Der Lebenszyklus einer APEX Applikation: Managing the Change

Dietmar Aust
Opal-Consulting
Köln

Schlüsselworte:

Oracle APEX, Konfigurationsmanagement, PL/SQL, Lebenszyklus

Einleitung

Jede Applikation verändert sich mit der Zeit. Diese Versionen in den unterschiedlichen Umgebungen (Entwicklung, Test, Produktion) im Auge zu behalten wird schnell zu einer nicht-trivialen Aufgabe, die jeder Entwickler lösen muss. Dieser Vortrag beschreibt einige Best Practices für das Konfigurationsmanagement einer APEX Applikation.

Unser Vorgehen ist in den letzten drei Jahren in einem Kundenprojekt entstanden, wurde kontinuierlich angepasst und hat sich mittlerweile vielfach bewährt. Es umfasst Methoden für das Management der Anforderungen, eine standardisierte Struktur des Dateisystems, automatische DDL-Extraktion und die Integration mit Subversion. Das Verfahren ist übersichtlich und einfach umzusetzen. Durch die stringente Umsetzung einfacher Konventionen wurde die Qualität und Zuverlässigkeit in den letzten Releases deutlich gesteigert.

Da APEX Applikationen zu einem sehr großen Teil PL/SQL basiert sind, betrifft der Schwerpunkt dieses Vortrages die saubere Propagation der Änderungen über die verschiedenen Systeme hinweg, ist somit auch in reinen PL/SQL Umgebungen von Interesse.

Hintergrund / Problemstellung

Seit 2007 entwickeln wir die APEX-Applikation „Spots“ bei der Telekom Shop Vertriebsgesellschaft mbH als zentrales Shop-Informationssystem mit Schnittstellen zu anderen relevanten Systemen. Üblicherweise sind 2-3 Entwickler im Team. Die APEX Applikation besteht aus 140 Seiten, das Datenmodell aus 200 Tabellen, 100 Packages und insgesamt 3000 Datenbankobjekte.

Wir haben pro Jahr vier Releases, im Durchschnitt 100 (50-180) geänderte bzw. neu hinzugefügte Datenbankobjekte. Für ein Release benötigen wir in der zwei- bis dreiwöchigen Testphase normalerweise vier interne Revisionen mit dem Test-Team.

Mit der Zeit haben wir einige Problemfelder identifiziert, für die wir aktiv nach Lösungen gesucht haben. Die Probleme, mit denen wir zu kämpfen hatten, waren sowohl rein technischer als auch organisatorischer Natur, beide hatten jedoch Auswirkungen auf die Qualität bzw. verursachten Zusatzaufwand.

<u>Technische Probleme</u>

Anfangs hatten wir noch kein Werkzeug für die Versionskontrolle eingesetzt. Daher kam es gelegentlich vor, dass wir uns gegenseitig Änderungen an den Datenbank-Packages, Views oder Triggern überschrieben hatten.

Der Versionsstand der Dateien, Datenbank-Instanzen oder Oracle Schemata war nicht immer deutlich erkennbar:

- Welchen Versionsstand hatten die Dateien im Dateisystem (Installationsdateien, WAR-Dateien, Javascript, CSS oder auch Excel-Lieferungen)?
- Welche Version der Applikation ist gerade in der Entwicklungs-, Test-, Abnahme- und Produktionsumgebung installiert? Wann wurde sie jeweils geändert?
- Auf welcher Instanz wurde das Korrekturskript xyz ausgeführt? Und wo noch nicht?

Für die Auslieferung eines Patches oder eines neuen Software-Releases mussten jeweils DDL-Skripte für alle Änderungen am Datenmodell sowie DML-Skripte für die Aktualisierung der Konfigurationsdaten bereitgestellt werden. Immer lückenlos alle Änderungen zu propagieren war regelmäßig extrem aufwändig und wir mussten nacharbeiten, bis die Abnahme endlich den geplanten Stand aufwies.

Nicht-technische Probleme

Die Probleme waren jedoch nicht nur rein technischer Natur. Wir waren sehr unzufrieden mit der operativen Abwicklung im Projekt. So mussten wir folgende Fragestellungen jederzeit im Blick behalten:

- Was sind die aktuell geplanten Anforderungen im Release? Wenn es eng wird, was können wir dann noch schieben (Muss-Anforderungen, Kann-Anforderungen)?
- Wer arbeitet jetzt genau woran und wie ist der Status? Wie hoch ist dann der Gesamtaufwand pro Entwickler? Kann jemand noch etwas übernehmen?
- Wenn wir in das nächste Meeting mit der Fachabteilung gehen, was sind die offenen Punkte und Fragen?
- Wie hoch war jeweils die Schätzung zu den einzelnen Funktionen? Und sind wir noch in der Zeit? Wann muss ich melden, dass es ein Problem gibt?
- Die „lästige“ Dokumentation:
    - Welche Funktionen sind nun tatsächlich in diesem Patch/Release geliefert worden?
    - Welche Use-Cases haben sich geändert und was muss durch die Testabteilung validiert werden?
    - Die neuen und geänderten Funktionen müssen in die Gesamtdokumentation wieder eingefügt werden.

Die Lösung

Wir haben aktiv nach Lösungen zu den oben genannten Fragestellungen gesucht. Damit es längerfristig funktionieren kann, musste unsere Lösung folgende Kriterien erfüllen:

- Einfachheit (nur wenige Regeln, die einfach zu befolgen sind)
- Transparenz (es muss möglichst offensichtlich sein, was zu tun ist)
- Konsistenz (möglichst keine „Sonderlocken“, die gleichen Strukturen überall verwenden)
- Sicherheit (das Verfahren muss es mir erschweren, Fehler zu machen)

Welche Objekte haben wir für die Umsetzung betrachtet? Unser Konzept haben wir umfassend ausgelegt, so dass die Anforderungen, gelieferte Dateien, Datenbankobjekte und auch die Dokumentation mit betrachtet wird.

Die von uns erarbeitete Lösung besteht somit aus den folgenden Bestandteilen:

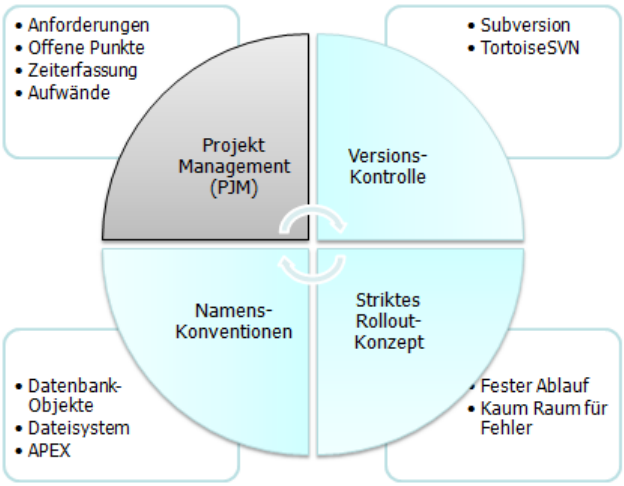

Abb. 1: Bestandteile der Lösung

Projektmanagement

Wir haben intern ein Werkzeug für das Projektmanagement implementiert, um die wichtigsten Fragestellungen per Knopfdruck beantworten zu können. Da wir dort ebenfalls unsere Zeiten erfassen, können wir ebenfalls den geplanten Aufwand den geleisteten Stunden gegenüberstellen, um so einfacher Budgetüberschreitungen feststellen zu können. Wir haben es leichtgewichtig als operatives Werkzeug umgesetzt. Die meisten anderen Werkzeuge unterstützen die Planung, unser Werkzeug eher die Bedürfnisse der Entwickler.

Der Teil des Projektmanagements ist nicht der Schwerpunkt des Vortrages, dennoch ist dies für uns ein extrem wichtiger Erfolgsfaktor. Die hier sonst üblichen Verfahren auf Basis von Excel Dateien sind an vielen Stellen nicht praktikabel. Eine Applikation auf Basis einer Datenbank ist unbedingt zu empfehlen.

Versionskontrolle

Für die Versionskontrolle haben wir Subversion auf einem zentralen Server eingesetzt. Jeder Entwickler hat dann unter Windows TortoiseSVN als Client genutzt. Dieser ist direkt im Windows Explorer integriert und stellt visuell mit überlagernden Icons dann den Zustand der einzelnen Verzeichnisse und Dateien dar. Somit kann man sofort erkennen ob eine Datei unverändert, geändert oder neu hinzugefügt wurde. Außerdem kann man mit den Kontextmenüs dann direkt auf den Dateien bzw. Ordnern die Subversion-Befehle ausführen.

Die Basis für alle Dokumente und Skripte ist das Dateisystem. Die Versionskontrolle in der Datenbank haben wir über den simplen Checkin-Checkout Mechanismus in Quest Toad umgesetzt.

Wenn wir ein Datenbank-Objekt (Tabelle, View, Package, etc.) ändern oder neu anlegen, dann gehen wir wie folgt vor:

- Wir setzen eine exklusive Sperre für das Datenbank-Objekt mittels Quest Toad.

- Die korrespondierende Datei (mit den DDL-Anweisungen) speichern wir im Dateisystem zu dem aktuellen Patch bzw. Release.
- Im aufrufenden Patch-Skript wird dann die Datei manuell registriert.

Namenskonventionen

Klare Namenskonventionen für alle Objekte und Dateien haben einen unschätzbaren Wert. Sie schaffen Transparenz und Übersichtlichkeit, vor allem weiß jeder sofort Bescheid, wie die Dinge zu benennen sind.

Datenmodell

Welche Konvention man nun letztlich verwendet, ist nicht entscheidend. Wichtig ist nur, dass es eine einfache, transparente Konvention gibt. Außerdem muss sichergestellt werden, dass die Einhaltung der Konvention regelmäßig überprüft wird bzw. sogar zuerst eine Freigabe zu erfolgen hat.

Verzeichnisse

Generell werden die Verzeichnisse im Singular benannt, also „Anforderung“ anstelle von „Anforderungen“. Prinzipiell ist es egal, es wird für jeden Fall Vor- oder Nachteile geben. Es ist von Vorteil, sich für eine Sichtweise zu entscheiden.

Generelle Übersicht

Verzeichnis	Kommentar
10-Projektrichtlinie	Coding Guidelines, Namenskonventionen
20-Vertragliches	Angebote, Rechnungen, Leistungsscheine
30-Projektmanagement	Meetings (geordnet nach Datum), Aufwandsschätzungen, Releaseplanung, etc.
40-Umsetzung	Die eigentliche Umsetzung des Releases
40-Umsetzung > 10-Anforderung	Die Anforderungen
40-Umsetzung > 20-Spezifikation	Die genaue Spezifikation der Anforderungen. Hier befindet sich sowohl die Spezifikation des aktuellen Releases als auch die Gesamtbeschreibung des Systems. Bei uns sind alle Anforderungen durchnummeriert, daher haben wir hier noch Unterverzeichnisse, in denen wir die Details (Emails, Testdaten, Screenshots, etc.) sammeln, z. B.: Spezifikation-Detail\001-Anmietvertrag Spezifikation-Detail\002-Kunde-verlängert-Auftrag
40-Umsetzung > 30-Design	Datenmodell, Architekturdiagramme
40-Umsetzung > 40-Implementierung	Die Source-Dateien
40-Umsetzung > 50-Test	Testkonzept, Testfallbeschreibung und Ergebnisprotokolle. Die Skripte für den Test befinden sich im Verzeichnis: 40-Implementierung\test
40-Umsetzung > 60-Betrieb	Betriebsdokumente, also Betriebshandbuch, Installationsanweisung, Datenschutzkonzept, etc.
50-Lieferung >	Die Installationspakete der Software mit Release Notes, Betriebshandbuch, Installationsanweisungen, etc.

60-Betrieb >	Dieses Verzeichnis ist aus der Sicht des Betriebes zu verstehen. Ist der Betrieb ausgelagert an andere Dienstleister, werden sie diese Struktur bei sich haben. Hier befinden sich die Skripte für den 3rd Level Support.
70-Referenz >	

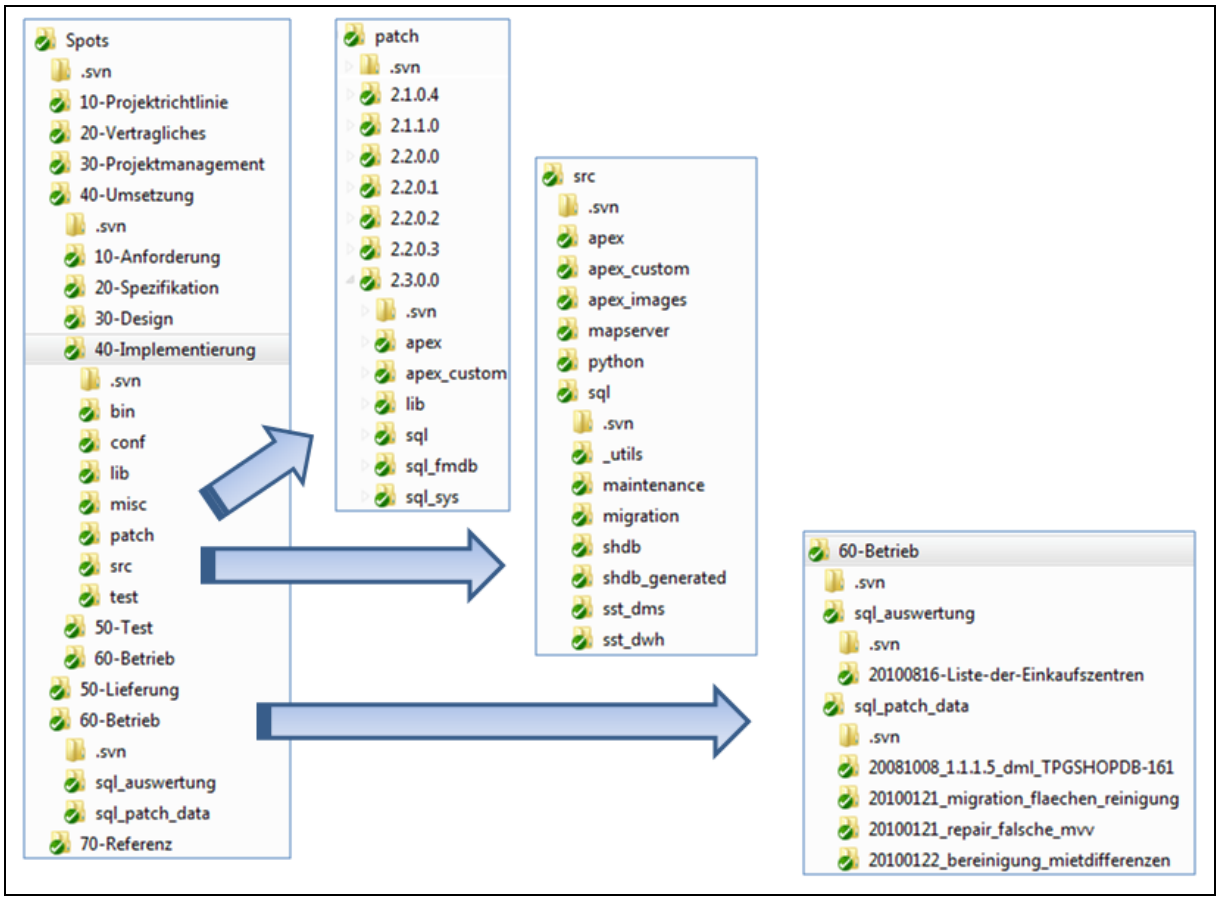

Abb. 2: Projektverzeichnisse

Verzeichnis 40-Umsetzung > 40-Implementierung > src

Verzeichnis	Kommentar
src	Hier befinden sich alle Source Dateien, geordnet nach der Programmiersprache
src > apex	Die APEX Applikationsdateien, Exporte der Themes und statischen Dateien z.B. f104_Spots_v2.2.0.0.sql
src > apex_custom	Dateien, die auf dem Applikationsserver ausgeliefert werden. Bei uns im Projekt verwenden wir den virtuellen Pfad /apex_custom/ für die Auslieferung spezieller Javascript- und CSS Dateien. Das /i/

	Verzeichnis sollte nicht verwendet werden!
src > apex_images	Obsolet, sollte nicht mehr verwendet werden. Dies stammt noch von der Verwendendung des /i/ Verzeichnisses von APEX.
src > python	In unserem Fall haben wir Python Skripte im Einsatz. Je nach Bedarf würde man dann auch src > java, src > perl oder src > php verwenden.
src > sql	Hier sind die Datenbank-Sourcen zu finden. Sie sind nach dem jeweiligen Datenbank-Schema organisiert.
src > sql > maintenance	Hier sind die Wartungsskripte, die dem Betrieb dann für die regelmäßigen Wartungsaufgaben zur Verfügung gestellt werden.
src > sql > shdb	Unser zentrales Datenbank-Schema lautet SHDB. Hier befinden sich die Skripte, die manuell erstellt werden und nicht automatisch generiert werden können bzw. sollen. Z.B. haben wir hier die Einrichtungsskripte für Advanced Queueing, die Installation der Basis-Konfigurationsdaten sowie die Einrichtung der Datenbank-Jobs.
src > sql > shdb_generated	Hier finden sich die Sourcen, die automatisch aus der Datenbank extrahiert wurden.

Verzeichnis 40-Umsetzung > 40-Implementierung > patch

Verzeichnis	Kommentar
patch	Hier befinden sich alle Source Dateien, um einen neuen Patch bzw. Release zu installieren. Der Patch ist nach der Ebene der Versionsnummern genau so organisiert wie der Source-Baum.
src > apex	Die APEX Applikationsdateien, Exporte der Themes und statischen Dateien z.B. f104_Spots_v2.2.0.0.sql
src > apex_custom	Dateien, die auf dem Applikationsserver ausgeliefert werden. Bei uns im Projekt verwenden wir den virtuellen Pfad /apex_custom/ für die Auslieferung spezieller Javascript- und CSS Dateien. Das /i/ Verzeichnis sollte nicht verwendet werden!
src > lib	Hier befinden sich unterstützende Dateien, die bei jedem Patch benötigt werden, z.B. für die Registrierung der Versionnummer und die Übersicht der installierten Komponenten.

Verzeichnis 60-Betrieb

Verzeichnis	Kommentar
60-Betrieb	Dieses Verzeichnis ist aus der Sicht des Betriebes zu verstehen. Ist der Betrieb ausgelagert an andere

	Dienstleister, werden sie diese Struktur bei sich haben. Hier befinden sich die Skripte für den 3rd Level Support.
60-Betrieb > sql_auswertung	Hier befinden sich die Skripte sowie die Ergebnisse der Anfragen aus dem 3rd Level Support über die Lieferung von Auswertungen. Die benötigten SQL-Skripte werden hinterlegt.
60-Betrieb > sql_patch_data	Alle Konfigurationsänderungen, die mittels DML (insert, update, delete) an der Applikation vorgenommen wurden.

Dateinamen

Der Name jeder Datei im Patch Verzeichnis steht schon vorher fest. In den allermeisten Fällen wird <Objektname>.sql verwendet.

Alle Dateinamen werden in Kleinbuchstaben geschrieben, dies stellt sicher, dass die Skripte (wenn sie unter Windows erstellt wurden) auch direkt unter Linux oder Unix Systemen laufen.

Verwendung	Konvention	Beschreibung	Beispiel
Tabelle	<tabelle>.sql	Tabellendefinition	fm_bookings.sql
	<tabelle>_ref.sql	Referentielle Integration, Foreign Keys	fm_bookings_ref.sql
	<tabelle>_data.sql	DML (Insert, Update, Delete), Konfiguration oder Veränderung von Daten	fm_bookings_data.sql
View	<view>.sql	View-Definition	fm_bookings_v.sql
Prozedur	<prozedur>.sql	Prozedur-Definition	create_booking.sql
Grants	grants.sql	Alle Grants werden zusammengefasst.	grants.sql
Package	<package>.pks	Package Header	fm_booking.pks
	<package>.pkb	Package Body	fm_booking.pkb

Striktes Rollout-Konzept

Wie ist nun der Ablauf einer Änderung im System?

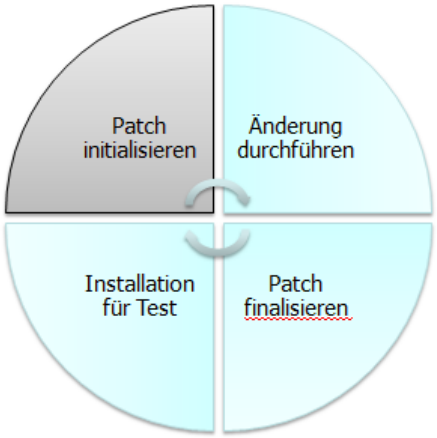

Abb. 3: Rollout-Prozess: Multiple Iterationen für das interne Testen

Für die Zusammenarbeit mit dem Test-Team werden jeweils mehrere Iterationen durchlaufen.

Initialisierung des Patches

Es wird ein neues Verzeichnis für den Patch erstellt, z.B. 3.2.0.0. Wir haben eine Zip-Datei, die die aktuelle Struktur des Patch-Verzeichnisses und deren Basisinhalte bereits enthält. Dieses Template-Verzeichnis verwenden wir dann.

Danach richten wir das patch.sql Skript ein:

```
##
/*============================================================================
  $Id: _patch.sql 701 2009-11-11 11:09:48Z aust.dietmar $
============================================================================*/
set define '^'
set timing off
set pagesize 50000
set linesize 80
set serveroutput on size unlimited
set sqlblanklines on

--#########################
define VERSION=3.2.0.0
--#########################
spool _patch_v^VERSION..log
@@lib/_require_user SHDB_200
@@lib/_patch_start
@@lib/_aq_stop
@@lib/_set_app_offline

------------------------------------------------------------------------------

set define off

--prompt ***
--@@sql/.sql
--@@lib/_pause
```

```
prompt ************************************************************
prompt ** Sequences
prompt ************************************************************

prompt ************************************************************
prompt ** Synonyms (consuming)
prompt ************************************************************

--prompt ***
--@@sql/_synonyms_consuming.sql
--@@lib/_pause

prompt ************************************************************
prompt ** Types
prompt ************************************************************

prompt ************************************************************
prompt ** Tables
prompt ************************************************************

prompt ************************************************************
prompt ** Foreign Keys
prompt ************************************************************

prompt ************************************************************
prompt ** Views
prompt ************************************************************

prompt ************************************************************
prompt ** Procedures
prompt ************************************************************

prompt ************************************************************
prompt ** Functions
prompt ************************************************************

prompt ************************************************************
prompt ** Package Headers
prompt ************************************************************

prompt ************************************************************
prompt ** Package Bodies
prompt ************************************************************

prompt ************************************************************
prompt ** Trigger
prompt ************************************************************

prompt ************************************************************
prompt ** Data
prompt ************************************************************

prompt ************************************************************
prompt ** Scripts, hier können Skripte stehen, die nicht in den
prompt ** sonstigen Rahmen passen, sollten aber mit script_ beginnen
```

```
prompt ************************************************************

prompt ************************************************************
prompt ** Grants (all in file grants.sql)
prompt ************************************************************

--prompt ***
--@@sql/_grants.sql
--@@lib/_pause

prompt ************************************************************
prompt ** Synonyms (providing)
prompt ************************************************************

--prompt ***
--@@sql/_synonyms_providing.sql
--@@lib/_pause

-----------------------------------------------------------------------
-- Hinweise nach der Installation (Post Installation Instruktionen)
-----------------------------------------------------------------------

prompt ************************************************************
prompt **
prompt ************************************************************


-----------------------------------------------------------------------------
--

@@lib/_aq_start
@@lib/_patch_end

set define '^'
host find "ORA-" _patch_v^VERSION..log
host find "SP2-" _patch_v^VERSION..log

host grep "ORA-" _patch_v^VERSION..log
host grep "SP2-" _patch_v^VERSION..log

spool off
exit##
```

<u>Änderungen durchführen</u>

Sobald eine Änderung durchgeführt wird, wird manuell das passende Skript im Patch-Verzeichnis angelegt und dieses im patch.sql registriert. Die leeren Dateien zu erstellen sowie diese auch im Skript zu registrieren, wird für alle Datenbankobjekte so durchgeführt.
Der Inhalt muss jedoch nur für folgende Objekte auch gefüllt werden:

- Änderungen der Tabellen, hier das reine DDL-Statement zur Änderung, z.B. `alter table xxx add column (xxx number);`
- Änderungen an den Daten, also die *_data.sql Skripte

Alle anderen Dateien bleiben leer! Diese werden am Ende des Patches automatisch generiert.

Patch finalisieren

Wurden alle Änderungen wie gewünscht durchgeführt, muss der Patch noch finalisiert werden. Dazu werden erneut die aktuellen Sourcen aus der Datenbank in das Dateisystem extrahiert. Die benötigten Dateien werden dann manuell in das Patch-Verzeichnis kopiert und die existierenden Leer-Dateien überschrieben. Bei Views, Packages, Triggern, etc. können wir gefahrlos die vollständige, aktuelle Definition übernehmen. Bei Tabellen funktioniert das nicht.

Installation in der Testumgebung

Anschließend installieren wir diesen Patch in der Testumgebung. Damit wir diese Installation mehrfach laufen lassen können, verwenden wir die Flashback-Funktionalität von Oracle. Z.B. setzen wir zuerst einen Restore Point, bevor wir den Patch installieren:

```
create restore point BEFORE_REL_2_3_0_0;
```

Fazit

Das beschriebene Verfahren wurde in den letzten drei Jahren in einem tatsächlichen Kundenprojekt entwickelt und hat sich bereits mehrfach bewährt. Wir haben es kontinuierlich verbessert und dadurch unsere Qualität erheblich steigern können. Alle Dinge im Projekt haben jetzt eine Ordnung und eine feste Struktur. Dadurch entsteht Klarheit, Transparenz und Sicherheit im Umgang für alle Beteiligten.

Kontaktadresse:

Dietmar Aust
Zum Tilmeshof 11
D-50859 Köln

Telefon:	+49(0)173-5322 955
Fax:	+49(0)221-17099759
E-Mail	dietmar.aust@opal-consulting.de
Internet:	http://www.opal-consulting.de
BLOG:	http://daust.blogspot.com
APEX-Tools:	http://www.opal-consulting.de/tools (zur freien Nutzung und kostenlos)
APEX-Buch:	http://apex-xe-praxis.de/