Ansätze und Verfahren der Datenmigration

Datenmigration im Kontext des Reengineering

Modul Software Reengineering 2015/2016

Fachbereich Informatik
Arbeitsbereich Softwarekonstruktion & Werkzeuge
Universität Hamburg

21. Mai 2015

Abstract
Die Erhaltung von Daten aus dem Kontext von Legacy-Systemen spielt eine entscheidende Rolle beim Reengineering von Software. Auswahl, Extraktion und Konvertierung vorhandener Daten in ein neues Umfeld sind nur einige der Tätigkeiten der Datenmigration.
Unterschiedliche Strategien zur Migration von Daten setzen unterschiedliche Schwerpunkte um diese ihrem Kontext entsprechend durchführen zu können. Die Betrachtung aus Sicht sowohl der Datenquellen als auch der umliegenden Anwendungssysteme bildet den Kern dieser Strategien. Unterschiedliche Strategien kommen dabei in verschiedenen Situationen zum Einsatz und besitzen jeweils eigene Vor- und Nachteile.
Darüber hinaus bildet die Datenmigration ein hohes Risiko für das Unternehmen. Um dieses Risiko zu reduzieren, existieren verschiedene Prozessmodelle. Diese versuchen, den Ablauf der Datenmigration in einzelne Phasen zu strukturieren und die jeweiligen Kernaufgaben zu identifizieren. Sie bilden die Grundlage für die Entscheidung, welche der unterschiedlichen Vorgehensweisen für die Datenmigration an effektivsten und mit dem geringsten Risiko durchzuführen ist.
Die Vorgehensweisen beleuchten dabei die technische Prozedur der Datenmigration vom Legacy-System zum neuen System. Zu entscheiden ist, ob eine schrittweise oder komplette Migration durchgeführt werden soll. Beide Ansätze eignen sich nicht für jedes System. Sie müssen für eine erfolgreiche Datenmigration mit Bedacht gewählt werden.
Ziel dieser Ausarbeitung ist es, die allgemeine Thematik der Datenmigration zu erläutern, sowie Strategien, Prozessmodelle und Vorgehensweisen vorzustellen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

1 Einleitung

Die Datenmigration spielt im Kontext des Software-Reengineering eine entscheidende Rolle. Basis vieler Softwaresysteme bilden dynamische Daten in unterschiedlichen Formaten und auf Basis unterschiedlicher Technologien. Geschäftsprozesse basieren häufig auf bestimmten Datenstrukturen und -schemata. Als Migration bezeichnet man dabei das Verlagern von Daten auf andere Medien, das Konvertieren in neue Formate und die Restrukturierung bereits vorhandener Daten [Morris, 2012]. Die Anpassung der umliegenden Software-Systeme spielt dabei eine wichtige Rolle [Henrard u. a., 2002].

Konkret werden im Kontext der Datenmigration vorhandene Daten für das Reengineering von Software verwertet. Wichtige Kunden- oder Geschäftsdaten müssen auch nach Reengineering, Anpassung und Neuentwicklung von Softwaresystemen weiterhin genutzt werden können. Ein strukturiertes Vorgehen bei der Analyse, Konvertierung, Portierung und Anpassung dieser Daten bildet die Grundlage der Datenmigration.

Unterschiedliche Strategien versuchen auf verschiedenen Ebenen, die Migration von Daten zu strukturieren. Im Kontext des Reengineering von Daten müssen nicht alleine Datenquellen und -schemata angepasst werden; Im Zusammenhang mit ursprünglichen Daten entwickelte Legacy-Systeme müssen neuen Datenquellen angepasst und entsprechend adaptiert werden [Henrard u. a., 2002].

Für Durchführung und Einführung der Datenmigration im Kontext von Legacy-Systemen existieren verschiedene Ansätze. Diese versuchen den Ablauf von Migrationsprojekten mittels Prozessmodellen zu strukturieren. Ein allgemeines Prozessmodell, welches auf jedes Unternehmen ohne entsprechende Anpassungen angewandt werden kann, existiert jedoch nicht. Nur wenn alle Gegebenheiten und Abläufe eines Unternehmens betrachtet werden, kann eine erfolgreiche Migration mit minimalen Risiko durchgeführt werden [Wu u. a., 1997b] [Ackermann, 2005].

Aus einer Vielfalt von Ansätzen und Strategien ergeben sich je entsprechende Vor- und Nachteile. Schon eine grobe Analyse zeigt Stärken und Schwächen der jeweiligen Methodiken. Unterschiedliche Ebenen für die Anwendung der Strategien lassen sich auf verschiedene Anwendungsbereiche und Beteiligte zurückführen.

Der Rest dieser Ausarbeitung ist wie folgt strukturiert: Kapitel 2 beschreibt die Datenmigration im Kontext des Reengineering und gibt Hinweise auf die Wichtigkeit und Relevanz der Durchführung. Kapitel 3 stellt den generellen Ablauf einer Migration mit jeweiligen Tätigkeiten vor. Kapitel 4 stellt Strategien der Datenmigration auf zwei Ebenen vor und beschreibt deren Vor- und Nachteile sowie deren Einsatzbereich und Kapitel 5 beschreibt das Vorgehen bei der Einführung der Ergebnisse der Datenmigration. Abschließend gibt Kapitel 6 eine bündige Zusammenfassung der Inhalte.

2 Datenmigration im Kontext des Reengineering

Als zentrale Rolle des Reengineering von Software versteht sich eine "Untersuchung (Reverse Engineering) und Änderung des Systems, um es in neuer Form zu implementieren" [Chikofsky und Cross, 1990]. In diesem Zusammenhang versucht das Konzept der Datenmigration eben dieses Ziel im Hinblick auf bereits vorhandene Daten zu erreichen. Geschäftskritische Daten bilden einen wichtigen Bestandteil vieler Softwaresysteme. Legacy-Systeme bauen häufig auf primitiven Mitteln zur Datenhaltung auf [Henrard u. a., 2002].

Unterschiedliche Ansätze der Datenmigration finden in spezifischen Kontexten Anwendung. Je nach Anwendungsgebiet der Migration existieren Techniken und Vorgehensweisen für die Datenmigration. Diese unterscheiden sich in den einzusetzenden Hilfsmitteln, Beteiligten sowie Auswirkungen auf andere Systemteile. Direkte Auswirkungen zeigen sich dabei in umliegenden Softwaresystemen, welche auf die Nutzung der Daten angewiesen sind.

Die Planung der Datenmigration stellt meist nur einen Teil des Reengineerings von Softwaresystemen dar. Prozessmodelle zur Durchführung von Migrationen in Projekten bilden diese Prozesse strukturiert ab. Einführungsstrategien bilden konzeptionell den Abschluss der Datenmigration. Nach erfolgten Analysen, Tests und entsprechenden Anpassungen in den betroffenen Systemen kann die Einführung der Änderungen anhand verschiedener Strategien erfolgen.

2.1 Datenmigration

> "Data migration is the selection, perparation, extraction, transformation and permanent movement of appropriate data that is of the right quality to the right place at the right time and the decommissioning of legacy data stores"
>
> [Morris, 2012, S. 7]

Eine Strukturierung von Daten außerhalb von Softwaresystemen bildet häufig den Kern wichtiger Geschäftsprozesse. Auf der zugrundeliegenden Dynamik externer Datenquellen skalieren die Aufgaben unterschiedlichster Softwaresysteme. Durch das Vorhalten von Daten in externen Quellen können Informationen zur Laufzeit der System dynamisch nachgeladen und persistiert werden.

Das Reengineering von Software beinhaltet meist nicht alleine eine Restrukturierung von Softwaresystemen. Zugrundeliegende Daten sind auf die Nutzung und den speziellen Kontext des Systems ausgelegt. Hohe Anforderungen an Stabilität und Verlässlichkeit machen diese Daten zu einer zentralen Komponente von Legacy-Systemen.

Anders als eine Portierung von Software oder Systemen beruft sich die Datenmigration somit auf das Verschieben und Restrukturieren von Daten in neuen Datenquellen.

Aus ihrer zentralen Rolle innerhalb von Legacy-Systemen leitet sich ein hoher Geschäftswert der Daten ab. Beispiele finden sich etwa in Form von Daten der Kunden einer Bank, dem Katalog von Warenhäusern oder einer Verwaltung von Stückgütern im Betrieb von Containerterminals.

Datenmigration erfasst vorrangig Daten im gröberen Sinn. Dies reicht von Datenbanken, Geschäftsdaten in Excel-Tabellen bis hin zu eigenen Datenformaten. Ziel der Migration ist vorrangig das Erhalten von Daten. Der Geschäftswert dieser Daten steht als zentrales Argument für die Modernisierung von Datenhaltung und -management. Neben der physischen Migration von Daten, der Aktualisierung von Systemsoftware und letztendlich der Modernisierung der Modellierung der Daten reichen zahlreiche Ansätze der Datenmigration. Die Integration der Migration erfordert eine Verzahnung mit den Prozessen des Reengineering und der kontinuierlichen Nutzung von Legacy-Systemen.

2.2 Exemplarisches Szenario

Um die Kernaufgaben und den Kontext der Datenmigration zu erläutern, lässt sich ein exemplarischer Anwendungsfall für die Datenmigration im Kontext von Legacy-Systemen anführen.

Kontext: Ein Versandhaus setzt seit Jahren ein in COBOL verfasstes Legacy-System sowie ein System zum Datenmanagement auf Basis von ISAM[1] ein. Die Aufgabe des Systems ist die Verwaltung von Lagerbeständen und Kundendaten. Die Datenmodellierung orientiert sich an etablierten Geschäftsprozessen.

Im Zuge einer Evolution des Unternehmens soll das Softwaresystem neu entwickelt werden. Um bestehende Geschäftsprozesse korrekt abzubilden, erfolgt eine umfassende Analyse. Auf Basis dieser Daten soll das neue System implementiert werden. Da sich das neue Anwendungssystem ähnlich dem bestehenden System verhalten soll, müssen bereits vorhandene Kunden- und Bestandsdaten übertragen werden. Zu diesem Zweck ist die vorhandene Datenhaltung an das neue System anzupassen. Neue technische und konzeptionelle Überlegungen erfordern unter Umständen neue Denkweisen bei der Nutzung von vorhandenen Kundendaten. Die Datenmigration schließt in diesem Kontext die Konvertierung und die Übertragung vorhandener Daten auf das neue System ein. Die Nutzung des neuen Systems kann so mit den migrierten Daten erfolgen.

[1] *Indexed Sequential Access Method*, eine Methode zum Indizieren von Daten für den effizienten Zugriff. Ursprünglich entwickelt vom IBM

2.3 Risiken der Datenmigration

Um die Relevanz der Datenmigration weiter hervorzuheben, erläutert [Morris, 2012] Probleme und Risiken, die im Zuge der Datenmigration häufig auftreten. Im Jahr 2011 lagen etwa 40% der Projekte zur Datenmigration außerhalb von Zeit und Kostenplan, oder schlugen gesamtheitlich fehl [Howard, 2011]. Zu den auftretenden Problemen gehören unter anderen:

- **Technologiezentrierung**
  Oft wird die Datenmigration lediglich als technisches Problem verstanden [Morris, 2012, S. 9] und dementsprechend ohne fachliche Betrachtung vorhandener Daten durchgeführt.

- **Mangel an Spezialisten**
  Die Analyse vorhandener Daten und die folgende Migration erfordert unter Umständen Spezialisten oder Analysten für eine diffundierte Analyse. Eine Synthese aus fachlichen und technischen Problematiken kann im Kontext der Datenmigration eine saubere Trennung zwischen geschäftlichen und technischen Problemen erschweren.

- **Unterschätzen der Datenmigration**
  Die Datenmigration wird häufig unterschätzt. Im Kontext von Legacy-Systemen ist die tatsächliche Tiefe durchzuführender Anpassungen potentiell unbekannt. Das Erfassen der dazu notwendigen Tätigkeiten kann so leicht ein tiefes Eindringen in vorhandene System erfordern.

- **Problemzuweisung**
  Die Beziehungen zwischen (Migrations-)Projekt und alltäglichem Geschäft sind selten eindeutig. Unkontrollierte Zuweisungen der Problematiken im Zuge der Migration zu jeweils einem der Beteiligen können unbeabsichtigt zur Rekursion bei der Zuweisung einzelner Problembereiche führen [Morris, 2012, S. 9].

Im Hinblick auf die genannten Problematiken wird die Wichtigkeit der koordinierten Durchführung der Datenmigration deutlich. Hinweise für die erfolgreiche Durchführung von Projekten zur Datenmigration finden sich etwa in [SAS, 2009], [Oracle, 2011]. Unterschiedliche Strategien, Vorgehensmodelle und Einführungsstrategien versuchen dabei, die Durchführung entsprechend zu strukturieren.

3 Generelles Vorgehen

Die Datenmigration stellt einen risikoreichen Aufgabenbereich dar. Aus diesem Grund sollte diese nicht ohne eine vorhergehende Planungsphase und ohne Struktur durchgeführt werden. Ein allgemeingültiges Verfahren mit welchem das Risiko der Datenmigration reduziert werden kann existiert jedoch nicht [Wu u. a., 1997b, S. 3]. Dennoch existieren verschiedene Ansätze, um die den Prozess der Migration zu strukturieren. Einer dieser Ansätze ist das von Klaus Haller, Florian Matthes und Christopher Schulz in Zusammenarbeit mit Unternehmen aus Automobil-Industrie und dem Finanzsektor entwickelte Prozessmodell für Datenmigration [Haller u. a., 2012, S. 2f.].

Dieses Prozessmodel unterteilt die Migration in vier Abschnitte: Die Initialisierung, die Migrationsentwicklung, dem Testabschnitt und die Umstellung auf das neue System. Für jeden dieser Abschnitte werden dabei die wichtigsten Tätigkeiten dargestellt [Haller u. a., 2012, S. 5f]. Je nach Unternehmen und der Migrationssituation können die Tätigkeiten variieren.

Während der Initialisierung muss die aktuelle Situation erfasst werden. Hierbei gilt es herauszufinden, um was für ein Legacy-System es sich handelt, wo und wie die Daten gespeichert sind, auf was für ein System migriert werden soll und welche Daten übertragen werden sollen [Haller u. a., 2012, S. 7]. Auf dieser Grundlage kann eine Entscheidung darüber getroffen werden, welche Strategien für die Migration verwendet werden können. Diese Information sind somit essentiell, um das weitere Vorgehen planen zu können und eine initiale Aufwandseinschätzung erstellen zu können [Haller u. a., 2012, S. 7]. Ebenfalls wird auf Grundlage der gesammelten Informationen eine Migrationsplattform eingerichtet. Mit Hilfe dieser Migrationsplattform werden im folgenden die benötigten Migrationswerkzeuge, -skripte usw. entwickelt und getestet [Haller u. a., 2012, S. 7].

Im Abschnitt zwei, der Migrationsentwicklung, wird zunächst die vorangegange Analyse der auf dem Legacy-System vorhandenen Daten vertieft. Zu diesem Zweck wird ein Backup des Legacy-Datenbestands auf die Migrationsplatform überspielt. Somit kann verhindert werden, dass während der Analyse Probleme oder Störungen auf dem Legacy-System auftreten [Haller u. a., 2012, S. 7]. Bei der Analyse ist festzustellen, welche Datenschemata und -formate auf dem Legacy-System vorhanden sind und auf dem angepassten oder neu entwickelten System vorhanden sein müssen. Besonders wichtig ist hierbei herauszufinden, worin sich diese Schemata voneinander unterscheiden. Dieses Wissen ermöglicht es, Migrationsskripte zu entwickeln, mit welchen die Daten automatisiert konvertiert und übertragen werden können [Haller u. a., 2012, S. 7f.].

Nachdem die Struktur der Daten analysiert worden ist, müssen die Daten selbst betrachtet werden. Man versucht auf diese Weise unvollständige, inkonsistente oder doppelte Datensätze zu identifizieren. Um Probleme im neuen System zu vermeiden, sollten solche Datensätze bereinigt werden. Die Bereinigung (engl.: *Cleansing*) kann entweder vor der Datenmigration auf dem Legacy-System durchgeführt werden, während der Migration

mit Hilfe der Migrationsskripte oder im Anschluss auf dem neuen System [Haller u. a., 2012, S 7f.]. Mittels dieser Bereinigung werden die inkonsisten und unvollständigen Datensätze repariert und redundante Datensätze entfernt [Rahm und Hai Do, 2010, S. 7f.].

Die in der Migrationsentwicklung entwickelten Skripte müssen anschließend im Testabschnitt ausgiebig auf ihre Funktionalität überprüft werden. Entsprechende Tests werden auch hier auf der Migrationsplattform ausgeführt [Haller u. a., 2012, S. 8f.]. Gefundene Fehler werden korrigiert, Migrationsskripte müssen überprüft werden. Dieser Prozess wird solange wiederholt, bis die Migrationsskripte fehlerfrei sind und deren Ergebnisse als korrekt validiert wurden [Haller u. a., 2012, S. 8f.].

Abschließend folgt die Umstellung auf das neue System. Dieser Abschnitt beinhaltet die Migration vom Legacy- auf das neue System mit Hilfe der zuvor entwickelten Migrationsskripte. Je nach gewählter Vorgehensweise bleibt das Legacy-System bis zum Abschluss der Migration und Sicherstellung der Funktionalität des zukünftigen Systems im Einsatz [Bisbal u. a., 1999, S. 107].

Nachdem die Datenmigration abgeschlossen wurde, ist es in der Regel sinnvoll, einen Erfahrungsbericht zu erstellen. In diesem soll festhalten werden, in welchen Phasen der Migration es zu Problemen kam und wie diese behoben wurden. Auf diese Weise soll der Migrationsprozess im Unternehmen aktiv verbessert werden. Ebenfalls soll sichergestellt werden, dass in zukünftigen Migrationsprojekten nicht dieselben Probleme auftreten [Haller u. a., 2012, S. 10].

4 Strategien zur Datenmigration

Als Kernelement der Datenmigration stehen unterschiedliche Strategien zur Verfügung [Henrard u. a., 2002]. Datenbanken und -quellen bilden die grundlegende Ebene. Schemata der Datenhaltung müssen angepasst und konvertiert werden. Auf Ebene der Anwendung müssen in Folge der Migration der Daten umliegende Softwaresysteme mit Zugriff auf diese Daten ihre Nutzung der Datenquellen anpassen. Da unter Umständen Schnittstellen und Datenformate verändert wurden, muss die spezifische Nutzung dieser der Evolution auf Datenbankebene angepasst werden.

Einzelne Strategien bieten Vor- und Nachteile. Sie sind von unterschiedlichen Gruppen innerhalb eines Unternehmens oder Migrationsprojektes vorzunehmen.

4.1 Datenbankebene

Auf Ebene der Daten erfolgt sinnvollerweise eine Anpassung von zugrundeliegenden Technologien, Datenbankschemata oder Datenformaten. Gründe für eine Migration auf dieser Ebene finden sich auf Ebene der Hintergründe des zugrundeliegenden Reengineerings. Sie reichen von einem Wechsel auf zukunftssichere Technologien oder einer Anpassung von Geschäftsprozessen. Unterschiedliche Ansätze begünstigen oder hemmen dabei das Erreichen eben dieser Ziele.

4.1.1 Physisch

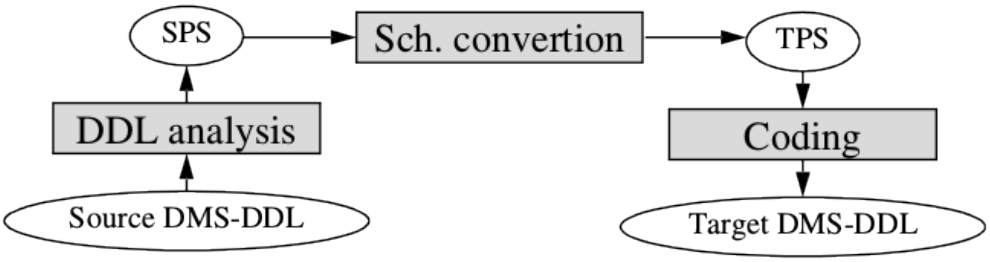
Die Migration von Daten auf physischer Ebene schließt die Konvertierung von Datenformaten und -schemata auf rein struktureller Ebene ein. Daten werden dabei ohne Betrachtung der semantischen Zusammenhänge auf neue Formate oder Technologien übertragen. Beispiel ist etwa die Abbildung relationaler Datenmodelle auf Objekt-orientierte Schemata [Alhajj und Polat, 2001], [Behm u. a., 1997]. Dabei werden Daten und ihre Modellierung in ein neues Schema übertragen, fachlich dennoch weitestgehend unverändert übernommen und nicht hinterfragt.

Grundsätzlich muss zunächst das vorhandene Format der Daten analysiert werden. Aus den gewonnenen Erkenntnissen kann die Übersetzung auf neue Schemata und Strukturen erfolgen. Sind, etwa beim Wechsel eines Datenbankformates in ein anderes, keine äquivalenten Konstrukte vorhanden, müssen Strukturen, welche diesen am ähnlichsten sind, gewählt werden.

Sind Datenformate und Schemata konvertiert und angepasst, müssen vorhandene Daten in die neue Umgebung eingefügt werden. In diesem Schritt ist das Kopieren der Daten aus dem ursprünglichen Format in die neue Umgebung vorgesehen. Unter Umständen müssen Datensätze konvertiert werden. Skripte und Tools zur Unterstützung begleiten diesen Prozess [Henrard u. a., 2002].

Abbildung 1 zeigt am Beispiel von Datenbanken eine physische Konvertierung. Das vorliegende DDL[2]-Schema des Quell-DMS[3] wird analysiert. Die entstandene Abbildung des SPS (*Source Physical Schema*) wird in ein TPS (*Target Physical Schema*) konvertiert. Diese Übersetzung des Schemas muss letztendlich innerhalb des Ziel-DMS kodiert werden [Henrard u. a., 2002].

Abbildung 1: Physische Konvertierung



Quelle: [Henrard u. a., 2002], Abbildung 2

Durchführung auf physischer Ebene sind meist elementare Konvertierungen und müssen nicht zwangsweise von Analysten oder fachlich geschultem Personal vorgenommen werden. Die rein physische Konvertierung kann automatisiert durchgeführt werden [Abiteboul u. a., 1999].

Nachteil der physischen Konvertierung ist ein fehlender Mehrwert der entstandenen Daten [Henrard u. a., 2002]. Vorhandene Schemata werden soweit wie möglich auf andere Technologien übertragen. Eine fachliche und konzeptuelle Analyse der Daten bleibt aus. In diesem Fall stellt die Migration der Daten lediglich ein Umkopieren in ein neues Datenformat dar.

Die rein physische Migration eignet sich vor allem bei einem Wechsel der technischen Umgebung der Daten, etwa einem Wechsel der Version des verwendeten DMS oder einem Wechsel des Herstellers.

4.1.2 Konzeptuell

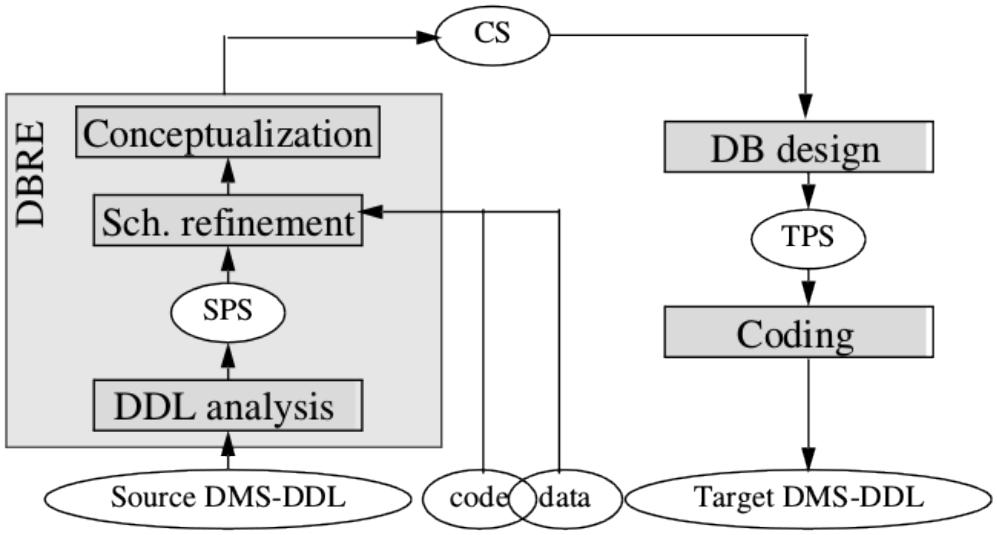
Anders als auf physischer Ebene werden Daten auf konzeptueller Ebene auch semantisch betrachtet. Schemata und Datenformate werden evaluiert und neu konzeptioniert. Aus der fachlichen Betrachtung der Daten ergeben sich unter Umständen umfangreiche Änderungen an den Schnittstellen der Systeme. Fachliche Konvertierungen erfordern ein Umdenken. Dieses kann die fachliche Qualität der migrierten Daten verbessern.

Die konzeptuelle Migration der Daten führt auf vorhandenen Schemata und Datenformaten eine semantische Analyse durch. Anhand der aus dieser gewonnenen Informationen

[2] *Data Definition Language* - Sprache zur Beschreibung von Datenformaten
[3] *Data Management System* - Datenbank-Verwaltungs-System

wird eine Neukonzeptionierung der Modellierung vorgenommen. Abbildung 2 zeigt das Vorgehen während der konzeptuellen Migration. Das Quell-DMS Schema wird mithilfe eines Database Reengineering (DBRE) analysiert [Henrard u. a., 2002]. Ziel dieses Vorganges ist die Bereinigung vorhandener Daten [Rahm und Hai Do, 2010] [Hernández und Stolfo, 1998]. Auf Basis einer neuen Konzeptualisierung der Schemata wird die Konvertierung ähnlich zur physischen Migration vorgenommen. Neue Formate werden in das Ziel-Format übertragen und kodiert. Die Konvertierung und Migration der eigentlichen Daten in die neue Umgebung erfolgt ebenfalls mithilfe von Konvertierungen. Diese ist, im Gegensatz zur physischen Konvertierung, umfangreicher. Neue Abbildungen und Modellierungen müssen aus vorhandenen Daten abgeleitet werden. In diesem Zusammenhang spielt auch das Reengineering der Anforderungen an die zugrundeliegenden Daten eine Rolle [Aiken u. a., 1994].

Abbildung 2: Konzeptuelle Konvertierung



Quelle: [Henrard u. a., 2002], Abbildung 2

Auf konzeptueller Ebene spielt die fachliche Betrachtung vorhandener Daten eine wichtige Rolle. Das Verfahren ist nicht zwangsweise trivial. Analysen und konzeptuelle Konvertierung benötigen dabei ein gewisses Verständnis der Semantik entsprechender Daten.

Eine konzeptuelle Migration von Daten sorgt vor allem für eine saubere Portierung vorhandener Daten. Schemata werden semantisch korrekt an neue Umgebungen und Formate angepasst. Eine potentielle Säuberung der Daten verringert unter Umständen Redundanzen oder nicht benötigte Datensätze und -formate.

Der für eine konzeptuelle Migration notwendige Aufwand ist vergleichsweise hoch. Benötigte Spezialisten und erforderliches fachliches Verständnis von Daten und Umgebung sorgen eventuell für hohe Kosten.

4.2 Anwendungsebene

Sind Datenformate und Daten selbst konvertiert und migriert, müssen umgebende Softwaresysteme angepasst werden oder bei Neuentwicklung entsprechend konzipiert werden. Veränderte Datenformate beziehungsweise Technologien setzen oft einen veränderten Zugriff auf diese Daten voraus. Um den Aufwand für Änderung innerhalb der nutzenden Anwendungen möglichst zu reduzieren, existieren drei Strategien der Anpassung von Anwendungen. Die zeichnen sich durch unterschiedliche Ebenen der Anpassung aus. Je nach Tiefe der Anpassung liegen ihnen unterschiedliche Ansätze zugrunde.

4.2.1 Wrapper

Um den Aufwand für Änderungen innerhalb von Softwaresystemen gering zu halten, bietet sich die Implementation eines Wrappers oder Adapters an. Dieser stellt eine Schnittstelle zwischen dem ursprünglichen, unveränderten Anwendungssystem und der neu strukturierten Datenquelle dar. Der Wrapper wird dabei als zusätzliche Schicht zwischen beiden Komponenten eingefügt. Er nimmt Aufrufe der ursprünglichen Anwendung entgegen, übersetzt sie in Anfragen an die neue Datenquelle und konvertiert die Ergebnisse der Anfrage für das ursprüngliche System. Er fungiert als Vermittler zwischen beiden Systemen und erspart so den Aufwand umfangreicher Änderungen für Abfragen im Anwendungssystem.

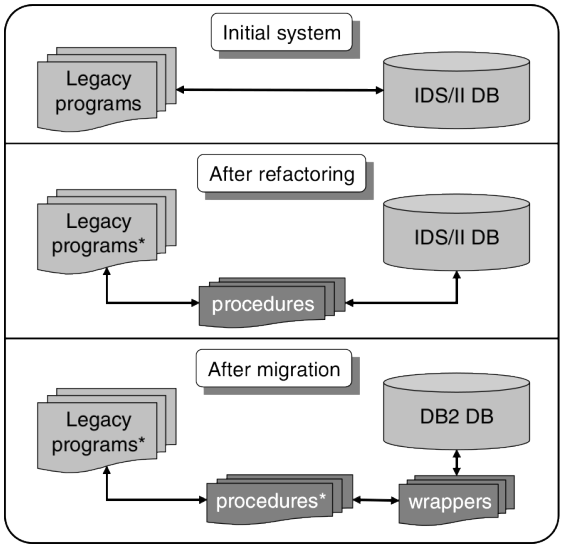
Abbildung 3 zeigt das Schema zur Nutzung eines Wrappers. Die Abstraktion erfolgt dabei auf Basis von Prozeduren zum Aufruf der neuen Datenquelle durch den Wrapper. Vor der Einbindung des Wrappers müssen diese in die Legacy-Anwendung eingefügt werden. Nur so kann der spätere Aufruf des Wrappers anstatt der ursprünglichen Datenquelle erfolgen Henrard u. a. [2008].
Ein fiktives Legacy-System dient der Illustration. Zu seinen Aufgaben gehört der Zugriff auf Kundendaten. Diese werden im Legacy-System mit naiven Mitteln zur Datenhaltung verwaltet [Henrard u. a., 2002]. Beispiel 1 zeigt den ursprünglichen Aufruf der Datenquelle. Das nachfolgende Beispiel 2 illustriert den neuen Zugriff auf den Wrapper. Die Anpassung an dieser Schnittstelle ist gering [Henrard u. a., 2002][4].

Quellcode-Beispiel 1: Beispiel Legacy-Programm mit Aufruf der ursprünglichen Datenquelle

```
READ PRODUCT KEY IS PROD-CODE
    INVALID KEY GO TO ERR-123.
DELETE PRODUCT END-DELETE
```

[4] Die Quellcode-Beispiele auf den folgenden Seiten sind [Henrard u. a., 2002] entnommen und dienen lediglich der Illustration

Abbildung 3: Einsatz eines Wrappers auf Anwendungsebene



Quelle: [Henrard u. a., 2008], Abbildung 1

Quellcode-Beispiel 2: Beispiel Legacy-Programm mit Aufruf der neuen Datenquelle

```
CALL WR-ORD-MNGMT
    USING "READKEY","PRODUCT","PROD-CODE", PRODUCT,WR-STATE.
IF STATUS OF WR-STATE NOT = 0 THEN
    GO TO ERR-123.
CALL WR-ORD-MNGMT
    USING "DELETE","PRODUCT","", PRODUCT, WR-STATE.
```

Deutlicher Vorteil in der Verwendung eines Wrappers ist die Tatsache, dass die Änderung des Anwendungssystems eher gering ausfällt. Der Einsatz eines Wrappers gefährdet die Integrität des nutzenden Softwaresystems nur minimal. Das Risiko wird dabei reduziert [Henrard u. a., 2002].

Durch Implementation eines Wrappers entsteht zusätzlicher Aufwand für eben diese Implementation und Integration des Wrappers in die bestehende Systemlandschaft. Die Einführung eines Wrappers ist häufig eine Übergangslösung. Ohne entsprechendes Verständnis des Anwendungssystems ist die Einführung eines Wrappers sinnvoller als tiefgreifende Änderungen im System [Henrard u. a., 2002].

Erfolgt der Zugriff auf eine Datenquelle an vielen Stellen im Quellcode der Legacy-Anwendung, ist die Einführung eines Wrappers mit vergleichsweise wenig Aufwand verbunden. Anders als die Anpassung von Aufrufen oder der Veränderung der Programmlogik, reduziert sich die Einführung eines Wrappers auf das Austauschen der Aufrufe der Datenquelle. Verarbeitung und Nutzung der Datenquelle werden vom Wrapper wie in der ursprünglichen Quelle bereitgestellt. Die relativ simple Anpassungen beim Einsatz eines Wrappers und ein sehr geringer Aufwand für die Anpassung im Legacy-System selbst sind die Vorteile bei Nutzung eines Wrappers. Entgegen dem geringen Aufwand

im System selbst, erfordert die Implementation eines Wrappers entsprechendes technische Verständnis der neuen Datenquelle. Dies garantiert die korrekte Nutzung der Daten durch die Legacy-Anwendung. Durch den bestehenden Umfang zur Nutzung durch das Legacy-System können technische Möglichkeiten der neuen Datenquelle unter Umständen nicht genutzt werden.

4.2.2 Statements

Eine detailliertere Anpassung der Anwendungssysteme stellt das Anpassen der Aufrufe der Datenquelle dar. Der Zugriff auf Daten erfolgt durch entsprechende Anweisungen im Quellcode einer Anwendung. Die Anpassung entsprechender Statements im Quellcode nimmt dabei die Anpassung des Anwendungssystems vor. Anpassungen in entsprechenden Statements sind der Kern dieser Strategie. Es wird lediglich der tatsächliche Zugriff auf die Datenquelle verändert, nicht jedoch seine Struktur. Sind umfassende Änderungen im Bezug auf die eingesetzte Technologie durchgeführt worden, reicht eine bloße Anpassung der Statements unter Umständen nicht aus.

Beispiel 3 zeigt den Aufruf der Datenquelle vor Anpassung der Statements. Die Anpassung in Beispiel 4 verdeutlicht die Anpassung. Wo zuvor der Zugriff mit Sprachmitteln des Legacy-Systems erfolgte, sind Statements nach der Anpassung auf die Nutzung von SQL ausgelegt. Die entsprechenden Anpassungen sind umfangreicher als die Nutzung eines Wrappers [Henrard u. a., 2002].

Quellcode-Beispiel 3: Aufruf der Datenquelle vor Anpassung der Statements

```
    MOVE CUS-CODE TO ORD-CUSTOMER.
    START ORDER KEY >= ORD-CUSTOMER.
    MOVE 0 TO END-FILE.
    PERFORM READ-ORD UNTIL END-FILE = 1.
READ-ORD SECTION.
BEG-ORD.
    READ ORDER NEXT
    AT END MOVE 1 TO END-FILE GO TO EXIT-ORD.
    <<processing current ORD record>>
EXIT-ORD.
    EXIT.
```

Quellcode-Beispiel 4: Aufruf der neuen Datenquelle: Zugriff via SQL

```
    EXEC SQL declare cursor ORD_GE_K1 for
    select ORD_CODE,ORD_CUSTOMER,ORD_DETAIL
    from ORDER where ORD_CODE >= :ORD-CODE
    order by ORD_CODE END-EXEC.
    ...
    EXEC SQL declare cursor ORD_GE_K2 for
    select ORD_CODE,ORD_CUSTOMER,ORD_DETAIL
    from ORDER where ORD_CUSTOMER >= :ORD-CUSTOMER
    ORDER BY ORD_CUSTOMER END-EXEC.
    ...
    MOVE CUS-CODE TO ORD-CUSTOMER.
    EXEC SQL open ORD_GE_K2 END-EXEC.
    MOVE "ORD_GE_K2" to ORD-SEQ.
    MOVE 0 TO END-FILE.
    PERFORM READ-ORD UNTIL END-FILE = 1.
READ-ORD SECTION.
BEG-ORD.
    IF ORD-SEQ = "ORD_GE_K1"
    THEN
    EXEC SQL fetch ORD_GE_K1 into :ORD-CODE,
    :ORD-CUSTOMER,:ORD-DETAIL END-EXEC
    ELSE IF ORD-SEQ = "ORD_GE_K2"
    THEN
    EXEC SQL fetch ORD_GE_K2 into :ORD-CODE,
    :ORD-CUSTOMER,:ORD-DETAIL END-EXEC
    ELSE IF ...
    END-IF.
    IF SQLCODE NOT = 0 THEN
        MOVE 1 TO END-FILE GO TO NO-ORD.
    <<processing current ORD record>>
EXIT-ORD.
    EXIT.
```

Vorteil der Anpassung von Statements ist eine technisch saubere Lösung. Anpassungen an den Abfragen, etwa Änderungen an SQL-Statements, sind eher simpel in der Durchführung. Die Änderung der Statements erfordert nicht zwangsweise ein tiefgreifendes Verständnis des Anwendungssystems.

Anpassungen an Statements reagieren auf eine veränderte Schnittstelle der Datenquelle. Größere strukturelle oder konzeptionelle Änderungen können jedoch nicht berücksichtigt werden.

Das Anpassen von Statements im Anwendungssystem ist sinnvoll, wenn an nur wenigen Stellen im Quellcode Anpassungen vorgenommen werden müssen. Ist etwa der Zugriff auf die Datenquelle gekapselt, ist die Anpassung nur in geringem Umfang durchzuführen. Im Gegensatz zum Einsatz eines Wrappers an der Schnittstelle ist die Änderung von Statements eine integrierte Lösung und greift in das Anwendungssystem ein. Um die Anpassung korrekt durchführen zu können, ist ein grundlegendes Verständnis der betreffenden Anwendung notwendig.

4.2.3 Logik

Um alle Möglichkeiten neuer Datenquellen und -technologien nutzen zu können, muss unter Umständen die Logik des zugreifenden Anwendungssystemes verändert werden. Dabei werden Datenstrukturen, Algorithmen und Zugriff innerhalb der Software massiv verändert, um auf die neue Umgebung zu reagieren [Henrard u. a., 2002].

Für eine Anpassung der Logik ist es erforderlich, entsprechende Passagen innerhalb der Anwendungssoftware an die neue Datenquelle anzupassen. Beispiel 5 zeigt den ursprünglichen Zugriff des Legacy-Systems. Die angepasste Logik ist in Beispiel 6 einzusehen. Als Unterschied zur reinen Anpassung der Statements sind die Auswirkungen auf andere Programmteile meist größer [Henrard u. a., 2002].

Quellcode-Beispiel 5: Urspüngliche Logik für den Zugriff auf die Datenquelle

```
DISP-ORD.
    READ ORDER KEY IS ORD-CODE
    INVALID KEY GO TO ERR-CUS-NOT-FOUND.
    PERFORM DISP-ORD-CUS-NAME.
DISP-ORD-CUS-NAME.
    MOVE ORD-CUSTOMER TO CUS-CODE
    READ CUSTOMER INVALID KEY
        DISPLAY "ERROR : UNKOWN CUSTOMER"
    NOT INVALID KEY
        DISPLAY "ORD-CODE: " ORD-CODE NAME.
```

Quellcode-Beispiel 6: Angepasste Zugriffslogik für Datenquelle in SQL

```
DISP-ORD.
    EXEC SQL
        SELECT O.CODE, C.NAME INTO :ORD-CODE, :NAME
        FROM ORDER O, CUSTOMER C
        WHERE O.CUS_CODE = C.CODE
            AND O.CODE = :ORD-CODE
    END-EXEC.
    IF SQLCODE = 0 THEN
        DISPLAY "ORD-CODE: " ORD-CODE NAME
    ELSE
        GO TO ERR-CUS-NOT-FOUND.
```

Die Anpassung der Logik und damit potentiell die Nutzung aller Möglichkeiten der neuen Datenquellen beziehungsweise Schnittstellen erhöht den Nutzen der Anwendungssoftware. Kann das Softwaresystem von den neuen Möglichkeiten Gebrauch machen, erhöht dies möglicherweise auch den Geschäftswert der Komponente.

Hinter der Anpassung der Logik eines Anwendungssystemes verbirgt sich immer ein gewisses Risiko. Je umfangreicher die Änderungen ausfallen, desto höher ist das Risiko. Gerade funktionale Änderungen, wie der Zugriff und die Arbeit mit Datenquellen, kann ein hohes Risiko beim Reengineering von Legacy-Systemen mit sich bringen.

Neben dem Einsatz eines Wrappers und der Anpassung von Statements bildet die Anpassung der Logik die umfangreichste Strategie auf Anwendungsebene. Die Umsetzung erfordert umfassendes Verständnis der fachlichen Logik des betreffenden Softwaresystems. Wie die bereits genannten Ansätze stellt sich auch die Anpassung der Logik durch Vor- und Nachteile dar. Unterschiedliche Strategien und Ansätze können die Migration auf unterschiedlichen Ebenen unterstützen. Welche Strategie einzusetzen ist, hängt stark vom Kontext, dem bestehenden Legacy-System und dem technischen und fachlichen Wissen der Beteiligten eines Migrationsprojektes ab.

5 Einführungsstrategien der Datenmigration

Das vom Unternehmen angewandte Prozessmodell beinhaltet ebenso die Wahl einer Einführungsstrategie, welche den technischen Ablauf und die Einführung der Datenmigration beschreibt. Da die gewählte Einführungsstrategie den weiteren Verlauf des Prozessmodells beeinflusst, wird diese in einer frühen Phase, nachdem die Kerndaten von Legacy- und neuem System analysiert wurden, ausgewählt. Eine spätere Revision dieser Entscheidung ist nicht oder nur sehr schwer möglich. Allerdings kann nicht jede Einführungsstrategie in jedem Kontext für die Durchführung einer Datenmigration genutzt werden. Im Folgenden werden zu diesem Zweck drei Einführungsstrategien mit ihren jeweiligen Vor- und Nachteilen vorgestellt.

5.1 Big-Bang-Ansatz

Der *Big-Bang* oder auch als *Cold Turkey* bekannte Ansatz ist wohl das älteste Model zur Durchführung einer Datenmigration. Nachdem das Legacy- und das neue System ausreichend analysiert und deren Unterschiede gefunden wurden, wird beim Big-Bang-Ansatz das Legacy-System abgeschaltet. Somit müssen auch sämtlicher vom Legacy-System gestützte Geschäftsprozesse unterbrochen werden [Wu u. a., 1997b, S. 4].

Während dieser Unterbrechung wird ein Dump des gesamten Datenbestands des Legacy-Systems angelegt[5]. Dieser wird anschließend mittels der hierfür entwickelten Migrationsskripte in das neue System übertragen [Brodie und Stonebraker, 1993, S. 3]. Bis die Datenmigration abgeschlossen ist, kann weder das neue noch das Legacy-System im Unternehmen eingesetzt werden. Daraus resultiert ein Zeitraum, in welchem das Unternehmen ohne ein lauffähiges System auskommen muss und somit stark in seiner Geschäftsfähigkeit eingeschränkt ist [Brodie und Stonebraker, 1993, S. 3f.].

Die Vorteile diese Ansatzes beschränken sich auf das verbesserte Programmverständnis, die Performance und die bessere Wartbarkeit des neuen Systems, nachteilig hingegen ist ein hohes Risiko bei Fehlschlag der Einführung [Bisbal u. a., 1999, S. 105]. Das hohe Risiko des Big-Bang-Ansatz besteht vor allem darin, dass der gesamte Datenbestand auf einmal migriert wird. Eventuell auftretende Fehler können somit erst Festgestellt werden wenn die Datenmigration abgeschlossen ist, wodurch der ganze Prozess wiederholt werden muss. Neben dem hohen Risiko besteht bei Nutzung Big-Bang-Ansatzes allerdings auch noch der Nachteil der zeitweisen Unterbrechung aller vom Legacy- bzw. dem neuen System unterstützten Geschäftsprozesse um die Datenmigration durchführen zu können [Wu u. a., 1997b, S. 4].

Zusammenfassend bietet der Big-Bang-Ansatz auf der einen Seite eine unkomplizierte Datenmigration. Auf der anderen Seite jedoch stehen ein hohes Risiko bei der Einführung sowie Ausfallzeiten während der Einführung der Änderungen. In diesem Zeitraum

[5] Ein *Dump* (Auszug) ist ein Abbild des Speicherzustandes etwa einer Datenbank zu einem bestimmten Zeitpunkt

können Geschäftstätigkeiten nur eingeschränkt oder gar nicht durchgeführt werden. Der Big-Bang-Ansatz findet aufgrund des hohen Risikos in der heutigen Zeit sehr selten Anwendung in Unternehmen.

5.2 Chicken-Little-Ansatz

Der Chicken-Little-Ansatz stellt eine weitere Einführungsstrategie zur Datenmigration dar. Entwickelt wurde dieser Ansatz von Michaell Brodie und Michael Stonebraker im Rahmen des 1991 begonnenen DARWIN-Projekts der University of California in Berkeley [Zoulafy, 2002]. Der Chicken-Little-Ansatz ist im Gegensatz zum Big-Bang-Ansatz ein inkrementelles Vorgehen zur Datenmigration. Durch das inkrementelle Vorgehen sollen Schwächen des Big-Bang-Ansatzes beseitigt werden. So können bei Chicken-Little-Ansatz beispielsweise Ausfallzeiten der von Legacy- bzw. neuen System unterstützten Geschäftsprozesse während der Datenmigration vermieden oder zumindest auf ein minimum reduziert werden [Zoulafy, 2002].

Das inkrementelle Vorgehen ermöglicht hierbei eine Entwicklung und Einführung aller Änderungen im Rahmen kleinerer Module. Entwickelt und eingeführt werden zunächst wenige Anpassungen. Inkrementell können weitere Funktionalitäten des Legacy-Systems und damit der migrierten Daten übertragen werden [Wu u. a., 1997b, S. 2].

Um auch während der Datenmigration die kontinuierliche Arbeit mit Softwaresystemen zu ermöglichen, kommt beim Chicken-Little-Ansatz ein sogenanntes Gateway zum Einsatz. Dieses Gateway verbindet Legacy- und das neue System nach Änderung oder Neuentwicklung miteinander und koordiniert deren Kommunikation untereinander. Somit können das neue und das Legacy-System solange parallel ausgeführt werden, bis das neue System alle Daten enthält und alle Funktionalitäten übernehmen kann [Wu u. a., 1997b, S. 2]. Die neuen Daten, die während der Zeit entstehen in der das neue System noch nicht alle Funktionen übernommen hat, werden dabei auf dem für diese Funktion zuständigen System gespeichert. Folglich werden Daten, welche durch bereits ans neue System übertragene Funktionen erstellt worden sind auch dort gespeichert und nicht mehr auf dem Legacy-System [Wu u. a., 1997b, S. 2]. In gewissen Kontexten ist es sinnvoll, Daten zeitweise in beiden Systemen zu pflegen. Ursprüngliche Schemata und Datenquellen werden erst verworfen, sobald zukünftige System diese Aufgaben verlässlich erfüllen können.

Das Gateway erfüllt nun zwei Aufgaben beim Übertragen der Daten an das neue System. Zum einen ist dies das Bereitstellen der noch nicht migrierten Daten an das neue System und zum Anderen das Bereitstellen der bereits migrierten Daten an das Legacy-System. Ersteres wird auch als umgekehrtes Gateway (engl.: *reverse Gateway*) und letzteres als Vorwärts-Gateway (engl.: *forward Gateway*) bezeichnet [Wu u. a., 1997b, S. 2]. Je nachdem, wie stark sich Datenbankschemata von Legacy- und neuem System unterscheiden, stellen vorwärts- und umgekehrtes Gateway unterschiedlich komplexe Module dar. Mit steigender Komplexität dieser Module sinkt allerdings auch die Performanz der Systeme

während der Migrationsphase [Bisbal u. a., 1999, S. 109].

Ähnlich dem Big-Bang-Ansatz, bietet der Chicken-Little-Ansatz die erwarteten Verbesserungen in Performance, Wartbarkeit sowie im Verständnis des Systems [Bisbal u. a., 1999, S. 108]. Allerdings fallen bei Einsatz des Chicken-Little-Ansatzes keine Ausfallzeiten, in denen weder mit Legacy- noch mit dem neuen System gearbeitet werden kann, an. Ermöglicht wird dies durch den Einsatz des Gateways, wodurch auch während der Migrationsphase mit dem gesamten System gearbeitet werden kann [Wu u. a., 1997b, S. 2]. Darüber hinaus bietet der Chicken-Little-Ansatz noch weitere Vorteile. Es muss beispielsweise nicht das gesamte Legacy-System auf einmal ersetzt werden. Stattdessen kann dieses Modul für Modul erneuert werden. Dies erleichtert Entwicklung bzw. Anpassung des neuen Systems [Brodie und Stonebraker, 1993, S. 3]. Die inkrementelle Anpassung des neuen Systems bietet auch Vorteile in der Fehlerbehandlung. So müssen bei Auftreten von Fehlern nur die entsprechenden Schritte wiederholt werden. Wenn etwa ein Problem bei der Datenmigration eines entwickelten Moduls auftritt, muss nach der Fehlerbehebung lediglich die Datenmigration von diesem Modul wiederholt werden. Durch die Fehlerbehebung gewonnene Kenntnisse können ebenfalls in den nächsten Schritten genutzt werden, um Probleme zu vermeiden, bevor diese entstehen [Brodie und Stonebraker, 1993, S. 3].

Neben den genannten Vorteilen hat der Chicken-Little-Ansatz auch einige Nachteile aufzuweisen. Zwar erlaubt es der Einsatz eines Gateways während der Migrationsphase, mit einem kombinierten System zu arbeiten, dafür stellt die Entwicklung des Gateway eine Herausforderung dar. Sie zieht unter Umständen hohe Kosten mit sich und erfordert eine hohes fachliches Verständnis beider Systeme. Daten, welche auf beiden Systemen vorhanden sind, müssen konsistent gehalten werden. Um die Interoperabilität der beiden Systeme zu ermöglichen, muss das Gateway semantische und technische Verknüpfungen beider Schemata herstellen. Je unterschiedlicher die Datenbankschemata von Legacy- und neuem System sind, desto komplexer erscheint diese Aufgabe [Wu u. a., 1997b, S. 2f.]. Darüber hinaus muss bei komplexeren Gateways auch mit Einbußen im Hinblick auf Performance gerechnet werden, wenn ein vorwärts- beziehungsweise umgekehrtes Gateway zum Einsatz kommt [Bisbal u. a., 1999, S. 109].

Alles in Allem betrachtet bietet der Chicken-Little-Ansatz viele Vorteile. Die inkrementelle Herangehensweise minimiert viele Risiken und Probleme, wie z.B. die Ausfallzeiten des Big-Bang-Ansatzes. Aus diesem Grund eignet sich Chicken-Little auch zum Einsatz in größeren Unternehmen. Dennoch muss bedacht werden, dass die Durchführung des Chicken-Little-Ansatzes auch schnell sehr komplex werden kann.

5.3 Butterfly-Ansatz

Im Rahmen des MILESTONE-Projekt, welches 1996 startete, wurde in einer Kooperation vom Trinity College Dublin, Broadcom Éireann Research, Telecom Éireann und Ericson der sogenannte Butterfly-Ansatz entwickelt [Wu u. a., 1997a, S. 202]. Der Butterfly-

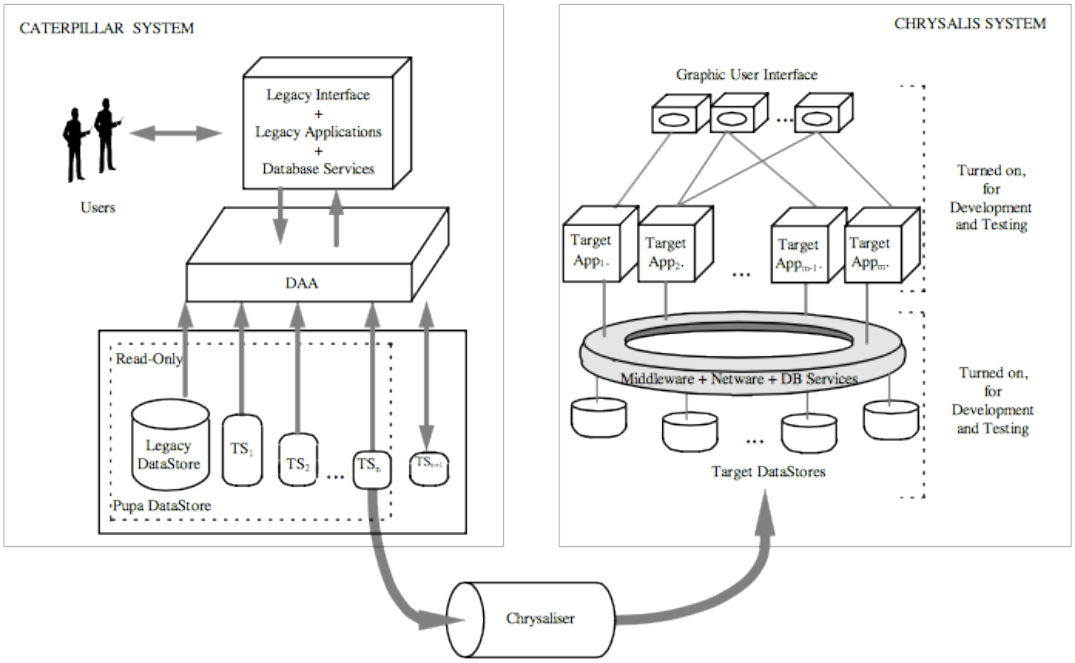
Ansatz benötigt keine Gateways, die zwischen den Systemen vermitteln [Wu u. a., 1997a, S. 202]. Folglich ist es auch nicht möglich, dass während der Datenmigration auf beiden Systemen gearbeitet werden kann. Allerdings ist diese Aussage nicht mit einer längeren Ausfallzeit, wie beim Big-Bang-Ansatz, gleichzusetzen. Stattdessen wird beim Butterfly-Ansatz mit temporären Datenspeichern gearbeitet, die die kontinuierliche Arbeit mit dem Legacy-System während der Datenmigration ermöglichen. Das neue System kommt erst zum Einsatz. sobald alle Daten auf dieses übertragen worden sind, bis dies geschehen ist, bleibt das Legacy-System im Einsatz [Wu u. a., 1997b, S. 3].

Zu Beginn der Datenmigration wird beim Butterfly-Ansatz der aktuelle Datenbestand des Legacy-Systems eingefroren. Ab diesem Zeitpunkt können auf dem Legacy Datenbestand nur noch Lese-Aktionen durchgeführt werden, jedoch können dort keine neuen Daten mehr hinterlegt werden [Wu u. a., 1997a, S. 202]. Um dennoch mit dem Legacy-System weiterarbeiten zu können, bedient sich der Butterfly-Ansatz eines temporären Speichers sowie eines *Data-Access-Allocator* (DAA). Dieser Data-Access-Allocator enthält Informationen darüber, wo welche Daten im eingefrorenen Datenbestand zu finden sind und welcher temporärer Speicher gerade aktiv ist [Wu u. a., 1997a, S. 202]. Wie in Abbildung 4 zeigt, wird der Data-Access-Allocator zwischen die Anwendungen und Dienste des Legacy-System und deren Datenbank geschaltet. Werden nun neue Daten ins Legacy-System eingegeben oder werden vorhandene bearbeitet, sorgt der Data-Access-Allocator dafür, dass das die entsprechenden Daten aus dem eingefrorenen Datenbestand gelesen werden und neue oder im aktuellen temporären Speicher abgelegt werden [Wu u. a., 1997a, S. 202].

Durch Data-Access-Allocator und temporäre Speicher wird sichergestellt, dass während der Datenmigration mit dem Legacy-System normal weitergearbeitet werden kann. Für die Übertragung der Daten auf das neue System wird allerdings noch ein *Daten-Transformator*, ein sogenannter *Chrysaliser* benötigt [Wu u. a., 1997a, S. 202]. Dieser Chrysaliser übernimmt die Rolle der Migrationsskripte. Er enthält die Informationen über die Datenbankschemata des Legacy- und des neuen Systems sowie die Schemata des Legacy-System, welche umgewandelt werden müssen [Wu u. a., 1997a, S. 202].

Mit Hilfe des Chrysaliser werden nun die Daten aus dem eingefrorenen Datenbestand an das neue System übertragen. Sobald alle Daten aus dem eingefrorenen Datenbestand übertragen wurden, wird ein weiterer temporärer Speicher `t2` angelegt und der Datenbestand des alte temporäre Speicher eingefroren. Anschließend muss der Data-Access-Allocator entsprechend angepasst werden [Wu u. a., 1997a, S. 202]. Der Chrysaliser kann nun damit beginnen, den temporären Speicher `t1` auf das neue System zu übertragen. Ist auch `t1` übertragen wird `t2` eingefroren und ein temporärer Speicher `t3` für neue im Legacy-System angelegte Daten angelegt (siehe Abbildung 4) [Wu u. a., 1997a, S. 202]. Mit jeder neuen Iteration des temporären Speichers reduziert sich dessen Größe. Dieser Umstand lässt sich darauf zurückführen, dass in der Zeit die für die Migration des initialen Datenbestands benötigt wird, der Datenbestand in der Regel nicht verdoppelt. Der

Abbildung 4: Datenmigration mittels des Butterfly-Ansatzes



Quelle: [Wu u. a., 1997b, S. 6], Abbildung 11

temporäre Speicher `t1` müsste somit auch kleiner sein als der initiale Datenbestand des Legacy-Systems. Analoges gilt für die folgenden Iterationen des temporären Speichers: $t_n < t_{n+1}$ [Wu u. a., 1997a, S. 202].

Dieser Vorgang wird solange wiederholt, bis der temporäre Speicher eine zu Beginn der Datenmigration festgelegte Größe erreicht. Wenn der temporäre Speicher diese Größe nun erreicht und es an der Zeit ist, diesen durch den Chrysaliser auf das neue System zu migrieren, wird dieser wie auch schon seine Vorgänger eingefroren. Allerdings wird kein neuer temporärer Speicher eingerichtet. Stattdessen wird ab diesem Zeitpunkt der Betrieb auf dem Legacy-System eingestellt damit die letzten Daten migriert werden können [Wu u. a., 1997a, S. 202]. Aus diesem Grund sollte die Größe des letzten temporären Speichers so gewählt werden, dass eine schnelle Migration von diesem aus möglich ist. Auf diese Weise kann die Ausfallzeit des Systems minimal gehalten werden [Wu u. a., 1997a, S. 202]. Nachdem die Datenmigration abgeschlossen ist, kann das neue System in Betrieb genommen und das Legacy-System endgültig abgeschaltet werden [Wu u. a., 1997a, S. 204].

Neben den allgemeinen Vorteilen, wie z.B. verbesserter Wartbarkeit, bietet der Butterfly-Ansatz eine minimale Beeinträchtigung durch Systemausfälle, in denen die vom System unterstützten Geschäftsprozesse nicht ausgeführt werden können. Durch die Wahl einer entsprechend kleinen Größe des letzten temporären Speichers kann die anfallende Aus-

fallzeit flexibel an die Bedürfnisse des Unternehmens angepasst werden [Wu u. a., 1997a, S. 204f.]. Darüber hinaus kann beim Butterfly-Ansatz im Gegensatz zu anderen Ansätzen die für die Migration benötigte Gesamtzeit relativ sicher anhand des initialen Datenbestands des Legacy-Systems sowie der Geschwindigkeit von Data-Access-Allocator und Chrysaliser eingeschätzt werden. Durch die Einschätzung des Zeitaufwands ist es dem Unternehmen möglich, die Migration besser zu planen und so unnötige Problematiken zu vermeiden. Beispielsweise kann verhindert werden, dass die Migration des System in eine Zeit fällt, in welcher das System stärker als normal beansprucht wird [Wu u. a., 1997a, S. 204]. Des Weiteren entfällt beim Butterfly-Ansatz die Notwendigkeit zur Entwicklung eines Gateways, um während der Datenmigration weiterarbeiten zu können, da das Legacy-System bis zum Abschluss im Betrieb bleibt. Dies bietet ebenfalls den Vorteil, dass keine Probleme mit der Konsistenz von Daten auftreten können [Wu u. a., 1997b, S. 3].

Allerdings hat der Butterfly-Ansatz auch Nachteile, welche bei der Entscheidung, welche Einführungsstrategie verwendet werden soll, berücksichtigt werden müssen. So muss zwar kein komplexes Gateway entwickelt werden, an dessen Stelle allerdings ein Data-Access-Allocator, welcher temporäre Speicher einrichtet und die Anfragen aus dem Legacy-System entsprechend umleiten kann. Ebenso ein Chrysaliser, welcher die Daten auf das neue System übertragen kann [Wu u. a., 1997b, S. 3]. Sowohl Data-Access-Allocator als auch Chrysaliser sind entscheidend dafür, wie schnell und sicher eine Migration mit dem Butterfly-Ansatz durchgeführt werden kann. Aus diesem Grund wird auch hier ein hohes technisches Verständnis für die Entwicklung benötigt [Wu u. a., 1997a, S. 204]. Des Weiteren kann es passieren, dass je nach zu migrierenden Legacy-System ein große Anzahl an temporären Speichern benötigt wird. Folglich kann es zu einem hohen Hardwarebedarf für Speicher kommen, wodurch wiederum die Kosten der Migration in die Höhe getrieben werden [Bisbal u. a., 1999, S. 109f.].

Zusammenfassend betrachtet stellt der Butterfly-Ansatz wie auch schon Big-Bang und Chicken-Little keine allgemeingültige Lösung dar. Auch hier muss beachtet werden, dass der Butterfly-Ansatz sich nicht zur Migration mit jedem Legacy-System eignet.

6 Fazit

Die Datenmigration im Kontext des Reengineering ist kein triviales Verfahren. Durch die Vielschichtigkeit von Unternehmen und Softwaresystemen eignen sich in unterschiedlichen Kontexten verschiedene Strategien. Als häufig unterschätztes Verfahren hat die Migration von Daten einen hohen Geschäftswert, etwa die Erhaltung geschäftskritischer Kundendaten. Mangelnde Planung und undefinierte Herangehensweisen können Projekte der Datenmigration scheitern lassen. Durch das Nutzen des Prozessmodells, das Nutzen von Strategien zur Durchführung und Einführungsstrategien kann die Migration strukturiert werden.

Sowohl die Migration der Daten selbst, als auch die Anpassung umliegender Anwendungen und Softwaresysteme spielt eine zentrale Rolle. Unterschiedliche Konzepte bieten jeweils fundierte Strategien für die Migration. Aspekte wie fachliches und technisches Wissen um Schnittstellen und Datenmodelle können Grundlage für die Auswahl einer Strategie sein. Der zukünftige Geschäftswert nach durchgeführten Änderungen kann bei der Auswahl ebenfalls eine Rolle spielen. So können neue technische Möglichkeiten nach Austausch der Datenbank-Plattform effizientere Schnittstellen bereitstellen und Zugriffe redundanzfrei gestalten.

Bei der Durchführung und bei der Wahl einer Strategie ist stets zu beachten, dass keine universelle Lösung existiert, welche grundlegend das beste Ergebnis liefert. Vielmehr muss für jedes Unternehmen jede mögliche Vorgehensweise genau betrachtet werden. So bietet sich beispielsweise der Chicken-Little-Ansatz für ein Unternehmen an, welches eine Legacy-Anwendungen Stück für Stück nacheinander ersetzen will, um das alltägliche Geschäft nicht zu gefährden. Wenn das Unternehmen allerdings plant, ein System komplett zu ersetzen und dabei Ausfallzeiten minimal halten will, eignet sich womöglich der Butterfly-Ansatz besser. Dies allerdings nur einer sehr geringen Anzahl von Fällen.

Um die richtige Entscheidung treffen zu können, muss immer die Gesamtsituation analysiert werden. Was für ein Unternehmen liegt vor, sollen nur die Daten auf ein neues System migriert werden oder muss das neue System auch erst noch entwickelt werden? Für diese Fragen und weitere müssen Antworten gefunden werden. Aus diesem Grund stellt ein geplantes Vorgehen eine wichtige Rolle für die erfolgreiche Datenmigration. Auch hierbei ist anzumerken, dass keine allgemeingültige Planung existiert und diese an entsprechende Gegebenheiten angepasst werden muss.

Die Durchführung der Datenmigration ist ein dynamisches Verfahren. Im speziellen Kontext von Unternehmen, Projektteam und vorhandenen Systemen sind zu nutzenden Strategien individuell abzustimmen. Gerade die Ungewissheit, welche Legacy-System häufig mit sich bringen, ermöglichen keine statische Auswahl einer Musterlösung für die Datenmigration.

Literatur

[Abiteboul u. a. 1999] Abiteboul, Serge ; Cluet, Sophie ; Milo, Tova ; Mogilevsky, Pini ; Zohar, Sagit: Tools for Data Translation and Integration. In: *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* (1999), S. 3–8

[Ackermann 2005] Ackermann, Ellen: *Ein Referenz-Prozessmodell zur Software-Migration*, Universität Koblenz-Landau, Abteilung Koblenz, Dissertation, 2005

[Aiken u. a. 1994] Aiken, Peter ; Muntz, Alice ; Richards, Russ: DoD Legacy Systems: Reverse Engineering Data Requirements. In: *Commun. ACM* 37 (1994), Nr. 5, S. 26–41

[Alhajj und Polat 2001] Alhajj, Reda ; Polat, Faruk: Reengineering Relational Databases to Object-Oriented: Constructing the Class Hierarchy and Migrating the Data. In: *Proceedings of the Eighth Working Conference on Reverse Engineering* (2001), S. 335–344

[Behm u. a. 1997] Behm, A. ; Geppert, A. ; Dittrich, K.R.: On the migration of Relational Schemas and Data to Object-Oriented Database Systems. In: *Proceedings of Re-Technologies in Information Systems* (1997)

[Bisbal u. a. 1999] Bisbal, J. ; Lawless, D. ; Wu, Bing ; Grimson, J.: Legacy information systems: issues and directions. In: *Software, IEEE* 16 (1999), Sep, Nr. 5, S. 103–111. – ISSN 0740-7459

[Brodie und Stonebraker 1993] Brodie, Michael L. ; Stonebraker, Michael: DARWIN: On the incremental migration of legacy information systems. In: *Distributed Object Computing Group, Technical Report TR-0222-10-92-165, GTE Labs Inc* (1993)

[Chikofsky und Cross 1990] Chikofsky, Elliot J. ; Cross, James H., II: Reverse Engineering and Design Recovery: A Taxonomy. In: *IEEE Softw.* 7 (1990), Nr. 1, S. 13–17

[Haller u. a. 2012] Haller, Klaus ; Matthes, Florian ; Schulz, Christopher: A Detailed Process Model for Large Scale Data Migration Projects. In: *Business Information Systems* Bd. 117. Springer Berlin Heidelberg, 2012, S. 165–176

[Henrard u. a. 2002] Henrard, Jean ; Hick, Jean-Marck ; Thiran, Philippe ; Hainaut, Jean-Luc: Strategies for Data Reengineering. In: *Proceedings of the Ninth Working Conference on Reverse Engineering* (2002)

[Henrard u. a. 2008] Henrard, Jean ; Roland, Didier ; Cleve, Anthony ; Hainaut, Jean-Luc: Large-Scale Data Reengineering: Return from Experience. In: *Proceedings of the 15th Working Conference on Reverse Engineering* (2008), S. 305–308

[Hernández und Stolfo 1998] Hernández, Mauricio A. ; Stolfo, Salvatore J.: Real-World Data is Dirty: Data Cleansing and the Merge/Purge Problem. In: *Data Mining and Knowledge Discovery 2(1)* (1998), S. 9–37

[Howard 2011] Howard, Philip: *Data migration snippets.* http://www.bloorresearch.com/analysis/data-migration-snippets/, Zuletzt aufgerufen am 21. Mai 2015. 2011. – Zuletzt aufgerufen: 21. Mai 2015

[Morris 2012] Morris, Johny: *Practical Data Migration.* 1. Auflage. British Informatics Society Ltd., 2012

[Oracle 2011] Oracle: *Successful Data Migration - Whitepaper.* http://www.oracle.com/technetwork/middleware/oedq/successful-data-migration-wp-1555708.pdf, Zuletzt aufgerufen am 21. Mai 2015. 2011

[Rahm und Hai Do 2010] Rahm, Erhard ; Hai Do, Hong: Data Cleaning: Problems and Current Approaches. In: *IEEE Computer Society Technical Committee on Data Engineering* (2010)

[SAS 2009] SAS: *Enhancing Your Chance for Successful Data Migration - Whitepaper.* http://www.sas.com/resources/whitepaper/wp_5969.pdf. 2009. – Zuletzt aufgerufen am 21. Mai 2015

[Wu u. a. 1997a] Wu, Bing ; Lawless, D. ; Bisbal, J. ; Richardson, R. ; Grimson, J. ; Wade, V. ; O'Sullivan, D.: The Butterfly Methodology: a gateway-free approach for migrating legacy information systems. In: *Engineering of Complex Computer Systems, 1997. Proceedings., Third IEEE International Conference on*, Sep 1997, S. 200–205

[Wu u. a. 1997b] Wu, Bing ; Lawless, Deidre ; Bisbal, Jesus ; Grimson, Jane ; Wade, Vincent: Legacy Systems Migration - A Method and its Tool-kit Framework. In: *Proceedings of the APSEC'97/ICSC'97: Joint 1997 Asia Pacific Software Engineering Conference and International Computer Science Conference* (1997), S. 312 – 320

[Zoulafy 2002] Zoulafy, Federico: *Issues and Challenges Facing Legacy Systems.* http://www.developer.com/mgmt/article.php/1492531/Issues-and-Challenges-Facing-Legacy-Systems.htm, Zuletzt aufgerufen am 21. Mai 2015. 2002