Agile Methoden kommen bei der Neu- und Weiterentwicklung von BI-Projekten immer häufiger zur Anwendung. Ein wesentlicher Faktor für den Erfolg dieser Projekte ist das Testen, insbesondere im Backend-Bereich. Die Notwendigkeit für tägliche, automatisierte Testläufe ergibt sich aus den kurzen Release-Zyklen, wie sie in der agilen Entwicklung üblich sind. Zudem macht es die ständig wachsende Zahl an Regressionstests ab einem gewissen Punkt unmöglich, die Tests in einer annehmbaren Zeit manuell durchzuführen. Dieser Artikel zeigt, welche Voraussetzungen für automatisiertes Testen in BI-Projekten geschaffen werden müssen und welche Werkzeuge sich in der Praxis bewährt haben.

Agile BI in der Praxis – agiles Testen

Andreas Ballenthin und Thomas Flecken, OPITZ CONSULTING GmbH

Die Einführung einer erfolgreichen Testautomatisierung bedarf einiger Grundlagen. Zunächst werfen wir einen Blick auf die Team-Zusammensetzung. Das agile Vorgehensmodell, nach dem wir entwickeln, sieht vor, dass die Teammitglieder sämtliche Bereiche des Wertschöpfungs-Prozesses abdecken. Das umfasst sowohl die Konzeption, die Entwicklung im Backend und im Frontend als auch die Unit-, Verbund- und Regressions-Tests. 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 Test-Automatisierung 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 Test-Datenbanken, Administratoren-Rechte für das eingesetzte ETL-Tool (Informatica PowerCenter) 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 ist
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 gegebenenfalls 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 Team-Mitglieder ungewohnt und erfordert anfangs eine gewisse Disziplin, bis sie selbstverständlich geworden ist. Ein einmal erstellter Test-

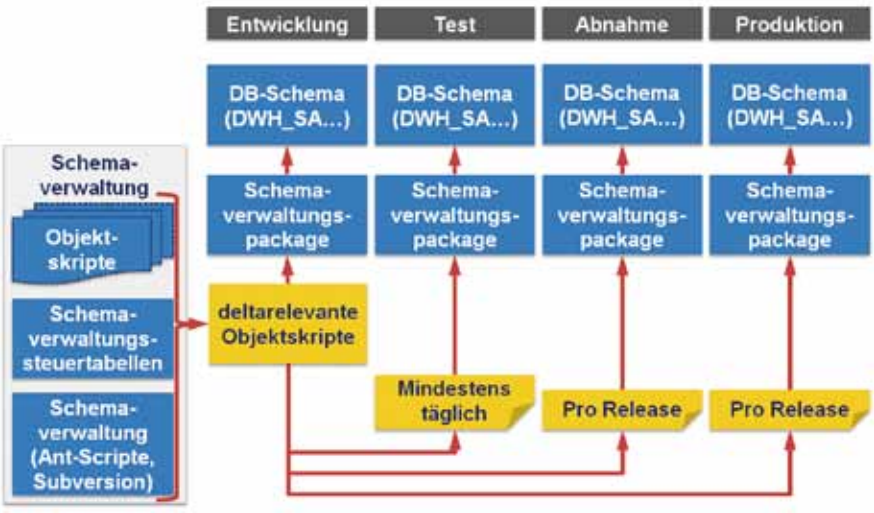

Abbildung 1: Automatisiertes Deployment – DDL, DCL, Daten-Migrationen

fall darf nicht als einzige Wahrheit gesehen werden, da bei der Implementierung des Programmcodes unerwartete Erkenntnisse über die Daten beziehungsweise deren Qualität erlangt werden können, die ein grundsätzliches Umdenken zur Folge haben.

Datenbank-Deployments per OC-Schemaverwaltung

Bevor wir uns Gedanken über die technische Umsetzung der Test-Automatisierung machen können, müssen wir zunächst sicherstellen, dass auch die Deployments der Datenbank- und ETL-Inhalte automatisch durchgeführt werden können.

Um die Datenbank-Inhalte auf einen von ihnen gewünschten Stand zu bringen, haben sich die Autoren für die Verwendung der OC-Schemaverwaltung (OCSV) entschieden. Dieses auf „Ant" basierende Werkzeug vergleicht den Ist-Zustand einer Oracle-Datenbank mit dem in Skripten definierten Soll-Zustand. Sofern Unterschiede festgestellt werden, führt die Schemaverwaltung die erforderlichen DDL-, DML- und DCL-Befehle aus (siehe Abbildung 1).

ETL-Deployments per Informatica-Kommandozeilen-Tools

In ihrem Projekt verwenden die Autoren Informatica für die Entwicklung und Steuerung der ETL-Prozesse. Eine Automatisierung des ETL-Deployments ist nicht nur unabdingbar für eine funktionierende Test-Automatisierung, 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 der Client-Tools, etwa des Repository-Managers.

Der Ausgangspunkt ist wie bei der Datenbank-Entwicklung der zentrale Testserver. Hier erstellen wir ein Unix-Shell-Skript, das den gesamten Ablauf des automatisierten ETL-Deployments kontrolliert. Der Inhalt eines jeden Deployment-Pakets wird durch je eine Repository-Query im Quell-Repository festgelegt. 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 einschließlich 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.

Frequenz der Deployments

Für die Test-Automatisierung haben die Autoren festgelegt, dass mindestens täglich Datenbank- sowie ETL-Deployments durchgeführt werden. So ist sichergestellt, dass beide Programm-Komponenten zueinander passen. Auch untertägige Deployments sind bei Bedarf problemlos durchführbar. Programmteile, die in einem noch nicht auslieferbaren Zustand sind, werden vom Entwickler nicht in Subversion eingecheckt und sind damit auch vom Deployment ausgenommen.

Baseline-Dumps

Die zentrale Komponente der automatisierten Tests ist der schon mehrfach angesprochene Testserver. Zu Beginn der Test-Automatisierung war dies eine virtuelle Maschine mit Windows-Betriebssystem, die Anfang des Jahres durch eine virtuelle Maschine mit Linux als Betriebssystem abgelöst wurde. Diese virtuelle Maschine wird nun auch von anderen Scrum-Teams des Kunden als Testserver genutzt.

Der Testserver benötigt über Secure-Shell (beziehungsweise WinSCP als skriptbarer Secure-Shell-Ersatz unter Windows) Zugriff auf alle notwendigen Server ohne Passwort-Eingabe. Im Kundenfall sind dies zwei Informatica-Server, ein Subversion-Server sowie der Data-Warehouse-Datenbankserver mit der DWH-Test-Instanz. Zusätzlich benötigt der Testserver SQLNet-Zugriff auf die DWH-Instanz und zur Informatica-Instanz für die Verarbeitung von SQL-Skripten. Im Optimalfall steht ein kompletter Oracle-Client zur Verfügung; existiert nur ein Oracle-Instant-Client, so können beispielsweise die Oracle-Clients der DWH-Instanz verwendet werden.

Eine wesentliche Grundfunktionalität ist das Aufsetzen oder Wiederaufsetzen auf dem letzten produktiv gesetzten Software- und Daten-Stand. 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 Da-

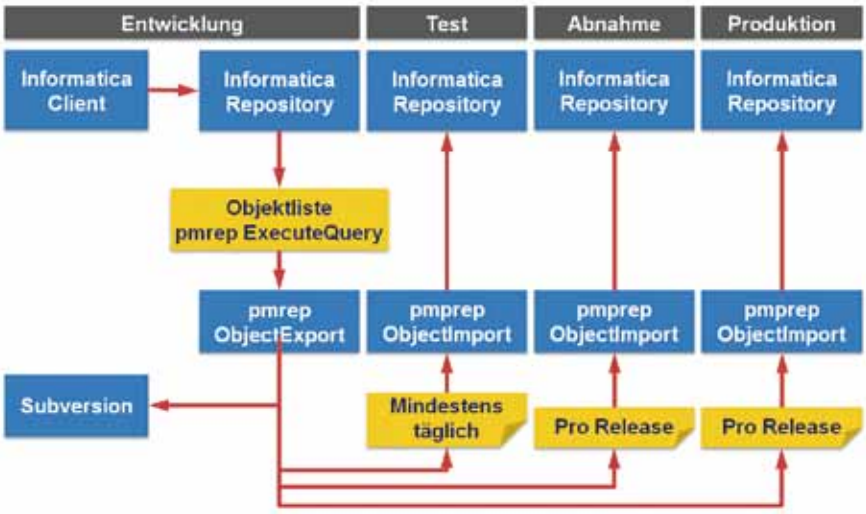

Abbildung 2: Automatisiertes Deployment mit Informatica

tenstands bezeichnen die Autoren als „Baseline-Dumps“. In der Regel werden Datenbank-Schemata um nicht notwendige Daten bereinigt; so enthalten unsere Staging-Tabellen beim Export nur noch genau eine Zeile. Es muss unbedingt darauf geachtet werden, diese Baseline-Dumps auf einem separaten Storage zu sichern, weil sie eine sogenannte „Golden Source“ darstellen.

Statt jeden abhängigen Server an Subversion anzubinden, haben die Autoren 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 Baseline-Dumps 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 Baseline-Dumps eine hinreichende Basis.

Initial- & Delta-Loads

Mit Testplänen muss es möglich sein, sowohl Initial-, Teil-Initial- (beispielsweise bei der Initialisierung einer neuen Faktentabelle) sowie Delta-Loads zu testen. Erfahrungsgemäß reicht es nicht, mit Initial- beziehungsweise Teilinitial-Loads zu arbeiten, denn zeitliche Abgrenzungsprobleme wie Nachläufer fand man regelmäßig erst nach mehreren Delta-Loads.

Testfälle

Idealerweise entstehen Testfälle bereits während der zuvor geschilderten testgetriebenen Entwicklung. Testfälle müssen gegen das Entwicklungs- und das Testsystem ausführbar sein. Implementierte Testfall-Kategorien sind beispielsweise:

- Datenabgleiche innerhalb einer Schicht des Data Warehouse (z.B. Aggregate vs. Detail-Fakten)
- Datenabgleiche zwischen den Schichten des Data Warehouse
- Datenabgleiche „End-to-End“ (Fakt oder Dimension zum Quellsystem)
- Prüfen auf Codierungs-Standards (Hier werden Prüfungen im ETL-Repository vorgenommen)
- Prüfen, ob alle Foreign Keys deployt sind und ob Tabellen- oder Spalten-Kommentare fehlen
- Hilfs-Skripte zum Warten auf das Ende von ETL-Prozessen
- Erstellen von Flat-Files, die bei ihrer Verarbeitung definierte Metrikwerte generieren
- Generierung vorsätzlich falscher Flat-Files zur Prüfung von Protokoll-Funktionalitäten
- Prüfen, ob Fehler-Tabellen leer sind

Die Testfälle bestehen in der Regel aus drei Komponenten:

- *Testfallregistrierung*
  Es muss gewährleistet sein, dass ein Testfall unter allen Umständen einen Eintrag im Test-Protokoll 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 Batch-File (zum Aufruf des Entwickler-Clients, insbesondere im Rahmen der Testfall-Entwicklung)
- *Testfall-Source-Code*
  In allen bisher codierten Testfällen ist dies ein SQL-Statement oder ein anonymer PL/SQL-Block

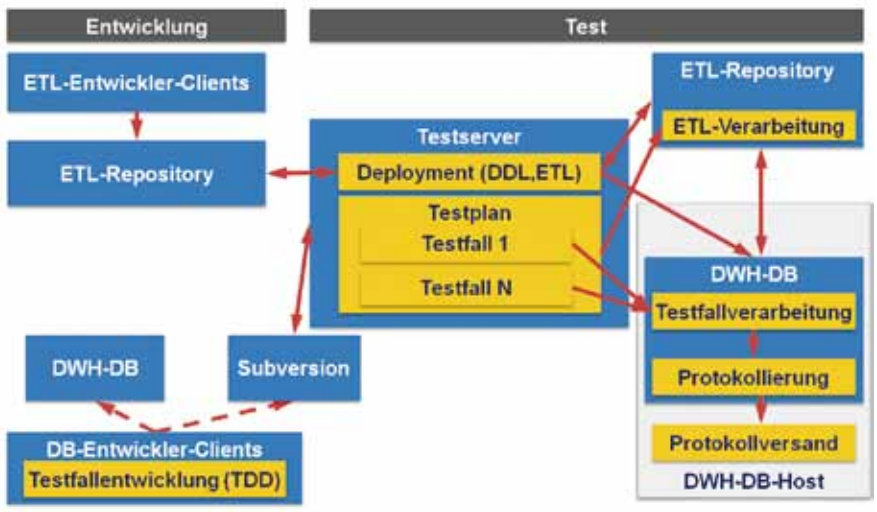

Abbildung 3: Die Architektur der Test-Suite

Diese Methodik wird nachfolgend am Beispiel des Prüfens von Spalten-Kommentaren illustriert. Die Autoren haben sich die Konvention auferlegt, dass jede Tabelle in jeder Spalte einen Spalten-Kommentar erhält, abgesehen von Stage-Tabellen und systemgenerierten Tabellen wie „SYS_EXPORT%“, die von Export Dump generiert werden. Die Testfall-Registrierung „sqlplus /nolog @register_testfall 01010_DDL $1“ sorgt nun dafür, dass der Testfall einen Testprotokoll-Eintrag erzeugt. Später in der Abarbeitung der Testfall-Gruppe wird der Testfall mit „sqlplus /nolog @01010_DDL.sql $1“ aufgerufen und es wird ein Skript ausgeführt (siehe Listing 1). Nun gibt es drei mögliche Testergebnisse:

- *Der Testfall ist erfolgreich*
  Dann wird der Status des Testfalls auf „erfolgreich“ geändert
- *Der Testfall ist nicht erfolgreich*
  Dann wird der Status des Testfalls auf „nicht erfolgreich“ geändert
- *Der Testfall wurde nicht aufgerufen oder ist syntaktisch falsch*
  Dann bleibt der Testfall mit dem Status „registriert“ im Test-Protokoll stehen.
  Dies kann bei Tabellenstruktur-Änderungen, also im Rahmen von Regressionstests, aber auch bei noch nicht fertig codierten Testfällen im Rahmen des Test Driven Developments der Fall sein.

Synthetische Testdaten anreichern und löschen

Bei einer Reihe von Testfällen, die zum Beispiel ein korrektes Verhalten in Feh-

```
declare
  v_testfall varchar2(10)  := '01010_DDL';
  v_result   varchar2(1)   := 'E';
  v_notes    varchar2(255) := NULL;
begin
  select case when count(*)=0 then 'S' else 'E' end,count(*)||' feh-
lende Kommentare'
    into v_result,v_notes
    from (
           select 1
             from all_col_comments c
            inner join all_tables t
               on c.owner=t.owner
              and c.table_name=t.table_name
            where t.owner like 'DWH%'
              and t.table_name not like 'SYS_EXPORT%'
         );
  result_testfall (v_testfall,v_result,v_notes);
  commit;
exception when others then
  result_testfall (v_testfall,v_result,v_notes);
  commit;
end;
/
```

Listing 1

ler-Situationen prüfen, sind in der Regel Anreicherungen mit synthetischen Testdaten nötig. Im Anschluss werden die Daten gelöscht. Abhängig von der Verarbeitungsdauer der ETL-Prozesskette haben die Autoren zwei Verfahren implementiert:

- Wenn die Prozesskette zeitlich eher lange dauert, werden alle nötigen synthetischen Testdaten im Übernahme-Prozess aus dem Quellsystem in die Staging Area oder als Zwischenschritt zwischen Staging Area und Core angereichert
- In allen anderen Fällen werden zunächst alle synthetischen Daten verarbeitet, geprüft und verworfen, um danach die Nutzdaten zu verarbeiten

Zum Löschen der angereicherten Daten sind drei Verfahren implementiert:

- Löschen per SQL. Dies setzt voraus, dass man eine Abgrenzung der Daten finden kann, was in Aggregationen unter Umständen nicht sinnvoll möglich ist.
- Zurücksetzen auf den vorherigen Datenstand vor der Verarbeitung über Export Dump, hinterher über Import Dump
- Ausführen von Flashback Table nach der Verarbeitung

Testpläne

Eine Testgruppe ist eine sinnvolle Zusammenstellung von Testfällen und die kleinste sinnvoll von Testplänen aufzurufende logische Einheit. Die Autoren haben beispielsweise Testgruppen für folgende Aktivitäten erstellt:

- Das Einspielen von Baseline-Dumps
- Den Initial- oder Delta-Load aus dem Vertragsmanagement-System
- Den File-basierten Load von Umsatzdaten
- Die Verarbeitung von Testfällen zum Vertragsmanagement oder von Umsatzdaten

Die Testpläne sind somit Zusammenstellungen von Testfall-Gruppen. Von einem Test-Operator wird täglich die Entscheidung des Teams darüber herbeigeführt, welche Testfall-Gruppen und somit welcher Testplan in der kommenden Nacht laufen soll. In der Regel wird täglich nachts getestet.

Auf dem Linux-Testserver werden Testpläne ganz klassisch über cron verwaltet. Unter Windows fungiert die Windows-Aufgabenplanung als Scheduling-Komponente.

Fazit

Mit den geschilderten Maßnahmen erreicht das Scrum-Team eine hohe Softwarequalität. Die vollständige Test-Automatisierung sorgt für ein frühes Erkennen von Fehlern und Datenqualitätsproblemen. Die Testabdeckung ist konstant hoch, manuelle Tests werden weitgehend vermieden. Die Investition in eine Test-Automatisierung ist sowohl aus technischen als auch aus betriebswirtschaftlichen Gesichtspunkten insbesondere für Agile BI unverzichtbar.

Andreas Ballenthin
andreas.ballenthin@
opitz-consulting.com

Thomas Flecken
thomas.flecken@
opitz-consulting.com