Risiken erkennen und beherrschen. Erfolgreich durch Krisen steuern. Das ist das Kerngeschäft der Unternehmensberatung S/E/ Strategie und Ergebnisse, Mittelstandsberatung GmbH in Düsseldorf. Dieser Artikel zeigt, wie ein Satz aus Apex-Applikationen die damit verbundenen Aufgaben und Maßnahmen effektiv unterstützt.

Krisensicher

Jens Gauger, IGL GmbH

Wie in vielen Apex-Projekten begann auch dieses Projekt mit einem Prototyp im Jahr 2009, der eine isolierte Anforderung behandeln sollte. Bereits Jahre davor begann IGL mit der Entwicklung von Applikationen auf Basis des damals noch „HTML DB" genannten Frameworks. In dieser Zeit wurden viele bisher in Office-Anwendungen oder per Papier durchgeführte Prozesse nach Apex überführt. So entstand mit der Zeit ein ganzer Baukasten aus Anwendungen für verschiedenste Aufgaben. Häufig behandelten die Applikationen allgemein verwendbare Geschäftsvorfälle, beispielsweise als Angebotsverwaltung oder Vertriebssteuerung. Daher bestand schon immer die Intention, diese Applikationen wiederholt mit leichten Anpassungen in neuen Projekten zum Einsatz zu bringen.

Bei einem Treffen mit S/E/ Anfang 2008 wurden einige dieser Module vorgestellt. S/E/ war von Anfang an begeistert von der Idee ein Werkzeug zu bekommen, um Informationen, die während eines Beratungsprojekt anfallen, an einem zentralen Ort strukturiert speichern zu können. Die Möglichkeit, den beteiligten Personen einen geregelten Zugriff auf aufbereitete Informationen zu geben und sie somit stärker in das Beratungsprojekt einbeziehen zu können, erschien reizvoll. Speziell das Killer-Feature „Interactive Reports" erregte große Aufmerksamkeit, weil damit eine Brücke zwischen strukturierten und reglementierten Daten einerseits und individuellen Auswertungsbedürfnissen andererseits geschlagen wird.

Der Prototyp

Neben einigen vorhandenen Applikationen sollte auch eine vollständig neue Anwendung für die Verwaltung von Maßnahmen entstehen. In einem Beratungsprojekt erfolgt nach einer Ist-Analyse üblicherweise der eigentliche Projektstart mit einem Workshop, bei dem in einer größeren Runde die Ergebnisse der Ist-Analyse aufbereitet und gemeinsam mit Geschäftsführung und Mitarbeitern des Unternehmens Gegenmaßnahmen erarbeitet werden. Das Haupt-Augenmerk liegt darauf, dass diese Maßnahmen nachhaltig abgearbeitet werden und der Bearbeitungsfortschritt in einer Historie nachvollziehbar ist. Die dafür konzipierte Anwendung nimmt diese Maßnahmen auf, versieht sie mit Zuständigkeiten und verwaltet Meilensteine und Fertigstellungstermine. Über Ampelfunktionen lässt sich die Einhaltung von Meilensteinen überwachen. Durch diese Vorgehensweise wird im Projekt eine wesentlich höhere Verbindlichkeit bei der Bearbeitung von Maßnahmen erreicht. In regelmäßigen Projekt-Meetings werden die fälligen Maßnahmen besprochen und per Statuswechsel aktualisiert. Die allgemein bekannte Situation, dass die in einem Kick-off-Meeting erarbeiteten Aufgaben nur unzureichend dokumentiert sind und die beteiligten Personen sich in den Folge-Meetings nicht mehr an Aufgaben erinnern können oder sich gegenseitig die Verantwortung zuschieben, wird somit vermieden.

Die mit Apex realisierte Maßnahmensteuerung (siehe Abbildung 1) entwickelte sich in den darauf folgenden Jahren zur Kern-Anwendung der gesamten Applikation. Erweiterte Funktionen wie E-Mail-Benachrichtigung, PDF-Druckerzeugung und Charts kamen mit der Zeit hinzu.

Es geht voran ...

Nachdem die erste Version des Projekt-Cockpits in einigen Projekten erfolgreich zum Einsatz kam, entstand Bedarf an weiteren Modulen. In Anlehnung an Grundprinzipien des Krisenmanagements entstanden thematisch strukturiert folgende Module (siehe Abbildung 2):

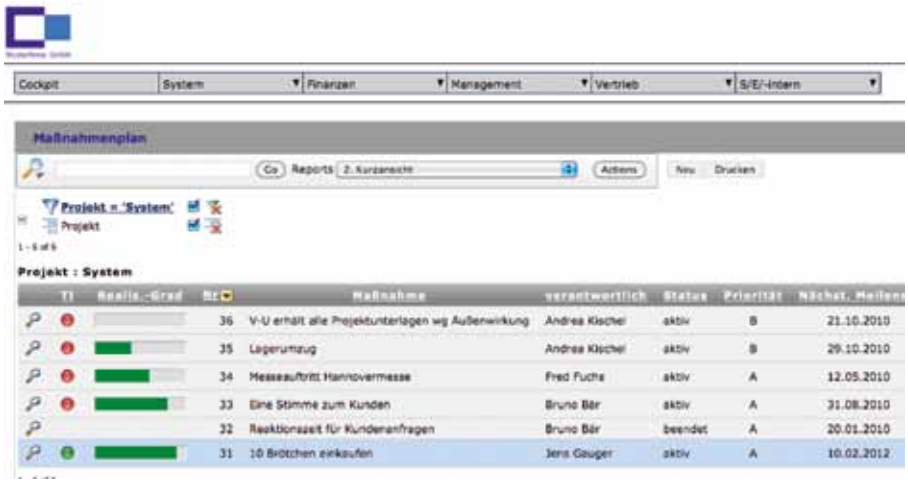

Abbildung 1: Screenshot Maßnahmensteuerung

Abbildung 2: Grundprinzipien des Risiko- und Krisenmanagements

- Finanzen (Forderungs- und Verbindlichkeitenmanagement, Liquiditätssteuerung und Avalmanagement)
- Management (Dokumente, Maßnahmen, Vorgangssteuerung, Audit & Revision)
- Vertrieb (Angebotsverwaltung, Vertriebssteuerung, Vorkalkulation)

Eine Kernanforderung war, dass projektabhängig eine individuelle Zusammenstellung der Module möglich sein sollte, um eine höchstmögliche Bedarfsanpassung zu erreichen. Aus entwicklungstechnischer Sicht bot sich an, nicht eine große Apex-Anwendung zu erstellen, sondern die Funktionsbereiche auch technisch in einzelne Anwendungen aufzuteilen. Ein typisches Problem in diesem Zusammenhang stellen die Stammdatenpflege, die Authentifizierung und die Berechtigungssteuerung dar. Dies sollte idealerweise zentral konfiguriert werden können. Authentifizierte Sessions sollen zwischen den Anwendungen kommuniziert werden – quasi ein applikationsinternes Single-Sign-on.

Um das Problem zu lösen, wurde eine zusätzliche Basis-Applikation entwickelt, die genau diesem Zweck dient. Hier wurde alles hineingepackt, was im weitesten Sinne Stammdaten sind. Auch die Benutzerverwaltung und Berechtigungssteuerung ist hier angesiedelt. Jede Fach-Applikation enthält wiederum Authentifizierungs- und Autorisierungsschemata, die auf diese Daten zurückgreifen. Zusätzlich wurden noch spezielle Funktionen wie internes Messaging, Wiedervorlage und konfigurierbare Feldvorbelegung integriert.

In der Nachbetrachtung hat sich das Konzept des modularen Aufbaus jedoch nur in Teilbereichen als erfolgreich erwiesen. Heute ist es so, dass grundsätzlich alle Module ausgeliefert werden, da zu Projektbeginn nicht genau festgelegt ist, welche Module schlussendlich zum Einsatz kommen. Die Entscheidung darüber hängt auch vom Wunsch des Endkunden ab, in welcher Ausprägung er das Projekt-Cockpit auf Dauer nutzen möchte.

Ein positiver Aspekt des modularen Aufbaus liegt aber in der individuellen Anpassbarkeit. Es ist möglich, einzelne Module an Projektbedarf oder Kundenwunsch anzupassen, ohne die ganze Anwendung an sich berücksichtigen zu müssen. Dies passiert häufig im Bereich Schnittstellen. Projektcockpit enthält im Finanzbereich Module zum Import von OPOS-Daten aus verschiedenen Quellen. Hier besteht beinahe bei jeder neuen Installation Anpassungsbedarf.

Außenbeziehungen

Grundsätzlich haben sich der Informationsbezug aus unterschiedlichen Quellen und die Informationsbereitstellung in Form von pixelgenauen Reports als wesentliche Betätigungsfelder bei der weiteren Entwicklung des Projekt-Cockpits erwiesen. Während bei der Erstellung von pixelgenauen Reports auf die Lösungen von Jaspersoft zurückgegriffen werden konnte und die Integration in der Apex-Umgebung mithilfe der hervorragenden „JasperReportsIntegration“ von Dietmar Aust mühelos gelang, war bei der Übernahme von Daten aus fremden Quellen weitaus mehr Entwicklungsaufwand zu betreiben. Typischerweise besteht immer der Bedarf für die Übernahme von OPOS-Daten, also „Offene-Posten“-Listen für Debitoren und Kreditoren aus der Finanzbuchhaltung.

Aber auch komplexere Quellen wie Leistungsverzeichnisse im GAEB-Format waren zu verarbeiten. Der ursprüngliche Ansatz bestand in der Implementierung über Perl. In der zuständigen Apex-Anwendung wurde ein Upload-Dialog angeboten, der die übernommene Datei per „UTL_FILE“ ins Dateisystem schob. Dort fungierte das unter Linux verfügbare „inotify“ verbunden mit dem Add-on „incron“ als Trigger für die weitere Verarbeitung. Dieser Ansatz hatte allerdings einige Nachteile. Zum einen war „incron“ teilweise instabil, zum anderen konnten Fehler bei der Verarbeitung, etwa aufgrund von Formatabweichungen, nicht in einer für den Anwender transparenten Art und Weise behandelt werden, da Fehler, die auf der Shell auftraten, nicht in der Apex-Anwendung durchgereicht wurden.

Nicht zuletzt spielte beim Anwender immer wieder das Thema „nativer Excel-Import“ eine große Rolle. Es existierte zwar eine rudimentäre Excel-Unterstützung über das Linux-Tool „xls2csv“, leider aber nur für OLE32-Dateien. Im neuen Format erhaltene Excel-Dateien mussten immer manuell nach CSV gespeichert werden, um sie verarbeiten zu können. Neben dem umständlichen Handling gab es hier auch immer wieder Probleme mit Textfeldern, die Sonderzeichen oder Zeilenschaltungen enthielten. Auch Formatvorgaben bei numerischen Werten und Datumsgrößen führten häufig zu Schwierigkeiten.

Die Alternative wäre eine feste Anbindung per definierten Schnittstellen-Format. Zu den wenigsten Datenquellen können aber direkte Verbindungen hergestellt werden, da sich dafür entweder keine Möglichkeit auf

Seiten der Datenquelle anbietet oder der Aufwand für die Implementierung einer solchen Schnittstellen-Übertragung in keinem adäquaten Verhältnis zum Nutzen steht. Einen Datenexport nach Excel beherrschen aber die meisten Systeme aus dem Stand. Auf Dauer musste also eine Lösung her, die es dem Anwender ermöglichen sollte, Excel-Dateien gleich welcher Version im Ursprungsformat importieren und auf Abweichungen im Verarbeitungsfortschritt direkt im Dialog reagieren zu können.

Immer wieder dieses Excel ...

Beim Import von nativen Excel-Dateien stößt man auf viele Probleme. Beispielsweise ist der Wechsel bei Microsoft vom OLE2-Format (Endung „.xls“ bis Excel 2003) über das XML-Format und schließlich zum Office-Open-XML-Standard (Endungen „.xlsx“, „.xlsm“ etc. ab Excel 2003/2007) eine echte Herausforderung in der Praxis. Beide Formate kursieren nach wie vor und es kann nicht von einem vorher festgelegten Versionsstand ausgegangen werden. Auch kann es nicht dem Anwender überlassen werden zu ermitteln, in welcher Version eine Excel-Datei gerade vorliegt. Oftmals fehlt dafür das technische Verständnis oder die Erfahrung. Diese Entscheidung muss also der Import-Prozess fällen.

Nach anfänglichen Experimenten mit der im Apex Listener enthaltenen Funktionalität „apex.excel2collection“, die aufgrund der Restriktion auf maximal fünfzig Spalten und die Bindung an das OLE2-Format schnell wieder verworfen wurde, entschied man sich für die Entwicklung einer eigenen Lösung. Die Basis dafür bildet das schon im Apex Listener als Grundlage dienende Projekt „Apache POI“. Mit dieser Java API ist es möglich, Office-Dokumente zu erzeugen oder zu lesen, ohne dafür Microsoft Office als Frontend verwenden zu müssen. Für diese Zwecke kamen die Component-APIs für Excel „HSSF“ (Excel 97-2007) und „XSSF“ (.xlsx) in Betracht.

Die Implementierung erfolgte als J2EE-Servlet auf einem Apache Tomcat und die Integration mit Apex über eine PL/SQL-Prozedur, die einen HTTP-Request erzeugt und an den Server sendet. Das Servlet ermittelt selbstständig, welches Excel-Format vorliegt, und spricht die korrekte API an. Die erzeugten Daten werden in zwei Tabellen direkt in die Datenbank geschrieben, wobei auch die strukturelle Unterteilung einer Excel-Datei in einzelne Blätter berücksichtigt wird.

In Apex wiederum wurde eine Anwendung erstellt, mit der auf diese Tabellen zugegriffen werden kann. Um die Daten entsprechend dem späteren Verwendungszweck validieren zu können und dem Anwender passende Werkzeuge zur Behandlung von negativen Validierungen an die Hand zu geben, wurde ein „Schablonen“-Konzept erdacht. Damit ist es möglich, für Datenblöcke (Zeilen von x bis y in einem Excel-Arbeitsblatt) Format-Definitionen für Spalten festzulegen. So lassen sich beispielsweise Spalten mit Datums- oder numerischen Werten schnell auf korrektes Format und Vollständigkeit (per not-null-Schalter) prüfen. Sollten Abweichungen auftreten, kann der Anwender die Daten direkt im Dialog anpassen. Ein Wechsel nach Excel und ein erneuter Import sind nicht notwendig. Die abschließende Bearbeitung nach der Validierung ist dann wiederum spezifisch an den jeweiligen Zweck angepasst.

Dieser „universeller Daten-Import“ (UDI) genannte Prozess hat die produktive Arbeit mit dem Gesamtsystem nochmals erheblich verbessert und zu einer höheren Anwenderzufriedenheit und damit zu einer Verringerung der Support-Calls geführt.

Architektur und Sicherheit

Die Architektur hinter der Anwendung sieht neben einer Oracle-XE-Datenbank und (momentan noch) Apex 4.0 einen Tomcat-Server für die Services JasperReportsIntegration, UDI und Apex Listener vor. Vor den Listener ist zusätzlich ein Apache HTTP-Server geschaltet, der als Reverse Proxy fungiert und auch ein URL-Rewriting durchführt. Die Verbindung zwischen Client und Server wird per „https“ verschlüsselt. Ein direkter Zugriff auf die Tomcat-Services durch den Client ist nicht vorgesehen. Sämtliche Anfragen des Client, die die Tomcat-Services betreffen, werden durch Apex getunnelt. Die Anfragen an die Tomcat-Services werden zusätzlich über Tokens validiert, um einen Missbrauch auszuschließen. Innerhalb der Apex-Applikation sorgt ein gestaffeltes Rollenkonzept ergänzt um zusätzliche Benutzerattribute für einen geregelten Zugriff.

Fazit

Seit Inbetriebnahme der Version 1.0 im November 2009 hat sich die Anwendung zu einem inzwischen nur schwer wieder wegzudenkenden Steuerungs-Instrument in den Beratungsprojekten der Unternehmensberatung S/E/ entwickelt.

Dies zeigt einmal mehr, wie sehr sich Apex als Werkzeug für alle Ausprägungen vom Prototyping bis zur fertigen Applikation eignet. Zu Beginn eines Entwicklungsprojekts sind oftmals die Anforderungen an die Applikation noch nicht klar definiert, auch aufgrund der Unkenntnis des Leistungsvermögens von Apex.

Die Möglichkeit, mit vergleichsweise geringem Aufwand einen Prototyp schaffen zu können, der die Projektanforderungen illustriert und als Diskussionsgrundlage für die weitere Entwicklung dient, ist nicht hoch genug einzuschätzen. Apex in der aktuellen Version ist inzwischen auch weit davon entfernt, lediglich ein Tool zur Ablösung von Excel-/Access-Anwendungen zu sein.

Jens Gauger
jens.gauger@igl-systems.de