Workflows der Software-Migration

Rainer Gimnich
IBM Software Group
Enterprise Integration Solutions (EIS)
D-65936 Frankfurt
gimnich@de.ibm.com

Andreas Winter
Universität Koblenz-Landau
Institut für Softwaretechnik,
D-56016 Koblenz
winter@uni-koblenz.de

Zusammenfassung
Dieser Beitrag skizziert die grundlegenden Phasen der Software-Migration und überprüft diese anhand exemplarischer Migrationsvorhaben.

1. Einleitung

Neben den zentralen Software-Reengineering-Vorhaben wie (korrektive) Wartung, Systemerhaltung, Integration, Erweiterung, Sanierung, Redokumentation, oder Ablösung [Sneed et. al. 2004, Winter 2004] nimmt die Software-Migration eine hervorgehobene Rolle ein. Die Software-Migration kristallisierte sich auch als zentrales Diskussionsthema des Workshops Reengineering Prozesse (RePro) heraus, der im Oktober 2004 in Koblenz stattfand [RePro 2004]. Dieser Beitrag skizziert nach einer Begriffsklärung, aufbauend auf den Diskussionen des Workshops, die wesentlichen Workflows von Migrationsvorhaben und zeigt die Anwendung dieser Workflows in konkreten Migrationsprojekten.

2. Migration

Migration bezeichnet die Überführung eines Softwaresystems in eine andere Zielumgebung. Migrationen sind rein technische Transformationen mit einer klaren Anforderungsdefinition. Das zu migrierende Altsystem beschreibt eindeutig die Systemfunktionalität, deren Erhalt nach erfolgreicher Migration durch Regressionstests überprüft werden kann. Die hiermit verbundene einfachere Überprüfung von Projektergebnissen erleichtert auch eine Vergabe von Migrationsaufgaben an externe Dienstleister [Sneed et al. 2004].
Migrationen werden meist durch geänderte Anforderungen an Softwaresysteme ausgelöst. Organisatorische Änderungen der Unternehmensprozesse, Erweiterungen der Softwarefunktionalität oder externe Anforderungen erfordern Überarbeitungen und Erweiterungen der Softwaresysteme. Ohne die Überführung des bestehenden Softwaresystems in eine neue Umgebung sind diese Anforderungen nicht oder nur mit großem Aufwand realisierbar. Sich aus solchen Anforderungen ergebende *nicht-funktionale Anforderungen* an die Systemumgebung werden in Migrationsprojekten bearbeitet. Software-Migrationen sind als der Teil umfassenderer Reengineering-Maßnahmen zu verstehen, bei dem vorhandene Softwareartefakte in eine neue Umgebung überführt werden, ohne hierbei die Fachlichkeit zu verändern.
Auf technischer Ebene können Software-Migrationen die Änderung der Hardware-Umgebung, der Laufzeitumgebung, der Software-Architektur und der Software-Entwicklungsumgebung auslösen. Die *Hardware-Migration* umfasst den Wechsel der zu Grunde liegenden Hardwareumgebung wie z.B. den Wechsel einer Mainframe-Umgebung zu einer verteilen Unix-Umgebung. Die *Migration der Laufzeitumgebung* bezieht sich auf den Wechsel der dem System zugrunde liegenden Systemsoftware, wie beispielsweise Betriebssysteme oder verwendete Datenbankmanagement-Systeme. *Architektur-Migration* bezeichnet die grundlegende Änderung der Systemstruktur, wie sie beispielsweise beim Übergang von monolithischen Systemen zu Mehr-Schichten-Architekturen auftritt [Hasselbring et al. 2004]. Die Überführung der Entwicklungsumgebungen wie z.B. der Wechsel der Programmierumgebung von Assembler zu COBOL [Borchers/Moritz 2005] wird als *Migration der Entwicklungsumgebung* bezeichnet. Migrationen der Hardware-Umgebung, der Laufzeitumgebung, der Software-Architektur und der Software-Entwicklungsumgebung bedingen sich in der Regel auch gegenseitig.

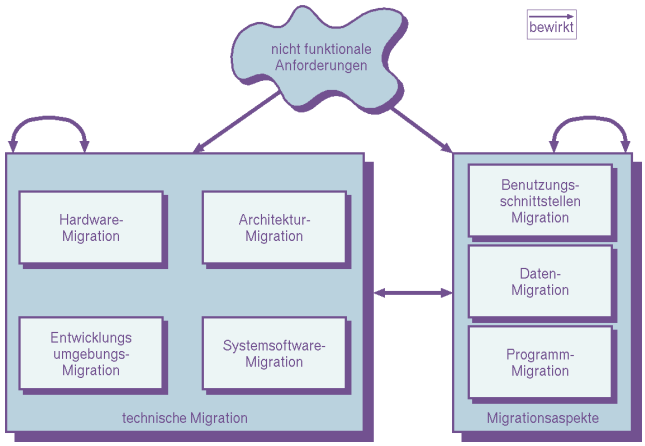

Abbildung 1 Migrationen

Je nach Zielsetzung des Reengineering-Vorhabens sind von einer Software-Migration folgende *Softwareaspekte* betroffen: Daten, Benutzungsschnittstellen und Programme [Sneed et al. 2004]. *Datenmigration* umfasst die Überführung der Datenbestände eines Systems in ein anderes System. Hierbei sind neben den eigentlichen Daten auch die den Daten zugrunde liegenden Datenstrukturen oder Schemata zu migrieren. *Programm-Migration* behandelt die Transformation der ausführbaren Programmlogik. *Benutzungsschnittstellen-Migration* umfasst die Übertragung der Interaktionskomponenten mit dem Nutzer. Auch die Migration dieser Aspekte bedingt sich gegenseitig. Abbildung 1

skizziert die Zusammenhänge zwischen den einzelnen Migrationsstypen.

3. Workflows der Migration

Referenz-Prozessmodelle zur Software-Migration ermöglichen die geplante, dokumentierte und kontrollierte Durchführung von Migrationsvorhaben. Die Bereitstellung von Referenzprozessen, basierend auf „Best Practices“, ermöglicht die zielgerichtete Entwicklung, Anpassung und Optimierung individueller Migrationsprozesse. Vorgehensmodelle zur Software-Migration aus der Praxis werden u.a. bei [Sneed 1999], [Collogia 2004], [RePro 2004] und [Sneed 2004] beschrieben. Diesen Prozessmodellen gemeinsam sind die im Folgenden kurz skizzierten *Migrations-Workflows*, die innerhalb eines Migrationsprojekts, analog der Workflows im Rational Unified Process [Kruchten 2000], iterativ und in unterschiedlichen Intensitäten in den Phasen *Konzeptualisierung*, *Entwurf*, *Konstruktion* und *Übergang* ausgeführt werden (vgl. Abbildung 2).

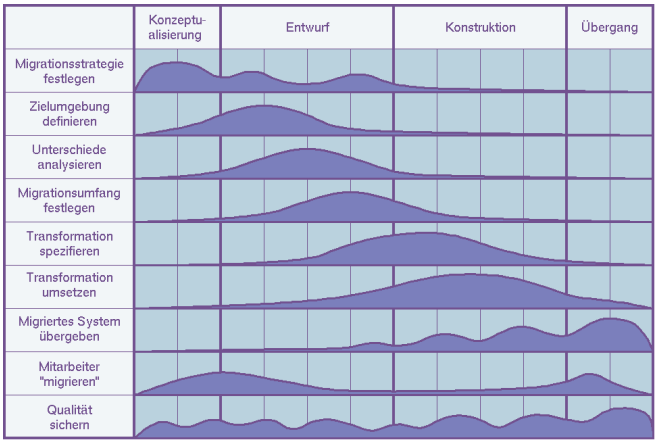

Abbildung 2 Workflows der Software-Migration

Migrationsstrategie festlegen

Grundsätzlich bieten sich drei Strategien, Altsysteme in neue Umgebungen zu überführen: *Neuentwicklung*, d.h. direkte Implementierung in der neuen Umgebung, *Kapselung (Wrapping)*, d.h. Belassen des Altsystems in seiner Umgebung und Bereitstellung von Zugriffsschnittstellen im Neusystem, und *Konversion*, d.h. direkte Übertragung oder (teil-)automatisierte Transformation in die Zielumgebung. In Abhängigkeit von der Qualität des vorliegenden Systems, der angestrebten Qualität des Neusystems und dem zur Verfügung stehendem Budget ist für jede zu migrierende Komponente das Umsetzungsverfahren festzulegen [Sneed 1999].

Neben der Entscheidung über die Umsetzung der Migration ist ebenfalls über die Art der Übergabe des migrierten Systems zu entscheiden: Bei der *Big-Bang-Migration* erfolgt die Ablösung des Altsystems durch das Neusystem auf einen Schlag. Bei der *„sanften“ Migration* wird das Altsystem inkrementell durch das Neusystem abgelöst [Brodie/Stonebraker 1995]. Mit Vorliegen weiterer Informationen über Alt- und Zielsystem sind die Migrationsstrategien an die neuen Erkenntnisse anzupassen.

Zielumgebung definieren

Bevor eine Migration durchgeführt wird, ist festzulegen, wie die neue Zielumgebung aussehen soll. Festzulegen sind hier insbesondere Hardwareumgebung, Systemsoftware, Entwicklungsumgebung und Software-Architektur sowie die hiervon betroffenen Aspekte der Benutzungsschnittstellen, Daten und Programme.

Unterschiede analysieren

Die Bewertung des Altsystems hinsichtlich der generellen Migrationsfähigkeit stellt eine wesentliche Voraussetzung zur Durchführung eines Migrationsprojekts dar.

Hierzu ist gegebenenfalls eine Redokumentation des Altsystems mit Methoden und Techniken des Reverse Engineering erforderlich. Unterschied zwischen Alt- und Zielsystem beziehen sich beispielsweise auf die Ansteuerung von Benutzungsschnittstellen, die Zugriffstechniken auf Daten und die Kopplung ausführbarer Programme durch Jobsteuerungen oder Programmaufrufe.

Migrationsumfang festlegen

Die qualitative und quantitative Analyse der zu migrierenden Softwareobjekte wie Benutzungsschnittstellen, Masken, Programme, Jobs, Dateien und Datenbanken und der hierzwischen vorliegenden Abhängigkeiten und Vergleiche mit früheren Migrationsprojekten erlauben Rückschlüsse über den Aufwand des Migrationsvorhabens [Sneed 2003]. Die Analyse der Systemunterschiede auf der Ebene einzelner Artefakte wird detailliert. Zentrale Kriterien sind hier insbesondere Funktionsumfang, Qualität und Wiederverwendbarkeit der relevanten Komponenten des Altsystems. Durch Verwendung Metrik-basierter Verfahren kann entschieden werden, ob einzelne Komponenten durch Neuentwicklung, Wrapping oder Konvertierung migriert werden. Im Rahmen von Migrationsvorhaben sollte ein möglichst hoher Anteil an Wiederverwendung angestrebt werden.

Transformationen spezifizieren

Vor der Durchführung der eigentlichen Umsetzungen des Altsystems in das Zielsystem sind die notwendigen Transformationen festzulegen. Je nach Migrationsvorgehen sind Konversionsregeln zu definieren, Kapselungen auf Basis geeigneter Middleware festzulegen, oder das Zielsystem aufbauend auf der Funktionalität des Altsystems zu spezifizieren. Hierbei ist eine Vielzahl unterschiedlicher Programmier-, Job-, Datenbank- und Maskensprachen zu berücksichtigen, für deren Transformationen nur in Spezialfällen Werkzeuge vorhanden sind [Sneed 1999]. Generische Programmtransformationsansätze, wie beispielsweise TXL [Cordy et al. 2002], basieren auf der Definition von Transformationsregeln auf Basis der Grammatiken von Quell- und Zieldokumenten. Generelle Transformationsansätze, deren Erprobung in konkreten Migrationsvorhaben jedoch noch aussteht, sind in [Czarnecki/Helsen 2003] skizziert.

Transformation umsetzen

Die eigentliche Umsetzung der Migration durch Konversion oder Kapselung der Programme und Daten erfolgt aufgrund der zuvor getroffenen Entscheidungen. Hierbei sollten iterativ jeweils zusammengehörige Module migriert werden. [Sneed et al. 2004] empfiehlt zunächst die Migration der Daten, anschließend die Migration der Programme und zum Schluss die Migration der Benutzungsoberflächen. Dieser Migrationsschritt erfordert insbesondere gute Kenntnis der Zielumgebung.

Migriertes System übergeben

Abhängig von der gewählten Migrationsstrategie erfolgt die komplette Übergabe des Zielsystems auf einen Schlag (big bang) oder inkrementell, in mehreren Schritten (sanft). Im letzten Fall existieren beide Systeme zeitweise nebeneinander, so dass der Parallelbetrieb mit geeigneten Maßnahmen synchronisiert werden muss.

Mitarbeiter „migrieren“

Zur Migration von Softwaresystem ist es erforderlich, dass sowohl die Mitarbeiter, die mit der Migration betraut sind, als auch die Mitarbeiter, die später mit dem migrierten System arbeiten, entsprechend qualifiziert werden. Für die Mitarbeiter in der Software-Entwicklung bezieht sich die Qualifikation neben den Methoden und Techniken zur Softwaremigration auch auf den Umgang mit Alt- und Zielsystem.

Qualität sichern

Qualitätssichernde Maßnahmen der Migration müssen sicherstellen, dass die funktionale Äquivalenz von Alt- und Zielsystem gewährleistet ist. Diese Aktivität gilt als größter Kostenfaktor eines Migrationsprojekts [Sneed et al. 2004]. Soweit teilautomatisierte Transformationen zur Konvertierung eingesetzt werden, sind diese ausführlich zu testen. Generell empfiehlt sich die Erprobung der Transformationen an geeigneten Programmen, Daten und Masken. Das Vorliegen des Altsystems erlaubt die Validierung der funktionalen Äquivalenz durch Regressionstests.

4. Praktische Relevanz

Das hier skizzierte, abstrahierte Vorgehensmodell für Software-Migrationen wird anhand von komplexen Projektbeispielen auf seine generelle Anwendbarkeit überprüft. Diese beinhalten unterschiedliche Migrationsstypen, die in jeweils unterschiedlicher Intensität verschiedene technische Migrationen betreffen und unterschiedliche Migrationsaspekte behandeln. Projektbeispiele:

Programm-zentrierte Migration: Umstellung zentraler Schlüsselnummern.

Daten-zentrierte Migration (Instanz und Schema): Migration einer dateibasierten Datenhaltung in eine relationale Datenbank.

Architektur-zentrierte Migration schrittweise Umstellung einer Mainframe-Realisierung in eine Service-orientierte Architektur [Gimnich 2005].

Die konkrete Bearbeitung der Migrations-Workflows ist in allen Beispiel-Projekten sehr unterschiedlich, sie lässt sich aber nach dem skizzierten Vorgehensmodell auf einheitliche Weise beschreiben. Der Workflow *Transformationen spezifizieren* wird etwa bei den Projekten zur Programm- und Daten-zentrierten Migration nur einmal durchlaufen. Die Spezifikation lässt sich z.B. über Legacy Transformation Patterns [Hess 2005] auf Architektur-, Daten- und Programmseite relativ einfach erstellen. Im Projekt zur Architektur-zentrierten Migration ist dagegen dieser Workflow naturgemäß mehrfach im Gesamtprojekt zu durchlaufen und wesentlich umfangreicher, da Geschäftsprozess, Architektur und Programme tiefgreifend betroffen sind.

5. Zusammenfassung und Ausblick

Wir haben auf der Basis von [RePro 2004] und eigenen Erfahrungen die Grundzüge eines Referenzmodells für Software-Migrationen vorstellt und deren praktische Verwendbarkeit und Nutzen für weitere Migrationsprojekte exemplarisch überprüft. Dieses Vorgehensmodell ist noch im Detail zu definieren und in der Breite zu validieren, wobei wir gern auf die Erfahrungen der RePro- und WSR-Teilnehmer zurückgreifen möchten.

Literatur

[Borchers/Moritz 2005] J. Borchers, Moritz: B. Genauigkeit von Aufwandsschätzungen in Reengineering-Projekten - Erfahrungen aus einer Sprachumstellung von Assembler nach COBOL, Informatik Forschung und Entwicklung, 2005.

[Brodie/Stonebraker 1995] M.L. Brodie, M. Stonebraker: Migrating Legacy Systems. Morgan Kaufmann, San Francisco, California, 1995.

[Czarnecki/Helsen 2003] K. Czarnecki, S. Helsen: Classification of Model Transformation Approaches, OOPSLA'03 Workshop on Generative Techniques in the Context of MDA, http://swen.uwaterloo.ca/~kczarnec/ECE750T7/czarnecki_helsen.pdf.

[Collogia 2004] collogia AG: Reengineering - Modernisierung von Altsystemen. Köln. Version 5.4. http://www.collogia.de/95.0.html.

[Cordy et al. 2002] J.R. Cordy, T.R. Dean, A.J. Malton and K.A. Schneider: Source Transformation in Software Engineering using the TXL Transformation System, Journal of Information and Software Technology 44(13)2002, pp. 827-837.

[Gimnich 2005] R. Gimnich: Nutzung von Legacy-Software in Service-orientierten Architekturen, SQM 2005, http://www.sqs-conferences.com/abstracts/gimnich.pdf.

[Hasselbring et al. 2004] W. Hasselbring, R. Reussner, H. Jaekel, J. Schlegelmilch, T. Teschke, S. Krieghoff: The Dublo Architecture Pattern for Smooth Migration of Business Information Systems: An Experience Report. 26th ICSE, 117-126, Edinburgh, Scotland, 2004.

[Hess 2005] H. M. Hess: Aligning technology and business: Applying patterns for legacy transformation. IBM Systems Journal, 44(1) 2005.

[Kruchten 2000] P. Kruchten: The Rational Unified Process, An Introduction. Addison-Wesley, Upper Saddle River, 2nd edition, 2000.

[RePro 2004] U. Kuhlmann, H. Sneed, A. Winter: Workshop Reengineering Prozesse, Fallstudien, Methoden, Vorgehen, Werkzeuge. Fachberichte Informatik, 11/2004, Universität Koblenz-Landau, 2004.

[Sneed 1999] H. Sneed: Objektorientierte Softwaremigration, Addison-Wesley, Bonn, 1999.

[Sneed 2003] H. Sneed: Aufwandsschätzung von Reengineering-Projekten, Wirtschaftsinformatik, 45(6)2003.

[Sneed et al. 2004] H. Sneed, M. Hasitschka, M. Teichmann: Software-Produktmanagement, Wartung und Weiterentwicklung bestehender Anwendungssysteme. Dpunkt Verlag, Heidelberg, 2004.

[Winter 2004] A. Winter: Software-Reengineering, Werkzeuge und Prozesse, Workshop der GI-Fachgruppe „Software-Wartung“, 15. Oktober 2004, Stuttgart http://www.iste.uni-stuttgart.de/se/people/opferkuch/ AK-Wartung/Positionspapiere/winter.pdf.