Agile BI in der Praxis
Agiles Testen

Andreas Ballenthin, Thomas Flecken

OPITZ CONSULTING Deutschland GmbH

Schlüsselworte:

Agile BI, SCRUM, Informatica, Oracle Warehouse Builder, Oracle Data Integrator, Business Objects, Projektmanagement, BI, DWH, Automatisiertes Deployment, Automatisierte Testen

Einleitung

Der Erfolg agiler BI-Projekte hängt in angesichts der kurzen Releasezyklen in großem Maße von einer hohen Testautomatisierung, Testabdeckung und Testfrequenz ab. Der Vortrag zeigt den engen Zusammenhang von automatisierten Deployments und den anschließenden Tests anhand einiger Praxisbeispiele aus der Oracle- und Informatica-Toollandschaft. Zudem wagen wir einen Blick auf die Möglichkeiten automatisierter Front-End-Tests mit SAP Business Objects.

Automatisiertes Testen

Die Einführung einer erfolgreichen Testautomatisierung bedarf einiger Grundlagen, welche wir nachfolgend betrachten wollen.

Zunächst werfen wir einen Blick auf die Teamzusammensetzung. Das agile Vorgehensmodell, nach dem wir entwickeln, sieht vor, dass die Teammitglieder sämtliche Bereiche des Wertschöpfungsprozesses abdecken. Das umfasst sowohl die Konzeption, die Entwicklung im Backend und im Frontend als auch die Unit-, Verbund- und Regressionstests. Entwickler und Tester sind als Rollen zu verstehen, die nicht dedizierten Personen zugewiesen sind, sondern von Story zu Story oder sogar von Task zu Task wechseln können.

Dem Team sollte der Mehrwert der Testautomatisierung stets bewusst sein. In einem ersten Schritt ist es hilfreich, die Implementierung von automatisierbaren Testfällen als Element der „Definition of Done“ aufzunehmen.

Eine weitere, wichtige Voraussetzung ist die Verantwortung des Scrum-Teams für das Testsystem. Es muss jederzeit in der Lage sein, Abläufe und Inhalte nach den Bedürfnissen des anstehenden Testlaufs anzupassen, ohne dabei auf eine weitere Partei zugreifen zu müssen. Das Team benötigt folglich die DBA-Rolle auf den Entwicklungs- und Testdatenbanken, Administratoren-Rechte für das eingesetzte ETL-Tool und Rechte auf den User, unter dem das ETL-Tool installiert wurde.

Die Praxis zeigt, dass testgetriebene Entwicklung (TDD) maßgeblich zum Erfolg der Story-Implementierungen beiträgt.

Im Wesentlichen geht man bei dieser Methode wie folgt vor:

1. Ein Testfall wird erstellt, bevor der Programmcode implementiert wird.
2. Der Testfall wird ausgeführt und schlägt wie erwartet fehl.
3. Der Programmcode wird implementiert.
4. Der Testfall wird erneut ausgeführt und zeigt, ob die Implementierung des Programmcodes erfolgreich war.

In diesem Kontext enthält ein Testfall nicht nur die möglicherweise auftretenden Fehlersituationen, sondern auch die Features des zu erstellenden Codes.

Die Vorteile dieser Vorgehensweise sind vielfältig. So erhalten wir auf diese Art und Weise ein genaues Bild von dem zu erwartenden Ergebnis, an dem wir uns bei der eigentlichen Programm-Implementierung orientieren können. Direkt im Anschluss können wir die Funktionalität testen und ggf. den Programmcode oder den entsprechenden Testfall anpassen. Zudem stellen wir sicher, dass der Test mit der Entwicklung Schritt halten kann. Durch das Persistieren der Testfälle in Subversion gehen sie nach dem Entwicklertest nicht verloren, sondern werden sofort für den Regressionstest verfügbar gemacht und eingesetzt.

Diese Arbeitsweise ist in der Regel für die Teammitglieder ungewohnt und erfordert anfangs eine gewisse Disziplin, bis sie selbstverständlich geworden ist. Ein einmal erstellter Testfall darf nicht als einzige Wahrheit gesehen werden, da bei der Implementierung des Programmcodes unerwartete Erkenntnisse über die Daten bzw. deren Qualität erlangt werden können, die ein grundsätzliches Umdenken zur Folge haben.

ETL/ELT-Deployment

Eine Automatisierung des ETL-Deployments ist nicht nur unabdingbar für eine funktionierende Testautomatisierung, sie sorgt auch für eine spürbare Reduzierung der Fehler, die beim manuellen Deployment auftreten können. Darüber hinaus ist die Performance der Kommandozeilen-Tools ungleich höher als gleichartige Operationen unter Verwendung von Client-Tools.

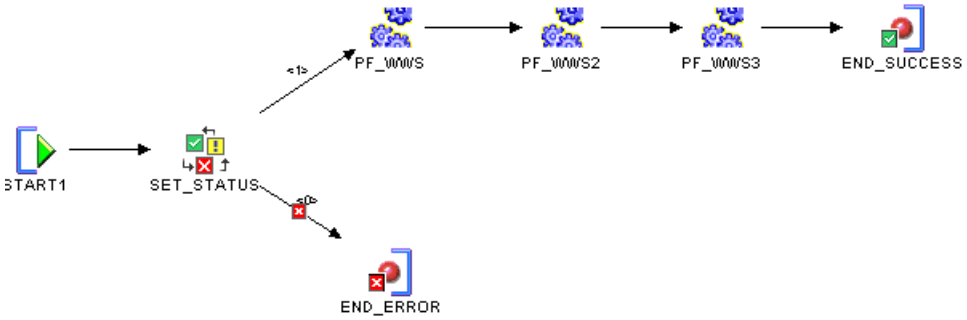
An ein ETL/ELT-Deployment stellen wir folgende Forderungen:

- Komplette Automatisierung
  Ein Deployment muss sowohl automatisiert nachts als auch automatisiert auf Anforderung durchgeführt werden können. Deshalb ist es notwendig, dass keinerlei manuelle Eingriffe oder Ergänzungen im Zuge des Deployments notwendig sind.
- Nachvollziehbarkeit
  Es muss nachvollziehbar sein, welche Artefakte ein Deploymentpaket beinhaltet und ob diese Artefakte erfolgreich deployed wurden.

- Vollständigkeit
  Es muss gewährleistet sein, dass ein Deployment alle relevanten Artefakte umfasst.
- Geringer Aufwand zur Bestückung des Deployments
  Für einen Entwickler muss es sehr einfach sein, Artefakte zu einem Deployment hinzuzufügen. Idealerweise werden Objektgruppen deployed.
- Kein DDL-Deployment
  Das ETL-Deployment soll kein DDL-Deployment, also kein Deployment von Nicht-ETL-Artefakten, durchführen. Dafür gibt es wesentlich geeignetere Werkzeuge.
- Test- oder Entwicklungsobjekte
  Nicht fertige Objekte oder Testobjekte (wie Workflows zum Test eines einzelnen Mappings) sollen nicht deployed werden.

Deployment OWB

Die bisher einfachste gefundene Möglichkeit, alle Mappings und Workflows eines Projektes zu deployen, ist ein Export mit maximaler Abhängigkeitstiefe. Wir integrieren jedes Mapping in einen Deployment-Workflow. Werden Workflows zur Ablaufsteuerung verwendet, so rufen wir diese als Subworkflows auf. Die Ausführung des Deploymentworkflows muss sofort zu einem Fehler führen.



Dann exportiert man mit

```
OMBEXPORT MDL_FILE 'c:/owbdeployment/deployment.mdl' FROM PROJECT
'DEPLOYMENT' COMPONENTS (PROCESS_FLOW_MODULE 'DEPLOYMENT') DEPENDEE_DEPTH
MAX OUTPUT LOG TO 'c:/deployment/export_deployment.log'
```

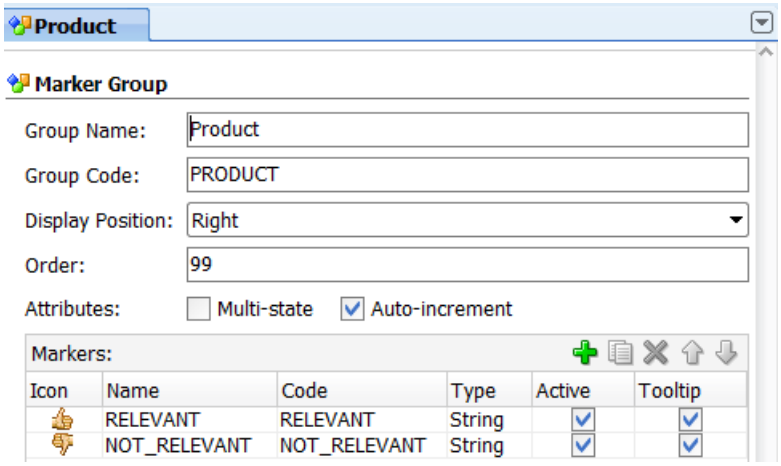
und importiert mit

```
OMBIMPORT

 MDL_FILE 'c:/owbdeployment/deployment.mdl' USE UPDATE_MODE MATCH_BY

UNIVERSAL_IDENTIFIER OUTPUT LOG TO 'c:/deployment/import_exployment.log'
```

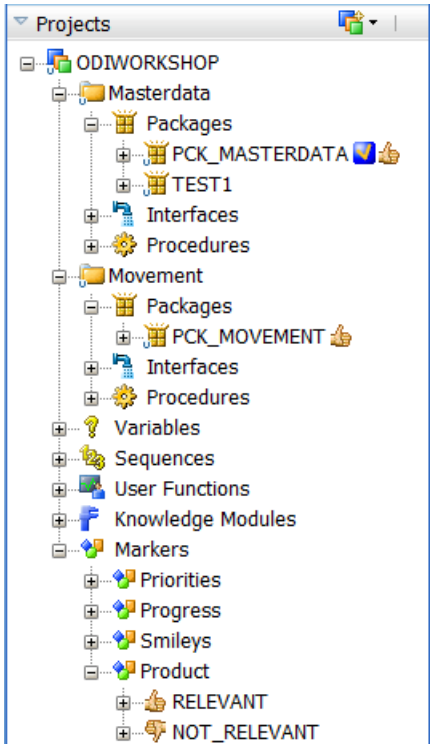
In der Zielumgebung kann man Mappings und Workflows mit OMBPlus deployen.

Deployment ODI

Wir betrachten in diesem Kapitel ausschließlich das Deployment von Szenarien, also im Grunde das Deployment aus einem Development Work Repository in ein Execution Work Repository. Um produktzugehörige Szenarien von Testszenarien unterscheiden zu können, definiert man am besten eine Marker Group "Product"



und markiert alle produktzugehörigen Szenarien - wir verwenden wie immer nur Package-Szenarien - als produktrelevant.



Nun können wir uns mit

```
SELECT scen_name AS pack_name
```

```
  FROM (SELECT LAST_VALUE(d.scen_name)
                 OVER (PARTITION BY c.pack_name
                 ORDER BY scen_version) AS scen_name,
               MAX(scen_version)
                 OVER (PARTITION BY c.pack_name ) max_scen_version,
               scen_version,
               c.pack_name
          FROM snp_obj_state a
          JOIN snp_state2 b on (a.i_state = b.i_state)
          JOIN snp_package c ON (a.i_instance = c.i_package)
          JOIN snp_scen d ON (c.i_package = d.i_package)
         WHERE state_code = 'PRODUCT_RELEVANT'
       )
 WHERE scen_version = max_scen_version;
```

alle relevanten Szenarien selektieren und so ein Script

```
startcmd OdiGenerateAllScen -PROJECT=ODIWORKSHOP -MARKER=PRODUCT_RELEVANT
```

```
startcmd OdiExportScen "-SCEN_NAME=PCK_MASTERDATA" "-SCEN_VERSION=-1" "-
FILE_NAME=c:\deployment\SCEN_PCK_MASTERDATA.xml"   "-FORCE_OVERWRITE=YES"
"-RECURSIVE_EXPORT=YES"  "-XML_VERSION=1.0"  "-XML_CHARSET=ISO-8859-1"  "-
JAVA_CHARSET=ISO8859_1"
```

```
startcmd OdiExportScen "-SCEN_NAME=PCK_MOVEMENT" "-SCEN_VERSION=-1" "-
FILE_NAME=c:\deployment\SCEN_PCK_MOVEMENT.xml" "-FORCE_OVERWRITE=YES" "-
RECURSIVE_EXPORT=YES"  "-XML_VERSION=1.0"  "-XML_CHARSET=ISO-8859-1"  "-
JAVA_CHARSET=ISO8859_1"
```

zum Export generieren lassen. Gleichzeitig erzeugen wir ein Script

```
startcmd OdiImportScen "-FILE_NAME=c:\deployment\SCEN_PCK_MASTERDATA.xml"
"-IMPORT_MODE=SYNONYM_INSERT_UPDATE"
```

```
startcmd OdiImportScen "-FILE_NAME=c:\deployment\SCEN_PCK_MOVEMENT.xml"
"-IMPORT_MODE=SYNONYM_INSERT_UPDATE"
```

zum Import. Die erzeugten XML-Files und das Importfile transportieren wir zum Zielserver und importieren dort.

Deployment Informatica

Den Inhalt eines jeden Deployment-Paketes legen wir durch je eine Repository-Query im Quell-Repository fest. Dort bestimmen wir die Workflows, die ausgeliefert werden sollen und konfigurieren die Query so, dass auch alle von den Workflows abhängigen Objekte selektiert werden. Das Ergebnis der Query dient als Basis für den vom Shell-Skript ausgelösten XML-Export. Das dadurch erzeugte Deployment-Paket wird inkl. Kontrolldatei auf den Server mit dem Ziel-Repository kopiert. Ein weiteres, parametrierbares Shell-Skript auf dem Zielserver stößt schließlich den Workflow an, der die XML-Datei importiert.

Die Protokollierung des ETL-Deployments erfolgt in einer Log-Datei, die am Ende des Prozesses auf dem Testserver abgelegt und in Subversion eingecheckt wird.

Backendtests

Eine wesentliche Grundfunktionalität ist das Aufsetzen oder Wiederaufsetzen auf dem letzten produktiv gesetzten Software- und Datenstand. Dazu exportieren wir am Tag einer Auslieferung auf das Abnahmesystem den erfolgreich getesteten Datenstand des Testsystems. So besteht in den Tests des nächsten Sprints immer wieder die Möglichkeit, diesen Datenstand zurückzusichern. Den Export dieses Datenstands bezeichnen wir als Baselinedumps. In der Regel werden Datenbankschemata um nicht notwendige Daten bereinigt, so enthalten unsere Stagingtabellen beim Export nur noch genau eine Zeile. Es muss unbedingt darauf geachtet werden, diese Baselinedumps auf einem separaten Storage zu sichern, weil sie eine sogenannte „Golden Source" darstellen. Statt jeden abhängigen Server an Subversion anzubinden, haben wir uns dazu entschieden, jedes auf einem abhängigen Server auszuführende Skript vor jeder Ausführung vom Testserver zum abhängigen Server zu transferieren und auszuführen. Führender Server ist also immer der Testserver. Im Fall der Baselinedumps besteht diese Notwendigkeit, denn impdp steht auf dem Testserver nicht zur Verfügung. Wenn ein komplettes Release Upgrade getestet werden soll, so ist der Daten- und DDL-Stand der Baselinedumps eine hinreichende Basis.

Mit Testplänen muss es möglich sein, sowohl Initialloads, Teilinitialloads (beispielsweise bei der Initialisierung einer neuen Faktentabelle) sowie Deltaloads zu testen. Aus unserer Erfahrung heraus reicht es nicht, mit Initialloads bzw. Teilinitialloads zu arbeiten, denn zeitliche Abgrenzungsprobleme wie Nachläufer fanden wir regelmäßig erst nach mehreren Deltaloads.

Idealerweise entstehen Testfälle bereits während der zuvor geschilderten testgetriebenen Entwicklung. Testfälle müssen gegen das Entwicklungs- und das Testsystem ausführbar sein.

Von uns implementierte Testfallkategorien sind beispielsweise

- Datenabgleiche innerhalb einer Schicht des DWHs (z. B. Aggregate vs. Detailfakten),
- Datenabgleiche zwischen den Schichten des DWHs,
- Datenabgleiche End-to-End (Fakt oder Dimension zum Quellsystem).
- Wurden Codierungsstandards eingehalten? Hier werden Prüfungen im ETL-Repository vorgenommen.
- DDL-Tests: Sind alle Foreign Keys deployed? Fehlen Tabellen- oder Spaltenkommentare?
- Hilfsskripte zum Warten auf das Ende von ETL-Prozessen,
- Erstellen von Flatfiles, die bei ihrer Verarbeitung definierte Metrikwerte generieren,
- Generierung „vorsätzlich falscher" Flatfiles zur Prüfung von Protokollfunktionalitäten.
- Sind Fehlertabellen leer?

Unsere Testfälle bestehen in der Regel aus drei Komponenten:

- Testfallregistrierung: Es muss gewährleistet werden, dass ein Testfall unter allen Umständen einen Eintrag im Testprotokoll generiert, selbst wenn der Testfall syntaktisch falsch ist.
- Aufruf des Testfalls: Den Aufruf eines Testfalls codieren wir in der Regel als Shell-Skript (zum Aufruf aus einem Testplan) und als Batchfile (zum Aufruf vom Entwicklerclient insbesondere im Rahmen der Testfallentwicklung)
- Testfall-Source-Code: In allen bisher codierten Testfällen ist dies ein SQL-Statement oder ein anonymer PL/SQL-Block.

Front-End-Tests

Die Anzahl der Regressionstests im Front-End nimmt mit voranschreitender Projektdauer stetig zu. Um dort dennoch eine gleichbleibend hohe Testabdeckung gewährleisten zu können, haben wir einige Front-End-Testfälle in Form von Reports erstellt, welche automatisiert durch einen Trigger ausgeführt und später durch das Auditing ausgewertet werden können. Die Vorgehensweise ist dabei wie folgt:

Nach Vollendung des Backendtests wird eine Semaphor-Datei erzeugt, welche vom Server des Front-End-Systems abgeholt und durch ein System-Ereignis in SAP BO konsumiert wird. An diesem System-Ereignis wiederum hängen Instanzen der automatisierten Testberichte, die ausgeführt werden sobald das System-Ereignis eintritt.

Das SAP-BO-Auditing schreibt detaillierte Informationen zur Berichtsausführung mit, wie z.B. Erfolg/Misserfolg der Ausführung und die Dauer. In einem weiteren Schritt ist es uns gelungen, auch die Anzahl der zurückgegebenen Zeilen aus dem Auditing auszulesen.

Der automatisierte Front-End-Test ermöglicht es somit, schnell darüber Auskunft zu bekommen, ob ein Bericht nicht mehr lauffähig ist, was auf veränderte Datenstrukturen zurückzuführen wäre. Zusätzlich hat es sich als sehr sinnvoll erwiesen, spezielle Berichte zu erstellen, welche die sogenannten Kontexte in SAP-BO prüfen.

Kontaktadresse:

Andreas Ballenthin
Thomas Flecken
OPITZ CONSULTING Gummersbach GmbH
Kirchstraße 6
D-51647 Gummersbach

andreas.ballenthin@opitz-consulting.com
thomas.flecken@opitz-consulting.com

Telefon: +49 (0) 2261-6001-0
Fax: +49 (0) 2261-6001-4000
Internet: www.opitz-consulting.com