Synchronisierte “Multi-Stream” Entwicklung mit Siebel

Referent: Raoul Mayr, Co-Referent: Oliver Seiffert
IBM Global Business Services
Hannover

Schlüsselworte:

Oracle Business-Software, Siebel CRM, Release, Branching, Softwareeinführung, Praxiserfahrungen, Parallele Entwicklung

Einleitung

Siebel bietet in seinem Tool-Set für die parallele Entwicklung von Releases, d.h. Konfiguration via *Siebel Tools*, Scripting und weiteren Nicht-Repository Artefakten wie z.B. LOVs oder Systempreferenzen, nur sehr begrenzte und für komplexe Entwicklungen keine Möglichkeiten an. Es ist daher in Projekten ab einer gewissen Komplexität zwingend notwendig organisatorische Rahmenbedingungen und Prozesse zu schaffen, um eine stabile parallele Entwicklung zu ermöglichen.

Bevor wir in die detaillierten organisatorische Rahmenbedingungen und Prozesse einsteigen, lassen sie uns zunächst ein gemeinsames Verständnis zum *Release Management* und *Branching* schaffen.

Das *Release Management* ist ein Prozess, welches die Bündelung von Software-Änderungen, welche zu einem spezifischen Release gehören, zu einem Paket und dessen ordnungsgemäße Installierbarkeit im Infrastruktur-Team sicherstellt. Das Release Management beinhaltet u.a.:

- die Aufgaben der Festlegung des funktionalen Umfangs,
- die Festlegung des Zeitplans und der Freigabe des Releases,
- die Dokumentation der Änderungen und die Versionierung

Das *Branching* erfolgt, wenn mehr als ein Release parallel entwickelt und/oder unterstützt werden soll. Das ist z.B. dann der Fall, wenn sich ein Release 1 in der Produktion befindet, parallel aber bereits an der Weiterentwicklung an Release 1.1 gearbeitet wird. In diesem Falle müssen 2 unabhängige Entwicklungsstränge existieren. Zum Beginn von Release 1.1, wird daher ein *Branch* für das Release 1.1 geschaffen, welche alle Applikationsartefakte vorhält.

Die Planung der Releases hat als *Zwangsplanungkomponente* die Anforderung, dass die Parallelität der Releases eine zeitversetzte Parallelität darstellt. Während sich ein Release X im Produktivbetrieb befindet, ist das Release X+1 im Systemtest, und bspw. das Release X+2 in der Entwicklung. Das Release X+3 befindet sich schon in der Designphase gemäss dem Wasserfallentwicklungsmodell. Eine komplette Parallelität der Entwicklungsstränge, ist auf Basis der Eingangs beschriebenen Komplexität der Siebel Applikation nicht möglich. *Siebel* bietet hierfür im Standard nur eine äußerst eingeschränkte Funktionalität in Form von *SIF*

(Siebel Import File) - und *Patch*-Dateien an. Aus unserer Erfahrung, verkompliziert der Einsatz dieser Mechanismen in vielen Fällen den Prozess der Parallelentwicklung, da diese beim Transport komplexer Änderungen zwischen Releases fehleranfällig sind.

Organisation

Die Entwicklung mehrerer Releases parallel läßt sich am effektivsten Umsetzen, wenn jedes Release durch ein eigenes Projektteam betreut werden kann. Als vorteilhaft hat sich die folgenden Kombination von Ressourcen in der Praxis herausgestellt: Projektleiter, Siebel Architekt, Technical Development Lead und eine entsprechende Anzahl von Siebel Business Analysten und Siebel Entwicklern. Verantwortlich für das Release (*Lead*) sind Projektleiter, Siebel Architekt und Technical Development Lead. Nur diese Personen haben im allgemeinen den vollumfänglichen Überblick über geplante fachliche - und technische Änderungen und können (*müssen*) korrigierend während der Design- und Entwicklungsphase eingreifen. Um eine zielgerichtete Kommunikation im Release sicherstellen zu können, sollten diese Personen primär die Außenkommunikation zu anderen Releases und dem Kunden übernehmen.

Die Ressourcen eines Releases können prinzipiell auch rollierend zwischen den geplanten Releases eingesetzt werden. Dies ist oftmals sogar notwendig, da keine Projekte über einen unendlichen Personalpool verfügen. *Vorsicht* – Je mehr gerade die Personen im Lead in anderen Releases verplant werden, um so deutlicher sind die zu erwartenden Qualitätseinschnitte im Hauptrelease.

Technische Voraussetzungen der Infrastruktur

Soll in einem Projekt mindestens 1 Release parallel entwickelt werden, sind pro Release eigene Umgebungen (*pro zu entwickelnden Release*) von der Infrastruktur zur Verfügung zu Stellen. Als sinnvoll haben sich dabei die folgenden Umgebungen herausgestellt:

- Entwicklungsumgebung,
- Build-Umgebung zur Verwaltung der Nicht-Repository Artefakte,
- Systemtestumgebung.
- Optional - u.U. eine Integrationstestumgebung für Schnittstellen

Die Anzahl dieser Umgebungen ist abhängig von der Anzahl der geplanten parallelen Releases. Möchte man also 2 Releases parallel entwickeln, so sind obigem Beispiel folgend 8 virtuelle und physikalische Umgebungen notwendig. In Projekten, wo die Applikation große Datenmengen zu verwalten hat, ist ggf. auch eine Performance-Test-Umgebung erforderlich. Da diese Umgebung im Aufbau in der Regel an das Produktivsystem angelehnt ist, sollte durch organisatorische Massnahmen sichergestellt werden, dass diese von allen Releases für Performance-Tests genutzt werden kann. Eine Verdreifachung dieser Umgebung ist möglich, allerdings ist dies mit erheblichen Budget verbunden, und meistens auch nicht erforderlich aufgrund der unterschiedlichen Startzeitpunkte der Releases.

Wichtig ist die *Einheitlichkeit* aller Umgebungen pro Release. Datenbank- und Betriebssystemversionen müssen identisch sein, um jegliche Inkompatibilitätsprobleme

auszuschliessen. *Keine Regel ohne Ausnahme*. In diesem Falle ist die Ausnahme die technische Notwendigkeit für ein neues Release auf eine andere Datenbank- und/oder Betriebssystemversion zu wechseln. Hier kommt es zwangsläufig zu temporär begrenzten Versionsunterschieden. Kommt es dennoch zu Unterschieden, z.B. in der Betriebssystemversion der Produktionsumgebung zur Systemtestumgebung, ergeben sich hieraus oftmals aufwendig und teilweise fast nicht zu analysierende Fehlerbilder, bspw. lassen sich bestimmte „Core Dumps" der Applikation nur unter UNIX-Systemen aber nicht unter Windows reproduzieren.

Eine oftmals unterschätze Fragestellung, ist die Sicherstellung, dass Entwickler immer am „*richtigen*" Release arbeiten. Ein unachtsam geöffnetes Siebel Tools und der Systemtest steht eine Woche später für 3 Tage. Genau das kann verhindert werden, wenn entweder eindeutige Benennungen der *Siebel Tools* Umgebungsauswahlen erfolgen, oder dediziert, über eindeutige Entwicklerkennungen pro Release, die Eindeutigkeit erzwungen wird. Auch ist über die Entwicklung über *Windows Terminal Server* oder virtuellen PCs, die pro Release zur Verfügung gestellt werden, denkbar. Gerade hier haben sich z.B. Signalfarben (Rot, Grün, Gelb) als Hintergrundbild bewährt.

Mit der Anzahl der Umgebungen steigt unweigerlich der Aufwand des *Deployments*. Diese Fragestellung lässt sich nur adressieren, wenn entweder das Infrastruktur Team mehr Ressourcen erhält oder das *Deployment* automatisiert werden kann. Hier kann der *Siebel*-eigene *Application Deployment Manager* (*ADM*) unterstützen, der allerdings in Bezug auf einige Artefakte Beschränkungen unterliegt. Ebenfalls werden im Siebel Standard nicht alle Artefakte unterstützt. Dies muss in einigen Fällen in Siebel Tools nachkonfiguriert werden. Eine voll automatisierbare Alternative, die viele Artefakte von Haus aus unterstützt, ist hier *IBM's Automated Deployment Process* (*ADP*).

Der i-Punkt nach einem erfolgreichen Deployment, ist der sogenannte Sanity Test. Dieser stellt sicher, dass ein erfolgreiches Deployment auch in einer lauffähigen Applikation geendet hat. Ziel ist dabei die grundlegenden Funktionen der jeweiligen Applikation zu testen. Auch hier ist eine Automatisierung sinnvoll. Unterstützt wird dies durch entsprechende Testtools, wie HP Mercury Quicktest Pro oder IBM's Rational Functional Tester.

Prozesse und Gruppen

Process Board

Das *Process Board* dient dazu release-übergreifende Prozess basierende Themen zu besprechen, wie z.B. die nächsten Schritte in den Releases, die zu Abhängigkeiten bei anderen Releases führen.

Im *Board* selbst sollten regelmäßig die Projektleiter und Architekten aller Releases vertreten sein. Nur diese Konstallation ermöglicht die Entscheidungsfähigkeit.

Ein weiterer Punkt für das *Process Board*, ist die einheitliche Festlegung von Vorgehensweisen bei der Entwicklung der Releases, bspw. *Object* vs. *Project Locking*, das

release-übergreifend identisch behandelt werden sollte, um einheitliche Entwicklungsprozesse sicherzustellen.

Change Control Board

Das *Change Control Board* (CCB) besteht aus technischen Vertretern aller Releases **und** Teams (inkludiert Vertreter des Kunden), die die Funktionalität ihres zu betreuenden Releases kennen. Im CCB selbst werden alle Datenmodelländerungen und auch alle Änderungen an der bestehenden Geschäftslogik besprochen und zwischen den Teilteams und den Release-Verantwortlichen abgestimmt.

Warum sollte das durchgeführt werden ?

Beispiel – Es kann sich als durchaus schwerwiegender Fehler für ein Release erweisen, wenn eine neue Spalte für ein Folgerelease eingeführt, aber dabei vergessen wird für entsprechende Initialwerte zu sorgen. Setzen Suchkriterien aus dem momentanen Release bereits auf dieser Spalte auf, können sich gravierende Performanceprobleme ergeben. Ein weiteres Beispiel ist auch die Abstimmung innerhalb eines Releases, wenn schnittstellenseitig eine *Business Component* geändert wird, aber das eigentlich verantwortliche Team für diese *Business Component* diese Änderung nicht mitbekommt.

Qualitätssicherung „Vertrauen ist gut, Kontrolle ist besser"

Die Qualitätssicherung ist ein wesentlicher Beitrag für den Erfolg bei der parallelen Entwicklung von *Siebel* Releases. Nur wie kann man diese sicherstellen ?

Eine ausreichende Qualität bei der parallelen Entwicklung ist nur zu erreichen, indem restriktive QA-Prozesse aufgesetzt werden Es ist zwingend erforderlich regelmäßige stichprobenartige Qualitätsanalysen, wenn möglich auf Ticketbasis, zusammen mit dem oder den Entwicklern, durchführen. Dies sichert zum einem die Prüfung der Qualität, hilft aber auch dem jeweiligen Entwickler in der Verbesserung seiner Qualität. Ebenfalls können die Ergebnisse solcher Qualitätsanalysen konsolidiert in einer Schulungsmassnahme münden.

Eine weitere Maßnahme ist die regelmäßige Prüfung der *Configuration* und *Scripting* Qualität der Applikation, die man zum Beispiel mit *Oracle's Siebel Configuration and Scripting Reviews* durchführen kann. Dieser Prozess kann aber auch automatisiert werden durch den Einsatz eines eigenen Tools, wie z. B. *IBM's DAVID for Siebel*.

Der spätere Erfolg des Releases im Produktivbetrieb, ist auch abhängig von der Performance der Anwendung. Um dies sicherstellen zu können, ist eine stete Kontrolle der entwickelten Funktionen erforderlich. Hierbei unterstützen den Entwickler *Objectmanager-Logdateien* (und hier die inkludierten SQLs) und seit *Siebel* Version 7.8 auch die *Siebel SARM Dateien*. SARM Dateien erlauben eine detaillierte Analyse der *Application Response* von *Views*, *Applets*, *Business Services* und *Workflows*. Bei der Auswertung der O*bjectmanager-Logdateien* kann mittels *IBM's SOMSA Tool* eine Top Ten der *SQL* Langläufer mit *Event Context* erstellt werden. *IBM's iSARM Tool* bietet darüberhinaus die Möglichkeit, die

kryptischen *SARM* Dateien aufzubereiten, um einfacher auf mögliche Performance Engpässe aufmerksam zu werden.

Kontaktadresse:

Raoul Mayr
IBM Deutschland GmbH
Global Business Services
Laatzener Str. 1
30539 Hannover

Telefon: +49(0)160 9781 7092
Fax: +49(0)69 6645 5641
Mail raoul.mayr@de.ibm.com
Internet: www.ibm.com/de