JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Mainzer Informatik-Berichte

Uwe Kaiser
Petr Kroha
Andreas Winter (Hrsg.)

3. Workshop Reengineering Prozesse (RePro 2006)

Software Migration

November 2006

Informatik-Bericht Nr. 2 / 2006

Institut für Informatik · Fachbereich 08

Johannes Gutenberg-Universität Mainz
Institut für Informatik – FB 8
D 55099 Mainz
ISSN 0931-9972

Vorwort

Die Workshop-Reihe Reengineering Prozesse (RePro) ist aus dem Workshop Software Reengineering der gleichnamigen GI Fachgruppe hervorgegangen, und befasst sich mit Methoden und Vorgehensmodellen zur Durchführung von Software Reengineering Projekten. Wie auch der vorangegangene Workshop stellt die dritte Auflage von RePro die *Software-Migration* ins Zentrum.

Informations- und Kommunikationstechnologien entwickeln sich ständig weiter. Im Gegensatz dazu arbeiten viele Unternehmen mit veralteter Software. Der Grund ist, dass diese Systeme das gesamte, historisch gewachsene Know-how der Unternehmen verkörpern und durch jahrzehntelange Wartung auch stabil arbeiten. Auf der anderen Seite stehen diesen Vorteilen gravierende Nachteile wie hohe Wartungs- und Lizenzkosten, geringe Anpassungsflexibilität an moderne Technologien und rückläufiges Programmierwissen gegenüber. Die Softwaremigration bietet Möglichkeiten, dieses Defizit zu überwinden und etablierte Softwaresysteme in modernen und flexiblen Umgebungen bereitzustellen. Softwaremigration beinhaltet neben der (teil)automatischen Konvertierung von Programmen aus antiquierten Programmiersprachen wie z. B. COBOL oder PL/I in moderne(re) Sprachen wie C++ und Java unter anderem auch die Integration in neue Betriebsysteme, die Modernisierung von Datenhaltung und Benutzeroberfläche und Übertragung in modernere Softwarearchitekturen.

Ziel des Workshops ist der Erfahrungsaustausch zu konkreten Migrations-Projekten. Hierzu wurden Referenten aus Systemhäusern und Entwicklungs- und Wartungsabteilungen eingeladen, um über ihre Projekte zu berichten. Die Tagungsbeiträge beinhalten Berichte und Erfahrungen aus Migrationsprojekten verschiedener Branchen (Automobilbau, Telekommunikation, Tourismus, Banken und Versicherungen) sowie Darstellungen aktueller Forschungsvorhaben aus dem universitären Umfeld. Außerdem werden auch wieder Software-Werkzeuge zur Unterstützung der Migration demonstriert.

Den Referenten sei an dieser Stelle herzlich dafür gedankt, dass sie sich bereit erklärt haben, über ihre Erfahrungen zu berichten und ihre Überlegungen zur Diskussion zu stellen. Auch wenn Migrations-Projekte und andere Reengineering-Aktivitäten heute alltägliche Tätigkeiten in der Software-Entwicklung sind, ist es jedoch noch nicht üblich, hierüber offen zu berichten. Vielfach werden Reengineering-Projekte fälschlicherweise als “Reparaturmaßnahmen an fehlerhafter Software” betrachtet. Die Beiträge des Workshops zeigen jedoch deutlich, dass Migrationsprojekte die Werterhaltung vorhandener Softwaresysteme sicherstellen und ihre Wertsteigerung erst ermöglichen.

Der dritte *Workshop Reengineering Prozesse*, der von der Firma pro et con (`http://www.proetcon.de/`), Chemnitz, der Fakultät für Informatik der TU Chemnitz (`http://www.tu-chemnitz.de/informatik/`) und der Fachgruppe Software-Reengineering der Gesellschaft für Informatik (`http://www.uni-koblenz.de/sre/`) organisiert wird, fand am 23./24. November 2006 im Hotel “Chemnitzer Hof” in Chemnitz statt. Die lokale Organisation übernahm Herr Dr. Wolfgang Bormann mit seinem Team, dem an dieser Stelle herzlich für die perfekte Vorbereitung und Durchführung des Workshops gedankt sei.

November 2006

Uwe Kaiser
pro et con Innovative Informatikanwendungen GmbH

Petr Kroha
Technische Universität Chemnitz
Fakultät für Informatik

Andreas Winter
Johannes Gutenberg-Universität Mainz
Institut für Informatik

Programm

Donnerstag, 23. November 2006

12.30:13.15		Anmeldung und Lunch
13.15:13.30	**U. Kaiser** (pro et con GmbH), **P. Kroha** (TU Chemnitz). **A. Winter** (GI, FG Reengineering)	Begrüßung

Software Migration und Werkzeuge

13.30:14.00	**P. Kroha, L. Rosenhainer** (TU Chemnitz)	*Textuelle Anforderungen und Software Migration*
14.00:14.30	**D. Uhlig** (pro et con GmbH)	*IDE für eine BS2000-Migration auf Eclipse-Basis*
14.30:15.00	**J. Bach, M. Doppler, M. Schulze** (Debeka Hauptverwaltung)	*Repositorygestützte Erkennung von Schnittstellen in einem hochintegrierten Anwendungssystem*
15.00:15.30		Vorstellung der ausgestellten Migrations-Tools
15.30:16.15		Tooldemonstration und Kaffeepause

Software Migration in der Praxis

16.15:16.45	**W. Teppe** (Amadeus Germany GmbH)	*ARNO: Migration von Mainframe Transaktionssystemen nach UNIX*
16.45:17.15	**J. Hahn** (T-Systems GEI GmbH)	*Software Migration einer Mainframe-Anwendung — ein Praxisbericht*
17.15:17.45	**T. Schuller** (MAN Nutzfahrzeuge AG/TDB)	*HIT: Harmonisierung und Integration von Anwendungssoftware bei MAN/TDB*
18.45		Abendveranstaltung Villa Esche

Freitag, 24. November 2006

Software Migration an der Schnittstelle Forschung/Praxis

09.00:09.30	**A. Winter** (Johannes Gutenberg-Universität Mainz), **J. Ziemann** (Iwi Saarbrücken)	*Modellbasierte Migration nach SOA*
09.30:10.00	**R. Gimnich** (IBM Software Group, Enterprise Integration)	*SOA-Migration - Ansätze und Projekterfahrungen*
10.00:10.30	**G. Rünger, M. Kühnemann** (TU Chemnitz/Informatik)	*Modellgetriebene Transformation von Legacy Business-Software*
10.30:11.00		Tooldemonstration und Kaffeepause

Software Migration in der Praxis

11.00:11.30	**G. Salvi** (HAL Knowledge Solutions AG)	*Migration der UBS Kern-Applikationen von UNISYS nach z/OS — Best Practice*
11.30:12.00	**J. Borchers** (Steria Mummert Consulting AG)	*Kritische Erfolgsfaktoren beim Abnahmetest in Redevelopment-Projekten — Erfahrungen aus einem Großprojekt*
12.00:12.30	**J. Boos, M. Voß, J. Willkomm, A. Zamperoni** (sd & m AG)	*Lösungsmuster in der Planung industrieller Migrationsprojekte*
12.30:12.45	**U. Kaiser** (pro et con GmbH), **P. Kroha** (TU Chemnitz), **A. Winter** (GI, FG Reengineering)	Schlußworte
12.45:13.30		Tooldemonstration und Lunch

Textuelle Anforderungen und Software-Migration

Petr Kroha, Lars Rosenhainer
Fakultät für Informatik, TU Chemnitz, 09107 Chemnitz
{kroha, lro}@informatik.tu-chemnitz.de

Zusammenfassung

Künftige Software-Migrationsprobleme können gemildert werden, wenn bei der heutigen Entwicklung komplexer Softwaresysteme der Einhaltung von Konsistenz zwischen natürlichsprachlicher Anforderungsbeschreibung und des davon abgeleiteten objektorientierten Analysemodells stärkere Beachtung geschenkt wird. Wird dies getan, können die einem zu migrierenden System zugrundeliegenden, ursprünglichen Anforderungen und deren Bezug zum System identifiziert (Stichworte: Traceability, Reverse Engineering) werden, was zu einem besseren Systemverständnis führt und damit wertvolle Hilfe für den Migrationsprozess bietet.

1 Einführung

Die Probleme der Software-Migration lassen sich in mehrere Gruppen unterteilen. Eine von diesen Gruppen bilden Verständnisprobleme, die durch lückenhafte, inkonsistente oder sogar fehlende Dokumentation verursacht sind. Da sich Technologien der Softwareentwicklung schnell weiter entwickeln und in Unternehmen häufig der Zwang besteht, neueste Technologien einzusetzen, ist es sehr wahrscheinlich, dass die heute konstruierten Softwaresysteme bald zu Altsystemen werden und eventuell migriert werden müssen. Um die künftige Migration von in der Gegenwart entwickelten Softwaresystemen leichter durchführen zu können, sollten entsprechende unterstützende Methoden und Werkzeuge angewendet werden. Grundlegend sind dabei Verständlichkeit und Konsistenz der Dokumentation, die uns auch nach Jahren ermöglicht, die damalige Analyse und Entwicklung nachvollziehen zu können.

2 Projekt TESSI – die Idee

Die teuersten Fehler bei der Softwareentwicklung entstehen dadurch, dass ein Auftraggeber seine Wünsche und Anforderungen an das für ihn zu entwickelnde oder anzupassende Softwaresystem oft nur teilweise, ungenau und/oder widersprüchlich beschreibt. Eine derart mangelhafte Beschreibung kann von den Mitarbeitern eines Softwareunternehmens (Auftragnehmer) nur partiell und ungenau begriffen werden, was aller Wahrscheinlichkeit nach zu einer Implementierung führt, die die eigentlichen Anforderungen des Auftraggebers nur unzureichend abbildet und folglich diesen nicht zufriedenstellen kann.

Die Kommunikation zwischen Auftraggeber und -nehmer ist auch deshalb schwierig, weil gewöhnlicherweise der IT-fremde Auftraggeber (Kunde) seine Wünsche und Anforderungen mittels einfachem natürlichsprachlichen Text beschreibt, während der Auftragnehmer (Softwareentwickler) sein Anforderungsverständnis zusätzlich durch UML-, ER- oder andere Diagramme präzisiert, die für den Kunden aber nur in wenigen Ausnahmefällen verständlich sind.

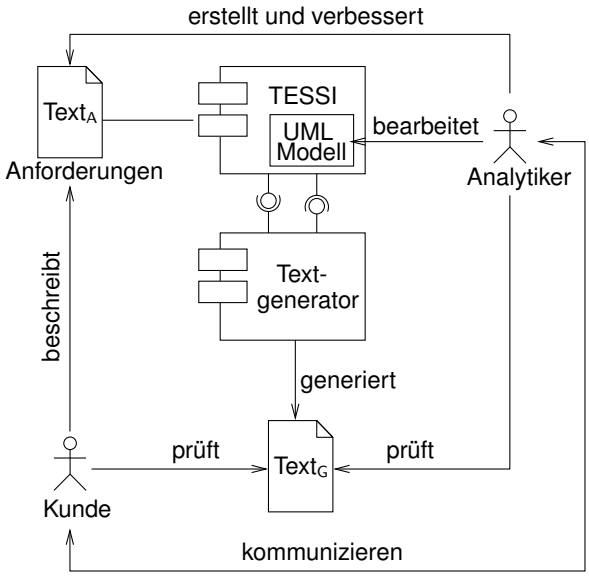


Abbildung 1: Anforderungsanalyse und -validierung mit TESSI ($Text_A$: zu analysierender Anforderungstext; $Text_G$: automatisch von TESSI generierter Text)

Im Rahmen des Projektes TESSI [KS97, Kro00, KGR06] wurden eine Methode (vgl. Abb. 1) und ein dazugehöriges Werkzeug entwickelt, die speziell die Phase der Anforderungserfassung, -analyse und -validierung in einem Softwareprojekt unterstützen. Der TESSI-Nutzer (Analytiker aus dem Softwarehaus) verfasst – eventuell in Zusammenarbeit mit dem Kunden[1] – einen Text, der die Anforderungen des Kunden an das zu entwickelnde Softwaresystem beschreibt, und analysiert anschließend diesen Spezifikationstext.

Das Werkzeug unterstützt den Analytiker dabei, einem Wort oder einer Wortgruppe (kurz: Textstellen), die er im Text ausgewählt hat, eine bestimmte Rolle im zukünftigen System zuzuordnen, indem er ein entsprechendes Modellelement ableitet. Über den Typ des Modellelementes muss

[1] Der Spezialfall aus der industriellen Praxis, bei dem der Kunde selbst einen ersten Text liefert, ist ebenfalls mit dem TESSI-Prozess vereinbar.

der Analytiker selbst entscheiden. Modellelemente können statisch (Akteure, Anwendungsfälle, Klassen, Attribute, Methoden, Assoziationen) oder dynamisch (Prozesse als Ketten von gesendeten Nachrichten und Ereignisse, welche die Prozesse starten und stoppen) sein.

Zu einem Modellelement (z.B. Akteur *user* oder Attribut *title* der Klasse *medium*) können nun umgekehrt Textstellen im Anforderungstext gefunden werden, die Informationen über diese Rolle enthalten (z.B. in: *The **user** specifies one or more **titles** and obtains a list of **media** which contain the **titles**.*). Synonyme oder grammatische Variationen (z.B. *customer* oder *users* für *user*) können den entsprechenden Modellelementen zugewiesen werden, sodass auch diese im Text aufgefunden werden.

Im Text werden Textstellen, die einem bestimmten Modellelement zugeordnet sind, entsprechend des Modellelementtyps farbig markiert. Somit lässt sich beim Lesen des Textes wiederum auf die Rolle der jeweiligen Textstelle im Modell schließen. Die farbige Markierung hat gegenwärtig allerdings den Nachteil, dass im Falle der Zuordnung einer Textstelle zu mehreren Modellelementen die visuelle Information über die Mehrfachzuordnung verloren geht, da jeweils nur eine Farbe angezeigt wird (*Search medium* kann beispielsweise der Name eines Anwendungsfalles oder einer Kollaboration sein).

Während der Analyse unterstützt TESSI die Entwicklung eines UML-Modells, das dem Spezifikationstext entspricht. Der Spezifikationstext ist in der Sprache des Kunden formuliert und spiegelt dessen Vorstellungen wider. Das entwickelte UML-Modell repräsentiert dagegen die Vorstellungen des Analytikers. Aus diesem UML-Modell lassen sich lesbare Texte in verschiedenen Formaten generieren, die dem Kunden vorgelegt werden können. Dieser kann nun feststellen, wie der Analytiker das Problem begriffen und gelöst hat. Findet der Kunde Widersprüche zu seiner Vorstellung oder eine unvollständige Repräsentation davon, können diese Fehler rechtzeitig korrigiert werden. Als Ergebnis eines iterativen Prozesses entsteht somit ein Pflichtenheft und ein daraus abgeleitetes Modell des zu entwickelnden Softwaresystems.

Das UML-Modell wird in XMI (XML Metadata Interchange), einem Standardformat der OMG zum Austausch von Modelldaten, gespeichert. Dadurch kann es prinzipiell von anderen UML-oder MDA-Tools importiert[2] und dort weiterverarbeitet werden.

3 Bedeutung von TESSI für den Software-Migrationsprozess

Durch den iterativen, rückgekoppelten Prozess von Anforderungserfassung, -analyse und -validierung sowie die enge Bindung zwischen Textstellen und Modellelementen kann ein hoher Grad an Konsistenz bzw. Nachvollziehbarkeit (Traceability) zwischen natürlichsprachlichem Text und Analysemodell erreicht werden. Somit ist es möglich, die einem zu migrierenden System zugrundeliegenden Anforderungen und deren Bezug zum Modell zu identifizieren, was zu einem besseren Systemverständnis führt und damit den Migrationsprozess unterstützt. Darüberhinaus wird eine systematische und einfachere Durchführung von eventuellen künftigen Änderungen in der Anforderungsspezifikation oder am Modell ermöglicht.

Es liegt auf der Hand, dass der beschriebene Prozess insbesondere (aber nicht ausschließlich) im Zusammenhang mit dem Thema Software-Migration seine Stärken umso mehr entfalten kann, wenn bei der Softwareentwicklung ein modellgetriebener Ansatz (Stichworte: MDSD, MDA[3]) verfolgt wird. Durch die *first-class* Fokussierung auf ein plattformunabhängiges, fachspezifisches Modell und der Generierung entsprechenden Quellcodes daraus wird Konsistenz zwischen Code und Modell und über TESSI auch zwischen Code und Anforderungsspezifikation erreicht – ein äußerst wünschenswertes Szenario für ein Migrationsvorhaben.

Literatur

[KGR06] KROHA, P., P. GERBER und L. ROSENHAINER: *Towards Generation of Textual Requirements Descriptions from UML Models*. In: ZENDULKA, J. (Herausgeber): *Proceedings of the 9th International Conference Information Systems Implementation and Modelling ISIM2006*, Band 105 der Reihe *ACTA MOSIS*, Seiten 31–38, April 2006.

[Kro00] KROHA, PETR: *Preprocessing of Requirements Specification*. In: IBRAHIM, M., J. KÜNG und N. REVELL (Herausgeber): *Proceedings of Database and Expert Systems Applications: 11th International Conference (DEXA'2000), London, UK, September 4-8, 2000*, Nummer 1873 in *Lecture Notes in Computer Science*. Springer, September 2000.

[KS97] KROHA, P. und M. STRAUSS: *Requirements Specification Iteratively Combined with Reverse Engineering*. In: PLASIL, F. und K. G. JEFFERY (Herausgeber): *SOFSEM'97: Theory and Practice of Informatics*, Nummer 1338 in *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[SV05] STAHL, THOMAS und MARKUS VÖLTER: *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management*. dpunkt-Verlag, Heidelberg, 1. Auflage, 2005.

[2] In der industriellen Praxis hat sich leider gezeigt, dass aufgrund verschiedener XMI- und UML-Versionen ein einfacher Austausch von UML-Modellen zwischen verschiedenen Tools oft nicht möglich ist. Dieser Umstand schränkt auch die gegenwärtige Exportfähigkeit von TESSI-Modellen ein.

[3] MDSD steht für **M**odel-**D**riven **S**oftware **D**evelopment und MDA für die **M**odel **D**riven **A**rchitecture der OMG, vgl. z.B. [SV05].

Eine integrierte Entwicklungsumgebung (IDE) für die BS2000-Migration auf der Basis von Eclipse

Denis Uhlig
pro et con Innovative Informatikanwendungen GmbH, Annaberger Str. 240, 09125 Chemnitz,
mailto: Denis.Uhlig@proetcon.de

Zusammenfassung:

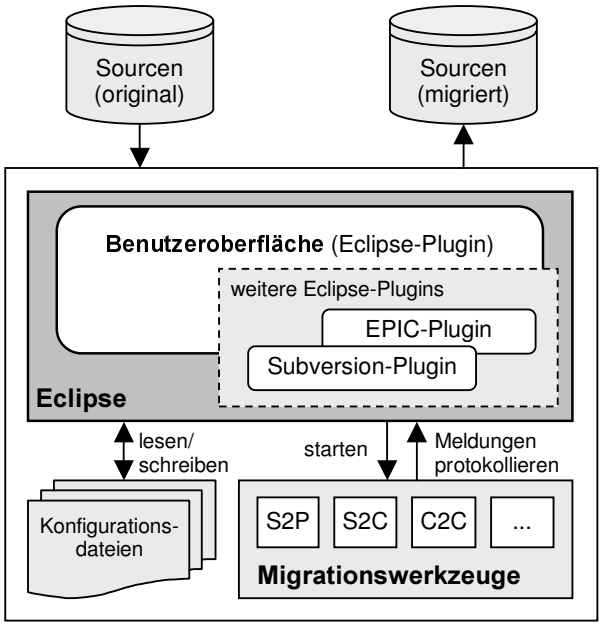
Komplexe Migrationsprojekte sind nur unter Einsatz von Werkzeugen in einem akzeptablen Zeit- und Budgetrahmen zu realisieren. Es existiert am Markt eine große Menge von Tools unterschiedlicher Hersteller, welche die verschiedenen Migrationskomponenten einer Softwareapplikation (Daten bzw. Datenbasis, Programme, Benutzeroberflächen, Middleware) mit unterschiedlicher Qualität unterstützen. Die genannten Werkzeuge arbeiten zum großen Teil kommandozeilenorientiert und autonom voneinander. Sie sind nur schwer intuitiv bedienbar, ihr Konfigurations- und Bedienungsaufwand ist hoch und die Effektivität leidet. Mit integrierten Entwicklungsumgebungen, welche die Steuerung und die Migrationsergebnisse der Werkzeuge harmonisieren, können Aufwände für Migrationsprojekte reduziert werden.

1. Ausgangssituation

Die Firma pro et con entwickelte im Rahmen eines Migrationsprojektes eine Reihe von kommandozeilenorientierten Tools, welche alle Aspekte einer BS2000-Migration in UNIX-Umgebungen adressieren (Daten, Nutzeroberflächen und Programme). Die Werkzeuge sind ausgetestet und befinden sich im praktischen Einsatz. Die Vereinigung dieser Werkzeuge in einer integrierten Entwicklungsumgebung (IDE) unter dem Namen BS2 Migration Manager soll deren Nutzung effektivieren.

2. BS2 MigMan

Die Grundlage des Werkzeugs BS2 MigMan bildet das Open- Source- Framework Eclipse. Dieses bietet neben den Qualitäten als IDE für die verschiedenen Sprachen (z. B. Java, C/C++) auch eine Plattform zur Anwendungsentwicklung. Auf dieser Basis ist es möglich, die vorhandene Entwicklungsumgebung durch zusätzliche Plugins zu erweitern. Die Nutzung von Eclipse begründete sich nicht nur durch dessen weite Verbreitung, sondern u.a. auch durch die ergonomische, funktionale Benutzeroberfläche und durch die vielfältigen und umfangreichen Integrations- und Kombinationsmöglichkeiten für zusätzliche Komponenten.



BS2 MigMan wurde als offene, generische Plattform realisiert. Die Architektur wird in der obigen Abbildung dargestellt.
Zur Steuerung der Migrationswerkzeuge werden Konfigurationsdateien, die in XML formuliert sind, genutzt. Diese enthalten Informationen über die Werkzeuge und ihre Optionen. Über spezielle Editoren werden die Konfigurationsdateien bearbeitet und die integrierten Werkzeuge gesteuert. Dadurch wird eine einheitliche Verwaltung von Werkzeugen und Migrationssourcen erreicht. Soll ein neues Werkzeug integriert werden, so ist nur das Formulieren und Einbinden einer weiteren Konfigurationsdatei notwendig.

3. Integrierte Migrations- Werkzeuge

SPL-to-C++ Translator (S2C)
Dieses Werkzeug realisiert eine automatische Konvertierung von Programmen, die in System Programming Language (SPL) kodiert wurden, nach C++. Die konvertierten Programme nutzen ein Laufzeitsystem, welches den BS2000-Kontext emuliert. Bekannte Technologien wie z. B. Überlagerungen bleiben in ihrer Funktionalität somit in der Zielumgebung bestehen.

SDF-to-Perl Konvertierer (S2P):
S2P ist ein Werkzeug für die Konvertierung von SDF- (System Dialog Facility) Prozeduren in Perl-Scripte. Die Konvertierung der SDF-Kommandos wird als gleichnamiger Perl-Funktionsaufruf mit analogen Parameterlisten realisiert. Das front- end von S2P kann optional zum Reverse Engineering genutzt werden. Es sind solche sinnvollen Funktionalitäten wie die Ermittlung ähnlicher SDF- Prozeduren (Clones) integriert. Die Architektur des dazugehörigen Laufzeitsystems besteht aus zwei Ebenen. Die untere Ebene bildet die Zugriffsschicht auf das Betriebssystem. Hier werden Elemente wie die Interprozeß-Kommunikation abgebildet. Die zweite Ebene stellt die Verbindungsschicht zwischen den Skripten und der Betriebssystemebene her. Die Ebene beinhaltet die Implementierung der konvertierten SDF-Kommandos. Die unter BS2000 genutzten Dienstprogramme (z. B. sort) werden ebenfalls bereitgestellt. Es wird dazu keine lizenzpflichtige Software verwendet.

COBOL-to-COBOL Konvertierer (C2C):
Diese in BS2 Migman integrierte Komponente konvertiert zwischen verschiedenen COBOL-Dialekten. Das betrifft z.B. Formatanpassungen der Sourcen, das Auskommentieren bestimmter, für den Mainframe typischer Paragraphen u.a.. Die Sourcen können nach der Konvertierung mit den unter UNIX (Linux) verfügbaren COBOL-Compilern genutzt werden.

4. Weitere BS2- Komponenten

Eine alternative Möglichkeit der funktionalen Erweiterung der IDE besteht in der Integration sogenannter Plugins. Für BS2 MigMan werden die Open-Source-Plugins EPIC [1] (Eclipse Perl Integration) und das Versionsverwaltungssystem Subversion [2] (SVN) verwendet. Eclipse ermöglicht die Entwicklung und Integration weiterer Plugins mit zusätzlichen, nutzerspezifischen Funktionalitäten.

5. Fazit und Ausblick

BS2 MigMan vereint in der aktuellen Version Werkzeuge der Softwaremigration unter einheitlicher Steuerung und ergonomischer Oberfläche. Eine Vervollständigung von BS2 MigMan durch Integration weitere Werkzeuge (Bearbeitung von SAM-/ ISAM- Files, Bildschirmmasken,..) über die Methode der Werkzeugintegration ist für die nahe Zukunft geplant.

[1] Eclipse Perl Integration (EPIC):
http://e-p-i-c.sourceforge.net/
[2] Eclipse Subversion Integration (subclipse):
http://subclipse.tigris.org/

Repositorygestützte Erkennung von Schnittstellen in einem hochintegrierten Anwendungssystem

Johannes Bach, Max Doppler, Martin Schulze
Debeka-Hauptverwaltung, D-56058 Koblenz
{johannes.bach,max.doppler,martin.schulze}@debeka.de

Zusammenfassung

Dieser Beitrag stellt dar, wie das zentrale *Software-Repository* der Debeka verwendet wird, um das *Kernsystem* der Debeka in *Teilsysteme* zu zerlegen und *Brücken* zwischen diesen zu bestimmen.

1 Unternehmen

Die *Debeka-Gruppe* besteht im Wesentlichen aus mehreren rechtlich eigenständigen Unternehmen: dem Krankenversicherungsverein a.G., dem Lebensversicherungsverein a.G., der Allgemeinen Versicherung AG, der Pensionskasse AG und der Bausparkasse AG. Im Gegensatz zu vielen anderen Versicherungskonzernen betreibt die Debeka schon von Anfang an ein *übergreifendes Rechenzentrum* für alle Teilunternehmen und entwickelt ihre Software in einer Hauptabteilung der Hauptverwaltung in Koblenz. Den größten Teil der Software bildet das *Debeka-Kernsystem*, das in COBOL geschrieben ist, auf einem CODASYL-Datenbanksystem beruht und auf einem Bull-Großrechner mit Betriebssystem GCOS8 läuft.

2 Kernsystem der Debeka

Das Kernsystem der Debeka besteht zunächst aus den *Fachanwendungen* (Bestands-/Vertragsverwaltung, Leistungsbearbeitung etc.) der einzelnen Unternehmen. *Zentrale Serviceanwendungen* stellen den Fachanwendungen unternehmensübergreifende allgemeine Funktionen (Verwaltung von Mitgliederdaten, Inkasso und Exkasso, Personalverwaltung, etc.) über entsprechende Zugriffsroutinen (COBOL-Unterprogramme) zur Verfügung. Dadurch ergibt sich ein *hochintegriertes Anwendungssystem*.

Ein Programm, an dem der hohe Integrationsgrad besonders deutlich wird, ist eine Online-Anwendung, die Personen-, Adress- und Kontodaten aus dem Partnersystem und Personaldaten aus der Personalverwaltung anzeigt. Gleichzeitig werden Daten über Versicherungsverträge aus den verschiedenen Bestandssystem abgerufen und in Form eines Vertragsspiegels in die Anzeige integriert. Nur die Mitgliederdaten können direkt in dieser Maske geändert werden. Für eine Bearbeitung der anderen Daten wechselt man in die entsprechenden Bearbeitungsprogramme der Fachanwendungen.

Auch auf fast allen anderen Dialogmasken werden Daten aus anderen Teilsystemen angezeigt, die ebenfalls von *Schnittstellen in Form von Unterprogrammen* bereitgestellt werden (z.B. der zuletzt bearbeitende Sachbearbeiter).

Das Kernsystem wird von ca. 4000 Online-Usern verwendet, die ca. 2 Mio. Transaktionen pro Tag durchführen. Es besteht aus ca. 10000 COBOL-Programmen mit 10,5 Mio. LOC und 10000 Jobs mit 1 Mio. LOC; die Datenbanken haben ein Volumen von ca. 2 TByte.

3 Software-Repository

Den Entwicklern der Debeka steht als Informationsquelle ein *zentrales Repository* („Zentrales Debeka-Repository“ / ZEDER) zur Verfügung, das die Softwarelandschaft abbildet und als Grundlage für das Auftragswesen und die Produktionsübernahme von Softwarebausteinen dient [BachSchulze2006]. Es beruht auf einer Oracle-Datenbank, auf welche die Anwendungen über ein gemeinsames Repository-API zugreifen. Befüllt wird das System manuell bei der Auftragserfassung über das Werkzeug ZEPRA („Zentrales Projektunterstützungs- und Auftragssystem“) und automatisiert durch Quellcode-Scanner und einen Importer bei der Produktionsübernahme. Im täglichen Betrieb greifen die Entwickler lesend mit dem ZEDERBrowser, einer Eclipse-Anwendung, generisch auf die Inhalte des Repositorys zu. Mit ihm navigieren sie von Objektmengen zu benachbarten oder über Abkürzungslinks erreichbaren Objekte, um Abhängigkeiten zu ermitteln und Objekt-Informationen anzuzeigen. Für das im folgenden beschriebene Projekt gibt es eine Anwendung, die über spezielle Anfragen genau die benötigten Informationen extrahiert und zur Weiterverarbeitung aufbereitet.

Das Repository speichert Informationen über die Module des Kernsystems sowie einiger dezentraler Systeme *grob- bis mittelgranular* und über die Datenbanken *feingranular*. Beispielsweise werden zu einem COBOL-Programm Zugriffe auf logische Dateien oder Felder von Datenbanken, Aufrufe von Unterprogrammen und Einbindung von Copybooks im Repository abgebildet, nicht aber interne Variablen oder der Kontrollfluss. Die Datenbanken sind bis auf die Feldebene abgebildet.

Typische Abfragen eines Entwicklers sind beispielsweise: Welche Unterprogramme werden direkt oder indirekt von einer Menge von Programmen aufgerufen? Welche Programme werden in einem Job oder in einer Menge von Jobs verwendet? Welche Programme sind betroffen, wenn ein Datenbank-Objekt (oder ein Copybook) geändert wird?

4 Teilsystembildung und Brückenerkennung

In einem aktuellen Projekt stellt sich nun die Aufgabe, das Kernsystem in *Teilsysteme* zu zerlegen und *Brücken* zwischen den Teilsystemen zu finden. Damit soll es ermöglicht werden, das Kernsystem in mehreren Phasen

auf eine neue Zielumgebung zu *migrieren*. Aus technischen Gründen kann aus der Zielumgebung nur lesend auf die nicht migrierten Teilsysteme zugegriffen werden, jedoch lesend und schreibend in umgekehrter Richtung. Die Brücken müssen also so gefunden werden, dass Schreibzugriffe nur aus den nicht migrierten Teilsystemen in die neue Zielumgebung erfolgen. Das *Repository* kommt sowohl zur *Brückenerkennung* zum Einsatz, als auch zur Erkennung *vorbereitender Maßnahmen*, mit denen problematische Brücken vor der Migration entfernt werden können.

4.1 Brückenerkennung

Im ersten Schritt werden aufgrund von *Namenskonventionen* die Online- und Batchprogramme sowie die Jobs und Datenbanken der in einer Phase zu migrierenden Teilsysteme bestimmt. Mit Hilfe des Repositorys werden ausgehend von den Hauptprogrammen durch *rekursive Anfragen* sämtliche Unterprogramme ermittelt, die von diesen Hauptprogrammen direkt oder indirekt verwendet werden. Die *Zugriffe auf Datenbanken* in den aufgerufenen Unterprogramme können so dem jeweiligen Hauptprogramm zugeordnet werden. Durch die rekursive Suche entstehen theoretisch mögliche Aufrufketten, aus denen die Brücken aus dem Zielsystem in noch nicht migrierte Teilsysteme abgeleitet werden können. Analog wird dieselbe Analyse für die Hauptprogramme der nicht migrierten Teilsysteme durchgeführt. Da dies eine *statische Analyse* ist, kann natürlich nicht festgestellt werden, ob die Codeteile, die den Aufruf enthalten, in einer spezifischen Situation tatsächlich durchlaufen werden. Dies kann nur in einer *manuellen* oder *dynamischen Codeanalyse* abschließend beurteilt werden.

4.2 Brückenvermeidung

Auf der Basis der ermittelten Brücken wird schließlich die *Zuordnung von Modulen zu Teilsystemen* so vorgenommen, dass möglichst wenige Brücken zwischen dem Zielsystem und den nicht migrierten Teilsystemen bestehen und Schreibzugriffe nur aus den noch nicht migrierten Teilsystemen in Richtung Zielsystem erfolgen. Im folgenden werden beispielhaft fünf Arten von Brücken beschrieben und es wird dargestellt, wie sie mit Hilfe des Repositorys gefunden und beseitigt werden.

4.2.1 Fremdzugriffe

Ein *Fremdzugriff* liegt vor, wenn ein Programm aus einem Teilsystem direkt auf die Datenbank eines anderen Teilsystems zugreift. Dies wird dadurch erkennbar, dass in einer Aufrufkette das Hauptprogramm oder ein Unterprogramm desselben Teilsystems direkt mit einem DML-Befehl auf eine Datenbank eines anderen Teilsystems zugreift. Ein Fremdzugriff würde die gleichzeitige Migration beider Teilsysteme erforderlich machen. Deshalb werden Fremdzugriffe durch Aufrufe von Unterprogrammen ersetzt, die von dem anderen Teilsystem als Schnittstellen bereitgestellt werden.

4.2.2 Schreibende Datenbank-Zugriffe aus dem Zielsystem

Ein *Schreibzugriff aus dem Zielsystem* liegt vor, wenn ein Programm aus dem Zielsystem ein Unterprogramm eines nicht migrierten Teilsystems aufruft, in dem ein schreibender Datenbankzugriff erfolgt. Dies ist, wie oben erwähnt, technisch nicht möglich. Deshalb müssen die Hauptprogramme auf dem alten System belassen werden, falls der Datenbankzugriff nicht vermieden werden kann.

4.2.3 Brückenrücksprünge

Ein *Brückenrücksprung* liegt vor, wenn die Plattform in einer Transaktion mehr als einmal gewechselt wird. Er ist dadurch erkennbar, dass zu einer ermittelten Brücke in einer Richtung auch Aufrufe in der anderen Richtung erfolgen. Brückenrücksprünge sind nicht erwünscht, da sie die Transaktionssicherheit gefährden. Entweder kann ein Entwickler durch manuelle Analyse ausschließen, dass die identifizierten Rücksprünge in der Praxis tatsächlich durchlaufen werden, oder die beteiligten Module sind soweit zu modifizieren, dass ein Rücksprung vermieden wird.

4.2.4 Plattformübergreifende Jobabläufe

Ein *plattformübergreifender Jobablauf* kann im Repository direkt abgelesen werden, wenn in einem Job ein Programm eines anderen Teilsystems aufgerufen wird, das sich auf der anderen Plattform befindet. Ein solcher Aufruf ist aus technischen Gründen ebenfalls nicht möglich. In diesem Fall muss entweder die Zuordnung des Jobs oder des Programms geändert oder der Job geteilt werden.

4.2.5 Plattformübergreifende Dateizugriffe

Ein weiterer Fall betrifft den *Datenaustausch zwischen Jobs mit Hilfe von Dateien*. Mit Hilfe des Repositorys wird festgestellt, ob Jobs, die auf einer oder mehreren gemeinsamen Dateien arbeiten, unterschiedlichen Teilsystemen zugeordnet sind. Ist dies der Fall, werden die Jobs um einen FTP-Transfer erweitert.

Fazit

Durch geeignete Anfragen können die Beziehungsinformationen des Repositorys genutzt werden, um ein hochintegriertes Anwendungssystem in möglichst unabhängige Teilsysteme zu zerlegen. Weiterhin unterstützt das Repository die Detailanalyse von Brücken zwischen den Teilsystemen und hilft bei der Einleitung von Maßnahmen zur Brückenvermeidung. Dadurch ist es möglich, das Anwendungssystem in mehreren Phasen auf ein neues Zielsystem zu migrieren.

Literatur

BachSchulze2006 Johannes Bach, Martin Schulze: *Migration des Debeka-Software-Repositorys auf ein RDBMS.* In Rainer Gimnich, Andreas Winter (Hg.): *Proceedings of 8. Workshop Software-Reengineering*, Softwaretechnik-Trends, Band 26(2), 2006.

Software Migration einer Mainframe-Anwendung - Ein Praxisbericht

Jörg Hahn
T-Systems GEI GmbH
Clausstraße 3, 09126 Chemnitz
+49 371 5359-115 (Tel.)
E-Mail: Joerg.Hahn@t-systems.com

Ausgangspunkt

T-Systems übernahm Anfang 2004 die Verantwortung für die Wartung und Weiterentwicklung der Anwendung REDI3. Der Auftraggeber ActiveBilling GmbH & Co. KG übergab die bis dahin durch einen externen Dienstleister entwickelte und gewartete Anwendung an T-Systems mit dem Ziel, die Wartungskosten signifikant zu senken.

Mit über zwei Millionen fakturierter und gedruckter Rechnungen täglich und rund 42 Millionen geführter Debitorenkonten gehört ActiveBilling als Tochtergesellschaft der Deutschen Telekom AG rund um Billing und Collection zu den Marktführern in Europa. Bei der hier vorgestellten Anwendung handelt es sich um ein Auftragslenkungssystem zur maschinellen Übernahme bzw. Weitergabe rechnungsrelevanter Daten aus der Auftragserfassung. Das System besteht aus einem Batch- und einem Dialog-Teil mit 30.000 Windows-PC zur Nachbearbeitung, Neuerfassung und zum Recherchieren von Geschäftsfällen.

Unter der Rahmenbedingung, dass die funktionale Weiterentwicklung nicht beeinträchtigt werden darf, erhielt T-Systems die Aufgabe, den unter IBM Mainframe betriebenen Batch-Teil auf die Sun Solaris Plattform des Dialog-Teils dieser Anwendung zu migrieren.

Eine erhebliche Betriebskostenersparnis ließ sich erwarten, da die Dialog- und Batch-Verarbeitung komplementäre Lastprofile aufwiesen und nach der Migration eine Plattform komplett wegfallen konnte. Eine Reduktion der Plattformen bildete außerdem eine notwendige Basis für weitere Refactoring-Maßnahmen.

Entscheidungsgrundlage

Im Rahmen einer Vorstudie fand eine Überprüfung der Machbarkeit der Migration und des Business Case statt. Das in diesem Zusammenhang erstellte Migrationskonzept beschrieb die Lösungsansätze und eine Abschätzung der wesentlichen Aufwände.

Weiterhin wurde dazu im Rahmen einer Pilotierung ein Teilprozess der Anwendung migriert und in einer betriebsnahen Testumgebung vermessen. Die Auswahl dieses Referenzprozesses erfolgte mit dem Ziel, durch eine Hochrechnung der Messergebnisse eine Abschätzung des Hardwarebedarfs der kompletten Anwendung zu ermöglichen sowie die Lösungsansätze des Migrationskonzeptes zu validieren.

Erst nach erfolgreichem Abschluss der Konzeption und der Pilotierung fiel die Entscheidung über die Durchführung der Migration.

Prämissen

Der erste Kernpunkt der Umstellungsstrategie legte fest, dass Anpassungen an der Software auf das unumgängliche Minimum reduziert werden (1:1 Portierung), um damit eine Komplexitätsreduzierung der Migration zu erreichen. Weitere Refactoring-Maßnahmen wurden erst für spätere Versionen geplant.

Eine weitere Prämisse bestand darin, die Anpassungen der Software an die neue Plattform parallel zur fachlichen Weiterentwicklung durchzuführen. Durch Automaten ließen sich die währenddessen auf der alten Plattform entwickelten fachlichen Änderungen in die Entwicklungsumgebung der Migration übernehmen. Auf dieser Basis war es möglich, die Anwendung unabhängig vom Zeitdruck notwendiger fachlicher Weiterentwicklungen an die neue Plattform anzupassen.

Der abschließende Parallelbetrieb der Anwendung mit alter und neuer Plattform verhalf entscheidend dazu, die Qualität und die Betreibbarkeit der Anwendung sicherzustellen.

Die Migrationsaufgabe

Folgende technische Parameter charakterisierten die zu migrierende Anwendung:

- Dialog-Teil der Anwendung bestehend aus 30.000 Windows PC und dem Dialog-Server mit einer Informix Datenbank unter Solaris
- Batch-Teil auf IBM-Mainframe mit 15 DB/2 Datenbanken (15 LPAR) die insgesamt 1,25 TB Plattenplatz für Nutzdaten benötigten
- Synchronisation der Daten zwischen den 15 DB/2 und der Informix Datenbank

- ca. 1000 COBOL-Module (1,8 Mio Lines of Code), ca. 600 Jobs, ZEKE-Jobnetzsteuerung
- ca. 70 Schnittstellen zu anderen Anwendungen
- 5000 MIPS Spitzenlast

Für die Migration der Anwendung auf die Sun Solaris Plattform waren nachfolgende Anpassungen notwendig:

- manuelle Ablösung von ZEKE durch UC4
- Ablösung JCL durch Shellskripte (bash) über einen eigenen Konverter
- manuelle Ablösung der Dienstprogramme und REXX-Skripte durch Perl-Skripte
- Ablösung eines einzelnen CICS-Programmes durch Eigenentwicklung
- Migration der 15 DB/2 Datenbanken zu 15 Instanzen einer Oracle Datenbank
- Ablösung des IBM COBOL Compilers durch Micro Focus COBOL
- Entwicklung eines eigenen SQL Konverters
- Einsatz von Präprozessoren für das Generieren zusätzlicher Modulaufrufe in die COBOL-Sourcen

Der Parallelbetrieb

Nach Realisierung und Test der migrierten Anwendung wurde das Gesamtsystem unter Echtbedingungen auf der neuen Plattform parallel zum Betrieb ausgeführt. Jeder Testlauf begann mit der Durchführung der Datenmigration analog dem Vorgehen des geplanten Versionswechsels und anschließendem Anwendungsstart.

Um einen exakten Vergleich aller Funktionen auf beiden Plattformen zu ermöglichen, bestand die Notwendigkeit, auch alle Benutzereingaben des Dialogs parallel in das alte und neue Batch-System zu leiten. Eine entsprechende Anpassung der bestehenden Anwendung ermöglichte das.

Nach Ablauf eines Verarbeitungstages wurden dann alle von der Anwendung geänderten Daten einschließlich der Datenbankinhalte maschinell in ein normiertes Format konvertiert und miteinander verglichen.

Mit dem Parallelbetrieb fand nicht nur die Überprüfung der Funktionalität der Anwendung statt. Neben dem Training der Betriebsmannschaft erfolgte auch die Validierung der betrieblichen Verfahren und Werkzeuge.

Vor allem die Stabilität des zukünftigen Betriebes hinsichtlich Performance, Robustheit und betrieblicher Abläufe ließ sich damit sicherstellen.

Erfahrungen

Das Vorgehen mit den folgenden Schwerpunkten hat sich bewährt:

- Pilotierung
- Entwicklung der Migration parallel zur Weiterentwicklung mit automatischer Übernahme fachlicher Änderungen
- Durchführung eines Parallelbetriebs

Der Aufwand zur Erstellung von Automaten für die Übernahme fachlicher Änderungen in den Quellcode, für die Datenmigration, für den Vergleich der Daten und für Konvertierungen war lohnenswert. Gleiches gilt für den Parallelbetrieb.

Letztlich wurde nicht nur die Anwendung migriert, sondern die Analyse bzw. Anpassung der Software führte auch automatisch zum Aktualisieren der fachlichen und technischen Dokumentation.

Durch das gewählte Vorgehen und den hohen Automatisierungsgrad konnte die Migration zeitlich von der Umsetzung fachlicher Weiterentwicklungen entkoppelt werden. Das hat entscheidend zur Risikominimierung beigetragen.

Weiterhin lassen sich folgende Empfehlungen geben:

- Für die Detail-Klärung von Schnittstellen sollte genügend Zeit und Aufwand eingeplant werden.
- Aus einem Softwareprojekt wird ein „Change-Projekt“, falls die Betriebsmannschaft noch keine umfangreichen Erfahrungen auf der Zielplattform besitzt. Dementsprechend sind Zeit und Aufwand für Kommunikation und Schulungen einzuplanen.
- Das Anpassen der Schnittstellen von Partnersystemen und deren Inbetriebnahme kann durch ein Ausweichkonzept vom Zeitpunkt der Wirkbetriebsaufnahme der migrierten Anwendung entkoppelt werden.

Fazit

Die portierte Anwendung hat im Januar 2006 erfolgreich den Betrieb aufgenommen. Von Anfang an war die Fehlerrate sehr gering und es traten keinerlei Abbrüche auf. Mit dieser Migration wurde die Basis für weitere Modernisierungen geschaffen.

Das hier umgesetzte Vorgehen lässt sich generell auf weitere Migrationsvorhaben bzw. Refactoring-Maßnahmen anwenden.

Eine Plattformmigration ist eine gute Alternative gegenüber der Neuentwicklung einer Anwendung. Die Risiken eines solchen Migrationsprojektes sind kontrollierbar und ein sukzessiver Übergang zu neuen Technologien und Paradigmen wird damit ermöglicht.

Das Migrationsprojekt HIT „(H)armonisierung und (I)ntegration von (T)DB-Datenbanken“ bei MAN Nutzfahrzeuge AG

Theo Schuller, Dipl. Ing. (FH)
Teamleiter Parametrierung,
Zentralbereich für Technische Dienstleistung und Beratung,
MAN Nutzfahrzeuge AG, München,
mailto: Theo.Schuller@de.man-mn.com

Die Abteilung Technische Dienstleistung und Beratung (TDB) der MAN Nutzfahrzeuge Gruppe erstellt unter anderem Dokumente wie z.B. Hersteller-Bescheinigungen/ -Bestätigungen/ -Freigaben und –Genehmigungen für Fahrzeugumbauten sowie Bestätigungen über Fahrzeugparametrierung (kurz Parametrierbescheide). Diese Dokumente werden weltweit an Töchter-/ Importgesellschaften, Behörden, Service-Niederlassungen, Werkstätten, amtliche Überwachungsorganisationen, Aufbaufirmen, Kunden uvm. versendet. Diese sind verbindliche Dokumente und haben wesentlichen Einfluss auf Zulassungsdokumente, Garantieleistungen und Wartungsverträge.

Im Bereich der Fahrzeugelektronik bieten die Parametrierbescheide eine verbindliche Beschreibung der an einem Fahrzeug von autorisierten zentralen Stellen vorgenommenen Änderungen in der Elektronikarchitektur, angefangen von Hardware- über Softwarekomponenten und Funktionsparametersätze bis hin zu fahrzeugindividuellen Einzelparameter-einstellungen.

Derzeit werden jährlich ca. 20.000 ein- bis dreiseitige Dokumente erstellt, Tendenz steigend mit der Zunahme der zu betreuenden Fahrzeugpopulation im Feld.

In der Migrationsdatenbank werden derzeit 140.000 Datensätze für die hier genannten Dokumente verwaltet. Dabei werden alle kundenspezifischen Modifikationen eines jeden einzelnen Fahrzeuges berücksichtigt. Die Fahrzeuge entsprechen über deren gesamten Lebenszyklus einer Serienfertigung in Losgröße Eins. Für jedes einzelne Fahrzeug wird ein Fahrzeuglebenslauf dokumentiert und der weltweiten Serviceorganisation zur Verfügung gestellt.

Da Nutzfahrzeuge langlebige Wirtschaftsgüter sind, müssen diese fahrzeugbezogenen Daten über einige Jahrzehnte hinaus weltweit zur Verfügung stehen. Letztendlich soll auch ein künftiger Oldtimer, der seine treuen Dienste in einer beliebigen Region der Welt erbringt, bezüglich seiner Hard- und Softwarekomponenten betreut werden können. Nebst einer klassischen Ersatzteilelogistik ist eine Fahrzeugdatenlogistik von unternehmensstrategischer Bedeutung.

Nach dem Abschluss des zusammen mit der Fa. PRO ET CON GmbH umgesetzten Migrationsprojektes HIT wodurch die derzeit in einzelne historisch gewachsene Datenbanken vorhandenen Daten in eine einheitliche Datenstruktur überführt werden, wird die für Nutzfahrzeuge verantwortliche Fachabteilung TDB diese Applikation systematisch zum Unternehmensstandard ausbauen und in andere Bereiche portieren, um damit weitere technische Dienstleistungen für MAN-Produkte abbilden zu

können (Betreuung von Fahrzeugeinbaumotoren, Linien- und Reisebusse, Marine-, Bahn- und Industriemotoren). Damit wird eine gleich bleibende Qualität bezüglich Inhalt, Konsistenz und Revisionssicherheit dieser produktbezogenen Daten und Dokumente erreicht. Bereichsübergreifende Synergieeffekte lassen eine Potenzierung der Wirkung hinsichtlich einer Produktivitätssteigerung als zusätzliches wirtschaftliches Ziel des Projektes zu.

Heute an mehreren Stellen anfallende Verwaltungsaufwände werden nach der Migration zusammengeführt und auf ein Minimum reduziert, da eine weitestgehende Automation der Prozesse vorgesehen ist.

In der Präsentation wird ein erster Schwerpunkt die Darstellung der Migration von Fahrzeugdaten aus dem Bereich Fahrzeugparametrierung sein. Hier werden Metadaten verwendet um eine automatische Protokollierung von Transaktionen in offizielle Dokumente zu überführen.

Als ein weiterer Schwerpunkt werden die Ansätze der Vernetzung der HIT-Applikation mit einem bereits bestehenden CM-System (content management) der MAN, welcher die Verfügbarkeit eines Dokumentes in mehreren Sprachen gewährleistet, vorgestellt. Ein bedeutender wirtschaftlicher Vorteil, welches mit der Umsetzung dieses Migrationsprojektes entsteht, ist die Reduzierung der entstehenden Kosten für die Lokalisierung einzelner Dokumente bei gleichzeitiger Vermeidung von Informationsverlusten durch personenabhängige Interpretation von in Fremdsprachen verfassten Dokumenten. Sowohl die Antragsteller aus derzeit über 60 Ländern als auch die Sachbearbeiter in der Zentrale werden von Übersetzungs- und Interpretationsarbeiten entlastet, da jede Seite ein Dokument in der eigenen Sprache zu derselben Transaktion vorfinden kann.

Um die Verantwortlichkeiten festzulegen und ein Eskalationsverfahren für den Betrieb der Applikation zu installieren, werden für alle Schnittstellen zwischen HIT und anderen MAN-Anwendungen sowohl Abteilungen als auch Fachpersonal genannt. Diese sind bei Störungen des automatischen Ablaufs der installierten Prozesse in der Lage, mittels geeigneter Werkzeuge, die Schnittstellen manuell auf der entsprechenden Rückfallebene zu bedienen. Dafür müssen realistische Szenarien durchgespielt werden, Mitarbeiter müssen geschult werden, die Verantwortlichkeiten und Prozeduren für den Eskalationsfall müssen sowohl in den einzelnen Fachbereichen als auch in der zentralen IT-Verwaltung hinterlegt werden.

Als Ergebnis des Migrationsprojektes HIT werden nicht nur mehrere historisch gewachsene Datenbanken zusammen-geführt, sondern auch in anderen Unternehmensbereichen vorhandene Daten so zusammengeführt, dass die bereits installierten Werkzeuge für statistische Untersuchungen um weitere Qualitäts-merkmale erweitert werden können. Damit können zum Beispiel Kundenwünsche für die Entwicklung künftiger Produkte besser und präziser erfasst, quantifiziert und bewertet werden.

Die Nähe des Fahrzeugherstellers MAN zu seinen Kunden spiegelt sich nicht nur in den Premiumprodukten und der Qualität der angebotenen after-sales Dienstleistungen wieder, sondern auch in der technologischen Fähigkeit, Kunden-bedürfnisse direkt aus dem weltweiten Markt zu erfassen und proaktiv in die vorhandenen Produktenstehungsprozesse (PEP) zu integrieren. Damit werden in der Praxis gewonnene Erkenntnisse direkt in die Entwicklung neuer Produkte eingesteuert, womit der Anspruch der Technologieführerschaft des Unternehmens MAN deutlich unterstrichen wird.

Model-based Migration to Service-oriented Architectures

Andreas Winter
Johannes Gutenberg-Universität Mainz
Institut für Informatik,
D-55128 Mainz
http://www.gupro.de/~winter

Jörg Ziemann
Institut für Wirtschaftsinformatik im
Deutschen Forschungszentrum für künstliche Intelligenz
D-66123 Saarbrücken
http://www.iwi.uni-sb.de/

Abstract

The systematic development of service-oriented architectures in conjunction with reusing already existing software requires integration of model based software development techniques and software migration strategies. Correspondingly, we propose an approach that combines and extends methods from enterprise modelling and re-engineering for developing service-oriented architectures.

1. Motivation

Service-oriented architectures (SOA) promise flexible composition of components implementing well defined business tasks to facilitate adaptability of software systems, enabling enterprises to cope with fast changing business requirements. The development of SOA requires a profound knowledge of enterprises. *Enterprise process models* describing static as well as dynamic aspects of an organization are an established means to capture such knowledge. While static elements like business functions and organizational structures of an enterprise shape individual services, dynamic business process models drive the definition of service orchestrations. *Services* specify encapsulated functionality independent from concrete implementation issues. Services may interact to provide compound services. The functionality of services is implemented by *components* fulfilling the service specification, which can easily be composed to comprehending software systems. [Arsanjani, 2004].

On the other hand, most enterprises already run software systems that are established and time-proven. These legacy systems usually are not implemented in a service-oriented way. They provide software functionality supporting enterprises business processes by monolithic and deeply interwoven software modules. Thus, evolving legacy systems towards supporting changing business processes is limited.

Keeping legacy systems evolvable requires migration into more flexible architectures. Software migration is viewed as a transformation of software systems into a new environment, without changing its functionality [Gimnich/ Winter, 2005].

This paper discusses an approach towards migrating legacy systems to service oriented architectures by relating the intended enterprise model to the legacy software system.

2. Migration to SOA

Migrating legacy software systems to service-oriented architectures is influenced by the intended enterprise model and the legacy software system. In SOA migrations, the intended enterprise model defines the business needs of the target design (cf. [Ackermann et al., 2005]), which is determined by services and their interactions. The legacy system already contains required functionality of the service oriented target system.

In order to build flexible and reusable systems which obtain the full business potential of SOA, a sound methodology is required, taking into account existing systems and current business requirements of enterprises as well. This includes the definition of services and service interactions regarding business needs, reflected in the new enterprise model, combined with already implemented functionality based on earlier (probably no longer available) enterprise models. In a more technical vein, components implementing services have to be discovered and extracted from the legacy system and have to be transferred into the intended SOA implementation [Ziemann et al., 2006].

The presented approach on migration to service oriented architectures comprises technologies from software reengineering and business process modelling. Fig. 1 sketches a horseshoe model [Kazman et al., 1998] on SOA migration.

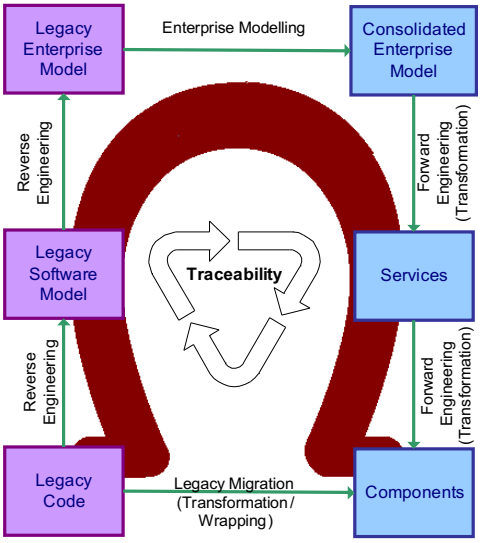


Figure 1 SOA-Migration Horseshoe

Legacy and target system are viewed on three conceptual levels showing the *code level* (Legacy Code, Components), the (platform specific) *model level* (Legacy Software Model, Services), and the *conceptual level* (Legacy Enterprise Model, Consolidated Enterprise Model). The left part presents artifacts from the legacy system. The right part presents the service oriented target system following the SOA layering [Arsanjani, 2004].

The approach follows a model-based manner. Software artefacts on all levels of abstraction are represented by appropriate models. These models provide the foundation for further *reverse engineering, enterprise modelling, forward engineering*, and *legacy migration* activities.

Reverse Engineering lifts representation of a software system on a higher level of abstraction. The main objective of the reverse engineering step in a SOA migration is to support the understanding of already existing software systems and deriving information for higher level software and enterprise models. Starting from its *source code*, re-documentation activities are used to extract a *software model*, which allows identifying services provided by the legacy system. Using additional techniques from business process modelling the software model (and other available artefacts) are used to derive an enterprise model for the legacy system. Re-documentation activities include static analysis to identify major components and dynamic analysis to determine information exchange.

Enterprise Modelling techniques are used to capture current enterprise requirements and to optimize the resulting models for a model-driven transformation to SOA. The requirements from the legacy system are summarized in the *legacy enterprise model.* Deriving the legacy enterprise models requires, next to business modelling techniques, additional requirements engineering techniques to (partially) extract those information form the *legacy software model*. Current enterprise models will be consolidated with legacy enterprise models and adapted for opportunities offered by SOA, e.g. by a stronger focus on enterprise components and new process concepts like service choreography. The outcome is a *consolidated enterprise model*. This model contains the non-functional requirements to migrate to service oriented architecture and all requirements valid for those services taken over from the legacy system.

Forward engineering comprises all activities to realize the resulting software system. In a model driven manner, the *consolidated enterprise model* represents the platform independent model, which is transformed into a platform specific model. In service oriented architectures, this platform specific model (*services*) specifies all required services and their interaction. Further transformation (and implementation) steps result in implementations by *components*. Since service implementations already exist in the legacy systems, this transformation step includes identification of those components from the legacy code.

Legacy Migration (Transformation/Wrapping) deals with transferring source code to a new environment. In SOA migration, this new environment is a set of SOA *components*. The migration activities include the discovery of service implementation in the *legacy code*, the extraction of these modules, and their adoption into the target SOA environment by wrapping or transformation.

These migration activities strongly rely on the models, serving as base for extracting abstractions or transformations. Integrating these models according to a shared metamodel provides traceability between the participating models. **Traceability** visualizes the interconnection between participating models. During migration to a service oriented architecture traceability facilitates the identification of appropriate chunks of legacy code to be extracted for certain services. Following traceability links from services in the service model, via the consolidated enterprise model, the legacy enterprise model, the legacy software model, and to the legacy code, establishes references between the implementation of services by components and the legacy implementation. From a business side, traceability is important to ensure the compliancy of IT systems with enterprise demands and for controlling purposes, e.g. for correlating the outputs of a certain component to a business function. If the resulting SOA implementation only wraps some legacy components, these traceability links also support further maintenance activities.

3. Conclusion

This proposal originates in discussions started at the Workshop "Software-Reengineering and Services", held at WI 2006 in Passau. An important result of these discussions was the adoption, that successful software migration to service oriented architectures has to bridge (business oriented) enterprise models and (technical oriented) software models. The here presented approach tries to combine enterprise modelling and software reengineering techniques to a holistic software migration method.

4. References

[Ackermann2005ERD] Ackermann E., Gimnich, R., Winter, A.: Ein Referenz-Prozess der Software-Migration (erweiterte Kurzfassung), Softwaretechnik-Trends. 25(4). 2005, 20-22.

[Arsanjani, 2004] Arsanjani, A.: Service-oriented modeling and architecture, How to identify, specify, and realize services for your SOA, IBM, 2004, http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/

[Gimnich/Winter, 2005] Gimnich, R., Winter, A.: Workflows der Software-Migration, Softwaretechnik-Trends. 25(2). 2005, 22-24.

[Kazman et al.,1998] Kazman R., Woods S., Carriere, J.: Requirements for Integrating Software Architecture and Reengineering Models: CORUM II, Working Confernce on Reverse Engineering, IEEE Computer, 1998, 154-163.

[Ziemann et al., 2006] J. Ziemann, K. Leyking, T. Kahl, D. Werth: Enterprise Model driven Migration from Legacy to SOA. In: Gimnich, R.; Winter, A.: Workshop Software-Reengineering und Services, MKWI, Passau, 2006.

SOA Migration – Approaches and Experience

Rainer Gimnich
IBM Software Group
SOA Advanced Technology / Enterprise Integration Solutions
Wilhelm-Fay-Str. 30-34, D-65936 Frankfurt
gimnich@de.ibm.com

Summary

Due to their economic and technological benefits, Service Oriented Architectures (SOA) are valued more and more by companies in all industries and sizes. This paper explains SOA briefly and presents approaches for migrating to an SOA in 'real life'. We will discuss how SOA design methods and proven software reengineering technology can be combined in order to support a company's SOA adoption roadmap.

1. SOA

Service Oriented Architecture (SOA) is a business-centric IT architectural approach that supports integrating business as linked, repeatable business tasks, or **services**. Web Services provide a standardized, cost-effective implementation of such services (but this is not mandatory for an SOA). The services that a company wants to expose (internally or to the market) must be commonly understood by the business and the IT departments (see central 'layer' in Figure 1).

Then the business requirements can be implemented with a view to services, which also form major elements of the **business processes** to support: The services are invoked where they are required in the process; they can also be accessed directly from consumers (e.g. via a portal).

Apart from the processes there are other concepts that are important on the business side, for example the business competencies and their use, the business drivers and needs, the business performance context, etc. All of these provide valuable input for service modeling.

The **service model** includes identification of service candidates, service exposure decisions, service specification and realization decisions. The service specification includes service component specifications and flow specifications (both of services and components).

This leads us to the **service components** layer, where the Enterprise Components pattern has been applied successfully for grouping functionality. The service components usually rely on **operational systems** for their implementation (e.g. custom applications or software packages, 'legacy' in a positive sense).

This is a major feature of an SOA: the architecture is achieved in an evolutionary way, evaluating and migrating the existing landscape (or particular business areas) into an SOA in a stepwise manner. Each step can leverage its own benefits (higher reuse, faster time-to-market, lower development / maintenance effort, etc.).

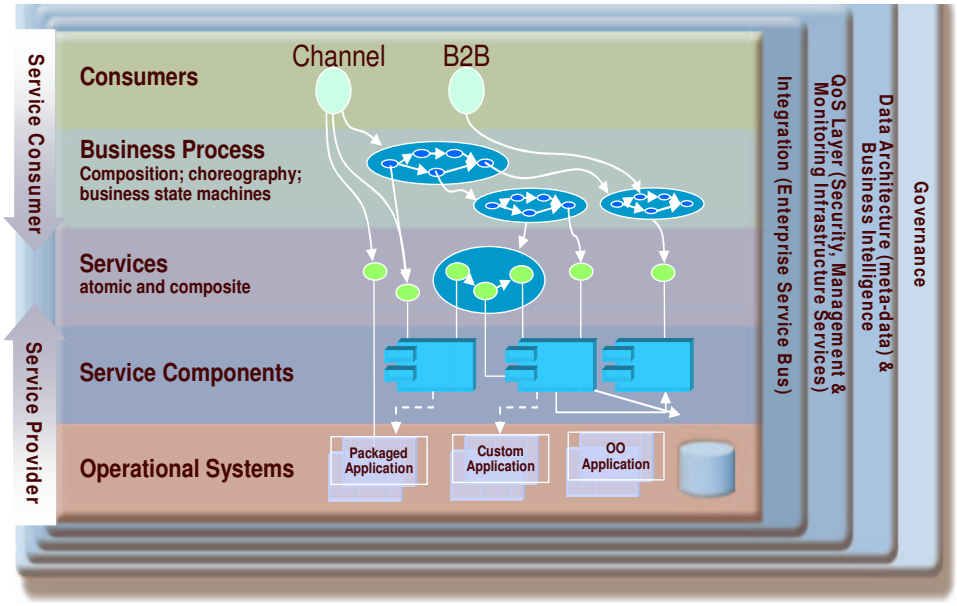


Figure 1: SOA conceptual layers

2. SOA Migration

Migration in IT means the move to a new technical environment, mostly for business reasons and for fulfilling (new) non-functional requirements. A software system is typically migrated in one or more aspects: to a different data management, communication, transaction, programming, or user interface environment (according to [Sneed et al. 2005]).

In addition, [Gimnich/Winter 2005] point out 'migration types' (dimensions) that can also apply in combination: migration of architecture, development environment, system software and hardware. The authors have sketched a sample SOA migration project, which

is – of course – an architecture migration but additionally entails introducing new tools into the development environment and new products for service monitoring, but no hardware changes.

[Channabasavaiah et al. 2004] show that SOA migration can help solve some of the application software problems we still face today: complexity, redundant and non-reusable programming, and multiple (point-to-point) interfaces. From their experience, an architecture migration into SOA is based on the following key requirements:

- **Leverage existing assets:** Existing systems (or assets from them) need to be included in the targeted SOA. They provide benefits such as suitability, efficiency, and stability. Often, this entails transforming existing assets but also, tactically, encapsulating and integrating existing assets is desirable.
- **Support all required types of integration:** Integration on user interaction (e.g. portals) level, application connectivity, information integration, process integration; potentially supported by design for integration and new component models such as SCA [SCA 2005].
- **Allow for incremental implementation and migration of assets:** Reduced migration complexity, plus the capability to produce incremental return-on-investment.

3. Project Examples

Let us consider two projects which materialize different approaches to SOA requested by their respective top-level management. Project 1 (in the finance industry) follows a top-down approach to service-modeling and selective implementation of services. Project 2 (in manufacturing) focuses on a bottom-up analysis of existing systems and the breadth of service delivery. Both projects make use of the same service modeling method, tailored to specific needs.

4. Service Modeling Support for Migration

Industrial-strength methods to model services are used by combined business and IT teams and provide important quality aspects in SOA implementation [Gimnich 2006]. Our method combines a generally top-down driven SOA design with bottom-up elements (application portfolio assessment and program understanding techniques).

In this approach, the business domain for SOA migration is analyzed using Component Business Modeling (CBM) or other methods for defining the business competencies in non-overlapping, largely stand-alone Business Components, while documenting their support to the business processes.

These business components provide a good starting point for designing services: each component has a set of consumed and offered services that are provided by other business component. Now the processes that use these business services can be decomposed, and the service candidates are refined. A second technique (goal-service modeling) provides further top-down and also middle-out support: looking at the business goals, sub-goals and key performance indicators to be achieved by the intended services. A third technique produces service candidates by **analyzing existing assets** (e.g. programs, APIs, transactions) already used in the domain. These three techniques are combined and complete the service identification. After this, service exposure decisions are made, and the resulting services then go into the specification and **realization decisions** phases. The latter phase includes analyses on whether a particular service should be based on existing assets – potentially by wrappering and **integrating** a legacy function or by **transforming** legacy –, or purchased on the market, or custom-developed.

The SOA design method used is called SOMA (Service Oriented Modeling and Architecture) [Arsanjani 2004].

5. Migration Planning and SOA Governance

Within each company, a specific SOA Roadmap needs to be defined, including all sub-projects to be performed in the SOA transition. A governance framework is established for the roles & responsibilities (e.g. design authority) involved in the transition, including the various migration tasks.

Wile Project 2 tends to have a stronger starting point for the technical migration tasks than Project 1, the business goals and the organizational aspects of the migration must also be met. So the method use is extended for business alignment and completeness 'checks' in the overall approach.

6. SOA Migration Tooling

The above-mentioned methods, in particular the CBM and SOMA approaches have to be supported as widely as possible in the development environment. One example implementation is the SOA Integration Framework (SOA-IF) used by IBM in a number of projects.

For legacy inclusion, support is required at least for analyzing existing assets (e.g. by interfacing the WebSphere Studio Asset Analyzer). Support in the legacy transformation decisions and actual transformation can be achieved using dedicated tools (e.g. Asset Transformation Workbench).

References

[Arsanjani 2004] A. Arsanjani: Service-Oriented Modeling and Architecture. http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/, November 2004.

[Channabasavaiah et al. 2004] K. Channabasavaiah, K. Holley, E. M. Tuggle: Migrating to a Service-Oriented Architecture. White paper, G224-7298, IBM, 2004.

[Gimnich 2006] R. Gimnich: Quality Management Aspects in SOA Building and Operating. SQM 2006, http://www.sqs-conferences.com/, May 2006.

[Gimnich/Winter 2005] R. Gimnich, A. Winter: Workflows der Software-Migration. WSR 2005, Softwaretechnik-Trends 25:2, May 2005. Presentation: http://www.uni-koblenz.de/sre/Conferences/WSR/Wsr2005/Programm/gimnichwinter.pdf

[SCA 2005] Service Component Architecture. White paper V0.9, BEA/IBM/Interface21/IONA/Oracle/SAP/Siebel/Sybase, November 2005, http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-sca/SCA_White_Paper1_09.pdf

[Sneed et al. 2005] H. Sneed, M. Hasitschka, M.-T. Teichmann: Software-Produktmanagement. dpunkt, Heidelberg, 2005.

Modellgetriebene Transformation von Legacy Business-Software

Matthias Kühnemann
Fakultät für Informatik
Technische Universität Chemnitz
kumat@informatik.tu–chemnitz.de

Gudula Rünger
Fakultät für Informatik
Technische Universität Chemnitz
ruenger@informatik.tu–chemnitz.de

Zusammenfassung

In zahlreichen Unternehmen befinden sich Business-Softwaresysteme im Einsatz, deren Programmarchitektur und Softwareinfrastrukur den heutigen Anforderungen, die an die Sicherheit, Leistungsfähigkeit, Adaptierbarkeit sowie an die Kosten und Verfügbarkeit moderner Zielplattformen gestellt werden, oftmals nicht mehr gewachsen sind. Das Forschungsprojekt TransBS[1] beschäftigt sich mit der Realisierung eines modellgetriebenen Transformationswerkzeuges, das Softwareentwickler bei der Transformation erprobter, monolithischer Business-Software in eine moderne, verteilte und workflowbasierte Client-Server Architektur unterstützen soll. Das Werkzeug soll für eine inkrementelle, parametrisierbare und adaptierbare Überführung verschiedene Zwischendarstellungen der Ausgangs- und Zielsoftware auf der Basis der modellgetriebenen Softwareentwicklung nutzen.

1 Einführung

Eine monolithische Business-Software repräsentiert oftmals typische Aspekte einer Legacy-Problematik, d.h. ein über Jahre gewachsenes Softwaresystem, welches beständig durch Expertenwissen erweitert und adaptiert wurde [2]. Die Transformation der monolithischen Ausgangssoftware in eine prototypische verteilte Referenzimplementierung kann dabei in inkrementellen, transparenten und konfigurierbaren Transformationsschritten erfolgen. Die für diesen Ansatz hilfreiche Infrastruktur kann auf der *Modellgetriebenen Softwareentwicklung* (MDA - Model Driven Architecture) [4] basieren. Die MDA ermöglicht das Speichern von Zwischendarstellungen der Ausgangs- und Zielsoftware in Form von plattformunabhängigen und plattformspezifischen Softwaremodellen. MDA-Werkzeuge erlauben die automatisierte und interaktive Generierung dieser Softwaremodelle und der damit assoziierten und kompilierbaren Quelltextstrukturen.

Als exemplarische Ausgangssoftware betrachten wir das monolithische Softwaresystem GBware[2], das sich in einer Vielzahl von Unternehmen erfolgreich im Einsatz befindet. Ein Ziel des Forschungsprojektes TransBS ist eine prototypische Überführung des GBware Legacy-Programmcodes in ein verteiltes Softwaresystem GBwareD (GBware distributed) im Client-Server-Stil. GBwareD soll zu diesem Zweck ein offenes und erweiterbares Client-Server-Framework CBFrame für Business-Software nutzen können [5]. CBFrame soll modular aufgebaut sein und über ein Kommunikationsinterface verfügen, in dem Teile existierender Business-Softwaresysteme als Komponenten integriert werden können. Das Framework wird dabei unterschiedliche Aspekte der Verteiltheit, Sicherheit und Performance abdecken. Für die Überführung der GBware Module in GBwareD Komponenten, die in einem Client-Server-System verteilt ausgeführt werden können, soll ein Transformationswerkzeug (im folgenden TransBS genannt) entwickelt werden. Das funktions-, korrektheits- und merkmalserhaltene Transformationswerkzeug soll eine inkrementelle Überführung unterstützen sowie geeignete Beschreibungs- und Spezifikationssprachen für die Modulinteraktion bereitstellen.

2 Modellgetriebene Transformation

Die OMG (Object Management Group) will mit MDA einen Standard für eine modellgetriebene Softwareentwicklung schaffen. MDA-Werkzeuge unterstützen einen Softwareentwickler bei der Transformation eines plattformunabhängigen Softwaremodells (PIM) in ein plattformabhängiges Modell (PSM) und in einen kompilierbaren Quelltext [1]. Ein inkrementeller und konfigurierbarer Transformationsansatz kann durch die Überführung eines leicht adaptierbaren und erweiterbaren PIMs in potentiell unterschiedliche PSMs ermöglicht werden, wobei das erzeugte PSM die Zielarchitektur festlegt. Die OMG favorisiert UML (Unified Modeling Language) weitestgehend für die Erstellung eines PIMs. Oftmals wird zur Modelldarstellung eine höhere, domänenspezifische Modellierungssprache (DSL - Domain Specific Language) verwendet, deren Definition man oftmals durch ein UML-

[1]Das Projekt TransBS - Transformation monolithischer Business-Softwaresysteme in verteilte, workbasierte Client-Server-Architekturen wird seit 2006 durch das Bundesministerium für Bildung und Forschung (BMBF) unter dem Förderkennzeichen 01 IS F10 gefördert. Weitere Informationen sind unter www.transbs.de zu finden. Das Projektkonsortium setzt sich aus der Berndt & Brungs Software GmbH, der Universität Bayreuth und der Technischen Universität Chemnitz zusammen.

[2]GBware© (General Businessware), Berndt & Brungs Software GmbH

Profil erhält, das an eine Business-Domäne angepasst wurde. Es können neue UML-Profile definiert oder vorgefertigte Profile (z.B. J2EE/EJB Profile Example) verwendet werden. MDA-Werkzeuge unterstützen die Transformation in verschiedene Zielarchitekturen (z.B. CORBA, J2EE, .NET) und unterschiedliche Zielsprachen (z.B. Java, C++). Nur durch die Erweiterung eines UML-Klassendiagramms um zusätzliche Informationen ist die Erzeugung eines kombilierbaren Quellcodes überhaupt möglich. Zu diesen Informationen zählen Stereotypen, Eigenschaftswerte (tagged values) und Bedingungen (constraints). Soll beispielsweise eine beliebige Klasse als Datenbankklasse abgespeichert werden, so beschreibt ein Stereotyp die Datenbankklasse, der Primärschlüssel ist ein Eigenschaftswert, und die Bedingung legt einen Wertebereich fest. In einem nächsten Schritt kann das PSM in kompilierbaren Quelltext übersetzt werden, wobei die Geschäftslogik und Prozessalgorithmen der Business-Software oftmals in Form von Templates (die Platzhalter enthalten) implementiert werden. Neben templateorientierten Transformationsansätzen werden auch patternorientierte und metamodellorientierte Transformationsansätze für die Generierung von Quellcode eingesetzt.

Die MDA bietet bisher keine Unterstützung zur Behandlung von Legacy-Software. Dieser Ansatz soll im Forschungsprojekt TransBS untersucht werden. Ein allgemeiner Ansatz für eine solche Überführung besteht in der Analyse des Legacy-Programmcodes, um eine (gedankliche) Umstrukturierung einer Referenzimplementierung in 3 getrennte Codebereiche vorzunehmen: *Individueller Code*, *Generischer Code* und *Schematischer, sich wiederholender Code*. Individueller Code bezeichnet Programmcode, der für jede spezielle Applikation neu implementiert werden muss [6]. Generischer Code kann unabhängig von der speziellen Applikation wieder verwendet werden. Der Transformationsansatz aus einem Softwaremodell in einen verteilten, workflowbasierten Programmcode eignet sich insbesondere für den schematisch, sich wiederholenden Codebereich.

Die Frage, wie die Analyse des Legacy-Programmcodes durchgeführt werden soll, um ein passendes PIM zu generieren, ist noch offen. In Betracht kommen sowohl manuelle Analyseverfahren (Programmierer, Dokumentation, bestehende Programmstruktur), als auch automatisierte, compilerbasierter Techniken (Parser, Code-to-Code, Source-Code-Graph für Darstellung der Modulabhängigkeiten).

3 Anforderungsspezifikation TransBS

Ein erster Entwurf für die Realisierung eines Transformationswerkzeuges TransBS auf der Basis der MDA könnte wie folgt aussehen. Das Werkzeug TransBS verwendet ein PIM und ein (oder mehrere) PSM(s) zur internen Darstellung der verteilten Codestruktur der Zielsoftware GBwareD. Das PIM erhält man aus dem Legacy-Programmcode GBware durch verschiedene (Analyse-)Ansätze in einem ersten Schritt (*code-to-model Transformation*). Dazu zählen Reengineering, das Studium der GBware-Dokumentation, sowie compilerbasierte und manuelle Analyseverfahren. Konfigurationen und Adaptionen der Zielsoftware können in einem zweiten Schritt bei der Transformation des PIMs in ein PSM gegebenenfalls interaktiv durch den TransBS Anwender vorgenommen werden (*model-to-model Transformation*) [3]. Dies schließt inkrementelle, transparente und interaktive Transformationsschritte nicht aus, da zum einen Transformationen in immer konkretere Modelle (sowohl PIMs als auch PSMs) durch die MDA-Idee angeregt werden und zum anderen auch mehrere verschiedene PSMs durch den TransBS Anwender interaktiv erzeugt werden können. Der dritte und letzte Schritt erzeugt den kompilierbaren Quellcode der verteilten Zielsoftware GBwareD durch die Überführung des PSM in Programmcode (*model-to-code Transformation*). Ermöglicht wird diese Überführung durch Abbildungen und Transformationsregeln von PSM-Modellelemente auf Klassenelemente des Frameworks CBFrame. Die Referenzimplementierung von CBFrame ist hierfür in einer Zielarchitektur realisiert und kann auf einer ausgewählten Zielplattform prototypisch ausgeführt werden. Als Zielarchitekturen für eine verteilte Implementierung des Client-Server-Frameworks kommen EJB (Enterprise Java Beans) und CORBA (Component Object Request Broker Architecture) in Betracht.

Literatur

[1] A. Andresen. *Komponentenbasierte Softwareentwicklung.* Hanser Verlag, 2004.

[2] M.C. Feathers. *Working Effectively with Legacy Code.* Prentice Hall, 2004.

[3] V. Gruhn, D. Pieper, and C. Röttgers. *MDA. Effektives Softwareengineering mit UML2 und Eclipse.* Springer, 2006.

[4] MDA Model Driven Architecture, http://www.omg.org/mda.

[5] G. Muller, J.L. Lawall, J.M. Menaud, and M. Südholt. Constructing component-based extension interfaces in legacy system code. In *Proc. of the 11th workshop on ACM SIGOPS European workshop.* AMC Press, 2004.

[6] T. Stahl and M. Völter. *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management.* dpunkt Verlag, 2005.

Migration der UBS-Kernapplikationen von Unisys nach z/OS best practice

Guido Salvi
Hal Knowledge Solutions AG
P.O. Box 1554
CH-8301 Glattzentrum Wallisellen
Guido.Salvi@halks.com

Wer sind wir?

HAL wurde 1984 gegründet. HAL ist heute marktführender Anbieter von Application Portfolio Solutions und etablierter Anbieter von Migration und Re-engeneering Dienstleistungen. HAL Lösungen werden direkt oder in Zusammenarbeit mit Global Service Partners wie IBM und EDS an führende Unternehmen der Welt ausgeliefert. Dazu gehören Unternehmen wie UBS, Banca Intesa, Linea Directa, Alitalia, TNT, Kühne&Nagel, Telefonica sowie Renault Nissan. HAL ermöglicht den Kunden, Applikations-Supportkosten bis zu 30% einzusparen, strategische Sourcing Entscheidungen zu treffen und sowohl externe als auch interne Teams zu verwalten, sowie „Regulatory Compliance" Anforderungen gerecht zu werden. Weitere Informationen zu HAL finden Sie unter www.halks.com.

Unsere Lösungen

Mit dem Produkt *Kb-AIM Data*© bietet HAL eine Lösung zur Datenstrukturanalyse und zur Generierung von Testdaten-Umgebungen. *Kb-AIM Data*© unterstützt die Unternehmen in der Erzeugung von wiederholbaren und zuverlässigen Tests. Die Unternehmen können automatisch eine repräsentative Teilmenge aus der Applikationsinfrastruktur auswählen und die Daten zusätzlich anonymisieren (Data Masking), um so die Anonymität in der Testumgebung zu gewährleisten.

Mit *Kb-AIM Code*© bietet HAL eine Application Portfolio Management Lösung an, welche den Unternehmen Transparenz über IT Applikationen bietet, welche die Grundlage zu wichtigen Geschäftsprozessen sowie ein wesentlicher Bestandteil unternehmerischer Entscheidungsprozesse sind.

Die technischen Analysen basierend auf *Kb-AIM Code*© liefert auch die Grundlagen für die von HAL angebotenen Applikationsmigrations und Reengineering Dienstleistungen. HAL hat in den letzten Jahren über 70 erfolgreiche und hoch automatisierte Applikationsmigrations- und Reengineering-Projekte durchgeführt. Zur Automation werden speziell angepasste Werkzeuge eingesetzt, welche eine schnelle Migration, hohe Qualität, geringes Risiko, minimalen Testaufwand und minimale Freeze-Phase für die Applikationswartung garantieren. HAL hat nachweisbare Kompetenzen für folgende Applikationsmigrations und Reengineering-Initiativen:

- Code Migrationen
  IMS/DC → CICS
  Natural → COBOL oder Java
  Mantis → Cobol
  PL1 → Cobol
  Visual Basic → Java
- Daten Migrationen
  IMS/DB → DB2
  Adabas → DB2
  Adabas → Oracle
  VSAM → Oracle oder DB2
- Plattformwechsel und Downsizing
  Unisys OS2200 mit Cobol/DMS → z/OS mit Cobol und DB2
  IBM z/OS/Cobol/"irgendeine DB" → IBM iSeries/Cobol/DB400
  IBM VSE/Cobol/"irgendeine DB" → IBM iSeries/Cobol/DB400

UBS: Grösste IT-Migration Europas

UBS AG hat mit dem Partner Hal Knowledge Solutions ihr integriertes Gesamtbankensystem „Abacus" komplett migriert.

In der Rekordzeit von 12 Monaten wurden sämtliche Abwicklungsapplikationen von der

abzulösenden Unisys-basierten Plattform OS2200 auf die neue IBM-Plattform z/OS gebracht. Der Einsatz von Methoden und Werkzeugen der Hal Knowledge Solutions war dabei ein wesentlicher Faktor. Sie ermöglichten bei der Konvertierung von Programmen einen Automatisierungsgrad von über 98%.

UBS war sich bewusst, dass es sich um ein äusserst anspruchsvolles Projekt handelte. Der normale Betrieb musste jederzeit und mit laufenden Anpassungen sichergestellt sein. Denn auf der abzulösenden Plattform liefen sämtliche, für die täglichen Geschäfts-abwicklungen nötigen Funktionalitäten, wie sie für eine Grossbank unverzichtbar sind. Dazu gehören unter anderem Kundendaten, die Abwicklung von Geschäftsfällen, der Zahlungsverkehr, der Wertpapierhandel, das Electronic Banking und vieles mehr. Dies berührt täglich hunderttausende von Kunden und einen Zahlungsverkehr in Milliarden Höhe.

Für den Wechsel mussten 2‘000 Online- und 5‘000 Batch-Programme sowie 3‘000 Datenbank-Objekte, 10‘000 Dateien, über 300‘000 Programmabläufe - Jobs - und knapp 40‘000 Batchnetze migriert werden.

Um ein solches Projekt erfolgreich durchzuführen, sind nebst höchster Professionalität in Management, Technik und Komplexität sowie Top-Einsatz, einige weitere Eigenschaften unabdingbar für den Erfolg: Ein hervorragendes Team, das Wissen um absolutes Vertrauen und Verlass auf Commitment und Fähigkeiten jedes Spezialisten, und nicht zuletzt manchmal auch der Mut zu unkonventionellen Lösungen. Für den Erfolg unabdingbar ist, dass die Geschäftsleitung solch ein Projekt voll unterstützt.

Aus der Sicht von HAL war entscheidend, dass UBS einen Projektleiter stellte, welcher diese Voraussetzungen zu schaffen und diese Eigenschaften permanent zu mobilisieren vermochte.

UBS und Hal stellten ein konsequent durchgeführtes Qualitäts- und Risiko-Management sicher. Das Single Source Management (es gibt immer nur eine Quelle) war ein wesentliches Element und konnte trotz nötiger Änderungen am laufenden System dauernd aufrecht erhalten werden.

Auf die Risiken angesprochen, erklärt UBS, dass es natürlich in jedem Projekt Risiken gibt. Auch in diesem grossen Projekt waren der UBS die Risiken bekannt. UBS und HAL haben sie systematisch, emotionslos und professionell gemanagt. So war es denn auch möglich, das Projekt im Dezember 2004 erfolgreich abzuschliessen und unmittelbar danach die Jahresendverarbeitung durchzuführen, die übrigens ebenfalls sehr erfolgreich verlief.
Aufgrund aller Testberichte und Parallelläufe in der Produktion, waren für die UBS die Risiken jederzeit berechenbar und konnten entsprechend gehandhabt werden. Ohne diese Gewissheit wäre die Migration nicht so kurz vor dem Jahresende und nicht an drei Wochenenden möglich gewesen.

Besonders herausfordernd waren aus Sicht der Projektleitung die laufend nötigen Programmanpassungen und Erweiterungen in Abacus während der Migrationszeit, die Umstellung von ASCII auf EBCDIC, der Wechsel von einem hierarchischen auf ein relationales Datenbanksystem, das korrekte Tuning der Systeme, die rechtzeitige Bereitstellung aller Middleware-Komponenten und aller Batch-Jobs sowie eine hohe Verfügbarkeit der Schlüsselpersonen.

Die Migration selbst war mit Blick auf die langfristige IT-Systemausrichtung und -erneuerung in der UBS wohl von grosser Wichtigkeit, ist aber als eine Phase des gesamten Strategic Solution Programs (SSP) zu betrachten. Bereits vor einigen Jahren hat die UBS das SSP etabliert, die umfassende Erneuerung der Informatiksysteme, um sich auf die Anforderungen der Kunden und Märkte für die Zukunft einzurichten. Die Dynamik in der Finanzdienstleistungsbranche erfordert flexiblere Systeme und Abläufe als in der Vergangenheit, eine schnelle Anpassung an die Kunden- und Marktbedürfnisse und die rasche Bereitstellung neuer Lösungen.

Über UBS AG
UBS ist eines der führenden globalen Finanzinstitute und einer der weltweit grössten Anbieter im Wealth-Management-Geschäft. UBS gehört zu den wichtigsten Investmentbanken und Wertpapierhäusern und zählt zu den führenden Vermögensverwaltern weltweit. Im Privat- und Firmenkunden-geschäft in der Schweiz ist UBS Markt-führerin.

Kritische Erfolgsfaktoren beim Abnahmetest in Redevelopment-Projekten

Erfahrungen aus einem Großprojekt

Jens Borchers
Senior Manager Banking
Steria Mummert Consulting AG
Hans-Henny-Jahnn_Weg 29
22085 Hamburg
jens.borchers@steria-mummert.de
jensborchers@acm.org

Abstract

Redevelopment stellt eine besondere Form des Reengineering dar. Dabei werden mit einer Kombination aus Reverse und Forward Engineering-Methoden die wesentlichen Funktionen eines bestehenden Systems – im vorliegenden Fall auf einer völlig neuen Plattform – erneut entwickelt. Im Rahmen der Entwicklung und Produktionsvorbereitung spielen Abnahmetests als letzte fachliche Teststufe eine wesentliche Rolle. Um diese erfolgreich, d.h. sicher und wirtschaftlich durchführen zu können, müssen einige Faktoren beachtet werden. Diese werden im nachfolgenden dargestellt.

1. Besonderheiten von Redevelopment-Projekten

In vielen Unternehmen basieren wesentliche Kernsysteme noch immer auf Plattformen und auf Entwicklungssprachen, die nicht mehr zeitgemäß sind. Im konkreten Fall ist dieses ein Hostsystem, auf dem ein im Kern über 20 Jahre altes, zu fast 100% in Assembler geschriebenes System die wesentlichen Services abbildet. Die darin abgebildeten Funktionen sind aber für das Unternehmen von zentraler Bedeutung und müssen dafür für die Zukunft neu ausgerichtet werden.

Bekanntlich gibt es für die Evolution von Softwaresystemen mehrere Alternativen (vgl. [Bo02], nämlich in diesem Fall

- Migration auf rein technischer Ebene, z.B. Sprachumstellung auf aktuelle Sprache
- Ablösung durch Standardsoftware, die es aber für das abzulösende System nicht gibt
- komplette Neuentwicklung auf Basis neu erstellter Spezifikationen

Nachdem einige Versuche mit einer direkten Sprachmigration sich als langfristig nicht tragfähig erwiesen hatten, wurde die Entscheidung für eine komplette Neuentwicklung in einer J2EE-Architektur getroffen.

Dabei sollten die bestehenden Funktionen weitestmöglich 1:1 erneut entwickelt werden. Es handelt sich also um einen klassischen Fall von „Redevelopment“. Bei diesem Redevelopment handelt es sich also um eine Anwendung von Reverse Engineering (vgl. zur Begriffsbildung [Ba96]) und aktuellen Methoden des Forward Engineering.

Wie bei alten Systemen üblich, konnte auch hier auf keine vollständige und detaillierte Dokumentation zurückgegriffen werden. Daher wurden alle Funktionen völlig neu spezifiziert, z.T. unter Nutzung von UML. Die Vorgaben dazu wurden aus einer Kombination von Dokumentation, Experten-Know-how und im Extremfall durch Ausprobieren am aktuellen System erarbeitet.

Ziel war es, das derzeitige System - in neuer Technologie - in seinen wesentlichen Funktionen nachzubauen. In einzelnen Bereichen wurden aber von vornherein Verbesserungen eingeplant, so dass das resultierende System das bestehende zu ca. 90% identisch ersetzt und sich im Rest unterscheiden kann.

Aufgrund des Redevelopment-Ansatzes konnten die sonst bei Reengineering-Projekten üblichen rigiden Mechanismen [Bo97] bei der eigentlichen Überführung der Funktionen nicht angewendet werden, es war allerdings wie dort ein entsprechend umfangreicher Test zum Nachweis der korrekten Überführung der Funktionalität erforderlich, da es sich auch um eine besonders kritische Anwendung mit Außenwirkung auf viele andere Unternehmen handelt.

2. Teststrategie und Testdurchführung

2.1 Globale Teststrategie

Im Rahmen der Entwicklung durchlaufen zunächst alle üblichen Softwarekomponenten die üblichen Teststufen der Entwicklung, also insbesondere Unit- und Integrationstest. Diese werden im Rahmen der normalen Entwicklungsorganisation durchgeführt.

Die letzte Stufe vor der Prüfung der betrieblichen Aspekte (Performance, Durchsatz, Stabilität etc.) stellt der fachliche Abnahmetest dar, der im vorliegenden Fall von einer getrennten Organisationseinheit und in der technischen Abwicklung von einem Outsourcer durchgeführt wurde.

Durch diese rigide Trennung wurde sichergestellt, dass insbesondere durch die dadurch erforderlichen Liefermechanismen mit entsprechenden Qualitäts-Gateways immer ein absolut definierter Software-Release-Level getestet wurde.

Die fachlichen Tests, d.h. die Spezifikation der eigentlichen Testfälle wurde durch eine eigenständige Organisationseinheit vorgenommen. Die Mitarbeiter dieser Abteilung sind auch an den Spezifikationen der Software beteiligt gewesen und haben die Testfalldefinitionen entlang den Softwarespezifikation aufgebaut.

Durch diese Abteilung wird letztlich auch die Korrektheit der Software bestätigt, bevor sie in Betrieb genommen werden kann.

2.2 Testziele und entsprechende Nachweise

Wie bereits oben angemerkt, handelt es sich um das Redevelopment eines bestehenden und „lebenden" Systems.

Getestet wird dabei gegen also gegen zwei parallele – im Einzelfall ggf. auch konträre – Ziele:

- „klassisch" gegen die fachliche Spezifikation der neuen Software
- Regression gegen das Verhalten des bestehenden Systems, da dieses für 90% der Funktionen wieder identisch vorhanden sein muss.

Es war also ein Abnahmetest aufzusetzen, der beiden Zielen gerecht wird und dabei weitestgehend ausnutzt, dass mit dem bestehenden System und dem dafür bekannten Verhalten eine sichere Referenz vorlag, gegen die im Einzelfall auch noch die Korrektheit der neuen Spezifikationen selbst zu prüfen war.

2.3 Testdurchführung

Die Tests wurden auf einer getrennten Hardware- und Systemumgebung durchgeführt. Dabei konnten im Extremfall drei verschiedene Softwareversionen mit jeweils unterschiedlichen Datenbeständen getestet werden.

Zusätzlich stand eine Host-Testumgebung zur Verfügung, auf der die Referenztests (genauer: Herstellung von Referenztestergebnissen) durchgeführt wurden.

Alle fachlichen Testfälle wurden von der fachlichen Abnahmeabteilung in Excel-Dateien mit einem festen Format abgelegt. Dieses Format spiegelt die klassische Aufteilung

- Zustand der relevanten Datenobjekte vor dem Testfall
- Testfall („Geschäftsvorfall") selbst
- Erwartetes Ergebnis des Testfalls
- Erwarteter Datenzustand der relevanten Objekte nach dem Testfall

strukturiert.

Aus diesen fachlichen Vorgaben wurden alle technischen Testläufe automatisch generiert, d.h. manuelle Tests sind nur für einen kleineren Teil der GUI-Testfälle erforderlich gewesen.

Die zugehörigen Datenarchive wurden vor und nach jedem Testlauf, der zwischen wenigen 100 und mehreren 1000 Testfälle enthalten konnte, gesichert und in entsprechend verwalteten Archivstrukturen abgelegt.

Dabei wurden alle fachlichen Testfälle zunächst gegen das bestehende Hostsystem durchgeführt, um eine erste Referenz zu erhalten. In späteren Stufen wurden dann Releases der neuen Software zur neuen Testbaseline gemacht.

3. Kritische Erfolgsfaktoren

Wie oben beschrieben, stellt der Test in einem Redevelopment-Projekt eine gemischte Form eines klassischen Reengineering-Tests und einem Neuentwicklungstest dar. Dabei muss man die Vorteile, die man durch Vorliegen eines realen Systems, dessen Verhalten ein wesentlicher Teil der Vorgabe ist, so weit wie möglich nutzen, um zu einer ersten regressionsfähigen Testergebnisbasis zu kommen.

Als wesentliche Erfolgsfaktoren haben sich in diesem Projekt herausgestellt:

- Die Verfügbarkeit absolut kontrollierter „Reinraum"-Testumgebungen, auf die Entwickler – schon technisch verhindert – keinerlei Zugriff haben.
  - Testumgebungen, gerade auf Host-, aber auch in J2EE-Umgebungen stellen einen potentiellen Ressourcenengpass dar, der gravierende Auswirkungen auf den Testfortschritt haben kann
- Die weitestgehende Automatisierung aller Testdurchführungen ist anzustreben. Dazu gehören:
  - alle Lade- und Entladeprozesse für Datenarchive
  - das eigentliche Durchführen von Testläufen, im GUI-Bereichen mit entsprechenden Tools wie z.B. WinRunner, SilkTest
  - der automatisierte Vergleich aller Testergebnisse, damit nur noch diejenigen einer fachlichen Prüfung zugeführt werden müssen, die sich gegenüber der aktuellen Baseline verändert haben (oder sich in Fehlerfällen verändern mussten!)
  - entsprechendes automatisches Reporting über Testdurchführungen, Ergebnisstatistiken
- Eine enge Kopplung mit einem Fehlertracking-System stellt sicher, dass zu jedem Abweichungsreport ein entsprechender Testfall referenziert werden kann. Dazu muss es möglich sein, den Testfällen eine eindeutige Identifikation zu geben.
- Das Konfigurationsmanagement bleibt einer der kritischsten Punkte. Trotz aller Tools kommt es immer wieder zu Fehlersituationen, die allein auf eine falsche Version von Teilkomponenten zurückzuführen sind und in der Konsequenz ganze Release-Level obsolet machen.

- Die gesamte Testmaschinerie muss mit großer Geschwindigkeit betreibbar sein, da gerade in den Phasen vor dem eigentlichen Produktions-Release die neu zu testenden Releases in immer schnelleren Folgen vorgelegt werden.
- Und last, but not least muss es ein fachlich und methodisch versiertes Testteam geben, das in der Lage ist, Testfälle so zu erstellen und zu bewerten, dass mit größtmöglicher Wahrscheinlichkeit sichergestellt wird, dass genau das Verhalten erreicht wurde, welches sich einerseits aus dem bestehenden System und den definierten Abweichungen davon ableitet.

4. Fazit

Tests in Redevelopment-Projekten unterliegen natürlich zunächst den gleichen Anforderungen wie die in Reengineerring-Projekten, allerdings „verschärft“ durch die Existenz neuer fachlicher Spezifikationen und den bewusst eingeführten Abweichungen.

Nur ein rigides, automatisiertes Vorgehen kann dabei letztlich zum Erfolg führen. Der Aufbau einer entsprechenden Organisation und Infrastruktur stellt dabei – wie bekannt – den wesentlichen Erfolgsfaktor dar. Wenn entsprechende Ressourcen, sowohl technisch als organisatorisch, nicht zur Verfügung gestellt werden, sind alle hehren Ansätze zum Scheitern verurteilt. Glücklicherweise ist dieses im vorliegenden Fall vom Management erkannt worden. Damit konnte eine hohe Sicherheit in der Wiederherstellung der bestehenden Anwendung erreicht werden.

5. Literaturverzeichnis

[Ba96] U. Baumöl et.al., Einordnung und Terminologie des Software Reengineering, Informatik Spektrum, 19 (1996), S. 191-195

[Bo97] J. Borchers, Erfahrungen mit dem Einsatz einer Reengineering Factory in einem großen Umstellungsprojekt, HMD Nr. 194, März 1997

[Bo02] J. Borchers, Software Evolution Enabling - IT-Zukunftssicherung auf Basis bestehender Systeme, 4 . Workshop Software Reengineering, Bad Honnef, 29.-30. April 2006

Lösungsmuster in der Planung industrieller Migrationsprojekte

Jakob Boos, Dr. Markus Voß, Johannes Willkomm, Dr. Andreas Zamperoni

sd&m Research
sd&m AG software design & management
Berliner Str. 76, 63065 Offenbach
markus.voss@sdm-research.de

1. Einleitung

Die sd&m AG hat im letzten Jahr ihre Erfahrung aus dutzenden von Migrations-Projekten in ein Standard-Vorgehen zur Planung solcher Vorhaben gefasst. In [1] haben wir darüber bereits berichtet. Kern dieses Standard-Vorgehens ist eine systematische und auf klaren Konzepten beruhende Elimination von Freiheitsgraden der Lösung zur schrittweisen Ableitung von Ziel-Architektur und Ziel-Migrationsszenario. Der Gesamtprozess ist in Abbildung 1 illustriert.

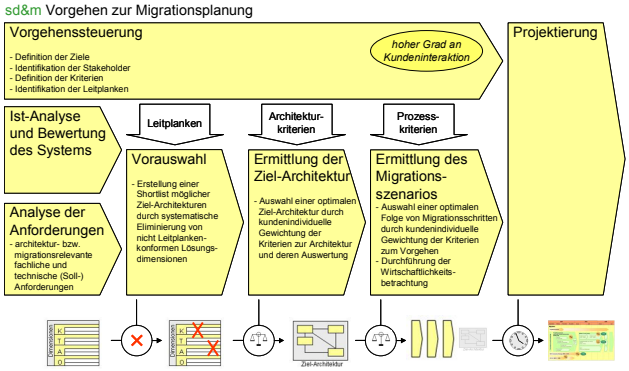


Abbildung 1: Migrationsplanung

In diesem Beitrag werden einige Best Practices bzw. Lessons Learned mit Mustercharakter zu unterschiedlichen Teilbereichen des Vorgehensmodells dargestellt. Konkret handelt es sich um die Teilbereiche der Festlegung von Leitplanken und Kriterien im Rahmen der Vorgehenssteuerung, der Ist-Analyse und Bewertung des Systems sowie der Ermittlung der Zielarchitektur. Diese entstammen jeweils unterschiedlichen industriellen Migrationsvorhaben, die sd&m in jüngster Zeit durchgeführt hat.

2. Großes Migrationsprojekt im Public Sector

Das zu migrierende System ist über 20 Jahre gewachsen. Es enthält 2 Datenbanken mit insgesamt 25 Mio. personenbezogenen Datensätzen, etwa 20–30 Anwendungen, die auf diese Datenbanken zugreifen und mehrere Tausend Benutzer in mehreren Hundert Behörden, die über verschiedene Infrastruktur (Host-Dialog bis Portal) auf das System zugreifen. Technologisch ist es außerordentlich heterogen – es umfasst sowohl große Anteile mit Host, Cobol und Natural, Anteile als Client/Server-Lösungen in C/C++ und Java sowie aktuelle Lösungsanteile auf Basis von J2EE mit einem Portal. Die Menge der am Projekt Beteiligten ist groß (2 Fachbereiche, 1 IT-Anwendungsentwicklung, 2 IT-Betrieb, 1 Stabsstelle, 1 Sicherheitsmanagement und 3 externe Dienstleister). Die Aufgabe von sd&m besteht in der Übernahme der Wartung des bestehenden Systems, der Konzeption einer service-orientierten Zielarchitektur, der Erstellung einer Migrationsstudie und –planung und der Durchführung der eigentlichen Migrationsstufen. Das Gesamtvolumen beträgt über 150 Bearbeiterjahre.

In einem derart großen und komplexen Migrationsprojekt ist bereits die Definition der Leitplanken und der Bewertungskriterien eine Kunst. Die Projekterfahrung zeigt in diesem Fall vor allem, dass eine stringente Anwendung des Frameworks aus dem Standard-Vorgehen bereits echten Mehrwert stiftet.

Im Projekt wurden etwa 20 Leitplanken explizit definiert, darunter z.B. die Einhaltung des behördenweiten IT-Rahmenkonzepts und des IT-Grundschutzhandbuchs des BSI, die Verwendung von J2EE und der Quasar-Architektur [2], die Vermeidung dynamischer Webtechnologien, Linux als Server-Plattform etc.

Lessons learned im Bereich der Definition der Leitplanken sind z.B. die folgenden:

- Leitplanken geeignet stecken: Der Zielkorridor darf weder zu schmal noch zu breit sein. Dann bleibt ein sinnvoller (Merkmals-)Raum von Kriterien übrig.
- Gute Leitplanken eliminieren ganze „Kriterienzweige“: Wo Leitplanken definiert wurden, ist keine Aufnahme von Bewertungskriterien mehr nötig.
- Change Management einführen: Ergebnisse nicht auf Zuruf verändern - weil viele Beteiligte über einen längeren Zeitraum den Überblick behalten und Entscheidungen auch nach längerer Zeit nachvollziehbar sein müssen.
- Migrationsdimensionen (Architektur, Organisation, Technologie) nutzen: Sie helfen bei der Einordnung, wie und wo identifizierte Leitplanken wirken.
- Managementbeteiligung sicherstellen: Leitplanken sind oft auch nicht-technisch und der „operativen“ Ebene völlig unbekannt
- Workshop mit allen Interessensvertretern durchführen: Unsicherheiten auf Kundenseite zeigen sich schnell. Auf Widerspruchsfreiheit (innerhalb einer Mitarbeiterebene und zwischen den Mitarbeiterebenen) ist zu achten. Das geht nur, wenn alle Stakeholder aktiv beteiligt sind.

Bewertungskriterien wurden im Projekt ca. 100 erhoben. Diese wurden in ca. 8 Hauptgruppen und ca. 20 Gruppen unterteilt.

Lessons learned im Bereich der Definition der Bewertungskriterien sind z.B. die folgenden:

- Hierarchische Gruppierung: Sie hilft bei der Bewertung. Dabei sollte unbedingt Top-down vorgegangen werden. Techies neigen zu Depth-First und müssen „eingefangen“ werden.
- Kriterien für alle Dimensionen der Migration betrachtet, dann aber auf das Wesentliche beschränken.
- Mit erstem Entwurf in den ersten Kundenworkshop gehen: Sonst dauert es zu lange.
- Die Kriterien und –gruppen mit Hilfe eines Punktesystems gewichten (Gewichtungspunkte): Top-down Verteilung einer festen Anzahl von Gewichtungspunkten auf Hauptgruppen, Gruppen und Einzelkriterien.
- Für die Bewertung „Ausgangslage – Zielsituation“ ist es notwendig, die Erfüllung eines Kriteriums in „gut“, „mittel“ oder „schlecht“ möglichst genau zu erläutern. Beispiel: Kriterium „Homogenität der Programmiersprachen“: gut = eine Sprache, mittel = 2-3 Sprachen, …).
- Gewichtung von Einzelkriterien fällt den Interessensvertretern schwer. Hauptgruppen gehen gut.
- Messung des Erfolgs der Migration über „Delta der (gewichteten) Kriterienerfüllung“: Ist nicht einziger Erfolgsmesser, führt aber zur Objektivierung des „gefühlten“ Erfolges.
- „Spinnendiagramme“ für Bewertung zeichnen: Das Verbesserungspotential wird grafisch aufgezeigt.

3. SOA-Roadmap für einen Pharmahändler

Aufgabe dieses Projekts war es, eine Roadmap zur Modernisierung einer über Jahre gewachsenen Systemlandschaft zu entwickeln. Hintergrund der Modernisierung waren Aspekte der Agilität und der Wirtschaftlichkeit.
Der Schlüssel zum Erfolg war die Anwendung eines bei sd&m für solche Zwecke erprobten und bewährten Beratungsmusters, welches die Dimensionen *Fachlichkeit, IT-Architektur* und *IT-Organisation* sowie deren Wechselwirkungen untereinander gleichermaßen berücksichtigt:

- Fachlichkeit: Wie sind die im Fokus stehenden Geschäftsprozesse gestaltet und wer sind die beteiligten Personen? Welche heutigen und zukünftigen besonderen Anforderungen werden an diese Geschäftsprozesse gestellt?
- IT-Architektur: Welche Systeme unterstützen die betrachteten Geschäftsprozesse? Welche Randsysteme existieren, die in den Betrachtungs-Scope mit aufgenommen werden müssen und welche weiteren Anforderungen an die zukünftige Architektur ergeben sich aus den Prozessanforderungen?
- IT-Organisation: Wie sind die IT-Prozesse gestaltet? Wie sieht die Mitarbeiterstruktur der IT aus? Wie ist der Betrieb organisiert und auf welche Art erfolgt ein IT-Controlling?

Lessons learned im Bereich der Ist-Analyse bei der Migration von Systemlandschaften waren dabei folgende:

- Die Betrachtung einer engen systemischen Kopplung der drei Dimensionen ist Grundvoraussetzung für die Erarbeitung eines tragfähigen Lösungsszenarios. Zwischen allen drei Dimensionen bestehen enge Abhängigkeiten, deren Identifikation und Auswirkungen eine der größten Herausforderungen darstellt.
- Eine qualitativ hochwertige Analyse aus Sicht dieser drei Dimensionen bedarf zum Teil sehr stark unterschiedlicher Berater-Skills, die schnell eine gemeinsame Sprache finden müssen.
- Im Rahmen der Ist-Analyse müssen bereits Systemkategorien erhoben werden, die für das Lösungsszenario nicht relevant zu sein scheinen, die aber für ein zu entwickelndes Migrationsszenario essentiell sind. So ist beispielsweise bereits hier aufzunehmen, ob es sich um Bestandskomponenten handelt. Damit kann dann nach Entwicklung des Lösungsszenarios ein Ablösungsszenario von „außen nach innen“ wie in [3] beschrieben, entwickelt werden.

4. Migration eines Logistiksystems

Das zu migrierende System unterstützt den gesamten Geschäftsprozess für eine termingerechte Bereitstellung von Materialien und Werkzeugen zum Bau von Kraftwerksanlagen in der ganzen Welt. Betrachtet man den Gesamtprozess, so werden die ersten Prozessschritte von mehreren vorgelagerten SAP-Systemen unterstützt. Nachgelagerte Prozessschritte werden wiederum durch SAP Systeme abgebildet. Das betrachtete und abzulösende System ist unter anderem dafür verantwortlich, Lieferungen zusammenzustellen und Exportfreigaben einzuholen.
Die Aufgabe bestand darin, drei unterschiedliche Lösungsansatze zu evaluieren und eine Empfehlung auszusprechen. Hierbei war die Rahmenbedingung stets die, dass das existierende System durch ein einzelnes neues System abzulösen ist. Verglichen wurde hierbei die Implementierung einer individuellen Lösung mit dem Einsatz von zwei unterschiedlichen Fertigkomponenten inklusive der damit verbundenen Anpassungen.
Die durch sd&m empfohlene Lösung bestand schließlich darin, eine Zentralisierung von übergreifender Prozesslogik vorzunehmen und dann in einem zweiten Schritt zu überprüfen, welche Bestandteile außerhalb der Prozesslogik durch fertige Komponenten abgebildet werden können. Fehlende Funktionalität wird dann durch individuell zu entwickelnde Komponenten bereitgestellt. Diese Zentralisierung von (übergreifender) Prozesslogik in dedizierte Komponenten der Systemlandschaft ist ein echtes, häufig wiederkehrendes Muster und spiegelt sich somit auch in der von sd&m entwickelten Referenzarchitektur der „kategorisierten Anwendungslandschaft“ wider [4].
Neu an der hier vorgestellten Fallstudie ist die Unterscheidung zwischen zwei zentralen Komponenten in der technischen Architektur für die Abbildung der Prozesslogik. Eine Komponente ist dabei für den Teil der Prozesslogik zuständig, die entweder automatisch oder durch die sukzessive Interaktion mit einem Benutzer gesteuert wird. Die andere zentrale Komponente ist dafür zuständig, dass effiziente Übergänge der Prozessverantwortung zwischen einzelnen Benutzern bzw. Benutzergruppen sichergestellt sind. Eine solche Komponente

wird oft auch als *Workflow-Steuerung* oder *Event-Manager* bezeichnet. Die Notwendigkeit zum Einsatz einer derartigen Komponente ergibt sich aus der Komplexität und Häufigkeit der Übergänge in der Prozessverantwortung zwischen den einzelnen Nutzern.
Als Mittel in der Prozessdefinition hat sich hierbei der Einsatz des genannten „Notenlinien"-Schemas bewährt, mit denen die Verantwortung für die einzelnen Prozessschritte dokumentiert wird. Über „Tonwechsel" können diese Übergänge identifiziert werden. Die Folgende Abbildung stellt diese spezielle Form der Dokumentation exemplarisch dar:

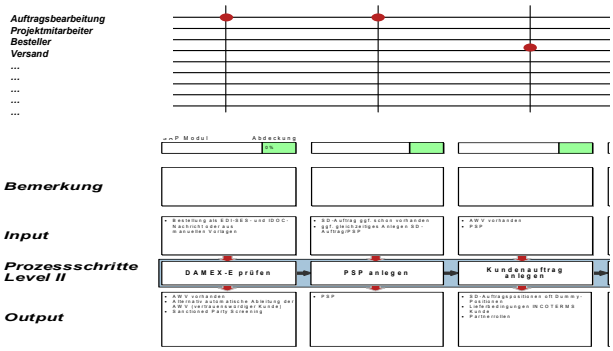


Abbildung 2:Soll-Prozess-Definition

Hierbei sind im unteren Teil die einzelnen Prozessschritte inklusive des benötigten Inputs sowie des erzeugten Outputs eines Prozessschrittes aufgetragen. Die Linien darüber repräsentieren einzelne Fachbereichsgruppen, die für die Durchführung der jeweiligen Prozessschritte verantwortlich sind.
Durch diese Form der Definition des Soll-Prozesses konnte die Verantwortlichkeit für die übergreifende Prozesssteuerung der jeweiligen Komponente eindeutig identifiziert werden. Das Ergebnis war ein Architekturvorschlag, der viel Flexibilität bei der Umgestaltung von Prozesslogik zulässt.

Literatur

[1] Voß, M.: *Synthese eines Vorgehens zur Migrationsplanung*. Workshop Software-Reengineering (WSR 2006), GI Softwaretechnik-Trends, Band 26, Heft 2, 2006

[2] Siedersleben, J.: Moderne Software-Architektur – umsichtig planen, robust bauen mit Quasar. dpunkt Verlag, 2004

[3] Schaumann, P.: *Altsysteme von außen nach innen ablösen*. IT Fokus 7/8, 2004

[4] Humm, B., Hess, A., Voß, M.: *Regeln für serviceorientierte Architekturen hoher Qualität*. Informatik-Spektrum 6/2006, Springer Verlag, 2006

Tool-Demos

BS2 MigMan - Migration Manager für BS2000 Architekturen

Anbieter/Hersteller: pro et con Innovative Informatikanwendungen GmbH
Referent: Denis Uhlig
Kurzbeschreibung:

BS2 MigMan ist eine IDE, welche eine automatische Migration von BS2000 Applikationen in UNIX-Umgebungen realisiert. Das Tool ist eine Eclipse-Anwendung mit folgenden Funktionalitäten:

- Anwendungsprogramme im BS2000 eigenen COBOL-Dialekt werden in allgemeingültige COBOL-Dialekte für UNIX konvertiert.
- Ein in das Tool integrierter Sprachkonvertierer übersetzt die BS2000-eigene Systemprogrammiersprache SPL in C++.
- SDF-Prozeduren werden nach Perl konvertiert.

Die komfortable graphische Oberfläche garantiert die Kontrolle über alle MigMan-Funktionen. Die integrierte Versionsverwaltung basiert auf dem Open Source Produkt “Subversion” und realisiert das Speichern verschiedener Projektzustände.

weitere Informationen:
`http://www.proetcon.de`

COBOL FGM - Flow Graph Manipulator für COBOL

Anbieter/Hersteller: pro et con Innovative Informatikanwendungen GmbH
Referent: Anja Beier
Kurzbeschreibung:

COBOL FGM unterstützt das Programmverstehen und die Redokumentation von COBOL-Applikationen. Es wird ein vollständiger Einblick in komplexe COBOL Projekte geliefert. Feingranulares Wissen eines einzelnen Programmes wird kombiniert mit Applikationswissen. Die Analyse von “embedded” Systemen wie SQL ist ebenso integriert wie die Auswertung von Präprozessordirektiven. Es werden verschiedene COBOL-Dialekte unterstützt. Integriert sind Datenflußanalysen und die graphische Darstellung des Steuerflusses der Programme in Form von Programmablaufplänen. Funktionale Erweiterungen aufgrund spezifischer Kundenanforderungen sind durch die Existienz einer tooleigenen Scriptsprache jederzeit möglich. Die Nutzung von COBOL FGM reduziert Wartungskosten. Programmverstehen und aktuelle Dokumentationen bilden die erste wichtige Stufe von Migrationsprojekten.

weitere Informationen:
`http://www.proetcon.de`

Kb-AIM V5 - Application Knowledge Platform

Anbieter/Hersteller: hal knowledge solutions ag
Referent: Guido Salvi
Kurzbeschreibung:

Die Application Knowledge Plattform (AKP) unterscheidet vier Wissenskategorien: Systemarchitektur, Systemqualität, Systementwicklung und die wirtschaftliches Bedeutung des Systems. Die Informationen können nach Kategorien, nach Geschäftseinheiten oder für das gesamte Unternehmen abgefragt werden. Zudem erstellt die AKP Software Metriken. Diese Metriken werden mit Hilfe einer Einschätzung des Software-Entwicklungsprozesses und durch die Bewertung von Informationen zur Applikationsgröße, Wartbarkeit sowie Komplexität ermittelt.

Die AKP ist modular aufgebaut und kann durch neue Programmiersprachen erweitert werden. Ebenso können wichtige, externe Informationen wie z.B. Konfigurations- und Change Management (CCM)- Daten, Informationen zur Programmleistung und Tools zur Ereignisüberwachung integriert werden.

Die Darstellung und die Dokumentation der Informationen erfolgt durch optimierte Entscheidungsmodule (Decision Support Modules):

- Dynamic Inventory
- Software Quality Management
- Outsource Contract Management
- Productivity Management
- Enterprise Impact Analysis
- Enterprise Technical Documentation

weitere Informationen:
`http://apm.microfocus.com/`

NStop MigMan - Migration Manager für HP NonStop Architekturen

Anbieter/Hersteller: pro et con Innovative Informatikanwendungen GmbH

Referent: Rene Groschopp
Kurzbeschreibung:

NStop MigMan ist eine IDE, welche alle Komponenten einer automatischen Migration von HP NonStop (Tandem) Applikationen in UNIX-Umgebungen unterstützt. In Screen-Cobol definierte Benutzeroberflächen werden zu JAVA-Clients, das hierarchische Filesystem Enscribe wird in eine relationale Datenbank (Oracle oder DB2) migriert. Auch für die Migration des proprietären Transaktionsmonitors Pathway in moderne Middleware (Tuxedo, Websphere, RPC,..) existiert Tool-Unterstützung. Schließlich werden Anwendungsprogramme im HP NonStop eigenen COBOL-Dialekt an allgemeingültigen COBOL-Dialekt auf UNIX-Umgebungen angepaßt. Ein in das Tool integrierter Sprachkonvertierer übersetzt die Tandem-eigene Systemprogrammiersprache TAL in C++. Eine komfortable graphische Oberfläche garantiert die Kontrolle über alle MigMan-Funktionen. Die integrierte Versionsverwaltung ermöglicht das Speichern verschiedener Projektzustände.

weitere Informationen:

`http://www.proetcon.de/DEUTSCH/MigMan/frame_migman_d.html`

SRA - Software Reengineering Architektur

Anbieter/Hersteller: pro et con Innovative Informatikanwendungen GmbH
Referent: Herr Uwe Erdmenger
Kurzbeschreibung:

SRA ist ein Werkzeugkasten zur Entwicklung von Migrations-Tools. Alle Migrations-Werkzeuge der Firma pro et con entstanden unter Nutzung von SRA. SRA selbst ist ebenfalls eine Eigenentwicklung. Daraus werden präsentiert:

1. Parsergenerator “BTRACC”:

   Generiert aus einer formalen Beschreibung den Quellcode eines Parsers. Im Vergleich zu anderen Parsergeneratoren (z.B. yacc) entfallen durch das zugrundeliegende Backtracking-Verfahren Notationsbeschränkungen der Eingabegrammatik und die Attributierung ist wesentlich komfortabler. Die Demonstration erfolgt anhand eines Praxisbeispiels (Generierung eines SPL-Parsers).
2. “Tree Handler” zur formalen Notation von Syntaxbäumen:

   Eingabe für das Werkzeug ist eine formale Notation eines Syntaxbaumes. “Tree Handler” generiert daraus den Code für die Klassen des Syntaxbaumes. “Tree Handler” wird eingesetzt bei der Entwicklung von Sprachkonvertierern (Translatoren).
3. “C-Gen” zur Generierung von C/C++-Quellcode:

   Der letzte Schritt bei einer toolgestützen Sprachkonvertierung besteht in der Zielcode-Generierung. "C-Gen" generiert automatisch aus der internen Repräsentation eines Programmes (Syntaxbaum) den strukturierten Zielcode. Mittels eines Config-Files wird auf das Zielcode-Layout nutzerspezifisch Einfluß genommen (z.B. Blockeinrückungen, Zeilenumbrüche, Formatierungen,...).

weitere Informationen:

`http://www.proetcon.de`

TESSI - Textual assistant

Anbieter/Hersteller: Lehrstuhl Informationssysteme und Softwaretechnik, TU Chemnitz
Referent: Michael Rentzsch
Kurzbeschreibung:

CASE-Werkzeuge unterstützen den Softwareentwicklungsprozeß erst ab einer Phase, zu der Diagramme (z.B. UML, Datenfluß usw.) gezeichnet werden. TESSI zwingt den Analytiker, schon zu Beginn der Anforderungsanalyse eine textuelle Beschreibung der Anforderungen zu erstellen und von diesen Anforderungen ein objektorientiertes Modell (auf UML-Basis) abzuleiten. Nach Abschluß der Modellbeschreibung durch den Analytiker generiert TESSI einen Text, der von diesem Modell automatisch abgeleitet wird. Dieser Text repräsentiert die Vorstellungen des Analytikers und ist außerdem für den Kunden verständlich. Der Kunde validiert den Text. Während dieses Prozesses generiert TESSI auch Metriken, welche die Schätzung von Projektaufwand und -kosten unterstützen.

Präsentiert wird ein Prototyp, der sich in Entwicklung befindet und bei studentischen Praktika eingesetzt wird.

weitere Informationen:

`http://www.tu-chemnitz.de/informatik/ISST/forschung/TESSI/`

ZEDERBrowser - Browser für das Zentrale Debeka-Repository (ZEDER)

Anbieter/Hersteller: Debeka-Hauptverwaltung
Referent: Martin Schulze
Kurzbeschreibung:

Der ZEDER-Repository-Browser der Debeka ist eine eigenentwickelte Eclipse-Anwendung, mit der die Entwickler der Versicherungsgruppe in den Inhalten des “Zentralen Debeka-Repositorys” (ZEDER) navigieren. Der Browser ist modellunabhängig und unterstützt die Orientierung innerhalb der Vererbungshierarchie der Repository-Objekte. Zur Unterstützung bei häufigen Fragestellungen dienen Abkürzungslinks und vordefinierte Anfragen.

weitere Informationen: