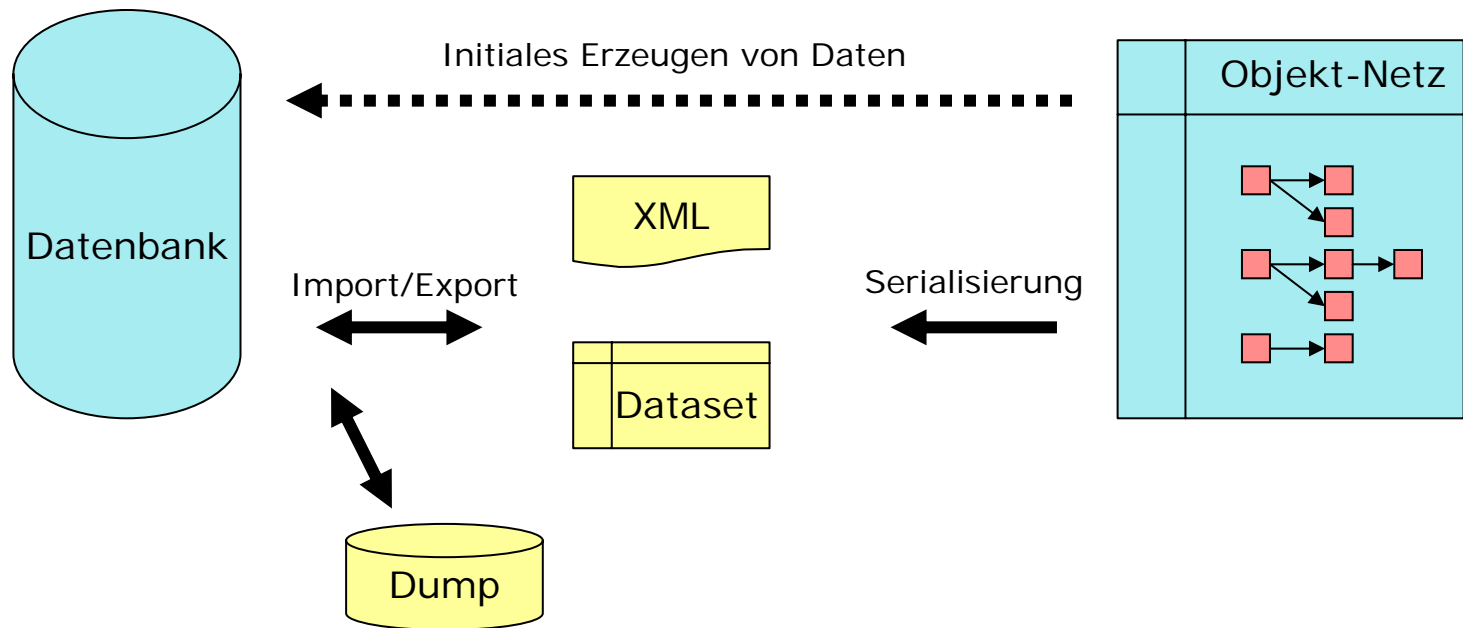
x| Ziel

- | Entwickler und Tester können auf realistische Anwendungsdaten zurückgreifen
- | Definierter Prozess der Datenbereitstellung
- | Effizientes Daten-Handling

„Testdaten und Revision der Anwendung sind synchron oder lassen sich einfach (=automatisch) synchronisieren.“

Erzeugung von Testdaten

- x| Zur Erzeugung von Testdaten kann eine Datenbank oder ein initiiertes Objektnetz als Quelle dienen.



- x| Testdaten, die aus unterschiedlichen Quellen stammen, können verglichen werden. (Test-Zusicherung)

x| Möglichkeiten zum initialen Erzeugen von Daten

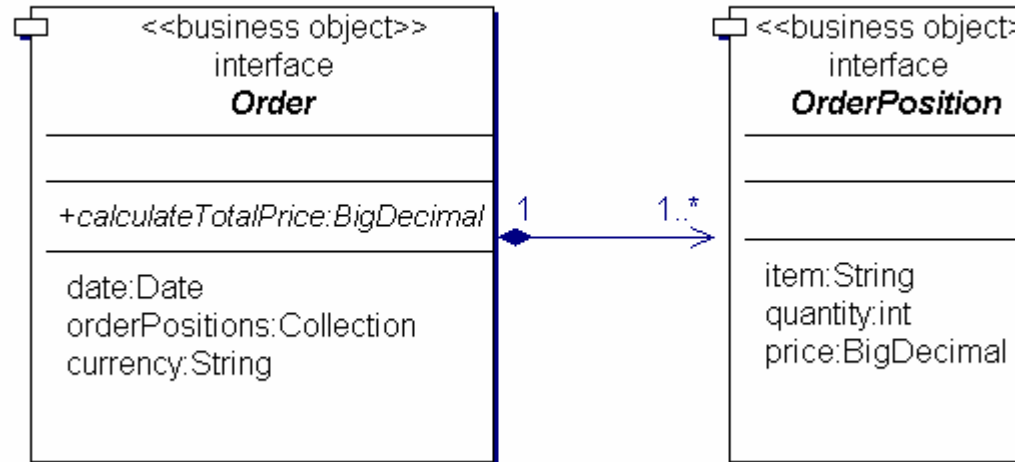
- Daten manuell über die Applikation einpflegen
- Produktivdaten verwenden, falls verfügbar
- Daten manuell über eine generische Applikation einpflegen
- Daten generieren, Meta-Informationen im Anwendungsmodell (Default Werte, Wertebereiche, etc.)
- Daten über Factories erzeugen (Quell-Code)

„Verwendung des Anwendungsmodells sichert Datenkonsistenz!“

Strategien zur Pflege der Testdaten

x| Vorüberlegung

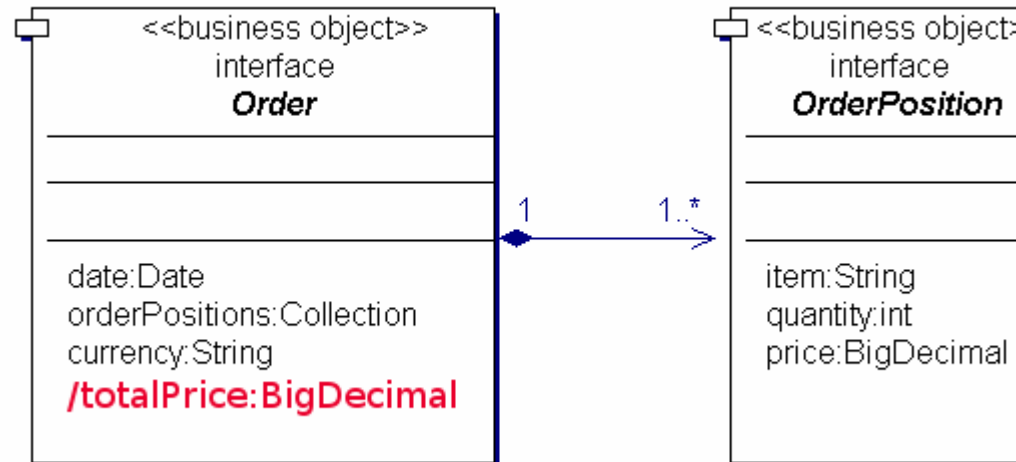
Anwendungsmodell Version 1:



x| Anforderung: Geschäftsmethode `calculateTotalPrice` wird als inperformant angenommen. Reporting soll Gesamtsumme ohne Aggregation aus der Datenbank lesen können.

- x| Lösungsansatz: Neues Attribut `total Price` (derived, persistent, mandatory) ersetzt die Geschäftsmethode `cal cul ateTotal Pri ce`.

Anwendungsmodell Version 2:



x| Naiver Ansatz, das Datenbank Schema anzupassen:

```
DROP TABLE ORDERS;  
CREATE TABLE ORDERS (  
    ID NUMBER PRIMARY KEY,  
    CURRENCY VARCHAR2 (5) NOT NULL,  
    ORDER_DATE DATE NOT NULL,  
    TOTAL NUMBER (22, 4) NOT NULL);
```

```
DROP TABLE ORDER_POSITION;  
CREATE TABLE ORDER_POSITION (  
    ID NUMBER PRIMARY KEY,  
    ORDER_ID NUMBER NOT NULL,  
    ITEM VARCHAR2 (32) NOT NULL,  
    QUANTITY NUMBER (22) NOT NULL,  
    PRICE NUMBER (22, 4) NOT NULL);
```

x| Konsequenzen dieses Ansatzes

- | Testdaten basierend auf Version 1 können nicht mehr geladen werden.
- | Existierende Entwickler-Daten werden bei der Integration gelöscht (drop statements)

„Müssen die Testdaten neu erstellt werden?“

x| Strategien

- 1) Migration von bestehenden Testdaten
- 2) Referenzdaten generieren/serialisieren

Migration von bestehenden Testdaten

x| Anpassung des Datenbank Schemas durch Migration

```
ALTER TABLE ORDERS ADD TOTAL NUMBER(22, 4);

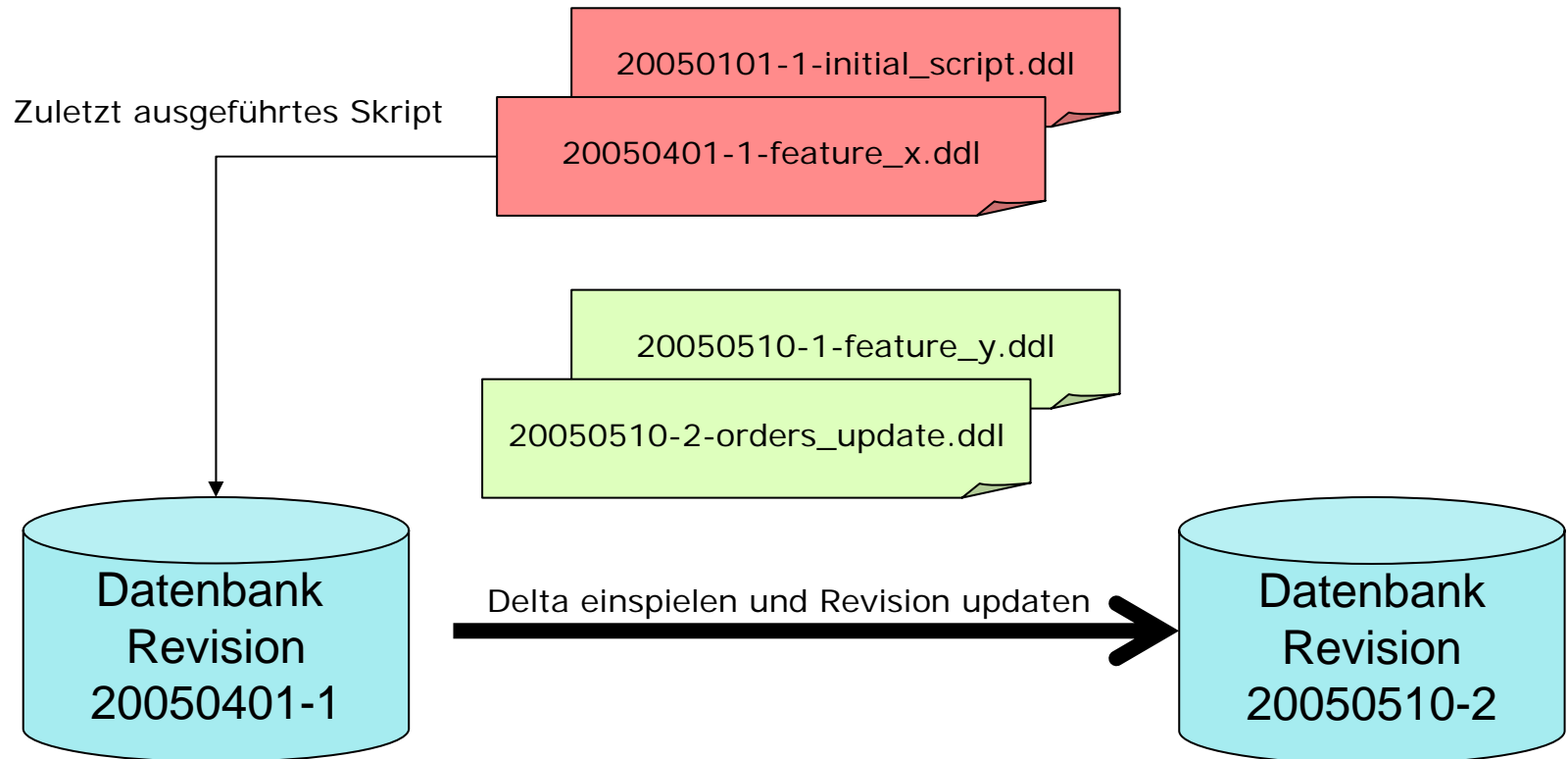
UPDATE ORDERS SET TOTAL =
  (SELECT SUM(PRICE*QUANTITY)
   FROM ORDER_POSITION
   WHERE ORDERS.ID = ORDER_POSITION.ORDER_ID);

ALTER TABLE ORDERS MODIFY TOTAL NUMBER(22, 4) NOT NULL;
```

x| Vorgehen

- | Pro Entwickler eine Datenbank/Datenbank Schema
- | Revision in der Datenbank persistent (YYYYMMDD-SEQ)
- | Pro Modelländerung ein Migrations-Skript mit Revisionspräfix
- | Aktualisierung des Datenbank Schemas bei Integration über einen Ant-Task
 - Die Revision wird aus der Entwickler Datenbank ermittelt
 - Migrations-Skripte, die eine neuere Revision aufweisen, werden auf der Entwickler-Datenbank ausgeführt
 - Die aktuelle Revision wird in die Entwickler-Datenbank geschrieben

x| Aktualisierung des Datenbank Schemas durch Ermittlung des Revisions-Deltas

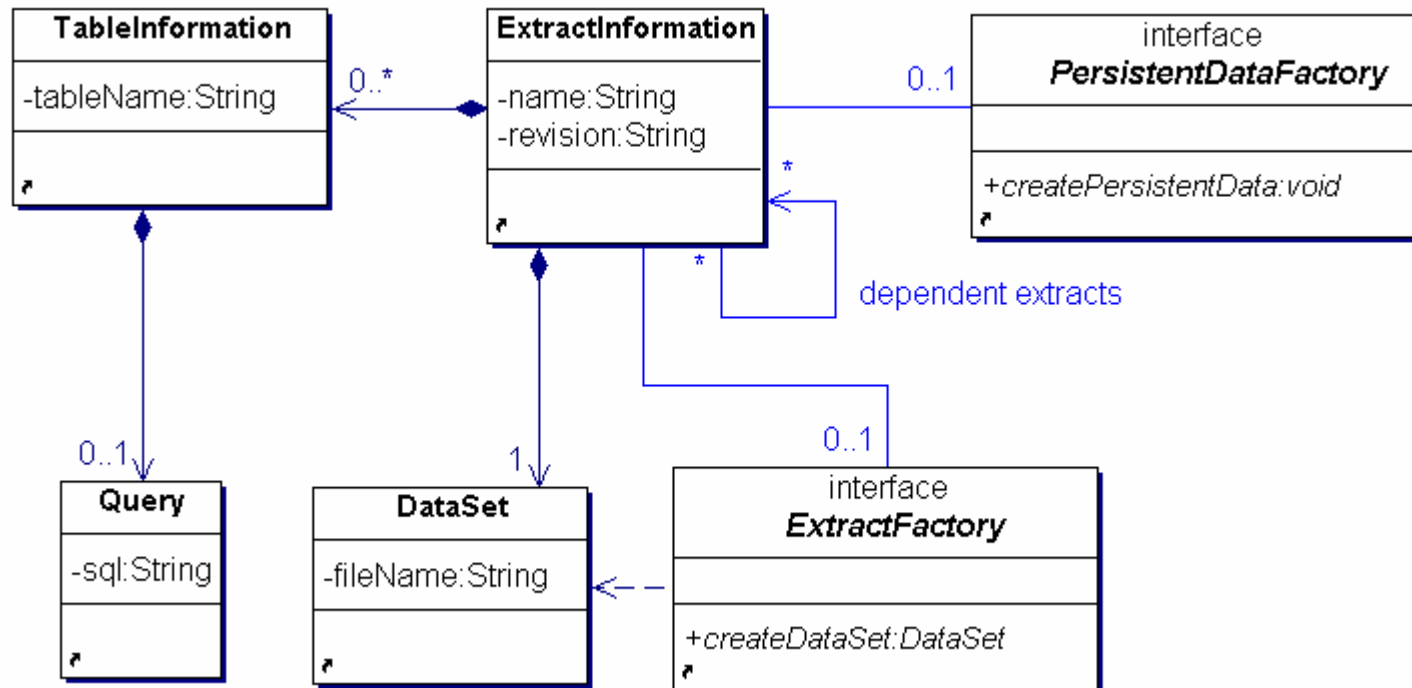


- x| Migration als prinzipielles Vorgehen bei der Entwicklung
 - | Migration ist verbindlich, sobald ein Produktivsystem vorhanden ist. Migration als prinzipielles Vorgehen vermeidet Bruch in der Vorgehensweise.
 - | Keine Entwickler Anwendungs-Daten gehen durch die Integration verloren.
 - | Integration des Produktiv-Systems wird bei der Entwicklung getestet.
 - | Möglichkeit, Testdaten ‚up to date‘ zu halten.

- x| Liegen Testdaten als vollständiger Datenbank-Export (Dump) vor, so können die Testdaten durch den vorher beschriebenen Migrations-Mechanismus aktualisiert werden.
 - => Import des Dumps initialisiert eine komplette Datenbank, für die lediglich das Delta in Bezug zu ihrer Revision eingespielt werden muß.

- x| Reine Testdaten (z.B. in XML Form) benötigen zusätzliche Informationen, um die Migration zu automatisieren.
 - | Name, zur Identifizierung
 - | Exportvorschrift
 - | File, um die Testdaten zu speichern
 - | Zugehöriger Revisionsstand
 - | Optional Factory zum initialen Erzeugen von Daten
 - => Testdaten + Beschreibung := Extract

x| Allgemeine Beschreibung von Testdaten



x| Konfiguration von Testdaten „Extracts“ in einer XML-Datei

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE extracts SYSTEM "extracts.dtd">
<extracts>

  <extract name="orders" revision = "20050510-2">
    <dataset filename="orders"/>
    <table name="orders"/>
    <table name="order_posi ti on"/>
    <table name="revi si on"/>
    <persi stentfactorycl ass= "OrderPersi stentDataFactory"
  </extract>

  <extract name="ful l _extract" revision = "20050510-2">
    <dataset filename="ful l _extract"/>
  </extract>

</extracts>
```

x| Resultierendes XML File zum Extract „orders“

```
<dataset>

<ORDERS ID="1" CURRENCY="EUR" ORDER_DATE="2005-04-18" TOTAL="5.5" />

<ORDER_POSITION ID="1" ORDER_ID="1" ITEM="4711" QUANTITY="1"
  PRICE="2.5" />
<ORDER_POSITION ID="2" ORDER_ID="1" ITEM="4712" QUANTITY="1"
  PRICE="3" />

<REVISION REVISION="1.0000" DELTA=" 20050510-2" />

</dataset>
```

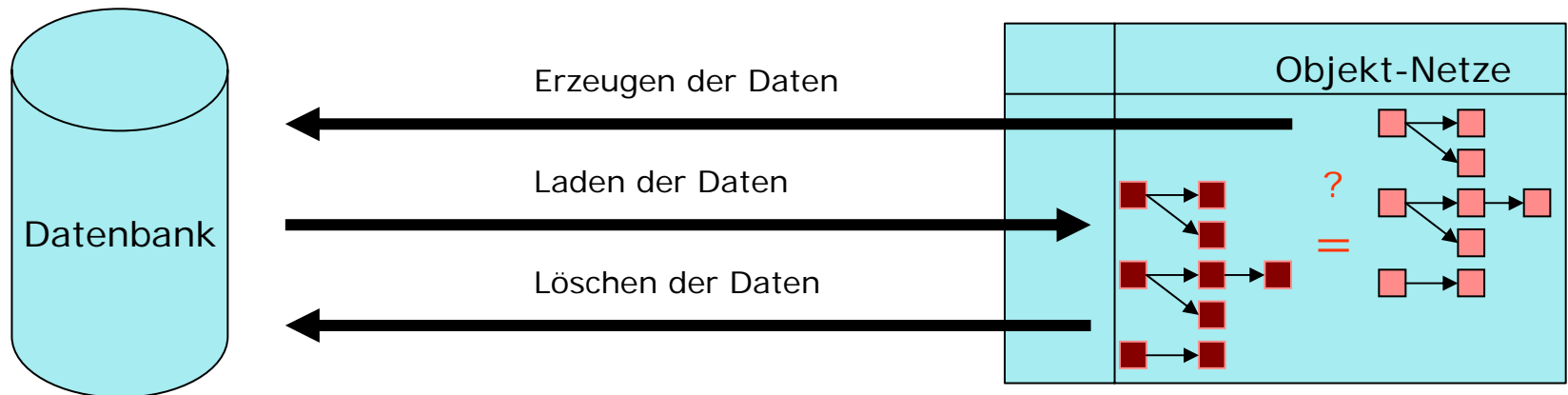
x| Mögliche Aktionen in Bezug auf Extracts

- | Laden eines Extracts (Optional Initiieren der Datenbank)
 - | Exportieren eines Extracts (Optional Löschen der Datenbank, Erzeugen von Daten)
 - | Migrieren eines Extracts
 - Laden des Extracts (Datenbank wird passend zur Revision des Extracts initiiert)
 - Datenbank auf aktuellen Revisionsstand migrieren
 - Exportieren des Extracts entsprechend seiner Exportvorschrift
- x| Die Aktionen können durch einen Ant-Task oder zur Laufzeit (Unit-Test) ausgeführt werden.

Erzeugung von Referenzdaten

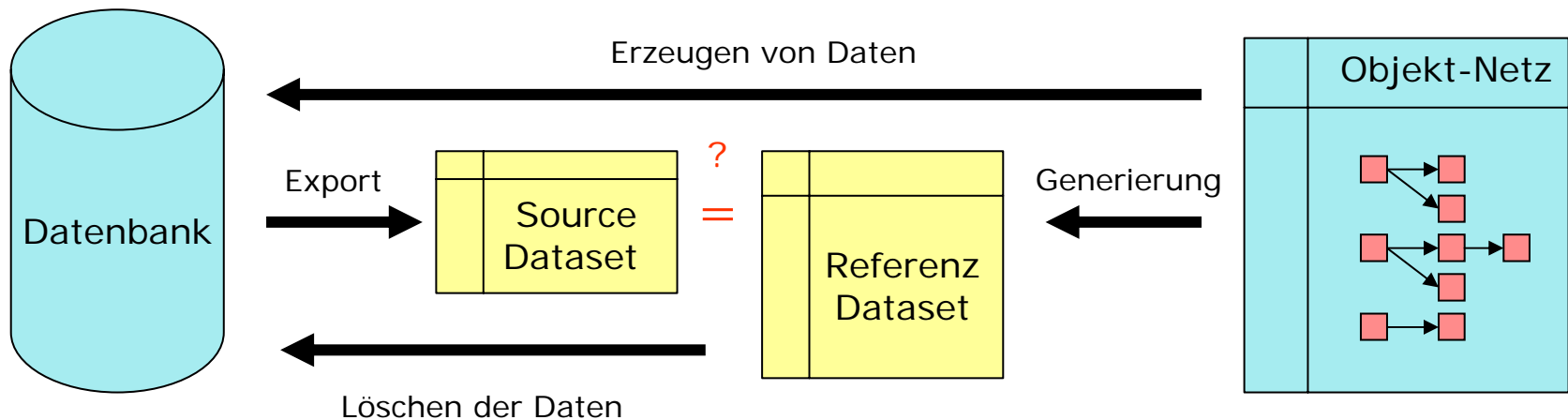
x| Zusicherung der Persistenz über einen Unit-Test

- | Erzeugen des Objektnetzes
- | Speichern über den Persistenzmechanismus
- | Laden eines Referenz-Objektnetzes
- | Vergleich der beiden Objektnetze
- | Löschen der Daten über den Persistenzmechanismus



x| Alternativer Ansatz

- | Erzeugen des Objektnetzes
- | Speichern über den Persistenzmechanismus
- | Generierung eines Referenz-Datasets
- | Vergleich mit dem persistenten Dataset
- | Löschen der Daten über einen Dataset



```
public void testCreateOrder() throws Exception{
    Transaction trx = getTransaction();

    Order order = (Order)createObject(Order.class);
    order.setCurrency("EUR");
    order.setDate(new Date());
    OrderPosition op =(OrderPosition)order.addNewToOrderPositions();
    op.setItem("123");
    op.setQuantity(3);
    op.setPrice(new BigDecimal ("30.00"));

    trx.commit();

    IDataSet ref = getNewCreatedDataSet();
    IDataSet source = getConnection().createDataSet(ref.getTableNames());
    assertEquals(reference, sourceDataSet);
}

public void tearDown() throws Exception {
    DatabaseOperation.DELETE.execute(getConnection(), getNewCreatedDataSet());
}
```

x| Vorteil

- | Quellcode für den Test wird übersichtlicher
- | Bei einer Modelländerung muss nur der Quellcode zum Erzeugen des Objektnetzes gepflegt werden
- | Vergleich und Löschen werden vom Testframework übernommen und sind effizient

Fazit

- x| Vorgestellte Vorgehensweisen vereinfachen die Pflege der Testdaten
 - | Reuse der Migrationsstrategie entkoppelt bestehende Testdaten von Modelländerungen
 - | Serialisierte Testdaten entsprechen immer dem aktuellen Modell

- x| Testen objektorientierter Software bleibt komplex

- x| Grad der Testabdeckung muß geschickt gewählt werden

- x| Bessere Qualität der Software durch
 - | Verwendung von Frameworks
 - | Generative Ansätze
 - | Reuse von Komponenten

Fragen & Diskussion