Das ARNO Projekt

Herausforderungen und Erfahrungen in einem großen industriellen Software-Migrationsprojekt

Werner Teppe, Robert Eppig

Amadeus Germany GmbH
Geschäftsbereich Development
Marienbader Platz 1
61348 Bad Homburg
wteppe@de.amadeus.com
reppig@de.amadeus.com

Abstract: In diesem Artikel fassen wir das Vorgehen und die Erfahrungen mit der Migration eines sehr großen Anwendungssystems zusammen. Über wichtige Aspekte des Projekts und den Projektverlauf berichteten wir regelmäßig auf der jährlichen WSR und der REPRO in Vorträgen.

Das Projekt ARNO hatte zum Ziel, alle Anwendungen einer Mainframe-Plattform auf UNIX zu migrieren, damit die komplette Systemplattform abzulösen und so erhebliche Hardware-, Softwarelizenz- und Infrastrukturkosten einzusparen. ARNO steht für **A**pplication **R**elocation to **N**ew **O**perating System. Die besonderen Herausforderungen bestanden darin, dass die umzustellenden Anwendungssysteme nicht stand-alone arbeiten, sondern Rechnerkopplungen zu rund 200 externen Partnersystemen unterhalten. Zudem sollte es möglich sein, während der mehrere Jahre dauernden Projektlaufzeit monatlich neue Anwendungsreleases herauszubringen, um Kundenanforderungen zu erfüllen. Außer den Anwendungen mussten das hoch performante Filehandlingsystem durch eine Datenbank abgelöst, die Middleware umgestellt und angepasst sowie umfangreiche Jobs (Skripte) auf das Zielsystem portiert werden. Die Systeme laufen in einem nahezu 7*24 Stunden Betrieb und wickeln in Spitzenzeiten ca. 750 Benutzertransaktionen pro Sekunde (TA/s) ab – dies entspricht rund 1500 technischen TA/s. Die Projektbeteiligten waren in unterschiedlichen Unternehmen über mehrere Standorte verteilt.

Keywords: Projekt ARNO, Migration, START-System, Ablösung von Mainframesystemen, Portierung, C++, PERL, SPL, Oracle, UTM, Projektmanagement von Großprojekten, BS2000, UNIX, Software-Reengineering, WSR

1 Amadeus und das Amadeus Germany (START-) System

Die Amadeus Germany GmbH ist Deutschlands führender Anbieter von IT-Lösungen für die Reisebranche. 1971 gegründet, liefert das Unternehmen heute ein umfassendes Angebot für den Vertrieb touristischer Leistungen aller Art über verschiedene Absatzkanäle wie zum Beispiel Reisebüros, Call Center, Kartenvorverkaufsstellen oder das Internet. Mit seinem Geschäftsbereich Corporate Solutions bietet Amadeus Germany darüber hinaus leistungsstarke Geschäftsreise-Lösungen für effizientes Travel Management in Unternehmen. Umfangreiche Trainingsmöglichkeiten runden das Portfolio ab. In Deutschland arbeiten 85 Prozent aller Reisebüros an rund 45.000 PCs mit dem modernen, leistungsstarken und hoch entwickelten Amadeus System.

Folgende Anbieter sind in Deutschland buchbar: rund 500 Fluggesellschaften, über 75.000 Hotels, 22 Mietwagenfirmen, rund 200 Reise- und Busveranstalter, 74 Verkehrsverbünde, 40 europäische Bahnen, 30 Fähranbieter, sechs Versicherungsanbieter, drei Event-Ticket-Anbietersysteme mit mehr als 1.000 Veranstaltern sowie acht Kreuzfahrtlinien.

Alleiniger Gesellschafter von Amadeus Germany ist die Amadeus IT Group SA, ein weltweit führender Anbieter von Technologie- und Vertriebs-Lösungen für die Reise- und Tourismusbranche. Über 90.270 Reisebüros und mehr als 29.660 Airline-Verkaufsbüros – und damit über 600.000 PCs – in über 217 Märkten weltweit nutzen das Netz und das leistungsstarke Datenzentrum.

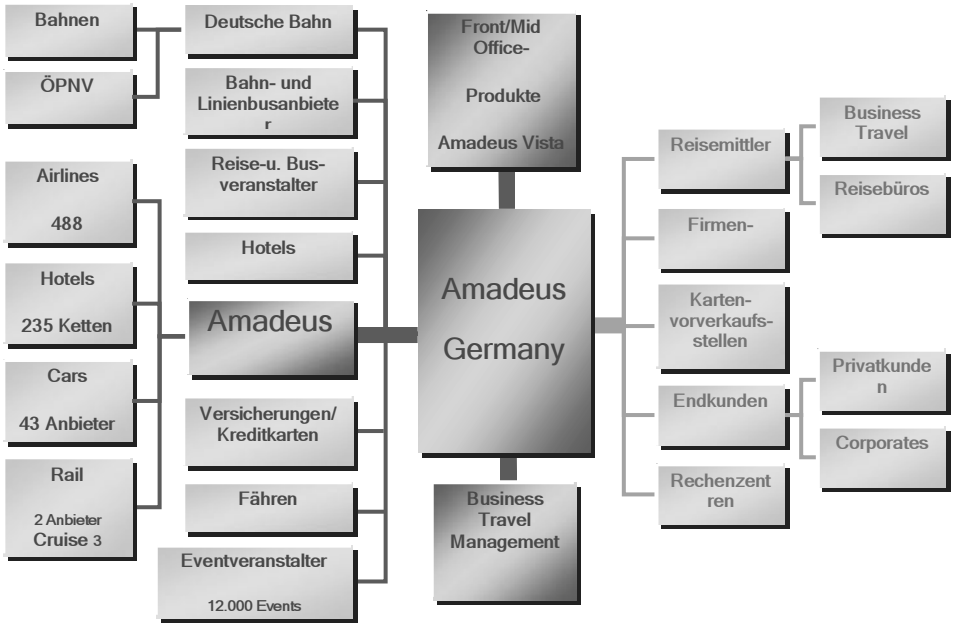

Abbildung 1: Das Amadeus Germany System als Integrator

Das Amadeus Germany (START-) System ist die Verbindung zwischen Anbietersystemen auf der einen Seite sowie Reisebüros und Endkunden auf der anderen Seite (siehe Abb. 1). Neben Reiseinformationen, Buchungsanfragen und -bestätigungen werden Reisedokumente und Buchhaltungsdaten zur Verfügung gestellt sowie Zahlungsflüsse zwischen den beteiligten Geschäftspartnern organisiert. Das Amadeus Germany System ist an das beschriebene internationale Amadeus System angeschlossen.

2 Ausgangssituation

2.1 Gewachsene Applikation

Das Amadeus Germany System wurde als START System von 1976 bis 1979 entwickelt und 1979 mit einem selbst aufgebauten Netzwerksystem für ca. 800 „dumme Terminals“, 500 Drucker und 3 Rechnerkopplungen zu Anbietersystemen in Betrieb genommen. Dabei wandten die Entwickler die damals neuesten Erkenntnisse der Informatik und des Software Engineering erfolgreich in der Praxis an [Par72, Den91]. So war es möglich, ca. 30 Jahre lang erhebliche Erweiterungen am System vorzunehmen (s. o.) sowie grundlegende Technologiewechsel im Mainframe-, Endgeräte- und Netzwerkbereich erfolgreich zu bewältigen und dabei die grundsätzliche Systemarchitektur beizubehalten. Die Anwendungen laufen auf mehreren Großrechnersystemen unter BS2000. Zusatzdienste erbringen mehrere UNIX- bzw. LINUX Systeme mit dedizierten Anwendungen.

2.2 Hauptgründe für eine Migration

In der Zwischenzeit wurden besonders im UNIX Umfeld neue, effektivere (grafische) Werkzeuge zur Softwareentwicklung bereitgestellt. Die Programmiersprache SPL (ein PL/I Ableger), in der die Anwendungen entwickelt sind, wird nur noch von wenigen Programmierern beherrscht. Nachwuchsprogrammierer sind in diesem Umfeld nicht leicht zu finden. Ein Anschluss an die objektorientierte Welt ist nur schwer möglich. Mainframes erfordern außerdem eine besondere Infrastruktur und sind auch in der Beschaffung erheblich kostenintensiver als z. B. Server auf UNIX Basis.

3 Vorgehen (Vorgehensmodell im Projekt)

3.1 Studie

Zu Beginn untersuchten die Projektverantwortlichen anhand einer Experten-Studie, ob und wie eine Ablösung der Systeme stattfinden kann. Im Anschluss stand die Analyse der Kosten des aktuellen Systems und der potentiellen neuen Systeme an sowie die Bewertung der Wartung, Weiterentwicklung und des Betriebs der Anwendungen auf altem und neuem System. Darüber hinaus standen drei Alternativen der Migration zur Debatte:

1. Die Neuentwicklung der Anwendungen (mit demselben Funktionsumfang)

2. Die Konvertierung der vorhandenen Programme, Skripte (Jobs) und Dateien

3. Die Entwicklung eines Compilers bzw. Frontends, der aus SPL Sourcen UNIX-Objektcode generiert (beibehalten der SPL-Quelltexte)

Hier die wesentlichen Kriterien für die Entscheidung, die über umfangreiche Einzelbewertungen herbeigeführt wurde: Alternative drei hätte zur Folge gehabt, dass die Weiterentwicklung in der bisherigen Sprache mit den entsprechenden Nachteilen (verfügbares Personal, fehlende Objektorientierung) hätte durchgeführt werden müssen. Für die Alternative zwei erwarteten die Projektverantwortlichen einen ähnlichen Aufwand wie für die Alternative drei, für Alternative eins hingegen einen vielfach höheren Aufwand. Sie wäre gleichbedeutend gewesen mit einer Neuentwicklung des Systems bei gleichem Funktionsumfang – allerdings mit dem Nachteil, dass alle Fehler eines neu entwickelten Systems (wahrscheinlich wesentlich mehr) hätten gefunden und behoben werden müssen. Hingegen hätten die Entwickler bei Alternative zwei „nur“ die Fehler der Konvertierung ausgleichen müssen. Zusätzlich wäre den Anwendern die Neuentwicklung eines Systems aus rein technischen Gründen ohne Funktionserweiterung nur schwer zu vermitteln gewesen. Die Wahl fiel damit auf Alternative zwei. Weitere Entscheidungen: Als Zielsprache legte das Team C++ fest und ermöglichte so die Erschließung der Welt der Objektorientierung. Als Skriptsprache wurde PERL verwendet, und als Datenbanksystem der Konzernstandard Oracle definiert. Einige der umzustellenden Anwendungen liefen bereits unter dem Transaktionsmonitor UTM, der bei gleicher Funktionalität auch unter UNIX zur Verfügung steht, so dass auch die Middleware festgelegt werden konnte.

Die Migration der SPL-, SDF- (Skript-Sprache) und Dateiobjekte war werkzeugunterstützt und weitgehend automatisch geplant. Da die Entwicklung dieser Werkzeuge eine einmalige Aufgabe im Projekt bzw. im Unternehmen war und nicht zur Kernkompetenz von Amadeus gehört, war ein Outsourcing dieser Aufgabe die Konsequenz. Die eigentliche Migration war wegen des erforderlichen Fachwissens und der Kenntnisse über die einzelnen Prozesse in-house vorgesehen. Auch diese Grundsatzentscheidung trug wesentlich zum Projekterfolg bei [Tep03, Tep06],

3.2 Projektorganisation

Am Projekt waren verschiedene Konzernbereiche sowie rechtlich eigenständige Firmen beteiligt. Aus diesem Grund war es unabdingbar, dass im Steuerungsgremium des Projekts – dem Steering Committee als Ansprechpartner des Projektleiters – die Geschäftsführer der jeweiligen Firmen vertreten waren. Nur auf diesem Weg waren die nötige Priorisierung gegenüber anderen Vorhaben und kurze Entscheidungswege möglich und so die nötige „Management Awareness“ gewährleistet. Es bestand die Gefahr, dass Projekte mit „sichtbarem“ Nutzen für den Kunden während der Projektlaufzeit dazwischen geschoben würden. Erfahrungsgemäß führt dies bei Migrationsprojekten dieser Größenordnung zu Ressourcenkonflikten und überlangen Laufzeiten. Häufig führt dies zum Scheitern der Projekte. Zu beachten ist außerdem, dass bedingt durch die relative lange Projektlaufzeit, das Projektziel durch die laufenden Änderungen am System zu einem „moving target“ werden kann. Ein definierter Change Management Prozess ist daher für den Projekterfolg unabdingbar.

3.3 Vorbereitung der Migration („erst sanieren, dann migrieren“)

Eine Migration der Applikation war nicht ohne vorbereitende Maßnahmen unter BS2000 möglich. Eine vollständige Inventur aller Programmquellen, aller eingesetzten Skripte (Jobs) und aller Hilfsprogramme anderer Hersteller war zur Ermittlung des umzustellenden Gesamtpakets zwingend erforderlich. Nur so konnte der genaue Projektumfang definiert werden. Dabei wurden auch Elemente identifiziert, die bereits nicht mehr benötigt wurden und aus dem System hätten entfernt werden können (z. B. „tote Zweige“, also Programmteile oder Abläufe die gar nicht mehr durchlaufen werden). Dazu wurde eine Qualitätsoffensive im BS2000 Umfeld vor der Konvertierung durchgeführt. Zusätzlich wurde mit den Produktmanagern diskutiert, welche Produkte während der Projektlaufzeit vom Markt genommen werden sollten. Diese konnten damit aus dem Migrationsportfolio gestrichen werden und mussten nicht umgestellt werden. So konnte der Konvertierungs- und Testaufwand deutlich reduziert werden. Das ARNO Projekt war zugleich auch ein geeigneter Anlass, eine Produktbereinigung durchzuführen.

3.4 Der Main Migration Process (MMP)

Aus Kostengründen wurde festgelegt und spezifiziert, dass die Migrationswerkzeuge STC, JTC und FTO nicht den kompletten Sprachumfang der jeweiligen Systeme konvertieren sollen [Erd04, EU07]. Daher war ein iteratives Vorgehen bei der Konvertierung nötig. Basis aller Konvertierungen war die jeweils in Produktion befindliche Version einer Source (siehe Abb. 2). Eine Source wurde z. B. vom STC konvertiert. War sie fehlerfrei, wurde sie in das Repository der erzeugten C++ Sourcen aufgenommen. Gab es Fehler, wurde untersucht, ob die SPL-Source angepasst oder der

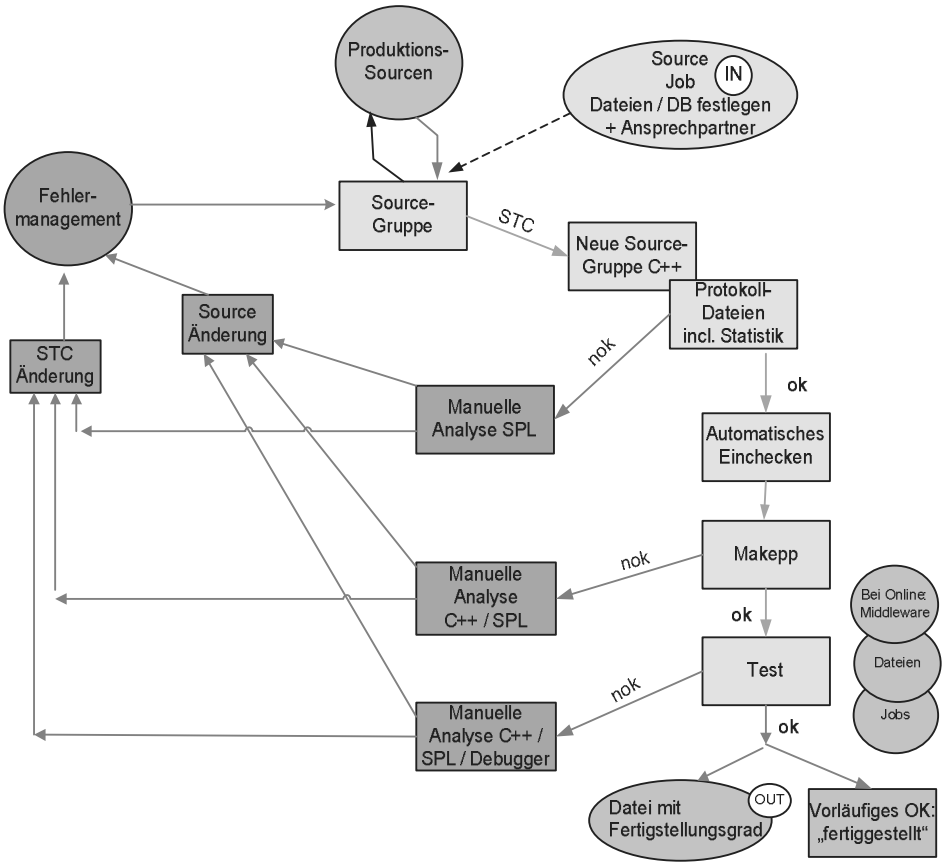

Abbildung 2: Der Main Migration Process (MMP)

STC Konverter verbessert werden. Dies hing unter anderem von der Anzahl ähnlicher „Problemfälle“ ab. Waren alle benötigten Objekte konvertiert, wurden sie vom C++ Compiler übersetzt und gebunden (makepp). Gab es bei diesem Vorgang Fehler, wurde wieder entschieden, ob der Quelltext oder der STC angepasst werden. Konnte ein Objekt erfolgreich gebunden werden, wurde es getestet. War der Test erfolgreich, waren somit alle Komponenten migriert. Andernfalls war wieder oben beschriebene Entscheidung zu treffen. Dieses Vorgehen galt für alle Programme, Jobs, Dateien und Datenschnittstellen. Die Änderungen an den Originalobjekten im BS2000 wurden über einen funktionalen Gesamttest der Produktion wieder zugeführt, damit zum endgültigen Systemumstieg SPL und C++ Sourcen denselben Funktionsumfang hatten und die Programme unter BS2000 auch ablauffähig waren („Verifikation in Produktion“). Der Zustand jeder einzelnen Source wurde vom Konfigurationsmanagementsystem verwaltet. Festgehalten wurde auch, mit welcher Konverterversion ein Objekt migriert wurde, da Änderungen am Konverter zur Folge haben konnten, dass ein schon positiv umgesetztes Objekt plötzlich nicht mehr konvertierbar gewesen wäre.

3.5 Der integrierte Testprozess (ITP)

Beim Testen der umgestellten Systeme waren neben den Projektmitarbeitern und den Mitarbeitern aus Entwicklungsabteilungen auch Mitarbeiter aus dem Produktmanagement und dem Help Desk beteiligt. Es wurde ein Testprozess aufgesetzt, der mit Hilfe des Konfigurationsmanagementsystems die Testergebnisse dokumentierte und bei Fehlern dafür sorgte, dass die Fehlermeldung als RFD (Request for Development) festgehalten wurde. Die Testkoordination hatte damit jederzeit den Überblick über den Gesamtzustand aller Tests sowie über die festgestellten Fehler und konnte diese RFDs den entsprechenden Entwicklern im Projekt zur Korrektur zuweisen. Über den MMP korrigierte Objekte wurden dann dem Testprozess wieder zugeführt.

Diese vorbereitenden Maßnahmen vereinfachten die Umstellung wesentlich und ermöglichten in großen Teilen das Vorhaben erst.

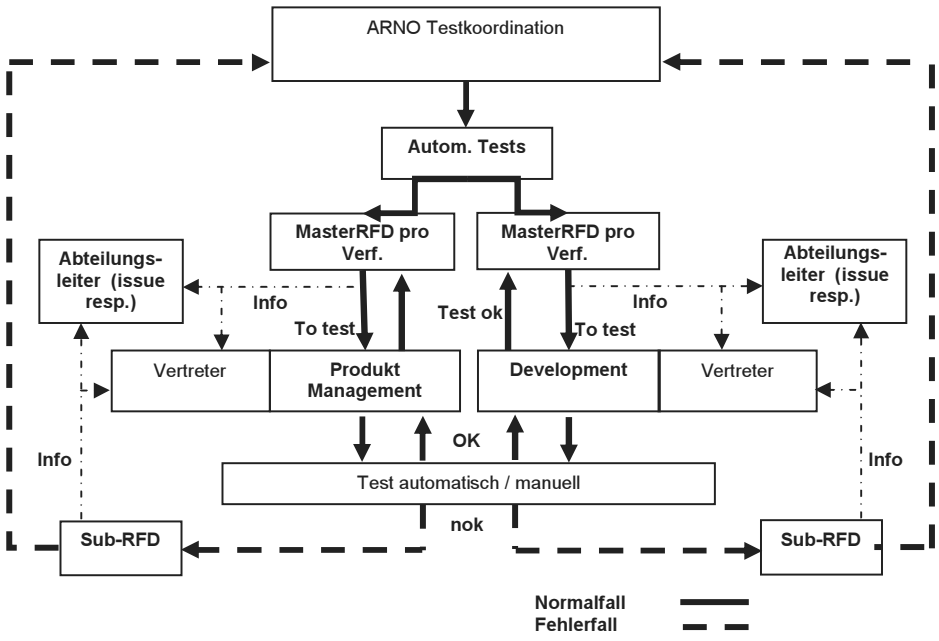

Abbildung 3: Integrierter Testprozess: Informationsfluss der RFDs für Testkoordination und im Entwicklungsprozess

3.6 Mitarbeiter

Bei Projektbeginn wurde vereinbart, dass Entwicklungs- und Produktionsmitarbeiter des Altsystems „mitmigriert“ werden, um das Anwendungs- und Produktionswissen zu erhalten. Es war klar, dass dies ein erhebliches Asset des Unternehmens ist (Verfahrenswissen ist wichtiger als das Beherrschen einer Programmiersprache), welches es zu erhalten gilt. Daher wurden umfangreiche Weiterbildungsmaßnahmen im Bezug auf die Zielsysteme durchgeführt. Ausgiebige Informationsveranstaltungen sowie die Einführung von Projekt-WIKIs und -BLOGs stellten die Wissenssammlung und Informationsweitergabe sicher. Diese Form erleichterte auch die Nachdokumentation des (Alt-)Systems. Das Vorgehen führte zu einem wichtigen Change-Prozess im gesamten Unternehmen.

3.7 Übergabe an die Linienorganisation

Da der Betrieb der Rechnersysteme und der Anwendungen in anderen Unternehmensbereichen lag als das Projektteam und die -leitung und die Projekt-organisation geringen Einfluss auf die Systemadministration und die Organisation des Rechenzentrums hat, wurde im Projektauftrag festgelegt, dass mit der Fertigstellung der Migrationsobjekte (nachgewiesen durch System-, Acceptance- und Performancetests) die Verantwortung auf die Linienorganisationen übergeht. Dies schloss auch die Übergabe der umgestellten Sourcen an die Developmentabteilungen ein. Durch Einbeziehung der betroffenen Entwicklungs-, Wartungs-, und Produktionsbereiche in die Projektarbeit wurde die Übergabe erfolgreich vorbereitet und ermöglicht (Beteiligung der Betroffenen). Die Mitarbeiter hatten zunächst eine Rolle im Projekt und führten später ähnliche Aufgaben in der Linie weiter.

4 Technische Durchführung des Projekts

4.1 SPL to C++

Die zu migrierenden Anwendungen lagen unter BS2000 in SPL vor. Dabei handelt es sich um eine prozedurale Programmiersprache, die systemnahe und effiziente Programmierung unter BS2000 ermöglicht. Da durch die große Menge an umzustellenden Programmcodes ein manuelles Neuschreiben nicht möglich war, musste eine automatisierte Umsetzung erfolgen. Die Verwendung der unter Unix weit verbreiteten, ebenfalls prozeduralen Programmiersprache C war nahe liegend. Da in C aber nicht alle Sprachelemente von SPL identisch zur Verfügung stehen, mussten sie in Form einer Kompatibilitätsschicht ‚SPLCompat’ bereitgestellt werden.

Da sich solche fehlenden Elemente am effektivsten durch objektorientierte Programmierung realisieren ließen, wurde C++ als Zielsprache gewählt. Diese Kompatibilitätsschicht beinhaltet unter anderem die Datentypen, die in SPL vorhanden

sind, für die aber weder C++ noch die C++ Standardbibliothek Entsprechungen enthält. Dazu gehören zum Beispiel Klassen zur Zeichenketten- und Bitleistenverarbeitung mit allen Operatoren und exakt dem unter SPL vorhandenen Verhalten. Der Konverter STC (SPL to C++) erzeugte aus dem SPL Code einen unter Unix verwendbaren C++ Code unter Verwendung der erwähnten Bibliothek SPLCompat. Dabei mussten auch etliche nicht triviale Probleme wie die unterschiedliche Sichtbarkeit von Variablen in den verwendeten Programmiersprachen behandelt werden. Hier wurden Regeln definiert, wie SPL-Konstrukte auf entsprechende C++-Konstrukte umzusetzen sind. Wesentlich half dabei, dass durch strikte Programmierrichtlinien nicht alle unter SPL denkbaren Probleme behandelt werden mussten. Die Einhaltung dieser Richtlinien wurde bereits vor der Konvertierung überprüft und Abweichungen unter BS2000 im SPL Code korrigiert. Erst dadurch war eine eindeutige Umsetzung überhaupt möglich. Diese Regeln mussten im Laufe der Umstellungsphase mehrmals im Detail angepasst werden.

Ein weiteres Problem ergab sich dadurch, dass die unter C++ erzeugte Definition von Variablen und Konstanten hinsichtlich Typ und Sichtbarkeit erst durch die jeweilige Nutzung definiert werden konnte. Viele dieser Variablen und Konstanten werden aber in Includes (Headerfiles) definiert, die in mehreren Teilprogrammen verwendet werden. Die Nutzung dieser Variablen und Konstanten ist in den einzelnen Teilprogrammen aber nicht exakt identisch. Das führte dazu, dass für ein und dasselbe SPL-Include oft mehrere verschiedene, inkompatible C++-Includes erzeugt wurden. Die Anzahl dieser Mehrfach-Includes wurde im Wesentlichen durch eine geschickte Anpassung der Regeln für die Erzeugung der C++-Includes reduziert. Wo das nicht möglich war, wurde durch Modifikation des SPL-Codes (eine Umformulierung des Programms) eine eindeutige Umsetzung erreicht. Die wenigen verbleibenden Probleme wurden dann manuell nach der Konvertierung (in der Zielsprache) angepasst. Trotz aller Bemühungen konnte die Codekonvertierung nicht zu hundert Prozent automatisiert ablaufen. Selbst durch alle Richtlinien und durch die Reduktion des Sprachumfangs konnten nicht immer eindeutige Umsetzungsregeln erzielt werden. Für die verbleibenden Problemfälle erzeugte der Konverter STC Warnungen. Diese wurden dann manuell überprüft und gegebenenfalls angepasst. Kontrolliert wurde dies mit Hilfe eines gut organisierten und speziell auf diese Arbeiten angepasstes Konfigurationsmanagements. Insgesamt betrachtet, konnte bei der Sprachumsetzung ein sehr großer Automatisierungsgrad erreicht werden.

4.2 JCL to Perl

Außer den Programmen war auch eine große Anzahl von Prozeduren nach Unix zu migrieren. Auch hier war bedingt durch den Umfang keine manuelle Umstellung möglich. Im Gegensatz zum Quellcode der Programme waren die vorhandenen Quellen für die Prozeduren aber wesentlich inhomogener und schlechter definiert. Zum einen besteht die unter BS2000 verwendete Job Control Language (JCL) eigentlich aus mehreren verschiedenen Sprachen. Im Einsatz waren dabei ISP und SDF. Zum anderen waren viele Prozeduren einfache Hüllen für Hilfsprogramme, wie zum Beispiel ‚sort' zum Sortieren von Dateien. Andere gerufene Hilfsprogramme wie der BS2000 Editor EDT oder der Report Generator Easytrieve enthalten selbst eine Art interne Prozedursprache.

Um den Code zu homogenisieren und damit eine automatisierte Konvertierung sowie eine automatisierte Analyse überhaupt erst zu ermöglichen, wurden im ersten Schritt alle Prozeduren mit Hilfe des Programms SDF-CONV von Fujitsu Siemens bearbeitet. Die Ausgabe waren neue, einheitliche Prozeduren in einer wohl definierten Untermenge des Sprachumfangs von SDF. Diese neuen Prozeduren konnten dann von einem etwas einfacher gehaltenen Konverter JTP (JCL to Perl) nach Perl umgestellt werden.

Für die in den Prozeduren meist genutzten Hilfsprogramme wurden in Perl Ersatzmodule bereit gestellt. Auch wenn es beispielsweise für das Hilfsprogramm ‚sort' durchaus unter Unix vergleichbare Ersetzungen gibt, so gibt es beim jeweiligen Funktionsumfang der Produkte Abweichungen. Das machte einen einfachen Austausch der Programmaufrufe unmöglich. Die neu erzeugten Perl Module konnten dagegen mit Hilfe von formal definierbaren Umsetzungsregeln durch JTP an Stelle der Aufrufe der BS2000 Hilfsprogramme eingefügt werden.

Einige Konstrukte wurden trotzdem manuell umgesetzt. Beispielsweise wäre die Entwicklung eines Analyse- und Konvertierungstools für die Editor (EDT) interne Prozedursprache deutlich aufwendiger gewesen, als die Prozeduren mit diesen Elementen unter Unix neu zu schreiben. Auch konnte (analog zum Vorgehen beim STC) durch JTP eine vollständige Liste der nicht konvertierbaren Codebereiche erstellt werden.

4.3 Filehandler to Oracle

Unter BS2000 wurde zur Datenhaltung eine selbst entwickelte, sehr systemnahe Programmbibliothek mit dem Namen ‚Filehandler' verwendet. Sie enthielt die unbedingt notwendigen Funktionen einer Datenbank. Eine Umstellung dieses Programmteils wäre sehr aufwendig gewesen. Durch die große Nähe zum Betriebssystem hätte der Programmteil im Wesentlichen nicht konvertiert werden können, sondern neu entwickelt werden müssen. So waren beispielsweise auch einige Programmteile im /390 Assembler geschrieben.

Nach eingehender Bewertung wurde konsequenterweise die bei Amadeus als Standard eingesetzte Datenbank Oracle 10 verwendet und so in die Applikationsarchitektur eine zusätzliche Tier eingefügt. Da vorher eine Bibliothek (Filehandler) mit klar definierter Schnittstelle vorlag, konnte das ohne Anpassung der Applikationslogik erfolgen. Aufbauend auf die existierende Schnittstelle wurde eine Zwischenschicht entwickelt. Ihre Aufgabe ist es, die alten Filehandler Aufrufe auf in SQL realisierte Datenbankabfragen der Oracle Datenbank umzusetzen. Dabei mussten nicht nur der Funktionsumfang nachgebildet, sondern auch beispielsweise Oracle Fehlermeldungen auf entsprechende Filehandler Fehlercodes umgesetzt werden. Da die Schnittstelle wohl definiert vorlag, konnte hier eine vollständig automatisierte Umsetzung erfolgen.

Durch die Verwendung dieser Zwischenschicht bleibt die saubere Entkopplung der Applikationsschicht von der Datenhaltungsschicht bestehen. Diese Kapselung ermöglicht relativ einfach einen späteren Austausch der Datenhaltung zum Beispiel durch die Wahl eines anderen Datenbankherstellers.

4.4 DCAM-Monitor to openUTM

Die Mainframeanwendungen beinhalteten auch eine selbst entwickelte Middlewareschicht, den DCAM-Monitor – basierend auf den vom BS2000 bereitgestellten Basis-Kommunikationsmethoden (Data Communication Access Method). Da sie speziell für die Anforderungen unserer Anwendungen entwickelt wurde, war sie sehr effizient und belegte wenige Ressourcen.

Diese Schicht war zwangsläufig ebenfalls systemnah programmiert. Bei einer automatisierten Umstellung dieses Quellcodes hätte also nicht nur eine Konvertierung von SPL nach C++ stattfinden müssen, es hätten auch alle Systemaufrufe auf exakte Entsprechungen unter Unix umgesetzt werden müssen. Das wäre nur durch eine zusätzliche Zwischenschicht möglich gewesen, die alle BS2000 Systemaufrufe unter Unix emuliert. Neben großem zusätzlichen Aufwand hätte das auch bedeutet, nicht das Programm zu migrieren, sondern unter Unix einen BS2000 Kommunikations-Emulator zu schaffen. Das hätte nicht dem Projektauftrag entsprochen, sich von der BS2000 Funktionalität zu trennen.

Deshalb wurde entschieden, eine Standard-Middleware einzusetzen. Die Wahl fiel auf openUTM (Universeller Transaktionsmonitor) von Fujitsu Siemens. Dieses Produkt ist gleichermaßen unter BS2000 und Unix verfügbar. Dabei sind auch alle Programmierschnittstellen auf beiden Plattformen identisch. So war es möglich, die Migration in zwei unabhängigen Schritten durchzuführen. Im ersten Schritt wurde unter BS2000 die Middleware in allen Anwendungen auf UTM umgestellt und in Produktion gebracht. Diese Anwendungen konnten dann ohne Änderungen an der Architektur oder an den Middlewareschnittstellen umgesetzt werden und waren direkt unter Unix einsatzbereit. Für die Kommunikationspartner war dieser Plattformwechsel transparent. So konnte im kritischen Schritt der Migration vermieden werden, auch noch gleichzeitig mit der Middleware die Anwendungsarchitektur umzustellen (1:1 Migration).

5 Test

Das erklärte Ziel der Migration war es, die Umstellungen für die Kunden ohne negative Auswirkungen und im Idealfall sogar völlig unbemerkt durchzuführen. Da dem System durch den Umstieg keinerlei neue Funktionalität hinzugefügt wurde, galt es nachzuweisen, dass das neue System sich in allen relevanten Aspekten exakt so wie das alte System verhält. Um diesen Nachweis führen zu können, waren unterschiedliche Tests notwendig. Die Tests der Batchabläufe waren vergleichsweise einfach. Hier war ‚nur' sicherzustellen, dass die Jobs unter UNIX bei gleichen Eingabedateien die gleichen Ergebnisdateien wie unter BS2000 erzeugen. Die in beiden Systemen unterschiedlichen Codierungen (EBCDIC in BS2000, Latin-9 unter UNIX) mussten beim Dateivergleich selbstverständlich beachtet werden. Der Vergleichstest der Online-Anwendungen war wegen der vielen angekoppelten Partnersysteme mit ihren eigenen Datenbanken ungleich schwieriger.

5.1 Testverfahren bei normalem Versionswechsel

Für die im BS2000 laufenden Anwendungen wurden im Jahr etwa zwölf Versionen erstellt und ausgerollt. Darin waren jeweils neben Wartungs- und Pflegearbeiten auch neue Funktionalitäten enthalten. Hierzu gab es ein etabliertes und über Jahrzehnte bewährtes Testverfahren.

Neben dem Nachweis der Implementation der neuen Funktionen fokussierten sich die Tests darauf, sicherzustellen, dass sich diese Änderungen nicht negativ auf das bestehende System auswirken. Dazu wurden einfache Regressionstests durchgeführt. Für manche, selten geänderte Bereiche mit geringen Wechselwirkungen zu neuen oder geänderten Komponenten war die Testabdeckung nicht tief genug für unser Migrationsprojekt. Zum Teil fehlten beispielsweise Negativtests. Vor allem für die besonders alten Komponenten waren kaum automatisierbare Testfälle vorhanden.

5.2 Testautomation

Um während der gesamten Entwicklungs- und Migrationszeit eine verlässliche Qualitätskontrolle zu ermöglichen, waren automatisierte Tests zwingend erforderlich. Ziel war es, innerhalb kurzer Zeit und ohne Einbeziehung der für die einzelnen Komponenten verantwortlichen Entwicklungsabteilungen jeden neuen Aufbau des Systems testen zu können. In Zusammenarbeit mit den bestehenden Testteams wurden dafür eine große Anzahl automatisierbarer Testfälle erarbeitet. So konnte erreicht werden, dass selbst mehrere Programmversionen pro Tag getestet werden konnten, ohne über die gesamte Projektlaufzeit permanent ein komplettes Testteam etablieren zu müssen.

Diese neu geschaffenen, automatisierten Tests stellen dabei ein weiteres positives Projektergebnis dar. Es ist jetzt in die regelmäßigen Testläufe integriert und wird so in Zukunft auf der neuen Systemplattform bei jedem neuen Release zur Qualitätssicherung beitragen.

5.3 Lasttest

Regelmäßige Lasttests waren für die Applikation unter BS2000 aus Kostengründen nicht üblich. Da sich das Lastverhalten zwischen aufeinander folgenden Versionen nur geringfügig änderte, wurde lediglich das Verhalten in der Produktion überwacht. Durch die Migration nach Unix wurden einige für das Lastverhalten wesentliche Komponenten komplett verändert. So wurde zum Beispiel die Architektur um eine neue Datenbankschicht erweitert. Damit hatte sich das Lastverhalten möglicherweise komplett geändert. Der Nachweis, dass die erforderliche Last bewältigt werden kann, war deshalb eine Voraussetzung für den Produktionsstart unter Unix.

Dazu musste erst eine Lasttestumgebung geschaffen werden. Die zu testende Applikation ist Teil eines komplexen Applikationsverbunds. Das erschwerte den Aufbau außerordentlich. Dennoch gelang es, einen Test zu definieren, der die Applikationsumgebung realistisch nachbildete und einen Dauertest ermöglichte. So wurden unter anderem Komponenten geschaffen, die externe Partnersysteme simulieren. Benötigt wurden außerdem realistische Testabläufe, die für diesen Test beliebig oft wiederholbar waren. Bei manchen Testfällen, wie beispielsweise dem Buchen eines Platzes für einen Flug, war das nicht einfach zu realisieren. Wenn ein Platz gebucht wurde, konnte derselbe Platz natürlich nicht noch mal gebucht werden, da dieser bereits besetzt war.

Trotzdem konnten die Tests erfolgreich durchgeführt werden und die Applikation mit der geforderten vollen Last mehr als zwölf Stunden betrieben werden. Nachgewiesen wurde dabei auch, dass keinerlei Leaks, wie bei der Verwendung von Speicher, von Semaphoren oder anderen Betriebsmitteln, vorliegen.

Dabei wiesen diese Tests nicht nur die Einhaltung der Anforderungen erfolgreich nach. Es konnten auf diese Weise auch mehrere Fehler frühzeitig erkannt und beseitigt werden, die unter Entwicklungsbedingungen ohne parallele Verarbeitung nicht erkennbar waren.

Außerdem wurden diese Tests so angelegt, dass sie ab sofort jederzeit einfach reproduzierbar sind. Sie sollen in Zukunft ebenfalls bei jeder neuen Programmversion durchgeführt werden und verbessern dadurch ab sofort die Qualitätssicherung der weiterentwickelten Software.

5.4 Replay einer Online Session aus der Produktion (Nachrichtenprotokolltest)

Selbst ausgeklügelte Tests decken nicht alle Fehler auf. Nicht an alle Konstellationen wird bei der Testfallerstellung gedacht. Und manche Fehler offenbaren sich erst durch das Zusammentreffen von Zufällen. Dadurch kommt es trotz intensiver Tests oft dazu, dass Probleme einer Applikation erst nach Produktionseinführung offenkundig werden. Um die Testabdeckung weiter zu erhöhen, wurde daher ein weiterer Test konzipiert. Dabei wurden alle Nachrichten der umzustellenden Anwendungen in der Produktion unter BS2000 aufgezeichnet, und später mit identischem Zeitverhalten an die neuen Applikationen unter Unix gesendet. Auch hier war die Simulation der angekoppelten Partnersysteme absolut notwendig, da sie sonst nicht in den Testablauf einbezogen werden konnten. Die Ergebnisse wurden dann verglichen und Abweichungen bewertet.

Dieser Test mag zunächst einfach erscheinen, hat aber im Detail mit enormen Schwierigkeiten zu kämpfen. Der Datenbestand der Anwendung musste zum Beispiel vor jedem Test exakt auf den Zustand zurückgesetzt werden, der auch direkt vor Aufzeichnungsbeginn bestand. Bereits geringe Abweichungen konnten zu einer enormen Fehleranzahl führen, da es in der Regel zu einer großen Anzahl von Folgefehlern kam. Weiterhin mussten alle beteiligten Komponenten im Applikationsverbund ebenfalls simuliert werden. Auch hier musste exakt das gemessene Zeitraster eingehalten werden. Dazu wurden zum Teil dieselben Simulatoren wie bei den Lasttests verwendet. Die Uhrzeit des Systems musste mit dem Testdatenstrom synchronisiert werden. Ansonsten

hätten beispielsweise Anfragen nach Flugverbindungen zu einem bestimmten Tag statt einer positiven Antwort eine Fehlermeldung ergeben, da das gewünschte Abflugdatum der Originalnachricht in der Vergangenheit gelegen hätte.

Der Vergleich der Ergebnisse erwies sich ebenfalls als nicht einfach. Selbst bei identischen Testläufen war das Ergebnis nicht exakt gleich. Grund dafür ist, dass es auf einem Mehrprozessorsystem bei zeitgleich gestarteten Verarbeitungen mit synchronisierten Teilabläufen nicht deterministisch ist, in welcher Reihenfolge diese Abläufe im Mikrobereich durchgeführt werden. Daher wurden eingehende Nachrichten nicht immer in derselben Reihenfolge verarbeitet, was zu Unterschieden in den Nachrichtenprotokollen und in den Datenbeständen führte. So konnte nie nach Ablauf einer 24 Stunden Test-Session der exakt gleiche Zustand erreicht werden. Eine automatische Analyse, ob Unterschiede ausschließlich durch diese Effekte hervorgerufen wurden, erwies sich als schwierig. Letztendlich blieben immer einige Abweichungen, die ausschließlich per Hand bewertet werden konnten.

5.5 Ergebnisse der Tests

Durch die große Zahl unterschiedlicher Tests konnte eine sehr hohe Qualität der Software beim Rollout erreicht werden. Dabei deckte jede einzelne Testart Fehler auf, die bei anderen Tests nicht sichtbar wurden. Damit war jede Testart unverzichtbar. Dadurch, dass die Tests in einer bei dieser Applikation vorher nie erreichten Tiefe durchgeführt wurden, konnte auch eine überraschend große Zahl an Fehlern aufgedeckt werden, die bereits unter BS2000 im Anwendungssystem enthalten waren, hier aber nicht sichtbar wurden oder zum Teil keine Auswirkungen hatten. Bei der 1:1 Umstellung waren solche Fehler konsequenterweise auch identisch nach Unix migriert worden.

6 Zusammenfassung / Resümee

Im Rahmen des ARNO Projekts wurden viele Fragestellungen aufgegriffen und diskutiert, da wichtige Fachleute und Wissensträger der Firma beteiligt waren. Dies führte zu wesentlichen Verbesserungen der Softwareentwicklungsumgebung, des Konfigurationsmanagements, der Testsystematik und der Testsysteme. Auch wurde der Softwareentwicklungs- und Inbetriebnahmeprozess überdacht und optimiert. Diese Verbesserungen können noch nicht abschließend in Bezug auf Kosteneinsparungen und Geschwindigkeitserhöhung bei der Softwareentwicklung beurteilt werden.

Entscheidende Erfolgsfaktoren dieses komplexen und wahrscheinlich einzigartigen Projektes waren das konsequente Changemanagement der Projektanforderungen und der Änderungen an den umzustellenden Objekten während der Projektlaufzeit unter Verwendung eines Konfigurationsmanagementsystems. Weitere Erfolgsfaktoren waren die besondere Aufmerksamkeit des Topmanagements (Besetzung des Steering Committees), die sinnvolle Verteilung der Aufgaben (Werkzeuge außer Haus zu bauen und die Migration selbst durchzuführen) sowie die gute Zusammenarbeit in einem hoch motivierten Projektteam, in dem viele hervorragende Ideen entwickelt und auch umgesetzt wurden.

Literaturverzeichnis

[Den91] Denert, E.: Software-Engineering. Springer-Verlag Berlin, 1991, ISBN 3-540-53404-0.

[Par72] Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Communications of the ACM, Vol. 15, No. 12, pp. 1053-1058, 1972.

[Erd04] Erdmenger, U.: Der pro et con Migrationmanager. Softwaretechnik-Trends, Band 24, Heft 2, Mai 2004.

[EU07] Erdmenger, U., Uhlig, D.: Konvertierung der Jobsteuerung am Beispiel einer BS2000-Migration. Softwaretechnik-Trends, Band 27, Heft 2, Mai 2007.

[Gut77] Guttag; J.V..: Abstract data types and the development of data structures. Communication of the ACM, Vol. 20, No. 6, pp. 396-404, 1977.

[Kai03] Kaiser, U., Erfahrungen bei der Entwicklung von Werkzeugen zum Reverse Engineering. Softwaretechnik-Trends, Band 27, Heft 1, Februar 2007.

[KKW06] Kaiser, U., Kroha, P., Winter, A. (Hrsg.): 3. Workshop Reengineering Prozesse (RePro 2006) – Software Migration. Softwaretechnik-Trends, Band 27, Heft 1, Februar 2007.

[DeM97] DeMarco, T: Warum ist Software so teuer? Carl Hanser Verlag, München, Wien, 1997, ISBN 3-446-18902-5

[Sne03] Sneed, H.: Aufwandsschätzung von Reengineering Projekten. Softwaretechnik-Trends, Band 23, Heft 2, Mai 2003.

[Tep03] Teppe, W.: Redesign der START Amadeus Anwendungssoftware. Softwaretechnik-Trends, Band 23, Heft 2, Mai 2003.

[Tep06] Teppe, W.: ARNO: Migration von Mainframe Transaktionssystemen nach UNIX. 3. Workshop Reengineering Prozesse (RePro 2006) – Software Migration. Mainzer Informatik-Berichte, Informatik-Bericht Nr. 2/2006, Reprinted in: Softwaretechnik-Trends, Band 27, Heft 1, Februar 2007.