Franz Christoph

Software-Reengineering-Konzept zur Modernisierung von Legacy-Systemen

eingereicht als

MASTERARBEIT

an der

HOCHSCHULE MITTWEIDA (FH)

UNIVERSITY OF APPLIED SCIENCES

Mathematik / Physik / Informatik

Chemnitz, 2009

Erstprüfer: Prof. Dr.-Ing. Wilfried Schubert
Zweitprüfer: Dipl.-Ing. Sven Andrä
Dipl.-Ing. Andreas Jaehnert

Vorgelegte Arbeit wurde verteidigt am:

Bibliographische Beschreibung:

Christoph, Franz:
Software-Reengineering-Konzept zur Modernisierung von Legacy-Systemen. - 2009. - 230 S. - Mittweida, Hochschule Mittweida (FH), Fachbereich Mathematik / Physik / Informatik, Masterarbeit, 2010

Referat:

Legacy-Systeme sind auch im einundzwanzigsten Jahrhundert fester Bestandteil in Unternehmen, welche Wartungs- und Pflegemaßnahmen erschweren und den heutigen Anforderungen nach Integration, Wiederverwendung und Geschäftsprozessabbildung nicht mehr gerecht werden. Zur Umsetzung neuer unternehmensweiter Strategien, ist die Modernisierung von Legacy-Systemen zu zeitgemäßen Zielarchitekturen und Technologien notwendig.

Auf Basis verschiedener Aktivitäten des Software Reengineering und auf Basis der Auswertung einer Marktstudie über Software-Modernisierung wird untersucht, wie ein Konzept aussieht, um Legacy-Systeme, die mit prozeduralen Programmiersprachen implementiert wurden, in ein Zielsystem mit Drei-Schichten-Architektur zu überführen. Anhand dieser Untersuchungen wird das 4-Phasen-Transformationskonzept entwickelt, welches Techniken und Vorgänge vorstellt, die einen Modernisierungsprozess unterstützen. An einem Fallbeispiel wird die Anwendbarkeit des 4-Phasen-Transformationskonzepts gezeigt.

Danksagung

An dieser Stelle möchte ich mich bei meinen Betreuern und all denjenigen bedanken, die mich bei der Anfertigung dieser Arbeit unterstützt haben.

Ein ganz besonderer Dank gilt meinen Betreuern, Herrn Prof. Dr.-Ing. Wilfried Schubert von der Hochschule Mittweida (FH), Herrn Dipl.-Ing. Sven Andrä und Herrn Dipl.-Ing. Andreas Jaehnert von der IBM Deutschland Enterprise Application Solutions GmbH Chemnitz. Die Zusammenarbeit und Unterstützung im Rahmen dieser Arbeit war sehr lehrreich und vorbildlich. Zudem möchte ich mich bei Herrn Prof. Dr. Konrad Schulz, von der Hochschule Mittweida (FH), für die Unterstützung in stilistischen Angelegenheiten und dem Sprachgebrauch bedanken.

Für die gute fachliche Zusammenarbeit und der Bereitstellung von Informationen danke ich Herrn Karl-Heinz Homilius und Herrn Stefan Dautz von der IBM Deutschland Enterprise Application Solutions GmbH Chemnitz.

Für die fachliche Unterstützung, im Rahmen der Modellierung und des Umgangs mit dem Software-Modellierungswerkzeug objectiF, möchte ich mich bei dem ehemaligen Studenten der Hochschule Mittweida (FH) Herrn Steffen Bönsch bedanken.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

A2A	Application-to-Application
API	Application Programming Interface
B2B	Business-to-Business
CIM	Computational Independent Model
COBOL	Common Business Oriented Language
COREM	Capsule Oriented Reverse Engineering Method
CORET	Capsule Oriented Reverse Engineering Technique
CSD	Computer-Systemdienste
DB2	DataBase 2, ursprünglich DB2 Universal Database (UDB)
DIN	Deutsches Institut für Normung
DoD	Department of Defense
EAI	Enterprise Application Integration
EJB	Enterprise Java Beans
ERM	Entity Relationship Model, (in deutsch kurz: ER-Modell)
ERP	Enterprise Resource Planning
EURO	Währung der europäischen Währungsunion
EVA	Eingabe, Verarbeitung, Ausgabe
GmbH	Gesellschaft mit beschränkter Haftung
GPL	General Public License
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IO	Input/Output
IPSA	Interaktives Projektsystem für Anwendungsentwicklung

ISO	International Organization for Standardization
ISPF	Interactive System Productivity Facility
IT	Information Technology
JDK	Java Development Kit
JEE	Java Platform, Enterprise Edition, früher J2EE
JSF	Java Server Faces
JSP	Java Server Pages
LGPL	Lesser General Public License
LOC	Lines Of Code
MDA	Model Driven Architecture
MDSD	Model Driven Software Development
MORPH	Model Oriented Reengineering Process for Human-Computer Interface
MSSQL	Microsoft SQL (Server)
MVS	Multiple Virtual Storage
OMG	Object Management Group
OS/390	Operating System / 390
OV	Objektverwaltung
P2P	Peer-to-Peer
PAP	Programmablaufplan
PIM	Platform Independent Model
PL/1	Programming Language One, (auch PL/I oder PL1, PLI)
PSM	Platform Specific Model
REXX	Restructured Extended Executor
SEQ	Sequenziell
SQL	Structured Query Language
SQLCA	SQL Communication Area
SRAH	Software Reengineering Assessment Handbook
TSO	Time Sharing Option
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
XML	Extensible Markup Language
z/OS	zero downtime / Operating System

Thesen

1. Die Umsetzung neuer unternehmensweiter Strategien und das steigende Interesse nach Wiederverwendung, Geschäftsprozessabbildung und Integration postuliert eine Modernisierung von Legacy-Systemen.

2. Das *4-Phasen-Transformationskonzept* basiert auf Aktivitäten des Software Reengineering und ermöglicht dadurch die Modernisierung eines Legacy-Systems.

3. Auf der Grundlage des *4-Phasen-Transformationskonzeptes* kann ein in einer prozeduralen Programmiersprache implementiertes Legacy-System in vier Phasen in eine Drei-Schichten-Architektur überführt werden.

4. Das *4-Phasen-Transformationskonzept* erlaubt ein systematisches, strukturiertes Vorgehen bei der Transformation eines Legacy-Systems in ein Zielsystem und sichert damit den optimalen Einsatz moderner Technologien bei überschaubarem Zeit- und Kostenaufwand.

5. Das als Fallbeispiel betrachtete Objektverwaltungssystem konnte mit dem *4-Phasen-Transformationskonzept* erfolgreich in ein JEE-Technologie basierendes Zielsystem mit einer Drei-Schichten-Architektur überführt werden.

1 Einleitung

1.1 Motivation

Die Anwendung der Software-Technik in einem Unternehmen ist in der Praxis nicht vorwiegend mit Neuentwicklungen von Software-Systemen beschäftigt. In der Regel existieren Software-Systeme, die seit vielen Jahren im Unternehmen im Einsatz sind und einen Prozess des Wachsens durchlaufen haben. Solche Software-Systeme wurden ständig weiterentwickelt, angepasst, geändert und haben dadurch einen gewissen Zustand an Komplexität und Größe erreicht. Des Weiteren kann ein Software-System wirtschaftlich sehr bedeutend für ein Unternehmen sein, weil geschäftskritische Prozesse darauf abgebildet wurden und somit eine überlebenswichtige Basis für ein Unternehmen darstellen. Zudem kann in ein Software-System viel Domänenwissen[1] bzw. Unternehmenswissen einfließen, was als solches meist nur in den Köpfen der Mitarbeiter existiert oder existiert hat und nicht in Form von Dokumenten vorliegt. Auf Grund dieser Tatsachen kann ein gewachsenes Software-System einen hohen Wert für ein Unternehmen darstellen und ist nicht ohne weiteres durch eine Neuentwicklung ersetzbar.

Diese Tatsache wurde schon sehr zeitig erkannt und hat Edward Yourdon in einem Buch wie folgt beschrieben:

> „*Immer mehr Organisationen wird klar, daß ihr vorhandener Softwarebestand als möglicherweise wertvolles Gut betrachtet werden muß. Anstatt die alten Systeme wegzuwerfen bzw. immer wieder zu reanimieren, bis sie endgültig tot sind, investieren die Organisationen auf Weltklasseniveau Geld in die Renovierung alter Systeme.*“ ([YOU93] S. 203)

1 Domänenwissen bezeichnet in diesem Zusammenhang die Gesamtheit des Wissensgebietes innerhalb eines Fachgebiets.

Weitere Argumente für eine Modernisierung sind die Anforderungen an ein Software-System, die einem ständig wachsenden Unternehmen entsprechen müssen. Die Anforderungen können vielfältig sein, spiegeln sich aber in der Regel in technologischen oder wirtschaftlichen Weiterentwicklungen wieder. Zum Beispiel ist der Einsatz neuer Technologien notwendig, weil Hardware oder Software eine gewisse Alterungsgrenze erreicht haben. Zudem ist die gegenseitige Abhängigkeit oft ein Grund für Erneuerungen, weil zum Beispiel Hardware-Plattformen nicht mehr den Anforderungen der Software genügen oder einfach nicht mehr vom Hersteller unterstützt werden.

Die Mitarbeiterentwicklung in Unternehmen kann auch zum Verfall von Software-Systemen beitragen. Durch Personalwechsel oder Ruhestand gibt es immer weniger Fachpersonal mit Kenntnis über ein Software-System. Neues Fachpersonal muss geschult werden und greift auf Software-Dokumente zurück, welche wahrscheinlich in einem schlechten Zustand oder nicht vorhanden sind. Wenn Fachpersonal ohne Fachkenntnisse und ohne ausreichende Dokumentation Wartungs[2]- und Pflegearbeiten durchführen muss, dann wird durch Unkenntnis Struktur bzw. Architektur zerstört; daraus resultiert eine Erhöhung der Komplexität eines Software-Systems.

> „*Die Komplexität eines unstrukturierten Programms erhöht sich nach einer Korrektur um 4 Prozent, nach einer Änderung um 17 Prozent ...*“ ([BAL298] S. 665)

Die Wartungsunfähigkeit ist unter anderem ein Charakteristikum für den Übergang eines Software-Systems in ein Legacy-System. Die Kosten der Software-Wartung steigen mit zunehmender Unkenntnis über ein Software-System und belasten das Budget für die IT in gravierender Höhe bis ein Software-System für ein Unternehmen finanziell untragbar wird.

> „*Between 60 and 80 percent of an average company's IT budget is spent on maintaining existing mainframe systems and applications.*“
> (Gartner Group[3], Analyst Report, March 2002)

2 Unter Wartung von Software versteht man nach [IEEE98] und [IEEE90] die Modifizierung eines Software-Produktes nach dessen Auslieferung, um Fehler zu beheben, Systemeigenschaften zu verbessern oder die Anpassung des Produktes an eine veränderte Umgebung – Institute of Electrical and Electronics Engineers (IEEE) ist ein weltweiter Berufsverband von Ingenieuren.

3 Die Gartner Inc. (ehemals Gartner Group) ist ein renommiertes Unternehmen für Marktforschung, Analyse und Beratung.

> „*Too much money for maintenance. With just 27% of 2004 spending planned for new development, maintenance costs are choking IT productivity.*“
> (Gartner, Executive Summary, September 2004)

Ein anderer Aspekt, sich für eine Modernisierung zu entscheiden, kann der finanzielle Faktor sein. In der Regel sollte die Modernisierung eines bereits bestehenden Software-Systems weniger als eine Neuentwicklung kosten, weil die Anforderungen an ein bestehendes Software-System weitestgehend bekannt sind und nicht mehr modelliert werden müssen. Zudem existieren oftmals Software-Dokumente[4], die das bestehende Software-System beschreiben. So sollte es bei dem richtigem Zeitpunkt der Modernisierung auch noch Fachpersonal geben, das Domänenwissen hat und im Laufe der Modernisierung zum Software-System befragt werden kann.

Zur Notwendigkeit der Durchführung einer Modernisierung kommt hinzu, dass noch eine Vielzahl von Legacy-Systemen betrieben werden und auf Hardware-Plattformen laufen, die zwanzig bis dreißig Jahre alt sind. Zudem wurden die Legacy-Systeme mit Programmiersprachen entwickelt, die technologisch veraltet sind und nicht mehr dem Stand der Technik entsprechen. Populäre Vertreter solcher Programmiersprachen sind zum Beispiel COBOL[5] oder PL/1[6]. Aber auch Software-Systeme, die mit Technologien wie Java oder C# entwickelt wurden, können als Legacy-Systeme zu einer Modernisierung herangezogen werden.

Eine weitere Motivationsgrundlage dieser Arbeit ist der für Modernisierungen fehlende Lehrinhalt an Hochschulen. An Hochschulen werden im Fachgebiet der Software-Technik hauptsächlich Techniken zur Neuentwicklung von Software-Systemen vermittelt, aber nicht Techniken, die zu einer Modernisierung von Software-Systemen führen. Eine ausgewogene Darstellung beider Entwicklungslinien würde einen zukünftigen Informatiker besser auf die immer bestehende Situation der Unternehmen vorbereiten. Diese Arbeit soll einen Beitrag

4 Software-Dokument ist ein allgemeiner Oberbegriff für Dokumente die ein Software-System beschreiben. (z.B.: Quell-Code, Entwurfsdokumente, Benutzerhandbücher, Pflichtenhefte)

5 Die Programmiersprache *Common Business Oriented Language* (COBOL) entstand in den 60er Jahren und wurde für den kaufmännischen Bereich konzipiert. Diese zeichnet sich durch strikte Trennung von Datendeklaration und prozeduralen Anweisungen aus.

6 Die Programmiersprache *Programming Language One* (PL/1) wurde in den 60er Jahren von der IBM entwickelt und sollte die Vorteile bestehender Programmiersprachen wie u.a. COBOL oder Fortran vereinigen.

zur Vermittlung von Begrifflichkeiten und Techniken der Modernisierung von Software-Systemen leisten.

Die hier dargestellten Aspekte und Probleme, die ein Unternehmen mit seinen Software-Systemen mit der Zeit haben wird oder bereits hat, betreffen jede Generation von Software-Systemen. Es wird immer ein Aufkommen an Modernisierungsprojekten geben. Deshalb ist der Bedarf an Konzepten, um Legacy-Systeme möglichst kostenschonend, schnell und risikoarm in ein neues Zielsystem zu überführen, groß. Ein Konzept mit konkreten Schritten und Techniken, wie ein Legacy-System zügig in ein zeitgemäßes Zielsystem zu transformieren ist, soll im Rahmen dieser Arbeit entwickelt werden.

> „*It is clear that new technology will no more replace legacy in total than legacy technology alone will satisfy the needs of today's Web-savvy users. The truth is that modernized legacy applications play a crucial role.*“ (Giga, Analyst Report, Sept. 2002)

Bei der Untersuchung dieser Aufgabe ergeben sich Fragen, die mit dieser Arbeit beantwortet werden sollen:

- Wann wird ein Software-System zum Legacy-System und was charakterisiert ein Legacy-System?
- Wann ist der richtige Zeitpunkt ein Software-System zu modernisieren?
- Wie sieht ein konkretes Konzept zur Modernisierung eines Legacy-Systems aus?
- Inwieweit kann ein Konzept zur Modernisierung von Legacy-Systemen beitragen?
- Welche vorhandenen Techniken können einen Modernisierungsprozess unterstützen?
- Wie kann nach einer Modernisierung verhindert werden, dass ein Software-System wieder zu einem Legacy-System wird?

1.2 Aufgabenstellung und Zielsetzung der Arbeit

Das Ziel dieser Arbeit ist die Erstellung eines Konzeptes, für einen Modernisierungs- bzw. Transformationsprozess, mit dem ein bestehendes Legacy-System, dass in einer prozeduralen Programmiersprache implementiert wurde, in ein funktional äquivalentes und dem Stand der Technik entsprechendes Zielsystem mit einer Drei-Schichten-Architektur überführt werden kann. Wie in Abbildung 1.1 schematisch dargestellt, soll dass Konzept eine Modernisierung durch verschiedene aufeinanderfolgende Phasen realisieren.

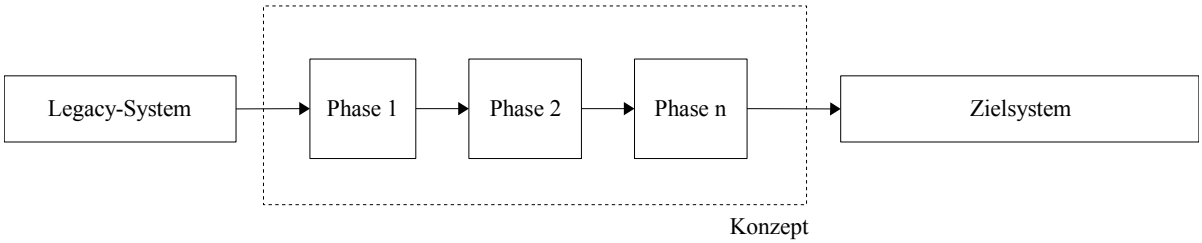


Abbildung 1.1: Zielsetzung der Arbeit (schematische Darstellung)

Im Zuge der Arbeit ist zu untersuchen, wie ein solches Konzept mit konkreten Phasen aussieht und welche vorhandenen Techniken und Vorgänge die Phasen unterstützen können, so dass die Modernisierung systematisch durchgeführt werden kann.

Modernisierung bedeutet zunächst die Untersuchung eines Legacy-Systems mit anschließender Überführung der wiedergewonnenen Informationen des Legacy-Systems in ein Zielmodell. Das Zielmodell dient anschließend als Grundlage, um das Legacy-System funktional äquivalent und technologisch, architekturbezogen in neuer Form zu implementieren.

Das Software Reengineering, als Teilgebiet der Software-Technik, umfasst dabei die Aktivitäten, die für solch einen Modernisierungsprozess notwendig sind und das Konzept charakterisieren. Terminologie und Charakterisierungen, unter anderem der Begriffe Software Reengineering und Legacy-System, werden im Kapitel 2 erläutert.

Nach Erstellung des Konzepts soll es anhand eines Fallbeispiels angewendet und seine Eignung nachgewiesen werden. Zudem sollen die am Fallbeispiel gewonnen Erfahrungen zur Verbesserung des Konzeptes beitragen. Im Fallbeispiel wird anhand des Konzeptes ein hauptsächlich in der Skriptsprache Restructured Extended Executor (REXX)[7] implementiertes Legacy-System in ein auf JEE[8] basiertes Zielsystem überführt.

Abgrenzungskriterien. Zur Einschränkung des Aufgabenumfangs soll das Konzept keine Beschreibungen von Verfahren, Methoden oder Vorgehensweisen enthalten, um Daten zwischen verschiedenen Datenbankmodellen zu migrieren. Diesbezüglich können Informationen unter [FO06] nachgeschlagen werden. Des Weiteren ist die Extraktion bzw. das Reverse Engineering von Datenstrukturen aus Datenbankmodellen bzw. aus Dateien nicht Gegenstand dieser Arbeit. An gegebener Stelle wird auf entsprechende Literatur verwiesen.

Zudem ist das Konzept nicht als Universalkonzept zu verstehen, um Legacy-Systeme mit beliebigen Programmierparadigmen und Architekturen in Zielsysteme mit beliebigen Programmierparadigmen und Architekturen zu transformieren. Die Anwendung des Konzepts wird dadurch beschränkt, dass für das Legacy-System und das Zielsystem konkrete Eigenschaften gelten müssen, die im Kapitel 3 festgelegt werden. Konkrete Eigenschaften sind zum Beispiel, dass nur solche Legacy-Systeme betrachtet werden, die mit einer prozeduralen Programmiersprache implementiert wurden und das als Architektur des Zielsystems eine Drei-Schichten-Architektur festgelegt wurde.

Das Konzept stellt auch keine Migrationsstrategie dar, bei der der Zeitpunkt der Übernahme von Daten bzw. der Zeitpunkt des Übergangs vom Legacy- zum Zielsystem festgelegt wird.

7 REXX ist eine Skriptsprache, welche in den achtziger Jahren von Mike Cowlishaw bei der IBM entwickelte wurde.

8 Die Java Platform Enterprise Edition (JEE) ist im Java-Umfeld eine Plattform zur Entwicklung von N-Schichten-Architekturen.

1.3 Voraussetzungen

Für das Verständnis dieser Arbeit wird ein Grundwissen über Software-Technik [BAL08] [BAL298] und die Unified Modeling Language (UML[9]) [UML09] vorausgesetzt.

In der Arbeit werden für analytische Zwecke Techniken angewendet, die auf Graphen basieren. Daher wird ein Grundwissen über graphentheoretische Begriffe [KN05] [VL96] vorausgesetzt. Zur Erstellung und optimalen Darstellung von Graphen wird das unter Tabelle 1.1 vorgestellte Software-Produkt verwendet, welches sich auf der Begleit-CD [CHRCD09] befindet.

yEd Graph Editor	**Beschreibung**	Java-Programm zur Erstellung und optimalen Darstellung von Graphen.
	Hersteller	http://www.yworks.com
	Version	3.3
	Lizenz	Freeware

Tabelle 1.1: Bearbeitungsprogramm für Graphen (Stand vom: 04.08.2009)

Zum Verständnis des Fallbeispiels sind Grundlagen über die Skriptsprache REXX [DJ05] [IBM97], Interactive System Productivity Facility (ISPF)[10] [DJ05] [LF05], PL/1 [MW90], Structured Query Language (SQL)[11] [BG07], Schichtenarchitekturen [DJ08] (Kapitel 3 und 4) und Java [UC09], [IDE02] von Vorteil. Es werden außerdem Erfahrungen mit dem Werkzeug *objectiF 7.0 Enterprise* für modellgetriebene Software-Entwicklung und der zugehörigen Vorlage *Web Application in Java* vorausgesetzt [BS09] (Kapitel 3 und Anhang D).

9 Die Unified Modeling Language (UML) ist eine Sprache zur Spezifikation, Visualisierung, Dokumentation und Konstruktion von Modellen für Software-Systeme.

10 ISPF ist ein Software-Produkt für Terminals im IBM-Großrechnerbereich, mit dem man u.a. für eigene Programme zeichenorientierte Benutzerschnittstellen aufbauen kann.

11 Structured Query Language (SQL) ist eine Abfragesprache für relationale Datenbanken.

1.4 Aufbau der Arbeit

Die Arbeit ist logisch in Vorbereitungs- und Analyseteil sowie Durchführungs- und Ergebnisteil unterteilt. Mehrere Kapitel bilden einen Teil und werden im Folgenden kurz vorgestellt.

Vorbereitung und Analysen

- **Kapitel 2 - Legacy-System und Modernisierung**
  Im zweiten Kapitel werden Terminologie und Charakterisierung der Begriffe Legacy-System und Software Reengineering erläutert. Eine Analyse verschiedener Strategien beim Umgang mit Legacy-Systemen ermittelt die strategische Ausrichtung des Konzeptes. Ergänzend zu den Vorbetrachtungen wird am Ende des Kapitels der idealtypische Software-Lebenszyklus erläutert. Dieser soll zeigen, in welchen Stadien Legacy-Systeme entstehen und soll Aufschluss geben, wann der richtige Zeitpunkt für eine Modernisierung ist.

- **Kapitel 3 - Konzeptkonkretisierung**
  Im dritten Kapitel wird anhand einer Marktstudie über Software-Modernisierung eine Bewertung durchgeführt, mit der die Notwendigkeit des Konzeptes bestätigt wird. Anhand dieser Bewertung werden Systemeigenschaften für das Legacy- bzw. Zielsystem abgeleitet, an denen sich das Konzept orientieren wird.

Das 4-Phasen-Transformationskonzept

- **Kapitel 4 - Das 4-Phasen-Transformationskonzept**
  Im vierten Kapitel wird das Software-Reengineering-Konzept als *4-Phasen-Transformationskonzept* vorgestellt und beschrieben. Zuerst wird das Vorgehen des Konzeptes in einer Übersicht allgemein beschrieben. Anschließend werden die einzelnen Phasen separat und deren Techniken und Vorgänge vorgestellt. In das Konzept gehen die aus den Kapiteln zwei und drei ermittelten Eigenschaften ein.

- **Kapitel 5 - Fallbeispiel**
  Im fünften Kapitel wird das *4-Phasen-Transformationskonzept* an einem konkreten Fallbeispiel Schritt für Schritt angewendet. Auf diese Weise wird die Durchführbarkeit und die Eignung des Konzeptes in seinem Eigenschaftsbereich gezeigt.

Ergebnisse, Erfahrungen und Diskussion

- **Kapitel 6 - Ergebnisse, Auswertung und Diskussion**
  Im sechsten Kapitel werden die Erfahrungen der Konzeptanwendung am Fallbeispiel und die Umsetzung des *4-Phasen-Transformationskonzeptes* ausgewertet. Die in dieser Arbeit aufgestellten Fragestellungen und Thesen werden im Kontext der Resultate bewertet und interpretiert.

- **Kapitel 7 - Abschlussbetrachtungen**
  Im siebten Kapitel werden die Ziele, der Ablauf und die Ergebnisse dieser Arbeit abschließend zusammengefasst dargestellt. In einem Ausblick werden abschließend Vorschläge für weiterführende Themen gegeben.

Anhang

- **A - Beispiel eines Kriterienkataloges**
  Im Anhang A wird der Aufbau eines Kriterienkataloges mit möglichen Kriterien zur Wiederverwendbarkeit durch Migration vorgestellt.

- **B - Data Dictionary der Objektverwaltung**
  Im Anhang B sind alle im Fallbeispiel dargestellten Daten in einem Data Dictionary zusammengefasst.

- **C - Programmablaufpläne (OV.REXX)**
  Der Anhang C beinhaltet ausgewählte Programmablaufpläne von Prozeduren aus dem REXX-Programm OV.REXX, um unter anderem Ablauflogik zu ermitteln.

- **D - Anforderungen des Fallbeispiels**
  Der Anhang D beinhaltet Anwendungsfall- und Aktivitätsdiagramme, sowie Anwendungsfallbeschreibungen und ein Glossar für das Fallbeispiel.

- **E - Plattformunabhängiges Modell**
  Der Anhang E beinhaltet die plattformunabhängigen Modelle des Zielsystems, welche im Fallbeispiel modelliert werden.

- **F - Tests für den Prototypen**
  Im Anhang F werden ausgewählte Testfälle und Testszenarien für den Prototypen des Zielsystems aufgelistet.

1.5 Verwendete Notationen

- In REXX können Zeichenketten durch folgenden Operator verknüpft werden: !! Weitere Informationen über die Konkatenation von Zeichenketten in REXX können unter ([DJ05] S. 70) nachgeschlagen werden.
- Zur Darstellung des Aufbaus von Datenstrukturen im Fallbeispiel wird die erweiterte Backus-Naur-Form[12] nach [ISO14977] verwendet.

12 Die Backus-Naur-Form (BNF), benannt nach John Backus und Peter Naur, ist eine kompakte formale Meta-Sprache, welche zur Darstellung kontextfreier Grammatiken genutzt wird (z.B.: Darstellung der Syntax höherer Programmiersprachen). Die erweiterte Backus-Naur-Form (EBNF) ist eine Erweiterung der ursprünglichen Backus-Naur-Form.

2 Legacy-System und Modernisierung

Im ersten Abschnitt dieses Kapitels wird der Begriff Legacy-System erläutert und seine Merkmale werden zu einer Übersicht zusammengefasst. Anschließend werden verschiedene Strategien beim Umgang mit Legacy-Systemen analysiert, um dem Konzept eine strategische Orientierung zu geben. Die strategische Orientierung wird anhand einer Marktstudie bekräftigt.

Im zweiten Abschnitt wird die Terminologie des Software Reengineering erläutert. Hier werden grundlegende Begriffe und der Zusammenhang zwischen Software-Technik und Software Reengineering aufgezeigt. Mit den Begriffen werden benötigte Aktivitäten des Software Reengineering erklärt, auf die das Konzept zurückgreifen wird.

Der letzte Abschnitt erläutert den idealtypischen Software-Lebenszyklus, welcher die Stadien aufzeigt, in dem Legacy-Systeme entstehen, und der Aufschluss über den richtigen Zeitpunkt für eine Modernisierung gibt.

2.1 Der Begriff Legacy-System

Das englische Wort *legacy* mit den Bedeutungen Vermächtnis, Erbschaft, Hinterlassenschaft oder Altlast ist nicht willkürlich gewählt worden. Software-Systeme können eine Lebensdauer von über dreißig Jahren erreichen, die zum größten Teil dafür nicht konzipiert und gedacht waren. In einer solchen Zeit kann die Komplexität eines Software-Systems ansteigen, weil sich Anforderungen des Geschäftsumfeldes ändern und dazu zwingen das Software-System anzupassen. Deshalb sollten Software-Systeme so konzipiert sein, dass sie sich an neue Anforderungen anpassen lassen. Ist ein Software-System nicht oder nur schwer anpassbar, in der Regel aus Gründern einer falschen oder sogar fehlenden Architektur oder weil Technologien veraltet sind und den Anforderungen nicht gerecht werden können, so wird das Software-System zum Legacy-System.

> „*Ein Legacy-System ist ein soziotechnisches System, welches Legacy Software enthält.*“ ([MAS07] S. 2)

Spricht man von einem Legacy-System, dann spricht man auch von einem soziotechnischen System. Darunter versteht man eine strukturierte bzw. organisierte Menge von Menschen und Technologien, die zusammen ein spezifisches Ergebnis produzieren. Demzufolge ist ein Legacy-System nur in Verbindung mit einem Unternehmen zu sehen, welches mit dem Unternehmen über mehrere Jahre gewachsen und dadurch vom Unternehmen schwer trennbar geworden ist. Schwer trennbar heißt unter Umständen auch, dass ein Ausfall des Legacy-Systems beim Unternehmen wirtschaftlichen Schaden hervorrufen kann. Durch diese Verwobenheit wird Legacy Software zu geschäftskritischer Software, welche sehr schwer oder nicht geändert bzw. ersetzt werden kann.

> *„Legacysoftware ist eine geschäftskritische Software, welche nicht, oder nur sehr schwer, modifiziert werden kann.“* ([MAS07] S. 2)

Weiterhin ist ein Legacy-System auch charakterisiert, dass immer weniger Fachpersonal Wissen darüber hat. Bei angenommenen dreißig Jahren Bestehen eines Legacy-Systems und einer durchschnittlichen Betriebszugehörigkeit eines Software-Entwicklers von fünf bis sieben Jahren, reduziert sich das Wissen über ein Legacy-System rapide. Unzureichende

Dokumentation von Änderungen verschlimmert das Verständnis für nachfolgendes Fachpersonal, welches zur Wartung eingesetzt wird. Die Fähigkeit zur flexiblen Anpassung eines Software-Systems an neue Anforderungen kann durch dieses Nichtwissen zerstört werden.

Ein Software-System kann weiterhin zu einem Legacy-System werden, durch die Tatsache, dass Hardware- und Software-Technologien altern. Zum Beispiel entspricht der Einsatz der Programmiersprachen COBOL oder PL/1 nicht mehr dem Stand der Technik.

In Abbildung 2.1 wird das Legacy-System als soziotechnisches Gebilde dargestellt. Jeder Teil dieses Gebildes verändert sich ständig und begünstigt die Veränderung eines Software-Systems zu einem Legacy-System.

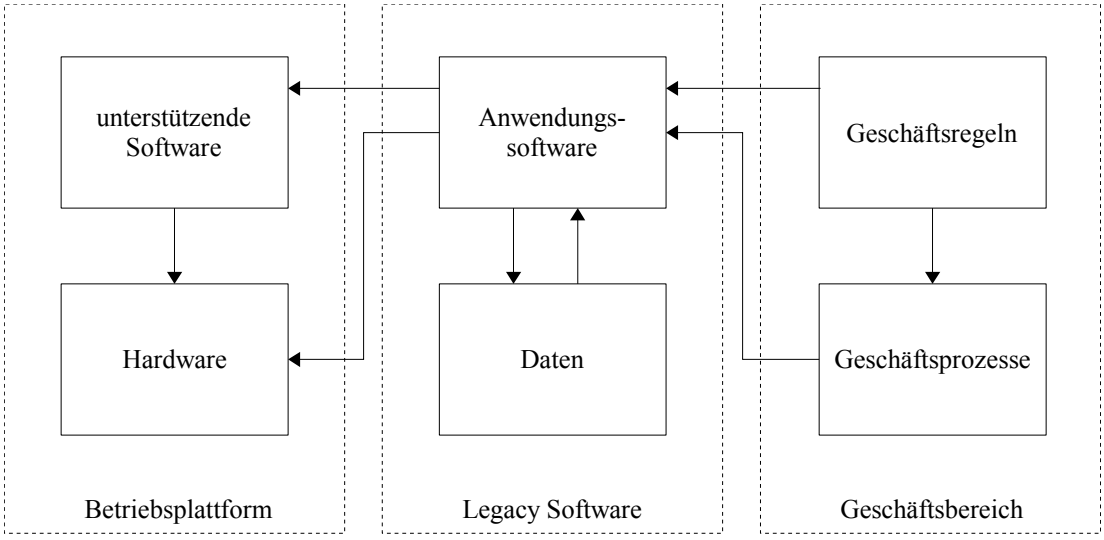


Abbildung 2.1: Das Legacy-System als soziotechnisches Gebilde [MAS07]

Betriebsplattform. Legacy Software läuft auf einer Betriebsplattform und kann von speziellen Plattformeigenschaften Gebrauch machen. Solche Plattformeigenschaften machen abhängig, sowohl Hardwareeigenschaften wie auch Betriebsplattform eigene Software die verwendet wird. Bei Legacy Software, die direkt für Großrechner (Mainframes) konzipiert wurde, ist das ein wichtiger Aspekt.

Legacy Software. Die Anwendungs-Software ist die eigentliche Legacy Software. Sie verarbeitet Daten, die in Dateien oder Datenbanken gehalten werden.

Geschäftsbereich. Der Geschäftsbereich beinhaltet die Geschäftsregeln und Geschäftsprozesse, welche die Legacy Software umsetzen muss. Die Geschäftsprozesse dienen dem Unternehmen zur Durchführung von Tätigkeiten, um das Ziel des Unternehmens zu erreichen. Die Tätigkeiten unterliegen einer Vielzahl von Geschäftsregeln.

Im Folgenden werden die wichtigsten Charakteristika zusammengefasst, die ein Software-System als Legacy-System definieren.

Technisch

- nicht mehr dem Stand der Technik entsprechende Technologien sind im Einsatz (Hardware, Software); zudem ist die Unterstützung veralteter Technologien nicht mehr gegeben
- nicht mehr dem Stand der Technik entsprechende Technologien können den gestiegenen Ressourcenbedarf nicht mehr abdecken
- die Möglichkeit zur Integration wird durch alte Technologien verschlechtert, unter anderem, weil keine Schnittstellen vorhanden sind

Betriebswirtschaftlich

- ein Software-System ist nicht mehr wartbar, weil unter anderem die Flexibilität bzw. Anpassbarkeit nicht mehr gegeben ist
- Wartungstätigkeiten werden mit hohem Aufwand durchgeführt, was finanziellen und zeitlichen Aufwand nach sich zieht
- der Mangel an Fachpersonal (*Know-how*-Trägern), welches sich mit einem Software-System auskennt, gefährdet die Betriebssicherheit
- die von Legacy-Systemen verwendeten Technologien werden an Hochschulen nicht mehr gelehrt, dadurch müssen Mitarbeiter eines Unternehmens speziell geschult werden

Strategisch

- das Software-System kann nicht mehr an neue Anforderungen eines Unternehmens angepasst werden; es ist auf längere Sicht gesehen nicht mehr in der Lage, seine Aufgabe zu erfüllen

- ein Unternehmen ist abhängig von einem Software-System, welches nicht ohne weiteres ersetzt werden kann

Die Bewertung eines Software-Systems anhand dieser Kriterien als Legacy-System ist jedoch nicht immer eindeutig. Für die technische und ökonomische Bewertung eines existierenden Software-Systems zur Bestimmung, ob das Software-System zu warten, einzustellen oder ob ein Software Reengineering notwendig ist, kann der im Software Reengineering Assessment Handbook (SRAH) [DoD97] enthaltene SRA-Prozess verwendet werden. Dieses Standardhandbuch stammt aus dem Verteidigungsministerium der Vereinigten Staaten von Amerika (Department of Defense DoD) mit Angaben und Anleitungen für Software-Systeme.

Nach der Begriffserklärung und der Charakterisierung des Legacy-Systems, werden im Folgenden Strategien beim Umgang mit Legacy-Systemen betrachtet. Diese Betrachtung zeigt verschiedene Modernisierungsstrategien auf, die Unternehmen bevorzugt durchführen und dem Konzept eine Orientierung geben können.

2.2 Strategien beim Umgang mit Legacy-Systemen

Besteht ein Legacy-System, so muss früher oder später die Entscheidung getroffen werden, was mit diesem geschehen soll. So stehen einem Unternehmen folgende Strategien beim Umgang mit Legacy-Systemen zur Verfügung:

- Abschaltung
- Software-Sanierung
- komplette Neuentwicklung
- Kapselung (wrapping)
- neue Standard-Software
- Migration

Abschaltung. Die vielleicht einfachste, aber für eine Modernisierung nicht in Betracht kommende Variante, ist die komplette Abschaltung bzw. Stilllegung des Legacy-Systems ohne Inbetriebnahme eines Ersatz-Systems.

Software-Sanierung. Software-Sanierung ist ein Oberbegriff für die Erhaltung eines lauffähigen Legacy-Systems, bei dem Maßnahmen des Software Reengineering eingesetzt werden mit dem Ziel, die Software-Qualität des Legacy-Systems zu verbessern und dadurch die Wartbarkeit und Erweiterbarkeit wiederzuerlangen oder zu erhalten. Bei dieser Variante wird keine Modernisierung durchgeführt.

Komplette Neuentwicklung. Bei einer kompletten Neuentwicklung wird das Legacy-System von Grund auf neu entwickelt. Hier werden die zum Zeitpunkt aktuellen Technologien, Software-Architekturen und Software-Entwicklungsmethoden verwendet. Bei dieser Strategie werden keinerlei Teile des Legacy-Systems wiederverwendet. Es erfolgt gegebenenfalls lediglich die Übernahme der Funktionalitäten und Daten des Legacy-Systems. In der Regel wird die Entwicklung des neuen Systems neben dem im Betrieb befindlichen Legacy-Systems durchgeführt. Nach der Fertigstellung wird das Legacy-System stillgelegt und durch das neue System ersetzt.

Kapselung. Bei einer Kapselung (wrapping) bleibt das Legacy-System in seiner Betriebsumgebung unangetastet und wird nur umhüllt, so dass neuere Technologien über eine Schnittstelle Zugriff auf das Legacy-System haben. Die Kapselung ist ein sehr einfacher Weg, Legacy-Systeme für neue Technologien zur Verfügung zu stellen, sie löst aber die bestehenden Nachteile eines Legacy-Systems nicht. Durch die zusätzliche Kapselung entsteht ein erhöhter Wartungsaufwand. Ein Beispiel für eine Kapselung ist das sogenannte *Screen Scraping*. Hier wird eine zeichenbasierte Benutzerschnittstelle durch eine graphische Benutzerschnittstelle überdeckt. Dabei dient ein W*rapper* als Schnittstelle zwischen Legacy-System und graphischer Benutzerschnittstelle.

Neue Standard-Software. Der Einsatz einer neuen Standard-Software als Ersatz für das Legacy-System ist eine andere Möglichkeit, mit einem Legacy-System umzugehen. Dieser Ersatz bietet auch die einfache Möglichkeit einer Auslagerung (outsourcing) eines geschäftlichen Bereiches an ein Dienstleistungsunternehmen, für welches das Legacy-System verantwortlich war. Sofern eine Standard-Software alle Anforderungen eines Unternehmens erfüllt und das Legacy-System vollständig ersetzen kann, kann dieses Vorgehen auch vorteilhaft sein. Ein Vorteil ist zum Beispiel, dass die ganze Software-Pflege und Software-Entwicklung in der Verantwortung des Dienstleistungsunternehmens ist. Nachteilig können die zu zahlenden Lizenzkosten sein, der zusätzliche Zeitaufwand für die Anpassung der Standard-Software an Unternehmensanforderungen oder die Schulung der Mitarbeiter.

Diese Ersatzlösung ist für Legacy-Systeme vorteilhaft, die nicht spezialisiert sind und allgemeingültige Geschäftsregeln bzw. Geschäftsprozesse umsetzen.

Migration. Die Migration beschäftigt sich mit der Überführung eines Legacy-Systems in ein Zielsystem, bei der die Wiederverwendung von Teilen oder die Gesamtheit des Legacy-Systems angestrebt wird, so dass das Zielsystem wieder gut wartbar und den Anforderungen eines Unternehmens entspricht und erweitert werden kann. Die Migration orientiert sich hauptsächlich am Erhalt des Datenbestandes und der zeitlich geordneten Überführung der wiederverwendbaren Teile des Legacy-Systems zum Zielsystem.

Die Migration ist ein weitläufiger allgemeiner Begriff und kann sich in verschiedene Teilgebiete unterteilen. So beschäftigt sich zum Beispiel die Datenmigration mit der Überführung

von existierenden Daten in eine neue Datenhaltungsumgebung. Die Hardware-Migration beschäftigt sich mit der Überführung eines bestehenden Software-Systems auf eine neue Hardware-Plattform. Je nach Ziel kann die Migration das gesamte Legacy-System umfassen oder nur eine spezialisierte Teilmigration. Bei einer Modernisierung kann die Migration verwendet werden.

Welche Strategien beim Umgang mit Legacy-Systemen werden bevorzugt?

Den Trend der gewählten Modernisierungsstrategien in Unternehmen zeigt eine Marktstudie aus der Schweiz (Abbildung 2.2): Unternehmen entscheiden sich hauptsächlich für Migrationswege oder für eine neue Standard-Software. Eine komplette Neuentwicklung ist dagegen die dritte Wahl.

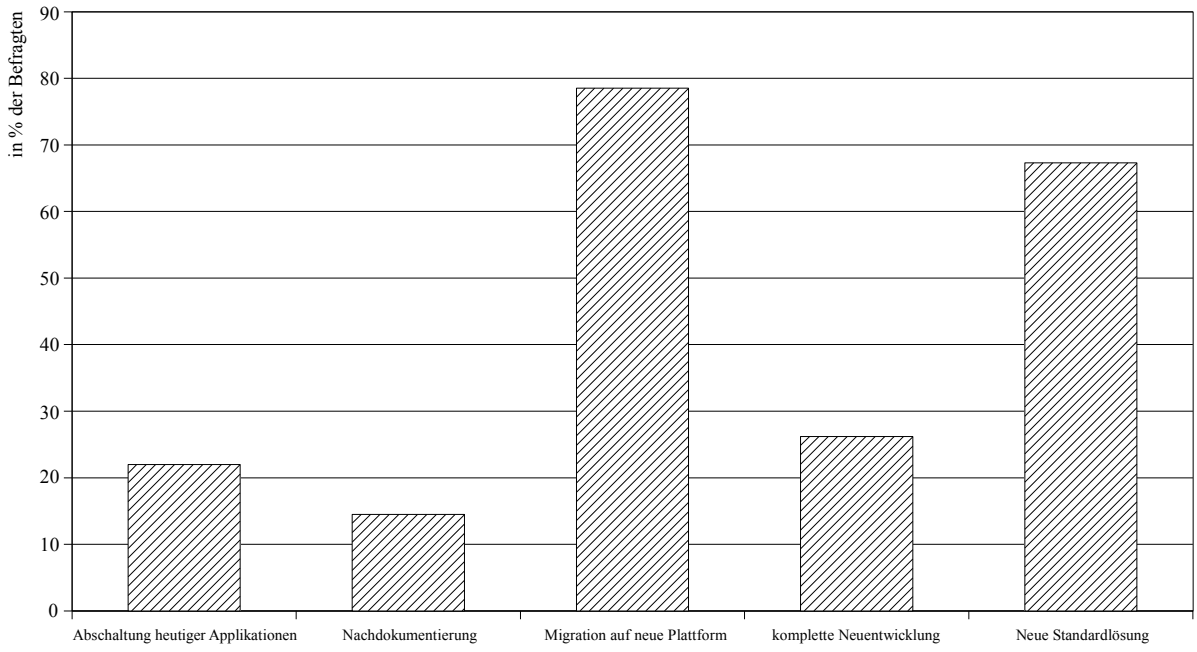


Abbildung 2.2: Lösungswege Software-Modernisierung (Schweizer-Marktstudie) [BP05]

Die Wahl der richtigen Strategie ist nicht nur von den Kosten, Risiken und dem Zeitaufwand abhängig, sondern auch daraus welche Geschäftsziele ein Unternehmen verfolgt und welche funktionalen und technologischen Faktoren diese unterstützen. Eine sogenannte *harte Modernisierung* (Tabelle 2.1) wird immer dann in Betracht gezogen, wenn das Legacy-System nur noch mit viel Aufwand (oder überhaupt nicht mehr) wartbar ist, oder die Leistung des Legacy-Systems nicht mehr gesteigert werden kann. Eine Neuentwicklung ist mit hohen

Risiken und Kosten verbunden, weil das neue Software-System erst entwickelt und getestet werden muss und oft nicht ausgereift ist. Bei einer *sanften Modernisierung* nach Tabelle 2.1 bzw. [BB05] wird davon ausgegangen, dass Teile bzw. das ganze Legacy-System in einem Zielsystem wiederverwendet bzw. Teile migriert werden können, so dass wichtige Funktionalitäten, die sich jahrelang bewährt haben, in ein Zielsystem migriert werden können.

Modernisierungsweg	**Kosten**	**Risiken**	**Zeitaufwand**
harte Modernisierung			
neue Standard-Software	hoch	tief	mittel
komplette Neuentwicklung	hoch	hoch	hoch
sanfte Modernisierung			
Migration	tief	tief	gering

Tabelle 2.1: Modernisierungslösungen im Vergleich [BB05]

Die Migration ist eine der meist gewählten Strategien beim Umgang mit Legacy-Systemen. Doch legen Migrationsstrategien wie *„Big Bang“* / *„Cold Turkey“*, *„Chicken Little“*, *„Database First“*, *„Database Last“* und *„Butterfly“* nur den Zeitpunkt und eine mögliche Reihenfolge zur Übernahme der Daten und Teile eines Legacy-Systems fest. Es werden aber keine Methoden bzw. Techniken und Maßnahmen zur Transformation des Legacy-System zu einem Zielsystem gegeben. Informationen über die genannten Migrationsstrategien können unter [EC05] nachgeschlagen werden.

Migration und Neuentwicklung. Für eine Modernisierung sind Hauptkriterien wie Daten- und Funktionsübernahme vom Legacy-System gegeben. Dadurch sind die Strategien Neuentwicklung und Migration zu kombinieren. Eine Neuentwicklung wird immer notwendig sein, wenn neue Technologien eingesetzt werden sollen und Systemteile aus verschiedenen Gründen nicht wiederverwendet werden können, sondern neu entwickelt werden müssen. Bei einer Modernisierung können neue Anforderungen oder Funktionalitäten hinzukommen, die erst entwickelt werden müssen. Eine Migration bietet sich immer dann an, wenn Systemteile bzw. Systemkomponenten wiederverwendet werden können oder existierende Daten im Zielsystem übernommen werden sollen.

In dieser Arbeit wird die Neuentwicklung und Migration in Kombination verstanden, wobei in aller Regel eine funktionale Migration bei einem Legacy-System durchgeführt werden kann. Bei der funktionalen Migration geht es im Wesentlichen um die Analyse eines Legacy-Systems, mit anschließender Transformation der bestehenden Anforderungen bzw. Funktionalitäten zu einem Zielmodell und der nachfolgenden Überführung des Zielmodells zur Zielplattform.

Nachdem die strategische Ausrichtung des Konzeptes dargestellt ist, wird mit dem Software Reengineering die Vorgehensorientierung des Konzeptes vorgestellt. Verschiedene Aktivitäten aus dem Bereich des Software Reengineering sollen angewendet werden. Grundlegende Begriffe und Zusammenhänge aus dem Gebiet des Software Reengineering werden im Folgenden erläutert.

2.3 Terminologie des Software Reengineering

Software Reengineering. Das Software Reengineering als ingenieurwissenschaftliche Disziplin ist ein Teilgebiet der Software-Technik und ist auch unter den Synonymen *Renovation* und *Reclamation* bekannt. Der bekannteste Versuch einer Begriffsklärung ist von Chikofsky und Cross [CC90] und lautet frei übersetzt wie folgt:

> „*Software Reengineering ist die Untersuchung und Modifikation eines Systems, um es in einer neuen Form wiederherzustellen und diese Form nachfolgend zu implementieren.*“ [CC90]

Das Software Reengineering umfasst also alle Aktivitäten, um ein bestehendes Software-System (Legacy-System) oder einzelne Teile dieses Legacy-Systems zu untersuchen und es gegebenenfalls mit Hilfe automatisierter Werkzeuge zu ändern und neu zu erstellen. In der Abbildung 2.3 wird dargestellt, wie sich Software Reengineering in die Software-Technik einordnet und welche Begriffe und Vorgänge in Beziehung stehen.

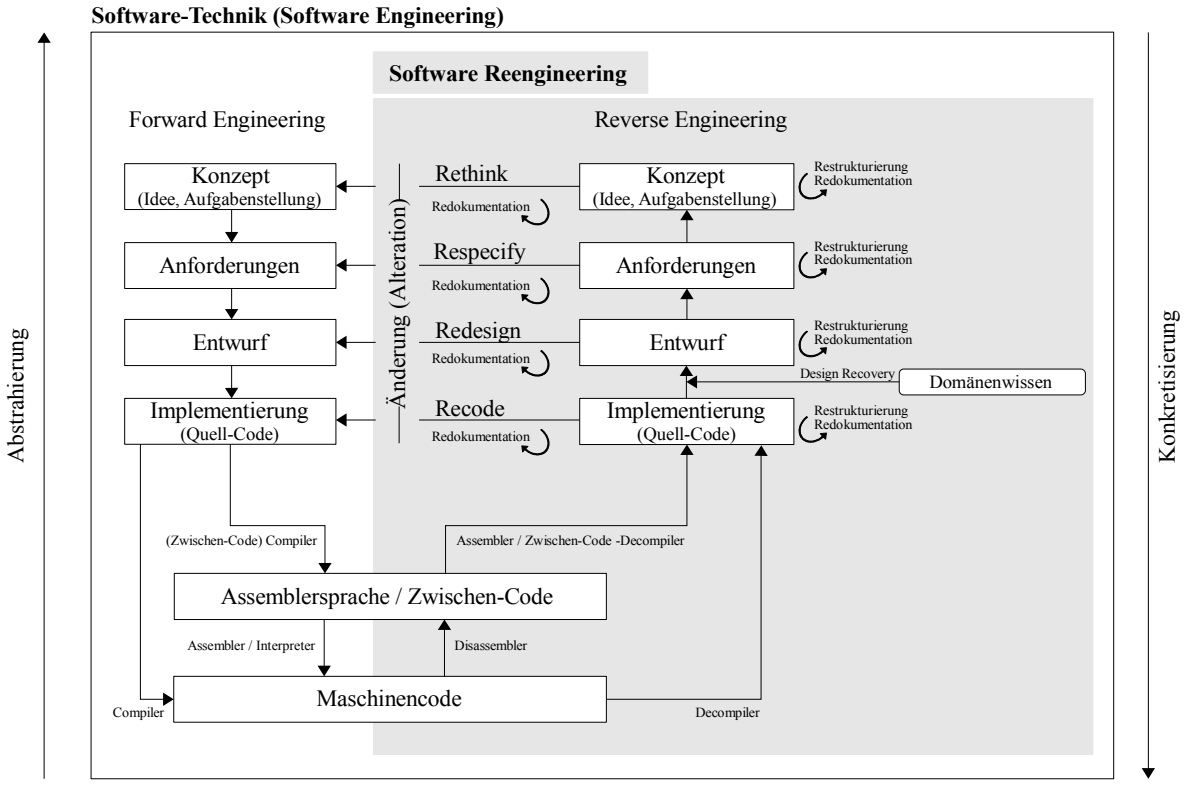


Abbildung 2.3: Einordnung Software Engineering – Software Reengineering

Die Abbildung 2.3 orientiert sich an der von Chikofsky und Cross eingeführten graphischen Begriffserklärung aus [CC90] und aus dem Hauptbeitrag aus [BB96]. Zur Ergänzung und zur Vollständigkeit der Gesamtübersicht wurde in der Abbildung 2.3 der Weg von der Implementierung bis zum Maschinencode (das fertige, lauffähige System) dargestellt.

Das Software Reengineering beinhaltet die Anwendung von folgenden Vorgängen und Aktivitäten, die das Konzept charakterisieren:

- (Forward Engineering)
- Reverse Engineering
- Design Recovery
- Restrukturierung (Restructuring) / Redesign
- Redokumentation

Forward Engineering. Der Begriff Forward Engineering beinhaltet die Anwendung der Methoden der Software-Technik und stellt damit die klassische Software-Entwicklung dar. Für das Verständnis ist es wichtig zu erkennen, dass es sich bei Forward Engineering um die Überführung von abstrakten Software-Dokumenten in konkrete Software-Dokumente handelt, bis hin zum lauffähigen Software-System (Abbildung 2.4).

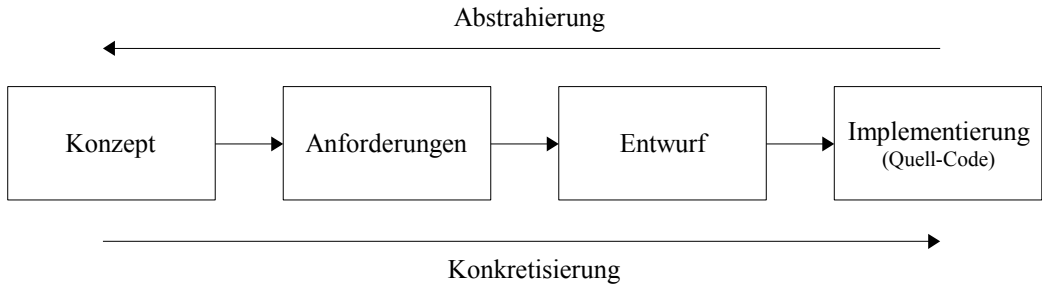


Abbildung 2.4: Forward Engineering [CC90]

Forward Engineering wird häufig als Prozess zur Modifikation und Neuimplementierung eines Software-Systems zum Software Reengineering hinzugezählt.

Reverse Engineering. Reverse Engineering ist das Pendant zum Forward Engineering und wird von Chikofsky und Cross [CC90] wie folgt beschrieben:

> „*Reverse Engineering umfasst die Analyse eines Systems, um die Komponenten und deren Beziehungen erkennen und das System in einer anderen Form oder auf einem abstrakteren Niveau darstellen zu können.*“ [CC90]

Die Hauptaufgabe von Reverse Engineering ist die Überführung von konkreten Software-Dokumenten in abstraktere Software-Dokumente, sozusagen die Umkehrung des Forward Engineering. Bei diesem Vorgang sollen durch methodische Analysen Software-Komponenten bzw. Software-Teilkomponenten eines Software-Dokumentes und ihre Beziehungen untereinander identifiziert und rekonstruiert werden, um das System auf einem höherem abstrakteren Niveau zu beschreiben (Abbildung 2.5).

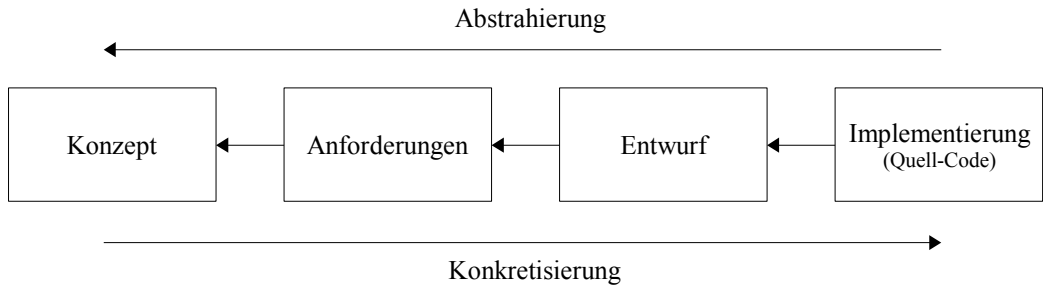


Abbildung 2.5: Reverse Engineering [CC90]

Reverse Engineering wird im Rahmen dieser Arbeit ausgehend vom Quell-Code nach Abbildung 2.6 als Rekonstruktionsvorgang angewandt, nicht jedoch ausgehend vom übersetzten lauffähigen Software-System.

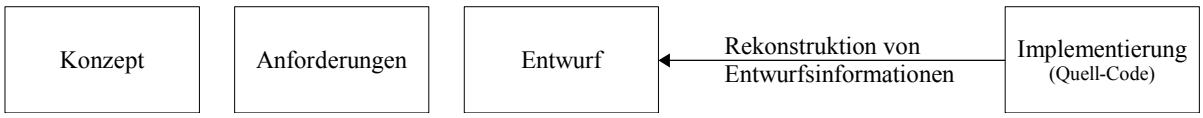


Abbildung 2.6: Reverse Engineering - Rekonstruktion der Architektur [KO04]

Reverse Engineering wird auch eingesetzt, um Strukturen, Zustände, Konstruktionselemente bzw. den kompletten Quell-Code aus dem fertigen System (Maschinencode) zurück zu gewinnen. Allerdings ist dieses Vorgehen rechtlich zu prüfen, weil in aller Regel Quell-Code-Wiedergewinnung aus Maschinencode wegen Rechtsansprüchen verboten ist.

Unterschiede zwischen Reverse Engineering und Forward Engineering

Um Forward Engineering und Reverse Engineering besser zu verstehen und um die Schwierigkeiten der Richtungen zu verdeutlichen, werden in der Tabelle 2.2 die Unterschiede beider Richtungen dargestellt [KO04].

Reverse Engineering	Forward Engineering
Die Anforderungen sind weitestgehend klar und sind bereits im Software-System enthalten.	Die Anforderungen an das Software-System sind noch weitestgehend unklar.
Aufwands-, Dauer- und Zuverlässigkeitsaussagen usw. existieren idealerweise aus vergangenen Erfahrungen.	Aufwands-, Dauer- und Zuverlässigkeitsaussagen usw. sind schwierig.
Das Software-System existiert.	Das Software-System existiert nicht.
Die Strukturen und Qualitätsanforderungen sind bekannt, sofern Software-Dokumente existieren.	Beim Entwurf ist noch alles offen.
Eine Änderung kann übergreifende Auswirkungen haben, weil es viele versteckte Abhängigkeiten gibt.	Bei einem sauberen Entwurf gibt es in der Regel keine versteckten Abhängigkeiten.

Tabelle 2.2: Unterschiede zwischen Reverse- und Forward Engineering

Design Recovery. Für das Verständnis des Begriffs Design Recovery wurde ebenfalls von Chikofsky und Cross [CC90] eine Begriffserklärung gegeben.

> „*Design Recovery ist eine Erweiterung des Reverse Engineering um die Anwendung von Domänenwissen, externen Informationen und Deduktion*[13] *oder Fuzzy Reasoning*[14] *zur Identifikation sinnvoller, abstrakter Beschreibungen eines Systems.*“ [CC90]

Man spricht von Design Recovery, wenn das Ziel des Reverse Engineering die Rekonstruktion des vollständigen Entwurfs eines Software-Systems ist und man für die vollständige Rekonstruktion auf alle relevanten Informationsquellen über ein Software-System zurückgreift.

13 Deduktion bezeichnet in diesem Zusammenhang die Schlussfolgerungsweise vom Allgemeinen auf das Besondere. Es werden Einzelerkenntnisse aus dem Allgemeinen gewonnen.

14 Fuzzy-Logik ist die Theorie u.a. für die Modellierung von Unsicherheiten und Unschärfen von umgangssprachlichen Beschreibungen. Fuzzy Reasoning bezeichnet die Anwendung der Fuzzy-Logik als Schlussfolgerungsweise.

Restrukturierung (Restructuring) und Redesign. Der Begriff der Restrukturierung wurde von Chikofsky und Cross [CC90] wie folgt allgemein beschrieben:

> „*Die Restrukturierung ist die Umformung von Entwicklungsdokumenten desselben Abstraktionsniveaus, wobei das externe Verhalten (Funktionalität, Semantik) nicht verändert wird.*“ [CC90]

In diesem Sinne bezieht sich die Restrukturierung auf die Veränderung und Verbesserung eines Software-Dokumentes auf demselben Abstraktionsniveau (Abbildung 2.7).

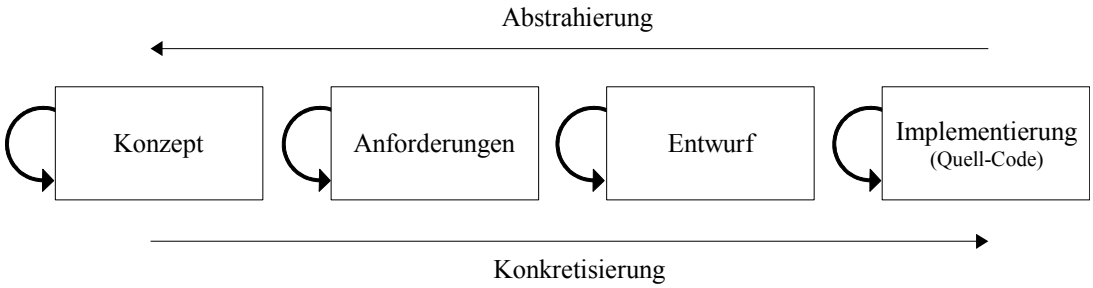


Abbildung 2.7: Restrukturierung [CC90]

Eine Erweiterung der Restrukturierung ist zum Beispiel das Redesign auf der Ebene des Entwurfs. Der Übergang zwischen der Restrukturierung und dem Redesign verläuft kontinuierlich und ist so zu unterscheiden, dass zum Beispiel die Einführung von Unterprogrammen zur Restrukturierung, aber eine Modularisierung zum Redesign zählt.

Redokumentation. Der Begriff der Redokumentation wurde von Chikofsky und Cross [CC90] wie folgt allgemein beschrieben:

> „*Die Redokumentation ist die Erzeugung oder Revision einer semantisch äquivalenten Beschreibung auf demselben Abstraktionsniveau.*“ [CC90]

Demzufolge werden bei einer Redokumentation alle relevanten wiedergewonnenen Informationen eines abstrakten Niveaus über ein Software-System neu dokumentiert oder auf einen aktuellen Stand gebracht. Die Redokumentation ist der wichtigste Teil beim Reverse Engineering, weil die Software-Dokumente dem Entwickler als Verständnisgrundlage über ein Software-System dienen.

2.4 Software-Lebenszyklus

Um besser zu verstehen, welche Zustände ein Software-System durchlaufen kann und wann der richtige bzw. notwendige Zeitpunkt für ein Software Reengineering bzw. für eine Modernisierung ist, muss man den Software-Lebenszyklus betrachten. Dazu gibt es ein idealtypisches Software-Lebenszyklusmodell von Rajlich und Bennett [RB00], bei dem die zeitliche Abfolge von möglichen Zuständen, welche Software-Systeme bzw. Legacy-Systeme durchlaufen, dargestellt wird.

Wird der Software-Lebenszyklus zunächst ohne die Ausprägung von parallelen Versionen betrachtet, so ergeben sich folgende Zustände:

- Software in der Anfangsentwicklung (initial development)
- Software-Evolution
- Software-Wartung und Pflege
- Stufenweise Einstellung der Software (phase-out)
- Stilllegung der Software (close-down)

In der Abbildung 2.8 wird der einfache Software-Lebenszyklus dargestellt.

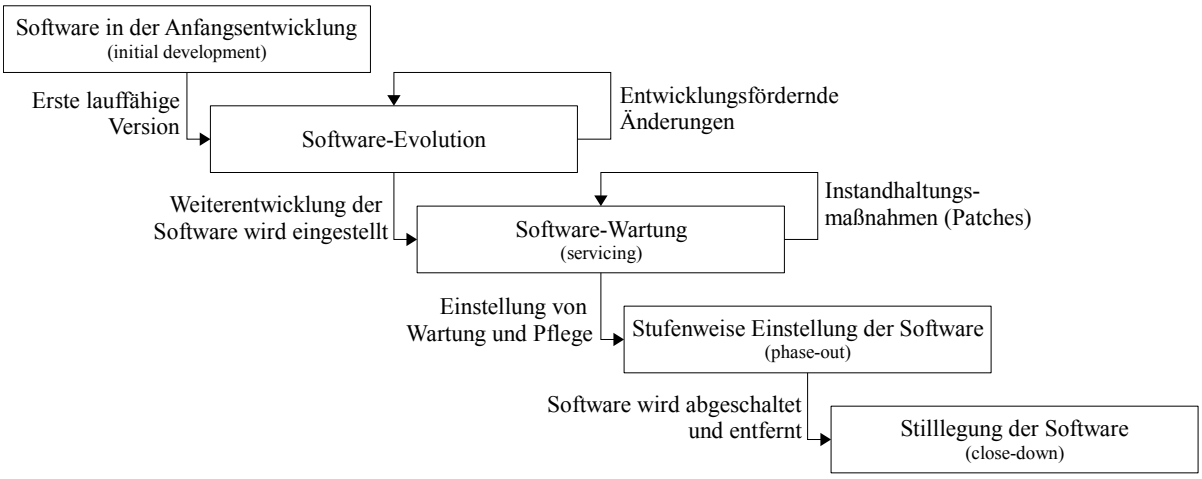


Abbildung 2.8: Der einfache Software-Lebenszyklus [RB00]

Zustand: Software in der Anfangsentwicklung. Dieser Zustand tritt ein, wenn die erste Version des Software-Systems in Betrieb genommen wird. In diesem Zustand kann man davon ausgehen, dass die Version noch nicht alle Funktionalitäten aufweist - sei es ein neues Produkt oder eine entstandene Version durch ein Reengineering. Ein wesentliches Charakteristika dieses Zustandes ist, dass sich die Architektur des Software-Systems über den gesamten Lebenszyklus nicht mehr ändert. Hierin liegt das Wissen über ein Software-System, was im Laufe eines Lebenszyklus verloren geht und dadurch ein Legacy-System kennzeichnet. [MAS07]

Zustand: Software-Evolution. Wenn die erste Version des Software-Systems eingeführt wurde und ein gewisses Maß an Domänenwissen vorhanden ist, wechselt das Software-System in den Zustand Software-Evolution. Der Zustand der Software-Evolution hat das Ziel des ständigen Anpassens von Benutzeranforderungen und der Beseitigung von Fehlern. Die haupttragende Kraft dieses Zustandes ist das Domänenwissen und die Architektur. Das Domänenwissen besteht als Wissen in den Köpfen des Fachpersonals und teilweise in Software-Dokumenten; es kann durch Personalwechsel verschwinden. Die Architektur kann durch den Zerfall des Wissens um Domänenwissen an Integrität verlieren. Verschwindet eine der beiden Voraussetzungen, dann wechselt das Software-System in die Stufe der Software-Wartung und Pflege (servicing). [MAS07]

Zustand: Software-Wartung und Pflege. In dem Zustand der Software-Wartung und Pflege liegt der Schwerpunkt in der Bereinigung von Fehlern und der Schließung von Sicherheitslücken. Zu kleineren Teilen finden noch Veränderungen statt. Die Wartungs- und Pflegeänderungen, bekannt als *Patches*, *Fixes* oder *Bugfixes*, werden in aller Regel an Endanwender ausgeliefert. Dieser Zustand repräsentiert auch eine gewisse Reife eines Software-System.

In diesem Zustand wird der Verfall von Domänenwissen oder der Verfall des Wissens um eine Architektur zu einem Problem gegenseitiger Behinderung. Je mehr das Wissen um eine Architektur verfällt, umso mehr Verständnis um Domänenwissen und eine konkrete Implementierung wird benötigt. Verfällt Domänenwissen und Wissen über eine Implementierung, erscheint das Software-System immer komplexer; es kann durch Unwissenheit zerstört werden. Unter Komplexität versteht man die steigende Anzahl an Abhängigkeiten, die durch

Unwissenheit über die Architektur entstehen. Legacy-Systeme befinden sich in den Regel in diesem Zustand. [MAS07]

Zustand: Stufenweise Einstellung der Software. Im Zustand der stufenweisen Einstellung eines Software-Systems werden keine Wartungs- und Pflegemaßnahmen mehr durchgeführt. Dies kann zum Beispiel durch die Einstellung eines Software-Systems durch den Hersteller hervorgerufen werden. Stellt ein Hersteller die Pflege eines Software-Systems ein und Endanwender benutzen das Software-System in diesem Zustand längerfristig weiter, so kommt es häufig zu Umgehungslösungen (work arounds). Diese entstehen durch Nichtverstehen des Software-Systems und Falschreinterpretationen der Semantik, was im Enddefekt zu großen Problemen und Aufwänden und zum Schluss zur Entfernung des Software-Systems führen kann. [MAS07]

Zustand: Stilllegung der Software. In diesem Zustand wird das Software-System (Legacy-System) endgültig abgeschaltet bzw. entfernt und steht nicht mehr zur Verfügung.

Der Software-Lebenszyklus mit mehreren Versionen. In der Abbildung 2.9 wird das Software-Lebenszyklusmodell um einzelne Versionen erweitert. Große evolutionäre Schritte gehen hier in eine neue Version über. Kleine evolutionäre Schritte geschehen im Verlauf einer Version, die dann automatisch in eine neue Version mitgeführt werden. Durch dieses Modell übergeht man den Zustand Software-Wartung im Sinne der Weiterentwicklung und geht gleich auf eine neue Version. Alte Versionen werden dann nur noch für eine gewisse Zeit unterstützt. [MAS07]

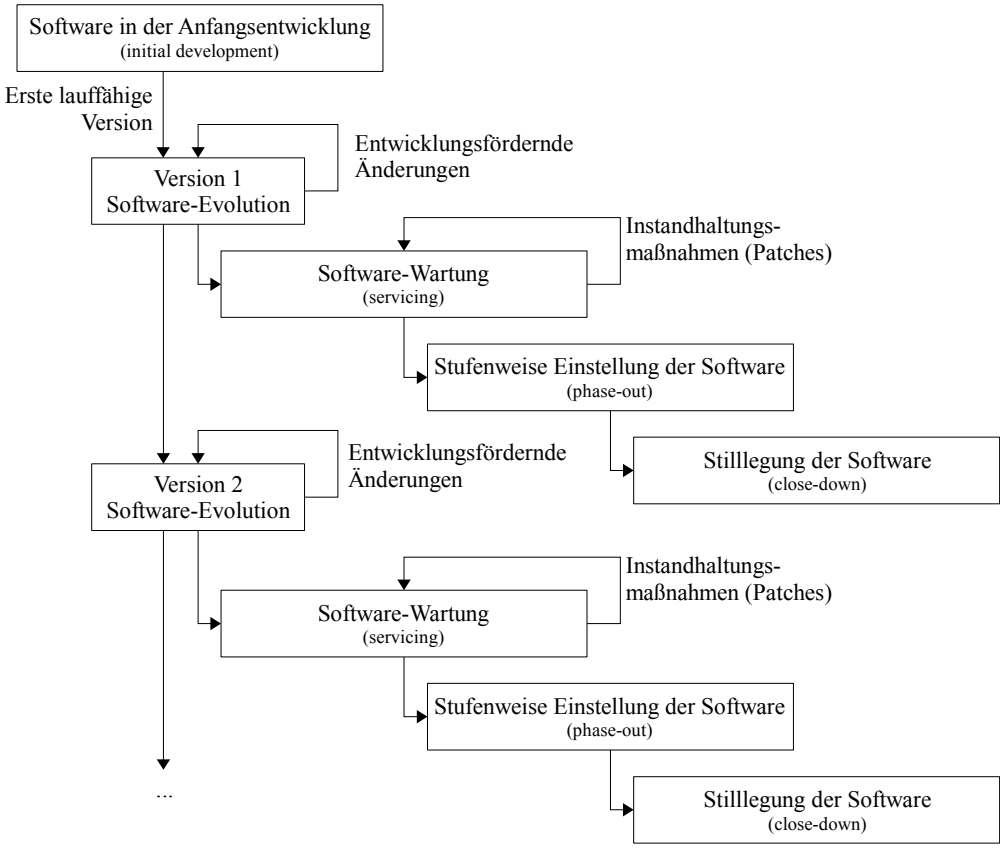


Abbildung 2.9: Der Software-Lebenszyklus mit mehreren Versionen [RB00]

Der beste Zeitpunkt für eine Modernisierung. Für die Beantwortung der Frage, wann der beste Zeitpunkt für eine Modernisierung eines Legacy-Systems ist, sollten die wesentlichen komplexen Übergänge nach Abbildung 2.10 am Software-Lebenszyklusmodell betrachtet werden.

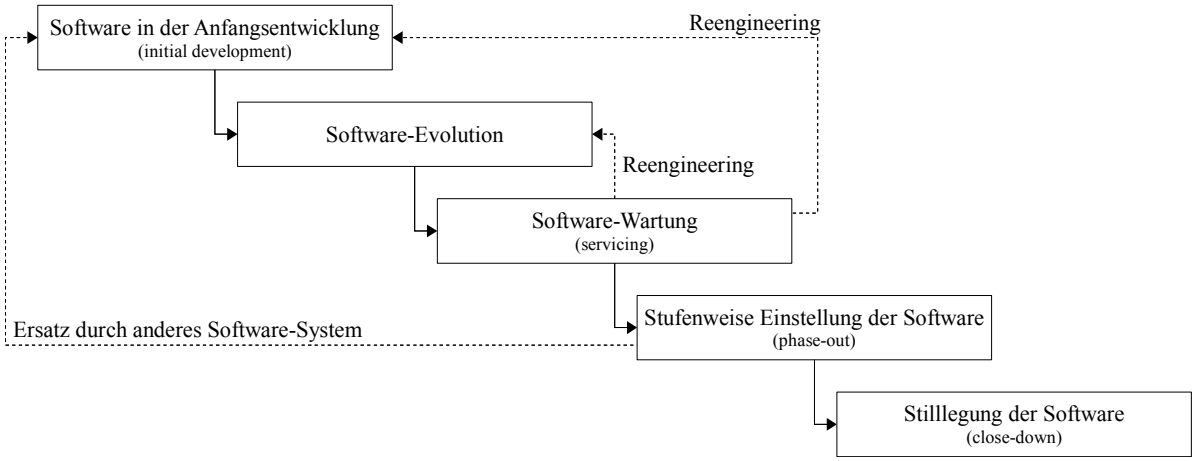


Abbildung 2.10: Komplexe Übergänge im Software-Lebenszyklus [MAS07]

Ein für die Modernisierung nicht vorteilhafter, aber möglicher Übergang, ist der Ersatz eines Software-Systems durch ein anderes. So kann ein Software-System durch eine neue Standard-Software ersetzt werden. Solche Übergänge haben nur dann Sinn, wenn sie ab der Phase Software-Wartung vollzogen werden, weil der Ersatz während der Entwicklung eines Software-Systems nicht sinnvoll sein kann.

Der beste Zeitpunkt, um ein Legacy-System zu modernisieren bzw. ein Software Reengineering durchzuführen, besteht während des Zustands Software-Wartung, weil hier der Wissensverfall voranschreitet, aber immer noch Wissen über das Software-System existiert. Je nach Technologiestand und Anforderungen sind die Übergänge vom Zustand Software-Wartung zum Zustand Software-Evolution oder zum Zustand Software in der Anfangsentwicklung zu wählen. Der Übergang zum Zustand Software-Evolution ist dann sinnvoll, wenn die Technologie den Anforderungen noch entspricht. Hier werden Teilerneuerung, Migrationen bzw. Sanierungsmaßnahmen mit Software-Reengineering-Aktivitäten durchgeführt. Architekturbezogene Änderungen sind auch möglich. Der Übergang zum Zustand Software in der Anfangsentwicklung ist dagegen durch Neuentwicklung und Migration geprägt, wo dem Stand der Technik entsprechende Technologien eingesetzt werden, die den alten und vor allem den neuen Anforderungen entsprechen.

Es ist zu beachten, dass der beste Zeitpunkt für eine Modernisierung am idealtypischen Software-Lebenszyklusmodell betrachtet wurde. Der Zeitpunkt für eine Modernisierung kann auch von anderen Faktoren abhängen. Zum Beispiel kann die Nutzungsdauer des Software-Systems, der Aufwand und die Kosten der Instandhaltung bzw. Wartung, Funktionalität und das Management eine entscheidende Rolle für den Zeitpunkt einer Modernisierung spielen.

Weiterführende Informationen über den Software-Lebenszyklus können unter [ISO12207] nachgeschlagen werden.

2.5 Zusammenfassung

Im ersten Abschnitt dieses Kapitels wurde der Begriff Legacy-System definiert. Zusätzlich zu den in der Motivation dargestellten Gründen der Modernisierungsnotwendigkeit von Legacy-Systemen, wurde die Notwendigkeit der Modernisierung mit der Definition des Legacy-Systems als soziotechnisches Gebilde verdeutlicht. Die wichtigsten Erkenntnisse aus diesem Abschnitt sind die Charakteristika, welche ein Software-System als Legacy-System ausmachen.

Nach der Legacy-System-Charakterisierung wurden Strategien beim Umgang mit Legacy-Systemen betrachtet, um eine mögliche strategische Ausrichtung für das Konzept zu ermitteln. Neben der Untersuchung möglicher Strategien wurde der Trend gewählter Modernisierungsstrategien in Unternehmen anhand einer Marktstudie über Software-Modernisierung untersucht, mit dem Ergebnis, dass eine Kombination aus Migration und Neuentwicklung als strategische Ausrichtung für das Konzept umzusetzen ist. Mit der Migration wird die Möglichkeit der Wiederverwendung von Teilen eines Legacy-Systems gesichert, wobei die Neuentwicklung zwingende oder zusätzliche Anpassungen, Ergänzungen oder Erneuerungen sichert.

Da das Konzept aus Aktivitäten des Software Reengineering aufgebaut werden soll, wurde die Terminologie des Software Reengineering vorgestellt. Die wichtigste Erkenntnis aus der Terminologie ist die Bedeutungszuordnung der unterschiedlichen Begriffe und das Verständnis um die Zusammenhänge zwischen dem Software Reengineering und der Software-Technik. Essenziell für das Konzept ist der Kreislauf des Software Reengineering nach Abbildung 2.3, weil das Konzept auf Basis des Reverse Engineering (vom Konkreten zum Abstrakten) und Forward Engineering (vom Abstrakten zum Konkreten) arbeitet.

Im letzten Abschnitt wurde der idealtypische Software-Lebenszyklus erläutert, weil an diesem die Stadien des Verfalls eines Software-Systems zum Legacy-System gezeigt werden können. Er bietet Aufschluss über den richtigen Zeitpunkt für eine Modernisierung. Der beste Zeitpunkt für eine Modernisierung ist nach dem idealtypischen Modell des Software-Lebenszykluses der Zustand Software-Wartung, weil das Wissen um ein Software-System in diesem Zustand noch besteht.

3 Konzeptkonkretisierung

Dieses Kapitel bekräftigt die Notwendigkeit des Software-Reengineering-Konzeptes und konkretisiert das Konzept durch die Festlegung von Legacy-System- und Zielsystemeigenschaften.

Im ersten Abschnitt wird einleitend die Notwendigkeit des Software-Reengineering-Konzeptes durch einen kurzen Blick auf die historische Entwicklung von Unternehmensanwendungen unterstrichen. Darauf aufbauend wird anhand einer Marktstudie die in Unternehmen noch große Anzahl an Legacy-Systemen gezeigt und in welcher Weise sich die Unternehmen dafür verantwortlich fühlen. Anhand der Marktstudie werden Eigenschaften des Legacy- bzw. Zielsystems abgeleitet. Diese Systemeigenschaften prägen das zu erstellende Konzept und konkretisieren es.

In den letzten zwei Abschnitten werden die Legacy-System- und Zielsystemeigenschaften zusammengefasst.

3.1 Analyse zur Konzeptkonkretisierung

Um die Eigenschaften des Legacy- bzw. Zielsystems zu konkretisieren und festzulegen, ist ein kurzer Blick auf die historische Entwicklung von Unternehmensanwendungen notwendig.

Bis Ende der achtziger Jahre wurden Anwendungen gezielt für spezifische Geschäftsaufgaben entwickelt, die unter anderem hoch spezialisierte monolithische Host-Anwendungen des Kerngeschäfts darstellten. Dazu gehören zum Beispiel Zahlungsverkehr, Buchungssysteme oder Vertragssysteme. Die Integration von Anwendungen als Insellösungen wurden durch eine Peer-to-Peer-Infrastruktur (P2P)[15] realisiert. In den neunziger Jahren war das Prozessdenken der treibende Faktor, was eine starke Integration von ERP[16]-Systemen nach sich zog. Doch die monolithischen Legacy-Systeme, die unter anderem das Kerngeschäft darstellten, blieben unverändert. Ende des zwanzigsten Jahrhunderts war der treibende Faktor ein gesamtheitliches Prozessdenken, was Middleware[17]-Lösungen (A2A[18], EAI[19]) und eine unternehmensübergreifende Kommunikation (B2B)[20] nach sich zog. Doch auch hier veränderten sich die monolithischen Legacy-Systeme unter anderem des Kerngeschäfts kaum.

Als populäre Stellvertreter für Legacy-Anwendungen und damit auch populäre Vertreter zur Umsetzung von Geschäftsaufgaben sind COBOL-Anwendungen zu nennen. Nach Schätzungen ist der Prozentsatz der in COBOL realisierten Anwendungen derzeit noch immer bei 50% - 60% weltweit.

Das noch große Vorkommen an Legacy-Systemen belegt auch eine Marktstudie aus der Schweiz [BP05]. In Abbildung 3.1 ist zu sehen, dass 49% der in der Marktstudie befragten

15 Die Peer-to-Peer-Infrastruktur (P2P) bezeichnet die dezentrale Vernetzung zwischen Systemen. Die Kommunikation in einem Netzwerk findet von System zu System direkt statt.
16 Ein Enterprise Resource Planning (ERP)-System ist ein umfangreiches Software-Produkt, welches die Ressourcenplanung eines Unternehmens unterstützt.
17 Eine Middleware bezeichnet ein Software-Produkt, was zwischen verschiedenen Software-Produkten vermittelt.
18 Application-to-Application (A2A) bezeichnet die Kommunikation zwischen Software-Anwendungen innerhalb eines Unternehmens.
19 Enterprise Application Integration (EAI) ist ein Konzept zur Bildung einer IT-Infrastruktur in Form einer Middleware, um verschiedene IT-Systeme bzw. Software-Anwendungen, wie z.B. ERP-Systeme, in einer Wertschöpfungskette miteinander zu koppeln.
20 Business-to-Business (B2B) bezeichnet die Kommunikation zwischen verschiedenen Unternehmen.

Unternehmen Anwendungen betreiben, die mehr als zwanzig Jahre alt sind. Deshalb kann angenommen werden, dass diese Legacy-Systeme mit prozeduralen Programmiersprachen implementiert wurden. Sie werden sich irgendwann nur noch durch eine Modernisierung an die sich ständig ändernden Anforderungen anpassen lassen.

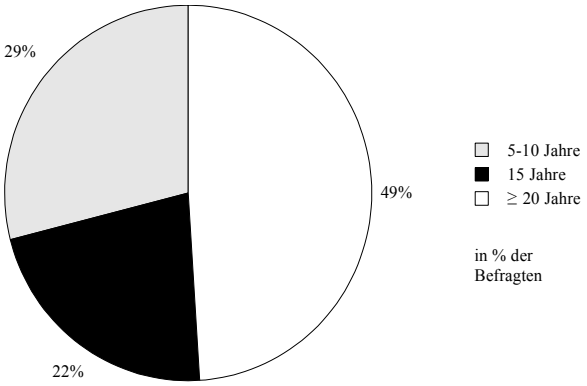


Abbildung 3.1: Alter IT-Anwendungen befragter Unternehmen [BP05]

Die Modernisierung der Legacy-Systeme ist daher eine strategische Notwendigkeit, was die Unternehmen auch wissen und erkannt haben. Diese Meinung vertritt auch der Gartner-Analyst Andy Kyte mit einem Beitrag aus dem Wirtschaftsmagazin CIO – IT-Strategie für Manager [KA08].

Nach einer Marktstudie aus der Schweiz [BP05] sieht ein überwiegender Teil der befragten Führungskräfte in ihrem Verantwortungsbereich Software-Sanierungs- und Modernisierungsvorhaben für Kern- und Legacy-Anwendungen vor (Abbildung 3.2).

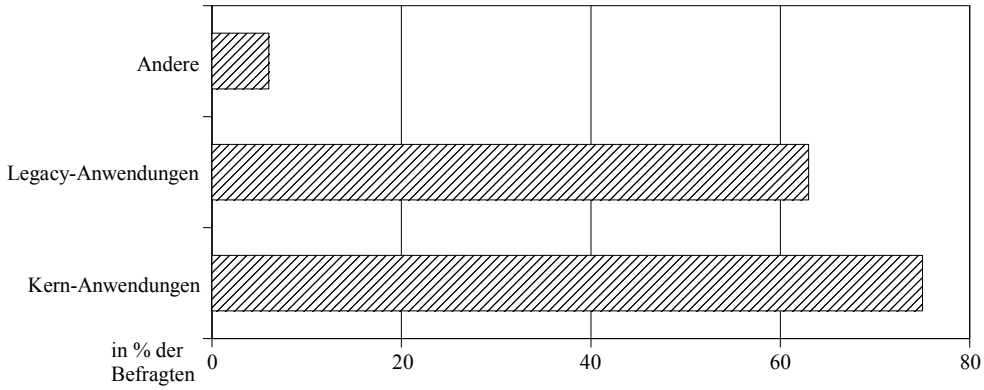


Abbildung 3.2: Anwendungsbereiche für Software-Modernisierung [BP05]

Das Ziel einer Modernisierung ist die Beseitigung jener Charakteristika, die ein Legacy-System ausmachen. Doch welche Ziele und damit Modernisierungsbeweggründe verfolgen Unternehmen?

Auch zu dieser Frage gibt die Marktstudie aus der Schweiz Antwort. Die Abbildung 3.3 zeigt, dass das Hauptziel der Unternehmen eine Modernisierung ihrer Legacy-Systeme zu zeitgemäßen Software-Architekturen ist, um neue unternehmensweite Anforderungen und Strategien umsetzen zu können, denn dem Legacy-System fehlen nötige Abstraktionsschichten, um den heutigen Anforderungen nach Wiederverwendung, Geschäftsprozessabbildung und Integration gerecht zu werden. Diesen Anforderungen kann eine Drei-Schichten-Architektur gerecht werden.

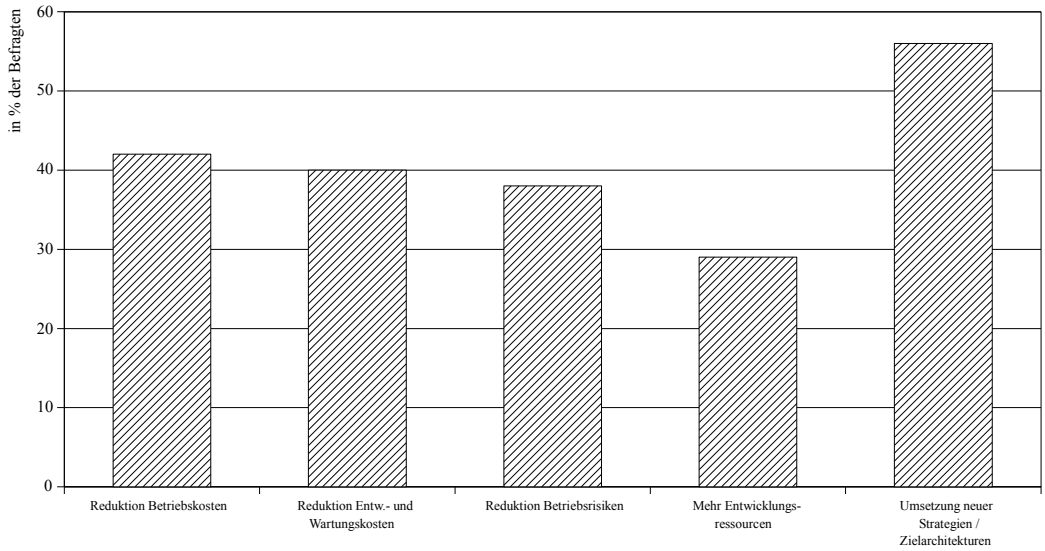


Abbildung 3.3: Ziele bei Software-Modernisierung (Schweizer-Marktstudie) [BP05]

Durch das Zugrundelegen einer Drei-Schichten-Architektur erhöht sich nicht nur die Wartbarkeit, weil ein strukturierter Aufbau und das mögliche Austauschen von Komponenten bzw. Modulen ohne die Beeinflussung der restlichen Komponenten oder Module möglich ist, sondern die Komponenten bzw. Module können auch getrennt voneinander entwickelt, getestet und gewartet werden. Zudem ist das System besser skalierbar, weil die Komponenten bzw. Module getrennt voneinander existieren.

Migrationsstrategien legen oft nur den Zeitpunkt und eine mögliche Reihenfolge zur Übernahme der Daten und Teile eines Legacy-Systems fest, ohne Verfahren und Techniken zur Transformation eines Legacy-Systems zu einem Zielsystem vorzugeben. Deshalb soll ein Konzept entwickelt werden, welches gegenüber den Migrationsstrategien konkrete Vorgänge und Techniken vorstellt, um Legacy-Systeme, die mit prozeduralen Programmiersprachen implementiert wurden, optimal in ein Zielsystem mit Drei-Schichten-Architektur zu überführen.

Um strukturiert vorzugehen, soll das Konzept in mehrere Phasen eingeteilt werden, wie das in Abbildung 3.4 dargestellt wird.

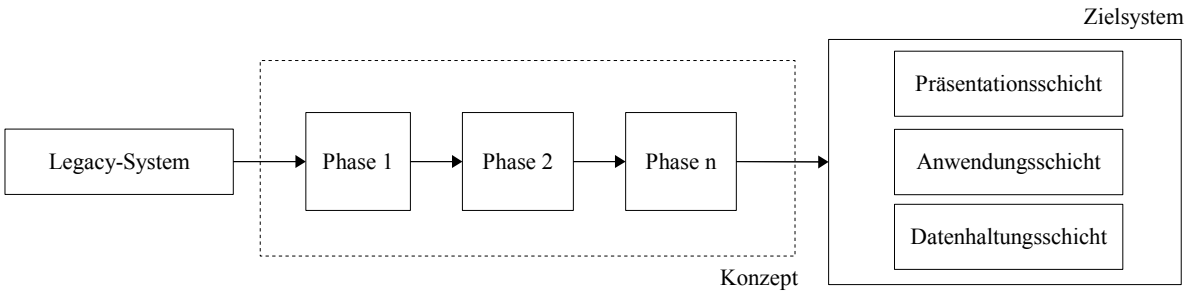


Abbildung 3.4: Soll-Konzept

In einer ersten Phase soll ausgehend von einem Legacy-System eine Informationswiedergewinnung und ein Verständnisaufbau anhand eines Reverse-Engineering-Vorgangs realisiert werden. In einer zweiten Phase sollen die wiedergewonnenen architekturbezogenen Informationen strukturiert werden, um diese unter anderem in einer dritten Phase strukturell auf die logischen Schichten der Zielarchitektur zu unterteilen. Die Unterteilung soll den Entwicklungsvorgang zum Zielsystem in einer vierten Phase verbessern, in dem Funktionalitäten des Legacy-Systems bereits im Vorfeld in Aufgabengebiete der logischen Schichten der Drei-Schichten-Architektur unterteilt werden. Ausgehend von den gesammelten Informationen über das Legacy-System, soll der Entwicklungsvorgang in einer vierten Phase den Ansatz der modellgetriebenen Software-Entwicklung nutzen. Die modellgetriebene Software-Entwicklung soll dem Wissensverfall und der Architekturzerstörung vorbeugen, in dem die Entwicklung und Wartung des Zielsystems modellgetrieben durchgeführt wird.

Das allgemeine Software-Reengineering-Konzept wird im Kapitel 4 spezialisiert und als *4-Phasen-Transformationskonzept* vorgestellt. Mit der Vorstellung werden den einzelnen Phasen gemäß Abbildung 3.5 die Aktivitäten des Software Reengineering zugeordnet.

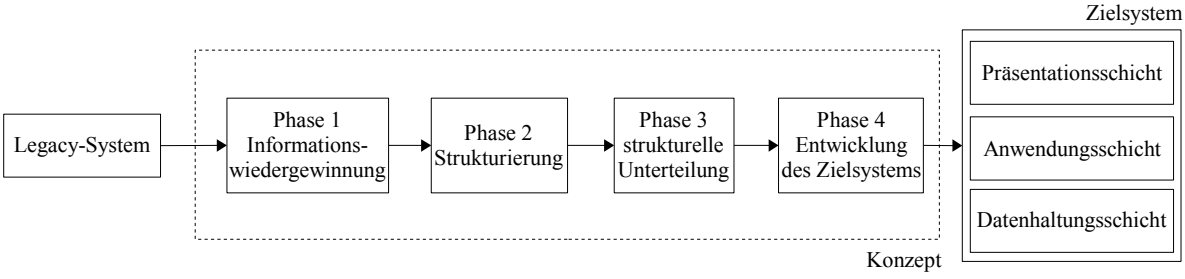


Abbildung 3.5: Soll-Konzept präzisiert

Im Weiteren werden als Anwendungsbedingung des Konzeptes die Legacy-System- und Zielsystemeigenschaften zusammengefasst und festgelegt.

3.2 Festlegung der Legacy-Systemeigenschaften

Über das Legacy-System werden folgende Eigenschaften vorausgesetzt:

- Das Legacy-System wurde mit einer prozeduralen Programmiersprache implementiert.
- Der prozedurale Programmierstil herrscht vor
- Das Legacy-System setzt kein verteiltes System um (z.B.: Client-Server-Anwendung).
- Wenn eine Datenbank als Datenhaltungsschicht verwendet wird, darf keine Anwendungslogik auf Seiten der Datenbank enthalten sein.

3.3 Festlegung der Zielsystemeigenschaften

Das Zielsystem soll folgende technologieunabhängigen Merkmale aufweisen:

- Das Zielsystem realisiert eine Drei-Schichten-Architektur mit Präsentations-, Anwendungs- und Datenhaltungsschicht (Abbildung 3.6).

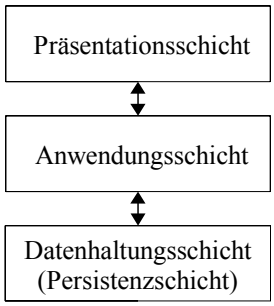


Abbildung 3.6: Drei-Schichten-Architektur

  - **Präsentationsschicht.** Die Präsentationsschicht repräsentiert die Benutzerschnittstelle und ist für die Präsentation von Daten und die Entgegennahme von Benutzereingaben zuständig.

  - **Anwendungsschicht (Logikschicht).** Die Anwendungsschicht enthält sämtliche fachlichen Funktionalitäten, um die Anwendung zu realisieren.

  - **Datenhaltungsschicht.** Die Datenhaltungsschicht ist für die dauerhafte Speicherung von Daten verantwortlich. Diese wird in der Regel durch eine Datenbank realisiert.

    Weiterführende Informationen über die Drei-Schichten-Architektur können unter [DJ08] (Kapitel 3 und 4) nachgelesen werden.

- Das Zielsystem wird mit einer objektorientierten Programmiersprache implementiert.

- In der Datenhaltungsschicht wird eine relationale Datenbank eingesetzt.

Die genaue Festlegung der Technologien, unter anderem welche Realisierungsplattform, welche Datenbank oder welche Programmiersprache verwendet wird, unterliegt dem Modernisierungsprojekt und ist je nach Anforderung frei bestimmbar. Im Laufe des Konzeptes werden Vorschläge für Realisierungsplattformen unterbreitet.

3.4 Zusammenfassung

In diesem Kapitel wurde die Notwendigkeit zur Erstellung eines Konzepts als Modernisierungsprozess unterstrichen. Anhand einer Marktstudie über Software-Modernisierung wurde gezeigt, dass Unternehmen eine große Anzahl an Legacy-Systemen betreiben und die Führungskräfte der Unternehmen in ihrem Verantwortungsbereich zukünftige Software-Sanierungs- und Modernisierungsvorhaben vorsehen. Es wurde auch gezeigt, dass zum größten Teil noch Legacy-Systeme betrieben werden, die durch ihr Alter den Rückschluss ermöglichen, dass Konzept für Legacy-Systeme zu spezialisieren, welche mit prozeduralen Programmiersprachen implementiert wurden.

Im Weiteren wurden die von Unternehmen verfolgten Ziele bzw. Modernisierungsbeweggründe untersucht, um das Konzept auf die Ziele auszurichten. Die Untersuchung ergab, dass Unternehmen auf neue Zielarchitekturen setzen, um neue unternehmensweite Anforderungen und Strategien umsetzen zu können. Diesen Anforderungen wird eine Drei-Schichten-Architektur gerecht und wurde als Zielsystemeigenschaft hinzugefügt.

Nach der Festlegung der wichtigsten Konzeptbedingungen, wurde das Konzept mit den Phasen konkretisiert. Dabei wurden die übergeordneten Aufgaben der Phasen vorgestellt. Eine Neuerung des Konzepts, gegenüber den Migrationsstrategien, ist die explizite Angabe und Unterstützung von Verfahren bzw. Techniken in den Phasen.

Im letzten Abschnitt dieses Kapitels wurden die Legacy- und Zielsystemeigenschaften in einer Übersicht zusammengefasst. Damit wurden die Bedingungen zur Anwendung des Konzepts festgelegt.

4 Das 4-Phasen-Transformationskonzept

In diesem Kapitel wird das Software-Reengineering-Konzept als *4-Phasen-Transformationskonzept* vorgestellt und beschrieben. Damit wird gezeigt, wie ein konkretes Konzept zur Modernisierung eines Legacy-Systems aussehen kann.

Im ersten Abschnitt wird eine Übersicht über das Konzept gegeben. Diese Übersicht stellt zu einem die Anwendungsbedingungen, Ziele und die Orientierung des Konzeptes dar; zum anderen wird der Ablauf des Konzeptes mit seinen Phasen Analyse, Redesign, Schichtentrennung und Forward Engineering als Gesamtübersicht kurz vorgestellt. Im weiteren Ablauf dieses Kapitels werden die einzelnen Phasen detailliert beschrieben.

Jeder Abschnitt einer Phase startet mit einer Beschreibung der Ziele. Durch eine anschließende einheitliche EVA-Prinzip[21]-Untergliederung, wird eine strukturierte Abarbeitung ermöglicht. So werden für jede Phase verschiedene konkrete Techniken bzw. Verfahren vorgestellt, die die Ziele einer Phase umsetzen und damit eine Modernisierung unterstützen. Damit die Techniken bzw. Verfahren die Ziele umsetzen können, werden für jede Phase bestimmte Software-Dokumente als Eingangs-Dokumente vorausgesetzt. Nach der Verarbeitung der Eingangs-Dokumente werden die Resultate einer Phase durch Ergebnis-Dokumente beschrieben, die wiederum als Eingangs-Dokumente für eine nachfolgende Phase verwendete werden können.

21 Das EVA-Prinzip (Eingabe, Verarbeitung, Ausgabe) ist das Grundarbeitsprinzip der klassischen elektronischen Datenverarbeitung.

4.1 Übersicht über das Konzept

Anwendungsbedingungen des Konzeptes. Um das Konzept anwenden zu können, müssen die für ein Legacy-System festgelegten Eigenschaften aus Kapitel 3.2 gelten. Zudem ist das Konzept zur Überführung in eine Drei-Schichten-Architektur ausgelegt und nutzt in der Phase Forward Engineering den Ansatz der modellgetriebenen Software-Entwicklung. Die Eigenschaften des Zielsystems wurden im Kapitel 3.3 festgelegt.

Ziel des Konzeptes. Das Ziel des Konzeptes ist die strukturierte systematische Modernisierung eines bestehenden Legacy-Systems, in dem konkrete Phasen vorgegeben werden, um ein Legacy-System schrittweise in ein funktional äquivalentes und technologisch zeitgemäßes Zielsystem zu überführen. Ein Teilziel ist die Umsetzung der wiedergewonnenen Anforderungen aus dem Legacy-System ins Zielsystem, sofern ihre Wiederverwendung sinnvoll und möglich ist. Zur Umsetzung werden in den Phasen spezielle Techniken vorgestellt, die eine teilweise automatisierte Unterstützung des Modernisierungsprozesses ermöglichen können. Zusätzlich soll der Zeitaufwand durch die eingesetzten Techniken vermindert werden.

Konzeptorientierung. Das Konzept orientiert sich beim Ablauf der Phasen an den Aktivitäten des Software Reengineering (Kapitel 2.3, Abbildung 2.3). Des Weiteren liegt der Schwerpunkt des Konzepts im Bereich der (System-)Architektur. Das Konzept ist strategisch eine Kombination aus Neuentwicklung und Migration.

Ablauf des Konzeptes

Wie in Abbildung 4.1 gezeigt, besteht das *4-Phasen-Transformationskonzept* aus den Phasen *Analyse*, *Redesign*, *Schichtentrennung* und *Forward Engineering*. Das Konzept ist auf dem Pipeline-Prinzip[22] aufgebaut.

22 Das Pipeline-Prinzip beschreibt allgemein, dass ein großer Schritt in mehrere nacheinander ausgewiesene Teilschritte zerlegt wird, wobei jeder Teilschritt separat bearbeitet wird. Ein Teilschritt kann nur betreten werden, wenn das Ergebnis des vorherigen Teilschrittes vorliegt.

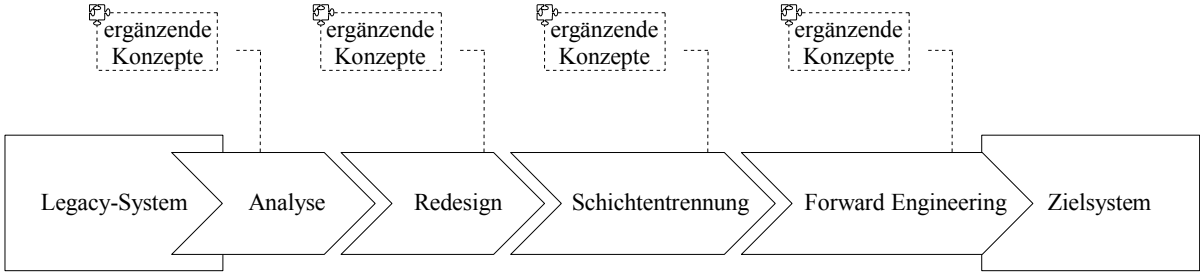


Abbildung 4.1: Konzept – Gesamtübersicht (schematische Darstellung)

Das Betreten einer Phase hat zur Bedingung, dass das Ergebnis-Dokument der vorangegangenen Phase vorliegt. Eine Ausnahme bildet die Phase Analyse. Hier gelten als Eingangs-Dokumente alle relevanten Software-Dokumente über das Legacy-System sowie das lauffähige Legacy-System selbst.

Ergänzende Konzepte. Im Arbeitsprozess der Phasen kann die Bearbeitung umfangreicher oder zusätzlicher Teilaufgaben notwendig sein, für die wiederum eigene Konzepte vorherrschen und verwendet werden können. Zum Beispiel können in der Phase Analyse spezielle Konzepte zur Extrahierung von Datenstrukturen aus Datenbankmodellen bzw. Dateien oder in der Phase Forward Engineering Konzepte die Migrationsstrategien beschreiben verwendet werden. Aus diesem Grund werden für jede Phase ergänzende (externe) Konzepte zugelassen, die dass Gesamtkonzept unterstützen und erweitern.

Mit der Zulassung ergänzender Migrationsstrategien ist auch die geregelte Übernahme der Daten des Legacy-Systems und der Zeitpunkt des Übergangs von Legacy- zu Zielsystem sichergestellt.

Im Folgenden wird der Ablauf der einzelnen Phasen, welcher in der Abbildung 4.2 als detaillierte schematische Gesamtübersicht dargestellt ist, kurz erläutert.

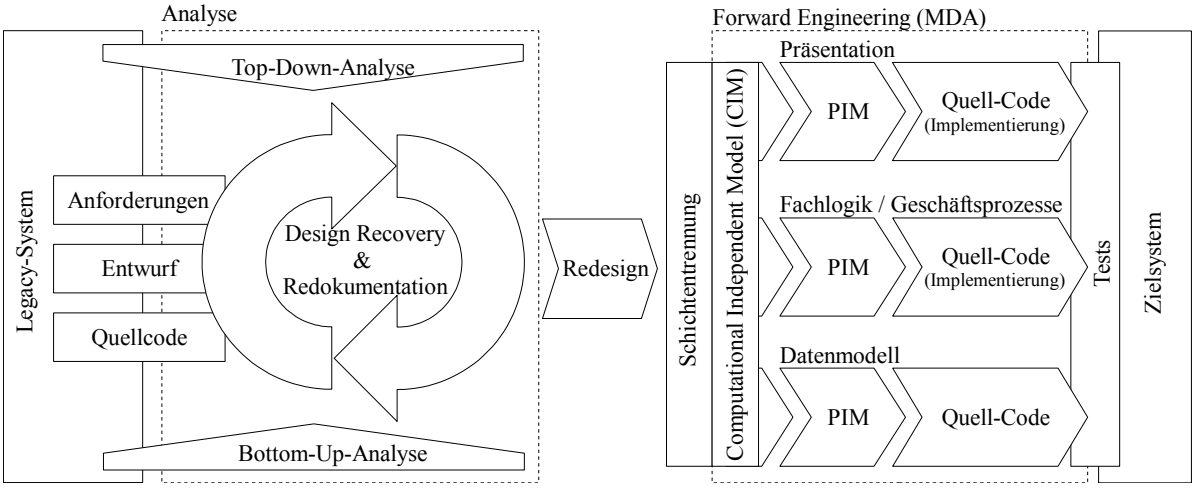


Abbildung 4.2: Konzept (Vorgehensmodell) – detaillierte Gesamtübersicht

Analyse. Die Phase Analyse dient hauptsächlich zur Erlangung des Systems- bzw. Programmverständnisses, wobei Methoden des Reverse Engineering zum Einsatz kommen. Das Ziel ist die Wiedergewinnung von Anforderungen, Fachlogik (Geschäftsprozesse) – Redokumentation und die Wiedergewinnung von Entwurfsinformationen (u.a. vorliegende Architektur, Datenstrukturen des Systems) – Design Recovery. In einem iterativen Prozess werden diese Ziele durch eine Top-Down-Analyse und eine Bottom-Up-Analyse erreicht. In der Top-Down-Analyse wird das Legacy-Systems ausgehend vom laufenden System und anhand von abstrakteren Software-Dokumenten untersucht. In der Bottom-Up-Analyse wird das Legacy-System ausgehend vom Quell-Code untersucht. Die Erkenntnisse beider Analyseteile bilden das Ergebnis-Dokument.

Redesign. Die Phase Redesign dient der Verbesserung der Struktur des Legacy-Systems, um Abhängigkeiten zwischen Artefakten sichtbar zu machen und Artefakte die eine funktionale Einheit bilden zusammenzufassen. Je nach Ebene können Artefakte Prozeduren, Funktionen, Module, Dateien, Programme sogar Pakete sein. Als Eingangs-Dokument werden die von der Analyse gewonnenen Erkenntnisse erwartet, vor allem die Erkenntnisse über die Systemarchitektur. Die Übersicht über eine mögliche Modulbildung der Artefakte dient als Ergebnis-Dokument.

Schichtentrennung. Die Phase Schichtentrennung dient zur strukturellen Unterteilung der Artefakte und oder der ermittelten Module aus der Phase Redesign in die drei logischen

Schichten der Drei-Schichten-Architektur, um die Aufgabengebiete der strukturierten Artefakte für das Zielsystem zu identifizieren und damit eine Entwicklungsbasis zu erhalten. Als Eingangs-Dokumente dienen die Ergebnis-Dokumente der Phase Analyse und der Phase Redesign. Die Einteilung der Artefakte bzw. Module in die drei Schichten bilden das Ergebnis-Dokument der Phase Schichtentrennung.

Forward Engineering. In der Phase Forward Engineering wird auf Basis der wiedergewonnenen bzw. auch neuen Anforderungen und der wiedergewonnenen Entwurfsinformationen, sowie der Informationen der Schichteneinteilung der Artefakte bzw. Module, dass Zielsystem nach dem Ansatz der modellgetriebenen Software-Entwicklung (MDA-Strategie Kapitel 4.5.3) entwickelt.

In einem ersten Schritt wird ein plattformunabhängiges Modell (Platform Independent Model (PIM)) aus den wiedergewonnenen Anforderungen, Entwurfsinformationen und Datenstrukturen erstellt. In einem zweiten Schritt wird aus dem plattformunabhängigen Modell ein Quell-Code-Grundgerüst erzeugt, welches als Basis zur Implementierung dient. Ziel der Phase Forward Engineering ist die Entwicklung des Zielsystems, welches zum Legacy-System funktional äquivalent ist.

Phasenuntergliederung nach EVA-Prinzip

Für ein zielgerichtetes Abarbeiten jeder einzelnen Phase, werden diese in folgende Teilabschnitte untergliedert:

Ziele einer Phase. In diesem Abschnitt werden die Ziele einer Phase beschrieben und welchen Stellenwert eine Phase für das Konzept darstellt.

Eingangs-Dokumente. Dieser Abschnitt beschreibt die Eingangs-Dokumente, welche für eine Phase erforderlich sind. Nur wenn die erforderlichen Eingangs-Dokumente vorhandenen sind, kann eine Phase durchgeführt werden.

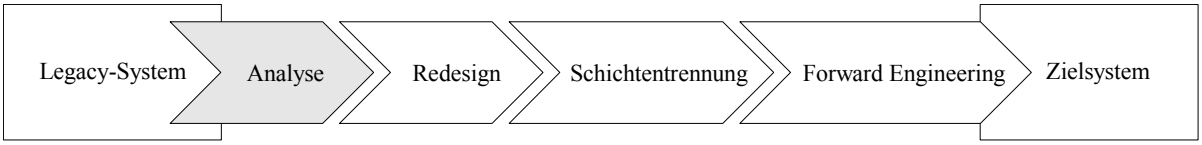
Vorgänge und Techniken. In diesem Abschnitt werden die in einer Phase notwendigen Vorgänge bzw. Maßnahmen beschrieben, um aus den Eingangs-Dokumenten die gewünschten Ergebnis-Dokumente zu erstellen und damit die Ziele einer Phase zu erfüllen. Zudem

werden Techniken bzw. Methoden beschrieben, die bei den Vorgängen verwendet werden. Es werden nur Techniken beschrieben, die zur Erfüllung des Zieles notwendig sind. Des Weiteren werden hier Werkzeuge genannt, die eine Automatisierung ermöglichen.

Ergebnis-Dokumente. Dieser Abschnitt beschreibt die Ergebnis-Dokumente und die damit verbundenen Resultate der Vorgänge in einer Phase.

Im Folgenden werden die einzelnen Phasen des Konzeptes detailliert beschrieben.

4.2 Phase 1 – Analyse



4.2.1 Ziele der Phase Analyse

Das Ziel der Phase Analyse ist das Wiedergewinnen des Verständnisses über das Legacy-System. Dies beinhaltet die Wiedergewinnung des Aufbaus und des Verhaltens des Legacy-Systems. Für die Dokumentation bzw. Belegung des Erkenntnisvorganges werden bei den Untersuchungen des Legacy-Systems Software-Dokumente auf einem abstrakten Niveau erstellt. Abstraktes Niveau bedeutet die Wiedergewinnung und nachfolgende Redokumentation der Anforderungen (funktionale, nichtfunktionale) und Fachlogik bzw. Geschäftsprozesse, sowie die Wiedergewinnung des Entwurfs des Legacy-Systems.

Die Wiedergewinnung des Entwurfs wird in der Teilphase *Design Recovery* realisiert. Hier werden vorwiegend statische Programmanalysen durchgeführt. Folgende Punkte sollen berücksichtigt werden:

- Systemarchitektur
- Komponenten
- Datenstrukturen
- Systeminterne und -externe Schnittstellen
- (Benutzerschnittstelle)

Die Wiedergewinnung der Benutzerschnittstelle ist Teilaufgabe der Phase Schichtentrennung; hier können auch mögliche Technologien, die die Benutzerschnittstelle umsetzen, identifiziert werden.

In der Teilphase *Redokumentation* sollen folgende Punkte berücksichtigt werden:

- funktionale Anforderungen
- nichtfunktionale Anforderungen

Vor der eigentlichen Analyse des Legacy-Systems wird in der Regel der Aufwand für eine Modernisierung und die Komplexität unter anderem anhand von Software-Metriken[23] beurteilt. Im Zuge dieser Arbeit wird auf Software-Metriken nicht weiter eingegangen, weil Betrachtungen über Software-Metriken den Rahmen dieser Arbeit überschreiten würden. Informationen über Software-Metriken und ihre Bewertung können unter [TG00] nachgeschlagen werden. Zudem wird unter [WS03] (Beitrag 21) ein Vorschlag für eine Aufwandsschätzung von Software-Reengineering-Projekten gegeben.

4.2.2 Eingangs-Dokumente

Als Eingangs-Dokumente der Phase Analyse zählen alle vorhandenen textlich erfassten Dokumente über ein Legacy-System, Mitarbeiter mit Kenntnissen über das Legacy-System und das laufende Legacy-System selbst. Die wichtigsten Software-Dokumente werden im Folgenden aufgezählt.

- **Handbücher und Dokumentationen**
  u.a. Benutzer-, Administrative-, Entwickler-Handbücher/Dokumentationen, Schulungsunterlagen, Pflichtenhefte, Entwurfsdokumente, Programmierschnittstellenbeschreibungen (API[24])

23 Eine Software-Metrik ist eine Funktion, die eine Software-Einheit in einen Zahlenwert abbildet. Dieser berechnete Wert ist interpretierbar als der Erfüllungsgrad einer Qualitätseigenschaft der Software-Einheit. (IEEE Standard 1061-1998)

24 Application Programming Interface (API) bezeichnet eine Programmierschnittstelle, sowie dass Dokument in der die Programmierschnittstelle beschrieben ist.

- **Quell-Code**
  u.a. Quell-Code-Kommentare, Quell-Code-Bezeichner, Implementierung

- **Mitarbeiter mit Wissen über das Legacy-System (Domänenwissen)**
  u.a. Entwickler, Fachpersonal

- **Anwender mit Wissen über das Legacy-System (Anwenderwissen)**

- **Das laufende Legacy-System**

4.2.3 Vorgänge und Techniken

Der erste Schritt der Phase Analyse besteht aus der analytischen Untersuchung des Legacy-Systems. Dafür können zwei wesentliche Strategien beim Verständnisprozess dienlich sein. Zu einem ist das die Top-Down-Analyse und zum anderen ist das die Bottom-Up-Analyse.

Top-Down-Analyse. Die Top-Down-Analyse ist eine Strategie, bei der von einem abstrakten Niveau beginnend eine schrittweise Konkretisierung vorgenommen wird. Ausgehend von Software-Dokumenten oder vom laufenden Legacy-System sollen Erkenntnisse über das Legacy-System ermittelt werden. Bei Problembereichen werden in der Regel allgemein gehaltene Hypothesen aufgestellt, welche man durch Hinweise in Software-Dokumenten und bis auf die Quell-Code-Ebene herunter zu belegen bzw. zu konkretisieren versucht. Allerdings wird hier der Quell-Code nicht detailliert analysiert, sondern dieser dient der Belegung einer Hypothese, um die konkrete Struktur bzw. eine konkrete Funktion für ein Problembereich zu benennen.

Bottom-Up-Analyse. Die Bottom-Up-Analyse ist eine Strategie, bei der vom konkreten beginnend Abstraktionen auf höherem Niveau gebildet werden. In diesem Fall wird ausgehend vom Quell-Code eine Rückgewinnung von abstrakten Konzepten, die unabhängig von der Implementierung sind, versucht. Abstrakte Konzepte sind in diesem Fall der Entwurf und die Anforderungen des Legacy-Systems.

Beide Strategien sollten in einer Kombination (Meet-in-the-Middle) angewendet werden, um den Erkenntnisvorgang zu beschleunigen und um das Verständnis so detailliert wie möglich aufzubauen. Die in der Anwendung dieser Strategien erworbenen Erkenntnisse fließen in das Design Recovery und in die Redokumentation.

Design Recovery

Das Design Recovery, als Teilschritt der Phase Analyse, dient der Wiedergewinnung des Entwurfes eines Legacy-Systems in Zuhilfenahme aller verfügbaren Software-Dokumente und Mitarbeiter. Somit sollen Informationen über die Funktionalität, die Arbeitsweise und die Zusammenhänge bzw. Abhängigkeiten der Bestandteile eines Legacy-Systems wiedergewonnen werden. Der Wiedergewinnungsprozess wird durch die Top-Down- und Bottom-Up-Analyse geprägt. Sofern noch Entwurfsdokumente vorhanden sind können diese als Verständnisgrundlage verwendet werden. Jedoch ist immer davon auszugehen, dass Änderungen am Legacy-System im Zuge von Entwicklung und Wartung nicht dokumentiert wurden und deshalb die vorherrschende Architektur mit dem ursprünglichen Entwurfsdokument differiert.

Zur Wiedergewinnung von Entwurfsinformationen, die der vorherrschenden Situation entsprechen, werden abstrakte graphische Darstellungen aus dem Quell-Code des Legacy-Systems erzeugt. Abstrakte graphische Darstellungen können das Aufzeigen von Abhängigkeiten (Interaktionen) zwischen verschiedenen Bestandteilen eines Systems verbessern. So sind auch verschiedene Sichtweisen der Abhängigkeiten möglich. Zudem können sie unbenutzte Bestandteile (zum Beispiel unbenutzte Prozeduren bzw. Funktionen) aufzeigen oder bezüglich des Aufrufverhaltens zentrale oder periphere Bestandteile ermitteln.

Eine mögliche graphische Darstellung mit verschiedenen Sichtweisen kann der Kontrollflussgraph liefern, welcher im Folgenden vorgestellt wird.

Kontrollflussgraph. Mit einem Kontrollflussgraphen kann das Zusammenwirken der funktionalen Bestandteile eines Legacy-Systems analysiert werden. Ein Kontrollflussgraph ist ein gerichteter Graph und besteht aus einer Menge an Knoten und Kanten. Die Knoten sind durch gerichtete Kanten miteinander verbunden und können je nach Abstraktionsniveau

unterschiedlich definiert werden. Es wird zwischen folgenden zwei Darstellungsebenen von Kontrollflussgraphen unterschieden:

- **intraprozeduraler Kontrollflussgraph**

  Auf der Ebene des intraprozeduralen Kontrollflussgraphen sind die in Beziehung darzustellenden Knoten, Anweisungen oder auch Anweisungsblöcke. Solche Anweisungen, wie unter anderem Schleifen- oder Bedingungskonstrukte zur Steuerung des Kontrollflusses oder Variablenzuweisungen, stellt jede Programmiersprache zur Verfügung. Aus der Ausführungsreihenfolge der Anweisungen ergibt sich der intraprozedurale Kontrollflussgraph, wie das zum Beispiel in Abbildung 4.3 dargestellt wird.

```
01  less(a,b)
02  {
03  if (a < b){
04  return a;
05  }
06  else if (b < a){
07  return b;
08  }
09  else {
10  ...
11  }}
```

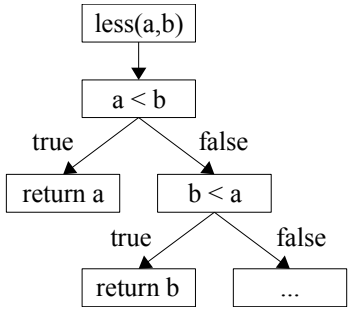


Abbildung 4.3: Beispiel – intraprozeduraler Kontrollflussgraph

Intraprozedurale Kontrollflussgraphen eignen sich gut zur Wiedergewinnung von Fachlogik bzw. Geschäftsprozessen aus Prozeduren bzw. Funktionen. Hier können implementierte Berechnungen und Abläufe graphisch ermittelt werden.

- **interprozeduraler Kontrollflussgraph**

  Auf der Ebene des interprozeduralen Kontrollflussgraphen sind die in Beziehung darzustellenden Knoten Prozeduren bzw. Funktionen. Der interprozedurale Kontrollflussgraph ergibt sich aus dem gegenseitigen Aufrufen der Prozeduren bzw. Funktionen. Deshalb wird die Darstellung solcher Aufrufbeziehungen als Aufrufgraph bezeichnet. Der Unterschied zum intraprozeduralen Kontrollflussgraph ist, dass beim Aufruf einer Prozedur bzw. Funktion immer der Rücksprung zum Aufrufort folgt. Immer dann wenn eine Prozedur bzw. Funktion den Aufruf einer Prozedur bzw. Funktion enthält, wird eine Kante zwischen aufrufender und aufgerufener

Prozedur bzw. Funktion gezogen. Ein Beispiel eines Aufrufgraphen wird in Abbildung 4.4 dargestellt.

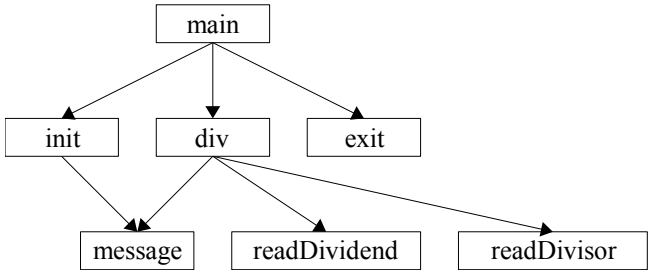


Abbildung 4.4: Beispiel – Aufrufgraph (Prozedur bzw. Funktionsebene)

Darstellung höhere Abstraktionsebenen. Aufrufgraphen können unter anderem auch zur Darstellung von Aufrufbeziehungen auf der Ebene von Modulen, Dateien oder Paketen verwendet werden, welche die Knoten eines Aufrufgraphen darstellen. Zu beachten ist, dass mit höherer Abstraktionsebene die Detailgenauigkeit abnimmt, aber auch die Übersichtlichkeit zunehmen kann. In der Abbildung 4.5 wird ein Aufrufgraph dargestellt, bei dem eine zu niedrige Abstraktionsebene gewählt wurde.

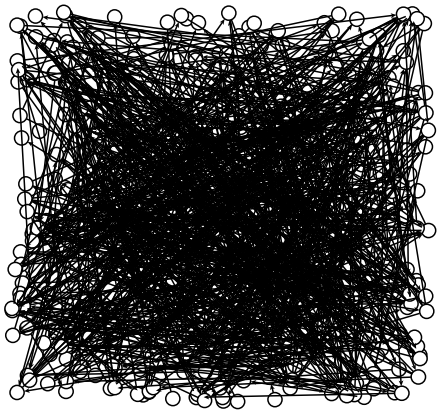


Abbildung 4.5: Beispiel – zu niedrige Abstraktionsebene

Legacy-Systeme können aus einer Vielzahl von zum Beispiel Prozeduren bzw. Funktionen bestehen, die im zweidimensionalen einen unübersichtlichen Aufrufgraphen nach sich ziehen können. Hier bietet sich eine Darstellung auf einer höheren Ebene an - zum Beispiel auf Ebene von Modulen. Die Abstraktionsebene sollte so gewählt werden, dass eine sinnvolle und aussagekräftige Darstellung entsteht.

Mit einem Aufrufgraphen können Aufrufhierarchien bei prozeduralen Programmiersprachen sehr gut dargestellt werden. Um aber den Datentransfer zwischen den Knoten darstellen zu können, muss der Aufrufgraph durch Datentransferinformation erweitert werden. Die Kombination von Aufrufgraph und Darstellung des Datentransfers zwischen den Knoten wird Strukturdiagramm (structure chart) genannt. Die Erweiterung wird im Folgenden kurz vorgestellt.

Strukturdiagramm. Der Aufbau des Strukturdiagramms ist analog dem Aufrufgraphen. Der Datentransfer zwischen den Knoten wird durch Pfeile entlang der Kanten dargestellt. Zudem können globale Variablen als separate Knoten mit entsprechenden Datenflusspfeilen dargestellt werden. In Abbildung 4.6 wird ein Beispiel eines Strukturdiagramms vorgestellt.

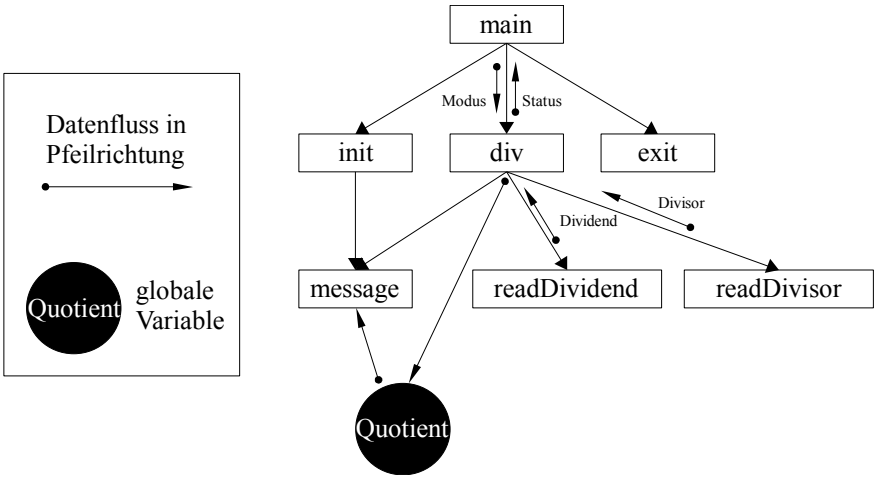


Abbildung 4.6: Beispiel – Strukturdiagramm (Prozedur bzw. Funktionsebene)

Bei sehr komplexen und großen Graphen bietet sich statt der Datenflusspfeile eine Beschriftung der gerichteten Aufrufkanten an. So zum Beispiel können mit dem Prädikat *in:* die Aufrufparameter und mit dem Prädikat *out:* die Rückgabeparameter dargestellt werden.

Neben der Darstellung der Aufrufhierarchie und der Darstellung des Datentransfers können durch Analysen des Kontrollflussgraphen (Kontrollflussanalyse) Anomalien festgestellt werden. Typische Anomalien wären die Erkennung von nicht erreichbaren Quell-Code (toter Code) oder duplizierter Quell-Code. Zudem können Datenflussanomalien aufgedeckt werden, wie zum Beispiel nicht initialisierte oder verwendete Variablen (Datenflussanalyse).

Datenstrukturen. Datenstrukturen und globale Variablen, die im Quell-Code direkt definiert sind, können durch das Strukturdiagramm ermittelt werden. Die Wiedergewinnung von Datenstrukturen aus Datenbanken oder Dateien ist eine umfangreiche Teilaufgabe, die auf ergänzende Konzepte verteilt wird. Information über Datenbank Reverse Engineering können unter [DA00] nachgeschlagen werden. Ein unterstützendes Werkzeug zur einfachen Wiedergewinnung eines Entity Relationship Model[25] (ER-Modell) und Meta-Informationen aus Datenbanken kann unter Tabelle 4.1 nachgeschlagen werden.

Anwendungslogik. Zur Ermittlung konkreter Implementierungen, konkreter Ablauflogik, kann der intraprozedurale Kontrollflussgraph verwendet werden. Abstraktere Diagramme, die direkt für die Darstellung von Algorithmen oder Programmablauflogik verwendet werden können, sind:

- Programmablaufplan (PAP) [DIN66001]
- Nassi-Shneiderman-Diagramm (Struktogramm) [DIN66261]

Diese Diagramme sind genormt und unter der angegebenen DIN-Nummer[26] zur Einsicht verfügbar.

Redokumentation

Die Redokumentation als Teilschritt der Phase Analyse dient hier der Dokumentation der durch die Top-Down- und Bottom-Up-Analyse wiedergewonnenen Anforderungen aus dem Legacy-System. Zu dokumentieren sind funktionale und nichtfunktionale Anforderungen, welche eine Aussage über die zu erfüllenden Eigenschaften und Leistungen des Legacy-Systems machen. Dabei beschreiben funktionale Anforderungen (*operational requirements, behavioral requirements)* das Verhalten und die nichtfunktionalen Anforderungen (*non behavioral* oder *non operational requirements*) allumfassende Eigenschaften und Qualitäten des Zielsystems.

25 Ein Entity Relationship Model (ERM) ist ein Modell bestehend aus einer graphischen Darstellung von Datenbeziehungen und deren Beschreibung. Es gibt eine Vielzahl von möglichen graphischen Darstellungsarten: u.a. Chen-Notation, Bachman-Notation, Min-Max-Notation/ISO, Krähenfuß-Notation, UML (UML wird hier als Standard verwendet)

26 Das Deutsche Intitut für Normung (DIN) ist die nationale Normungsorganisation für Deutschland.

Funktionale Anforderungen können graphisch durch die UML über Anwendungsfalldiagramme (use-case) dokumentiert werden. Eine ausführliche Beschreibung dieses Diagramms ist unter [UML09] verfügbar. Zudem ist die textbasierte Dokumentation zur Beschreibung eines Anwendungsfalles notwendig. Da die Detailgenauigkeit einer textbasierten Dokumentation durch unterschiedliche Vorlagen sehr stark variieren kann, wird hier die von [CO00] eingeführte Vorlage zur textbasierten Dokumentation von Anwendungsfällen verwendet. Ein Beispiel dieser Vorlage kann im Anhang D.3 des Fallbeispiels nachgeschlagen werden. Mit der Dokumentation der Anwendungsfälle besteht nun eine Spezifikation der funktionalen Anforderungen.

Im Rahmen der Redokumentation können die mit einem Programmablaufplan dargestellten Programmablauflogiken als Basis zur Erstellung von Aktivitätsdiagrammen verwendet werden.

Die nichtfunktionalen Anforderungen nach [SOM07] (Kapitel 6.1) werden in Textform dokumentiert. Die Qualitätsanforderungen sind nach [ISO9126] aufzustellen.

Mögliche Werkzeuge zur Unterstützung der Analyse

In der Tabelle 4.1 werden Werkzeuge vorgestellt, die zur Informationserhebung in der Phase Analyse eingesetzt werden können.

Tabelle 4.1: Phase Analyse – unterstützende Werkzeuge (Stand vom: 10.08.2009)

Werkzeug	Beschreibung	Unterstützt u.a.
Doxygen [DOX09] Lizenz: GNU[27]	Für automatische Redokumentation von Entwurfsinformationen aus Quell-Code	C++, C, Java, C#, Fortran, Python, ...
Rigi [RIGI09] Lizenz: open source	Graphisches Werkzeug zur statischen Analyse von Quell-Code	C, C++, COBOL

▼

27 Die GNU General Public License (GPL) und die Lesser General Public License (LGPL) sind von der Free Software Foundation (gemeinnützige Stiftung, welche die Entwicklung und Distribution freier Software fördert) herausgegebene Lizenzen für freie Software.

Werkzeug	Beschreibung	Unterstützt u.a.
SchemaSpy [SP09] Lizenz: LGPL	Java basiertes Werkzeug zur Redokumentation von u.a. Datenbankschemata (ER-Modell) und Meta-Informationen	DB2, Informix, MSSQL, Oracle, PostgreSQL, MySQL, ...
Weitere kommerzielle Werkzeuge können unter [LP09] nachgeschlagen werden.		

Mit *objectiF* (Tabelle 4.2) wird ein Werkzeug vorgestellt, welches zur Redokumentation und zur modellgetriebenen Software-Entwicklung in der Phase Forward Engineering verwendet werden kann. Im Bereich Redokumentation können unter anderem Anwendungsfalldiagramme inklusive Beschreibungen und zugehörige Aktivitätsdiagramme angelegt werden.

objectiF Eclipse Professional Edition	**Beschreibung**	Werkzeug für objektorientierte Analyse (UML) und modellgetriebene Software-Entwicklung
	Hersteller	www.microtool.de/objectif
	Version	7.0.259
	Lizenz	kommerziell

Tabelle 4.2: Entwicklungswerkzeug: objectiF (Stand vom: 29.07.2009)

4.2.4 Ergebnis-Dokumente

Die in der Phase Analyse entstandenen Ergebnis-Dokumente sind:

- Wiedergewonnene Spezifikation der funktionalen Anforderungen
- nichtfunktionale Anforderungen
- Entwurfsinformationen über das Legacy-Systems:
  (System-)Architektur, Komponenten, Datenstrukturen, Abhängigkeiten (Datenfluss), Ablauflogik

In der Abbildung 4.7 wird die Entstehung der Ergebnis-Dokumente schematisch dargestellt.

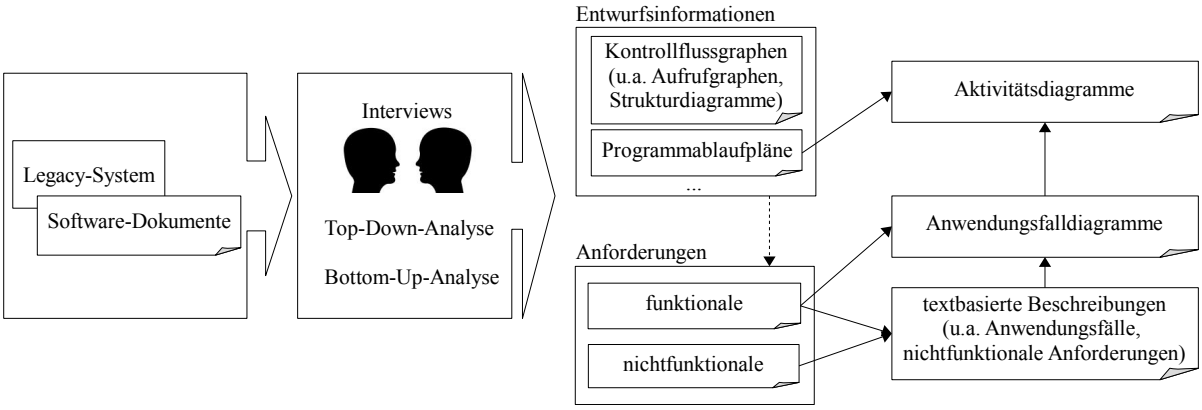
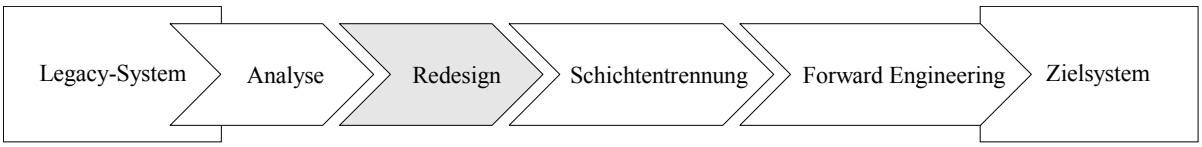


Abbildung 4.7: Übersicht Entstehung Ergebnis-Dokumente Analyse

Die Anforderungen und Entwurfsinformationen des Legacy-Systems dienen vor allem der Phase Forward Engineering als Grundlage für die Erstellung des Zielsystems.

4.3 Phase 2 – Redesign



4.3.1 Ziele der Phase Redesign

Das Ziel der Phase Redesign ist die Vertiefung des Legacy-Systemverständnisses, in dem die Struktur des Legacy-Systems sichtbar gemacht und verbessert wird. Die strukturelle Verbesserung beinhaltet deshalb folgende Frage: Gibt es enge gegenseitige Aufrufbeziehungen, die das sinnvolle Verschmelzen von Prozeduren bzw. Funktionen zu Modulen ermöglicht? Die Aggregation von Prozeduren bzw. Funktionen zu funktionalen Einheiten (Modulen) ist ein Hauptanliegen dieser Phase und für monolithisch aufgebaute Legacy-Systeme für die strukturelle Aufbereitung besonders wichtig. Es wird eine Modularisierung durchgeführt.

> „*Unter Modularisierung versteht man die Partitionierung eines Monolithen in seine funktional zusammenhängenden Module. Durch die geringere Größe der entstehenden Module sind diese besser zu überblicken, zu verstehen und zu warten.*“ ([MÜ97] S. 14)

Im Weiteren stellt diese Phase eine Basis zur Identifizierung und Bildung von Klassen dar, in dem Module als Grundlage zur Klassenfindung herangezogen werden können.

Weitere Ziele der Phase Redesign sind Erkenntnisse über den Charakter einer Prozedur, einer Funktion oder eines Moduls. So kann eine Prozedur, eine Funktion oder ein Modul Server-Charakter besitzen und eine Funktionalität anbieten oder Client-Charakter haben und Funktionalitäten nutzen. Zudem kann ein Ziel der Phase Redesign das Herauslösen von funktionalen Bestandteilen bzw. Modulen sein, um diese im Zielsystem wiederzuverwenden. Die Wiederverwendung kann soweit gehen, dass ganze Code-Teile in einer Ausgangsprogrammiersprache in eine Zielprogrammiersprache übersetzt werden. Je nach Ebene können neben Modulen auch andere Artefakte zu funktionalen Einheiten zusammengefasst werden. Zum Beispiel können Komponenten identifiziert und nach einer Entkopplung wiederverwendet werden.

4.3.2 Eingangs-Dokumente

Als Eingangs-Dokumente der Phase Redesign dienen hauptsächlich die aus der Phase Analyse wiedergewonnenen architekturbezogenen Entwurfsinformationen. Dazu zählen vor allem die erzeugten Aufrufgraphen und das wiedergewonnene Verständnis des Legacy-Systems.

4.3.3 Vorgänge und Techniken

Eine mögliche automatisierbare Technik zur Modulbildung ist die Dominanz-Analyse, welche zur Eingabe einen aus der Phase Analyse erzeugten Aufrufgraph voraussetzt. Die Dominanz-Analyse wird im Folgenden vorgestellt.

Dominanz-Analyse

Die Dominanz-Analyse ist ein Verfahren zur Zusammenfassung von Artefakten, die untereinander Abhängigkeiten bilden, so dass die strukturelle Aufteilung der Artefakte verbessert wird und die Abhängigkeiten zwischen den Artefakten durch Gruppenbildung verringert werden.

Die Dominanz-Analyse basiert auf dem graphentheoretischen Begriff der *Dominanz* [VL96] und setzt einen Flussgraphen mit folgenden Eigenschaften voraus.

- **Definition des Flussgraphen**
  Ein Flussgraph G = (N, B, w) ist ein gerichteter Graph mit einer Knotenmenge N und einer Kantenmenge B. Zudem gibt es einen Wurzelknoten w ∈ N. Alle Knoten N des Flussgraphen G sind von der Wurzel w aus erreichbar. Jeder gerichtete Graph kann in ein Flussgraph überführt werden, in dem ein Wurzelknoten zu einem gerichteten Graphen hinzugefügt oder ein Knoten als Wurzelknoten ausgezeichnet wird. Ein Wurzelknoten muss ein Knoten ohne eingehende Kanten sein.

Aufrufgraphen sind Flussgraphen, wenn auf der Prozedurebene als Wurzelknoten eine Eintrittsprozedur bzw. eine Eintrittsfunktion in ein Programm verwendet wird.

Die Dominanz-Beziehungen bzw. die Dominanz-Relation zwischen den Knoten eines Flussgraphen werden durch einen Dominanz-Baum dargestellt, welcher durch die Definition der Dominanz-Relation erstellt werden kann.

- **Definition der Dominanz-Relation**
  Gegeben sei ein Flussgraph G = (N, B, w) mit einem Wurzelknoten w ∈ N.

  Ein Knoten u *dominiert* einen Knoten v, wenn u auf allen Pfaden von w zu v liegt. Der Knoten u wird dann *Dominator* von v genannt.

  Genau dann, wenn im Flussgraph u der nächste Vorgänger von v ist bzw. alle Pfade ausgehend vom Wurzelknoten über u zu v gehen, ist im Dominanz-Baum u der direkte Vaterknoten von v.

Somit ist der Wurzelknoten w Dominator aller Knoten, weil er als Startknoten auf jedem Weg liegt.

Im Weiteren gibt es eine Unterscheidung zwischen einer starken und einer normalen Dominanz-Relation. Bei einer starken Dominanz-Relation gibt es genau einen Vorgänger zu einem Knoten. Diese wird im Dominanz-Baum als durchgezogene Linie dargestellt. Bei einer

normalen Dominanz-Relation gibt es mehrere Vorgänger zu einem Knoten. Diese wird im Dominanz-Baum als gestrichelte Linie dargestellt.

In der Abbildung 4.8 wird der Ausgangs-Flussgraph und der dazugehörige Dominanz-Baum dargestellt. Zudem wird der Dominanz-Baum mit zusammengefassten Knoten dargestellt. Die zusammengefassten Knoten bilden Gruppen bzw. Module.

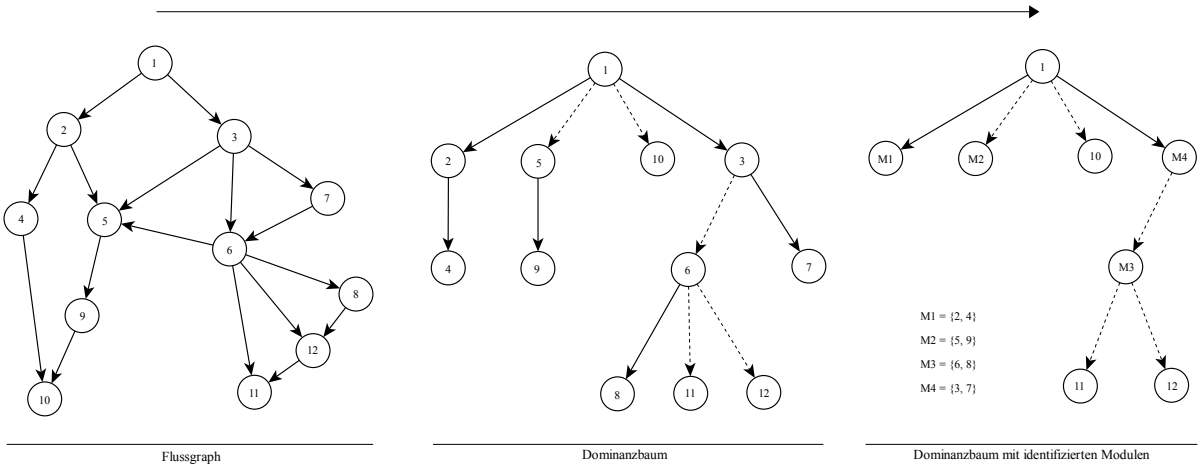


Abbildung 4.8: Graph, Dominanz-Baum, Dominanz-Baum mit identifizierten Modulen

Der durch die Dominanz-Analyse entwickelte Dominanz-Baum kann nun für eine Zusammenfassung von Artefakten zu Gruppen bzw. Modulen herangezogen werden, die eine gemeinsame Funktionalität realisieren. Für eine Gruppenbildung gibt es folgende Richtlinien [BR04]:

- Falls zwei Knoten durch starke Dominanz-Relation verbunden sind, bedeutet dies, dass die aufgerufene Funktion eine spezielle Teilfunktion der rufenden Funktion realisiert. Folglich liegt es nahe, alle Teilbäume eines Dominanz-Baumes zu Gruppen zusammenzufassen, welche ausgehend von der Wurzel des Teilbaumes lediglich starke Dominanz-Relationen enthalten.

- Falls ein Knoten eine normale eingehende Dominanz-Relation besitzt, wird die damit assoziierte Funktion von mehreren Funktionen aufgerufen. Diese Funktion stellt also einen allgemeinen Dienst zur Verfügung. Solche Funktionen können nicht unmittelbar zu größeren Gruppen zusammengefasst werden.

Die Ebene der strukturellen Aufteilung kann frei gewählt werden. Zum Beispiel kann eine Aufteilung auf Prozedur- bzw. Funktionsebene einer Programmiersprache stattfinden oder auf der Ebene von Dateien, Modulen oder Paketen.

Weiterführende Informationen und Algorithmen über die Dominanz-Analyse können unter [CV95], [CH01], [VL96] oder [BR04] nachgeschlagen werden.

Eine weitere Technik zur Modulbildung ist die Begriffsanalyse, welche im Folgenden vorgestellt wird.

Begriffsanalyse

Die Begriffsanalyse [SR99] ist ein nicht graphisches Verfahren zur Zusammenfassung von beliebig wählbaren Einheiten auf der Basis gemeinsamer Eigenschaften. Dieses Verfahren kann sowohl mit Verhaltensinformationen als auch mit strukturellen Informationen durchgeführt werden.

Prinzipiell gibt es bei der Begriffsanalyse eine Menge von Einheiten oder auch Objekten, die Eigenschaften haben. Die Menge der Eigenschaften werden der Menge von Objekten gegenübergestellt. Betrachtet man die Anwendung der Begriffsanalyse in der Phase Redesign, dann stellen Artefakte die Objekte dar. Zudem können Eigenschaft zum Beispiel die Nutzung von Variablen bzw. Datenstrukturen durch Funktionen sein. Oder repräsentative Quell-Code-Elemente werden zur Zuordnung von Funktionen zu einer logischen Schicht der Drei-Schichten-Architektur verwendet.

Um die Begriffsanalyse beispielhaft darzustellen, ist ein Programm gegeben, welches die Datenstrukturen *stack* und *queue* und mehrere Funktionen enthält. Gesucht sind inhaltlich zusammengehörige Funktionen, die zu Modulen zusammengefasst werden können und damit das Programm als System zerlegen. In Tabelle 4.3 sind die Funktionen als Objekte (O_n) und die zugehörigen Eigenschaften (E_n) dargestellt.

O_1	initStack
O_2	initQ
O_3	isEmptyStack
O_4	isEmptyQ
O_5	push
O_6	enq
O_7	pop
O_8	deq

E_1	Rückgabetyp ist die Struktur stack
E_2	Rückgabetyp ist die Struktur queue
E_3	Hat als Parametertyp die Struktur stack
E_4	Hat als Parametertyp die Struktur queue
E_5	Verwendet die Struktur stack
E_6	Verwendet die Struktur queue

Tabelle 4.3: Begriffsanalyse - Beispiel: Objekte (O) und Eigenschaften (E)

Den Inhalt der Begriffsanalyse bildet die Tabelle 4.4. Hier werden den Objekten die Eigenschaften zugeordnet, sofern ein Objekt eine Eigenschaft erfüllt. So zum Beispiel verwendet die Funktion initStack (O_1) die Struktur stack und hat diese zusätzlich als Rückgabetyp.

	E_1	E_2	E_3	E_4	E_5	E_6
O_1	✓				✓	
O_2		✓				✓
O_3			✓		✓	
O_4				✓		✓
O_5			✓		✓	
O_6				✓		✓
O_7			✓		✓	
O_8				✓		✓

Tabelle 4.4: Begriffsanalyse - Beispiel: Objekte, Eigenschaften Gegenüberstellung

Nach der Zuordnung werden die Eigenschaften so kombiniert, dass sich sinnvolle Module bzw. Gruppen bilden. Man ist bemüht, eine maximal große Menge an Objekten zusammenzufassen, die gemeinsame Eigenschaften besitzen (auch *Konzept* genannt).

Mögliche Konzepte, die sich aus der Tabelle 4.4 ergeben, werden in der Tabelle 4.5 dargestellt.

Konzeptbezeichner	Objekte	Eigenschaften	
k_1	O_2, O_4, O_6, O_8	E_6	Queue-Konzept
k_2	O_1, O_3, O_5, O_7	E_5	Stack-Konzept
k_3	O_4, O_6, O_8	E_4, E_6	isEmptyQ, enq, deq
k_4	O_3, O_5, O_7	E_3, E_5	isEmptyStack, push, pop
k_5	O_2	E_2, E_6	initQ
k_6	O_1	E_1, E_5	initStack

Tabelle 4.5: Begriffsanalyse - Beispiel: Konzepte

Wie die Tabelle 4.5 zeigt, sind die Konzepte k_1 und k_2 bereits mögliche Kandidaten für eine Modulbildung, da eine maximal große Menge an Objekten bzw. Funktion, die eine funktionale Einheit bilden, zusammengefasst wurden. So repräsentiert zum Beispiel das Konzept k_1 alle Funktionen die den abstrakten Datentyp queue beschreiben.

Das zur Erläuterung der Begriffsanalyse verwendete Beispiel zeigt, dass dieses Verfahren auch eine Möglichkeit zur Identifizierung von Datenstrukturen aus dem Quell-Code beim Design Recovery darstellt.

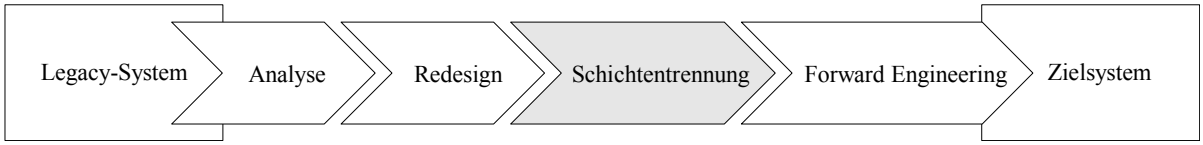
Die Möglichkeit der Automatisierung dieses Verfahrens wird in [SR99] beschrieben.

Die endgültige Entscheidung der Modulbildung trägt der Entwickler im Kontext des Legacy-Systems selbst. Eine grundlegende Basis sind die wiedergewonnenen Erkenntnisse über das Legacy-System und die Erfahrungswerte eines Entwicklers.

4.3.4 Ergebnis-Dokumente

Durch die genannten Techniken ist eine strukturelle Verbesserung des Legacy-Systems möglich. Es werden strukturelle Einheiten gebildet, die als Module das Legacy-System in funktionale Einheiten zerlegen. Zudem werden die Abhängigkeiten zwischen den Einheiten durch die Zerlegung deutlicher und vereinfacht. Die gebildeten Module und die erweiterten Erkenntnisse über das Legacy-System bilden das Ergebnis-Dokument dieser Phase. Die Module sind unter anderem Eingangs-Dokumente der Phase Schichtentrennung und stellen eine gute Basis dar, dass Legacy-System in die logischen Schichten der Drei-Schichten-Architektur zu zerlegen.

4.4 Phase 3 – Schichtentrennung



4.4.1 Ziele der Phase Schichtentrennung

Das Ziel der Phase Schichtentrennung ist die Identifizierung der Zugehörigkeit und Einteilung der aus der Phase Redesign ermittelten funktionalen Einheiten (Module bzw. Gruppen) zu den drei logischen Schichten der Drei-Schichten-Architektur (Präsentations-, Anwendungs-, und Datenhaltungsschicht), sofern eine Zerlegbarkeit durchführbar ist. Zudem sollen, je nach Zerlegbarkeit, Artefakte in die Kategorien Dialogsteuerung, Dienste und Datenzugriffssteuerung unterteilt werden. In der Abbildung 4.9 wird der Fall dargestellt, wenn zur Anwendungsschicht ermittelte Artefakte auch die Kategorien Dialogsteuerung, Dienste und Datenzugriffssteuerung umsetzen.

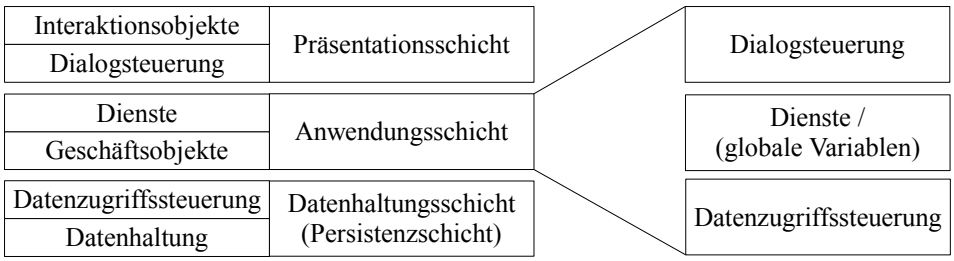


Abbildung 4.9: Drei-Schichten-Architektur und ihre Aufteilung

Mit dieser Schichtentrennung hat man ein zum Zielsystem funktional äquivalentes Ausgangssystem ohne veränderte Funktionalität. Durch die Einteilung der Module zu den logischen Schichten sind diese für eine Schicht identifiziert und können im Zielmodell gezielt umgesetzt oder wiederverwendet werden. Die Entscheidung, welche Artefakte nicht mehr benötigt werden, ist durch die Einteilung ebenfalls möglich.

4.4.2 Eingangs-Dokumente

Als Eingangs-Dokument der Phase Schichtentrennung dient eine sinnvolle Modul- bzw. Gruppeneinteilung aus der Phase Redesign, welche eine strukturierte Ausgangsbasis bildet. Zusätzlich sind die Entwurfsinformationen der Architektur zu verwenden, weil der Kontext der Abhängigkeiten zwischen den Modulen gewahrt sein muss.

4.4.3 Vorgänge und Techniken

Vor der Schichtentrennung ist es notwendig die Zerlegbarkeit der Architektur eines Legacy-Systems zu bewerten und das vorliegende Legacy-System dahingehend einzuteilen. Im Vorfeld kann damit unter anderem beurteilt werden, welche Qualität die Architektur des Legacy-Systems aufweist und inwieweit die Zerlegung möglich und sinnvoll ist. Nach [BS95] gibt es folgende drei Kategorien:

- zerlegbar
- teilweise zerlegbar
- unzerlegbar

Zerlegbar. Ist ein Legacy-System zerlegbar, dann sind Präsentationsschicht, Anwendungsschicht und Datenhaltungsschicht eigenständige Komponenten und bieten nach außen Schnittstellen an. Ein System der Kategorie zerlegbar ist die strukturell am besten geeignetste, weil die drei Schichten bereits existieren.

Teilweise zerlegbar. Ist ein Legacy-System teilweise zerlegbar, dann sind nur bestimmte Schichten eigenständige Komponenten. Die Schwierigkeit dieser Kategorie besteht in der Verwobenheit zwischen verschiedenen Schichten.

Unzerlegbar. Ist ein Legacy-System unzerlegbar, dann ist das Legacy-System eine unstrukturierte monolithische Einheit. Die Schichtentrennung wird dadurch sehr erschwert und ist bei stark verwobenen Strukturen nur dann möglich, wenn eine Zerlegung mit vielen Abhängigkeiten hingenommen wird und die Wiederverwendung keine große Relevanz hat.

Die Bewertung der Zerlegbarkeit wird im Vorfeld schon durch die Ergebnisse der Phase Redesign unterstützt. Zum Beispiel sind flache Baumstrukturen beim Dominanz-Baum ein Indiz für starke Abhängigkeiten, also eine monolithische Einheit – die Zerlegbarkeit nimmt ab. Im Gegensatz dazu sind tiefe Baumstrukturen ein Indiz für eine gute Zerlegbarkeit.

Im Weiteren werden zwei Techniken beschrieben, die eine Schichtentrennung ermöglichen.

Begriffsanalyse

Eine Technik zur Unterstützung der Phase Schichtentrennung wurde mit der Begriffsanalyse bereits in der Phase Redesign (Kapitel 4.3.3) beschrieben. Die Begriffsanalyse als universelle Technik wird in der Phase Schichtentrennung zur Erkennung der Präsentations-, Anwendungs- und Datenhaltungsschicht vorgestellt und kann auch zur Einteilung von Artefakten in die Schichten Dialogsteuerung, Datenzugriffssteuerung und Dienste verwendet werden.

Je nach durchzuführender Schichtentrennung sind Artefakte zu identifizieren, die als Objekte in der Begriffsanalyse zu Eigenschaften zugeordnet werden. Eigenschaften können zum Beispiel die Schichten Dialogsteuerung, Datenzugriffssteuerung und Dienste sein.

Die Begriffsanalyse ermöglicht durch den kreativen Freiraum des Verfahrens, die Zerlegung eines Legacy-Systems mit monolithischen Eigenschaften. Solch eine Zerlegung kann aber zur Folge haben, dass starke Abhängigkeiten zwischen den zerlegten Artefakten eines Legacy-Systems bestehen.

Die Anwendung der Begriffsanalyse in der Phase Schichtentrennung wird am Fallbeispiel im Abschnitt 5.3 demonstriert.

Neben der Begriffsanalyse, welche durch ihre Universalität zur Erkennung für jede Schicht verwendet werden kann, gibt es für die Trennung der Präsentationsschicht eine spezialisierte Technik. Mit dieser können zeichenorientierte Benutzerschnittstellen erkannt und in eine graphische Benutzerschnittstelle transformiert werden. Diese Technik wird im Folgenden kurz vorgestellt, wobei für die Phase Schichtentrennung die Erkennung der zeichenorientierten Benutzerschnittstelle im Vordergrund steht.

Model Oriented Reengineering Process for Human-Computer Interface (MORPH)

MORPH ist ein kompletter Reengineering-Prozess zur automatischen Überführung von zeichenorientierten Benutzerschnittstellen eines Legacy-Systems zu graphischen Benutzerstellen. Der gesamte Prozess wird in Abbildung 4.10 dargestellt.

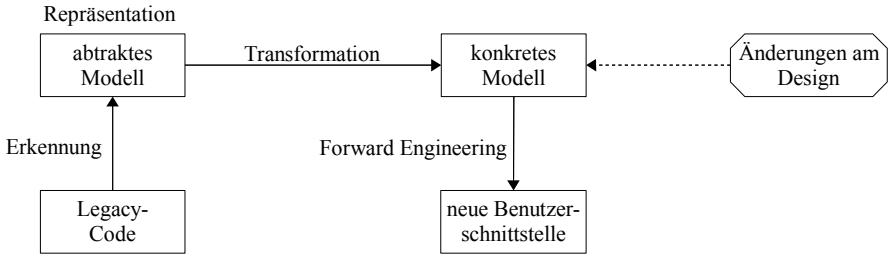


Abbildung 4.10: MORPH – Prozessübersicht [MOO]

- **Erkennung (detection)**

  Dieser Teilschritt dient der Analyse des Quell-Codes, um im Legacy-System die Benutzerschnittstellenbestandteile zu identifizieren. Dazu werden statische Analysen am Quell-Code des Legacy-System durchgeführt. Die statische Analyse beinhaltet den Aufbau von Kontrollflussgraphen, an denen eine für das Problem spezialisierte Datenflussanalyse durchgeführt wird. Die Datenflussanalyse ist auf die Identifizierung von Variablen spezialisiert, die von Ein- und Ausgabekonstrukten einer Programmiersprache verwendet werden. Ein Beispiel für ein Eingabekonstrukt in der Programmiersprache C ist zum Beispiel die Funktion *scanf("%a", &b)* mit den Variablen a und b. In REXX ist ein Ausgabekonstrukt zum Beispiel *say "Dein Name ist:" n*, mit der Variablen n.

  Die in der Analyse identifizierten speziellen Variablen dienen anschließend zur Erkennung von Interaktionsobjekten (z.B.: u.a. Textfeld, Listbox), die sich aus den sogenannten grundlegenden Benutzerinteraktionsaufgaben [FOL95] (Kapitel 8.2) *„basic user interaction tasks“* durch Regeln ableiten lassen.

- *selection*
  Der Benutzer wählt aus einer Menge oder aus einer Liste von Optionen
- *quantify input*
  Der Benutzer gibt numerische Daten ein
- *positioning*
  Der Benutzer definiert eine Bildschirmposition (x,y), um zum Beispiel an einer Stelle gezielt Daten ein- oder auszugeben.
- *text entry*
  Der Benutzer gibt Zeichenketten ein

Die Interaktionsobjekte werden bei MORPH durch Regeln über eine Mustererkennung aus den Analyseergebnissen (aus den erkannten Variablen) erkannt und im Teilschritt Repräsentation in ein abstraktes allgemeines Modell überführt. Die Regeln basieren auf verschiedenen definierten Eigenschaften jeder Interaktionsobjekte. Zum Beispiel wird in [MOO] festgelegt, dass ein *text entry* die Eigenschaften Modifizierbarkeit (z.B.: read-only), Länge (Anzahl der Zeichen) und Möglichkeit der Gültigkeitsprüfung aufweist. Daraus lassen sich zum Beispiel für *text entry* Interaktionsobjekte definieren. Einige beispielhafte Interaktionsobjekte werden in Tabelle 4.6 vorgestellt.

Modifizierbarkeit	**Länge**	**Gültigkeits-prüfung**	**Interaktionsobjekt**
editierbar	kurz (< 80 Zeichen)	ja	Single Line Text Field
editierbar	lang (> 80 Zeichen)	nein	Multiline Text Field
nur lesen	kurz (< 80 Zeichen)	nein	Read-only message

Tabelle 4.6: MORPH - Beispiel: Eigenschaften Interaktionsobjekte

Die Mustererkennungsregeln sind sehr umfassend und können direkt in der Dissertation unter [MOO] nachgeschlagen werden.

- **Repräsentation (representation)**
  In diesem Teilschritt wird aus den Ergebnissen des Teilschritts Erkennung eine abstrakte Darstellung (ein abstraktes Modell) aufgebaut, welches die vorhandene

Benutzerschnittstelle unabhängig von der eigentlichen Implementierung repräsentiert. Unter [MOO] ist das eine festgelegte textbasierte Notation der Eigenschaften jedes Interaktionsobjektes.

- **Transformation**
  In diesem Teilschritt werden die Informationen aus dem abstrakten Modell so verarbeitet, manipuliert bzw. mit zusätzlichen Informationen angereichert, dass bei einem Forward Engineering eine graphische Umgebung durch Modell-zu-Code-Transformation erstellt werden kann. Hier werden die identifizierten Interaktionsobjekte in eine neue graphische Umgebung (z.B.: Java Swing) umgesetzt. So kann ein *text entry* als bestimmtes Textfeld (z.B.: Java Swing: jTextField, jTextArea) oder ein *selection* in ein Menü oder eine ListBox umgesetzt werden.

Die detaillierte Beschreibung von MORPH kann unter [MOO] nachgeschlagen werden.

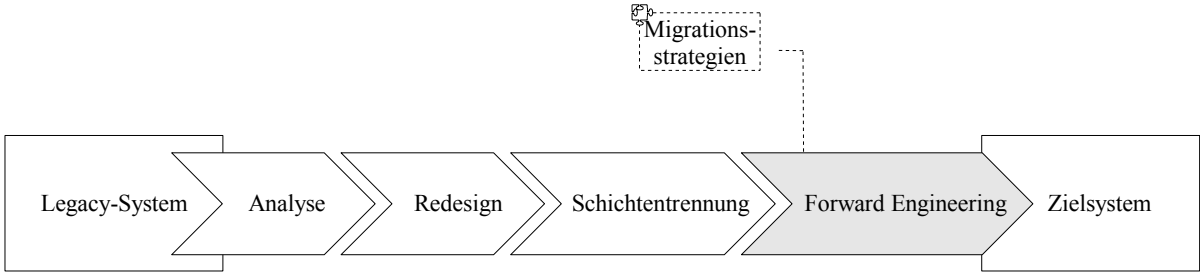
Im Falle der Schichtentrennung ist das Identifizieren der Ein- und Ausgabekonstrukte einer Programmiersprache wichtig und welche Prozeduren bzw. Komponenten die Hauptaufgabe einer Präsentationsschicht übernehmen.

Die erkannten Interaktionsobjekte können in der Phase Forward Engineering zum Aufbau der graphischen Benutzeroberfläche genutzt werden.

4.4.4 Ergebnis-Dokumente

In dieser Phase wurde unter anderem die Zerlegbarkeit des Legacy-Systems bewertet. Mit dieser Aussage kann im Kontext der Module je nach Kategorieeinordnung eine sinnvolle Einteilung der Module zu den logischen Schichten der Drei-Schichten-Architektur durchgeführt werden. Die Einordnung der Module zu den logischen Schichten ermöglicht die gezielte Zuordnung von Funktionalitäten. Diese Erkenntnisse der Schichtentrennung bilden das Ergebnis-Dokument und dienen der leichteren Überführung der Funktionalitäten des Legacy-System in das Zielmodell, welches in der Phase Forward Engineering nun gezielt modelliert werden kann.

4.5 Phase 4 – Forward Engineering



4.5.1 Ziele der Phase Forward Engineering

Das Ziel der Phase Forward Engineering ist die Modifizierung und Neuentwicklung des Legacy-Systems auf Grundlage der entstandenen Ergebnis-Dokumente der vorangegangenen Phasen (Anforderungen und Entwurfsinformationen). Modifizierung heißt unter anderem die verwendeten Technologien auf den Stand der Technik zu bringen. Ein Beispiel für solch eine Modifizierung ist die Anpassung der Anforderungen an das Zielsystem. Weitere im Bereich des Entwurfes sind:

- Paradigmenwechsel
- Programmiersprachenwechsel
- Wechsel der (System-)Architektur
- Benutzerschnittstelle
- Schnittstellen (intern, extern)
- Übergang zu einem neuen Datenverwaltungssystem
- Wechsel von Systemkomponenten

Das Endziel der Phase Forward Engineering ist das zum Legacy-System funktional äquivalente laufende Zielsystem.

4.5.2 Eingangs-Dokumente

Eingangs-Dokumente der Phase Forward Engineering sind zu einem die wiedergewonnenen Anforderungen aus der Phase Analyse und die Entwurfsinformationen des Design Recovery, der Phase Redesign und Schichtentrennung. Im Laufe dieser Phase müssen die Anforderun-

gen und Entwurfsinformation auf ihre Wiederverwendbarkeit und noch sinnvolle Relevanz geprüft werden.

4.5.3 Vorgänge und Techniken

Ergänzende Konzepte

Die Phase Forward Engineering entspricht der klassischen Software-Technik und beinhaltet somit den Software-Entwicklungsprozess. An dieser Stelle müssen spätestens Überlegungen getroffen werden, wie und wann existierende Daten aus dem alten Software-System in das neue modernisierte Software-System übernommen werden, wie und wann das alte durch das modernisierte Software-System ersetzt wird oder zu welchen Teilen. Diese Überlegungen setzten Migrationsstrategien bzw. Migrationskonzepte voraus. Da diese Teilaufgabe sehr umfangreich ist, wird an dieser Stelle auf vorhandene Konzepte unter [EC05] verwiesen.

Zur Steuerung des Entwicklungsprozesses wird die modellgetriebene Software-Entwicklung angewendet, welche im Folgenden kurz beschrieben wird.

Modellgetriebene Software-Entwicklung

Die modellgetriebene Software-Entwicklung (Model Driven Software Development MDSD) steht als Oberbegriff für Techniken, mit denen aus formalen Modellen, die ein Software-System beschreiben, automatisch Quell-Code oder im Idealfall das ganze lauffähige Software-System generiert werden kann. Zudem steht modellgetriebene Software-Entwicklung dafür, Entwicklung und Wartung ausgehend vom Modell durchzuführen. Eine mögliche Strategie der modellgetriebenen Software-Entwicklung stellt die Model Driven Architecture (MDA) [MDA09] dar, die von der Object Management Group (OMG)[28] standardisiert wurde.

Die MDA-Strategie trennt die fachliche und die technologische Sichtweise und unterscheidet diese, wie in Abbildung 4.11 schematisch dargestellt, durch verschiedene Modellebenen. Dazu gehört das Computational Independent Model (CIM), das Platform Independent Model (PIM) und das Platform Specific Model (PSM).

28 Die Object Management Group (OMG) ist ein internationales Konsortium, welches sich mit der Entwicklung von Standards beschäftigt. z.B.: MDA, UML

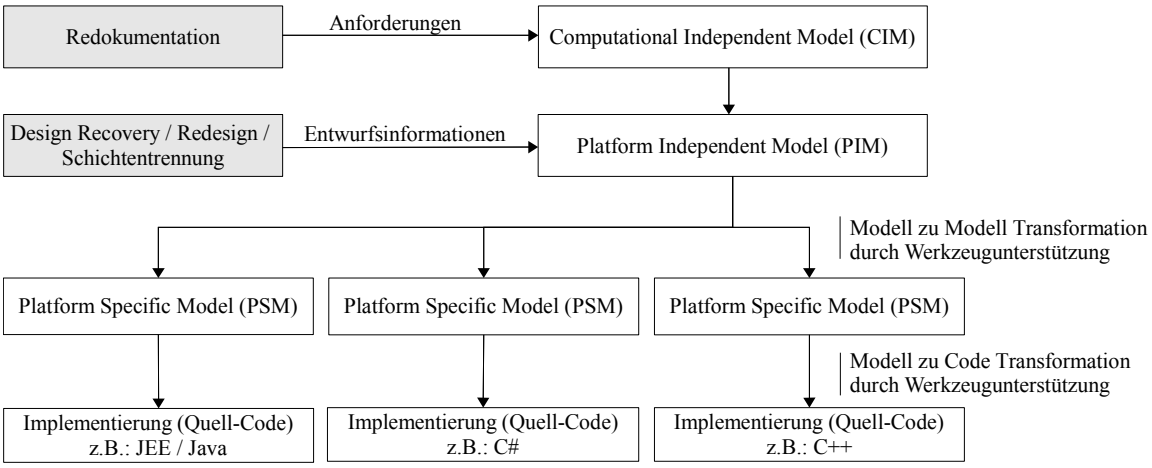


Abbildung 4.11: MDA-Prinzip

Computational Independent Model (CIM). Das CIM ist der Modelltyp mit der höchsten Abstraktionsebene und steht für die umgangssprachliche Beschreibung des zu entwickelnden Software-Systems, unabhängig von technischen Aspekten. Aus dem CIM werden keine weiteren Modelle transformiert, es dient ausschließlich der Dokumentation und als Ausgangsbasis für alle weiteren Modelle. Die Dokumentation wird unter Zuhilfenahme von UML-Diagrammen wie zum Beispiel Anwendungsfall- oder Aktivitätsdiagrammen unterstützt. Im Rahmen des Konzeptes werden im CIM die wiedergewonnenen und neuen Anforderungen des Legacy-Systems für das Zielsystem dokumentiert.

Platform Independent Model (PIM). In einem plattformunabhängigen Modell wird unabhängig von jeder Technologie das fachliche Wissen bzw. die Fachlogik, das Verhalten und der Aufbau des Software-Systems modelliert. Im Rahmen dieses Konzeptes fließen die wiedergewonnenen Entwurfsinformationen des Design Recovery, des Redesign und der Schichtentrennung in das plattformunabhängige Modell ein. Nach der Modellierung kann durch eine werkzeugunterstützende Transformation das plattformunabhängige Modell in ein plattformabhängiges Modell überführt werden.

Platform Specific Model (PSM). Das plattformabhängige Modell erweitert das plattformunabhängige Modell um konkrete technische Details. Solche technischen Details legen zum Beispiel die Programmiersprache und die genauen Datentypen, das Betriebssystem oder die Verwendung eines Anwendungs-Servers fest. Das plattformabhängige Modell beschreibt das Software-System mit den Mitteln der konkreten Implementierungstechnologie und der

Anwendung bestimmter Komponenten. Durch eine werkzeugunterstützende Transformation kann aus dem plattformabhängigen Modell Quell-Code für eine Implementierung generiert werden.

Jeweils für das plattformabhängige und das plattformunabhängige Modell werden zur Modellierung in der Regel UML-Diagramme verwendet. So zum Beispiel kommen Klassen-, Sequenz-, Zustands- oder Komponentendiagramme zum Einsatz.

Umsetzungsalternative. Alternativ zur vorgestellten Vorgehensweise ist in der MDA-Spezifikation vorgesehen, dass auf die Modell-zu-Modell-Transformation vom PIM zum PSM verzichtet werden kann [OMG03] (Kapitel 3.7). Der Verzicht bedeutet, dass aus dem PIM direkt Quell-Code erzeugt wird. Diese Alternative wird auch unter [SB03] (S. 38-45) als populäre Vorgehensweise beschrieben. Hersteller von Software-Modellierungswerkzeugen setzen in der Regel diese Alternative um.

Weiterführende Informationen zur modellgetriebenen Software-Entwicklung können unter [SV07] nachgelesen werden. Die Spezifikation der MDA-Strategie ist unter [MDA09] und [OMG03] verfügbar.

Einteilung der Anforderungen und Entwurfsinformationen in Kategorien

Die wiedergewonnenen Anforderungen (funktionale und nichtfunktionale) und Entwurfsinformationen aus den vorangegangenen Phasen müssen hinsichtlich ihrer Wiederverwendbarkeit überprüft werden. Diesbezüglich wird die in Tabelle 4.7 vorgestellte Kategorieneinteilung vorgenommen:

Kategorie	Farblegende
wiederverwendbar (übernehmbar)	grün
teilweise wiederverwendbar (mit Änderungen)	gelb
nicht wiederverwendbar (nicht übernehmbar)	rot
neu	grau

Tabelle 4.7: Forward Engineering – Kategorieeinteilung für Wiederverwendbarkeit

Im Folgenden wird die allgemeine Kategorieeinteilung beispielhaft für Anforderungen vorgestellt.

Anforderungen

Beispiel Kategorieeinteilung für Anforderungen. Anforderungen können durch Technologiewechsel überflüssig werden oder müssen angepasst werden. Zudem können fachliche Änderungen für eine Anpassung einer Anforderungen verantwortlich sein. Im Kontext des Zielsystems können auch neue Anforderungen hinzukommen, die sich unter anderem durch die gewählte Technologie ergeben.

Die wiedergewonnenen Anforderungen aus der Phase Analyse können im Vorgang der Überprüfung mit der Kategorieeinteilung wir folgt unterteilt werden:

- **Anforderungen sind wiederverwendbar (übernehmbar)**
  Diese Anforderungen sind sowohl fachlich und technologisch im Zielsystem umsetzbar und können direkt vom Legacy-System übernommen werden.

- **Anforderungen sind mit Änderungen wiederverwendbar**
  Diese Anforderungen können im Zielsystem nur umgesetzt werden, wenn diese an technologische oder fachliche Gegebenheiten angepasst werden. So kann für eine zeichenorientierte Benutzerschnittstelle das eindeutige Auswählen einer Option durch Eingeben eines Bezeichners an einer Cursor-Postion gefordert sein, wobei bei einer graphischen Benutzeroberfläche die Auswahl mittels einer Listbox, die alle Optionen beinhaltet, gefordert ist.

- **Anforderungen sind nicht wiederverwendbar (nicht übernehmbar)**
  Die Anforderungen aus dieser Kategorie werden im Zielsystem nicht umgesetzt. Gründe dafür können sein:

  - Eine Anforderung ist technologisch oder fachlich veraltet.

- Eine Anforderung ist mit der neuen Zieltechnologie nicht umsetzbar. So zum Beispiel kann die Steuerung einer Benutzerschnittstelle über Tastenkürzel nicht in einer HTML[29]-Seite umgesetzt werden.

- **neue Anforderungen**
  Diese Kategorie beinhaltet unter anderem Anforderungen, die sich aus den Eigenschaften der Zieltechnologie heraus neu ergeben. So kann bei einer Drei-Schichten-Architektur die Antwortzeit zwischen Aufruf eines Dienstes und Anzeige der gewünschten Information beim Clienten eine neue Anforderung sein. Zudem können im Prozess der Modernisierung neue Anforderungen als Wunsch- oder Musskriterium hinzukommen. Es können auch neue Anforderungen entstehen, wenn ein Legacy-System betriebssystemspezifische Software verwendet und diese im Zielsystem nicht vorhanden ist.

Die Abbildung 4.12 gibt eine übergreifende Übersicht über die Einteilung und Übernahme von Anforderungen zwischen Legacy-System und Zielsystem. Das Legacy-System besteht aus wiederverwendbaren und veralteten Anforderungen. Alle wiederverwendbaren Anforderungen werden übernommen und bilden mit neuen Anforderungen das Zielsystem.

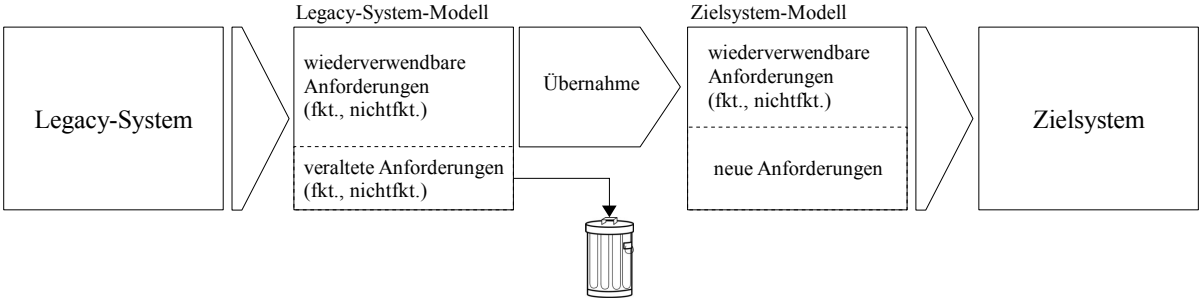


Abbildung 4.12: Forward Engineering – Übersicht über Einteilung der Anforderungen

Systembezogene Anforderungen. Bei systembezogenen Anforderungen handelt es sich unter anderem um die Auswahl der Zieltechnologie und Zielplattform. Da das Konzept eine Drei-Schichten-Architektur und eine objektorientierte Programmiersprache voraussetzt, werden unter anderem folgende Zielplattformen, welche die Zielsystemeigenschaften umsetzen, vorgeschlagen:

29 Die Hypertext Markup Language (HTML) ist eine textbasierte Auszeichnungssprache zur Strukturierung von unter anderem Texten, Bildern und Verweisen.

- Java Platform Enterprise Edition (JEE) [DJ08] (Kapitel 4.2.3)
- .NET[30] (C#) [DJ08] (Kapitel 4.2.4)

Je nach Anforderung kann eine relationale Datenbank frei bestimmt werden.

Anforderungen an ein Software-Modellierungswerkzeug. Vor der eigentlichen Zielsystemmodellierung ist die Entscheidung vorteilhaft, welche Software-Modellierungs- und Entwicklungswerkzeuge eingesetzt werden sollen. Diese Entscheidung kann Auswirkungen auf die Art und Weise der Modellierung und Implementierung haben. Für dieses Konzept ist ein Software-Modellierungswerkzeug zu empfehlen, welches in Zusammenarbeit mit der UML den modellgetriebenen Software-Entwicklungsansatz unterstützt und in der Lage ist, aus den Software-Modellen Quell-Code bzw. Projekte für die Implementierung zu generieren. Dahingehend werden folgende Vorschläge für Software-Modellierungswerkzeuge und Quell-Code-Generatoren unterbreitet, die die Generierung der vorgeschlagenen Zielplattformen unterstützen:

- *objectiF Eclipse Professional Edition* (wurde bereits im Kapitel 4.2.3 vorgestellt und wird im Fallbeispiel verwendet)

- *AndroMDA* [AND09] ist ein open source Werkzeug (Framework), welches Quell-Code aus Modellen generiert. Die Modellierung muss über ein separates Software-Modellierungswerkzeug (z.B.: StarUML[31]) durchgeführt werden, welches den Standard XMI[32] unterstützt. AndroMDA unterstützt unter anderem die Technologien Spring[33], Enterprise Java Beans (EJB), Java Server Pages (JSP), Java Server Faces (JSF) und .NET und kann für jede andere Technologie zur automatischen Generierung angepasst werden.

Nachdem für das Zielsystem die wiedergewonnenen Anforderungen angepasst, übernommen und dokumentiert wurden bzw. neue Anforderungen hinzugekommen sind, dienen diese als Basis für den Entwurf.

30 .NET ist eine von der Firma Microsoft entwickelte Plattform zur Entwicklung von N-Schichten-Architekturen.
31 StarUML ist ein Software-Modellierungswerkzeug und steht unter der GPL.
32 XML Metadata Interchange (XMI) ist ein Austauschformat zwischen Software-Modellierungswerkzeugen und ein Standard der Object Management Group (OMG).
33 Spring ist ein open source Framework zum Erstellen von Java-Anwendungen.

Entwurf

Nach der Konsolidierung der Anforderungen können die Entwurfsinformationen der gleichen Kategorieeinteilung unterzogen werden. So können die wiedergewonnenen Module oder bereits vorhandenen Komponenten nach Wiederverwendbarkeit eingeteilt werden. Wurde ein Modul in die Kategorie wiederverwendbar (grün) eingeteilt, dann kann dieses gleich als solches ins Zielsystem übernommen werden oder es wird zuvor erst durch eine Programmiersprachentransformation in die Zielsprache überführt. Wichtig ist, dass der Entwurf die Zielsystemeigenschaften berücksichtigt und die Anforderungen umsetzt. Es sind weitere folgende Punkte beim Entwurf zu berücksichtigen:

- **Neuentwicklung oder Wiederverwendung (Migration)**
  Im Zuge des Entwurfs und bereits schon vor der Kategorieeinteilung, muss entschieden werden, unter anderem welche Module oder Komponenten des Legacy-Systems funktional äquivalent neu entwickelt, übernommen oder migriert werden sollen. Solche Entscheidungen sind von vielen Faktoren abhängig, zum Beispiel: Aufwand und Kosten, Zieltechnologie, Anforderungen, Transformations- und Migrationfähigkeit der Teile eines Legacy-Systems. Deshalb sollte die Entscheidung zwischen Neuentwicklung oder Wiederverwendung im Rahmen des Konzeptes anhand eines Kriterienkataloges durchgeführt werden. Anhand eines Kriterienkataloges kann die Kategorieneinteilung angewendet werden. Darin sollten unter anderem Software-Qualitätsmerkmale nach [ISO9126] Berücksichtigung finden. Zum Beispiel kann festgelegt sein, welche Versionen von Komponenten (z.B.: Datenbank) für eine Wiederverwendung in Frage kommen. Bei Modulen können für eine Wiederverwendung gute Integrationseigenschaften vorausgesetzt werden (z.B.: Schnittstellen). Mögliche Kriterien und ein Beispielaufbau eines Kriterienkataloges können im Anhang A ab Seite 172 nachgeschlagen werden.

- **Paradigmenwechsel und Programmiersprachenwechsel**
  Beim Entwurf ist zu berücksichtigen, dass das Zielsystem dem objektorientierten Programmierparadigma unterliegt. Das hat zur Folge, dass vom imperativen Programmierparadigma (prozedural) zum objektorientierten Programmierparadigma transformiert werden muss. Im Vordergrund steht die Findung von Objekten. Eine

Basis zur Objektfindung ist das vollständige Verstehen des Legacy-Systems. Im Kontext des Legacy-Systems können die in der Phase Redesign ermittelten Module und die in der Phase Schichtentrennung eingeteilten Module zu den drei logischen Schichten zur Objektfindung herangezogen werden. Wichtig für die Objektfindung sind auch die in der Phase Analyse ermittelten Datenstrukturen. Durch die Einteilung der Module in der Phase Schichtentrennung können mögliche identifizierte Objekte einer Schicht zugeordnet werden.

An dieser Stelle können ergänzend zwei Konzepte herangezogen werden, um aus prozeduralen Systemen Objektkandidaten zu ermitteln:

- Capsule Oriented Reverse Engineering Method (COREM) [GK93]
- Capsule Oriented Reverse Engineering Technique (CORET) [GKM95]

- **Wechsel der (System-)Architektur**
  Die Module, welche in der Phase Schichtentrennung zu den drei logischen Schichten eingeteilt wurden, bieten für die Drei-Schichten-Architektur eine gute Basis zur Planung des Entwurfs. Zum Beispiel können die zur Schicht Dienste/Fachlogik eingeteilten Module zeigen, welche Fachlogiken bzw. Operationen umzusetzen sind. Auch mögliche identifizierte Komponenten (z.B.: eine Datenbank) des Legacy-Systems können hier eine Basis bilden und entsprechend wiederverwendet, migriert oder ersetzt werden.

- **Benutzeroberfläche / Benutzerschnittstelle**
  Der Entwurf der Benutzeroberfläche sollte im Wesentlichen an der ursprünglichen ausgerichtet sein, damit einem Benutzer der gewohnte Umgang erhalten bleibt. Je nach Zieltechnologie und Ausgangstechnologie sind Ausgangsinteraktionsobjekte zu Zielinteraktionsobjekte mit originalem Aufbau umzusetzen. Eine solche Zuordnung von Interaktionsobjekten wurde in der Phase Schichtentrennung mit dem MORPH-Verfahren bereits vorgestellt.

  Zusätzlich ist zu beachten, dass die Zielbenutzerschnittstelle ereignisgesteuert oder formulargesteuert arbeiten kann. Soll zum Beispiel eine Zielbenutzerschnittstelle mit Java Server Faces (JSF)-Technologie umgesetzt werden, dann wird auf HTML-Basis

ereignisgesteuert gearbeitet. In diesem Fall stellt unter anderem ein HTML-Knopf ein Auslöser (Trigger) für Ereignisse dar. Es werden zum Beispiel Dienstoperationen aufgerufen oder andere Aktivitäten. Wenn keine ereignisgesteuerte Verarbeitung bei zeichenorientierten Benutzerschnittstellen gegeben ist, sondern die Benutzerschnittstelle durch viele Eingabe- und Ausgabefunktionen repräsentiert wird, die im sequenziellen Ablauf einer Funktion vorherrschen, dann kann die Benutzerschnittstelle mit MORPH aus dem Quell-Code identifiziert werden. Anschließend müssen aus den Funktionen, die Benutzerschnittstellenanteile besitzen, Aktivitäten abgeleitet werden, die im Zielsystem als Dienstoperationen von Auslösern aufgerufen werden.

Bei graphischen Benutzeroberflächen bietet sich das Beachten von Entwurfsmustern an. Ein Entwurfsmuster ist Model-View-Controller (MVC), welches zwischen Datenmodell, Präsentation und Steuerung separiert. Auf diesem Entwurfsmuster basiert zum Beispiel die JSF-Technologie.

- **Übergang zu einem neuen relationalen Datenverwaltungssystem**
  In aller Regel werden Daten in einer Datenbank persistent gehalten. Von Dateistrukturen zur Haltung von Daten wird abgesehen. Sofern eine Datenbank als Komponente im Legacy-System existiert, müssen die Daten je nach Anforderung in eine relationale Datenbank migriert werden. Diese umfangreiche Teilaufgabe unterliegt einer Migrationsstrategie [EC05]. Auch die Migration von Daten zwischen verschiedenen Datenbankmodellen ist eine umfangreiche Teilaufgabe, die in Kombination mit einer Migrationsstrategie separat betrachtet werden muss und nicht Bestandteil dieser Arbeit ist.

Was wird modelliert und generiert? In Abhängigkeit der vorgeschlagenen Zielplattformen sind folgende (UML-) Modelle zu erstellen, welche zur Generierung benötigt werden:

- **Präsentationsschicht:** Oberflächenseiten(-navigation) und Trigger für Aktivitäten
- **Anwendungsschicht:** Dienste / Fachlogik
- **Persistenzschicht:** Datenmodell (Entities)

Es ist zu beachten, dass jeder Quell-Code-Generator unterschiedliche Restriktionen für den Aufbau eines Modells voraussetzt. Die Art wie das Modell aufgebaut sein muss oder welche Informationen ein Generatorwerkzeug benötigt variiert. Solche Restriktionen sind vor der Modellierung in Erfahrung zu bringen. Zum Beispiel variiert die Angabe von Stereotypen[34] bei UML-Diagrammen. Das Vorgehen zur Erstellung der benötigten Modelle ist für die vorgeschlagenen Zielplattformen im Wesentlichen gleich.

- **Datenmodell**

  Anhand eines Klassendiagramms wird das Datenmodell modelliert. Dabei repräsentiert eine Klasse eine Entität[35]. Bei der Modellierung des Datenmodells werden in den Klassen nur die Attribute, deren Datentypen und die Abhängigkeiten der Entitäten modelliert. Bei der Modellierung in objectiF muss einer Klasse der Stereotyp <<BusinessEntity>> und bei AndroMDA der Stereotyp <<Entity>> zugewiesen werden. Im Fallbeispiel wird gezeigt, wie das Datenmodell mit objectiF modelliert wird.

- **Dienste/Fachlogik**

  In der Anwendungsschicht wird die Fachlogik realisiert. Dazu werden in einem Klassendiagramm Klassen modelliert, welche als Dienste bezeichnet werden. Jede Klasse bietet Dienstoperationen an, welche als Methoden modelliert werden. Die zuvor im Datenmodell modellierten Entitäten können hier verwendet werden. Wird mit objectiF modelliert, dann muss einer Klasse der Stereotyp <<BusinessService>> und bei AndroMDA der Stereotyp <<Service>> zugewiesen werden. Ein Modell mit modellierten Diensten und Dienstoperationen wird im Fallbeispiel mit objectiF gezeigt.

- **Oberflächenseiten(-navigation) und Trigger für Aktivitäten**

  Im Modell für die Präsentationsschicht wird die Oberflächennavigation modelliert. Es können Trigger und die auszuführenden Aktivitäten bzw. Dienstoperationen im Modell angegeben werden. Mit der Oberflächennavigation werden die benötigten Dialoge bzw. Seiten für die Benutzeroberfläche definiert. Der graphische Entwurf der Benutzeroberfläche je Seite kann nicht modelliert werden. Je nach Generatorwerkzeug ist das Modell als Aktivitätsdiagramm (AndroMDA) oder Zustandsdiagramm

34 In der UML ist ein Stereotyp eine Erweiterung von Modellelementen, um Verwendungszusammenhänge anzugeben.

35 Eine Entität ist ein Informationsobjekt - ein eindeutig bestimmtes Objekt, welches Informationen beinhaltet.

(objectiF) zu erstellen. Bei AndroMDA werden Seiten definiert, in dem eine Aktivität als Zustand mit dem Stereotyp <<FrontEndView>> modelliert wird. In objectiF repräsentiert ein Zustand jeweils eine Seite mit dem Stereotyp <<State>> oder <<CompositeState>>. Die Zustände werden über Verbindungen (Transitions) verbunden. Im Fallbeispiel wird die Modellierung mit objectiF und Zustandsdiagrammen gezeigt.

Weitere Modelle müssen bei den vorgeschlagenen Zielplattformen und Generierungswerkzeugen nicht erstellt werden.

Je nach Wahl der Technologie werden nach der Erstellung der Modelle (PIM) pro Modell Projekte mit Quell-Code generiert. Zum Beispiel werden bei objectiF mit JEE-Technologie EJB- und JSP/JSF-Projekte erzeugt. Generiert wird schematisch sich wiederholender Code, welcher sich unter anderem aus Abhängigkeiten der Zielplattform bzw. Programmiersprache ergibt, und Code, welcher sich aus dem PIM umsetzen lässt.

Beim Vorgang der Implementierung ist im Wesentlichen individueller Code zu implementieren, welcher in den generierten Quell-Code-Grundgerüsten aus dem PIM einzufügen ist. Zum Beispiel werden modellierte Methoden im PIM zu Quell-Code-Methoden-Rümpfen umgesetzt, in denen nur noch die Logik implementiert werden muss. Weitere Informationen zur Implementierung bei modellgetriebener Software-Entwicklung können unter [SV07] und [BS09] (S. 14) nachgeschlagen werden.

Im Fallbeispiel wird gezeigt, welche JEE-Projekte erzeugt und wie die Modelle in die JEE-Projekte umgesetzt werden. Zusätzlich wird der Vorgang der Implementierung bei modellgetriebener Software-Entwicklung verdeutlicht.

Wiederverwendung und Entkopplung. Ein Kriterienkatalog, wie bereits vorgestellt, kann über Wiederverwendbarkeit entscheiden. Die wichtigsten Kriterien unter anderem bei Komponenten sind:

- geringe Kopplung
- abgeschlossene und isolierte Einheiten
- wohldefinierte, spezifizierte (standardisierte) Schnittstellen

Entsprechen Komponenten diesen Kriterien oder sind durch Anpassungen diese im Nachhinein erfüllbar, dann können im Zielmodell Schnittstellenklassen als Stellvertreter der Komponente modelliert werden, welche die Kommunikation über die angebotene Schnittstelle der Komponente realisieren (Abbildung 4.13).

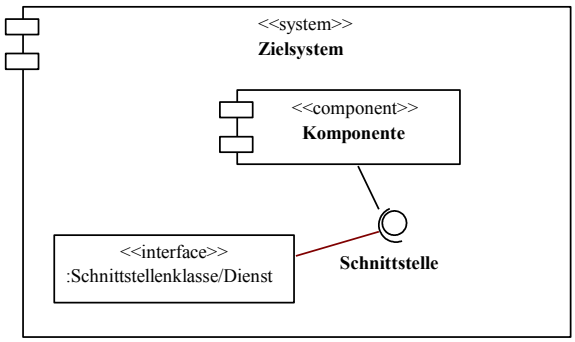


Abbildung 4.13: Wiederverwendung von Komponenten im Zielsystem

Komponenten können auch aus den ermittelten Modulen der Phase Redesign erstellt werden. Die Komponentenentkopplung ist von der Erkennung bestehender Abhängigkeiten geprägt. Dieser Vorgang ist nicht trivial, deshalb wird darauf nicht weiter eingegangen.

Wiederverwendbarer Quell-Code, soweit die Software-Qualität geprüft wurde, kann im Vorgang der Implementierung nach einer Code-zu-Code-Transformation im Zielsystem übernommen werden. Zum Beispiel können Fachlogiken bzw. Algorithmen, welche sich unter anderem in den Prozeduren der zur Schicht Fachlogik/Dienste eingeteilten Module befinden, extrahiert, transformiert und in modellierte Methoden im Zielmodell übernommen werden.

Tests. Das Zielsystem ist während der Phase Forward Engineering durch die in der Software-Technik üblichen Methoden zu testen. Das sind unter anderem:

- Systemkomponententests
- Integrationstests
- System- und Abnahmetest

4.5.4 Ergebnis-Dokumente

Die Phase Forward Engineering als klassischer Prozess der Software-Technik hat als Ergebnis-Dokumente das lauffähige Zielsystem mit entsprechendem Quell-Code, die Entwurfsdokumente mit den erstellten plattformunabhängigen Modellen und die dokumentierten Anforderungen.

4.6 Zusammenfassung

In diesem Kapitel wurde das *4-Phasen-Transformationskonzept* vorgestellt und beschrieben, mit dem ein strukturiert systematisch vorgehendes Konzept zur Modernisierung eines Legacy-Systems vorgestellt wurde. Wichtig für die Anwendung des Konzeptes ist die Beachtung der festgelegten Legacy- und Zielsystemeigenschaften.

Die wichtigste Erkenntnis aus der Beschreibung ist, dass das *4-Phasen-Transformationskonzept* aus den Phasen Analyse, Redesign, Schichtentrennung und Forward Engineering aufgebaut ist, welche beginnend bei der Phase Analyse nach dem Pipeline-Prinzip abgearbeitet werden. Das *4-Phasen-Transformationskonzept* zeichnet aus, dass in jeder Phase externe Konzepte zugelassen werden. Diese externen Konzepte sollen das *4-Phasen-Transformationskonzept* ergänzen und die Bearbeitung umfangreicher oder zusätzlicher Teilaufgaben ermöglichen.

Eine wichtige Erkenntnis aus der Phase Analyse ist, dass sie die Informationswiedergewinnung und den Verständnisaufbau über ein zu modernisierendes Legacy-System realisiert. Zum Verständnisaufbau wird die Top-Down- und Bottom-Up-Analyse in Kombination angewendet. Zur Wiedergewinnung von Entwurfsinformationen wird die Software-Reengineering-Aktivität Design Recovery als Spezialfall des Reverse Engineering angewendet, in der unter anderem statistische Quell-Code-Analysen durch die Anwendung von speziellen Graphen ermöglicht werden. Die statischen Analysen sind auf prozedurale Programmiersprachen ausgerichtet. In einer zweiten Teilphase werden die wiedergewonnen funktionalen und nichtfunktionalen Anforderungen des Legacy-Systems graphisch (UML) und textbasiert redokumentiert.

Die wichtigste Erkenntnis aus der anschließenden Phase Redesign ist, dass eine strukturelle Verbesserung eines Legacy-Systems anhand einer Modularisierung angestrebt wird, um unter anderem die Bestandteile und den Aufbau eines Legacy-Systems sichtbar zu machen. Um dies zu realisieren, wurden die Techniken Dominanz-Analyse und Begriffsanalyse vorgestellt.

Die Phase Schichtentrennung wurde eingeführt, um die Bestandteile des Legacy-Systems in die Schichten der Drei-Schichten-Architektur einzuteilen. Mit dieser Einteilung hat man ein zum Zielsystem funktional äquivalentes Ausgangssystem, welches ein gezieltes Berücksichtigen der Bestandteile des Legacy-Systems im Zielmodell ermöglicht. Zur Realisierung einer Schichtentrennung wurde die Begriffsanalyse vorgeschlagen. Als Spezialfall zur Trennung der Benutzerschnittstelle wurde der Reengineering-Prozess MORPH vorgeschlagen, welcher die Erkennung von Interaktionsobjekten bei zeichenorientierten Benutzerschnittstellen und deren Zuordnung zu Interaktionsobjekten zeitgemäßer Technologien ermöglicht.

Mit der Phase Forward Engineering wurde die letzte Phase des *4-Phasen-Transformationskonzepts* vorgestellt. Eine wichtige Erkenntnis aus dieser Phase ist, dass die MDA-Strategie zur Steuerung des Entwicklungsprozesses angewendet werden kann. Die wiedergewonnen Anforderungen und Entwurfsinformationen aus den vorangegangen Phasen werden auf Wiederverwendbarkeit geprüft und nach einer Kategorieeinteilung für Wiederverwendbarkeit eingeteilt. Um die Zielsystemeigenschaften gezielt umsetzen zu können, wurden die Zielplattformen JEE und .NET vorgeschlagen. Zudem wurden die Software-Modellierungswerkzeuge objectiF und AndroMDA genannt, mit denen die modellgetriebene Software-Entwicklung nach der MDA-Strategie möglich ist.

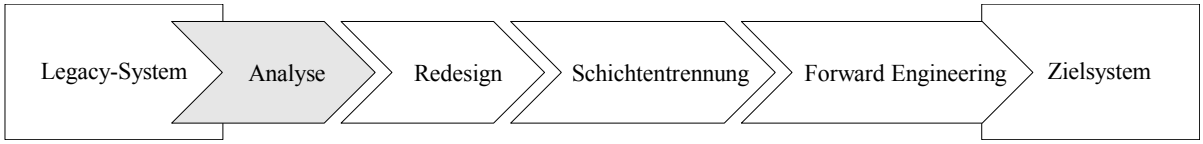
In Abhängigkeit der vorgeschlagenen Zielplattformen sind drei (UML-) Modelle zu erstellen. Für das Datenmodell und die Fachlogik bzw. Dienste ist je ein Klassendiagramm und für die Oberflächennavigation ist je nach Generatorwerkzeug ein Zustands- oder Aktivitätsdiagramm zu erstellen.

Das Ziel des *4-Phasen-Transformationskonzeptes* ist ein technologisch zeitgemäßes Zielsystem, welches das Legacy-System funktional äquivalent umsetzt.

5 Fallbeispiel

In diesem Kapitel wird das *4-Phasen-Transformationskonzept* an einem konkreten Fallbeispiel angewendet. Es wird gezeigt, dass das Konzept in seinem ausgewiesenen Einsatzbereich durchführbar und geeignet ist und das die vorgestellten Techniken bzw. Verfahren nur bei einer (Teil-) Automatisierung den Zeitaufwand der Modernisierung verkürzen können.

5.1 Phase 1 – Analyse



Beschreibung des Ausgangssystems (Legacy-System)

Als Fallbeispiel wird ein Legacy-System namens Objektverwaltung herangezogen, welches 1997 im Rahmen der Jahr-2000-Umstellung von der damaligen csd Computer-Systemdienste GmbH aus Chemnitz entwickelt wurde. Die csd Computer-Systemdienste GmbH ist eine Vorgängerfirma der IT-Services and Solutions GmbH. Die IT-Services and Solutions GmbH firmiert seit dem 1. Juli 2008 als IBM Deutschland.

Die Objektverwaltung diente als Unterstützung der Steuerung von Umstellungsprojekten. Solche Umstellungsprojekte, wie die Jahr-2000-Umstellung und die EURO-Umstellung, erfordern eine Änderung unterschiedlicher Objekte. Objekte können unter anderem Programme, Datenbankdateien, Dateien, Prozeduren oder Funktionen sein. Mit der Objektverwaltung wurde der Umfang aller zu bearbeitenden Objekte, die Zuordnung der Objekte zu Komplexen, der Bearbeitungszustand und welche Mitarbeiter als Bearbeiter auftraten, verwaltet.

Das eigentliche Objektverwaltungsprogramm wurde mit der aus dem Großrechnerbereich stammenden Skriptsprache REXX entwickelt und lief als Anwendung auf einem Großrechner unter dem Betriebssystem OS/390[36], sowie unter dem Nachfolger z/OS[37]. Zur eigentlichen Datenhaltung wurde eine DB2-Datenbank[38] verwendet. Zur Protokollierung von bestimmten Aktivitäten dienten Protokolldateien.

In Abbildung 5.1 wird das Legacy-System schematisch dargestellt.

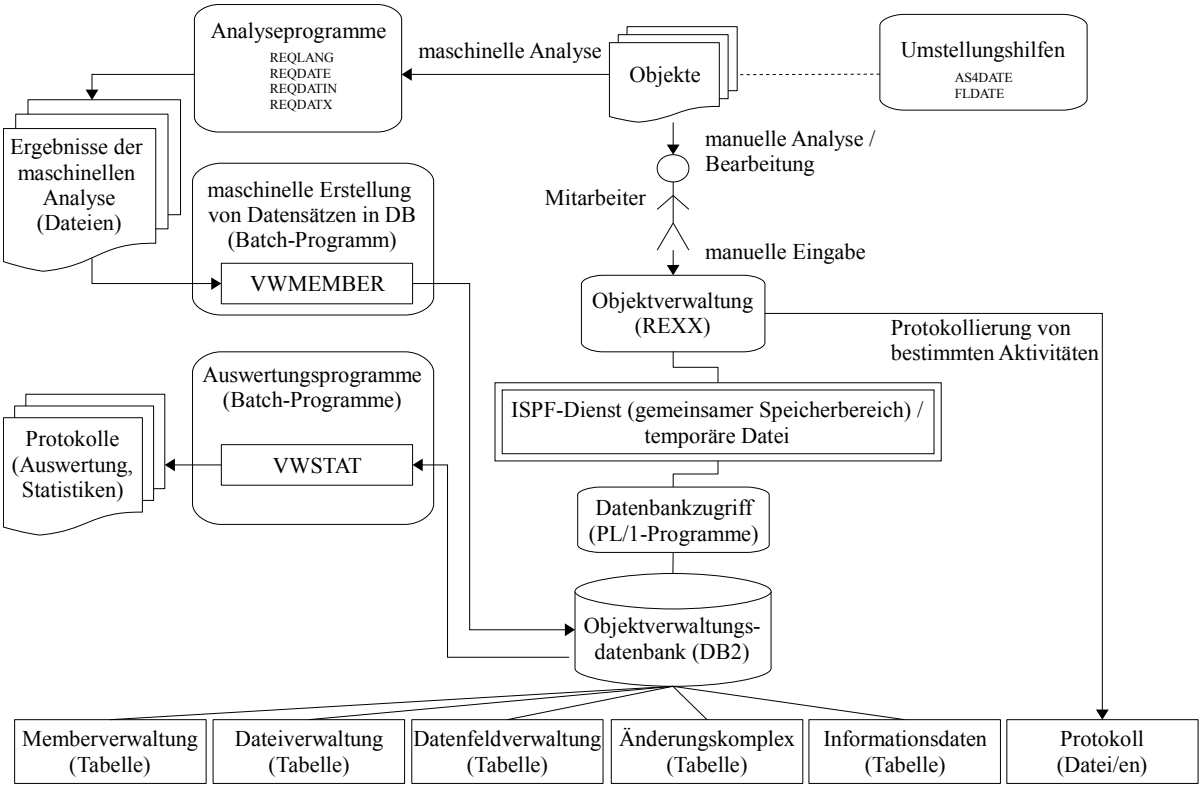


Abbildung 5.1: Ausgangssystem (Legacy-System) - schematische Übersicht

Die für eine Umstellung herangezogenen Objekte wurden in der Regel zuerst durch verschiedene Programme analysiert, wodurch automatisch Dateien mit Analyseergebnissen erstellt wurden. Mit einem separaten Programm (VWMEMBER) konnten die Analyseergebnisse aus den Dateien ausgelesen und automatisch in die DB2-Datenbank übertragen werden. Ein Mitarbeiter konnte Analyseergebnisse oder den Bearbeitungsstatus eines Objektes manuell über die Objektverwaltung eingeben und in der DB2-Datenbank abspeichern. Die Objektverwal-

36 OS/390 ist ein altes IBM-Großrechner-Betriebssystem und eine Vorgängerversion des Betriebssystems z/OS.
37 z/OS ist das aktuelle Betriebssystem für IBM-Großrechner, wobei das *z* im Namen für *zero downtime* steht.
38 DB2 ist ein von der IBM entwickeltes relationales Datenbankverwaltungssystem.

tung war für den Mitarbeiter auch eine Benutzerschnittstelle für die Verwaltung und die Abfrage von Bearbeitungszuständen.

Für die Kommunikation zwischen der Objektverwaltung und der DB2-Datenbank wurden PL/1-Programme verwendet. Der Datenaustausch zwischen der Objektverwaltung und einem PL/1-Programm wurde über einen gemeinsamen Speicherbereich (shared pool) realisiert, welcher durch ISPF bereitgestellt wurde. Teilweise kamen auch temporäre Dateien zum Einsatz, welche den Datenaustausch ermöglichten.

ISPF ist ein Software-Produkt für Terminals im IBM-Großrechnerbereich und ermöglicht für eigene Programme den Aufbau von zeichenorientierten Benutzerschnittstellen. Es besteht unter anderem aus einer Dialogverwaltung und einer Software-Konfigurations- und Bibliotheksverwaltung. ISPF kann mit REXX als auch mit PL/1 verwendet werden. Weiterführende Informationen über ISPF können unter [DJ05] oder [LF05] nachgeschlagen werden.

Zur Erstellung von Statistiken oder Übergabeprotokollen wurde wiederum ein separates PL/1-Programm aufgerufen, welches direkten Zugriff auf die DB2-Datenbank hatte.

Eingangs-Dokumente

Für die Phase Analyse lagen folgende Software-Dokumente als Eingangs-Dokumente vor:

- **Dokumentationen**
  Über das Objektverwaltungssystem lagen ein Entwurfsdokument, ein Präsentationsdokument und eine Kurzbeschreibung des Werkzeugumfanges (Programme) für ein EURO-Umstellungsprojekt vor. [OV00]

- **Quell-Code**
  Zusätzlich zu den Dokumentationen lagen die folgenden Software-Dokumente vor:

  - der REXX-Quell-Code zur Objektverwaltung
  - der Quell-Code der PL/1-Programme für die Kommunikation zwischen der DB2-Datenbank und der Objektverwaltung
  - der Quell-Code des PL/1-Programms zur Erstellung von Statistiken

- SQL-Anweisungsdateien zur Erstellung der Tabellen und Tablespaces[39] für eine DB2-Datenbank
- ISPF-Dateien als Vorlage für die Erstellung der zeichenorientierten Benutzerschnittstelle

Der Quell-Code befindet sich auf der Begleit-CD [CHRCD09].

- **Mitarbeiter mit Wissen über das Legacy-System (Domänenwissen)**
  Neben den Dokumentationen und dem Quell-Code stand ein Entwickler als Wissensträger für Interviews zur Verfügung.

Design Recovery

Im Verlauf der Analyse wird die Top-Down- und Bottom-Up-Strategie iterativ und in Kombination angewendet. Interviews mit dem Entwickler werden nach aufgestelltem Fragekatalog in der Top-Down- und in der Bottom-Up-Analyse durchgeführt. Zum Beispiel sind Fragen über Abläufe oder domänenspezifische Inhalte sinnvoll. Auf konkrete Fragen, welche Inhalt eines Fragekatalogs sein können, wird nicht weiter eingegangen.

Im Top-Down-Vorgang ist die Analyse der als Eingangs-Dokumente vorliegenden Dokumentationen und das Vertrautmachen mit dem Quell-Code des Legacy-Systems gegeben. Dadurch können bereits erste Erkenntnisse gemacht und Anforderungen identifiziert werden, welche in den Teilschritt der Redokumentation einfließen.

Die in der Abbildung 5.2 dargestellte Bildschirmmaske[40] repräsentiert die Einstiegsmaske der Objektverwaltung. Hier können die verschiedenen Verwaltungskomplexe durch das Drücken eines ausgewiesenen Zeichens ausgewählt werden. Unter dem ersten Punkt kann die Member-Verwaltung, unter dem zweiten Punkt kann die Dateiverwaltung, unter dem dritten Punkt kann die Datenfeldverwaltung und unter dem vierten Punkt können Änderungskomplexe angezeigt werden.

39 Ein Tablespace bezeichnet einen Speicherort, in dem alle Informationen (u.a. Tabellen, Indizes) gespeichert werden.

40 Eine Bildschirmmaske bzw. Maske bezeichnet eine Art Vorlage, die den definierten Inhalt einer zeichenorientierten Benutzerschnittstelle enthält.

Die Funktionen Datenfeldverwaltung und Auflösung von Copy-Strecken[41] werden im Fallbeispiel nicht weiter betrachtet.

Bevor ein Verwaltungskomplex (Abbildung 5.2) ausgewählt werden kann, bedarf es für die meisten Punkte der Eingabe eines eindeutigen Bezeichners. Zum Beispiel muss vor der Auswahl von Punkt vier (Änderungskomplex) eine dreistellige Änderungskomplexnummer angegeben werden. Danach kann Punkt vier gewählt werden und der Änderungskomplex mit der Änderungskomplexnummer wird in der Bildschirmmaske Änderungskomplex (Abbildung 5.4) angezeigt.

```
Objektverwaltung                                          tt.mm.jjjj / hh:mm
------------------------------------------------------------------------------

               F u n k t i o n s a u s w a h l

  1 - Programm, Copy-Strecke, Makro:                        Datei:

  2 - Datei/ DB-Name:

  3 - Datenfeldname:
      Copy-Strecke/Makro:

  4 - Änderungskomplex:

  5 - Informations-Datei          KZ-Typ:              KZ:

  6 - Auflösen von Copy-Strecken

  X - Programmende

  Auswahl:
------------------------------------------------------------------------------
```

Abbildung 5.2: Objektverwaltung - Bildschirmeinstiegsmaske zur Funktionsauswahl

In der Abbildung 5.3 wird die Bildschirmmaske der Member-Verwaltung dargestellt. Die Member-Verwaltung verwaltet Member. Ein Member ist im Zusammenhang mit der alten Anforderung der Objektverwaltung ein Makro, eine COBOL-Copy-Strecke oder eine Prozedur einer variablen Programmiersprache. Durch die Eingabe vorgegebener Zeichen können bestimmte Aktivitäten wie Bearbeiten/Speichern (Modify), Löschen (Delete) oder Beenden (Ende) ausgewählt werden.

41 Eine Copy-Strecke beschreibt in COBOL den Aufbau einer Datenstruktur, kann aber auch Quell-Code enthalten.

```
Memberverwaltung                                                    tt.mm.jjjj / hh:mm
------------------------------------------------------------------------------------

Member: xxxxxxxx                  Typ: x (xxxxxxxx)                Sprache: x (xxxx)
Dateiname: xx  (xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx)

Funktionskomplex   : xxxxxxxx                                    Bearbeiter: xxxxxx
Änderungskomplex   : xxx    xxx    xxx    xxx          Fremd-Projekt-Nr. : xxxxxxx
Status Bearbeitung : x  (xxx... ausführlicher Text)        Protokolldruck: x
Überstellungszähler: nn
nicht aktiv : x                    Grund: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Datum:
maschinelle     Schnell-     Fein-       Änderung     Vorübergabe    Übergabe
Analyse         Analyse      Analyse     geprüft
tt.mm.jjjj      tt.mm.jjjj   tt.mm.jjjj  tt.mm.jjjj   tt.mm.jjjj     tt.mm.jjjj

Auswahl: x
------------------------------------------------------------------------------------
M = Modify   D = Delete   X = Ende
```

Abbildung 5.3: Objektverwaltung - Bildschirmmaske Member-Verwaltung

In der Abbildung 5.4 wird die Bildschirmmaske der Anzeige der Änderungskomplexe dargestellt. Verschiedene Dateien und Member können zu einem Änderungskomplex gehören und werden hier angezeigt. Zu jedem Änderungskomplex kann auch eine Beschreibung angegeben werden.

```
Änderungskomplexe                                                   tt.mm.jjjj / hh:mm
------------------------------------------------------------------------------------
Änderungskomplex:  xxx

Beschreibung: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Dateien: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
         xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
         xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Member: xxxxxxxx      xxxxxxxx      xxxxxxxx      xxxxxxxx
        xxxxxxxx      xxxxxxxx      xxxxxxxx      xxxxxxxx
        xxxxxxxx      xxxxxxxx      xxxxxxxx      xxxxxxxx
        xxxxxxxx      xxxxxxxx      xxxxxxxx      xxxxxxxx
        xxxxxxxx      xxxxxxxx      xxxxxxxx      xxxxxxxx
        xxxxxxxx      xxxxxxxx      xxxxxxxx      xxxxxxxx

Auswahl:  x
------------------------------------------------------------------------------------
M = Modify   X = Ende
```

Abbildung 5.4: Objektverwaltung - Bildschirmmaske Änderungskomplexe

In der Abbildung 5.5 wird die Bildschirmmaske Dateiverwaltung dargestellt.

```
Dateiverwaltung                                                    tt.mm.jjjj / hh:mm
------------------------------------------------------------------------------------

Dateiname: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Dateiart : xxxx                      (File / DB2 / IMS / ADAbas)
DSORG    : xxxx            DEVTYPE: xx              Speicherbedarf: ca. xxxxx Cyl.
RECFM    : xxx             LRECL  : xxxxx / xxxxx          BLKSIZE: xxxxx / xxxxx

Funktionskomplex   : xxxxxxxx                                  Bearbeiter: xxxxxx
Änderungskomplex   : xxx    xxx    xxx    xxx
Umstellungsjob     : xxxxxxxx
Status Bearbeitung : x   (xxx... ausführlicher Text)
Überstellungszähler: nn

Datum:
Erfassung            Umstellvorschrift          Prüfung              Übergabe
tt.mm.jjjj           tt.mm.jjjj                 tt.mm.jjjj           tt.mm.jjjj

Auswahl: x
------------------------------------------------------------------------------------
M = Modify   D = Delete   X = Ende
```

Abbildung 5.5: Objektverwaltung - Bildschirmmaske Dateiverwaltung

In der Abbildung 5.6 wird die Bildschirmmaske INFO-Datei dargestellt. Diese Bildschirmmaske kann in der Funktionsauswahl durch Punkt fünf aufgerufen werden.

```
INFO-Datei                                                         tt.mm.jjjj / hh:mm
------------------------------------------------------------------------------------

Kennzeichentyp: xxxx


Kennzeichen: xxxxxxxxxx


Beschreibung: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx


Auswahl:  x
------------------------------------------------------------------------------------
M = Modify   D = Delete   X = Ende
```

Abbildung 5.6: Objektverwaltung - Bildschirmmaske INFO-Datei (-Tabelle)

Die INFO-Datei (Informations- und Kategorieeinträge) beinhaltet Kürzel bzw. Kennzeichentypen, welche zur schnellen Dateneingabe verwendet und auf den Bildschirmmasken zur Platzeinsparung eingesetzt wurden. Kürzel bzw. Kennzeichentypen wurden unter anderem für die Kategorien Dateien, Member-Typen, Dateiarten oder Programmiersprachen verwendet (z.B.: C für COBOL, R für REXX). Über die Bildschirmmaske INFO-Datei konnte die Bedeutung solcher Kennzeichentypen (Kürzel) abfragt und nachgelesen werden. Die Kenn-

zeichen (KZ) waren die ausgeschriebenen Kennzeichentypen (KZ-Typ) (z.B.: Programmiersprache: KZ-Typ: C; KZ: COBOL oder Dateiname: KZ-Typ: 03; KZ: ADA-DB3N).

Alle in dieser Arbeit ausgewiesenen Daten (Tabellen, Datenstrukturen, Konstanten u.a.) sind im Data Dictionary im Anhang B (ab S. 174) definiert und erklärt.

Die zeichenorientierte Benutzerschnittstelle setzt eine ereignisgesteuerte Verarbeitung um, in dem Aktivitäten durch Drücken von Auswahltasten (z.B.: Löschen - Auswahltaste D) ausgelöst werden.

Im Bottom-Up-Vorgang ist die statische Analyse des vorliegenden Quell-Codes durchzuführen. In einem ersten Vorgang wird überprüft, ob mehrere Quell-Code-Dateien zu Gruppen zusammenfaßbar sind, um vielleicht erste Strukturmerkmale zu identifizieren. Durch die Namensgebung der Quell-Code-Dateien war dies sehr einfach möglich. In der Tabelle 5.1 werden die Quell-Code-Dateien aufgelistet und nach Gruppen sortiert. Zusätzlich wird eine Beschreibung zur Bedeutung einer Quell-Code-Datei gegeben. Die Identifizierung und die Beschreibung jeder Quell-Code-Datei konnte aus den Quell-Code-Kommentaren und den vorhandenen Dokumentationen entnommen werden. Des Weiteren wird zur Angabe der Quell-Code-Sprache auch eine Literaturangabe gegeben, um entsprechende Grundlagen nachlesen zu können.

Tabelle 5.1: Quell-Code-Dateien des Objektverwaltungssystems

Quell-Code-Dateien	Beschreibung	Quell-Code-Sprache
OV.REXX	das Objektverwaltungsprogramm	REXX [DJ05] [IBM97]
OV.PLI(DB2IO) OV.PLI(DB2SEQ) OV.PLI(DB2NDVR) OV.PLI(DB2RSEQ) OV.PLI(VWAEND) OV.PLI(VWSTAT)	DB2-Zugriff für REXX über PL/1 DB2-Zugriff für REXX über PL/1 für Datenbankarbeiten für Datenbankarbeiten für Datenbankarbeiten für statistische Auswertung	PL/1 [MW90]
OV.PANELS(OV00) OV.PANELS(OV01) OV.PANELS(OV02) OV.PANELS(OV03)	Kopfinformationen für alle Masken Bildschirmmaske: Funktionsauswahl Bildschirmmaske: Member-Verwaltung Bildschirmmaske: Dateiverwaltung	ISPF-Panel-Definition [DJ05] (Kapitel 12.2) [IBM08]

▼

Quell-Code-Dateien	Beschreibung	Quell-Code-Sprache
OV.PANELS(OV04) OV.PANELS(OV05) OV.PANELS(OV06) OV.PANELS(OV11) OV.PANELS(OV12) OV.PANELS(PGLIST)	Bildschirmmaske: Datenfeldverwaltung Bildschirmmaske: Änderungskomplex Bildschirmmaske: INFO-Datei Bildschirmmaske: Datensatz-Liste Bildschirmmaske: Datei-Liste Bildschirmmaske: Copy-Strecken auflösen	
OV.MSGS(OV00) OV.MSGS(OV01) OV.MSGS(OV02) OV.MSGS(OV03) OV.MSGS(OV04) OV.MSGS(OV05) OV.MSGS(OV06) OV.MSGS(PGL00) OV.MSGS(PGL01)	verschiedene ISPF-Message-Definitionen die von den ISPF-Panel-Definitionen verwendet werden	ISPF-Message-Definition [ZVM08] [IBM08]
OV.DB2(CRECKS) OV.DB2(CRECKT) OV.DB2(CREDFS) OV.DB2(CREDFT) OV.DB2(CREDVS) OV.DB2(CREDVT) OV.DB2(CREIDS) OV.DB2(CREIDT) OV.DB2(CREMVS) OV.DB2(CREMVT)	Tablespace-Erstellung: VERWCKT Tabellenerstellung: VERWCKT Tablespace-Erstellung: VERWDFT Tabellenerstellung: VERWDFT Tablespace-Erstellung: VERWDVT Tabellenerstellung: VERWDVT Tablespace-Erstellung: VERWIDT Tabellenerstellung: VERWIDT Tablespace-Erstellung: VERWMVT Tabellenerstellung: VERWMVT	DB2 SQL-Anweisungen [BG07]

Nach der Gruppierung der Quell-Code-Dateien in Tabelle 5.1 ist zu erkennen, dass die eigentliche Objektverwaltung aus einem einzigen REXX-Programm besteht. Es existieren sechs PL/1-Programme, die für unterschiedliche Datenbankarbeiten erstellt wurden. Darunter zählen die PL/1-Programme, die der Objektverwaltung den Datenbankzugriff ermöglichen. Für jede Bildschirmmaske (auch ISPF-Bildschirmmaske) existiert eine ISPF-Panel-Definitionsdatei, welche die Beschreibung der zeichenorientierten Benutzerschnittstelle enthält. Ergänzend dazu existieren ISPF-Message-Definitionsdateien, in denen Ausgabenachrichten für die Benutzung in ISPF-Panel-Definitionsdateien vordefiniert sind. Für die Erstellung der nötigen Tabellen und Tablespaces existiert jeweils eine SQL-Anweisungsdatei.

Da die Objektverwaltung mit der proprietären Skriptsprache REXX implementiert wurde, ist die Möglichkeit der werkzeugunterstützenden Analyse rar bzw. durch freie Analysewerkzeu-

ge nach jetzigem Stand nicht gegeben. Deshalb werden alle Techniken manuell angewendet und alle zu erstellenden Graphen manuell aufgebaut. Die Graphen werden mit dem Graphenwerkzeug aus Kapitel 1.3 dargestellt.

Zur Einschätzung des Quell-Code-Umfangs der Objektverwaltung, werden für die wichtigsten Quell-Code-Dateien in Tabelle 5.2 verschiedene Software-Metriken ermittelt.

Tabelle 5.2: Objektverwaltung – Software-Metriken

Dateiname	**Software-Metriken**	
OV.REXX	LOC[42]	Quell-Code-Zeilen insgesamt: 3183 davon Quell-Code-Zeilen: 2247 (70,6%) davon Kommentarzeilen: 260 (8,17%) davon Leerzeilen: 676 (21,24%)
	Anzahl der Prozeduren bzw. Funktionen inklusive Hauptprogramm	61
OV.PLI(DB2IO)	LOC	Quell-Code-Zeilen insgesamt: 485 davon Quell-Code-Zeilen: 365 (75,26%) davon Kommentarzeilen: 53 (11%) davon Leerzeilen: 67 (13,9%)
	Anzahl der Prozeduren bzw. Funktionen inklusive Startfunktion	7
OV.PLI(DB2SEQ)	LOC	Quell-Code-Zeilen insgesamt: 336 davon Quell-Code-Zeilen: 226 (67,26%) davon Kommentarzeilen: 49 (14,58%) davon Leerzeilen: 61 (18,15%)
	Anzahl der Prozeduren bzw. Funktionen inklusive Startfunktion	5
OV.PLI(VWSTAT)	LOC	Quell-Code-Zeilen insgesamt: 593 davon Quell-Code-Zeilen: 405 (68,3%) davon Kommentarzeilen: 138 (23,27%) davon Leerzeilen: 50 (8,43%)
	Anzahl der Prozeduren bzw. Funktionen inklusive Startfunktion	8

▼

42 Lines of Code (LOC) – Anzahl der Programmzeilen – ist eine Software-Metrik für die Messung der Größe bzw. des Umfanges.

Dateiname	Software-Metriken	
OV.PANELS(...)	LOC	Quell-Code-Zeilen insgesamt: 1268 davon Quell-Code-Zeilen: 789 (62,22%) davon Kommentarzeilen: 308 (24,29%) davon Leerzeilen: 171 (13,48%)
Gesamtanzahl an Quell-Code-Zeilen: 5865 davon implementierte Quell-Code-Zeilen: 4032 (68,74%), Kommentarzeilen: 808 (13,77%), Leerzeilen: 1025 (17,47%)		

Der Umfang der Objektverwaltungsanwendung (OV.REXX) mit insgesamt 3183 Quell-Code-Zeilen und 61 implementierten Prozeduren wird als mittel eingeschätzt. Im Durchschnitt wird davon ausgegangen, dass der Bereich von 1 - 1000 Quell-Code-Zeilen übersichtlich und verständlich ist. Allerdings ist diese Grenze sehr subjektiv, welche sich durch die Erfahrung eines Entwicklers und die Quell-Code-Qualität verschieben kann.

Da das Objektverwaltungssystem aus Komponenten und mehreren einzelnen Programmen besteht, wird im ersten Schritt der Wiedergewinnung von Entwurfsinformationen ein Aufrufgraph auf der Anwendungs- bzw. Komponentenebene erstellt, um die Aufrufbeziehungen zwischen den einzelnen Programmen und Komponenten zu ermitteln. Dieser Aufrufgraph wird in Abbildung 5.7 dargestellt.

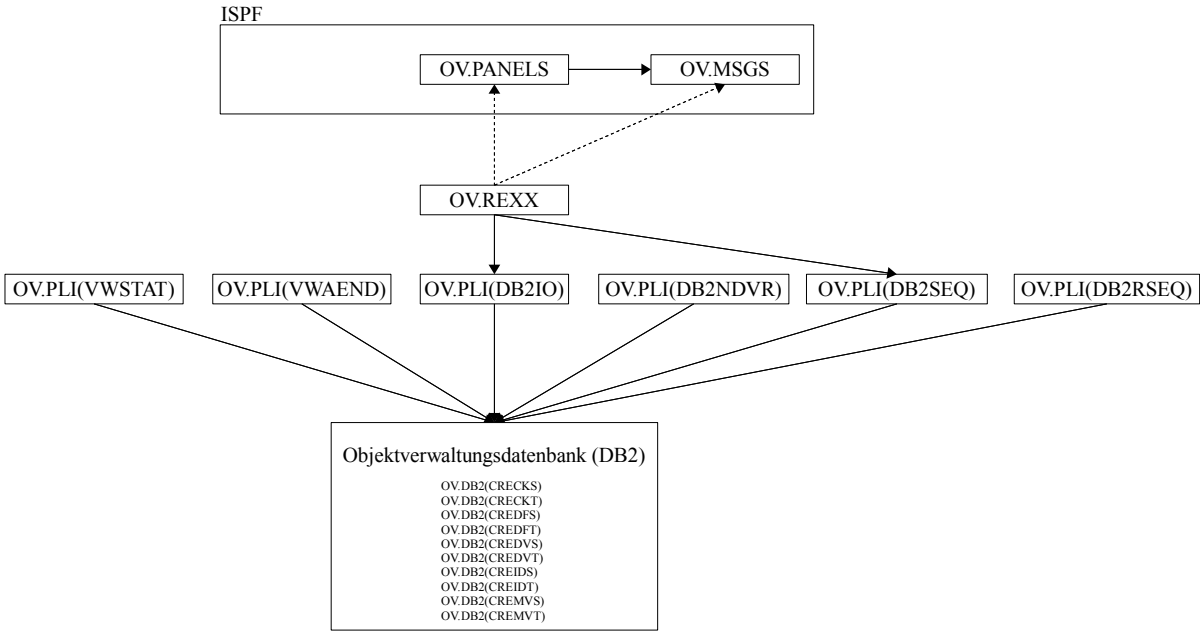


Abbildung 5.7: Aufrufgraph auf Anwendungs- bzw. Komponentenebene

Wie die Abbildung 5.7 zeigt, ruft das Objektverwaltungsprogramm OV.REXX die Programme OV.PLI(DB2IO) und OV.PLI(DB2SEQ) auf. Die Aufrufe wurden im Quell-Code der Datei OV.REXX durch die TSO[43]-Kommandos *RUN PROGRAM("KONST.DB.UPRO")* und *RUN PROGRAM("KONST.DB.UPROSEQ")* identifiziert. Die verwendeten Konstanten *KONST.DB.UPRO* (B.6 S. 177) und *KONST.DB.UPROSEQ* (B.6 S. 177) beinhalten die Namen der aufzurufenden PL/1-Programme. Diese PL/1-Programme ermöglichen dem Objektverwaltungsprogramm die Arbeit mit der DB2-Datenbank (Objektverwaltungsdatenbank).

Die DB2-Datenbank ist der zentrale Datenspeicher und wird von den Programmen OV.PLI(VWSTAT), OV.PLI(VWAEND), OV.PLI(DB2NDVR) und OV.PLI(DB2RSEQ) direkt genutzt. SQL-Datenbankabfragen kann PL/1 durch die Definition einer SQL Communication Area (SQLCA) [AIX2] realisieren. Die SQLCA ist ein Speicherbereich, welcher als Kommunikationsverbindung zwischen Anwendung und Datenbank benutzt wird.

Das Programm OV.PLI(VWSTAT) greift direkt auf die DB2-Datenbank zu, um eine statistische Auswertung zu Erstellen. Die Auswertung wird textbasierend in eine Datei abgespeichert. Die Programme OV.PLI(VWAEND) und OV.PLI(DB2NDVR) wurden speziell für bestimmte Umstellungsprojekte erstellt und werden nicht weiter betrachtet. Das Programm OV.PLI(DB2RSEQ) ist ein Hilfsprogramm zur Beschleunigung der Startphase des Objektverwaltungsprogramms, welches die gesamte Tabelle IDSATZ (B.4 S. 176) in eine Datei lädt, damit OV.REXX direkten Zugriff auf die Kennzeichen und Kennzeichentypen hat.

Ausgehend vom Objektverwaltungsprogramm wurde zusätzlich in Abbildung 5.7 die dynamische Verkettung der ISPF-Panel-Definitions- und ISPF-Message-Dateien als Bibliotheken dargestellt (gestrichelte Pfeile). Informationen über die dynamische Verkettung von ISPF-Bibliotheken können unter [LF05] nachgeschlagen werden. Diese Ergänzung wurde durchgeführt, weil über diesen Aufrufgraph bereits drei logische Schichten erkennbar sind. Die ISPF-Dateien stehen stellvertretend für das ISPF-Software-Produkt und somit für die Präsentationsschicht, das Objektverwaltungsprogramm und die zwei PL/1-Programme für den DB2-Zugriff stehen für die Anwendungsschicht und die Objektverwaltungsdatenbank (DB2) steht für die Datenhaltungsschicht. Alternativ können die zwei PL/1-Programme zu einer extra Datenschnittstellenschicht zugeordnet werden. Der dargestellte Aufruf zwischen

43 Time Sharing Option (TSO) ist ein Kommandozeileninterpreter für IBM-Großrechner-Betriebssysteme.

OV.PANELS und OV.MSGS kommt zustande, weil in OV.PANELS die in den OV.MSGS-Dateien definierten Nachrichten verwendet werden.

Im Weiteren wird der Aufrufgraph zu einem Strukturdiagramm erweitert, um die wichtigsten Datenflüsse zwischen den Programmen zu analysieren. Das Strukturdiagramm wird in Abbildung 5.8 dargestellt.

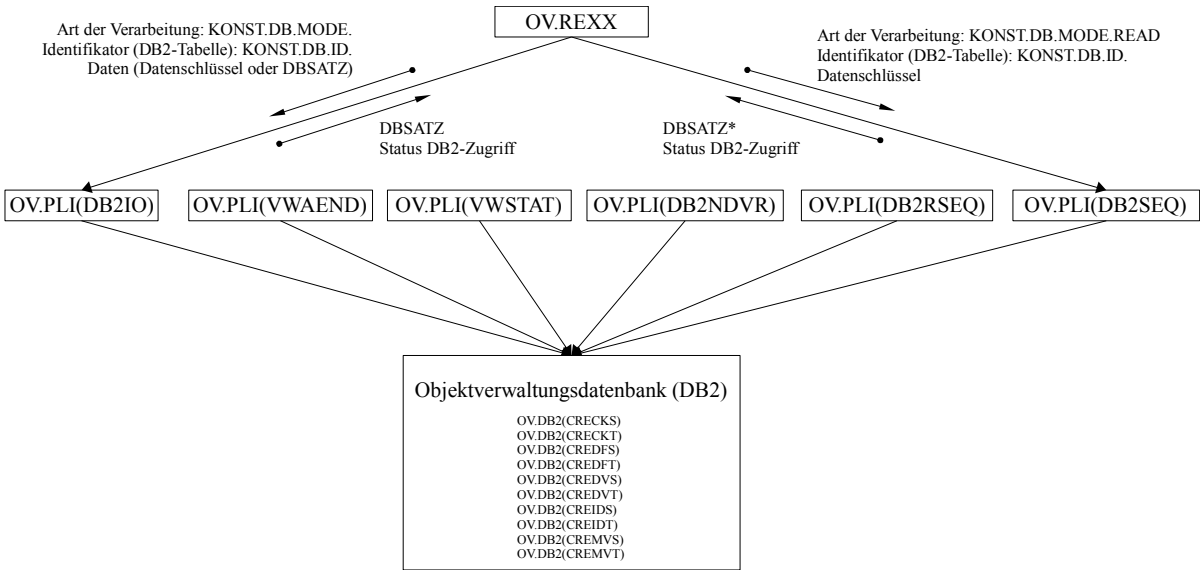


Abbildung 5.8: Strukturdiagramm auf Anwendungs- bzw. Komponentenebene

Wie die Abbildung 5.8 zeigt, kommuniziert das Objektverwaltungsprogramm OV.REXX mit den PL/1-Programmen, um Zugriff auf die DB2-Datenbank zu erhalten. Das Objektverwaltungsprogramm übergibt dem PL/1-Programm OV.PLI(DB2IO) je nach Logik den Verarbeitungsmodus *KONST.DB.MODE.* (neu, lesen, schreiben, löschen, usw.), einen Identifikator zur Wahl der DB2-Tabelle *KONST.DB.ID.* und den eigentlichen Datensatz *DBSATZ* bzw. den Schlüssel zum Datensatz. Vom PL/1-Programm OV.PLI(DB2IO) wird je nach Logik ein gelesener Datensatz oder eine Statusmeldung von der DB2-Datenbank zurückgegeben. Der Datenaustausch zwischen dem PL/1-Programm OV.PLI(DB2IO) und der Objektverwaltung OV.REXX wird durch einen gemeinsamen Speicherbereich realisiert. Wird bei einer Datenbankabfrage mehr als ein Datensatz gefunden, so wird das PL/1-Programm OV.PLI(DB2-SEQ) aufgerufen. Dieses schreibt alle gefundenen Datensätze *DBSATZ* in eine temporäre Datei, die von dem Objektverwaltungsprogramm ausgelesen wird.

Aufgerufen wird das PL/1-Programm OV.PLI(DB2IO) in der Objektverwaltung OV.REXX von der Prozedur *ov_db2pli* und das PL/1-Programm OV.PLI(DB2SEQ) von der Prozedur *ov_db2pli_seq*. Zu beachten ist, dass ein Unterprogramm in REXX sowohl Prozedur- als auch Funktionscharakter hat.

Die SQL-Anweisungen, die von den PL/1-Programmen OV.PLI(DB2IO) und OV.PLI(DB2-SEQ) zur DB2-Datenbank gesendet werden, beschränken sich je nach Gebrauch auf einfache SQL-Konstrukte (SELECT, INSERT INTO, UPDATE, DELETE).

In Abbildung 5.9 wird der Aufrufgraph für das PL/1-Programm OV.PLI(DB2IO) auf Prozedur- bzw. Funktionsebene dargestellt, wo je ein Knoten eine Prozedur ist.

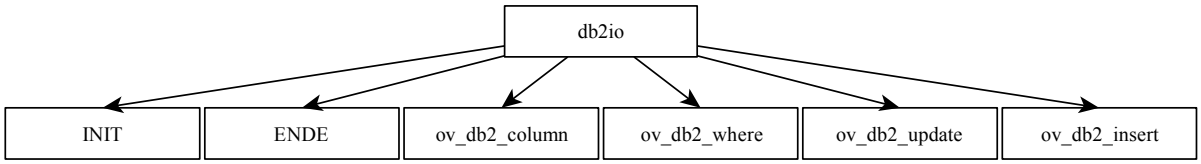


Abbildung 5.9: OV.PLI(DB2IO) – Aufrufgraph auf Prozedur- bzw. Funktionsebene

Die Abbildung 5.9 zeigt den eindeutigen hierarchischen Aufruf ausgehend von der Startprozedur *db2io*. Es werden je nach übergebenen Verarbeitungsmodus *KONST.DB.MODE.* die Prozeduren in der zweiten Ebene aufgerufen. Eine genauere Betrachtung des PL/1-Programms ist nicht notwendig, weil die Funktion als Datenbankzugriffshilfe eindeutig ist.

Der Aufrufgraph für das PL/1-Programm OV.PLI(DB2SEQ) wird in Abbildung 5.10 ausgehend von der Startprozedur *DB2SEQ* dargestellt.

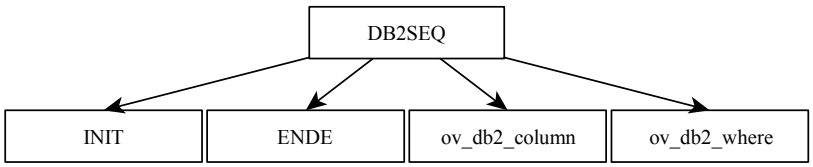


Abbildung 5.10: OV.PLI(DB2SEQ) – Aufrufgraph auf Prozedur- bzw. Funktionsebene

Weitere Betrachtungen sind aus Gründen der bekannten Funktionalität und der trivialen Logik nicht notwendig. Der Datenbankzugriff beschränkt sich auf das Auslesen von Daten.

Im nächsten Schritt wird das Objektverwaltungsprogramm OV.REXX analysiert. Das Objektverwaltungsprogramm enthält die komplette Funktionalität bzw. Anwendungslogik. Zur Wiedergewinnung der Entwurfsinformationen wird für das Objektverwaltungsprogramm ein Aufrufgraph auf Prozedur bzw. Funktionsebene erstellt, wo je ein Knoten eine Prozedur ist. Der Aufrufgraph wird in Abbildung 5.11 (S. 101) dargestellt. Eine Ausnahme stellt der Knoten Hauptprogramm dar, dieser ist keine explizite Prozedur, sondern repräsentiert einen Hilfsknoten für alle Anweisungen die vor der Prozedurendeklaration in REXX kommen. In REXX gibt es keine ausgezeichnete Startprozedur oder Startfunktion.

Dieser komplexe Aufrufgraph vermittelt die Aufrufhierarchie der Prozeduren, trotzt teilweise unübersichtlicher Linienzüge. Das Hauptprogramm ist die zentrale Steuerung und befindet sich auf höchster Ebene des Aufrufgraphen. Es ist sofort erkennbar, dass es keine alleinstehenden Knoten gibt, die weder eingehende noch ausgehende Kanten haben. Das bedeutet, dass es im Objektverwaltungsprogramm keine nicht erreichbaren Prozeduren gibt, die nicht erreichbaren Quell-Code (toter Code) darstellen.

Um die Übersicht und die Aussagekraft des Aufrufgraphen zu verbessern, wurden allgemeingültige und weniger wichtige Prozeduren und deren eingehenden und ausgehenden Kanten entfernt. Dieser reduzierte Aufrufgraph wird in Abbildung 5.12 (S. 102) dargestellt.

Ausgeblendet wurde die Prozedur *ov_message*, weil diese von einem überwiegenden Teil der Prozeduren verwendet wird und rein für die Anzeige von Nachrichten zur Verfügung steht. Es wurden auch die Zeit- und Datumsprozeduren ausgeblendet, weil diese allgemeingültige Dienste darstellen und nicht zur Kernlogik des Objektverwaltungsprogramms dazuzählen. Stark abhängig von den Zeit- und Datumsprozeduren ist die Prozedur *ov_protokoll*. Diese wurde im Zuge der Abhängigkeit vorläufig mit ausgeblendet. Die Prozeduren *ov_db2pli_delete*, *ov_db2pli_read*, *ov_db2pli_write* und *ov_db2pli* gehören zur Kernfunktionalität und wurden nur ausgeblendet, damit die Aufrufhierarchie noch sichtbarer wird. Die Prozedur *ov_lokation* stellt keine Kernfunktion zur Verfügung und kann deshalb ausgeblendet werden.

Nach der Reduzierung ist die Aufrufhierarchie in Abbildung 5.12 sehr deutlich erkennbar. Das Hauptprogramm ruft je nach Wahl des gewünschten Verwaltungskomplexes eine Panel-Prozedur zur Steuerung einer zeichenorientierten Bildschirmmaske auf. Vor allen anderen Aktivitäten wird die Initialisierungsprozedur *ov_init* und vor Beendigung des Objektverwaltungsprogramms wird die Exit-Prozedur *ov_exit* aufgerufen. Ausgehend von den Panel-Prozeduren werden in der nächsten Hierarchieebene zur Funktionalität zugehörige Prozeduren aufgerufen.

Weitere Erkenntnisse aus diesen Aufrufgraphen sind, dass nur Prozeduren höherer Ebene Prozeduren tieferer Ebene aufrufen. Es gibt keine Aufrufe ausgehend von Prozeduren auf niedrigerer Ebene zu Prozeduren auf höherer Ebene. Diese Erkenntnis zeugt davon, dass bei der Entwicklung und Wartung auf Struktur geachtet wurde. Zur Bekräftigung der eindeutigen Hierarchie kommt hinzu, dass es keine Zyklen innerhalb der Aufrufgraphen gibt. Die Prozeduren können sich nicht so gegenseitig aufrufen, dass Aufrufschleifen entstehen. Die Erkennung von Zyklen kann zum Beispiel werkzeuggestützt über das Graphenprogramm yEd Graph Editor ermöglicht werden.

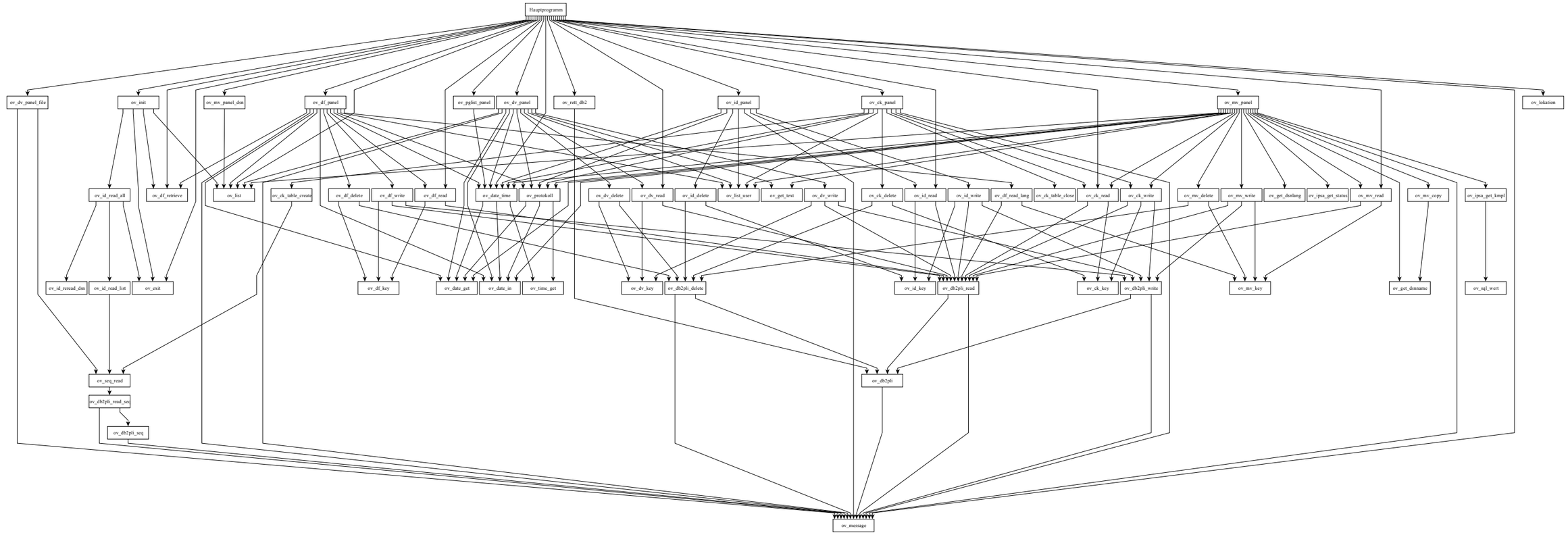


Abbildung 5.11: OV.REXX – Aufrufgraph auf Prozedur- bzw. Funktionsebene

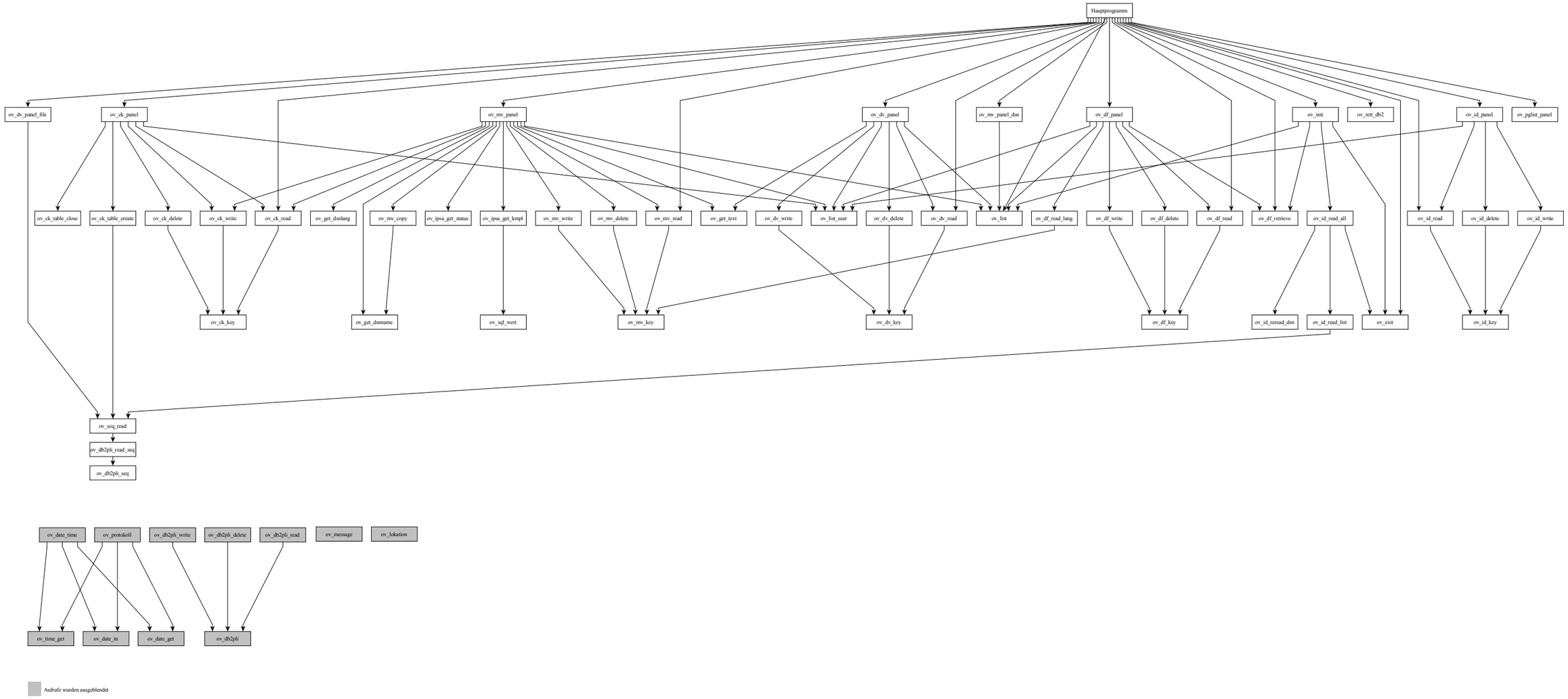


Abbildung 5.12: OV.REXX – reduzierter Aufrufgraph

Datenstrukturen. Nach der Wiedergewinnung der Aufrufhierarchie werden die Datenstrukturen ermittelt, welche die Objektverwaltung verwendet. Dazu gehören die Tabellen der DB2-Datenbank. Die Informationen über den Aufbau der Tabellen konnten aus einem Entwurfsdokument entnommen werden. Zusätzlich liefern die SQL-Anweisungsdateien die eindeutige Definition der Tabellen. Ein explizites Reverse Engineering, zur Wiedergewinnung der Datendefinitionen aus der DB2-Datenbank, ist nicht notwendig.

In der Tabelle 5.3 werden die Datenstrukturen der Persistenzschicht[44] aufgelistet. Dazu gehören die Tabellen der DB2-Datenbank und die Datenstruktur der Protokolldatei.

Beschreibung	**Name**	**Typ**	**Datenstruktur**
Member-Verwaltung	VERWMVT	DB2	MVSATZ
Dateiverwaltung	VERWDVT	DB2	DVSATZ
Änderungskomplexe	VERWCKT	DB2	CKSATZ
Informationsdaten	VERWIDT	DB2	IDSATZ
Protokolldatei	<BEZ>.VERW.PROTOK	Datei	PRSATZ

Tabelle 5.3: Objektverwaltung – Tabellen / Protokolldatei

Die Zugehörigkeit der Tabellen zu einem Verwaltungskomplex werden in der Tabelle 5.3 durch die Beschreibung verdeutlicht. Zum Beispiel enthält die Tabelle Informationsdaten (VERWIDT) alle Daten des Verwaltungskomplexes INFO-Datei.

Der Aufbau der Tabellen und die möglichen Beziehungen zwischen diesen, sowie die Datentypen der Tabellenspalten und die Primärschlüssel der einzelnen Tabellen, werden durch ein ER-Modell in der Abbildung 5.13 dargestellt. Das ER-Modell wurde aus den Definitionen der SQL-Dateien erstellt. Sehr auffällig ist die durchgängige Wahl zeichenkettenorientierter Datentypen (CHAR, VARCHAR). Das hat den Grund, weil Zeichenketten als Datentypen die einfache unkomplizierte Anzeige der Daten in einer zeichenorientierten Benutzerschnittstelle ermöglichen, ohne das Datentypen konvertiert werden müssen. Ein weiterer Grund ist, weil es in REXX nur den Datentyp Zeichenkette oder numerische Inhalte gibt. Datentypen werden nicht explizit definiert und eine Unterscheidung ist nicht sichtbar.

44 In der Informatik bezeichnet der Begriff Persistenz die Fähigkeit zur Speicherung von Daten in nichtflüchtigen Speichermedien (z.B.: Datenbanken). Die Persistenzschicht entspricht der Datenhaltungsschicht.

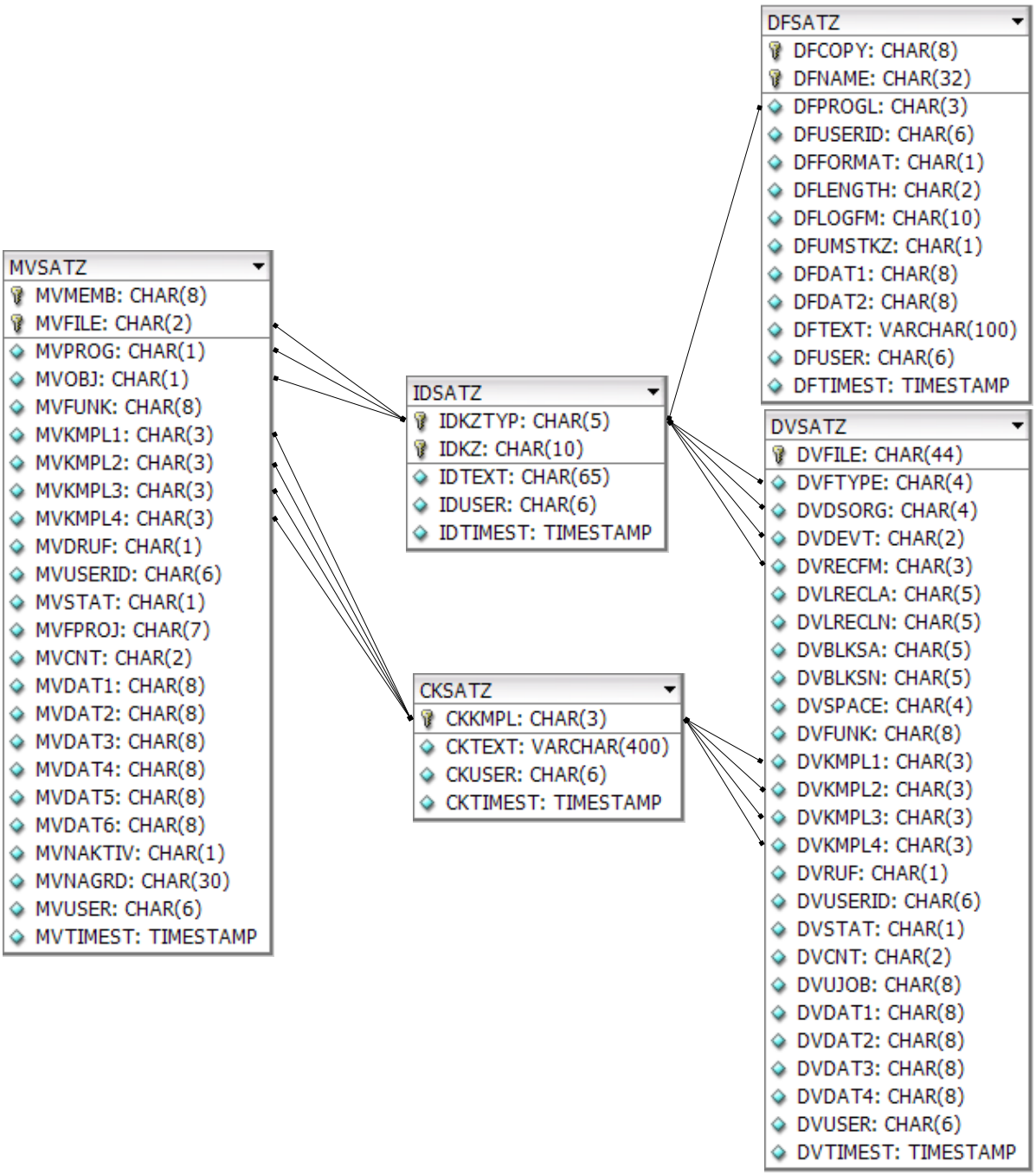


Abbildung 5.13: ER-Modell der OV-Datenbank mit möglichen Beziehungen

Fremdschlüsselbeziehungen zwischen den Tabellen sind nicht vorhanden. Eine Verknüpfung mit einer anderen Tabelle wird durch einfaches Speichern des Inhaltes eines Primärschlüssels realisiert. Zum Beispiel werden die Änderungskomplexnummern, die als Primärschlüssel (CKKMPL) in der Tabelle CKSATZ definiert sind, als einfache Daten (MVKMPLn) in der Tabelle MVSATZ eindeutig gespeichert.

Die versteckten Beziehungen zwischen den Tabellen werden im ER-Modell (Abbildung 5.13) frei dargestellt und konnten durch die im Entwurfsdokument enthaltenen Beschreibungen der Tabellen identifiziert werden.

Zur weiteren Untersuchung und Prüfung auf Datenstrukturen bzw. Daten die verwendet werden, werden die Datenflüsse zwischen den Prozeduren der Objektverwaltung untersucht. Dazu wird der bereits erstellte Aufrufgraph der Objektverwaltung OV.REXX zu einem Strukturdiagramm erweitert. Dieses Strukturdiagramm wird in Abbildung 5.14 auf Seite 107 dargestellt. Es ist zu beachten, dass wegen der Größe des Diagramms die hierarchische Anordnung von links nach rechts verläuft und die eingehenden und ausgehenden Datenflüsse durch die Kantenbeschriftung *in:* und *out:* dargestellt werden. Des Weiteren wurden die Prozeduren *ov_lokation* und *ov_message* ausgeblendet. Diese sind für die Untersuchung nicht relevant und verbessern durch ihre Ausblendung die Übersichtlichkeit des Diagramms.

Die erste Erkenntnis aus dem Strukturdiagramm ist, dass zum größten Teil Konstanten für Statusmeldungen und Parameter verwendet wurden. Die Menge an globalen Variablen, die zum Datenaustausch als explizite Übergabeparameter verwendet werden, ist gering. Zwischen den Prozeduren werden hauptsächlich Statusparameter übergeben und Statusmeldungen zurückgegeben, aber keine fachlichen Daten.

Der Austausch von fachlichen Daten wird mit globalen Variablen realisiert, welche aber nicht als Übergabeparameter identifizierbar sind. Zu beachten ist, dass in REXX eine Variable nicht explizit deklariert wird. In REXX können überall im Quell-Code Bezeichner als Variable verwendet werden, welche gleichzeitig global sind. Ein Anhaltspunkt für versteckte Variablen kann auch ein spezieller Sprachkonstrukt von REXX sein. Denn wenn eine Prozedur eine Variable aus einer anderen Prozedur verwenden will, kann das in REXX durch den Sprachkonstrukt *EXPOSE <Variablenname>* ermöglicht werden. Dieser Sprachkonstrukt muss direkt nach der Definition einer REXX-Prozedur stehen (z.B.: ov_db2pli_write: PROCEDURE EXPOSE dbsatz).

Die globale Variable *DBSATZ* ist vom Typ Zeichenkette und beinhaltet fachliche Daten. Diese wird von der Prozedur *ov_db2pli* mit den je nach Anforderung gewünschten Daten aus einer Tabelle (MVSATZ, IDSATZ, DFSATZ, CKSATZ, DVSATZ) gefüllt und der aufrufenden

Prozedur *ov_[dv|mv|id|df|ck]_read* zurückgegeben. Durch die Definition von Konstanten, die fest definierte Feldlängen der Tabellenfelder enthalten, können die Daten aus der Zeichenkette extrahiert werden.

Die globalen Variablen MVSATZ, IDSATZ, DFSATZ, CKSATZ und DVSATZ werden von den Prozeduren *ov_[mv|id|df|ck|dv]_write* als Übergabeparameter verwendet, um Daten über die Prozedur *ov_db2pli_write* in der DB2-Datenbank abzuspeichern.

Der Datenaustausch zwischen zeichenorientierter Benutzerschnittstelle und den Prozeduren der Objektverwaltung geschieht ebenfalls direkt im Quell-Code. In den ISPF-Panel-Definitionsdateien werden Variablen definiert, die in REXX nach dem Aufruf einer ISPF-Bildschirmmaske als globale Variablen zur Verfügung stehen. Diese Variablen sind im Quell-Code explizit durch den Prefix *p* und anschließendem Kürzel für den Verwaltungskomplex (z.B.: pmv... Member-Verwaltung oder pck... Änderungskomplex) identifizierbar und werden im Quell-Code von OV.REXX direkt verwendet. Anhand dieser Variablen können die Daten auf einer ISPF-Bildschirmmaske angezeigt und eingegebene Daten geholt werden. Die ISPF-Panel-Variablen werden im Quell-Code der Objektverwaltung an die globalen Variablen *ov.* übergeben. Der Aufbau der Variablen *ov.* wird im Data Dictionary B.6 auf Seite 177 beschrieben.

In der Abbildung 5.14 wurden die Beziehungen zwischen den Prozeduren und den globalen Variablen *ov.* und *p.* ausgeblendet. Die Ausblendung ermöglicht eine übersichtlichere Darstellung.

Zu beachten ist, dass Datenstrukturen bzw. Felder in REXX durch einen Punkt zwischen den Variablennamen definiert werden (z.B.: ov.mv.date = [ov.mv.date.1, ov.mv.date.2, ov.mv.date.i] - Abbildung 5.15). Ausführliche Informationen über die Definition von Variablen in REXX können unter [DJ05] nachgeschlagen werden.

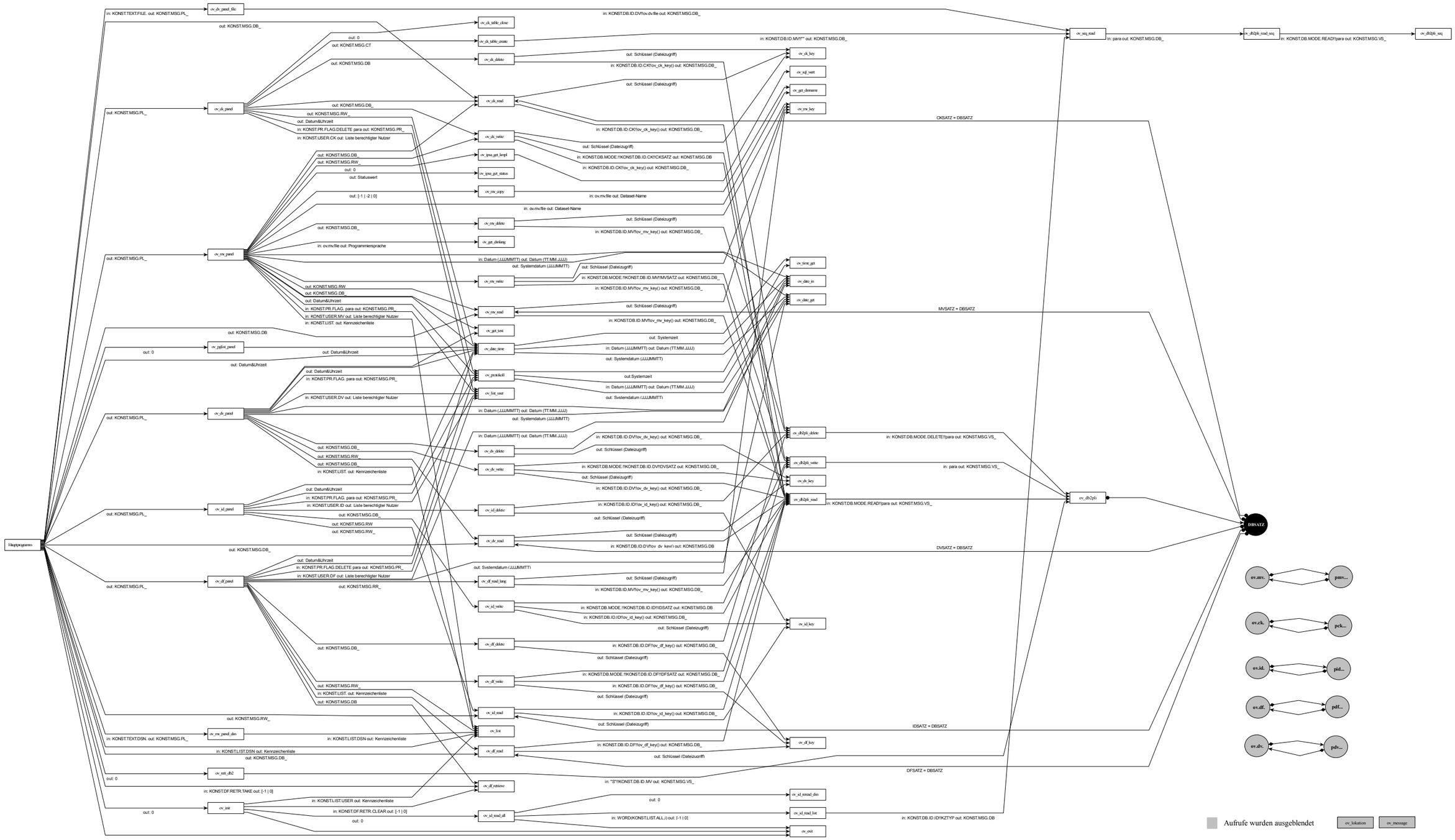


Abbildung 5.14: OV.REXX – Strukturdiagramm

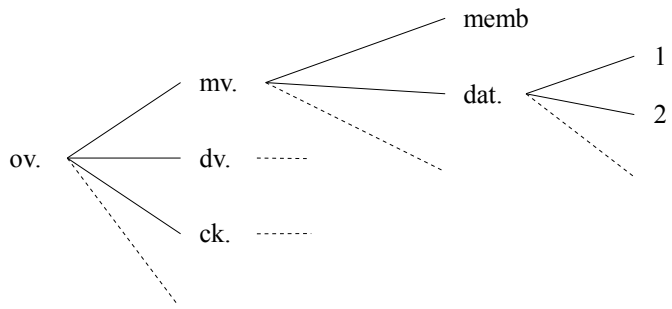


Abbildung 5.15: Beispiel: Datenstrukturen in REXX

Algorithmen und Fachlogik. Im weiteren Vorgehen sollen wichtige fachliche Abläufe aus dem Quell-Code ermittelt werden. Im Rahmen des Fallbeispiels wird dies beispielhaft durch Programmablaufpläne an den Verwaltungskomplexen Member-Verwaltung und Änderungskomplex durchgeführt. Diese Programmablaufpläne können im Anhang C ab Seite 181 eingesehen werden.

Ein Beispiel für eine aus den Programmablaufplänen erkannte Fachlogik, ist die in der Member-Verwaltung automatische Setzung von Datumswerten in Abhängigkeit eines Bearbeitungsstatus. In der Tabelle 5.4 wird der zugehörige Quell-Code aus der Prozedur *ov_mv_panel()* bei Funktionswahl M (Modify) gezeigt.

```
...   /* ov_mv_panel() */
2728  i=POS( ov.mv.stat, "N1M2S3F46U");
2729  IF (i > 3) THEN DO
2730        i=SUBSTR("1112233456",i,1);
2731        IF (ov.mv.dat.i = "") THEN ov.mv.dat.i=ov_date_get();
2732  END  /* if */
...   ...
2734  IF ((i == 3) & (ov.mv.stat ^= pmvstatret)) THEN
2735        ov.mv.dat.i=ov_date_get();
2736  IF i < 4 THEN DO
2737        DO i=i TO 2 BY -1
2738          IF (ov.mv.dat.i = "") THEN ov.mv.dat.i=ov_date_get();
2739        END /*do-to*/
2740  END
...   ...
```

Tabelle 5.4: Fachlogik (Datumssetzung in Abhängigkeit eines Status): OV.REXX

In der Tabelle 5.5 wird die Zuordnung von Datumswert und Status nach Code-Abschnitt Tabelle 5.4 Zeile 2728-2732 dargestellt. Zum Beispiel wird das Datum *Vorübergabe* nur bei Status *6* (Vorübergabe) gesetzt. Die zugehörige globale Variable ist *ov.mv.dat.5*. Im Code-Abschnitt Tabelle 5.4 Zeile 2734-2740 werden die Datumswerte der maschinellen Analyse separat gesetzt. Die Beschreibung aller Statuswerte kann unter [OV00] nachgeschlagen werden.

Position *i* = POS...	1	2	3	4	5	6	7	8	9	10
ov.mv.dat.*i*	1	1	1	2	2	3	3	4	5	6
Datum	maschinelle Analyse			Schnellanalyse		Feinanalyse		IPSA-Bearbeitung	Vorübergabe	Übergabe
Bearbeitungsstatus	N	1	M	2	S	3	F	4	6	U

Tabelle 5.5: Fachlogik: Zuordnung Datumswert zu Status (ov_mv_panel())

Es wurde festgestellt, dass sich in den ISPF-Panel-Definitionsdateien ebenfalls Logikbestandteile befinden, welche unter anderem Gültigkeitsprüfungen der Eingabedaten realisieren. Es lassen sich auch Prüfungen im Bereich der Benutzerrechte erkennen. Je nach Berechtigung eines Benutzers werden Flag-Variablen gesetzt, welche die Funktionsauswahl beeinträchtigen. Zum Beispiel wird die Auswahl M (Modify) nur angezeigt, wenn ein berechtigter Benutzer vorhanden ist. Berechtigt heißt, dass ein Bearbeiter (ein Benutzer) nur Speichern oder Löschen darf, wenn dieser der aktuelle Bearbeiter unter anderem eines Members ist. Nur mit der expliziten Anweisung eines Bearbeiters, kann unter anderem ein Member an einen anderen Benutzer zur Bearbeitung übergeben werden. Eine Umsetzung einer Benutzerverwaltung ist im Legacy-System nicht erkennbar. Somit setzt das Legacy-System ein sehr einfaches Mehrbenutzersystem um, wobei die Benutzerverwaltung sowie die Zugriffsrechte vom Betriebssystem verwendet werden.

Redokumentation

Im Weiteren werden die wiedergewonnenen Anforderungen und Aktivitäten stellvertretend für das Computational Independent Model der modellgetriebenen Software-Entwicklung vorgestellt. Die Redokumentation wird mit dem Werkzeug *objectiF* durchgeführt. Auf der Begleit-CD [CHRCD09] im Verzeichnis [Quellcode]/[ObjectIF_PROJEKT] befindet sich

das objectiF-Projekt, welches die folgenden Anwendungsfälle und Aktivitätsdiagramme beinhaltet.

Funktionale Anforderungen. In der Abbildung 5.16 werden die funktionalen Anforderungen des Objektverwaltungssystems dargestellt, welche im Prozess der Top-Down- und Bottom-Up-Analyse identifiziert wurden.

Abbildung 5.16: Anforderungen des Fallbeispiels – Kontextdiagramm

Wie die Abbildung 5.16 zeigt, gibt es die Akteure Mitarbeiter und Bearbeiter. Ein Mitarbeiter wechselt in die Rolle eines Bearbeiters, wenn dieser für die Bearbeitung unter anderem eines Members eingetragen ist. Zum Beispiel kann nur ein eingetragener Mitarbeiter als Bearbeiter einen Member bearbeiten, speichern oder löschen. Ein Bearbeiter kann die Verantwortlichkeit explizit an einen anderen Mitarbeiter abgeben.

Folgende Anwendungsfälle wurden verfeinert und können im Anhang D.1 ab Seite 185 nachgeschlagen werden:

- Member verwalten
- Dateien verwalten
- Änderungskomplexe verwalten
- Informations- und Kategorieeinträge verwalten

Die Beschreibungen der Anwendungsfälle können im Anhang D.3 ab Seite 189 und notwendige Begriffe können im Glossar Anhang D.4 auf Seite 199 nachgelesen werden.

Nichtfunktionale Anforderungen wurden nicht betrachtet.

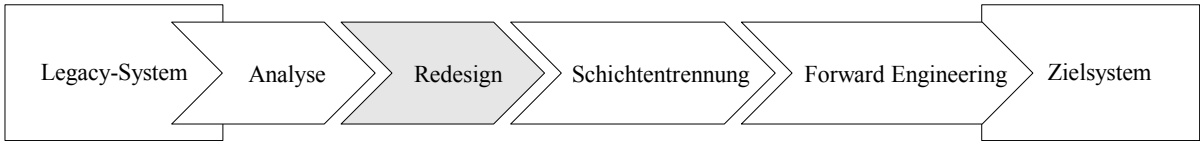
Fachliche Abläufe. Anhand der Programmablaufpläne lassen sich Aktivitätsdiagramme erstellen, die unter anderem die Anwendungsfälle präzisieren. Am Beispiel der erstellten Programmablaufpläne der Verwaltungskomplexe Member-Verwaltung und Änderungskomplexe wurden die Aktivitätsdiagramme für die Anwendungsfälle *Member verwalten* und *Änderungskomplexe verwalten* erstellt. Die Aktivitätsdiagramme können im Anhang D.2 ab Seite 186 nachgeschlagen werden.

Ergebnis-Dokumente

In der Phase Analyse wurden durch ein Design Recovery Entwurfsinformationen wiedergewonnen, wie das Zusammenwirken der Komponenten, die Aufrufhierarchien der wichtigsten Programme, Datenstrukturen und fachliche Abläufe. Es wurden auch die Anforderungen des Legacy-Systems wiedergewonnen und dokumentiert. Die Entwurfsinformation und die redokumentierten Anforderung sind die Ergebnis-Dokumente der Phase Analyse:

- Kontrollflussgraphen (Aufrufgraphen, Strukturdiagramme)
- Data Dictionary (Anhang B)
- Programmablaufpläne (Anhang C)
- Anwendungsfalldiagramme (Kontextdiagramm und Anhang D.1)
- Anwendungsfallbeschreibungen (Anhang D.3)
- Aktivitätsdiagramme (Anhang D.2)

5.2 Phase 2 – Redesign



Eingangs-Dokumente

Für die Phase Redesign wird der vollständige Aufrufgraph des REXX-Programms OV.REXX benötigt, um eine Modularisierung mit der Dominanz-Analyse zu ermöglichen. Zusätzlich wird der Quell-Code von OV.REXX benötigt und die Erkenntnisse aus der Phase Analyse, um alternativ zur Dominanz-Analyse die Modulbildung anhand der Begriffsanalyse durchzuführen.

Vorgänge und Anwendung der Techniken

Nachdem in der Phase Analyse das Objektverwaltungssystem untersucht wurde und der Aufbau bzw. die Architektur, die Datenstrukturen und die Abhängigkeiten der Artefakte ermittelt wurden, sollen anschließend die Prozeduren der Objektverwaltung OV.REXX strukturiert werden. Die Strukturierung wird nur am Objektverwaltungsprogramm durchgeführt, weil hier die gesamte Anwendungslogik verankert ist. Ziel ist die Strukturierung der vorhandenen Prozeduren, so dass Prozeduren, die eine funktionale Einheit bilden, zu Modulen bzw. Gruppen zusammengefasst werden können. Diese Module können dann je nach Anforderung und verwendeter Technologie wiederverwendet werden und dienen unter anderem als Vorlage zur Identifizierung von Klassen oder Objekten im Zielmodell. Die Phase Redesign vervollständigt das Programmverständnis und dient als Grundlage für die Phase Schichtentrennung.

Im weiteren Vorgehen wird die Dominanz-Analyse zur Strukturierung auf Prozedur- bzw. Funktionsebene angewendet. Als Eingangsgraph wird der komplette Aufrufgraph der Objektverwaltung OV.REXX herangezogen. Der zum Aufrufgraph zugehörige Dominanz-Baum wird in Abbildung 5.17 auf Seite 115 dargestellt. In Abbildung 5.18 wird der Dominanz-Baum mit identifizierten Modulen dargestellt.

Die kreisförmige Graphendarstellung in Abbildung 5.19 auf Seite 116 entspricht der Abbildung 5.18 und zeigt eine Darstellungsalternative. Bei dieser Graphendarstellung steht die zentrale Steuerung (Hauptprogramm) im Mittelpunkt und wird dadurch hervorgehoben.

Wie die Abbildung 5.17 zeigt, haben die meisten Kindknoten des Vaterknotens Hauptprogramm keine eigenen Kindknoten und unterhalten nur einfache Dominanz-Relationen. Die einfache Dominanz-Relation besagt, dass die betroffenen Prozeduren auch von anderen Prozeduren gleicher oder tieferer Ebene und nicht nur vom Vaterknoten aufgerufen werden. Durch diese flache Baumstruktur zeigt sich, dass eine sehr starke Abhängigkeit zwischen den Prozeduren vorliegt. Somit können die vielen einzelnen Prozeduren als allgemeine Dienste bewertet werden, wie das in der Dominanz-Analyse definiert ist. Zu beachten ist, dass die Dominanz-Analyse Module nicht nach der Semantik bildet, sondern nach den Aufrufabhängigkeiten des Aufrufgraphen. Dadurch wird das Verständnis, was ein Entwickler im Kontext des Legacy-Systems mitbringt, nicht berücksichtigt.

Nach der Dominanz-Analyse gibt es sieben Module, die aber durch die starken Abhängigkeiten nur wenige Prozeduren enthalten. Nach Abbildung 5.18 sind das folgende Module:

- Programminitialisierung
- Datenzugriff
- Datenfeldverwaltung
- Dateiverwaltung
- INFO-Datei (Informations- und Kategorieeinträge verwalten)
- Member-Verwaltung
- Änderungskomplex

Die Modul-Bezeichnungen können nur durch einen Entwickler über die wiedergewonnenen Erkenntnisse der Analyse sinnvoll vergeben werden.

Diese Modulbildung ist ein erster Ansatz, doch liefert die Dominanz-Analyse kein befriedigendes Ergebnis.

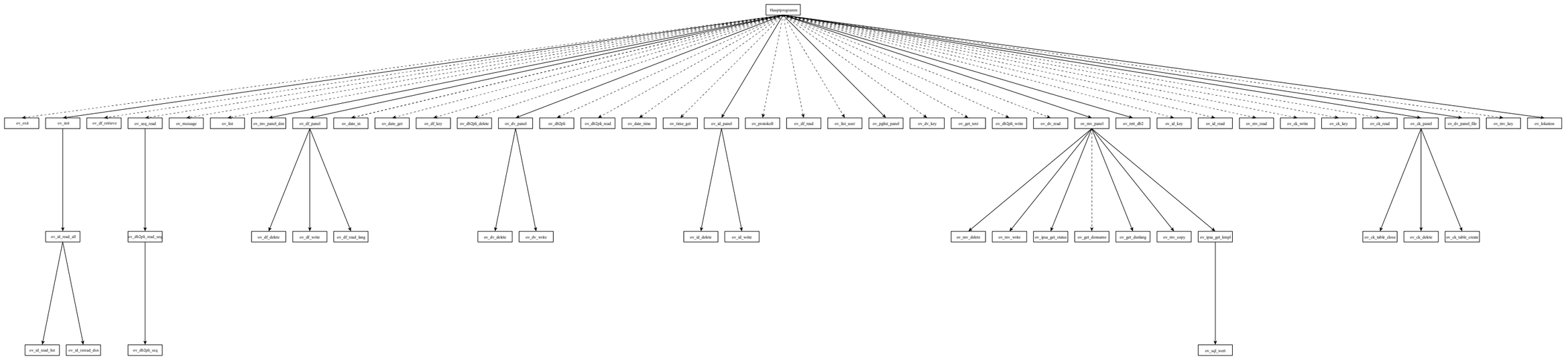


Abbildung 5.17: OV.REXX – Dominanz-Baum auf Prozedur- bzw. Funktionsebene

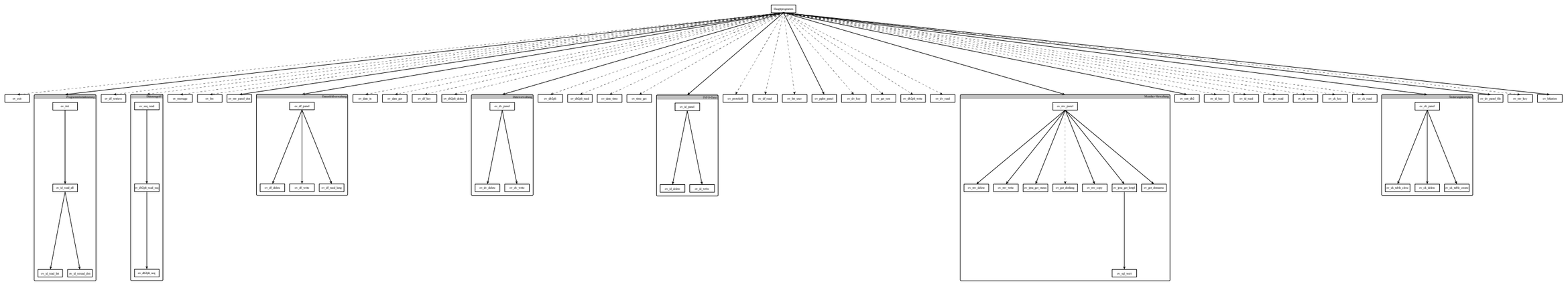


Abbildung 5.18: OV.REXX – Dominanz-Baum – Module bzw. Gruppen

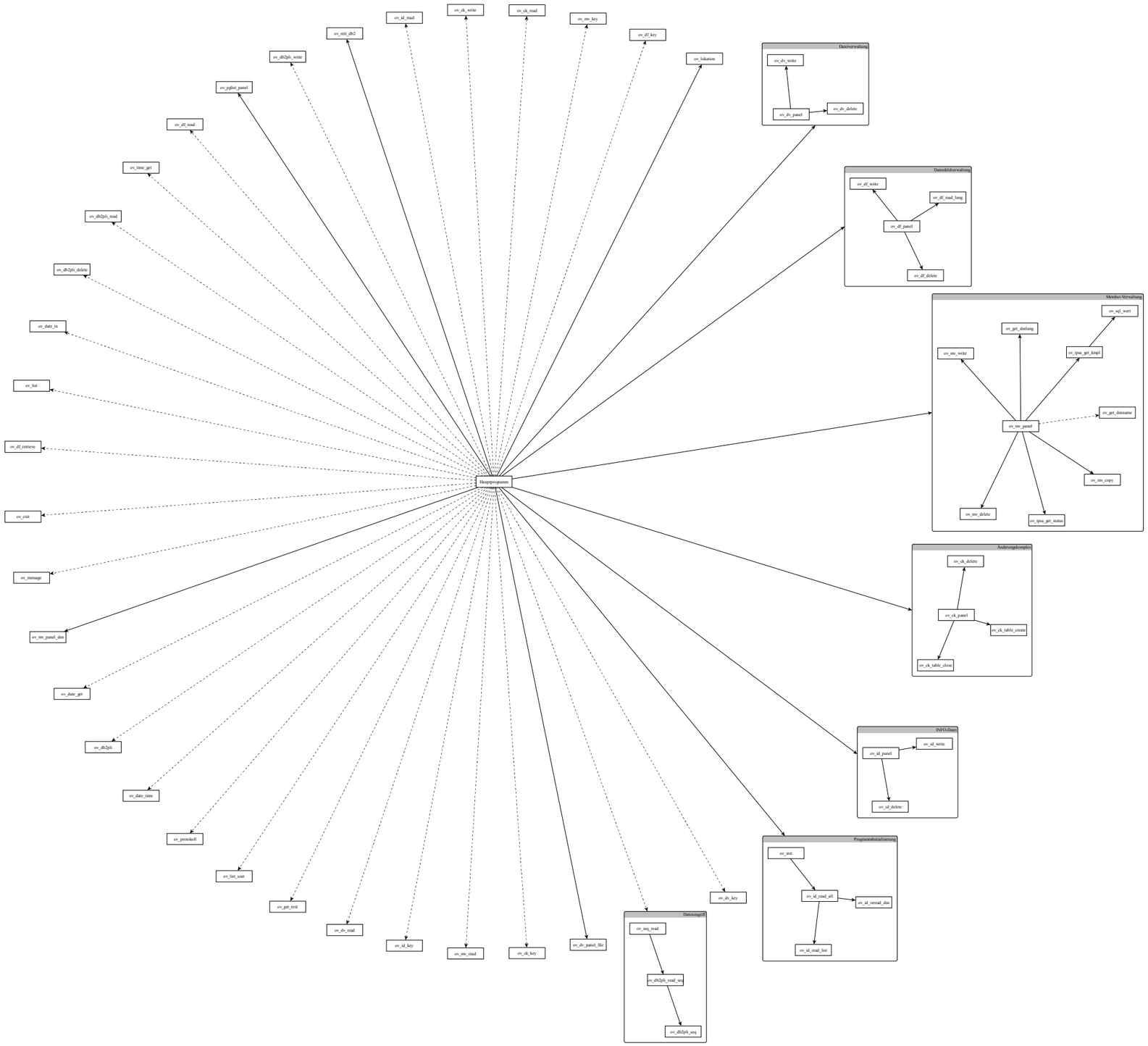


Abbildung 5.19: OV.REXX – Dominanz-Baum - Gruppen (kreisförmige Darstellung)

Eine Modularisierung auf Basis von Quell-Code-Elementen mit der Begriffsanalyse kann in diesem Fall eine bessere Modulbildung aufzeigen. Die Quell-Code-Bezeichner der Prozeduren können als Eigenschaften verwendet werden, weil sie die Funktionalitätszugehörigkeit gut beschreiben. Zudem sind bestimmte Prozeduren im Quell-Code durch Quell-Code-Kommentare thematisch unterteilt.

In der Abbildung 5.20 wird der Aufrufgraph der Objektverwaltung mit den gebildeten Modulen aus den Quell-Code-Bezeichnern dargestellt.

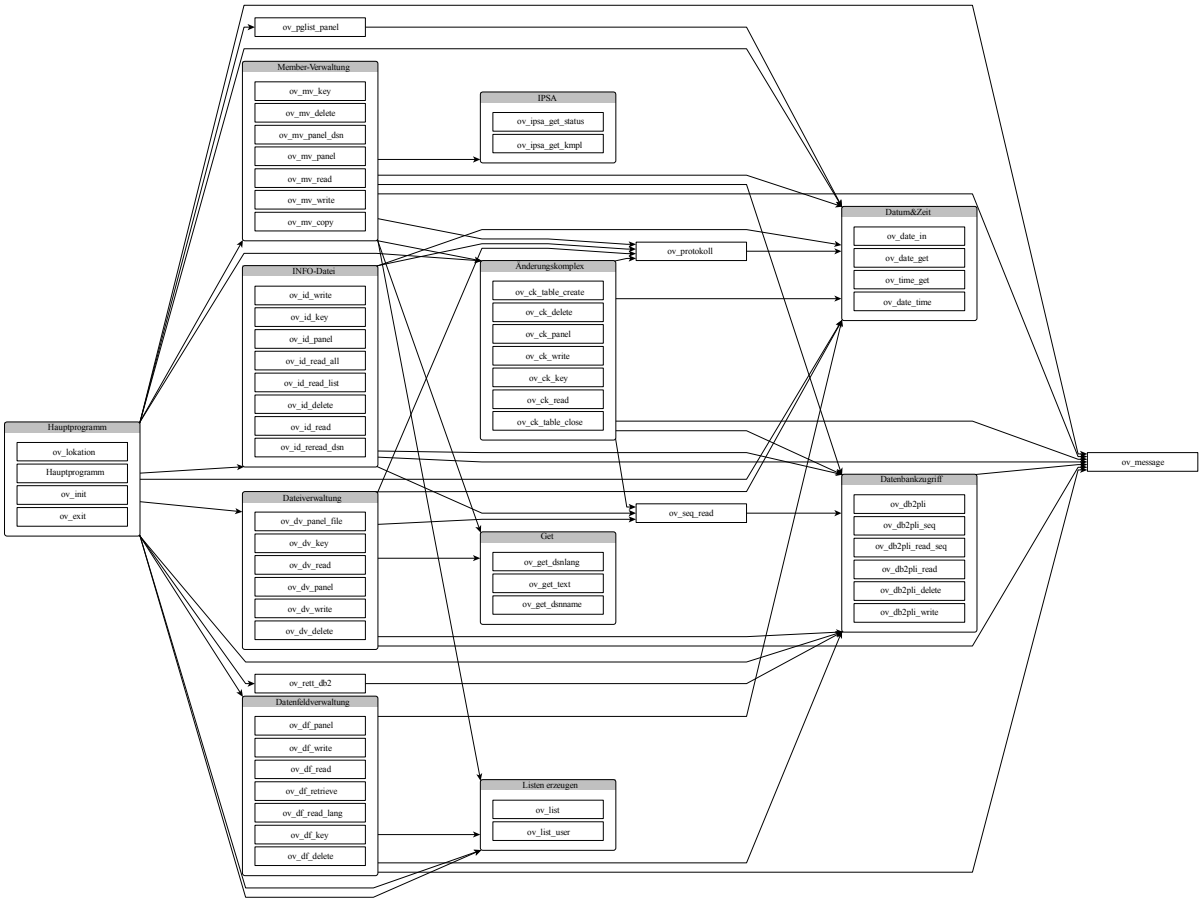


Abbildung 5.20: OV.REXX – Gruppen nach Quell-Code-Bezeichnern/Kommentaren

Wie die Abbildung 5.20 zeigt, sind auch die Module aus der Dominanz-Analyse wiederzufinden. Der Nachteil der Modularisierung anhand der Begriffsanalyse im Gegensatz zur Dominanz-Analyse ist, dass die ermittelten Module untereinander große Abhängigkeiten bilden können.

Die wichtigsten gebildeten Module sind:

- Hauptprogramm (Verwaltungskomplexe anzeigen und auswählen)
- Member-Verwaltung
- INFO-Datei (Informations- und Kategorieeinträge verwalten)
- Dateiverwaltung
- Änderungskomplex
- Datenfeldverwaltung
- Datenbankzugriff

Mit dieser Aufteilung wurden alle Prozeduren in Abhängigkeit ihrer Aufgabengebiete gut zusammenfasst. Zum Beispiel wurden alle Prozeduren die die Member-Verwaltung realisieren zusammengefasst. Auch wurden alle Prozeduren zusammengefasst, die für den Datenbankzugriff verantwortlich sind.

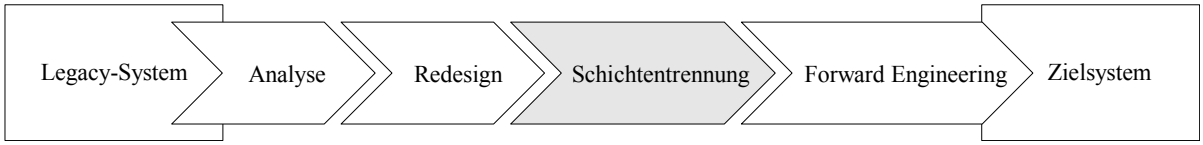
Das Modul *Get* enthält Hilfsprozeduren, welche nicht weiter betrachtet werden. Gleiches gilt für die Prozeduren ov_pglist_panel, ov_rett_db2 und ov_seq_read. Das Modul *Listen erzeugen* beinhaltet Prozeduren, um unter anderem Listen von berechtigten Benutzern zu erstellen.

Das Modul *IPSA* stellt eine Art externe Schnittstelle zur Verfügung. Es wird nicht weiter betrachtet, weil diese Prozeduren speziell nur für ein Umstellungsprojekt benötigt wurden. Das interaktive Projektsystem für Anwendungsentwicklung (IPSA) ist eine Art Versionsverwaltung von Dateien und Verzeichnissen beim Deutschen Ring Hamburg.

Ergebnis-Dokumente

In der Phase Redesign wurden die Prozeduren der Objektverwaltung strukturiert, in dem eine Modularisierung erreicht wurde. Diese Module repräsentieren das Ergebnis-Dokument der Phase Redesign und bilden zugleich die Grundlage der Phase Schichtentrennung. Es wurde auch die Erkenntnis gewonnen, welche Prozeduren gemeinsame Funktionskomplexe bilden.

5.3 Phase 3 – Schichtentrennung



Eingangs-Dokumente

Für die Phase Schichtentrennung werden die Erkenntnisse der Phase Analyse (z.B.: Aufrufgraphen) benötigt und die ermittelten Module aus der Phase Redesign. Für die Begriffsanalyse wird der Quell-Code des REXX-Programms OV.REXX benötigt.

Vorgänge und Anwendung der Techniken

Das Ziel der Phase Schichtentrennung ist die Identifizierung und Zuordnung der Bestandteile des Objektverwaltungssystems zu der Präsentations-, Anwendungs- und Datenhaltungsschicht. Da das Objektverwaltungsprogramm OV.REXX nicht nur Anwendungslogik, sondern auch Teile der Präsentations- und Datenhaltungsschicht beinhaltet, ist zusätzlich die Identifizierung und Zuordnung der ermittelten Module des Objektverwaltungsprogramms zu den Schichten Dialogsteuerung, Dienste und Datenzugriffssteuerung ein Ziel.

Das Objektverwaltungssystem ist eindeutig in Schichten zerlegbar. In der Abbildung 5.21 wird der Aufrufgraph des Objektverwaltungssystems auf Anwendungs- bzw. Komponentenebene mit getrennten Schichten dargestellt.

Das Software-Produkt ISPF bildet die Präsentationsschicht. Demnach gehören alle ISPF-Panel-Definitionsdateien und ISPF-Message-Dateien dazu. Diese beinhalten das definierte Erscheinungsbild der zeichenorientierten Benutzerschnittstelle und Teile der Dialogsteuerung. Alle PL/1-Programme und das Objektverwaltungsprogramm bilden die Anwendungsschicht. Die Datenhaltungsschicht wird von der DB2-Datenbank gebildet. Zu beachten ist, dass die aufgeteilten Schichten nicht im Sinne einer Client-Server-Anwendung zusammenarbeiten.

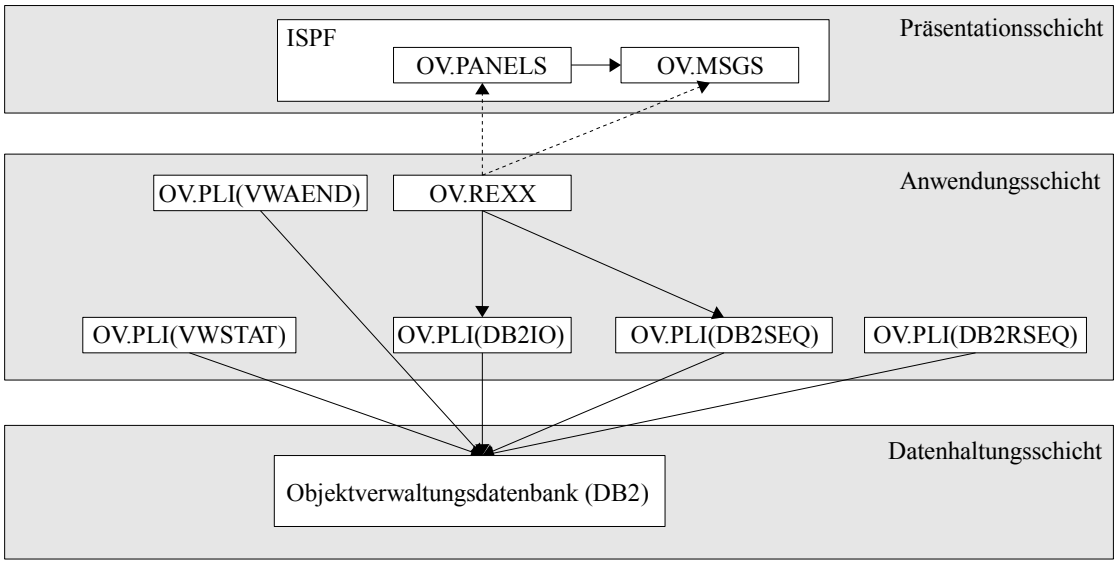


Abbildung 5.21: Aufrufgraph auf Anw.- bzw. Komp.-ebene mit getrennten Schichten

Im nächsten Schritt werden die Prozeduren bzw. die Module des Objektverwaltungsprogramms OV.REXX zerlegt. Der flache Aufbau des Dominanz-Baumes aus der Phase Redesign zeigte, dass die Prozeduren stark miteinander verwoben sind und dadurch keine befriedigende Modulbildung erreicht werden konnte. Das Objektverwaltungsprogramm wurde damit in teilweise zerlegbar bis unzerlegbar eingestuft. Die anschließende Modulbildung durch die Begriffsanalyse hatte zur Folge, dass die vielen Abhängigkeiten erhalten blieben und es keine eindeutigen Schnittstellen gibt.

Die Einteilung der Module zu den Schichten wird anhand der Begriffsanalyse durchgeführt.

Die Steuerung eines ISPF-Panels ist mit dem Aufruf einer ISPF-Bildschirmmaske verknüpft und im Quell-Code durch einen ISPF-Befehl identifizierbar. Diese ISPF-Befehle werden als Eigenschaften zur Identifizierung der Dialogsteuerung verwendet. Die Bedeutungen der ISPF-Befehle können unter [LF05] (Kapitel 19.3.5) nachgeschlagen werden. In der Tabelle 5.6 werden die Eigenschaften aufgelistet.

E_1	verwendet den ISPF-Befehl DISPLAY PANEL(...)
E_2	verwendet den ISPF-Befehl TBDISPL "..." PANEL(...)
E_3	verwendet den ISPF-Befehl SELECT PANEL(...)

Tabelle 5.6: Begriffsanalyse (Fallbeispiel): Dialogsteuerung – Eigenschaften

Die Tabelle 5.7 zeigt, welche Prozeduren welche Eigenschaften erfüllen. Genau diese Prozeduren rufen ISPF-Bildschirmmasken auf; was auch aus den meisten Prozedurbezeichnungen durch den Bezeichner *panel* ermittelbar ist.

		E_1	E_2	E_3
1	Hauptprogramm	✓		
2	ov_mv_panel	✓		
3	ov_ck_panel		✓	
4	ov_df_panel	✓		
5	ov_dv_panel	✓		
6	ov_id_panel	✓		
7	ov_pglist_panel			✓
8	ov_dv_panel_file		✓	
9	ov_mv_panel_dsn		✓	

Tabelle 5.7: Begriffsanalyse (Fallbeispiel): Dialogsteuerung – Gegenüberstellung

Im weiteren Vorgehen der Begriffsanalyse können nun Eigenschaften zusammengefasst werden. In diesem Fall ist es sinnvoll alle angegebenen Prozeduren und das Konstrukt Hauptprogramm zum Dialogsteuerungs-Konzept zusammenzufassen (Tabelle 5.8).

Konzeptbezeichner	**Objekte**	**Eigenschaften**	
k_1	1, 2, 4, 5, 6	E_1	Haupt-Panel-Steuerung
k_2	1, 2, … , 9	E_1, E_2, E_3	Dialogsteuerungs-Konzept
k_3	2,9	E_1, E_2	Dialogsteuerung-MV

Tabelle 5.8: Begriffsanalyse (Fallbeispiel): Dialogsteuerung – Konzepte

Für die Datenzugriffssteuerung können die in Tabelle 5.9 aufgelisteten Eigenschaften herangezogen werden. Die ausgewiesenen Konstanten beinhalten die Namen der PL/1-Programme für den Zugriff auf die DB2-Datenbank. Es muss demnach Prozeduren geben, welche die PL/1-Programme ausführen. Weitere Eigenschaften sind die zwei Prozeduren ov_db2pli und ov_db2pli_seq, welche eindeutige Kandidaten der Datenzugriffssteuerung sind.

E_1	verwendet die Konstante: KONST.DB.UPRO
E_2	verwendet die Konstante: KONST.DB.UPROSEQ
E_3	verwendet die Prozedur: ov_db2pli
E_4	verwendet die Prozedur: ov_db2pli_seq

Tabelle 5.9: Begriffsanalyse (Fallbeispiel): Datenzugriffssteuerung – Eigenschaften

Die Tabelle 5.10 listet alle Prozeduren auf, die eine oder mehrere Eigenschaften erfüllen.

		E_1	E_2	E_3	E_4
1	ov_db2pli	✓			
2	ov_db2pli_seq		✓		
3	ov_db2pli_delete			✓	
4	ov_db2pli_read			✓	
5	ov_db2pli_write			✓	
6	ov_db2pli_read_seq				✓
7	ov_rett_db2			✓	

Tabelle 5.10: Begriffsanalyse (Fallbeispiel): Datenzugriffs-Strg. – Gegenüberstellung

Offensichtlich sind nur die Prozeduren ov_db2pli und ov_db2pli_seq für den Aufruf und den Datenaustausch zwischen den PL/1-Programmen verantwortlich. Die anderen Prozeduren verwenden diese je nach gewünschter Datenbankarbeit (lesen, schreiben, löschen usw.). In diesem Fall ist die Zusammenfassung aller Eigenschaften zum Datenzugriffssteuerungs-Konzept sinnvoll, weil alle ermittelten Prozeduren unterschiedliche Datenbankarbeitsfunktionen umsetzen und eine Art zentrale Schnittstelle anbieten (Tabelle 5.11).

Konzeptbezeichner	**Objekte**	**Eigenschaften**	
k_1	3, 4, 5, 7	E_3	Nutzung gemeinsamer Prozedur ov_db2pli
k_2	1,2	E_1, E_2	Prozeduren rufen PL/1-Programme für Datenbankzugriff auf
k_3	1, 2, 3, 4, 5, 6, 7	E_1, E_2, E_3, E_4	Datenzugriffssteuerungs-Konzept

Tabelle 5.11: Begriffsanalyse (Fallbeispiel): Datenzugriffssteuerung – Konzepte

Mit den Erkenntnissen der Begriffsanalyse und den in der Phase Redesign ermittelten Modulen, können nun die Prozeduren bzw. die Module unterteilt werden. In der Abbildung 5.22 wird der Aufrufgraph des Objektverwaltungsprogramms mit unterteilten Schichten dargestellt.

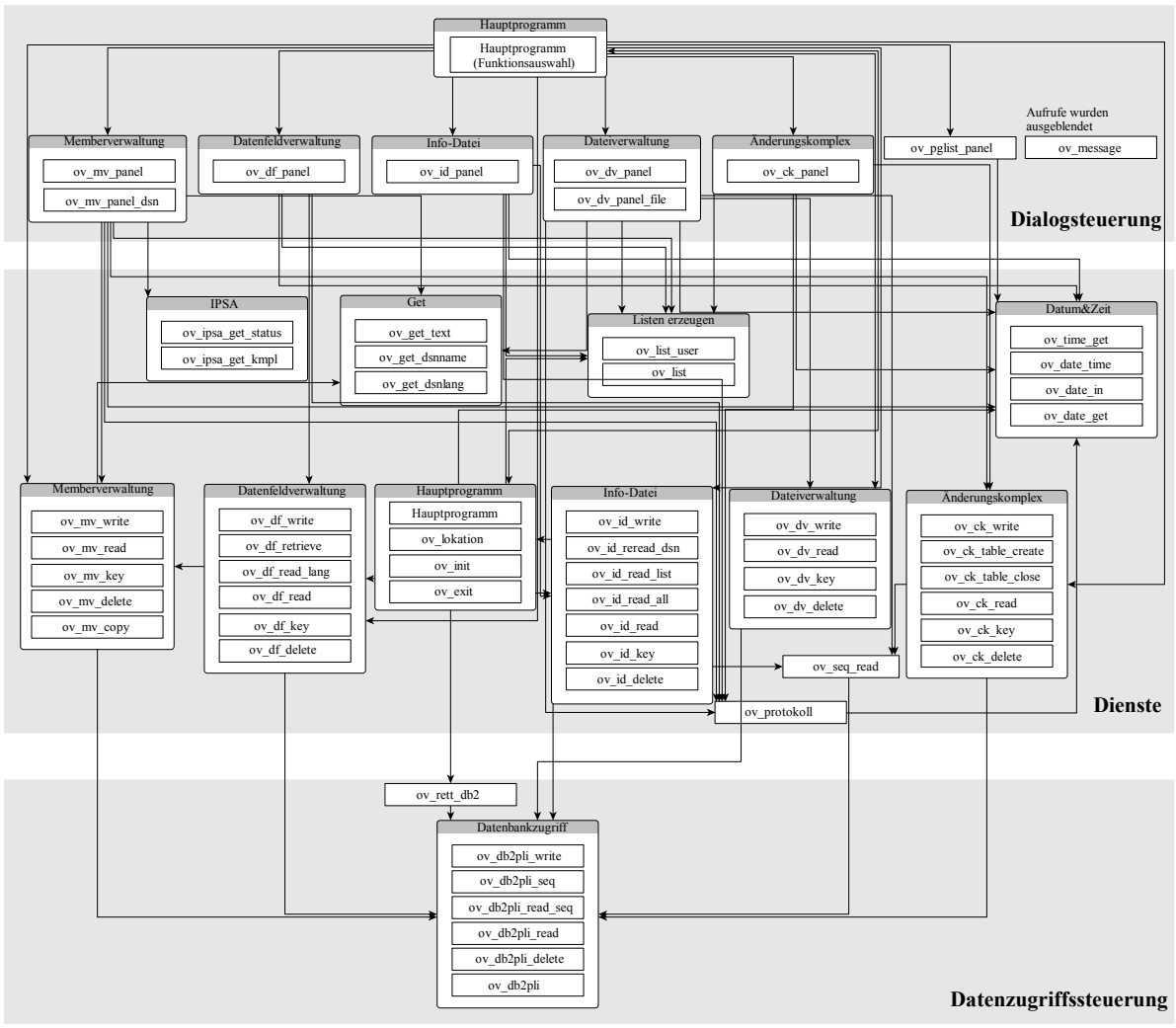


Abbildung 5.22: OV.REXX – Anwendungsschicht mit unterteilten Schichten

Die Abbildung 5.22 zeigt, dass als Grundlage die ermittelten Module der Phase Redesign verwendet wurden. Damit die Zuordnung in Dialogsteuerung, Dienste und Datenzugriffssteuerung umgesetzt werden kann, müssen einige Prozeduren aus den Modulen entnommen werden. Zum Beispiel werden alle ermittelten Prozeduren der Dialogsteuerungsschicht aus ihren Modulen entnommen. Um die Zugehörigkeit der entnommenen Prozeduren zu den Modulen zu waren, werden die Module doppelt dargestellt. Zur Dialogsteuerungsschicht

wurde auch die Prozedur ov_message zugeordnet, weil diese für die Nachrichtenanzeige verantwortlich ist. Alle Prozeduren, die nicht zur Dialog- oder Datenzugriffssteuerung zugeordnet wurden, gehören zur Schicht Dienste und stellen die Fachlogik oder allgemeine Dienste (z.B.: Datum&Zeit) dar.

Identifizieren von Interaktionsobjekten. Die Anwendung von MORPH, zur Erkennung der Präsentationsschicht bzw. der Benutzerschnittstelle, ist für dieses Fallbeispiel nicht notwendig, weil die ISPF-Panel-Definitionsdateien die Benutzerschnittstelle explizit identifizieren. Es können aber Interaktionsobjekte aus der zeichenorientierten Benutzerschnittstelle für den Aufbau der neuen graphischen Benutzeroberfläche identifiziert werden. Am Beispiel der ISPF-Panel-Definitionsdatei *OV.PANELS(OV02)* (Member-Verwaltung) werden Interaktionsobjekte identifiziert. In Tabelle 5.12 wird der zur Identifizierung benötigte Quell-Code-Ausschnitt aus der Datei OV.PANELS(OV02) angegeben.

Tabelle 5.12: Fallbeispiel: OV.PANELS(OV02) – Interaktionsobjekte identifizieren

```
...  ...
57   )ATTR
58    /* Normaler Text */
59    + TYPE(NT) SKIP(ON) /* NT: normal text */
60    /* Überschrift */
61    § TYPE(PT) SKIP(ON) /* PT: panel title */
62    /* Ausgabefeld */
63    $ TYPE(VOI) SKIP(ON) /* VOI: variable output information */
64    /* Eingabefeld */
65    _ TYPE(INPUT) CAPS(ON) COLOR(TURQ) HILITE(USCORE) /* input field */
66    /* Auswahlfeld */
67    ? TYPE(CEF) CAPS(ON) /* CEF: choice entry field */
68    /* Beschreibung */
69    = TYPE(PIN) SKIP(ON) /* PIN: panel instruction */
70    /* Feldbeschreibung */
71    * TYPE(ET) SKIP(ON) /* ET: emphasized text */
...  ...
77   )BODY CMD() EXPAND(##)
78   §csd          Memberverwaltung# #&pdate          +
79   -OV02#-#
```

▼

```
80  +
81  +Member:   $Z   + # # Typ:_Z*(&pmvobtxt)# #+Sprache:_Z*(&pmvpgtxt) +
82  +Dateiname:$Z *(&pmvfltxt)
83  +
84  +
85  +Funktionskomplex:   _Z       +# #+Bearbeiter:_Z     ++
86  +Änderungskomplex:   $Z  $Z  $Z  $Z  +# #+Fremdprojekt-Nr:$Z      +
87  +Status Bearbeitung: _Z*  (&pmvsttxt)# #+Protokolldruck:$Z+     +
88  +Überstellungszähler:$Z +
89  +nicht aktiv:        _Z+# #+Grund:_Z                              +
90  +
91  +Datum:
92  + maschinelle  Schnell-     Fein-        Änderung
93  + Analyse    Analyse    Analyse    geprüft    Vorübergabe   Übergabe
94  +$Z       + $Z      + $Z        + $Z         + $Z        + $Z      +
95  +
96  +
97  +Auswahl:?Z+
98  +
99  #-#
... ...
```

Ab der Zeile 57 *)ATTR* beginnt für die verschiedenen Panel-Elemente die Definition der Attributzeichen. Mit dem Attribut *TYPE(value)* wird die Kategorie eines Panel-Elementes definiert. Zum Beispiel steht das Zeichen + für *normalen Text (NT)* oder das Zeichen _ für *Eingabefeld (INPUT)*. Die Spezifikation aller möglichen Panel-Elemente kann unter [IBM08] (Kapitel 7.1.5.2) nachgeschlagen werden. Im Weiteren wird im Quell-Code ab Zeile 77 *)BODY* die Vorlage für die zeichenorientierte Benutzerschnittstelle definiert, wo die Zeichen aus der Sektion *)ATTR* verwendet werden.

Die erkannten Panel-Elemente lassen sich durch ihre Definition sehr leicht zu einem Interaktionsobjekt zuordnen. Zum Beispiel kann das Panel-Element *$* für *variable output information (VOI)* zum Interaktionsobjekt *Read-only message* zugeordnet werden. Dieses kann dann in einer expliziten Technologie wie Java Swing zu *jLabel* oder bei Java Server Faces zu *<h:outputText>* zugeordnet werden. Die Längen der Zeichenketten sind für die Member-Verwaltung, wie in der Tabelle im Anhang B.1 definiert, kurz (< 80 Zeichen) und benötigen

deshalb nur einfache zeilenbasierte Interaktionsobjekte. In der Tabelle 5.13 werden die vorläufigen Zuordnungen der Panel-Elemente für die Member-Verwaltung vorgegeben.

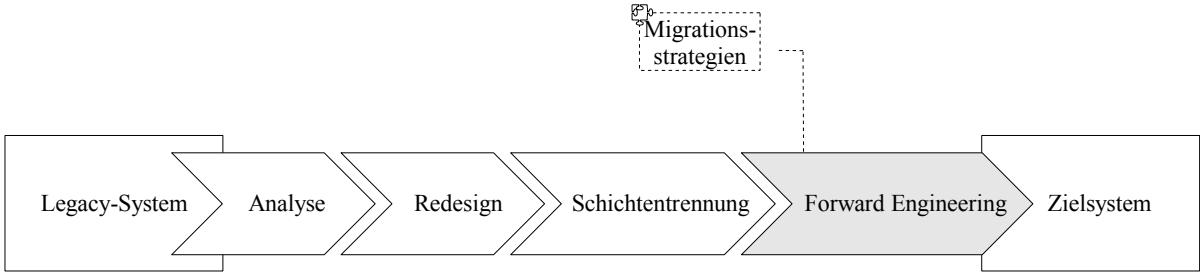
Beschreibung	Panel-Variable	Panel-Element	Interaktionsobjekt	z.B.: Java Server Faces
Member-Name	pmvmemb	$ (VOI)	Read-only message	<h:outputText>
Member-Typ	pmvobj	_ (INPUT)	Single Line Text Field	<h:inputText>
Dateiname	pmvfile	$ (VOI)	Read-only message	<h:outputText>
Programmiersprache	pmvprog	_ (INPUT)	Single Line Text Field	<h:inputText>
Funktionskomplex	pmvfunk	_ (INPUT)	Single Line Text Field	<h:inputText>
Bearbeiter	pmvuid	_ (INPUT)	Single Line Text Field	<h:inputText>
Änderungskomplex	pmvkmpl1-4	$ (VOI)	Read-only message	<h:outputText>
Fremdprojekt-Nr.	pmvfproj	$ (VOI)	Read-only message	<h:outputText>
Bearbeitungsstatus	pmvstat	_ (INPUT)	Single Line Text Field	<h:inputText>
Protokolldruck	pmvdruf	$ (VOI)	Read-only message	<h:outputText>
Überstellungszähler	pmvcnt	$ (VOI)	Read-only message	<h:outputText>
nicht aktiv	pmvnakt	_ (INPUT)	Single Line Text Field	<h:inputText>
Grund	pmvnagrd	_ (INPUT)	Single Line Text Field	<h:inputText>
Datum masch. Analyse	pmvdat1	$ (VOI)	Read-only message	<h:outputText>
Datum schnell. Analyse	pmvdat2	$ (VOI)	Read-only message	<h:outputText>
Datum Feinanalyse	pmvdat3	$ (VOI)	Read-only message	<h:outputText>
Datum Änd. geprüft	pmvdat4	$ (VOI)	Read-only message	<h:outputText>
Datum Vorübergabe	pmvdat5	$ (VOI)	Read-only message	<h:outputText>
Datum Übergabe	pmvdat6	$ (VOI)	Read-only message	<h:outputText>

Tabelle 5.13: Fallbeispiel: Member-Verwaltung – Interaktionsobjekte

Ergebnis-Dokumente

Durch die Schichtentrennung wurden die Bestandteile des Objektverwaltungssystems zu den drei Schichten der Drei-Schichten-Architektur zugeordnet. Zusätzlich wurden die Module bzw. die Prozeduren der Anwendungsschicht des Objektverwaltungsprogramms in die Schichten Dialogsteuerung, Dienste und Datenbankzugriffssteuerung zerlegt. Durch diese Zerlegungen hat man ein zum Zielsystem äquivalentes System, welches für die Modellierung des Zielmodells verwendet wird und somit als Ergebnis-Dokument gilt.

5.4 Phase 4 – Forward Engineering



Eingangs-Dokumente

Für die Phase Forward Engineering werden alle wiedergewonnenen Informationen über das Legacy-System der vorangegangenen Phasen benötigt. Es werden die wiedergewonnenen Anforderungen auf ihre Wiederverwendbarkeit hin überprüft. Zusätzlich dienen die Module aus der Phase Redesign zur Klassenfindung.

Anforderungen

Funktionale Anforderungen. Die wiedergewonnenen funktionalen Anforderungen des Legacy-Systems werden auf ihre Wiederverwendbarkeit hin eingeschätzt und in die Kategorien nach Kapitel 4.5.3 eingeteilt. Die Einteilung wurde in der Tabelle 5.14 vorgenommen.

Tabelle 5.14: Fallbeispiel: Forward Engineering – Kat.-einteilung. fkt. Anforderungen

Anforderung	Kategorie
Benutzer verwalten	
Member verwalten	
Member anzeigen	
Member bearbeiten oder neuen Member hinzufügen	
Member löschen	
Dateien verwalten	
Dateieintrag anzeigen	
Dateieintrag bearbeiten oder hinzufügen	
Dateieintrag löschen	

Anforderung	Kategorie
Änderungskomplexe verwalten	
Änderungskomplex erstellen	
Änderungskomplex löschen	
Änderungskomplex mit zugehörigen Dateien und Member anzeigen	
Änderungskomplexbeschreibung hinzufügen oder bearbeiten	
Informations- und Kategorieeinträge verwalten	
Informations- und Kategorieeintrag anzeigen	
Informations- und Kategorieeintrag bearbeiten und oder hinzufügen	
Informations- und Kategorieeintrag löschen	
Verwaltungskomplexe anzeigen und auswählen	
Statistiken, Auswertungs- und Übergabeprotokolle erstellen	

wiederverwendbar | nicht wiederverwendbar | teilweise wied. | neu

Alle funktionalen Anforderungen sind wiederverwendbar. Die Anforderung *Informations- und Kategorieeinträge verwalten* muss angepasst werden, weil die Kategorien nicht mehr in einer einzigen Tabelle verwalten werden und sich dadurch die Umsetzung der Verwaltungsaktivitäten (z.B.: löschen, anzeigen oder hinzufügen) ändern. Die Umsetzung der Verwaltung der Informations- und Kategorieeinträge wird nicht weiter betrachtet.

Eine im Zielsystem neue Anforderung ist *Benutzer verwalten*. Diese muss neu hinzugenommen werden, weil im Legacy-System keine Benutzerverwaltung realisiert, aber die Benutzer des Betriebssystems als Benutzer des Legacy-Systems verwendet wurden. Diese Situation spiegelt das soziotechnische Gebilde wieder, in dem das Legacy-System betriebsplattformeigene Eigenschaften verwendet, welche im Zielsystem nicht mehr vorhanden sind. Bei der Umsetzung des Legacy-Systems in ein JEE-Zielsystem ist eine Benutzerverwaltung erforderlich, um unter anderem Benutzer anlegen oder löschen zu können und die Bearbeitungsberechtigung zu realisieren. Das Zielsystem ist damit ein Mehrbenutzersystem.

Entwurf

Eine Wiederverwendung der im Fallbeispiel verwendeten Komponenten, zum Beispiel das ISPF-Software-Produkt, die PL/1-Programme oder das REXX-Programm, ist nicht möglich. Die Technologien entsprechen nicht mehr dem Stand der Technik und sind von den Eigenschaften der Betriebsplattform abhängig, welche in der Zielplattform nicht mehr vorherrschen. Die DB2-Datenbank könnte wiederverwendet werden. Im Fallbeispiel wird diese Option nicht berücksichtigt.

Die ermittelten Module des Objektverwaltungsprogramms sind für eine Wiederverwendung im Sinne einer Code-zu-Code-Transformation nicht geeignet, weil unter anderem die Abhängigkeiten zwischen den ermittelten Modulen zu stark sind und durch den massiven Einsatz von globalen Variablen keine einheitlichen Schnittstellen realisiert werden können. Die Prozeduren in den Modulen stellen keine isolierfähigen Kernfunktionalitäten dar, sondern sind unter anderem stark abhängig von den Funktionalitäten des ISPF-Software-Produkts.

Auswahl der Zieltechnologie und Systemkomponenten. Zur Umsetzung der in Kapitel 3.3 festgelegten Zielsystemeigenschaften wird die Java Platform Enterprise Edition (JEE)-Technologie herangezogen. Mit JEE können auf Basis der objektorientierten Programmiersprache Java Drei- und Mehrschichten-Architekturen umgesetzt werden. In Abbildung 5.23 werden die verschiedenen Schichten der verwendeten JEE-Architektur dargestellt.

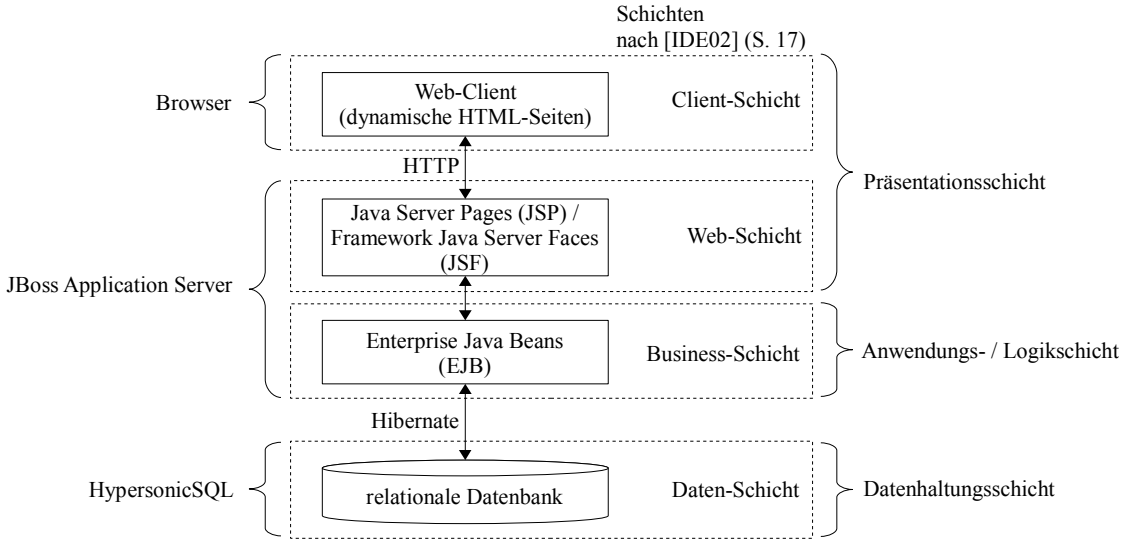


Abbildung 5.23: Fallbeispiel: Zielsystem (JEE - Architektur)

Die Anwendungs- bzw. Logikschicht wird durch die Enterprise Java Beans (EJB)-Technologie realisiert. Wobei EJB-Objekte (spezielle Java-Objekte) die Logik der Anwendung als Dienste zur Verfügung stellen. Diese EJB-Objekte werden von einem EJB-Container eines JEE-Anwendungs-Servers verwaltet. Die Geschäftsobjekte (Entitäten), also die Daten der EJB-Objekte, werden automatisch in einer relationalen Datenbank persistent gehalten.

Die Präsentationsschicht ist in Erzeugung und Anzeige der Benutzeroberfläche unterteilt. Für die Benutzeroberfläche stehen dynamische HTML-Seiten, welche nach dem Aufruf einer URL[45] über die Java Server Faces (JSF)-Technologie erzeugt und anschließend in einem Browser angezeigt werden. Die JSF-Technologie ist eine spezielle Erweiterung der Java Server Pages (JSP). Bei JSF wird im Unterschied zu JSP die Dialogsteuerung getrennt von der eigentlichen Definition der Benutzeroberfläche realisiert. So befindet sich die Dialogsteuerung in speziellen Java-Beans und die JSP-Dateien beinhalten lediglich Interaktionsobjekte und Variablen, die durch spezielle JSF-XML[46]-Tags[47] definiert werden. Bei JSP befinden sich Logik bzw. Dialogsteuerung und Interaktionsobjekte gemeinsam in einer JSP-Datei. Kontrollstrukturen werden bei JSP durch spezielle JSP-XML-Tags umgesetzt.

Detaillierte Informationen über die JEE-Technologie, vor allem über die verschiedenen JEE-Komponenten (EJB, JSP, JSF), können unter [DJ08] (Kapitel 4.2.3) nachgeschlagen werden.

Als Anwendungs-Server wird der JBoss Application Server und als relationale Datenbank wird HypersonicSQL verwendet. In der Tabelle 5.15 werden die Systemkomponenten der Laufzeitumgebung aufgelistet.

45 Ein Uniform Resource Locator (URL) ist ein einheitlicher Identifizierer für Ressourcen in einem Computernetzwerk.

46 Extensible Markup Language (XML) ist eine erweiterbare Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Textform.

47 Ein Tag bezeichnet im Bereich von Auszeichnungssprachen wie XML die in Kleiner- und Größerzeichen eingeschlossenen Kürzel (z.B.: <body>).

Laufzeitumgebung des Zielsystems		
JBoss Application Server (enthält Datenbank HypersonicSQL)	**Beschreibung**	Anwendungs-Server
	Hersteller	http://www.jboss.org/jbossas/
	Version	4.0.5.GA (stable)
	Lizenz	LGPL
HypersonicSQL	**Beschreibung**	relationale Datenbank (Java)
	Hersteller	http://hsqldb.org
	Lizenz	HSQLDB-Lizenz

Tabelle 5.15: Fallbeispiel: Laufzeitumgebung des Zielsystems (Stand vom: 30.07.2009)

Auswahl der Entwicklungswerkzeuge. Um das festgelegte Zielsystem mit der ausgewählten Technologie durch modellgetriebene Software-Entwicklung zu realisieren, wird das kommerzielle Entwicklungswerkzeug *objectiF* verwendet. Dieses stellt systemeigene Modelltransformationsvorlagen zur Verfügung, mit denen man verschiedene Zielsysteme nach dem MDA-Ansatz entwickeln kann. Der MDA-Ansatz wird in objectiF mit der Umsetzungsalternative realisiert. Zur Umsetzung des Zielsystems wird die Vorlage „*Web Application in Java*“ verwendet, mit der man ein Zielsystem auf Basis von JEE entwickeln kann. Auf eine detaillierte Beschreibung des Modellierungsvorgangs mit objectiF und der genannten Vorlage wird wegen des Umfangs verzichtet. Ein Leitfaden und detaillierte Informationen über den Modellierungsvorgang mit objectiF können unter [BS09] (Kapitel 3 und Anhang D) nachgelesen werden. Im Laufe der Entwurfsvorstellung werden einzelne Vorgänge der modellgetriebenen Software-Entwicklung mit objectiF deutlich. Für die Implementierung wird die integrierte Entwicklungsumgebung Eclipse verwendet, denn objectiF generiert bei Projekterstellung automatisch entsprechende Eclipse-Projekte. Die Kombination aus objectiF und Eclipse ermöglicht sogenanntes Round Trip Engineering, welches zwischen Modell (Diagramm) und Implementierung für Konsistenz sorgt. Die Konsistenz wird sicher gestellt, in dem nur in ausgewiesenen Bereichen (durch Kommentare) im generierten Quell-Code implementiert werden sollte.

In der Tabelle 5.16 werden die Entwicklungswerkzeuge, die zusätzlich zu objectiF verwendet werden, aufgelistet.

Entwicklungswerkzeuge		
Eclipse Ganymede IDE for Java EE	**Beschreibung**	integrierte Entwicklungsumgebung
	Hersteller	http://www.eclipse.org
	Version	3.4.2
	Lizenz	open source
Java Sun JDK	**Beschreibung**	Laufzeitumgebung und Werkzeuge für die Java-Entwicklung
	Hersteller	http://java.sun.com
	Version	6 update 13
	Lizenz	GPL

Tabelle 5.16: Fallbeispiel: Entwicklungswerkzeuge (Stand vom: 29.07.2009)

Die verwendeten Entwicklungswerkzeuge und Laufzeitumgebungen befinden sich auf der Begleit-CD [CHRCD09] im Verzeichnis [Software].

Plattformunabhängiges Modell. Die Modellierung des plattformunabhängigen Modells ist durch die Vorlage von objectiF vorgegeben und umfasst folgende Punkte:

- Datenmodell (MyEntities)
- Fachlogik/Dienste (MyServices)
- Oberflächennavigation/Oberflächenseiten (MyPresentation)

Der Oberflächenentwurf wird mit graphischen Mitteln realisiert, weil dieser in objectiF nicht modelliert werden kann.

Datenmodell (MyEntities). Im ersten Schritt wird über ein Klassendiagramm das plattformunabhängige Datenmodell realisiert, wobei nur die Attribute einer Klasse bei der Modellierung wichtig sind. Unabhängig heißt, dass bei den definierten Attributen einer Klasse plattformunabhängige Datentypen verwendet werden, welche objectiF bei der Code-Generierung durch äquivalente Java-Datentypen ersetzt. Zum Beispiel wird der unabhängige Datentyp *Alphanumeric* durch den Java-Datentyp *String* ersetzt. Im Klassendiagramm stellt eine Klasse eine Entität dar, welche in objectiF den Stereotyp *<<BusinessEntity>>* besitzen muss, damit sie bei der Generierung beachtet wird. Die Beziehungen zwischen den Klassen entsprechen der UML-Spezifikation und beinhalten die in einem ER-Modell gebräuchlichen

Kardinalitäten. Das modellierte plattformunabhängige Datenmodell des Objektverwaltungssystems kann im Anhang E.2 auf Seite 201 eingesehen werden.

Grundlage für das Datenmodell sind die wiedergewonnenen Tabellen aus der Objektverwaltungsdatenbank (Anhang B). Zum Beispiel konnten die Tabellen MVSATZ, DVSATZ und CKSATZ jeweils direkt als Entität übernommen werden. Zu beachten ist, dass die alten Primärschlüssel nicht mehr verwendet werden, aber als Bezeichner erhalten bleiben. Zum Beispiel bleibt der alte Primärschlüssel CKKMPL der Tabelle CKSATZ als Bezeichner eines Änderungskomplexes erhalten. Der alte Primärschlüssel wird durch einen Wert des Datentyps Long ersetzt, welcher automatisch bei der Code-Generierung erzeugt wird.

Die im Legacy-System verwendeten Kennzeichen aus der INFO-Tabelle (Informations- und Kategorieeinträge) wurden separiert. Jede Kategorie von Kennzeichen wird als eigenständige Entität realisiert. Zum Beispiel existiert eine Kategorie Programmiersprache, Membertyp oder Bearbeitungsstatus. Zur Protokollierung von Aktivitäten wurde im Legacy-System eine Protokolldatei verwendet, welche im Datenmodell als eigenständige Entität (PRSATZ) realisiert wurde. Neu ist die Entität Benutzer und die Möglichkeit Benutzergruppen und Gruppenrechte zu vergeben, damit nur berechtigte Benutzer bestimmte Aktivitäten (z.B.: Löschen, Bearbeiten) im Objektverwaltungssystem durchführen können.

Aus dem Datenmodell generiert objectiF automatisch ein Eclipse-Projekt zur Erzeugung von Entity Beans[48]. Die Entity Beans werden von einem Anwendungs-Server als Grundlage zur Erstellung der Persistenzschicht verwendet - es werden automatisch Tabellen in einer relationalen Datenbank erstellt. Die Entity Beans ermöglichen den Austausch der fachlichen Daten zwischen den Schichten. Zum Beispiel stellt die Entität (das Objekt) MVSATZ die frühere globale Variable MVSATZ dar.

Das ermittelte Modul Datenbankzugriff der Schicht Datenzugriffssteuerung wird vollständig durch den Persistenzmechanismus (Hibernate[49]) des Anwendungs-Servers ersetzt. Damit übernimmt die Plattform den Datenbankzugriff.

48 Entity Beans sind Enterprise Java Beans (EJB), die speziell die persistenten Daten abbilden.
49 Hibernate ist ein Framework und ein objekt relationaler Mapper, der Objekte wie Entity Beans in Datensätze einer relationalen Datenbank abbildet und umgekehrt.

Fachlogik/Dienste (MyServices). Um Fachlogik oder allgemeine Dienste im Objektverwaltungssystem zur Verfügung zu stellen, werden in einem Klassendiagramm Dienste in Form von Klassen modelliert. Diese Klassen müssen den Stereotyp <<*BusinessService*>> haben, damit sie bei der Code-Generierung beachtet werden. Verschiedene Dienstoperationen werden als Methoden in einer Klasse modelliert und müssen den Stereotyp <<*BusinessOperation*>> haben. Auch hier ist die Modellierung plattformunabhängig, bei der plattformunabhängige Datentypen verwendet werden.

Die modellierten Dienste des Objektverwaltungssystems sind im Anhang E.1 auf Seite 200 als Klassendiagramm verfügbar.

Die in der Phase Schichtentrennung zugeordneten Module zur Schicht Dienste bilden die Basis für die Entscheidung und Modellierung benötigter Dienste. Zum Beispiel ist der Dienst *Member-Verwaltung* vom Modul *Member-Verwaltung* übernommen worden. Auch die zugehörigen Prozeduren (ov_mv_write → setMemberDaten, ov_mv_read → startMV, ov_mv_delete → delMember) aus dem Modul wurden als Dienstoperationen übernommen.

Als allgemeiner Dienst wurde das Modul *Datum&Zeit* übernommen, welches die aktuelle Zeit und das aktuelle Datum liefert. Die Prozedur *ov_protokoll* wird als Dienst *Protokollverwaltung* übernommen und bietet die Operation *addProtokollEintrag* an. Das Modul *INFO-Datei* wird als Dienst *InfoKategorienVerwalten* modelliert. Dieser Dienst liefert nach erreichtem Arbeitsstand vorläufig nur Listen der verschiedenen Kategorien. Diese Listenfunktionen ersetzen zu einem großen Teil das Modul *Listen erzeugen*. Eine Verwaltung, um Kategorien neu zu erstellen oder zu löschen, kann problemlos im Modell realisiert werden. Neu hinzugekommen ist der Dienst *Benutzerverwaltung*. Im aktuellen Arbeitsstand werden nur Funktionen für die Anmeldung und Abmeldung am Objektverwaltungssystem realisiert. Die Ergänzung im Modell und die Implementierung, um Benutzer neu anlegen oder löschen zu können, ist problemlos möglich.

Bei der Modellierung ist zu beachten, dass jeder Dienst Datenbankzugriff hat. Das heißt, nach der Generierung des Quell-Codes besitzt jede Klasse ein Attribut, über das der Datenbankzugriff per Hibernate realisiert wird.

Aus dem Klassendiagramm generiert objectiF automatisch ein Eclipse-Projekt zur Implementierung und Erzeugung von Enterprise Java Beans.

Oberflächennavigation (MyPresentation). In objectiF werden mit Zustandsdiagrammen [UML09] benötigte Oberflächenseiten definiert und die Navigationsreihenfolge zwischen den Oberflächenseiten modelliert. Ein Zustand im Zustandsdiagramm mit dem Stereotyp <<*State*>> definiert jeweils eine JSP/JSF-Seite (*.jsp), diese werden später automatisch generiert. Zustände können auch verfeinert werden und sind dann vom Stereotyp <<*CompositeState*>>. Die Navigation zwischen den Zuständen wird durch Linienzüge (Transition) modelliert. Es können auch Bedingungen sowie Aktivitäten oder Trigger definiert werden. Mit Triggern können mehrere Aktivitäten (z.B.: Dienstoperationen) ausgeführt werden. Diese repräsentieren später auf der JSP/JSF-Seite HTML-Knöpfe. Bei der Quell-Code-Generierung werden in den speziellen JSF-Java-Beans zur Dialogsteuerung Konstrukte zum Aufruf einer Dienstoperation generiert.

Die Modellierung der Oberflächennavigation ist deshalb möglich, weil in der JSF-Technologie Navigationspfade in einer speziellen XML-Datei (faces-config.xml) definiert werden, die von den HTML-Knöpfen zum Aufrufen andere JSP/JSF-Seiten genutzt werden.

Die für das Fallbeispiel zu erstellenden JSP/JSF-Seiten ergeben sich aus den bereits vorhandenen zeichenorientierten Benutzermasken und den ermittelten Modulen aus der Dialogsteuerungsschicht. Zum Beispiel ergibt sich die JSP/JSF-Seite *Member-Verwaltung* aus dem gleichnamigen Modul mit der Prozedur *ov_mv_panel()* bzw. der gleichnamigen ISPF-Panel-Definitionsdatei *OV.PANELS(OV02)*. Neu hinzugekommen ist eine Benutzeranmeldeseite, welche die im Legacy-System verwendete Betriebssystemfunktion der Benutzerberechtigung ablöst und in der JEE-Anwendung die Zugangsberechtigung zum Objektverwaltungssystem regelt.

Benötigte Trigger können aus den Auswahlfunktionen der alten Oberfläche abgeleitet werden. Zum Beispiel kann für M (Modify) Speichern, D (Delete) Löschen oder X (Ende) Beenden verwendet werden. Auch die Navigationsreihenfolge der Bildschirmmasken ist aus dem Legacy-System verwendbar. Diese ist insbesondere durch die Top-Down-Analyse der

Oberfläche und über die Programmablaufpläne, der zur Dialogsteuerungsschicht zugehörigen Prozeduren (z.B.: ov_mv_panel), ermittelbar.

Die Zustandsdiagramme der Oberflächennavigation können im Anhang E.3 ab Seite 202 mit zusätzlichen Erläuterungen eingesehen werden.

Oberflächenentwurf. Nachdem die benötigten JSP/JSF-Seiten definiert und die Navigationsreihenfolge modelliert wurde, kann darauf aufbauend die Oberfläche entworfen werden. Dazu wird die ermittelte Zuordnung der Interaktionsobjekte verwendet, um Ein- und Ausgabedaten auf der graphischen Oberfläche möglichst äquivalent zur zeichenorientierten Benutzeroberfläche darzustellen (Tabelle 5.13, Kapitel 5.3). Anordnung und Aufbau der Interaktionsobjekte orientieren sich an der zeichenorientierten Benutzeroberfläche. Die Benutzeroberfläche wird später mit JSF-Tags und HTML in den definierten JSP-Dateien implementiert. Da die ISPF-Panel-Definitionsdateien die eindeutigen Benutzeroberflächen definieren, wäre eine automatische Transformation zwischen zeichenorientierter ISPF-Vorlage und graphischer Benutzeroberfläche der JSF-Technologie möglich.

In Abbildung 5.24 wird die JSP/JSF-Seite der Benutzeranmeldung dargestellt, wo der Benutzername und das Passwort eingegeben werden muss. Nach der Betätigung des Anmeldeknopfes und der erfolgreichen Überprüfung der Benutzerdaten, wird die Verwaltungskomplexanzeige nach Abbildung 5.25 aufgerufen.

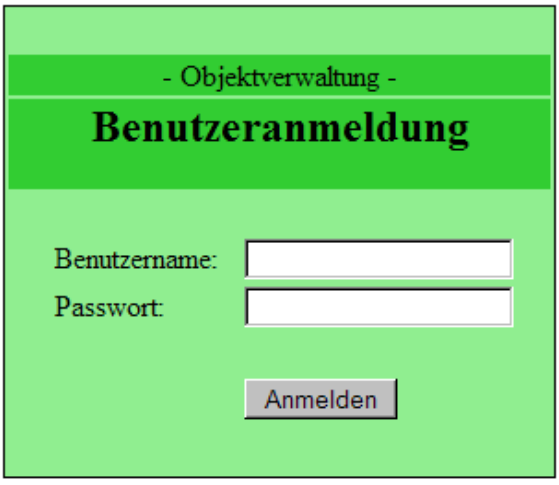


Abbildung 5.24: Fallbeispiel: Benutzeroberfläche – Benutzeranmeldung

Auf der Verwaltungskomplexanzeige werden Member und Änderungskomplexe nicht mehr über die Eingabe eindeutiger Kürzel ausgewählt, sondern alle verfügbaren Member und Änderungskomplexe werden je in einer Listbox zur Auswahl angezeigt. Um einen Member neu anzulegen, muss ein Bezeichner im Textfeld Neu eingegeben werden. Die Abbildung 5.25 setzt die Anforderung *Verwaltungskomplexe anzeigen und auswählen* um.

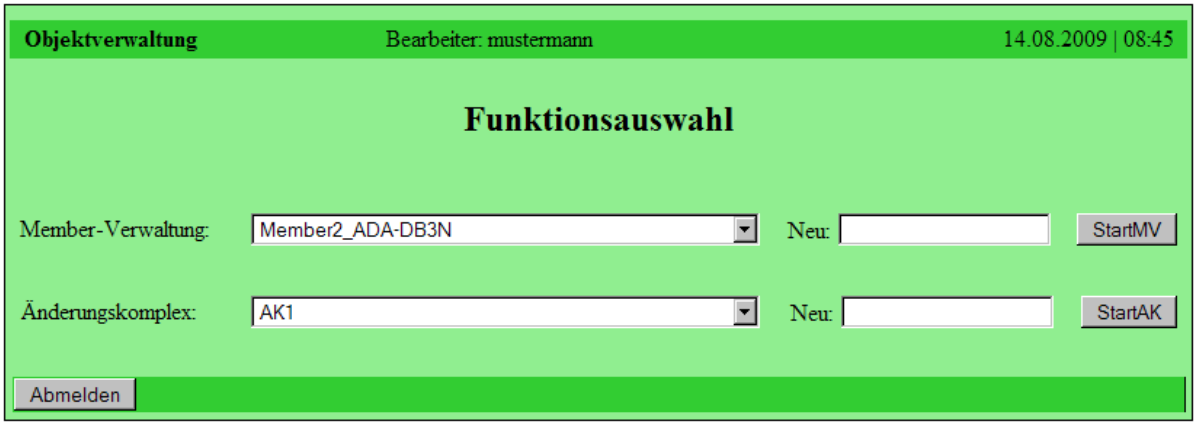


Abbildung 5.25: Fallbeispiel: Benutzeroberfläche – Verwaltungskomplexe

Je nach Auswahl eines Verwaltungskomplexes werden die Benutzeroberflächen nach Abbildung 5.26 (*Member verwalten*) und nach Abbildung 5.27 (*Änderungskomplexe verwalten*) angezeigt. Am Beispiel der Abbildung 5.26 wird gezeigt, dass auch die Kategorieeinträge, wie Sprache oder Typ, in einer Listbox zur Auswahl stehen und nicht mehr über die Eingabe eindeutiger Kürzel ausgewählt werden müssen.

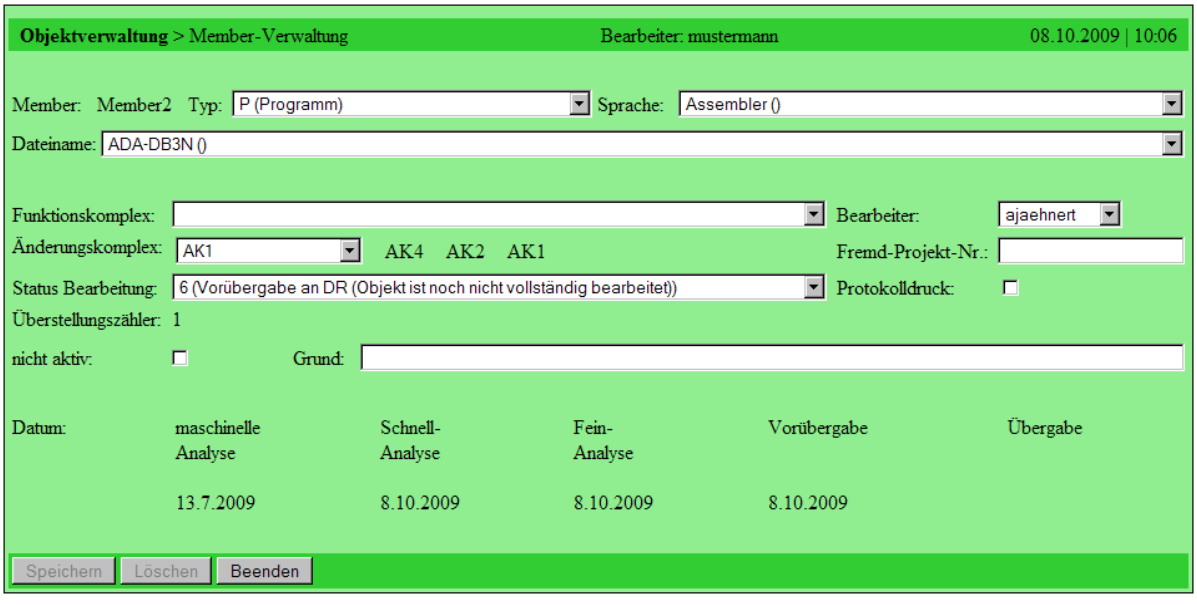


Abbildung 5.26: Fallbeispiel: Benutzeroberfläche – Member-Verwaltung

Die Abbildungen 5.26 und 5.27 zeigen unter anderem die Umsetzung der aus den ISPF-Panel-Definitionsdateien rekonstruierten Interaktionsobjekte. Es konnten Eingabe- und Ausgabeelemente identifiziert und modelliert werden. Zum Beispiel ist die Länge des Beschreibungstextes über ein Änderungskomplex nach Definition variabel und repräsentiert damit eine lange Zeichenkette. Das zugehörige Interaktionsobjekt entspricht somit einem Multiline Text Field und kann in der JSF-Technologie mit dem JSF-Tag <h:inputTextarea> umgesetzt werden.

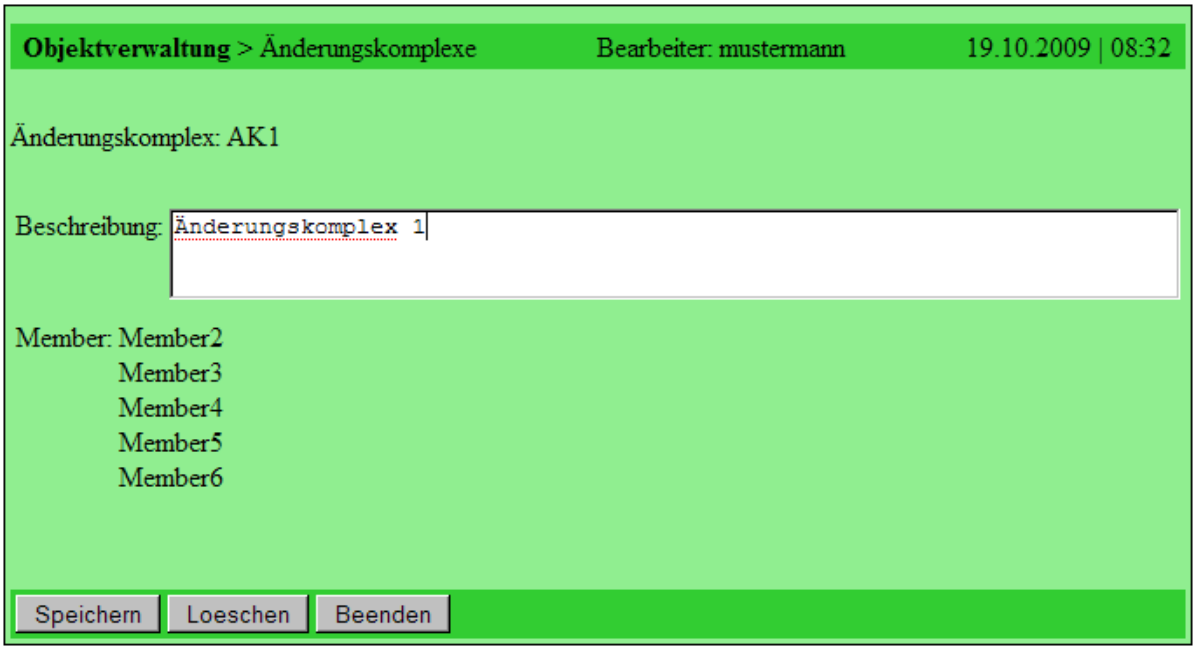


Abbildung 5.27: Fallbeispiel: Benutzeroberfläche – Änderungskomplexe

Realisierung des Prototyps

Quell-Code-Generierung und Implementierung. Nachdem das unabhängige Modell des Zielsystems in objectiF erstellt wurde, werden nun die einzelnen Modelle durch eine Modell-zu-Code-Transformation in Quell-Code-Grundgerüste umgesetzt. Dies geschieht in objectiF automatisch auf Knopfdruck. Es werden folgende Eclipse-Projekte erstellt:

- Datenmodell → Entity Beans (Eclipse-Projekt: MyEntities)
- Dienste → Enterprise Java Beans (EJB) (Eclipse-Projekt: MyServices)
- Oberflächendateien(-Navigation) → JSF-Projekt (u.a. JSP/JSF-Dateien, JSF-Java-Beans für Dialogsteuerung) (Eclipse-Projekt: MyPresentation)

Die Quell-Code-Grundgerüste stellen je nach Projektart Java-Klassen oder JSP-Dateien dar und enthalten die modellierten Eigenschaften. Zum Beispiel werden alle modellierten Dienstoperationen aus dem Modell Dienste zu Methoden einer Java-Klasse umgesetzt. Zu implementieren ist die Logik jeder Dienstoperation im Projekt MyServices und die Dialogsteuerung im Projekt MyPresentation. Der Oberflächenentwurf ist mit den Interaktionsobjekten im Projekt MyPresentation mit HTML und JSF-Tags umzusetzen. Im Projekt MyEntities bedarf es keiner Implementierung, weil dieses Projekt nur die Entitäten als Java-Klassen zur Verfügung stellt.

Auf einzelne Punkte der Implementierung wird nicht weiter eingegangen. Der vollständige Quell-Code bzw. die Eclipse-Projekte befindet sich auf der Begleit-CD [CHRCD09] im Verzeichnis [Quellcode]/[OV_new].

Das modernisierte lauffähige System (Prototyp). Der modernisierte Prototyp des Objektverwaltungssystems setzt die *Benutzeranmeldung* als Teil der Benutzerverwaltung, die Anforderungen *Verwaltungskomplexe anzeigen und auswählen*, *Member verwalten* und *Änderungskomplexe verwalten* um. Die Anforderungen *Benutzer verwalten*, *Informations- und Kategorieeinträge verwalten*, *Dateien verwalten* und *Statistiken, Auswertungs- und Übergabeprotokolle* erstellen wurden im Prototyp nicht umgesetzt.

Um das Objektverwaltungssystem verwenden zu können, muss der vorkonfigurierte JBoss Application Server auf der Begleit-CD [CHRCD09] im Verzeichnis [Software]/OVSYSTEM entpackt und über *jboss-4.0.5.GA\bin\run.bat* gestartet werden. Nach dem Start des Anwendungs-Servers muss in einem Browser folgende URL eingegeben werden:

- http://localhost:8080/OV1 (Benutzername: mustermann, Passwort: 0000)

Tests für den Prototypen. Im Anhang F ab Seite 206 werden für den Prototypen ausgewählte Testfälle und Testszenarien vorgestellt. Zugehörige Testdaten werden über ein SQL-Skript zur Verfügung gestellt, welches sich auf der Begleit-CD [CHRCD09] im Verzeichnis [Quellcode]/[Testdaten] befindet.

5.5 Zusammenfassung

In diesem Kapitel wurde das *4-Phasen-Transformationskonzept* konkret an einem Fallbeispiel angewendet. Es wurde gezeigt, dass das *4-Phasen-Transformationskonzept* durchführbar und geeignet ist. Dabei lässt es genügend Freiräume für Erweiterungen und Ergänzungen. Im Fallbeispiel wurde gezeigt, dass ein Legacy-System in vier Phasen in eine Drei-Schichtige-Zielarchitektur überführt werden kann.

Als Legacy-System wurde das Objektverwaltungssystem der ehemaligen csd Computer-Systemdienste GmbH aus Chemnitz herangezogen, welches aus einem REXX-Programm, aus PL/1-Programmen, aus einer DB2-Datenbank und dem ISPF-Software-Produkt, zur Umsetzung der Benutzeroberfläche bzw. Benutzerschnittstelle, aufgebaut ist.

Als Zielsystem wurde die Java Platform Enterprise Edition (JEE) verwendet, wobei Technologien wie EJB, JSP/JSF und Hibernate zum Einsatz kamen. Mit dem modernisierten Prototypen des Objektverwaltungssystems wurde das Ziel des *4-Phasen-Transformationskonzepts* erreicht.

6 Ergebnisse, Auswertung und Diskussion

In diesem Kapitel werden die Erfahrungen und Ergebnisse der Umsetzung des *4-Phasen-Transformationskonzeptes* und dessen Anwendung am Fallbeispiel ausgewertet und diskutiert. Die Diskussion bewertet und interpretiert zusätzlich die aufgestellten Fragestellungen und Thesen im Kontext der Resultate dieser Arbeit.

Im ersten Teil dieses Kapitels wird das *4-Phasen-Transformationskonzept* ausgewertet und diskutiert. Die Vor- und Nachteile des Konzeptes werden betrachtet und wie das Konzept umgesetzt wurde. Es werden die Erfahrungen der Anwendung des Konzeptes am Fallbeispiel ausgewertet, in dem unter anderem die guten bzw. weniger guten Umsetzungspunkte beispielhaft betrachtet werden.

Im zweiten Teil dieses Kapitels werden die im Konzept vorgestellten Techniken mit ihren Vor- und Nachteilen ausgewertet und diskutiert. Anschließend wird ein konkretes Fazit zu den verwendeten Software-Entwicklungswerkzeugen und ein allgemeines Fazit zur Werkzeugunterstützung im Fallbeispiel gegeben.

6.1 Auswertung des Konzeptes und Diskussion

Phaseneinteilung. Die Phaseneinteilung des Konzeptes ermöglicht ein ausreichendes strukturiertes Vorgehen mit Unterstützung durch Techniken und Vorgängen. Ein Vorteil der Phaseneinteilung ist, dass die Phasen abgeschlossene Einheiten darstellen, welche eine ihr zugewiesene Teilaufgabe bearbeiten. Nur Anhand einer Dokumentenübergabe können die Phasen miteinander kommunizieren. Jede Phase unterliegt der gleichen EVA-Prinzipuntergliederung, wodurch Techniken und Vorgänge einer Phase problemlos erneuert, ergänzt oder ausgetauscht werden können, ohne das das Gesamtkonzept zerstört wird. Im Prinzip können neue Phasen, welche das Konzept in Art und Umfang sinnvoll ergänzen und den Modernisierungsprozess vereinfachen bzw. erleichtern, als Einheiten zwischen den Phasen integriert werden. Eine Prüfung der Integrität des Konzeptes wäre dann notwendig.

Eine mögliche Verbesserung der Phaseneinteilung könnte erreicht werden, wenn die komplexe Phase Analyse in kleinere Phasen unterteilt wird. Zum Beispiel könnte man sie in die Phasen Design Recovery und Redokumentation zerlegen. Es ist auch möglich eine separate Phase Komponentenfindung einzufügen, welche aber eine Teilung der Phase Design Recovery zur Folge hätte. Auch Teilaufgaben, die zur Zeit noch an ergänzende Konzepte vergeben werden, können als separate Phasen in das Konzept integriert werden. Zum Beispiel kann die umfangreiche Teilaufgabe der Informationswiedergewinnung von Meta-Informationen aus Datenbanken als Phase integriert werden.

Kausalität bei Änderung der Ergebnisse einer Phase. Das Pipeline-Prinzip legt im Konzept die Abarbeitungsrichtung der Phasen fest. Sind durch Erkenntnisvorgänge Änderungen im Ergebnis einer abgearbeiteten Phase notwendig, dann müssen alle nachfolgenden Phasen komplett neu durchlaufen werden, auch wenn nachfolgende Phasen bereits abgearbeitet wurden. Denn eine Änderung der Ergebnisse einer Phase kann starke Änderungen in den Ergebnissen der nachfolgenden Phasen nach sich ziehen. Mit der beschriebenen Kausalität wird auch die Konsequenz von möglichen Sprüngen zwischen den Phasen verdeutlicht. Zum Beispiel gehört zu den Sprüngen das Überspringen von Phasen, wobei solch ein Vorgang voraussetzt, dass ein Ergebnis einer ausscheidenden Phase in den nachfolgenden Phasen nicht zwingend benötigt wird.

Die durch Sprünge zwischen den Phasen hervorgerufene erneute Abarbeitung von Phasen ist nachteilig für das Konzept, weil der Arbeits- und Zeitaufwand steigen kann.

Gut und schlecht geeignete Legacy-Systeme für das Konzept. Das wichtigste Maß für die Konzepteignung ist die Transformationsfähigkeit eines Legacy-Systems. Die Transformationsfähigkeit sagt aus, wie gut ein Legacy-System für eine Transformation in ein Zielsystem geeignet ist. Eine gute Transformationsfähigkeit beinhaltet, dass man aus dem Legacy-System alle für ein Zielsystem benötigten Informationen ermitteln und mit diesen Informationen die Funktionalitäten des Legacy-Systems im Zielsystem umsetzen kann. Die Transformationsfähigkeit beinhaltet die verschiedensten Kriterien, wie zum Beispiel die Zerlegbarkeit, der Aufbau und die Architektur eines Legacy-Systems und die daraus ermittelbaren Abhängigkeiten, Art der verwendeten Programmiersprache mit dem Informationsgehalt des Sprachumfangs, der individuelle Programmierstil eines Entwicklers und die Aussagefähigkeit der Quell-Code-Bezeichner und Quell-Code-Kommentare. Es sind auch Kriterien wichtig, die beurteilen wie gut oder wie schlecht aus den Informationen eines Legacy-Systems Objekte in einer objektorientierten Programmiersprache gebildet werden können. Unter anderem diese Kriterien entscheiden ob ein Legacy-System für dieses Konzept geeignet ist.

Betrachtungen zur Transformationsfähigkeit von Legacy-Systemen bzw. die Berücksichtigung der Transformationsfähigkeit bei zukünftigen Software-Systemen ist ein Thema, welches in zukünftigen Arbeiten weiter zu untersuchen ist und deshalb als weiterführendes Thema im Kapitel 7.2 ausgewiesen wird.

Werden einzelne Kriterien der Transformationsfähigkeit betrachtet, dann eignen sich für dieses Konzept zerlegbare Legacy-Systeme gut, weil bereits wohl definierte Strukturen vorhanden sind. Zudem können Komponenten und Teilsysteme leicht ermittelt und für eine Wiederverwendung herangezogen werden. Auch Abhängigkeiten sind durch definierte Schnittstellen leicht erkennbar.

Unzerlegbare Legacy-Systeme mit monolithischen Eigenschaften sind dagegen ungeeignet für das Konzept, weil Komponenten und Teilsysteme schwer zu ermitteln sind bzw. weil es keine erkennbaren Strukturen oder Architekturmuster gibt. Die Aufwände zur Erkennung von Strukturen erhöhen sich dadurch sehr stark und können dazu führen, dass nur noch eine

komplette Neuentwicklung des betrachteten Legacy-Systems in Frage kommt. Eine Aussage über die Unzerlegbarkeit von Legacy-Systemen ist schon dann möglich, wenn der Dominanz-Baum nur aus dem Wurzelknoten und einer anschließenden Ebene von Kindknoten besteht, welche keine weiteren Kindknoten haben. Die Phase Redesign kann deshalb allein mit dieser Technik zum Misserfolg führen, weil die Dominanz-Analyse eine schlechte Gruppenbildung bei hohen Abhängigkeiten liefert. Durch den kreativen Freiraum der Begriffsanalyse können auch Legacy-Systeme mit monolithischen Eigenschaften zerlegt werden, was für die Durchführung der Phase Redesign und Schichtentrennung vorteilhaft ist. Eine Konsequenz bei dieser Vorgehensweise ist, dass die vielen Abhängigkeiten zwischen den zerlegten Teilen eines Systems weiterbestehen.

Andere schlechte Voraussetzungen für das Konzept sind nicht aussagefähige bzw. fehlende Quell-Code-Bezeichner (z.B.: Variablen- und Prozedurnamen) und Quell-Code-Kommentare. Solche Semantik-Schwächen müssen durch Software-Dokumente oder Mitarbeiterauskünfte kompensiert werden. Ist das nicht möglich, dann können unter anderem Funktionalitäten nicht identifiziert werden, was bereits eine Störung im Erkenntnisprozess ab der Phase Analyse zur Folge hätte. Ein weiterer schlechter Punkt für das Konzept kann die Verwendung von Programmiersprachen sein, welche im Sprachumfang keine explizite Deklaration von Variablen vorsehen. Zum Beispiel wird die Erkennung von Daten mit REXX erschwert, weil in REXX Variablen deklarationslos und global verwendet werden.

Auswirkungen der Architekturorientierung. Die Orientierung des Konzeptes im Architekturbereich hat den Vorteil, dass von Anfang an die Bemühung zur Zerlegung des Legacy-Systems besteht. Je nach Zerlegbarkeit eines Legacy-Systems wird ermittelt, aus welchen Komponenten, Modulen bzw. Teilen das Legacy-System aufgebaut ist und welche Abhängigkeiten gebildet werden. Nach der Wiedergewinnung der Entwurfsinformationen, kann sich ein Entwickler durch dieses Vorgehen schnell einen Überblick auf Entwurfsebene verschaffen. Es können identifizierte funktionale Einheiten gezielt untersucht werden, um unter anderem Ablauf- bzw. Fachlogiken für konkrete Artefakte zu ermitteln. Je nach Anforderung können die Teile eines Legacy-Systems anhand eines aufgestellten Kriterienkatalogs bewertet und bei Wiederverwendung im Zielsystem berücksichtigt werden.

Das Festlegen der Drei-Schichten-Architektur für das Zielsystem, zog in der Entwicklung des Konzeptes die teilweise Spezialisierung von Phasen nach sich. Zum Beispiel ist die Phase Schichtentrennung darauf spezialisiert, Komponenten, Module bzw. Artefakte zu den logischen Schichten der Drei-Schichten-Architektur einzuteilen. Wenn eine andere Zielarchitektur umgesetzt werden soll, dann erschwert diese Spezialisierung eine Umstellung des Konzeptes.

Eine Umstellung auf N-Schichten-Architekturen (Mehr-Schichten-Architekturen) ist ausgehend von der festgelegten Drei-Schichten-Architektur möglich. Dahingehend ist eine Anpassung der Zielsystemeigenschaften und der Phasen Schichtentrennung und Forward Engineering notwendig. Die Anpassung der Phase Schichtentrennung würde die Zerlegung des Legacy-Systems in N-Schichten voraussetzen. In der Phase Forward Engineering muss beim Entwurf die geänderte Zielarchitektur berücksichtigt werden. Alle die im Konzept genannten Plattformen unterstützen N-Schichten-Architekturen.

Die Umstellung des Konzeptes auf eine andere Architektur, zum Beispiel auf eine serviceorientierte Architektur (SOA), ist problematisch. Die Orientierung des Konzeptes müsste auf die Erkennung von Services (Dienste) spezialisiert sein. Eine allgemeine Methode zur Erkennung von Diensten bietet die Firma IBM mit SOMA (Service-oriented modeling and architecture) [SOMA]. Die Methode beinhaltet keine Beschreibung von Techniken, wie das im *4-Phasen-Transformationskonzept* umgesetzt wurde.

Wahl der Zieltechnologie. Die festgelegten Zielsystemeigenschaften sind technologieunabhängig, wodurch die Wahl einer Zieltechnologie je nach Anforderung frei wählbar ist. Im Konzept wurden zwei mögliche Zielplattformen vorgestellt und im Fallbeispiel wurde eine davon verwendet. Die Wahl anderer Technologien ist im Sinne des Konzeptes möglich. Zu beachten sind die Entwurfsentscheidungen in der Phase Forward Engineering in Abstimmung mit den Eigenschaften der gewählten Technologie.

Maß an Neuentwicklung oder Migration. Das Maß an Neuentwicklung oder Wiederverwendung durch Migration ist von Legacy-System zu Legacy-System unterschiedlich und von den verschiedensten Kriterien abhängig. Die Entscheidung, wann neu entwickelt oder durch Migration wiederverwendet werden soll, muss durch separate Kriterienkataloge für

Anforderungen und die unterschiedlichsten Artefakte entscheidbar gemacht werden. Zum Beispiel kann eine Wiederverwendung durch Migration für Komponenten aus einem Legacy-System in Betracht gezogen werden, wenn sich eine Komponente sinnvoll in die Zieltechnologie integrieren lässt (z.B.: Version, Schnittstellen, Standards) und den verschiedensten Software-Qualitätsmerkmalen (z.B.: Integrationsfähigkeit) entspricht. Neben den Komponenten unterstützt und ermöglicht das Konzept die Wiederverwendung von Teilen eines Legacy-Systems (z.B.: Module) durch Migration. Zum Beispiel werden Teile eines Legacy-Systems als Module aufgebaut und sichtbar gemacht und können anhand eines Kriterienkatalogs in der Phase Forward Engineering zur Wiederverwendung berücksichtigt werden.

Je älter die Technologie eines Legacy-Systems ist, umso wahrscheinlicher ist die Neuentwicklung in Kombination mit einer funktionalen Migration. Denn die Betrachtungen beziehen sich auf Legacy-Systeme, welche allen Wartungs- bzw. Pflegemaßnahmen widerstehen und möglicherweise auf Betriebsplattformen laufen, die obsolet sind. Eine Neuentwicklung ist im Sinne des soziotechnischen Systems schon da notwendig, wo betriebsplattformspezifische Eigenschaften nicht mehr auf der Zielplattform vorhanden sind. Die Zerlegbarkeit des Legacy-Systems ist auch ein Grund für eine Neuentwicklung. Je mehr Abhängigkeiten vorherrschen, umso schwieriger lassen sich Teile eines Legacy-Systems mit expliziten Schnittstellen isolieren.

Anwendungseinschätzung des Konzeptes bei großen Modernisierungsprojekten. Die Anwendung des Konzeptes bei großen Modernisierungsprojekten ist nur durch die Unterstützung von Werkzeugen praktisch und durchführbar. Bei großen Modernisierungsprojekten können Legacy-Systeme mehrere tausend bis millionen Quell-Code-Zeilen aufweisen und aus vielen hundert Prozeduren bzw. Funktionen und oder Modulen bestehen. Im Fallbeispiel war schon allein die Erkenntnisgewinnung in der Phase Analyse ohne Werkzeuge sehr aufwendig. Deshalb ist es für große Projekte erforderlich, dass Werkzeuge für jede Phase im Konzept sondiert werden und bereitstehen.

Grad der Automatisierung. Der systematische Einsatz von Techniken ist mit der Phaseneinteilung durch spezialisierte Teilaufgaben gegeben. Um den zeitlichen Aufwand einer Modernisierung zu reduzieren, ist der Grad der Automatisierung wichtig, welcher von

automatischen Analysen und der Ergebnisauswertung bzw. Ergebnisverwertung durch einen Entwickler geprägt ist. Das heißt, die beschriebenen Techniken einer Phase können automatisiert Ergebnisse liefern, welche Entwickler als Entscheidungshilfe nutzen und wiederum für anschließende automatische Analysen verwenden können. Die Automatisierung als Entscheidungshilfe hat den Vorteil, dass Entwickler die Entscheidungsträger für Kriterien sind und viel besser bei unbefriedigenden Ergebnissen agieren können, als bei vollautomatischen feststehenden Transformationen. Dieses Vorgehen begünstigt Legacy-Systeme, die schlechte Voraussetzungen für eine vollautomatische Transformation mitbringen. Bei einer vollautomatischen Transformation muss ein Legacy-System ein gewissen Grad an Transformationsfähigkeit aufweisen (z.B. strukturierter Aufbau, erkennbare Architektur, Software-Qualität, Quell-Code-Bezeichner, usw.), damit alle benötigten Informationen, die zum Aufbau eines Zielsystems benötigt werden, automatisch ermittelt und aufbereitet werden können. Sobald aber solche Information nicht mehr eindeutig aus statischen Analysen ermittelt werden können, wird ein kreativer Aspekt benötigt. Dieser kreative Aspekt ist durch die aktive Auswertung bzw. Entscheidung eines Entwicklers gegeben. Zum Beispiel kann die Dominanz-Analyse in kürzester Zeit eine Modularisierung, bei einem angenommenen gut aufgebauten Legacy-System, automatisch durchführen. Doch weist das Legacy-System, wie im Fallbeispiel, hohe Abhängigkeiten auf, dann würde bei einer vollständigen Automatisierung die Modularisierung keine ausreichenden Ergebnisse liefern. An dieser Stelle muss ein Entwickler eingreifen, der entweder zusätzliche Techniken einsetzt oder die Analyseergebnisse durch Änderung einer anderen Sichtweise verbessert. Mit Änderung von Sichtweisen ist zum Beispiel die Begriffsanalyse gemeint, mit der man individuelle Objekte und Eigenschaften für eine Modularisierung wählen kann.

Wichtig für die Reduzierung des Zeitaufwandes einer Modernisierung und damit der Einsparung von Kosten, sind vorhandene Werkzeuge, welche dem Entwickler in jeder Phase schnell eine Analyse- bzw. Entscheidungshilfe geben.

Legacy-System-Vorbeugung. Die Verhinderung, dass nach der Modernisierung ein Software-System wieder zu einem Legacy-System wird, ist nicht hundertprozentig möglich. Durch den schnellen Entwicklungsprozess der IT-Technologien werden jetzt noch dem Stand der Technik entsprechende Technologien mit der Zeit automatisch obsolet. Deshalb besteht die

Aufgabe bei einer Entwicklung bzw. Modernisierung eines Software-Systems, die Transformationsfähigkeit zu maximieren und die Dokumentationen stets zu aktualisieren. Das beinhaltet, dass das Software-System bis zur endgültigen Stilllegung oder erneuten Modernisierung gut wartbar bleibt und im Rahmen sich ständig ändernder Anforderungen gut erweitern und ändern lässt.

Eine gute Transformationsfähigkeit kann man mit der MDA-Strategie erreichen, welche im Konzept angewendet wurde. Mit der MDA-Strategie wird das Risiko vermindert, dass ein Software-System wieder zu einem Legacy-System wird.

Die MDA-Strategie verfolgt unter anderem folgende Ziele, welche den Charakteristika eines Legacy-Systems entgegenwirken:

- Verbesserung der Wartbarkeit
- Handhabbarkeit von Komplexität durch Abstraktion
- Steigerung der Entwicklungsgeschwindigkeit
- Vermeidung von Redundanz
- Portabilität
- Verbesserung der Dokumentation
- Wiederverwendbarkeit

Die Möglichkeit der Risikoverminderung ist nur gegeben, wenn auch die Entwicklung und Wartung ausgehend vom Modell durchgeführt wird. Jedoch nicht wenn Änderungen im Quell-Code direkt vorgenommen werden, ohne die nachfolgende Ergänzung im Modell. Durch die klare Trennung von Abstraktionsschichten, welche unter anderem die Wiederverwendbarkeit und die Langlebigkeit der Modelle begünstigt, sind Änderungen von Entwicklungs- und Wartungsarbeiten im Modell dokumentiert. Dieser Vorteil begünstigt das Einarbeiten von Mitarbeitern in fremde Software-Systeme. Die Fachlichkeit einer Domäne ist in den Modellen konserviert und für jeden nachvollziehbar. Die Möglichkeit der Transformation, zwischen den Modellen der MDA-Strategie, begünstigt zusätzlich die Umstellung zu verschiedenen Zielarchitekturen (z.B.: JEE, .NET) oder Zielsprachen (z.B.: Java, C#).

Inhalte zur Vorbeugung, dass Software-Systeme nicht wieder zu Legacy-Systeme werden, erforscht das Fraunhofer-Institut für Software- und Systemtechnik mit Continuous Software Engineering [FH09]. Continuous Software Engineering ist eine effiziente Methode und Konstruktionslehre für evolutionsfähige IT-Landschaften. Die Forschungen umfassen unter anderem die Erforschung von Software-Systemen, die den sich verändernden Anforderungen gerecht werden und die Erforschung von effizienten Techniken für die kontinuierliche Wartung und Weiterentwicklung von Software-Systemen.

In Tabelle 6.1 werden die wesentlichsten Vor- und Nachteil des Konzeptes aufgelistet.

Tabelle 6.1: Auswertung 4-Phasen-Transformationskonzept: Vor- und Nachteile

Vorteile	**Nachteile**
Es werden konkrete Techniken und Vorgänge, welche die Modernisierung unterstützen, im Konzept angewendet.	Der Aufwand des Konzeptes steigt, je weniger Software-Dokumente das Legacy-System dokumentieren.
Eine konkrete Zieltechnologie bzw. Zielplattform ist je nach Anforderung frei wählbar.	Die Kausalität der Phasen kann den Aufwand bzw. den Zeitfaktor zur Durchführung des Konzeptes erhöhen.
Das Konzept legt nicht den Zeitpunkt und eine mögliche Reihenfolge zur Übernahme der Daten und Teile des Legacy-Systems fest, dieser Aspekt wird durch Migrationsstrategien als ergänzende Konzepte bereitgestellt.	
Vorteil: Eine Migrationsstrategie kann je nach Anforderung frei gewählt werden.	Nachteil: Die Überlegungen werden erst in der Phase Forward Engineering durch externe Migrationsstrategien berücksichtigt.
Das Konzept ist stark an den Zielsystemeigenschaften orientiert, hauptsächlich im Architekturbereich.	
Vorteil: Die Spezialisierung ermöglicht die optimale Ausrichtung des Konzeptes auf eine konkrete Zielarchitektur mit deren speziellen Eigenschaften.	Nachteil: Der Wechsel zu nicht auf Basis von N-Schichten aufgebauten Zielarchitekturen ist schwierig, weil die Anpassung der gesamten Konzeptorientierung notwendig ist.
Das Konzept ist technologieunabhängig, es wird lediglich ein Technologierahmen vorgegeben .	
Das Konzept unterstützt aktuelle Technologien im Zielsystem.	
Das Konzept unterstützt Wiederverwendung durch Migration (z.B.: Anforderungen).	

▼

Vorteile	Nachteile
Das Konzept unterstützt und vermittelt externe spezialisierte Konzepte zur Bearbeitung größerer Teilaufgaben.	
Der kreative Freiraum des Konzeptes ermöglicht eine 4-Phasen-Transformation von Legacy-Systemen mit monolithischen Eigenschaften.	
Der Umgang mit Komplexität wird durch die Problemteilung (Phasen, externe Konzepte) erleichtert.	

6.2 Auswertung des Fallbeispiels

Die Anwendung des Konzeptes konnte mit dem Fallbeispiel gut gezeigt werden. Es konnte die Erkenntnis gewonnen werden, dass selbst ein kleines Beispiel einen hohen Aufwand nach sich ziehen und die Anwendung des Konzeptes bei großen Legacy-Systemen, wie bereits dargestellt, nur durch Werkzeugunterstützung bewältigt werden kann.

Mit dem Fallbeispiel wurde auch deutlich, dass das Reverse Engineering in der Phase Analyse sehr aufwändig ist. Das Legacy-System muss vollständig verstanden werden, was eine aufwändige Bearbeitung von zum Teil unvollständigen textbasierenden Software-Dokumenten nach sich zieht. Diese Unvollständigkeit wird immer erst dann aufgedeckt, wenn geschriebenes nicht mehr im laufendem System bzw. Quell-Code nachweisbar ist oder umgekehrt. Deshalb ist zur Informationsgewinnung die häufige Konsultation der Mitarbeiter mit Wissen über das Legacy-System wichtig.

Der Aufwand der Informationswiedergewinnung und der Informationsaufbereitung kann unter Umständen größer werden, als bei einer normalen Software-Entwicklungstätigkeit im Rahmen einer kompletten Neuentwicklung. Deshalb ist im Vorfeld zu untersuchen, ob sich der Aufwand, ein Software Reengineering an einem Legacy-System durchzuführen, lohnt. Eine Vorfelduntersuchung wurde bereits mit dem SRA-Prozess [DoD97] vorgestellt, welche bei einer Erweiterung des Konzeptes als Phase vor der Phase Analyse realisiert werden kann.

Der Aufwand zur Abarbeitung der Phasen stieg im Fallbeispiel durch das Nichtvorhandensein von Werkzeugen, welche automatische statistische Quell-Code-Analysen an REXX-Programmen ermöglicht hätten. Mit einem Werkzeug hätte man Aufrufgraphen, Kontrollflussgraphen und Strukturdiagramme aus dem Quell-Code in Kürze automatisch aufbauen und anschließend auswerten können.

Die Informationsrückgewinnung für prozedurale Programmiersprachen hat sich in der Phase Analyse, mit den beschriebenen Techniken, als ausreichend herausgestellt. Das beinhaltet unter anderem die Darstellung der Abhängigkeiten auf verschiedenen Ebenen (z.B.: Dateien, Modulen, Prozeduren), welche durch die verschiedenen Kontrollflussgraphentypen abgedeckt werden.

Auf der Ebene von Prozeduren konnten Ablauflogiken und Algorithmen ermittelt werden, was aber eine Identifizierung der Funktionalität vorausgesetzt hat. Als wichtige Informationsträger zur Identifizierung haben sich im Quell-Code die Namensgebungen (z.B.: von Variablen, Prozeduren, usw.) und die Quell-Code-Kommentare bewährt. Auch Prozeduren, die funktionale Einheiten bilden, konnten durch die Namensgebung leichter erkannt werden.

Die Rückgewinnung der Anforderungen des Legacy-Systems unterlagen einem kreativen Prozess. Ansatzpunkte zur Identifizierung von Anforderungen waren vor allem Funktionalitäten im Kontext des Legacy-Systems, welche gut über ein UML-Werkzeug wie objectiF redokumentiert werden konnten.

Eine aus dem Fallbeispiel gewonnene Erkenntnis ist, dass die Redokumentation aus der Phase Analyse extrahiert und als separate Phase parallel zu den anderen Phasen des Konzeptes laufen sollte (Abbildung 6.1). Denn die Redokumentation ist phasenübergreifend und nicht allein Aufgabe der Phase Analyse bzw. wird nicht in der Phase Analyse abgeschlossen.

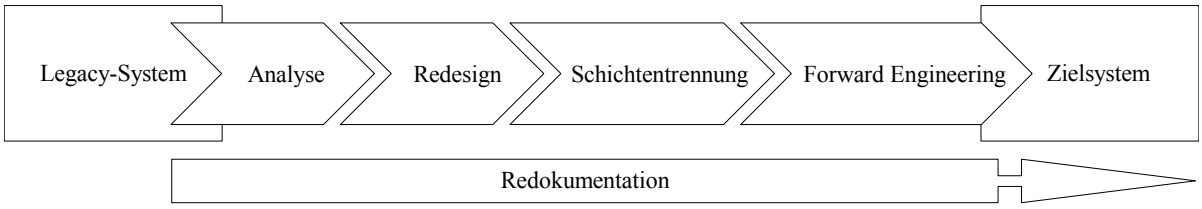


Abbildung 6.1: 4-Phasen-Transformationskonzept – Separierung der Redokumentation

In der aktuellen Konzeptversion werden die Software-Dokumente der Redokumentation durch die Phasen gereicht und bei einer Neuerkenntnis ergänzt und erweitert, was ein Neudurchlaufen durch das Pipeline-Prinzip ab der Phase Analyse verhindert. Die Redokumentation wird in der Phase Analyse initialisiert und über die Phasen vervollständigt.

Die Erkennung von Komponenten im Fallbeispiel war eindeutig, weil unter anderem Dateien der verwendeten Komponenten als Quell-Code vorlagen (z.B.: ISPF-Panel-Definitionsdateien, SQL-Anweisungsdateien). Die Zerlegung des REXX-Programms war schwieriger, weil starke Abhängigkeiten zwischen den Prozeduren und somit monolithische Eigenschaften vorherrschten. Die starken Abhängigkeiten konnten bereits im Dominanz-Baum erkannt werden. Mit der Dominanz-Analyse wurden einige Module gebildet, doch über die Prozedurbezeichnungen war erkennbar, dass die Module nicht vollständig zusammengefasst wurden. An dieser Stelle stand die Entscheidung fest, dass im Zuge der Modernisierung nur eine funktionale Migration mit Neuentwicklung in Frage kommen kann. Um aber eine bessere Basis für einen Entwurf in der Phase Forward Engineering zu haben, wurden die Prozeduren mittels Begriffsanalyse zu funktionalen Einheiten zusammengefasst, auch wenn dadurch die Abhängigkeiten zwischen den Modulen stark erhalten blieben und es keine eindeutigen Schnittstellen gab. Der Technologiestand des Legacy-Systems bot für eine Wiederverwendung keine geeignete Basis, deshalb war dieses Vorgehen vorteilhaft, um wenigstens eine gezielte Klassen- bzw. Objektfindung zu ermöglichen.

In der Phase Schichtentrennung wurden anhand der Begriffsanalyse die Module in die geforderten Schichten der Zielarchitektur eingeteilt, was eine Sichtweise auf Funktionalitätsgebiete ermöglichte. Zum Beispiel konnten für die Fachlogik/Dienste-Schicht im Zielsystem Objekte mit zugehörigen Methoden identifiziert werden.

Die Phase Forward Engineering konnte ohne Probleme durchgeführt werden, weil in den vorangegangenen Phasen benötigte Informationen zur Umsetzung des Zielsystems gesammelt wurden und die Zielsystemeigenschaften festgelegt waren. Im Fallbeispiel konnten die identifizierten funktionalen Anforderungen übernommen und neue hinzugefügt werden. Die Wiederverwendung von Komponenten und Modulen war wegen der Abhängigkeiten und dem alten Technologiestand nicht sinnvoll gewesen. Die verwendete Zielplattform und die

benötigten Entwicklungswerkzeuge wurden im Konzept vorgeschlagen und konnten im Fallbeispiel angewendet werden.

Umfangvergleich zwischen Legacy-System und Zielsystem

Für einen Umfangvergleich zwischen Legacy-System und Zielsystem wird die Software-Metrik LOC herangezogen. Bei dem Vergleich werden auf der Seite des Zielsystems benötigte Meta-Dateien für JEE-Projekte bzw. JEE-Pakete nicht berücksichtigt. Es werden auch allgemein für den Betrieb benötigte Pakete, Bibliotheken usw. nicht betrachtet.

In der Tabelle 6.2 wird der Umfang des Zielsystems (Prototyp) dargestellt. Für jede Schicht wird die Anzahl der generierten Dateien und der Anteil der generierten und implementierten Quell-Code-Zeilen aufgelistet. Der gesamte generierte Quell-Code-Anteil, in Abhängigkeit der im Prototyp umgesetzten Funktionalitäten, entspricht ca. 65%.

Zielsystem	Anzahl der Dateien & LOC	LOC gesamt
MyEntities	14 Java-Klassen: LOC generiert: 1677	1677
MyServices	15 Java-Klassen: LOC generiert: 638, LOC impl.: 498	1136
MyPresentation	15 Java-Klassen: LOC generiert: 1476, LOC impl.: 1513 16 JSP-Dateien: LOC generiert: 352, LOC impl.: 241	2989 593
	LOC gesamt	**6395**
	davon LOC generiert	4143 (64,78%)
	davon LOC implementiert	2252 (35,22%)

Tabelle 6.2: Auswertung Fallbeispiel: Zielsystem LOC-Metrik (Prototyp)

In der Tabelle 6.3 wird der Umfang des Legacy-Systems dargestellt. Es werden nur die relevanten Quell-Code-Anteile bzw. PL/1-Programme und ISPF-Panel-Definitionsdateien berücksichtigt, welche Anforderungen äquivalent zum Prototypen umsetzen.

Legacy-System	Anzahl der Dateien & LOC	LOC gesamt
OV.REXX	1 REXX-Programm, LOC: ca. Prototypumsetzung: 2776	2776
OV.PANELS(...)	6 ISPF-Panel-Definitionsdateien, LOC: 657	657
OV.PLI(...)	2 PL/1-Programme, LOC: 821	821
	LOC gesamt	**4254**

Tabelle 6.3: Auswertung Fallbeispiel: Legacy-System LOC-Metrik (Prototyp relevant)

Es ist zu erkennen, dass die Dateianzahl im Zielsystem größer ist, als die im Legacy-System. Dieser Unterschied kommt unter anderem durch die unterschiedlichen Technologien, Programmierparadigmen und Architektureigenschaften zustande. Zum Beispiel unterliegt das Zielsystem dem objektorientierten Paradigma, wodurch die Dateianzahl schon durch die Definition von mehreren Objekten bzw. Klassen steigt.

Im direkten Vergleich ist die Gesamtanzahl an Quell-Code-Zeilen beim Zielsystem größer, weil unter anderem zusätzliche Hilfsklassen generiert und in Java, im Gegensatz zu REXX, die Variablen deklariert werden. Die Programmiersprache Java und die JEE-Technologie verlangen mehr Verwaltungsdaten (overhead). Auch die umständliche Typenkonvertierung in Java ist für eine steigende Quell-Code-Zeilen-Anzahl zu nennen. Der größte Quell-Code-Anteil im Zielsystem geht von den Dialogsteuerungsklassen der JSF-Technologie aus, weil ein erhöhter Aufwand zur Umsetzung der Kommunikation zwischen Oberfläche, Dialogsteuerung und Daten (Session-Beans[50]) erforderlich ist.

Im Resümee ist der implementierte Quell-Code-Anteil im Zielsystem viel geringer als im Legacy-System. Der größte Anteil an Quell-Code wird im Zielsystem automatisch generiert. Dadurch ist im Wesentlichen nur noch die Fachlogik zu implementieren. Die Gesamtanzahl an Quell-Code-Zeilen im Zielsystem ist gegenüber dem Legacy-System unbedeutend. Viel wichtiger sind die Vorteile des Zielsystem. Gegenüber dem Legacy-System sind im Zielsystem getrennte Schichten vorhanden, welche eine problemlose Einzelwartung und Erweiterung ermöglichen. Nach Bedarf können unterschiedliche Benutzeroberflächen und Datenbankverwaltungssysteme angebunden werden. Neben diesen Aspekten sind mit dem Zielsystem wohl definierte Schnittstellen vorhanden, die eine bessere Integration oder Anbindung an andere Systeme ermöglichen.

50 Session Beans sind Enterprise Java Beans (EJB), welche speziell Vorgänge abbilden. Session Beans verwenden Entity Beans, um einen Zustand für eine Sitzung darzustellen.

6.3 Beurteilung der Techniken und Diskussion

Kontrollflussgraphen. Es hat sich gezeigt, dass Kontrollflussgraphen gut für die Darstellung von Aufrufhierarchien geeignet sind. Diese können automatisch aus Quell-Code aufgebaut und für automatische Analysen verwendet werden. Sie werden unter anderem von der Dominanz-Analyse oder von dem MORPH-Prozess als Analysebasis genutzt.

Dominanz-Analyse. Die Dominanz-Analyse, als statisches Quell-Code-Analyse-Verfahren zur Modularisierung in der Phase Redesign, hat den Vorteil, dass es automatisierbar ist. Zur Analyse wird nur ein Aufrufgraph oder ein Kontrollflussgraph mit einem ausgezeichneten Wurzelknoten vorausgesetzt. Das Verfahren ermöglicht die Modul- bzw. Gruppenbildung für eine sehr große Zahl an Artefakten und ist damit für große Projekte geeignet.

Anhand eines Dominanz-Baumes ist erkennbar, wie stark die Abhängigkeiten zwischen den Artefakten sind. Je flacher ein Dominanz-Baum ist, desto stärker sind die Abhängigkeiten zwischen den Artefakten. So eine Aussage kann bei einer Modernisierung die Relevanz zwischen Neuentwicklung und Migration verschieben. Zu stark abhängige Artefakte lassen sich schwer zerlegen bzw. die Aufwände die Abhängigkeiten zu verstehen erhöhen sich, was zum Beispiel eine Entscheidungsrichtung für eine Neuentwicklung sein kann.

Da die Gruppenbildung auf Basis eines Aufrufgraphen umgesetzt wird, ist eine starke Abhängigkeit zwischen den betrachteten Artefakten schlecht für eine Gruppenbildung. Denn Artefakte, welche von mehreren anderen Artefakten aufgerufen werden, gelten nach der Definition als allgemeine Dienste mit einer normalen Dominanz-Relation und können nicht unmittelbar zu Gruppen zusammengefasst werden.

Was die Dominanz-Analyse nicht umsetzen kann, ist die Einordnung bzw. die Bedeutungszuordnung der ermittelten Module bzw. Gruppen im Kontext des Legacy-Systems. Diese Aufgabe unterliegt weiterhin dem Entwickler.

Die Vor- und Nachteile der Dominanz-Analyse wurden in der Tabelle 6.4 zusammengefasst.

Vorteile	Nachteile
Eine Zerlegbarkeitsaussage über die betrachteten Artefakte ist möglich.	Die automatische Bedeutungszuordnung der Gruppen im Kontext des Legacy-Systems ist nicht möglich.
Das Verfahren ist für eine sehr große Zahl an Artefakten anwendbar und gut geeignet.	Die Gruppenbildung bei stark abhängigen Artefakten ist schlecht.
Die Gruppenbildung ist automatisierbar.	

Tabelle 6.4: Beurteilung Dominanz-Analyse: Vor- und Nachteile

Begriffsanalyse. Die Begriffsanalyse ist ein universelles Verfahren zur Bildung von Gruppen auf Basis unterschiedlichster Informationen. Diese Eigenschaft macht die Begriffsanalyse sehr flexibel in der individuellen Wahl geeigneter Eigenschaften und Objekte. Es werden Objekte durch Eigenschaften gruppiert, welche ein Entwickler im Kontext des Legacy-Systems festsetzt und in einem Konzept als Gruppe bzw. Modul festmacht. Verschiedene Gruppierungsbeispiele wurden unter anderem in den Phasen Redesign und Schichtentrennung gezeigt. Die Begriffsanalyse ist stark von den Erfahrungswerten eines Entwicklers geprägt und kann deshalb in der Entscheidung zu bildender Gruppen individuell ausfallen, was vorteilhaft oder nachteilig sein kann. Vorteilhaft ist dieser Aspekt im Gegensatz zur Dominanz-Analyse, da die Dominanz-Analyse aus gleichen Kontrollflussgraphen immer gleiche Gruppen bildet und es bei starken Abhängigkeiten zu einer schlechten Gruppenbildung kommen kann. Die Begriffsanalyse kann auch eine optimale Gruppenbildung liefern, wenn die Abhängigkeiten zwischen den Objekten stark sind. Denn der kreative Charakter des Verfahrens im Kontext des Legacy-Systems ist sehr hoch. Dieser kreative Aspekt kann aber auch dazu führen, dass zwischen den gebildeten Modulen zu viele Abhängigkeiten sind, wodurch eindeutige Schnittstellen nur schwer gebildet werden können. Bei der Dominanz-Analyse haben ermittelte Module nur die zum Vaterknoten eindeutige Abhängigkeit.

Die Begriffsanalyse ist nur bedingt für große Projekte geeignet. Denn bei großen Projekten, die eine Vielzahl von Objekten und Eigenschaften ermöglichen, ist dieses Verfahren ohne Werkzeugunterstützung aufwendig und führt schnell zu einem Verlust der Übersicht. Ein Werkzeug als Entscheidungshilfe könnte den Aufwand des Verfahrens stark reduzieren, wenn zum Beispiel automatisch Objekte und Eigenschaften aus dem Quell-Code ermittelt werden und ein Entwickler je nach Kriterium aus denen wählen kann. Nach der Wahl kann

das Werkzeug automatisch eine Zuordnung durchführen, woraus wiederum der Entwickler je nach Kriterium mögliche Konzepte wählen kann, welche für eine Gruppenbildung in Frage kommen.

Die Vor- und Nachteile der Begriffsanalyse wurden in der Tabelle 6.5 zusammengefasst.

Vorteile	**Nachteile**
Das Verfahren kann sowohl mit Verhaltensinformationen als auch mit strukturellen Informationen durchgeführt werden.	Das Verfahren ist für große Projekte bzw. für eine große Zahl an Objekten und Eigenschaften ohne Entscheidungswerkzeug aufwändig.
Das Verfahren ist in Wahl, Umsetzung und Zuordnung der Objekte und Eigenschaften individuell anwendbar und hat einen sehr hohen kreativen Anteil.	
Vorteil: Auch bei Legacy-Systemen mit monolithischen Eigenschaften ist eine Gruppenbildung möglich.	Nachteil: keine sinnvolle Gruppenbildung möglich bzw. ermittelte Module haben viele Abhängigkeiten
Das Verfahren ist universell für Zerlegungen bzw. Gruppenbildungen einsetzbar.	Das Verfahren kann nur im Rahmen einer Entscheidungshilfe für einen Entwickler automatisiert werden.

Tabelle 6.5: Beurteilung Begriffsanalyse: Vor- und Nachteile

MORPH. MORPH stellt eine Idee zur Informationsgewinnung aus Quell-Code dar, welche als Ansatz zur automatischen Transformation von einem Ausgangs-System zu einem Zielsystem genutzt werden kann. Es wird auf spezialisierte Weise eine Informationsgewinnung aus einer kombinierten Kontroll- und Datenflussanalyse vorgestellt. Der MORPH-Prozess ist spezialisiert, um zeichenorientierte Benutzerschnittstellen, welche durch Bildschirmein- und Bildschirmausgabefunktionen einer Programmiersprache repräsentiert werden, vollautomatisch zu erkennen und diese in eine graphische Benutzerschnittstelle zu überführen. Nicht mehr anwendbar ist dieser Prozess, wenn die zeichenorientierte Benutzerschnittstelle nicht mehr durch Bildschirmein- und Bildschirmausgabefunktionen einer Programmiersprache umgesetzt wird. Im betrachteten Legacy-System wurde die Benutzerschnittstelle durch die Komponente ISPF umgesetzt, weshalb MORPH nicht anwendbar war.

Ein Teilschritt aus dem MORPH-Prozess ist die Erkennung von Interaktionsobjekten und deren Zuordnung zu technologisch zeitgemäßen Oberflächenelementen. Dieser Teilschritt kann automatisiert werden. Zum Beispiel kann die in ISPF-Panel-Definitionsdateien definierte

zeichenorientierte Benutzerschnittstelle vollautomatisch in eine graphische Benutzerschnittstelle überführt werden. Ein manuell durchgeführtes Beispiel wurde im Fallbeispiel gezeigt.

In der Tabelle 6.6 wurden die wichtigsten Vor- und Nachteile von MORPH zusammengefasst.

Vorteile	Nachteile
MORPH ist ein kompletter Prozess, welcher auf spezialisierte Weise eine Idee zur Informationsgewinnung und Verarbeitung aus Quell-Code darstellt.	MORPH ist nicht anwendbar, wenn die Benutzerschnittstelle nicht durch Bildschirmein- und Bildschirmausgabefunktionen einer Programmiersprache umgesetzt wurde.
Bei Anpassung des Teilschritts Erkennung, kann MORPH auch für andere Benutzerschnittstellen genutzt werden.	MORPH ist nicht anwendbar, wenn bei einer Programmiersprache die Variablendeklaration nicht zum Sprachumfang gehört, weil Eigenschaften von Interaktionsobjekten über Variablendeklarationen ermittelt werden.
MORPH ist automatisierbar.	

Tabelle 6.6: Beurteilung MORPH: Vor- und Nachteile

6.4 Fazit zu den verwendeten Werkzeugen

Werkzeugunterstützung für Quell-Code-Analysen. Im Bereich der frei verfügbaren Analysewerkzeuge hat sich gezeigt, dass die Werkzeugunterstützung (z.B.: statische Quell-Code-Analysen und Hilfen zur Redokumentation) für ältere bzw. nicht populäre Programmiersprachen beschränkt ist. Zum Beispiel gibt es keine brauchbare Werkzeugunterstützung für in REXX geschriebene Anwendungen. Auch für COBOL oder PL/1 ist die Unterstützung beschränkt. Eine gute Werkzeugunterstützung kann man bei den Programmiersprachen C, C++ oder Java verzeichnen. Der Automatisierungsgrad dieses Konzeptes ist demnach stark von der vorliegenden Programmiersprache abhängig.

Software-Modellierungswerkzeug objectiF. Das Software-Modellierungswerkzeug *objectiF* bietet die im Konzept geforderten UML-Diagramme an und ist dadurch gut zur manuellen Redokumentation geeignet. Die Anwendung von objectiF als Entwicklungswerkzeug hat die Erkenntnis gebracht, dass die Transformationsalternative der MDA-Strategie umgesetzt

wurde und dadurch nur ein plattformunabhängiges Modell erstellt wird, welches direkt zu Quell-Code umgesetzt wird. Die Zieltechnologie und Architektur wird durch die bei Projektstart gewählte Vorlage festgelegt, weshalb die Plattformunabhängigkeit des zu erstellenden Modells (PIM) angezweifelt werden kann. Im Rahmen dieser Arbeit wurde nicht untersucht, ob dieses Modell durch einen Export auch für andere Software-Modellierungswerkzeuge oder für eine andere Vorlage in objectiF bereitgestellt werden kann (z.B.: über den Standard XMI).

Der Vorgang der modellgetriebenen Software-Entwicklung in objectiF ist geprägt von Unreife, Fehleranfälligkeit und Unflexibilität. Im Umgang mit objectiF musste das Fallbeispielprojekt durch nicht zu beseitigende Fehler mehrmals komplett neu erstellt werden. Zum Beispiel werden bestimmte Java-Dateien nur einmal bei Projekterstellung im von objectiF erzeugten Eclipse-Projekt erstellt, welche bei Änderung oder Verlust das ganze objectiF-Projekt bzw. Eclipse-Projekt unbrauchbar machen. Andere Fehlerquellen beziehen sich direkt auf die unausgereifte Modellierung und Handhabung des Programms. Zum Beispiel ist im Modell MyPresentation nur die Angabe einer Aktivität pro Trigger erlaubt, obwohl mehr definiert werden können. Eine weitere Inkonsistenz zwischen Modell und JSF-Technologie wird mit den Eintrittsaktivitäten in objectiF verursacht. Definierte Eintrittsaktivitäten, Methoden bzw. Funktionen die im Konstruktor einer Java-Dialogsteuerungsklasse ausgeführt werden, greifen auf Entitäten zu, welche noch nicht initialisiert wurden. Dieser Zugriff verursacht einen schwerwiegenden Fehler. Der Umgang mit objectiF ist nur für geschulte Entwickler empfehlenswert. Für nicht geschulte Entwickler ist die Ausweichung auf andere Software-Modellierungswerkzeuge zu empfehlen.

6.5 Zusammenfassung

Im ersten Abschnitt dieses Kapitels wurde die Umsetzung des Konzeptes ausgewertet und diskutiert. Es wurde die Phaseneinteilung ausgewertet, welche jederzeit eine Erweiterung bzw. Änderung ermöglicht. Mit der Kausalität bei Änderung der Ergebnisse einer Phase wurden die Konsequenzen der Abarbeitungsrichtung diskutiert. Die wichtigste Erkenntnis aus der Anwendung des Konzeptes ist die Transformationsfähigkeit von Software-Systemen. Diese muss als weiterführendes Thema weiter untersucht werden, um zum Beispiel im

Vorfeld die Transformationsfähigkeit von Legacy-Systemen bewerten zu können oder um zukünftige Software-Systeme transformationsfähig zu entwickeln bzw. die Transformationsfähigkeit auch nach Wartungs- und Pflegemaßnahmen zu erhalten. Um die Transformationsfähigkeit zu erhalten, wurde die MDA-Strategie als vorbeugendes Mittel diskutiert. Zum Abschluss wurden die Vor- und Nachteile des Konzeptes in einer Tabelle zusammengefasst.

Mit dem Grad der Automatisierung wurde auf die Schwierigkeit der Informationsgewinnung und Informationsauswertung eingegangen, wobei der kreative Anteil eines Entwicklers als Entscheidungsträger und Problemlöser unverzichtbar ist. Solange Software-Systeme keinen Software-Qualitätsmerkmalen entsprechen, die vollautomatische Transformierbarkeit ermöglichen, kann nur eine Semi-Automatisierung oder eine unvollständige Automatisierung ermöglicht werden.

Mit der Auswertung des Fallbeispiels wurden die Erfahrungen der Anwendung des Konzeptes dargestellt. Wichtige Erkenntnisse sind der unverzichtbare Einsatz von Analysewerkzeugen und die Durchführung einer Untersuchung vor Anwendung des Konzeptes, um den Aufwand des Software Reengineering für ein Legacy-System zu bestimmten. Ein Vergleich zwischen Legacy-System und Zielsystem, auf Basis der Software-Metrik LOC, vermittelte den Umfang zwischen Alt- und Neusystem. Es wurde gezeigt, dass die Gesamtanzahl an Quell-Code-Zeilen im Zielsystem größer ist, als im Legacy-System, aber der implementierte Anteil im Zielsystem wesentlich geringer ist als im Legacy-System. Durch den Einsatz modellgetriebener Software-Entwicklung können alle benötigten Grundgerüste aus den Modellen generiert werden, wobei im Wesentlichen nur noch die Implementierung von Ablauf- und Fachlogik notwendig ist.

Neben dem Konzept wurden die Techniken Dominanz-Analyse, Begriffsanalyse und MORPH beurteilt, in dem die Grenzen und die Möglichkeiten verdeutlicht wurden.

Den Abschluss dieses Kapitels bildete ein Fazit über das verwendete Software-Modellierungswerkzeug objectiF, wobei der Umgang des Werkzeugs beurteilt wurde. Ergänzend wurde eine Aussage über die Unterstützung von Quell-Code-Analyse-Werkzeugen im Bereich frei verfügbarer Werkzeuge gemacht, dessen Verfügbarkeit für ältere bzw. nicht populäre Programmiersprachen beschränkt oder nicht gegeben ist.

7 Abschlussbetrachtungen

7.1 Ziel, Ablauf und Ergebnisse der Arbeit

Das Ziel dieser Masterarbeit war, ein Konzept zu erstellen, welches ein Legacy-System, das mit einer prozeduralen Programmiersprache implementiert wurde, in ein Zielsystem mit einer Drei-Schichten-Architektur überführt.

Im ersten Teil dieser Arbeit wurde der Begriff Legacy-System erläutert und charakterisiert. Anschließend wurden verschiedene Strategien beim Umgang mit Legacy-Systemen analysiert, um zu bewerten, welche Strategien Unternehmen bevorzugen und um dem Konzept eine strategische Ausrichtung zu geben. Die strategische Ausrichtung, als Ergebnis der Untersuchung, war eine Kombination aus Migration und Neuentwicklung. Im Weiteren wurde die Terminologie des Software Reengineering dargestellt, um Aktivitäten, auf die das Konzept aufbaut, zu betrachten. Eine anschließende Erläuterung des idealtypischen Software-Lebenszykluses zeigte am Modell die Stadien der Entstehung eines Legacy-Systems und gab Aufschluss über den richtigen Zeitpunkt für eine mögliche Modernisierung.

Im zweiten Teil dieser Arbeit wurde die Notwendigkeit des Konzeptes bestätigt, in dem anhand einer Marktstudie über Software-Modernisierung eine Bewertung durchgeführt wurde. Anhand der Bewertung konnten konkrete Legacy- und Zielsystemeigenschaften abgeleitet werden, welche das Konzept spezialisieren. Die Spezialisierung umfasste unter anderem Legacy-Systeme, die mit einer prozeduralen Programmiersprache implementiert wurden, und die Drei-Schichten-Architektur im Zielsystem.

Im dritten Teil dieser Arbeit wurde das Software-Reengineering-Konzept als *4-Phasen-Transformationskonzept* allgemein und detailliert mit allen Phasen und Techniken beschrieben. An einem anschließenden Fallbeispiel wurde das *4-Phasen-Transformationskonzept* demonstriert und seine Eignung gezeigt.

Das Ergebnis dieser Arbeit ist das *4-Phasen-Transformationskonzept* (Abbildung 7.1), welches aus Aktivitäten des Software Reengineering aufgebaut ist und im Rahmen des festgelegten Eigenschaftsbereiches eine Modernisierung eines Legacy-Systems in vier Phasen ermöglicht.

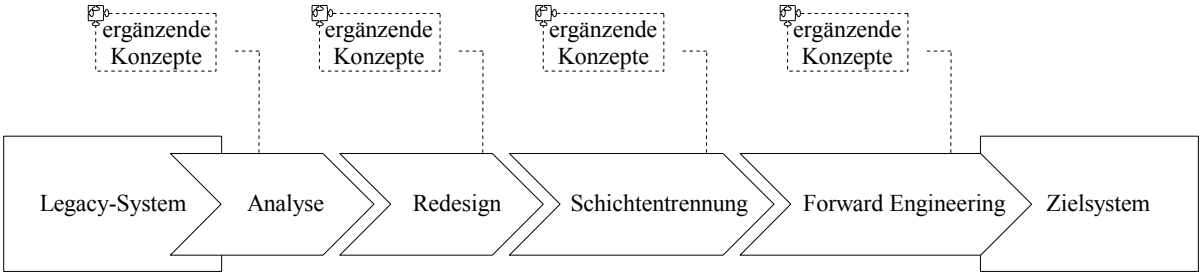


Abbildung 7.1: 4-Phasen-Transformationskonzept – schematische Übersicht

Weitere Erkenntnisse aus dieser Arbeit sind, dass Standards benötigt werden, welche Software-Qualitätsmerkmale messbar machen und die Transformationsfähigkeit gewährleisten.

7.2 Ausblick

Vorschläge zur Verbesserung und Erweiterung des Konzeptes wurden bereits im Laufe des Kapitels 6 vorgestellt und beschrieben. Dazu zählte die Vollständigkeit des Konzeptes, welche die Möglichkeit der Teilung der einzelnen Phasen erlaubt, um komplexe Phasen nochmals in einzelne Phasen aufzuteilen oder die ständige Erweiterung und Verbesserung der Techniken in den einzelnen Phasen ermöglicht. Im Zusammenhang mit dem Grad der Automatisierung des Konzeptes ist der Einsatz eines Werkzeugs als Entscheidungshilfe für einen Entwickler sinnvoll.

Weiterführende Themen. Eine aus dieser Arbeit hervorgehende Fragestellung beschäftigt sich mit der Transformationsfähigkeit von Software-Systemen:

- Welche Kriterien muss ein Software-System aufweisen (z.B.: Software-Qualität, erkennbare (System-) Architektur, struktureller Aufbau) bzw. welche Informationen müssen aus einem Software-System ermittelt werden können, damit es in ein Zielsystem überführt werden kann bzw. es transformationsfähig ist?

Im Idealfall können alle für die Entwicklung eines Zielsystems benötigten Informationen aus einem Ausgangssystem ermittelt werden, mit dem Ziel, dass das Zielsystem funktional äquivalent zum Ausgangssystem ist.

Im Eigenschaftsbereich des Konzeptes muss die Transformationsfähigkeit von Legacy-Systemen betrachtet werden, welche mit einer prozeduralen Programmiersprache implementiert wurden. Dann ist unter anderem zu untersuchen, wie muss die Software-Qualität und der Aufbau sein, um aus den Daten und Artefakten möglichst automatisch Objekte bzw. Klassen zu bilden.

Erste Ansatzpunkte zum Thema Transformationsfähigkeit von Software-Systemen können unter [FH09] mit der Forschungsarbeit über Continuous Software Engineering nachgeschlagen werden. Unter [LAZUSO09] können Ausführungen über zukünftige Software-Systeme und unter [WS03] (Beitrag 16) können Ausführungen über werkzeuggestützte Qualitätsuntersuchungen nachgelesen werden. Ein weiterer Ansatzpunkt wird voraussichtlich im Jahr 2010 mit einem neuen Standard für Software-Qualität entstehen, welcher das automatische Messen von Qualitätsmerkmalen ermöglichen soll und sich außerdem am ISO-Standard [ISO25000] orientiert. Informationen können unter *http://www.it-cisq.org* Consortium for IT Software Quality (CISQ) nachgeschlagen werden.

Ein weiteres Thema ist: Wie kann eine werkzeuggestützte vollautomatische Transformation realisiert werden, um ein Ausgangssystem in ein Zielsystem zu überführen? Diese Fragestellung baut auf der Transformationsfähigkeit von Software-Systemen auf.

Ein Ansatzpunkt zur Ermittlung von Informationen aus einem Ausgangssystem und der automatischen Transformationen liefert das vorherrschende Werkzeug Analysis and Renovation Catalyst (ARC) von der IBM. Passend dazu liefert diese Arbeit mit dem *4-Phasen-Transformationskonzept* Techniken zur Ermittlung von Informationen aus Ausgangssystemen im Zusammenhang mit verschiedenen Aktivitäten des Software Reengineering.

Literatur- und Quellenverzeichnis

[AIX2] IBM : *PL/I for AIX (V2.0)*. URL: <http://publib.boulder.ibm.com/infocenter/comphelp/v7v91/index.jsp?topic=/com.ibm.aix.pli.doc/ibmx2mst115.htm>, verfügbar am: 22.06.2009

[AND09] AndroMDA Team : *AndroMDA*. URL: <http://www.andromda.org>, verfügbar am: 18.08.2009

[BAL08] Balzert, Helmut : *Lehrbuch der Softwaretechnik: Softwaremanagement: Lehrbuch der Softwaretechnik*. - Spektrum Akademischer Verlag, 2008. - ISBN-13 978-3827411617

[BAL298] Balzert, Helmut : *Lehrbuch der Software-Technik - Band 2 (Software Management, Software Qualitätssicherung, Unternehmensmodellierung)*. - Berlin Heidelberg: Spektrum Akademischer Verlag, 1998. - ISBN-13 978-3827403018

[BB96] Baumöl, Ulrike ; Borchers, Jens ; Eicker, Stefan … : *Einordnung und Terminologie des Software Reengineering*. In: Informatik-Spektrum. - Berlin Heidelberg: Springer-Verlag. - 19 (August 1996) 4, S. 191-195

[BB05] Berger, Beat : *Legacy-Modernisierung: Auf die sanfte Tour kommt man oft weiter.* In: Netzwoche. - Basel: Netzmedien AG. - (2005) 38, S. 27

[BG07] Birchall, Graeme : *DB2 9 – SQL Cookbook*. URL: <http://mysite.verizon.net/Graeme_Birchall/cookbook/DB2V91CK.PDF>, verfügbar am: 03.06.2009

[BP05] Bosshard & Partner Unternehmensberatung (Autor: Berger, Beat) : *Software Modernisierung – Quo Vadis? ; Wie gehen IT Führungskräfte in der Schweiz mit der Herausforderung Software Modernisierung um?*. - März 2005. - 8 S. - Berikon (Schweiz), Eine Marktstudie

[BR04] Bennicke, Marcel ; Rust, Heinrich : *Programmverstehen und statische Analysetechniken im Kontext des Reverse Engineering und der Qualitätssicherung*. - 2004. - 128 S. - Fraunhofer-Institut für Angewandte Informationstechnik FIT, Forschungsprojekt: Virtuelles Software Engineering Kompetenzzentrum (VSEK), Report: ViSEK/25/D

[BS09] Bönsch, Steffen : *Untersuchungen zur Software-Produktlinien-Erstellung für eine Enterprise-Applikation unter Verwendung der modellgetriebenen Software-Architektur (MDA) und der modellgetriebenen Software-Entwicklung (MDSD)*. - 2009. - 107 S. - Mittweida, Hochschule Mittweida (FH), Informatik, Diplomarbeit, 2009

[BS95] Brodie, Michael L. ; Stonebraker, Michael : *Migrating Legacy Systems: Gateways, Interfaces & the Incremental Approach: Gateways, Interfaces and the Incremental Approach*. - Morgan Kaufmann Publishers Inc., 1995. - ISBN-13 978-1558603301

[CHRCD09] Christoph, Franz : *Begleit-CD zur Masterarbeit*. - 2009. - Mittweida, Hochschule Mittweida (FH), Fachbereich Mathematik / Physik / Informatik, CD

[CO00] Cockburn, Alistair : *Writing Effective Use Cases*. - Addison Wesley Verlag, 2000

[CV95] Cimitile, Aniello ; Visaggio, Guiseppe : *Software Salvaging and the Call Dominance Tree*. In: The Journal of Systems and Software. - New York: Elsevier Science Inc. - 28 (1995) 2, S. 117-127

[CC90] Chikofsky, Elliot J. ; Cross II, James H. : *Reverse Engineering and Design Recovery: A Taxonomy*. In: IEEE Software. - 7 (Januar 1990) 1, S. 13-17

[CH01] Cooper, Keith D. ; Harvey, Timothy J. ; Kennedy, Ken : *A Simple, Fast Dominance Algorithm*. In: Software-Practice and Experience (SPE). - John Wiley & Sons Verlag. - (2001)

[DA00] Davis, Kathi Hogshead ; Aiken, Peter H. : *Data Reverse Engineering: A Historical Survey*. In: WCRE: Seventh Working Conference on Reverse Engineering. - IEEE Computer Society. - (23-25 November 2000), S. 70 - 78

[DIN66001] DIN 66001 / ISO 5807. *Sinnbilder für Datenfluss- und Programmablaufpläne*, 1983

[DIN66261] DIN 66261. *Informationsverarbeitung; Sinnbilder für Struktogramme nach Nassi-Shneiderman*, 1985

[DJ05] Deuring, Johann : *REXX Grundlagen für die z/OS Praxis*. - Oldenbourg Wissenschaftsverlag, 2005. - ISBN-13 978-3486200256

[DJ08] Dunkel, Jürgen ; Eberhart, Andreas ; Fischer, Stefan … : *Systemarchitekturen für verteilte Anwendungen – Client-Server // Multi-Tier // SOA // Event-Driven Architecture // P2P // Grid // Web 2.0*. - München: Hanser Fachbuchverlag, 2008. - ISBN-13 978-3446413214

[DoD97] Department of Defense (DoD) USA : *Software Reengineering Assessment Handbook – Version 3.0*. - 1997. - 234 S.

[DOX09] Doxygen : *Redokumentationswerkzeug*. URL: <www.doxygen.org>, verfügbar am 10.08.2009

[EC05] Erdle, Christoph : *Legacy Migrationsstrategien*. - 2005. - 23 S. - München, Technische Universität München, Informatik, Seminararbeit

[FH09] Fraunhofer-Institut für Software- und Systemtechnik (ISST) : *Continuous Software Engineering (CSE)*. URL: <http://www.isst.fraunhofer.de/leittheme n/cse/about/index.jsp>, verfügbar am: 25.08.2009

[FO06] Fong, Joseph F.P. : *Information Systems Reengineering and Integration – Second Edition*. - London: Springer-Verlag, 2006. - ISBN-13 978-1846283826

[FOL95] Foley, James D. ; van Dam, Andries ; Feiner, Steven K. … : *Computer Graphics: Principles and Practice – Second Edition in C*. - Addison-Wesley Professional, 1995

[GK93] Gall, Harald ; Klösch, Rene : *Capsule oriented reverse engineering for software reuse*. - Berlin/Heidelberg: Springer-Verlag, 1993

[GKM95] Gall, Harald ; Klösch, Rene ; Mittermeir, R. : *Object-oriented re-architecturing*. - 5th European Software Engineering Conference (ESEC 95). - Berlin: Springer-Verlag, 1995

[IBM08] IBM : *z/OS V1R10.0 ISPF Dialog Developer's Guide and Reference – Document Number: SC34-4821-07*. URL: <http://publibfp.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/ispzdg70/CCONTENTS>, verfügbar am: 03.08.2009

[IBM97] IBM : *AS/400 Advanced Series – REXX/400 Reference – Version 4*. URL: <http://publib.boulder.ibm.com/iseries/v5r2/ic2924/books/c4157290.pdf>, verfügbar am: 08.06.2009

[IDE02] Ide, Markus : *J2EE – Programmierhandbuch: Einführung in die Java 2 Enterprise Edition*. - Software & Support Verlag GmbH, 2002

[IEEE90] Institute of Electrical and Electronics Engineers (IEEE) : *IEEE Standard 610.12-1990 – glossary of software engineering terminology*, 1990

[IEEE98] Institute of Electrical and Electronics Engineers (IEEE) : *IEEE Standard 1219-1998 – Standard for Software Maintenance*, 1998

[ISO12207] ISO/IEC 12207. *Information Technology – Software life cycle processes*, 1995

[ISO14977] ISO/IEC 14977. *Extended Backus-Naur Form (Extended BNF)*, 1996 (E)

[ISO25000] ISO/IEC 25000. *Software product Quality Requirements and Evaluation (SquaRE)*, 2005 (E) – enthält [ISO9126]

[ISO9126] ISO/IEC 9126. *Software-Technik – Qualitätsmerkmale für Software-Produkte*, Ausgabe 2001 (siehe auch ehemals DIN 66272)

[KA08] Kyte, Andy : *Anwendungen - Modernisierungen nötig*. In: CIO – IT-Strategie für Manager. - München: IDG Business Media Verlag. - 8 (Mai 2008), S. 6

[KN05] Krumke, Sven Oliver ; Noltemeier, Hartmut : *Graphentheoretische Konzepte und Algorithmen*. - Wiesbaden: Vieweg+Teubner-Verlag, 2005. - ISBN-13 978-3519005261

[KO04] Koschke, Prof. Dr. rer. nat. Rainer ; Simon D. : *Vorlesung Software-Reengineering*. - Oktober 2004. - 518 S. - Bremen, Universität Bremen, Mathematik und Informatik, Vorlesungsskript

[LAZUSO09] Rausch, Andreas ; Goltz, Ursula ; Engels, Gregor … : *1. Workshop für langlebige und zukunftsfähige Softwaresysteme 2009 – SE Konferenz 2009 Kaiserslautern*. - 2009. - 23 S. - Technische Universität Carolo-Wilhelmina zu Braunschweig, Institut für Programmierung und Reaktive Systeme, Informatik-Bericht Nr. 2009-05

[LF05] Lanz, Franz : *ISPF professionell nutzen – Das praxisorientierte Lehr- und Handbuch für den professionellen ISPF Benutzer.* - Oldenbourg Wissenschaftsverlag, 2005. - ISBN-13 978-3486576429

[LP09] Liggesmeyer, Prof. Dr.-Ing. Peter : *Liste von Werkzeugen zur Code-Analyse*. URL: <http://liggesmeyer.de/mediapool/31/310284/data/Werkzeuge.pdf>, verfügbar am: 27.10.2009

[MAS07] Masak, Dieter : *Legacysoftware: Das lange Leben der Altsysteme*. - Berlin Heidelberg: Springer-Verlag, 2006. - ISBN-13 978-3540254126

[MDA09] Object Management Group (OMG) : *OMG Model Driven Architecture*. URL: <http://www.omg.org/mda/>, verfügbar am: 04.06.2009

[MOO] Moore, Melody M. : *MORPH – User Interface Reengineering*. - 202 S. - Atlanta, Georgia (USA), Georgia State University, Dissertation. URL: <http://www.cis.gsu.edu/~mmoore/MORPH/dissertation/Dissertation.html>, verfügbar am: 24.07.2009

[MÜ97] Müller, Dr. rer. nat. Bernd : *Reengineering : Eine Einführung*. - Stuttgart: B. G. Teubner, 1997. - ISBN 3-519-02942-1

[MW90] Mell, Wolf Dieter ; Preuss, Peter ; Sandner, Peter : *Einführung in die Programmiersprache PL/1*. - Verlag: Bibliogr. Inst. + Brockha, 1990. - ISBN-13 978-3411007851

[OMG03] OMG : *The MDA Guide*. - Version 1.0.1. - 2003. - verfügbar unter [MDA09]

[OV00] IT-Services and Solutions GmbH : *Design-Dokument Objektverwaltung & Toolset „EURO“ Kurzbeschreibung*. - Chemnitz, 2000

[RB00] Rajlich, Vaclav T. ; Bennett, Keith H. : *A Staged Model for the Software Life Cycle*. In: IEEE Computer. - 33 (2000) 7, S. 66-71

[RIGI09] Rigi : *graphisches Analysewerkzeug für Legacy-Systeme*. URL: <http://www.rigi.cs.uvic.ca/>, verfügbar am: 27.10.2009

[SB03] Sturm, Thorsten ; Boger, Marko : *Softwareentwicklung auf Basis der Model Driven Architecture*. - Dpunkt Verlag, 2003

[SOM07] Sommerville, Ian : *Software Engineering*. - 8. Auflage. - München: Pearson Studium Verlag, 2007. - ISBN-978-3827372574

[SOMA] IBM : *Service-oriented modeling and architecture (SOMA)*. URL: <http://www.ibm.com/developerworks/library/ws-soa-design1/>, verfügbar am: 09.09.2009

[SP09] SchemaSpy : *Werkzeug zur Wiedergewinnung von Datenbankschemata (ER-Modell) bzw. Meta-Informationen aus Datenbanken*. URL: <http://schemaspy.sourceforge.net/>, verfügbar am: 27.10.2009

[SR99] Siff, Michael ; Reps, Thomas : *Identifying modules via concept analysis*. In: IEEE Transactions on Software Engineering. - 25 (Nov/Dec 1999) 6, S. 749-768

[SV07] Stahl, Thomas ; Völter, Markus ; Efftinge, Sven ... : *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management*. - 2. Auflage. - Heidelberg: Dpunkt Verlag, 2007. - ISBN-13 978-3898644488

[TG00] Thaller, Georg Erwin : *Software-Metriken einsetzen – bewerten – messen*. - 2. Auflage. - Berlin: Verlag Technik, 2000. - ISBN-13 978-3341012604

[UC09] Ullenboom, Christian : *Java ist auch eine Insel: Programmieren mit der Java Platform, Standard Edition – Version 6*. - 8. Auflage. - Galileo Press (Galileo Computing), 2009. - ISBN-13 978-3836213714

URL: <http://openbook.galileocomputing.de/javainsel8/>, verfügbar am: 05.06.2009

[UML09] Object Management Group (OMG) : *Unified Modeling Language (UML), Version 2.2*. URL: <http://www.omg.org/technology/documents/formal/uml.htm>, verfügbar am: 31.07.2009

[VL96] Volkmann, Lutz : *Fundamente der Graphentheorie*. - Wien: Springer-Verlag, 1996. - ISBN-13 978-3211827741

[WS03] Ebert, Jürgen ; Riediger, Volker ; Winter, Andreas … : *5. Workshop Software Reengineering*. - 2003. - 49 S. - Universität Koblenz – Landau, Bad Honnef, URL: <http://www.uni-koblenz.de/~ist/wsr/>, verfügbar am: 27.10.2009

[YOU93] Yourdon, Edward : *Die westliche Programmierkunst am Scheideweg (Die Schlüsseltechniken der Softwareentwicklung für das 21. Jahrhundert)*. - Hanser Elektronik / Fachbuch Verlag, 1993. - ISBN-13 978-3446175181

[ZVM08] IBM Corporation : *z/VM V5R2.0 CMS Application Development Guide – SC24-6069-01*. URL: <http://publib.boulder.ibm.com/infocenter/zvm/v5r3/index.jsp?topic=/com.ibm.zvm.v53.dmsa3/ispfmsg.htm>, verfügbar am: 03.06.2009

A Beispiel eines Kriterienkataloges

Im Folgenden werden einige Kriterien aufgelistet, welche zur Erstellung eines Kriterienkataloges für Wiederverwendbarkeit durch Migration berücksichtigt werden können. Ein Beispielaufbau eines Kriterienkataloges wird mit Tabelle A.1 gegeben.

Kriterien

- **Dokumentationen**
  Ein Kriterium kann die Verfügbarkeit von Dokumentationen sein, um zum Beispiel Informationen über Funktionalitäten, benötigte Infrastruktur, Voraussetzungen, Programmier- oder Kommunikationsschnittstellen zu ermitteln. Dokumentationen können Informationen für andere aufgestellte Kriterien liefern.

- **Art und Alter**
  Ab welcher Version, zum Beispiel bei einer Komponente (Datenbank), ist eine Wiederverwendung sinnvoll (z.B.: gegebene Funktionalitäten usw.)? Ist eine geforderte Code-zu-Code-Transformation bei Modulen möglich?

- **Lokalität**
  Ist zum Beispiel eine Komponente oder ein Modul in sich abgeschlossen und nur über eine Schnittstelle erreichbar? Kann eine Komponente unverändert wiederverwendet werden (Black-Box-Wiederverwendung) oder muss diese angepasst werden (White-Box-Wiederverwendung)? Befinden sich alle Definitionen (z.B.: Variablen) innerhalb eines Moduls? Stellt eine Komponente bzw. ein Modul einen isolierten fachlichen Teil eines Gesamtsystems dar?

- **Integrationsfähigkeit**
  Schnittstellen. Welche Schnittstellen bzw. Schnittstellenversionen werden unterstützt und benötigt? Sind diese wohldefiniert, spezifiziert, Standards? Ist eine geringe Kopplung durch die Schnittstellen realisierbar, um Komponenten schnell und einfach austauschen zu können?

Fehlende Lokalität mindert Integrationsfähigkeit. Die Nutzung von globalen Variablen verstärkt Kopplung und mindert Lokalität.

- **Software-Qualitätsmerkmale [ISO9126]**

  - **Funktionalität.** Ist die Fähigkeit der Interoperabilität gegeben (siehe zum Beispiel Integrationsfähigkeit)? Werden sicherheitsrelevante Vorschriften oder gesetzliche Bestimmungen erfüllt (z.B.: unberechtigter Zugriff)?

  - **Zuverlässigkeit.** Werden Normen oder Vereinbarungen zur Zuverlässigkeit erfüllt?

  - **Effizienz.** Ist ein gefordertes Leistungsniveau mit festgelegten Bedingungen realisierbar? Sind zum Beispiel Antwort- und Verarbeitungszeiten bei geforderter Funktionsausführung realisierbar?

  - **Übertragbarkeit.** Ist zum Beispiel eine Komponente auf eine festgelegte Hard- oder Software-Umgebung (Plattform) übertragbar?

Tabelle A.1: Beispielaufbau Kriterienkatalog

Kriterienkatalog für: ...						
Kriterium	**Kriterium erfüllt?**					
...	ja		nein		anpassbar	
...	ja		nein		anpassbar	
...	...		...		...	

Wiederverwendbar durch Migration? (nach Kategorieeinteilung für Wiederverwendbarkeit)	

Notwendige Anpassungen um Wiederverwendung zu ermöglichen: ...

B Data Dictionary der Objektverwaltung

B.1 Tabelle – Member-Verwaltung (MVSATZ)

Feldname	Datentyp	Beschreibung
MVMEMB	CHAR	1{Zeichen}8; Member-Name
MVFILE	CHAR	1{Zeichen}2; Dateikennzeichen
MVPROG	CHAR	1{Zeichen}1; Kennzeichen für Programmiersprache
MVOBJ	CHAR	1{Zeichen}1; Kennzeichen für Objekttyp
MVFUNK	CHAR	1{Zeichen}8; Funktionskomplex
MVKMPL1	CHAR	1{Zeichen}3; Änderungskomplex 1 (aktueller)
MVKMPL2	CHAR	1{Zeichen}3; Änderungskomplex 2
MVKMPL3	CHAR	1{Zeichen}3; Änderungskomplex 3
MVKMPL4	CHAR	1{Zeichen}3; Änderungskomplex 4
MVDRUF	CHAR	1{Zeichen}1; Flag für Protokolldruck
MVUSERID	CHAR	1{Zeichen}6; USERID
MVSTAT	CHAR	1{Zeichen}1; Bearbeitungsstatus
MVFPROJ	CHAR	1{Zeichen}7; Fremd-Projekt-Nr., z.B. P652013
MVCNT	CHAR	1{Zeichen}2; Überstellungszähler
MVDAT1	CHAR	8{Zeichen}8; Datum maschinelle Analyse (JJJJMMTT)
MVDAT2	CHAR	8{Zeichen}8; Datum Schnellanalyse
MVDAT3	CHAR	8{Zeichen}8; Datum Feinanalyse
MVDAT4	CHAR	8{Zeichen}8; Datum IPSA-Bearbeitung
MVDAT5	CHAR	8{Zeichen}8; Datum Vorübergabe
MVDAT6	CHAR	8{Zeichen}8; Datum Übergabe
MVNAKTIV	CHAR	1{Zeichen}1; Member n. aktiv oder übergeben an Fremdpr.
MVNAGRD	CHAR	1{Zeichen}30; Begründung zu MVNAKTIV
MVUSER	CHAR	1{Zeichen}6; User-ID der letzten Änderung
MVTIMEST	TIMESTAMP	Datum-/Zeitstempel der letzten Änderung

Tabelle B.1: Data Dictionary: Tabelle Member-Verwaltung (MVSATZ)

B.2 Tabelle – Dateiverwaltung (DVSATZ)

Feldname	Datentyp	Beschreibung
DVFILE	CHAR	1{Zeichen}44; Dateiname
DVFTYPE	CHAR	1{Zeichen}4; Dateiart (File, entladene DB)
DVDSORG	CHAR	1{Zeichen}4; Dateiorganisation
DVDEVT	CHAR	1{Zeichen}2; Gerätetype (MB oder DA)
DVECFM	CHAR	1{Zeichen}3; Satzformat
DVLRECLA	CHAR	1{Zeichen}5; Log. Satzlänge (alt)
DVLRECLN	CHAR	1{Zeichen}5; Log. Satzlänge (neu)
DVBLKSA	CHAR	1{Zeichen}5; Phys. Satzlänge (alt)
DVBLKSN	CHAR	1{Zeichen}5; Phys. Satzlänge (neu)
DVSPACE	CHAR	1{Zeichen}4; Dateigröße (ca. Speicherplatzbedarf in Zylinder)
DVFUNK	CHAR	1{Zeichen}8; Funktionskomplex
DVKMPL1	CHAR	1{Zeichen}3; Änderungskomplex 1 (aktueller)
DVKMPL2	CHAR	1{Zeichen}3; Änderungskomplex 2
DVKMPL3	CHAR	1{Zeichen}3; Änderungskomplex 3
DVKMPL4	CHAR	1{Zeichen}3; Änderungskomplex 4
DVDRUF	CHAR	1{Zeichen}1; Flag für Protokolldruck
DVUSERID	CHAR	1{Zeichen}6; USERID
DVSTAT	CHAR	1{Zeichen}1; Bearbeitungsstatus
DVCNT	CHAR	1{Zeichen}2; Überstellungszähler
DVUJOB	CHAR	1{Zeichen}8; Name des Umstellungsjobs
DVDAT1	CHAR	1{Zeichen}8; Erfassungsdatum
DVDAT2	CHAR	1{Zeichen}8; Datum Bestätigung Umstellungsvorschrift
DVDAT3	CHAR	1{Zeichen}8; Datum technischer Test
DVSAT4	CHAR	1{Zeichen}8; Übergabedatum
DVUSER	CHAR	1{Zeichen}6; User-ID der letzten Änderung
DVTIMEST	TIMESTAMP	Datum-/Zeitstempel der letzten Änderung

Tabelle B.2: Data Dictionary: Tabelle Dateiverwaltung (DVSATZ)

B.3 Tabelle – Änderungskomplex (CKSATZ)

Feldname	Datentyp	Beschreibung
CKKMPL	CHAR	1{Zeichen}3; Änderungskomplexnummer
CKTEXT	VARCHAR	1{Zeichen}400; Beschreibung
CKUSER	CHAR	1{Zeichen}6; User-ID der letzten Änderung
CKTIMEST	TIMESTAMP	Datum-/Zeitstempel der letzten Änderung

Tabelle B.3: Data Dictionary: Tabelle Änderungskomplex (CKSATZ)

B.4 Tabelle – Informations- und Prüfdatei (IDSATZ)

Feldname	Datentyp	Beschreibung
IDKZTYP	CHAR	1{Zeichen}5; Kennzeichentyp
IDKZ	CHAR	1{Zeichen}10; Kennzeichen
IDTEXT	CHAR	1{Zeichen}65; Bedeutung
IDUSER	CHAR	1{Zeichen}6; User-ID der letzten Änderung
IDTIMEST	TIMESTAMP	Datum-/Zeitstempel der letzten Änderung

Tabelle B.4: Data Dictionary: Tabelle Informations- und Prüfdatei (IDSATZ)

B.5 Protokolldatei (PRSATZ)

Feldname	Beschreibung
PRCC	1{Zeichen}1; Druckersteuerzeichen
PRGRUND	1{Zeichen}1; Protokollgrund (Delete / Iteration)
PRDATE	10{Zeichen}10; Datum (TT.MM.JJJJ)
PRTIME	8{Zeichen}8; Uhrzeit (hh.mm.ss)
PRUSERID	1{Zeichen}6; USERID
PRTEXT	1{Zeichen}94; Informationstext

Tabelle B.5: Data Dictionary: Protokolldatei (PRSATZ)

B.6 OV.REXX – Konstanten und globale Variablen

Tabelle B.6: Data Dictionary: OV.REXX - Konstanten und globale Variablen

Name	Beschreibung
KONST.DB.MODE.READ	"R"; Datenbankzugriffsmodus: Datensatz direkt lesen
KONST.DB.MODE.WRITE	"W"; Datenbankzugriffsmodus: Datensatz schreiben
KONST.DB.MODE.NEW	"N"; Datenbankzugriffsmodus: Datensatz erzeugen
KONST.DB.MODE.DELETE	"D"; Datenbankzugriffsmodus: Datensatz löschen
(KONST.DB.MODE.)	"S"; Datenbankzugriffsmodus: Systemtabellen letztes Backup lesen
KONST.DB.ID.MV	"MV "; Tabellenidentifikator: Member-Verwaltung
KONST.DB.ID.DV	"DV "; Tabellenidentifikator: Dateiverwaltung
KONST.DB.ID.DF	"DF "; Tabellenidentifikator: Datenfeldverwaltung
KONST.DB.ID.CK	"CK "; Tabellenidentifikator: Änderungskomplex
KONST.DB.ID.ID	"ID "; Tabellenidentifikator: Infodatei
KONST.DB.UPRO	"db2io"; PL/1-Programmname OV.PLI(DB2IO)
KONST.DB.UPROSEQ	"db2seq"; PL/1-Programmname OV.PLI(DB2SEQ)
KONST.DF.RETR.CLEAR	1; Daten löschen (ov_df_retrieve)
KONST.DF.RETR.TAKE	2; Daten aufnehmen (ov_df_retrieve)
KONST.LIST.ALL	"DATF DEVT DSN DSORG FILE FSTA FUNK LANG MEMB PSTA RECF USER NAKT"
KONST.LIST.DATF	"DATF"; Liste Datenfeldformate
KONST.LIST.DEVT	"DEVT"; Liste Gerätetypen; z.B.: Magnetbandgerät (MB)
KONST.LIST.DSN	"DSN"; Liste Dateikennzeichen
KONST.LIST.DSORG	"DSORG"; Liste Dateiorganisationen; z.B.: Direktzugriff (DA)
KONST.LIST.FILE	"FILE"; Liste Dateiarten
KONST.LIST.FSTA	"FSTA"; Liste Bearbeitungsstatus
KONST.LIST.FUNK	"FUNK"; Liste Funktionskomplexe
KONST.LIST.LANG	"LANG"; Liste Programmiersprachen
KONST.LIST.MEMB	"MEMB"; Liste Membertypen
KONST.LIST.PSTA	"PSTA"; Liste von Bearbeitungsstatus
KONST.LIST.RECF	"RECF"; Liste von Satzformaten
KONST.LIST.USER	"USER"; Liste aller Nutzer

▼

Name	Beschreibung
KONST.LIST.NAKT	"NAKT"; Liste nicht aktiver Member
KONST.MSG.CT_OK	0; fehlerfreie Ausführung (ov_ck_table_create)
KONST.MSG.CT_READERR	-1; Fehler beim Lesen (ov_ck_table_create)
KONST.MSG.CT_TBCREERR	-2; Fehler bei Table create (ov_ck_table_create)
KONST.MSG.CT_ERROR	-3; sonstige Fehler (ov_ck_table_create)
KONST.MSG.PL_OK	0; Verarbeitung erfolgreich (Panel)
KONST.MSG.PL_ABBRUCH	-1; Panel wurde abgebrochen (Panel)
KONST.MSG.PL_TBCREERR	-2; Fehler bei TBCREATE (Panel)
KONST.MSG.PL_ERROR	-3; Fehler-MSG erzeugt (Panel)
KONST.MSG.PL_CANTREAD	-4; kann Datensatz nicht lesen (Panel)
KONST.MSG.PR_OK	0; fehlerfreie Ausführung (ov_protokoll)
KONST.MSG.PR_WRONGID	-1; ungültiger Modi (ov_protokoll)
KONST.MSG.PR_CANTALOC	-2; Protokolldatei nicht allokierbar (ov_protokoll)
KONST.MSG.PR_CANTREAD	-3; Protokolldatei nicht lesbar (ov_protokoll)
KONST.MSG.PR_CANTCREA	-4; Protokolldatei nicht erzeugbar (ov_protokoll)
KONST.MSG.PR_CANTWRIT	-5; Protokolldatei nicht schreibbar (ov_protokoll)
KONST.MSG.RW_NOTCHG	1; Datensatz unverändert (ov_x_write); x=(df|dv|id|ck|mv)
KONST.MSG.RW_OK	0; Datensatz gespeichert (ov_x_write)
KONST.MSG.RW_RECCHG	-1; Datensatz ist verändert (ov_x_write)
KONST.MSG.RW_ERROR	-2; Fehler (ov_x_write)
KONST.MSG.VS_MULTIREC	1; mehrere Datensätze vorhanden (ov_db2pli_)
KONST.MSG.VS_OK	0; fehlerfreie Ausführung (ov_db2pli_)
KONST.MSG.VS_CALLERR	-1; Fehler bei CALL (ov_db2pli_)
KONST.MSG.VS_VGETERR	-2; Fehler bei VGET (ov_db2pli_)
KONST.MSG.VS_NOTFOUND	-3; kein Datensatz gefunden (ov_db2pli_)
KONST.MSG.VS_VSAMERR	-4; Fehler bei VSAM-Zugriff (ov_db2pli_)
KONST.MSG.VS_VPUTERR	-5; Fehler bei VPUT (ov_db2pli_)
KONST.PR.FLAG.DELETE	1; gelöschter Datensatz (Protokollparameter)
KONST.PR.FLAG.UEBERST	2; Überstellungsiteration (Protokollparameter)
KONST.TEXT.FILE.SELECT	"Auswahl der Datei"; Text für ov_dv_panel_file()
KONST.TEXT.FILE.FILENEED	"Keine Datei angegeben-bitte wählen."; Text für ov_dv_panel_file()

▼

Name	Beschreibung
KONST.TEXT.DSN.SELECT	"Auswahl des Dateikennzeichen"; Text für ov_mv_panel_dsn(); Dateikennzeichen (DKZ)
KONST.TEXT.DSN.NEW	"Datensatz wird neu erzeugt-DKZ wählen."; Text für ov_mv_panel_dsn()
KONST.TEXT.DSN.MORET2	"Es existieren mehrere Member-DKZ wählen."; Text für ov_mv_panel_dsn()
KONST.USER.CK	"C"; Nutzerrecht; Schreiben der Änderungskomplexe
KONST.USER.DF	"F"; Nutzerrecht; immer Schreiben in Datenfeldverwaltung
KONST.USER.DV	"D"; Nutzerrecht; immer Schreiben in Dateiverwaltung
KONST.USER.ID	"I"; Nutzerrecht; Schreiben der Info-Datei
KONST.USER.MV	"M"; Nutzerrecht; immer Schreiben in Member-Verwaltung
DBSATZ	1{Zeichen}n; temporäre globale Variable zur Übergabe von Datensätzen aus Datenbank. Je nach Datenstruktur wird Zeichenkette an globale Variablen MVSATZ, IDSATZ, DFSATZ, CKSATZ, DVSATZ übergeben.
MVSATZ	1{Zeichen}n; globale Variable, für Member-Daten, mvsatz = DBSATZ
DVSATZ	1{Zeichen}n; globale Variable, für Datei-Daten, dvsatz = DBSATZ
CKSATZ	1{Zeichen}n; globale Variable, für Änderungskomplex-Daten, cksatz = DBSATZ
DFSATZ	1{Zeichen}n; globale Variable, für Datenfeld-Daten, dfsatz = DBSATZ
IDSATZ	1{Zeichen}n; globale Variable, für INFO-Tabelle-Daten, idsatz = DBSATZ
ov.mv.satz	globale Variable, ist Kopie von mvsatz
(globale Variablen) ov.ck.text ov.ck.userid ov.ck.kmpl ov.ck.newflag	In jeder dieser globalen Variablen werden die Einzelwerte aus cksatz separat gespeichert CKTEXT CKUSER CKKMPL 1{Zeichen}1, Flag für neuen Änderungskomplex

▼

Name	Beschreibung
(globale Variablen) ov.mv.memb ov.mv.file ov.mv.prog ov.mv.obj ov.mv.funk datakpx=(ov.mv.kmpl.1, ov.mv.kmpl.2, ov.mv.kmpl.3, ov.mv.kmpl.4) ov.mv.druf ov.mv.userid ov.mv.stat ov.mv.statold ov.mv.fproj ov.mv.cnt ov.mv.dat.1 ov.mv.dat.2 ov.mv.dat.3 ov.mv.dat.4 ov.mv.dat.5 ov.mv.dat.6 ov.mv.naktiv ov.mv.nagrd	In jeder dieser globalen Variablen werden die Einzelwerte aus mvsatz separat gespeichert MVMEMB MVFILE MVPROG MVOBJ MVFUNK MVKMPL1, MVKMPL2, MVKMPL3, MVKMPL4 MVDRUF MVUSERID MVSTAT MVSTAT, Bei Statuswechsel – alten Status merken MVFPROJ MVCNT MVDAT1 MVDAT2 MVDAT3 MVDAT4 MVDAT5 MVDAT6 MVNAKTIV MVNAGRD
(globale Variablen) ov.dv. ov.df. ov.id.	 globale Variable für Dateiverwaltung (Einzelwerte entsprechen DVSATZ) globale Variable für Datenfeldverwaltung (Einzelwerte entsprechen DFSATZ) globale Variable für Informations- und Prüfdatei (Einzelwerte entsprechen IDSATZ)
(globale Variablen – von ISPF) pmv... pck... pid... pdf... pdv...	Globale Variablen von ISPF beginnen mit dem Prefix p und einem anschließendem Kürzel je nach Verwaltung. Diese werden in der Präsentationsschicht verwendet und sind auch in REXX global. (z.B. pmvmemb für Member-Verwaltung – Membername (MVMEMB))
Datenschlüssel	Entspricht dem primären Schlüssel einer gewählten Tabelle
Kennzeichenliste	Ist eine Liste je nach geforderter Option (KONST.LIST.) zurückgegeben von ov_list()

C Programmablaufpläne (OV.REXX)

C.1 Hauptprogramm – Fkt: 1 (Member-Verwaltung)

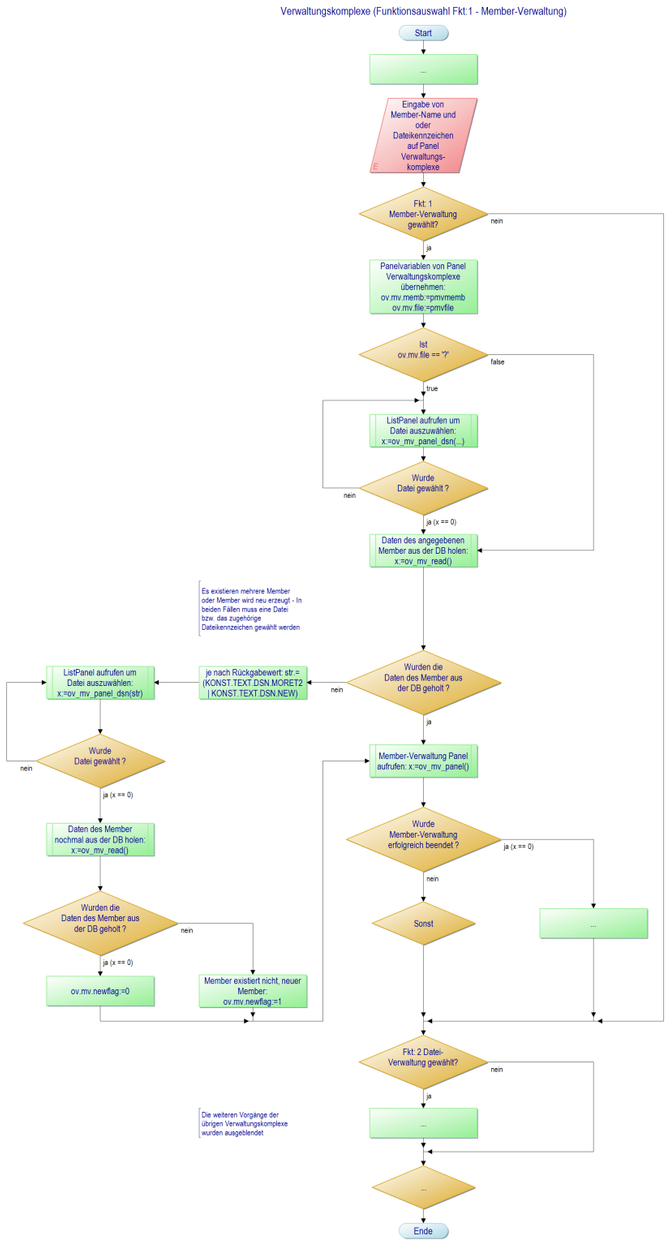


Abbildung C.1: Fallbeispiel – Programmablaufplan: Fkt. 1 (Member-Verwaltung)

C.2 Member-Datensatz schreiben: ov_mv_write()

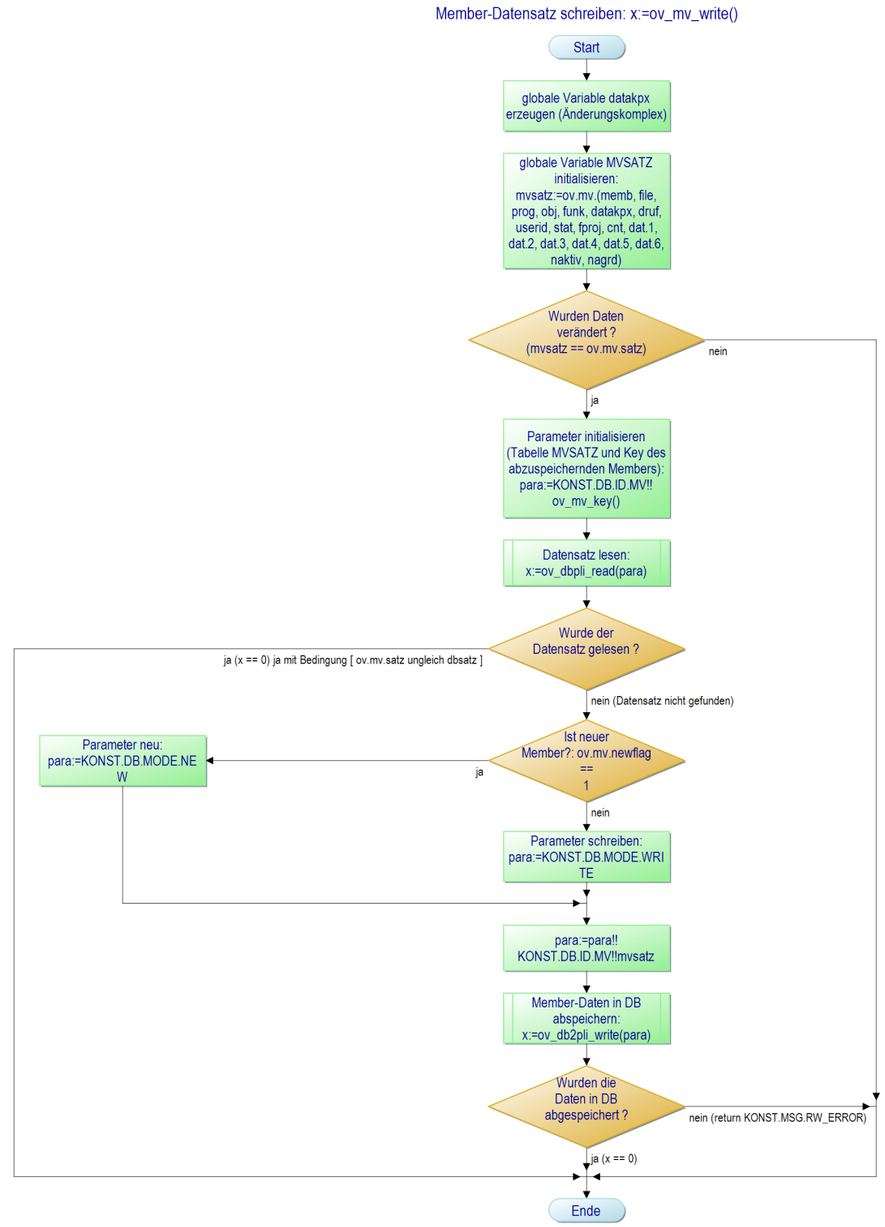


Abbildung C.2: Fallbeispiel – Programmablaufplan: ov_mv_write

C.3 Member-Verwaltung Panel: ov_mv_panel()

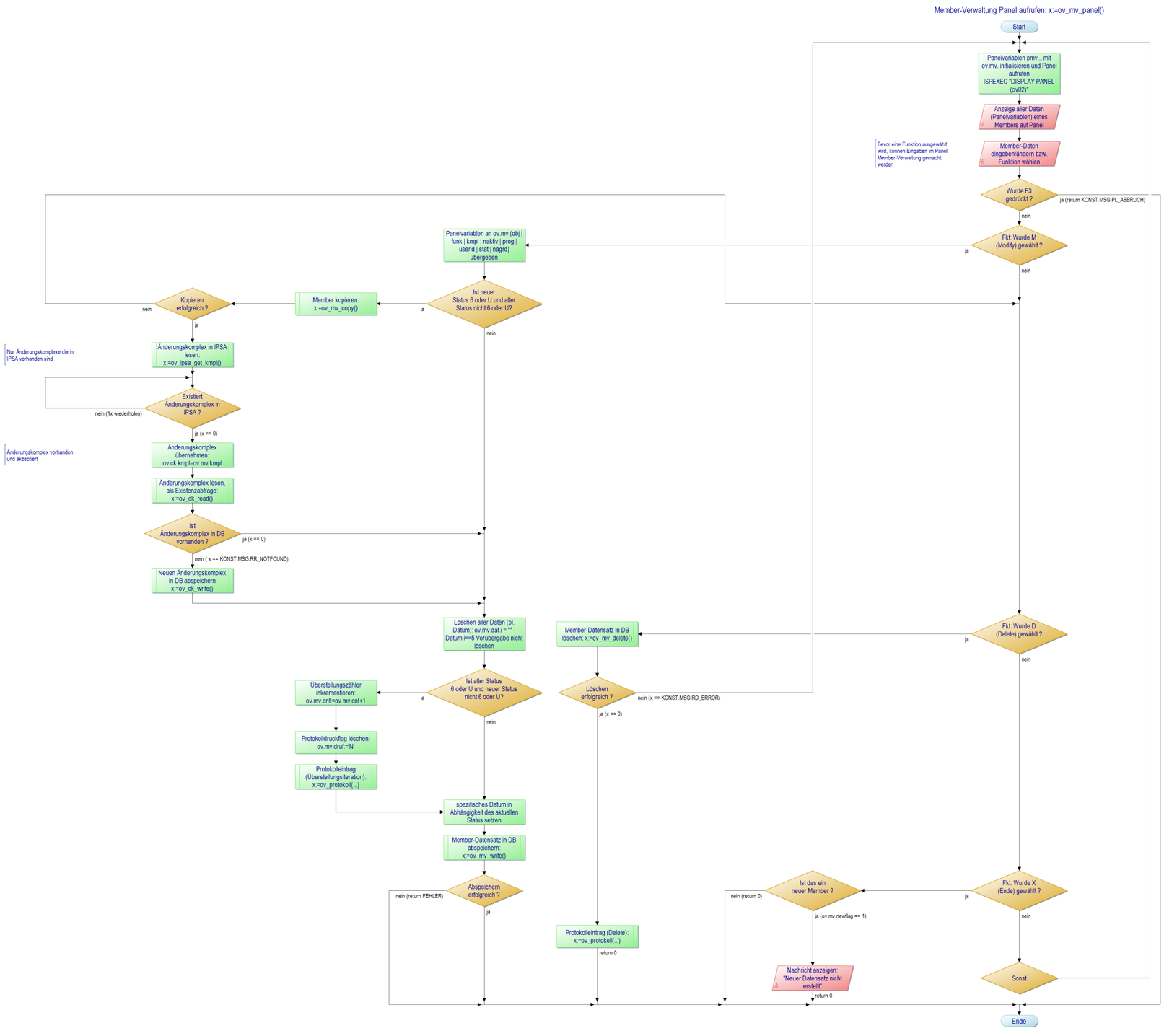


Abbildung C.3: Fallbeispiel - Programmablaufplan: ov_mv_panel

C.4 Hauptprogramm – Fkt: 4 (Änderungskomplexe)

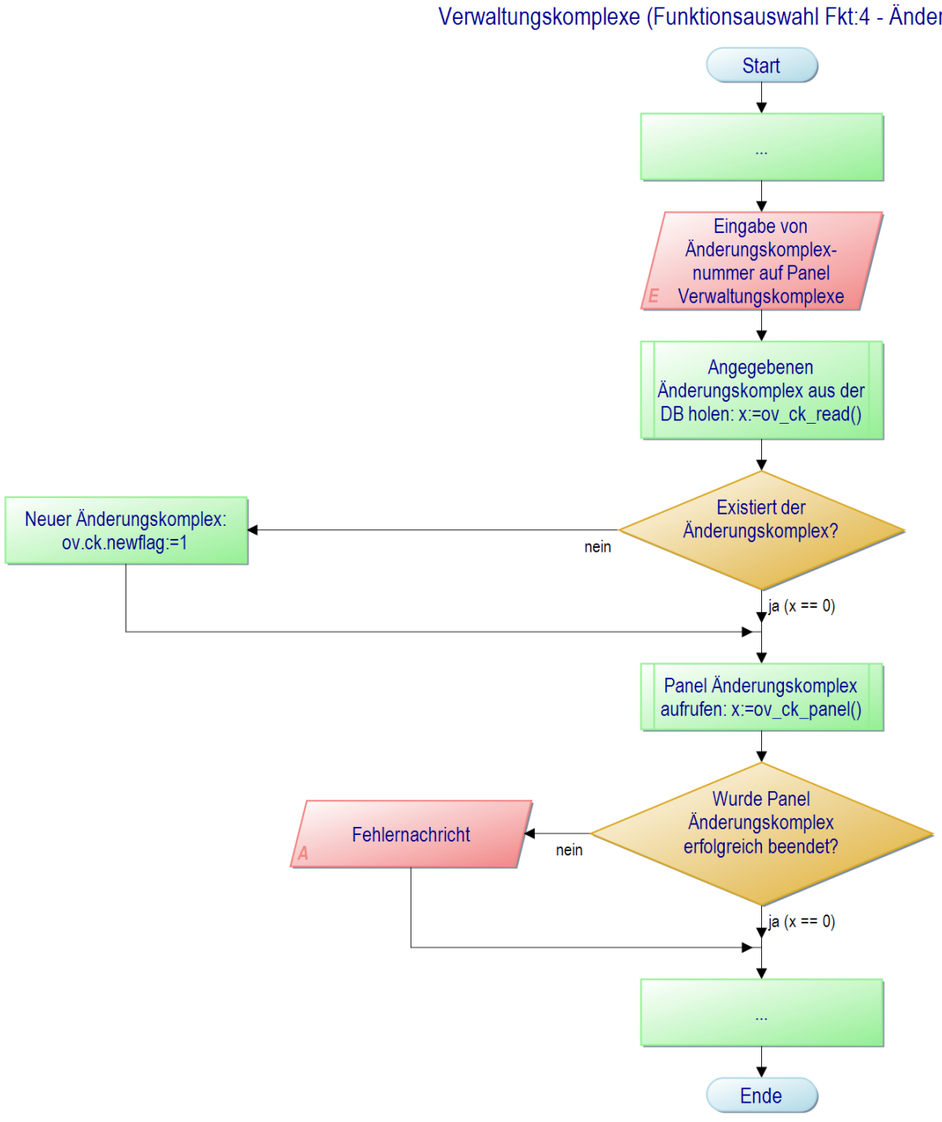


Abbildung C.5: Fallbeispiel: Programmablaufplan – Fkt. 4 (Änderungskomplexe)

C.5 Änderungskomplex schreiben: ov_ck_write()

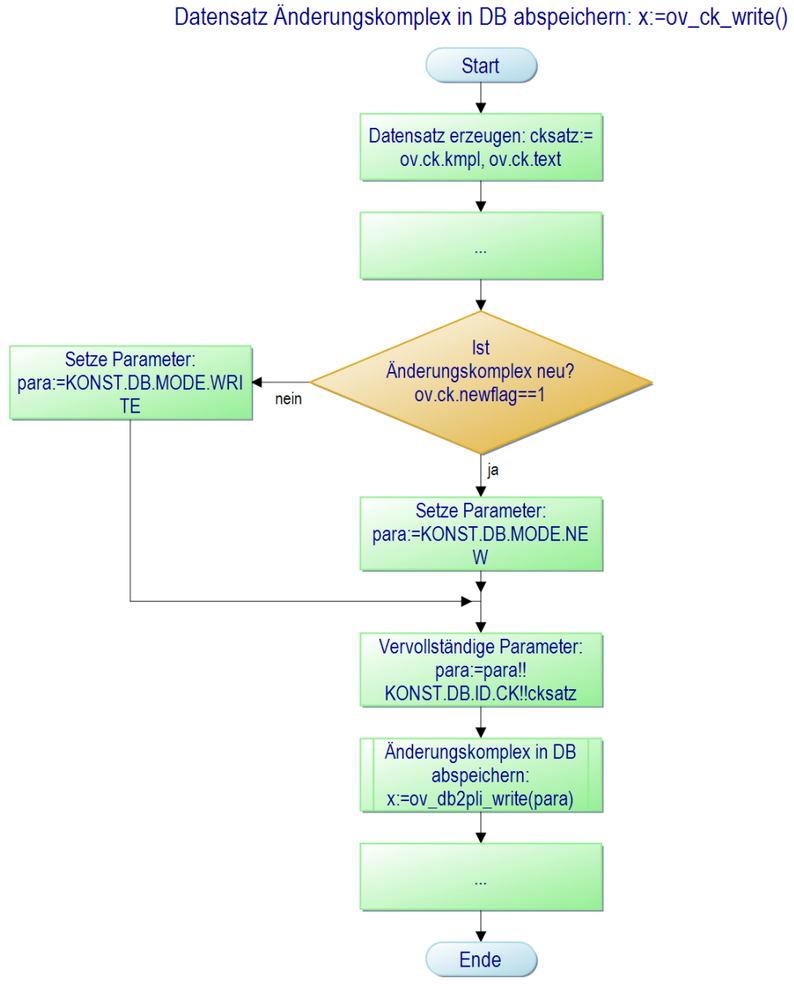


Abbildung C.4: Fallbeispiel – Programmablaufplan: ov_ck_write

C.6 Änderungskomplex Panel: ov_ck_panel()

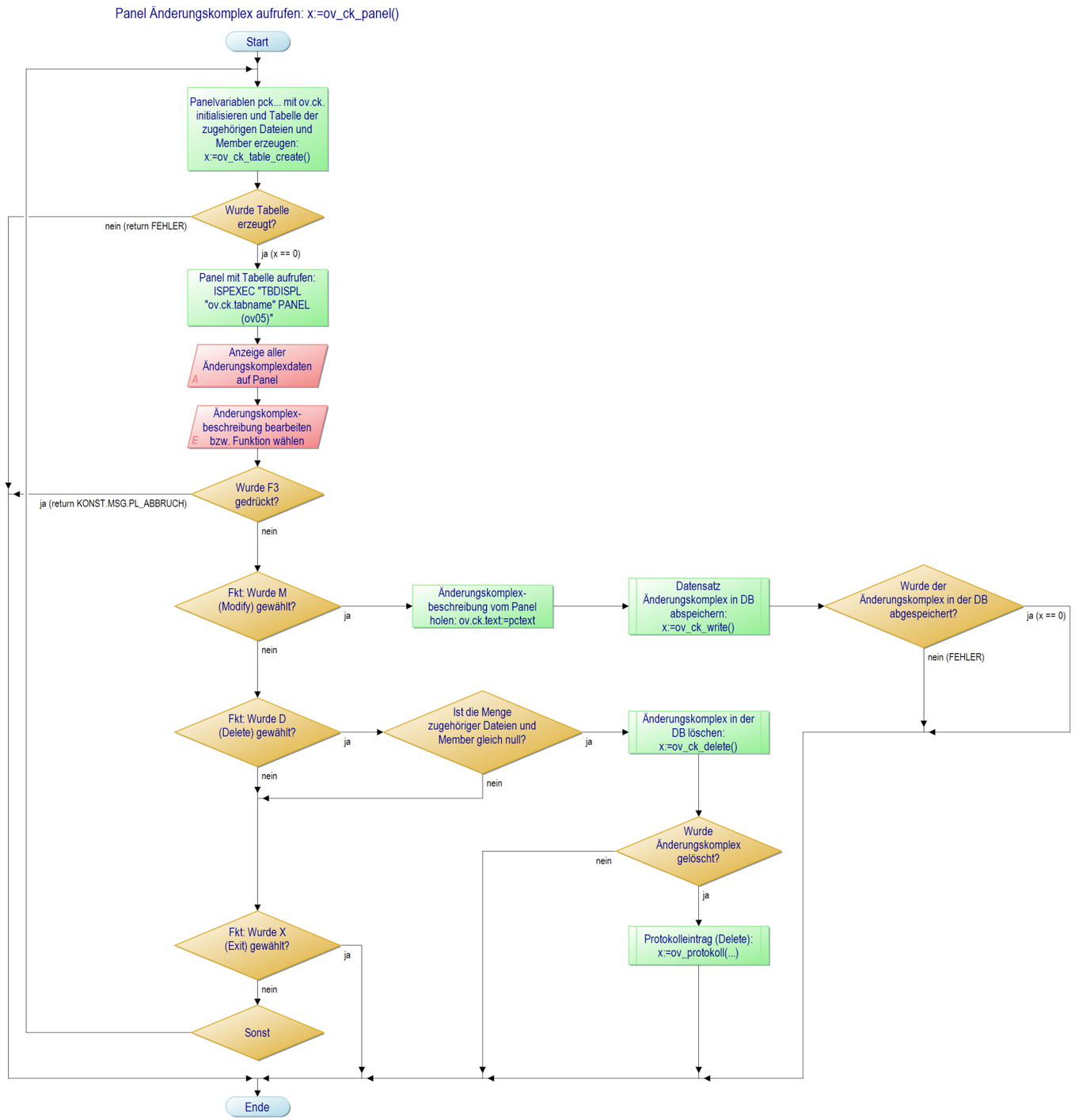


Abbildung C.6: Fallbeispiel – Programmablaufplan: ov_ck_panel

D Anforderungen des Fallbeispiels

D.1 Verfeinerung – Anwendungsfälle

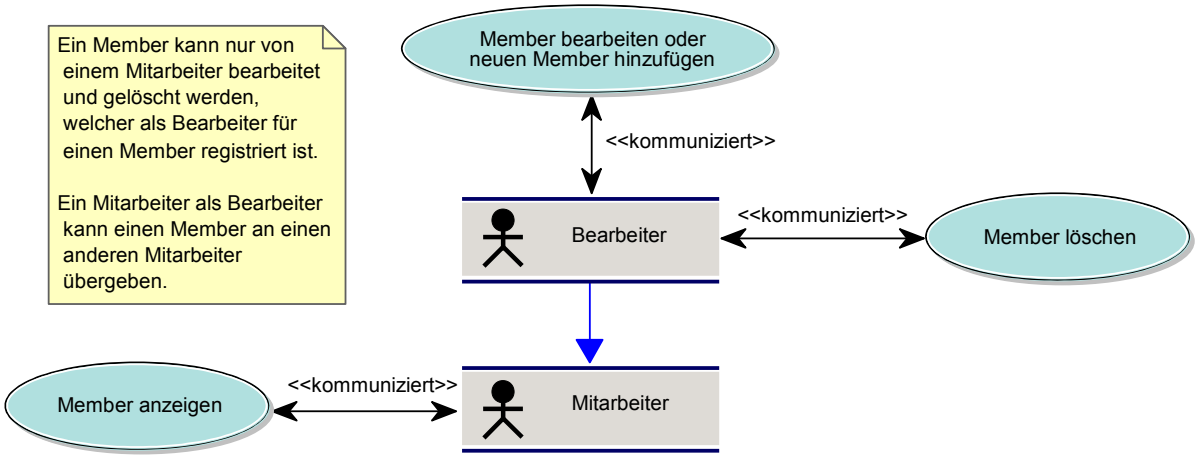


Abbildung D.1: Fallbeispiel: Anwendungsfall – Member verwalten

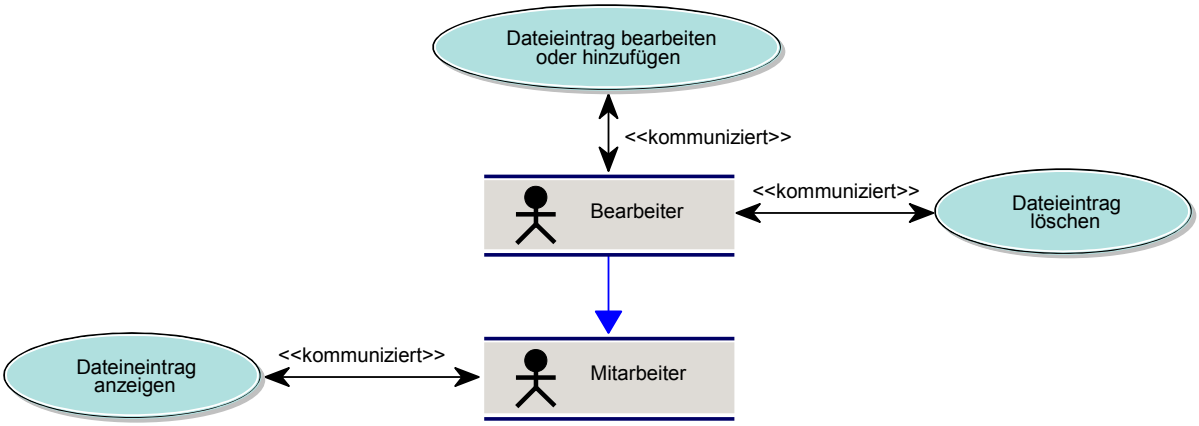


Abbildung D.2: Fallbeispiel: Anwendungsfall – Dateien verwalten

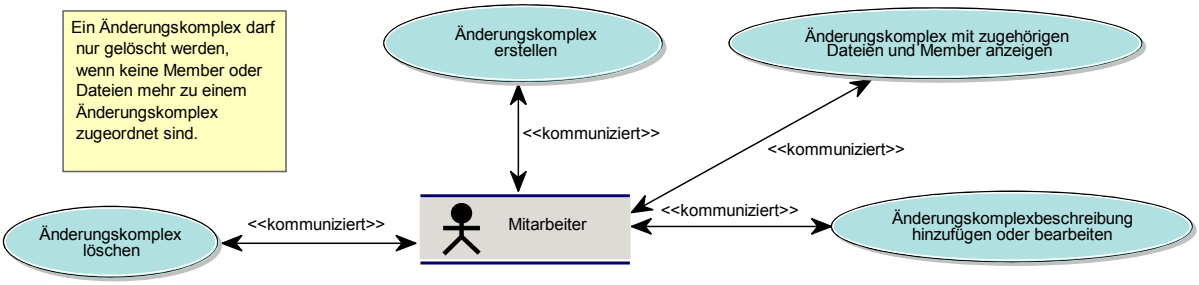


Abbildung D.3: Fallbeispiel: Anwendungsfall – Änderungskomplexe verwalten

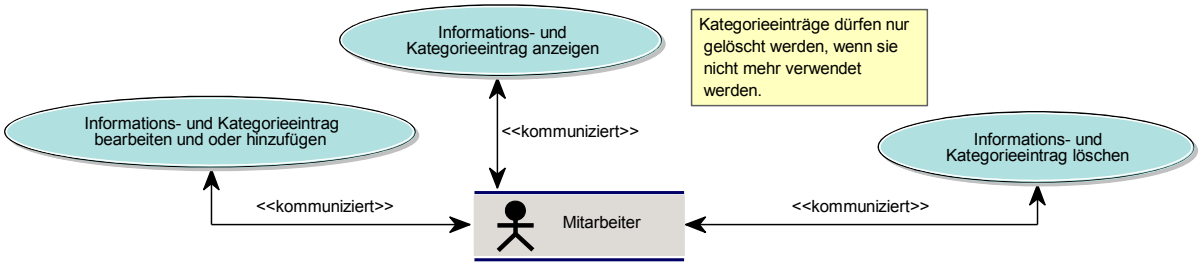


Abbildung D.4: Fallbeispiel: Anwendungsfall – Inf.- und Kategorieeinträge verwalten

D.2 Aktivitätsdiagramme

D.2.1 Änderungskomplexe verwalten

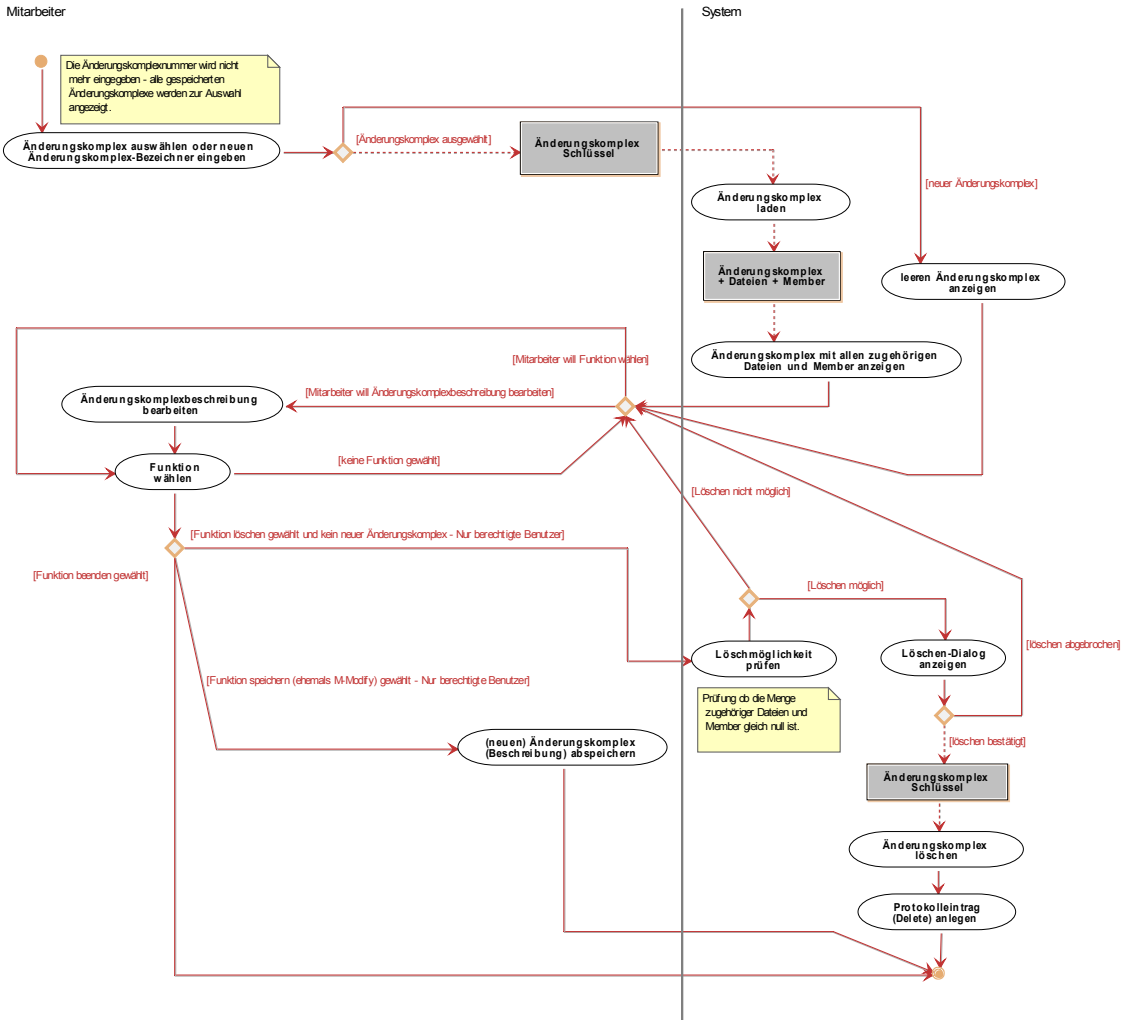


Abbildung D.5: Fallbeispiel: Aktivitätsdiagramm – Änderungskomplexe verwalten

D.2.2 Member verwalten

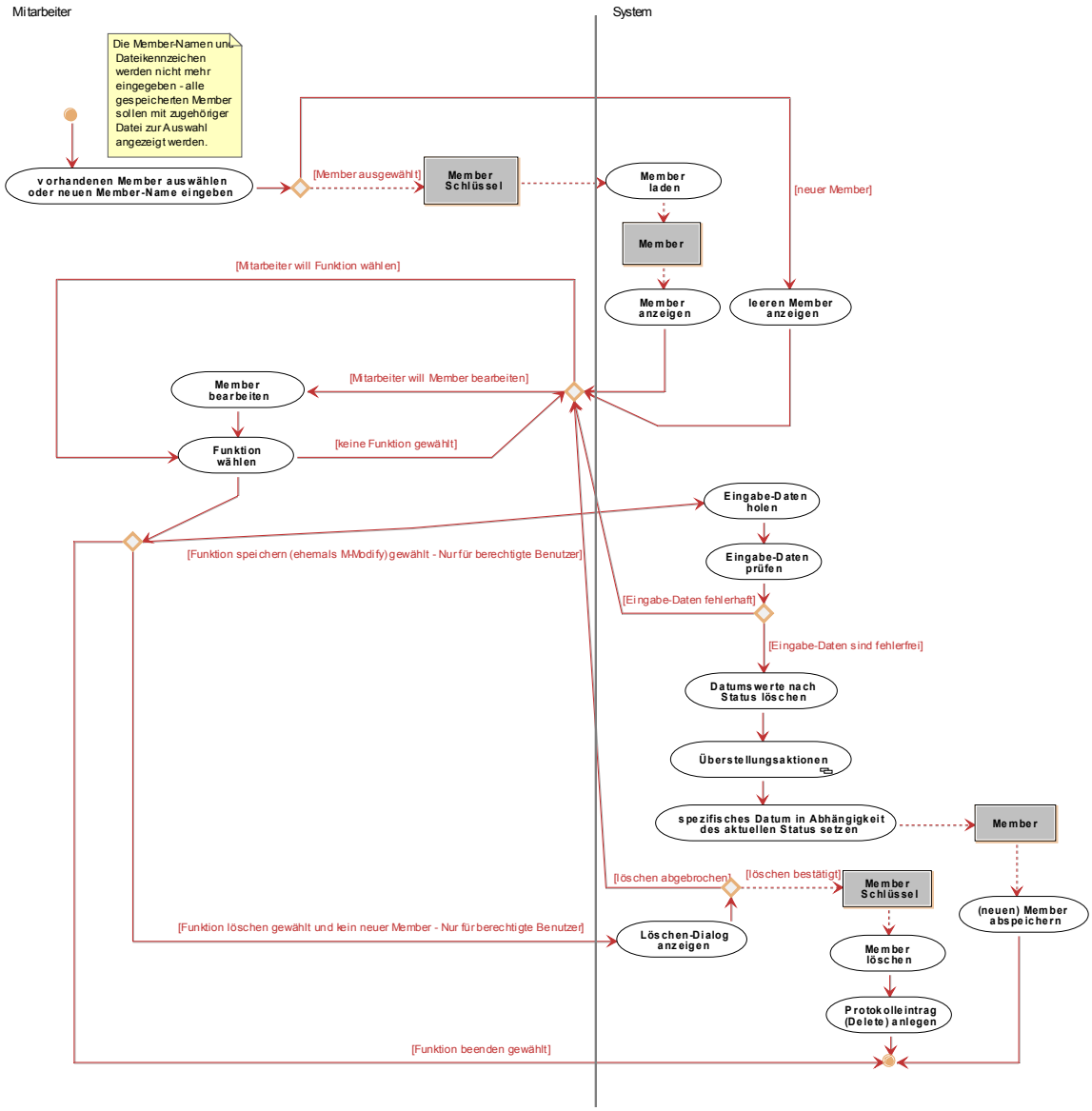


Abbildung D.6: Fallbeispiel: Aktivitätsdiagramm – Member verwalten

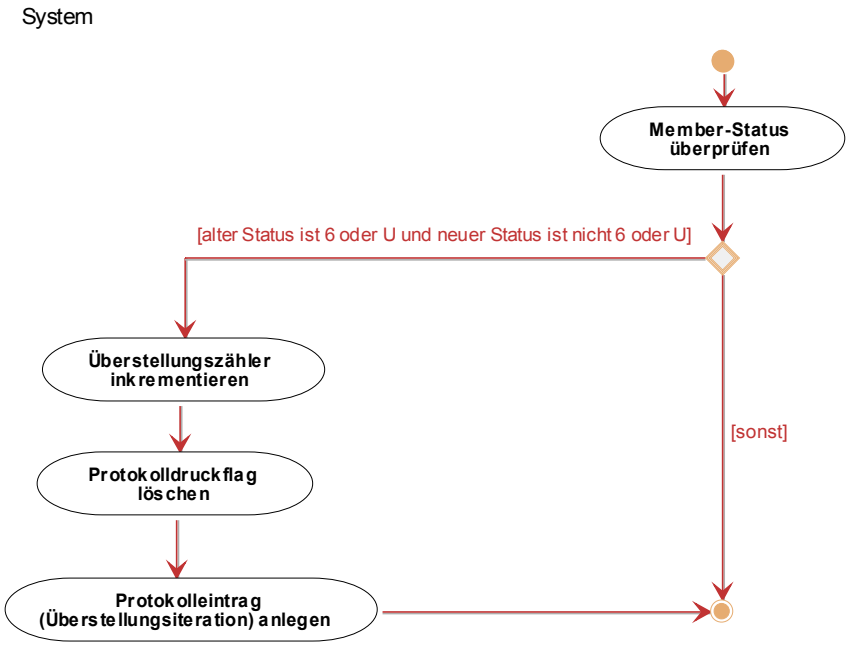


Abbildung D.7: Fallbeispiel: Akt.-Diag. – Verfeinerung „Überstellungsaktionen“

D.3 Allgemeine Beschreibung der Anwendungsfälle

Tabelle D.1: Anwendungsfallbeschreibung: Member anzeigen

1	**Titel des Anwendungsfalles**	**Member anzeigen**
2	**Ziel**	Das Ziel ist die Anzeige aller Informationen über einen vorhandenen Member.
3	**Vorbedingung**	Die zur Auswahl stehenden Verwaltungskomplexe werden angezeigt.
4 **4.1**	**Nachbedingung Erfolg**	Alle Informationen über einen gewählten Member werden in der Member-Verwaltung angezeigt.
4.2	**Nachbedingung Fehlschlag**	Der gewählte Member (-Name) existiert nicht. Die Member-Verwaltung wird nicht aufgerufen.
5	**Akteure**	Mitarbeiter, Bearbeiter
6	**Auslösendes Ereignis**	In der Verwaltungskomplexanzeige wurde der Name des Members gewählt und die Funktion Member-Verwaltung wurde ausgewählt.
7	**Eingabedaten**	Name des Members (zusätzlich zugehöriger Dateiname) (MVSATZ: MVMEMB, MVFILE)
8 **8.1**	**Beschreibung (Standardablauf)**	1 Ein Mitarbeiter wählt in der Verwaltungskomplexanzeige einen vorhandenen Member 2 Ein Mitarbeiter wählt den Verwaltungskomplex Member-Verwaltung aus. 3 Die Informationen über den Member werden in der Member-Verwaltung angezeigt
8.2	**Erweiterungen**	1a Zur eindeutigen Bestimmung des Members kann auch der zugehörige Dateiname mit angegeben werden.
8.3	**Alternativen**	3a Es werden keine Informationen über den Member angezeigt, weil der gewählte Member nicht existiert. Die Member-Verwaltung wird nicht angezeigt.

Tabelle D.2: Anwendungsfallbeschreibung: Member bearbeiten/hinzufügen

1	**Titel des Anwendungsfalles**	**Member bearbeiten oder neuen Member hinzufügen**
2	**Ziel**	Ziel ist das Bearbeiten oder das Hinzufügen eines Members.
3	**Vorbedingung**	Ein existierender Member wurde in der Verwaltungskomplex-anzeige gewählt oder ein neuer Member-Name wurde eingegeben. Alle Informationen über einen existierenden Member werden in der Member-Verwaltung angezeigt. Bei einem neuen Member wird die Member-Verwaltung ohne Informationen angezeigt. Ein existierender oder neuer Member, ohne einen registrierten Bearbeiter, kann beim ersten Bearbeiten von einem Mitarbeiter bearbeitet werden, welcher automatisch als Bearbeiter für den Member registriert wird. Nur dieser Mitarbeiter kann in der Rolle als Bearbeiter den Member bearbeiten; eine Übergabe zu einem anderen Mitarbeiter ist möglich.
4 **4.1**	**Nachbedingung Erfolg**	Die Eingabeprüfung ist erfolgreich. Alle Informationen eines Members wurden abgespeichert. Bei einem neuen Member wird der Member mit allen Informationen neu angelegt.
4.2	**Nachbedingung Fehlschlag**	Die Eingabeprüfung ist nicht erfolgreich. Die Informationen eines Members werden nicht abgespeichert. Der neue Member wird nicht angelegt.
5	**Akteure**	Bearbeiter
6	**Auslösendes Ereignis**	Ein Bearbeiter modifiziert bzw. bearbeitet die Informationen eines Members und wählt die Funktion speichern aus.
7	**Eingabedaten**	MVSATZ: MVMEMB, MVFILE, MVPROG, MVOBJ, MVFUNK, MVKMPL1, MVDRUF, MVSTAT, MVFPROJ, MVNAKTIV, MVNAGRD Folgende Daten werden automatisch vom System eingegeben: MVKMPL2-4, MVUSERID, MVCNT, MVDAT1-6, MVUSER, MVTIMEST
8 **8.1**	**Beschreibung (Standardablauf)**	1 Ein Mitarbeiter in der Rolle eines Bearbeiters bearbeitet die Informationen eines Members. 2 Ein Bearbeiter wählt die Funktion speichern aus 3 Die Eingabedaten werden auf Gültigkeit geprüft 4 In Abhängigkeit eines Status werden Datumswerte gelöscht 5 Überstellungsaktionen (siehe Aktivitätsdiagramm Abb. D.7) 6 Datumswerte in Abhängigkeit des aktuellen Status setzen 7 Alle Informationen des Members werden abgespeichert.
8.2	**Erweiterungen**	-
8.3	**Alternativen**	3a Die Gültigkeitsprüfung schlug fehl und der Speichervorgang bricht ab 7a Der Member existiert nicht; die Funktion speichern legt einen neuen Member an

Tabelle D.3: Anwendungsfallbeschreibung: Member löschen

1	**Titel des Anwendungsfalles**	**Member löschen**
2	**Ziel**	Ziel ist das Löschen eines Members.
3	**Vorbedingung**	Ein existierender Member wurde gewählt und alle Informationen über diesen werden in der Member-Verwaltung angezeigt.
4 **4.1**	**Nachbedingung Erfolg**	Der Member wurde gelöscht und die Verwaltungskomplexe werden angezeigt.
4.2	**Nachbedingung Fehlschlag**	-
5	**Akteure**	Bearbeiter
6	**Auslösendes Ereignis**	Ein Bearbeiter möchte den Member, dessen Informationen angezeigt werden, löschen und wählt die Funktion löschen aus.
7	**Eingabedaten**	-
8 **8.1**	**Beschreibung (Standardablauf)**	1 Ein Bearbeiter wählt die Funktion löschen aus. 2 Der Member wird gelöscht. 3 Der Löschvorgang wird protokolliert 4 Die Verwaltungskomplexe werden angezeigt.
8.2	**Erweiterungen**	-
8.3	**Alternativen**	-

Tabelle D.4: Anwendungsfallbeschreibung: Änderungskomplex anzeigen

1	**Titel des Anwendungsfalles**	**Änderungskomplex anzeigen**
2	**Ziel**	Ziel ist die Anzeige der Beschreibung eines Änderungskomplexes und die Anzeige aller Member und Dateien die zu einem Änderungskomplex gehören.
3	**Vorbedingung**	Die zur Auswahl stehenden Verwaltungskomplexe werden angezeigt.
4 **4.1**	**Nachbedingung Erfolg**	Die Beschreibung und alle zugehörigen Member und Dateien eines Änderungskomplexes werden angezeigt.
4.2	**Nachbedingung Fehlschlag**	Eine gewählte Änderungskomplexbezeichnung existiert nicht. Es werden keine Informationen angezeigt.
5	**Akteure**	Mitarbeiter
6	**Auslösendes Ereignis**	Die Änderungskomplexbezeichnung wurde gewählt und die Funktion Änderungskomplex wurde ausgewählt.

▼

7	**Eingabedaten**	Änderungskomplexbezeichnung (CKSATZ: CKKMPL)
8 **8.1**	**Beschreibung (Standardablauf)**	1 Ein Mitarbeiter wählt eine Änderungskomplexbezeichnung. 2 Der Mitarbeiter wählt den Verwaltungskomplex Änderungskomplex aus. 3 Die Informationen über den Änderungskomplex werden in der Änderungskomplexanzeige angezeigt.
8.2	**Erweiterungen**	-
8.3	**Alternativen**	3a Es werden keine Informationen über den Änderungskomplex angezeigt, weil die eingegebene Änderungskomplexbezeichnung nicht existiert.

Tabelle D.5: Anwendungsfallbeschreibung: Änderungskomplex löschen

1	**Titel des Anwendungsfalles**	**Änderungskomplex löschen**
2	**Ziel**	Ziel ist das Löschen eines Änderungskomplexes.
3	**Vorbedingung**	Ein existierender Änderungskomplex wurde gewählt und alle Informationen über diesen werden in der Änderungskomplexanzeige angezeigt.
4 **4.1**	**Nachbedingung Erfolg**	Der Änderungskomplex wurde gelöscht und die Verwaltungskomplexe werden angezeigt.
4.2	**Nachbedingung Fehlschlag**	Der Änderungskomplex kann nicht gelöscht werden. Es wird eine Mitteilung ausgegeben und anschließend wird zur Änderungskomplexanzeige zurückgekehrt.
5	**Akteure**	Mitarbeiter
6	**Auslösendes Ereignis**	Ein Mitarbeiter möchte den Änderungskomplex, dessen Informationen angezeigt werden, löschen und wählt die Funktion löschen aus.
7	**Eingabedaten**	-
8 **8.1**	**Beschreibung (Standardablauf)**	1 Ein Mitarbeiter wählt die Funktion löschen aus. 2 Prüfung der Löschmöglichkeit des Änderungskomplexes 3 Der Änderungskomplex wird gelöscht. 4 Der Löschvorgang wird in einer Protokolldatei vermerkt 5 Die Verwaltungskomplexe werden angezeigt.
8.2	**Erweiterungen**	-
8.3	**Alternativen**	2a Die Prüfung ergab, dass dem Änderungskomplex noch Member bzw. Dateien zugehörig sind. Der Änderungskomplex darf nicht gelöscht werden.

Tabelle D.6: Anwendungsfallbeschreibung: Änderungskomplex erstellen

1	**Titel des Anwendungsfalles**	**Änderungskomplex erstellen**
2	**Ziel**	Ziel ist das Erstellen eines Änderungskomplexes.
3	**Vorbedingung**	Eine neue Änderungskomplexbezeichnung wurde auf der Verwaltungskomplexanzeige eingegeben und die Änderungskomplexanzeige wird angezeigt.
4 **4.1**	**Nachbedingung Erfolg**	Ein neuer Änderungskomplex wurde angelegt.
4.2	**Nachbedingung Fehlschlag**	-
5	**Akteure**	Mitarbeiter
6	**Auslösendes Ereignis**	Ein Mitarbeiter wählt die Funktion speichern.
7	**Eingabedaten**	Änderungskomplexbezeichner (CKSATZ: CKKMPL), Beschreibung des Änderungskomplexes (CKSATZ: CKTEXT)
8 **8.1**	**Beschreibung (Standardablauf)**	1 Ein Mitarbeiter gibt eine Änderungskomplexbeschreibung ein. 2 Ein Mitarbeiter wählt die Funktion speichern aus. 3 Überprüfung ob Änderungskomplex bereits existiert 4 Der Änderungskomplex wird mit allen Informationen neu angelegt.
8.2	**Erweiterungen**	-
8.3	**Alternativen**	-

Tabelle D.7: Anwendungsfallbeschreibung: Äkmplx. hinzufügen/bearbeiten

1	**Titel des Anwendungsfalles**	**Änderungskomplexbeschreibung hinzufügen oder bearbeiten**
2	**Ziel**	Ziel ist die Bearbeitung der Beschreibung eines Änderungskomplexes.
3	**Vorbedingung**	Ein existierender Änderungskomplex wurde gewählt und alle Informationen über diesen werden in der Änderungskomplexanzeige angezeigt.
4 **4.1**	**Nachbedingung Erfolg**	Die Beschreibung des Änderungskomplexes wurde geändert und abgespeichert.
4.2	**Nachbedingung Fehlschlag**	-
5	**Akteure**	Mitarbeiter

▼

6	**Auslösendes Ereignis**	Ein Mitarbeiter bearbeitet die Beschreibung eines angezeigten Änderungskomplexes und wählt die Funktion speichern aus.
7	**Eingabedaten**	Beschreibung des Änderungskomplexes (CKSATZ: CKTEXT)
8 **8.1**	**Beschreibung (Standardablauf)**	1 Ein Mitarbeiter bearbeitet die Beschreibung des Änderungskomplexes. 2 Der Mitarbeiter wählt die Funktion speichern aus. 3 Die bearbeitete Beschreibung des Änderungskomplexes wird abgespeichert.
8.2	**Erweiterungen**	-
8.3	**Alternativen**	-

Tabelle D.8: Anwendungsfallbeschreibung: Dateieintrag anzeigen

1	**Titel des Anwendungsfalles**	**Dateieintrag anzeigen**
2	**Ziel**	Ziel ist die Anzeige aller Informationen über einen vorhandenen Dateieintrag.
3	**Vorbedingung**	Die zur Auswahl stehenden Verwaltungskomplexe werden angezeigt.
4 **4.1**	**Nachbedingung Erfolg**	Alle Informationen über die Datei werden in der Dateien-Verwaltung angezeigt.
4.2	**Nachbedingung Fehlschlag**	Der gewählte Dateiname existiert nicht. Es werden keine Informationen in der Dateien-Verwaltung angezeigt.
5	**Akteure**	Mitarbeiter, Bearbeiter
6	**Auslösendes Ereignis**	In der Verwaltungskomplexanzeige wurde der Dateiname gewählt und die Funktion Dateiverwaltung wurde ausgewählt.
7	**Eingabedaten**	Dateiname
8 **8.1**	**Beschreibung (Standardablauf)**	1 Ein Mitarbeiter gibt in der Verwaltungskomplexanzeige einen Dateinamen ein 2 Ein Mitarbeiter wählt den Verwaltungskomplex Dateiverwaltung aus. 3 Die Informationen über die Datei werden in der Dateiverwaltung angezeigt.
8.2	**Erweiterungen**	-
8.3	**Alternativen**	3a Es werden keine Informationen über die Datei angezeigt, weil der eingegebene Dateiname nicht existiert. Die Dateien-Verwaltung wird nicht angezeigt.

Tabelle D.9: Anwendungsfallbeschreibung: Dateieintrag bearbeiten/hinzufügen

1	**Titel des Anwendungsfalles**	**Dateieintrag bearbeiten oder hinzufügen**
2	**Ziel**	Ziel ist die Bearbeitung der Informationen über eine Datei oder das Hinzufügen eines neuen Dateieintrages. Wie im Anwendungsfall Member bearbeiten oder hinzufügen gilt hier auch der Rollenübergang eines Mitarbeiters.
3	**Vorbedingung**	Ein existierender Dateieintrag wurde gewählt und alle Informationen über diesen werden in der Dateiverwaltung angezeigt.
4 **4.1**	**Nachbedingung Erfolg**	Die Eingabeprüfung ist erfolgreich. Die Informationen über eine Datei wurden abgespeichert.
4.2	**Nachbedingung Fehlschlag**	Die Eingabeprüfung ist nicht erfolgreich. Die Informationen über eine Datei werden nicht abgespeichert.
5	**Akteure**	Bearbeiter
6	**Auslösendes Ereignis**	Ein Bearbeiter modifiziert bzw. bearbeitet die Informationen des Dateieintrages und wählt die Funktion speichern aus.
7	**Eingabedaten**	DVSATZ: DVFILE, DVTYPE, DVFUNK, DVKMPL1, DVDRUF, DVSTAT Folgende Daten werden automatisch vom System eingegeben: DVKMPL2-4, DVUSERID, DVCNT, DVDAT1-4 DVUSER, DVTIMEST
8 **8.1**	**Beschreibung (Standardablauf)**	1 Ein Bearbeiter bearbeitet die Informationen über eine Datei. 2 Ein Bearbeiter wählt die Funktion speichern aus. 3 Die Eingabedaten werden auf Gültigkeit geprüft. 4 Alle Informationen über eine Datei werden abgespeichert.
8.2	**Erweiterungen**	-
8.3	**Alternativen**	4a Die Gültigkeitsprüfung schlug fehl und die Informationen über die Datei werden nicht abgespeichert 4b Die Gültigkeitsprüfung war erfolgreich. Der Dateieintrag existiert noch nicht und wird als neuer Dateieintrag abgespeichert.

Tabelle D.10: Anwendungsfallbeschreibung: Dateieintrag löschen

1	**Titel des Anwendungsfalles**	**Dateieintrag löschen**
2	**Ziel**	Ziel ist die Löschung eines Dateieintrages.
3	**Vorbedingung**	Ein existierender Dateieintrag wurde ausgewählt und alle Informationen über diesen werden in der Dateien-Verwaltung angezeigt.
4 **4.1**	**Nachbedingung Erfolg**	Der Dateieintrag wurde gelöscht und die Verwaltungskomplexe werden angezeigt.
4.2	**Nachbedingung Fehlschlag**	-
5	**Akteure**	Bearbeiter
6	**Auslösendes Ereignis**	Ein Bearbeiter möchte den Dateieintrag löschen und wählt die Funktion löschen aus.
7	**Eingabedaten**	-
8 **8.1**	**Beschreibung (Standardablauf)**	1 Ein Bearbeiter wählt die Funktion löschen aus. 2 Der Dateieintrag wird gelöscht. 3 Der Löschvorgang wird protokolliert. 4 Die Verwaltungskomplexe werden angezeigt.
8.2	**Erweiterungen**	-
8.3	**Alternativen**	-

Tabelle D.11: Anwendungsfallbeschreibung: Inf.- und Kategorieeintrag anzeigen

1	**Titel des Anwendungsfalles**	**Informations- und Kategorieeintrag anzeigen**
2	**Ziel**	Ziel ist die Anzeige der allgemeinen Beschreibung bzw. Deutung der Kennzeichentypen bzw. Kennzeichen über ein Objekt.
3	**Vorbedingung**	Die zur Auswahl stehenden Verwaltungskomplexe werden angezeigt.
4 **4.1**	**Nachbedingung Erfolg**	Kennzeichentyp mit Kennzeichen und Beschreibung werden angezeigt.
4.2	**Nachbedingung Fehlschlag**	Der gewählte Kennzeichentyp oder das Kennzeichen existiert nicht. Es werden keine Informationen angezeigt.
5	**Akteure**	Mitarbeiter
6	**Auslösendes Ereignis**	Es wurde ein Kennzeichentyp und oder ein Kennzeichen eingegeben und die Funktion INFO-Tabelle verwalten wurde ausgewählt.
7	**Eingabedaten**	Kennzeichentyp, Kennzeichen

▼

8 **8.1**	**Beschreibung (Standardablauf)**	1 Ein Mitarbeiter gibt Kennzeichentyp und oder Kennzeichen ein. 2 Ein Mitarbeiter wählt den Verwaltungskomplex INFO-Tabelle verwalten aus. 3 Die Informationen über den eingegebene Kennzeichentyp bzw. das Kennzeichen werden in der INFO-Tabelle angezeigt.
8.2	**Erweiterungen**	-
8.3	**Alternativen**	3a Es werden keine Informationen über das Kennzeichentyp bzw. Kennzeichen angezeigt, weil der eingegebene Kennzeichentyp bzw. das Kennzeichen nicht existiert. Die INFO-Tabelle wird nicht angezeigt.

Tabelle D.12: Anwendungsfallbeschreibung: Inf.- und Kategorieeintrag löschen

1	**Titel des Anwendungsfalles**	**Informations- und Kategorieeintrag löschen**
2	**Ziel**	Ziel ist das Löschen eines Kennzeichentyps bzw. Kennzeichens.
3	**Vorbedingung**	Ein existierender Kennzeichentyp bzw. ein Kennzeichen wurde ausgewählt und alle Informationen über dieses werden in der INFO-Tabelle angezeigt.
4 **4.1**	**Nachbedingung Erfolg**	Der Kennzeichentyp bzw. das Kennzeichen wurde gelöscht und die Verwaltungskomplexe werden angezeigt.
4.2	**Nachbedingung Fehlschlag**	-
5	**Akteure**	Mitarbeiter
6	**Auslösendes Ereignis**	Ein Mitarbeiter möchte ein Kennzeichentyp löschen und wählt die Funktion löschen aus.
7	**Eingabedaten**	-
8 **8.1**	**Beschreibung (Standardablauf)**	1 Ein Mitarbeiter wählt die Funktion löschen aus. 2 Der Kennzeichentyp wird gelöscht. 3 Die Verwaltungskomplexe werden angezeigt.
8.2	**Erweiterungen**	-
8.3	**Alternativen**	2a Der Kennzeichentyp kann nicht gelöscht werden, weil er verwendet wird

Tabelle D.13: Anwendungsfallbeschreibung: Inf.- und Kat. bearbeiten/hinzufügen

1	**Titel des Anwendungsfalles**	**Informations- und Kategorieeintrag bearbeiten oder hinzufügen**
2	**Ziel**	Ziel ist die Bearbeitung der Informationen über ein Kennzeichentyp oder das Hinzufügen eines neuen Kennzeichentyps.
3	**Vorbedingung**	Ein existierender Kennzeichentyp bzw. ein Kennzeichen wurde ausgewählt und alle Informationen über dieses werden in der INFO-Tabelle angezeigt.
4 **4.1**	**Nachbedingung Erfolg**	Die Informationen über ein Kennzeichentyp wurden abgespeichert.
4.2	**Nachbedingung Fehlschlag**	-
5	**Akteure**	Mitarbeiter
6	**Auslösendes Ereignis**	Ein Mitarbeiter modifiziert bzw. bearbeitet die Informationen eines Kennzeichentyps und wählt die Funktion speichern aus.
7	**Eingabedaten**	IDSATZ: IDKZTYP, IDKZ, IDTEXT
8 **8.1**	**Beschreibung (Standardablauf)**	1 Ein Mitarbeiter bearbeitet die Informationen über ein Kennzeichentyp. 2 Ein Mitarbeiter wählt die Funktion speichern aus. 3 Alle Informationen über ein Kennzeichentyp werden abgespeichert.
8.2	**Erweiterungen**	-
8.3	**Alternativen**	-

Tabelle D.14: Anwendungsfallbeschreibung: Statistiken, Ausw.- Überg.-Pr. erstellen

1	**Titel des Anwendungsfalles**	**Statistiken, Auswertungs- und Übergabeprotokolle erstellen**
2	**Ziel**	Ziel ist das Erstellen von Auswertungsprotokollen und Statistiken über den Verwaltungsinhalt der Objektverwaltung.
3	**Vorbedingung**	-
4 **4.1**	**Nachbedingung Erfolg**	Ein Auswertungsprotokoll oder eine Statistik wurde erstellt.
4.2	**Nachbedingung Fehlschlag**	-
5	**Akteure**	Mitarbeiter
6	**Auslösendes Ereignis**	Ein Mitarbeiter möchte eine Statistik oder ein Auswertungsprotokoll erstellen.
7	**Eingabedaten**	-

▼

8 **8.1**	**Beschreibung (Standardablauf)**	1 Ein Mitarbeiter wählt ein vordefiniertes Statistik- oder Auswertungsprotokoll. 2 Das Objektverwaltungssystem erstellt das Statistik- oder Auswertungsprotokoll aus der vorliegenden Datenbasis.
8.2	**Erweiterungen**	-
8.3	**Alternativen**	-

D.4 Glossar

Objekt	Objekt ist der Oberbegriff für alle zu verwaltenden Elemente in der Objektverwaltung. Ein Objekt kann unter anderem ein Member oder eine Datei sein.
Member	Ein Member bezeichnet ursprünglich unter anderem ein Programm, ein Makro, eine Copy-Strecke, REXX- oder CLIST-Prozeduren.
Member-Verwaltung	In der Member-Verwaltung werden alle Informationen behandelt, die den aktuellen Entwicklungsstand der Member beschreiben, welche in einem Projekt zu bearbeiten sind.
Dateiverwaltung	In der Dateiverwaltung werden alle nicht Quell-Code-Dateien verwaltet. Zum Beispiel sequenzielle Dateien oder Datenbanken.
Dateieintrag	Ein Dateieintrag ist die Menge aller Informationen über eine Datei, welche in der Dateiverwaltung verwaltet werden.
Änderungskomplex	Ein Änderungskomplex ist eine Gruppenart, deren Mitglieder Member und Dateien sind und eine gemeinsame Änderungseigenschaft haben oder in Bezug einer Änderung in Kausalität stehen.
Verwaltungskomplex	Ein Verwaltungskomplex ist ein Oberbegriff für Member-Verwaltung, Dateiverwaltung, Datenfeldverwaltung, Informations- und Kategorieeinträge (INFO-Tabelle), Änderungskomplex
Kennzeichentyp (KZ-Typ) und Kennzeichen (KZ)	Kennzeichen sind die ausgeschriebenen Kennzeichentypen (z.B.: Programmiersprache: KZ-Typ: C; KZ: COBOL, Dateiname: KZ-Typ: 03; KZ: ADA-DB3N) In der Objektverwaltung wurden für die zu verwaltenden Objekte Kürzel verwendet, weil dadurch Platz auf den Bildschirmmasken eingespart und die Eingabe vereinfacht werden konnte.
INFO-Tabelle	Die INFO-Tabelle (Informations- und Kategorieeinträge) dient zur allgemeinen Beschreibung und Kontrolle der in der Objektverwaltung verwendeten Kennzeichentypen bzw. Kennzeichen.
Mitarbeiter bzw. Bearbeiter	Der Akteur Mitarbeiter ist ein Benutzer des Objektverwaltungssystems, welcher in die Rolle eines Bearbeiters wechselt, wenn er unter anderem als Bearbeiter eines Members registriert ist.

Tabelle D.15: Fallbeispiel – Glossar

E Plattformunabhängiges Modell

E.1 Fachlogik (MyServices)

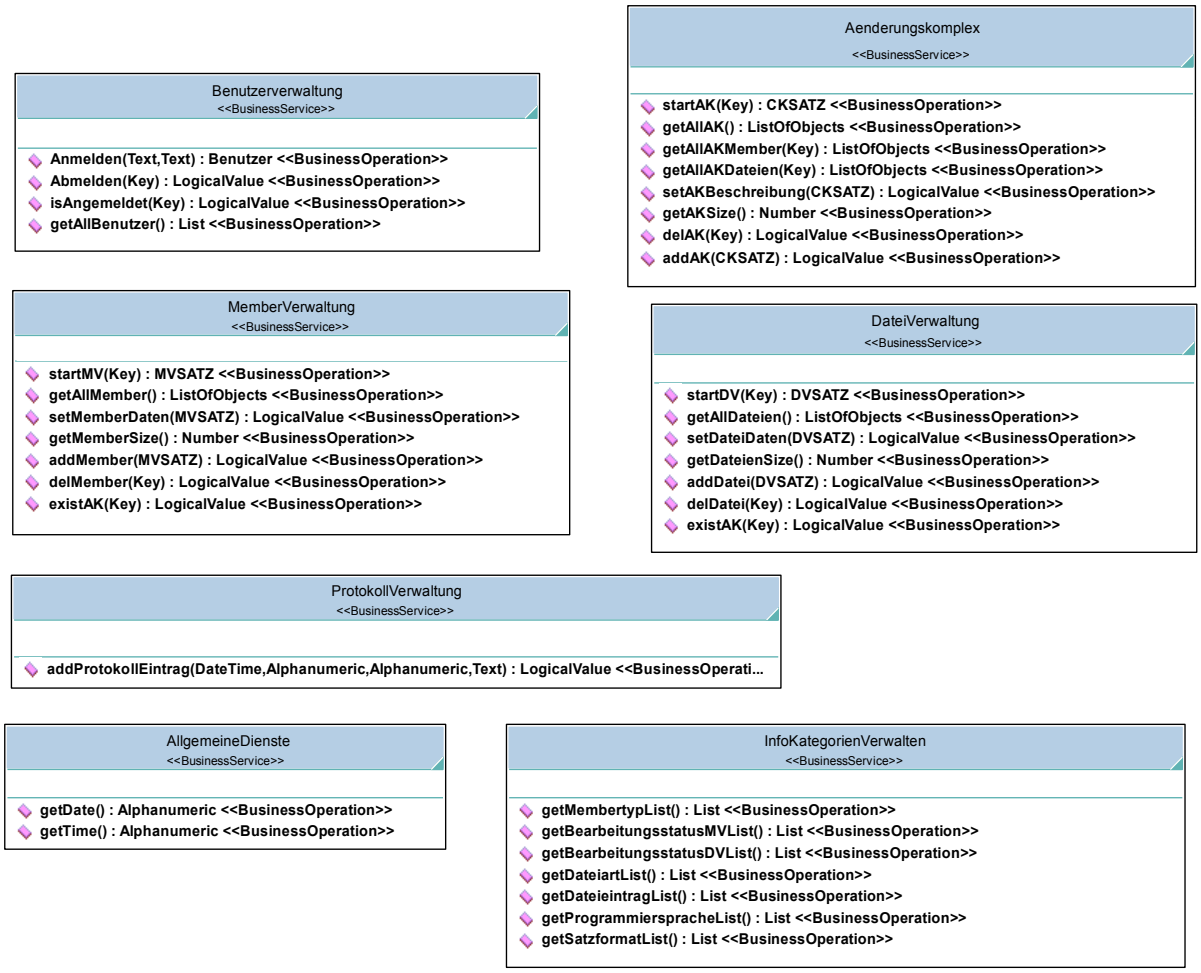


Abbildung E.1: Fallbeispiel – plattformunabhängiges Modell Dienste

E.2 Datenmodell (MyEntities)

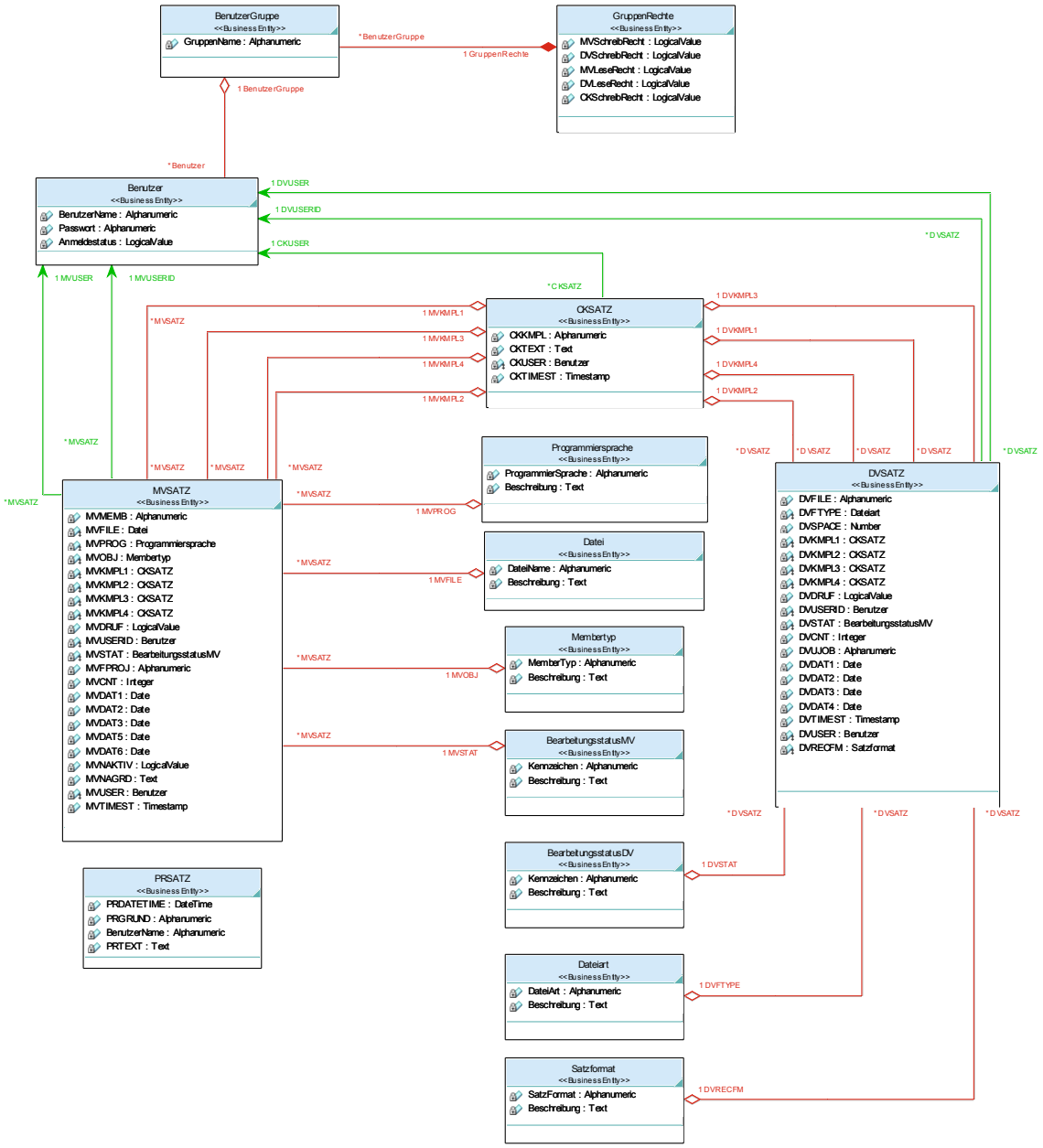


Abbildung E.2: Fallbeispiel – plattformunabhängiges Datenmodell

E.3 Oberflächennavigation (MyPresentation)

In Abbildung E.3 wird der Startzustand des Objektverwaltungssystems dargestellt. Als erstes wird eine JSP/JSF-Seite namens Benutzeranmeldung aufgerufen. Diese enthält einen Trigger (HTML-Knopf) namens *Anmelden* und löst die Aktivität *Benutzerdaten laden* aus. Hinter der Aktivität *Benutzerdaten laden* verbirgt sich der Aufruf einer Dienstoperation. Nach erfolgreicher Anmeldung, durch Überprüfung von Benutzername und Passwort, wird die JSP/JSF-Seite Verwaltungskomplexe in Abbildung E.4 aufgerufen. Die Abbildung E.4 ist die Verfeinerung des Zustandes Objektverwaltung.

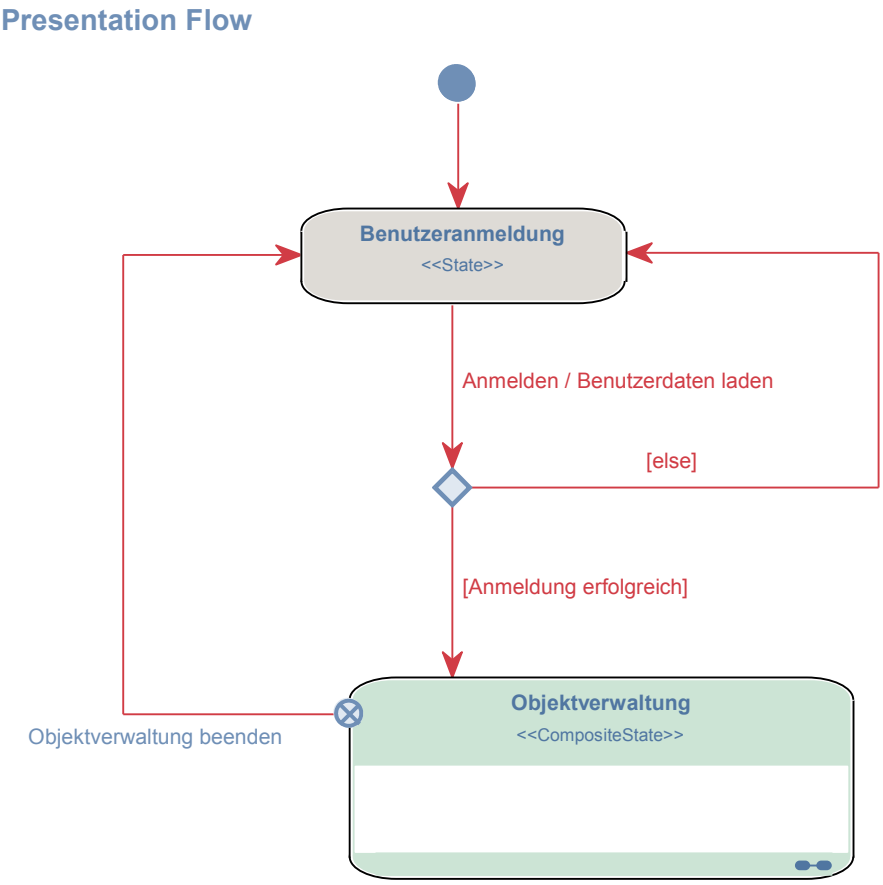


Abbildung E.3: Fallbeispiel – Oberflächennavigation - Benutzeranmeldung

Auf der Verwaltungskomplexseite ermöglichen die HTML-Knöpfe s*tartMV*, s*tartDV* und s*tartAK* den Aufruf der Verwaltungskomplexe Member-Verwaltung, Dateiverwaltung oder Änderungskomplexe, sofern unter anderem ein Member ausgewählt wurde. Über den HTML-Knopf *Abmelden* kann die Objektverwaltung verlassen werden, wodurch man zur Benutzeranmeldung zurückkehrt.

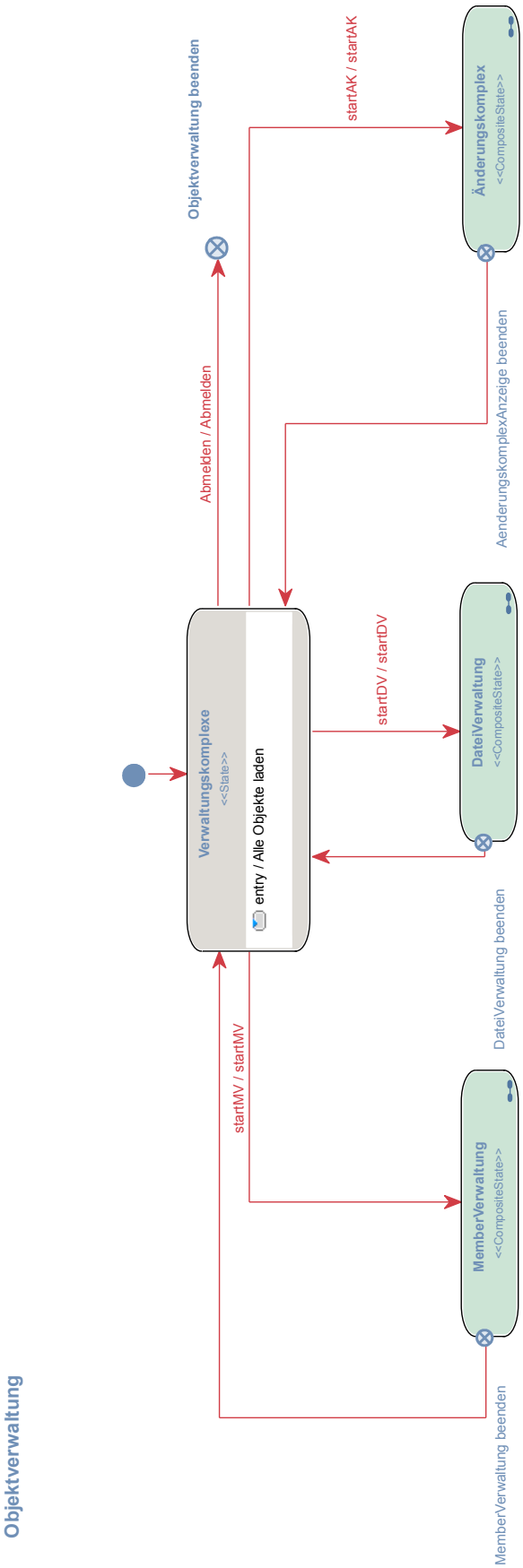


Abbildung E.4: Fallbeispiel – Verfeinerung von Zustand Objektverwaltung

Die folgenden Abbildungen E.5, E.6 und E.7 stellen die Oberflächennavigation der Member-Verwaltung, Datei-Verwaltung und Änderungskomplexe dar.

MemberVerwaltung

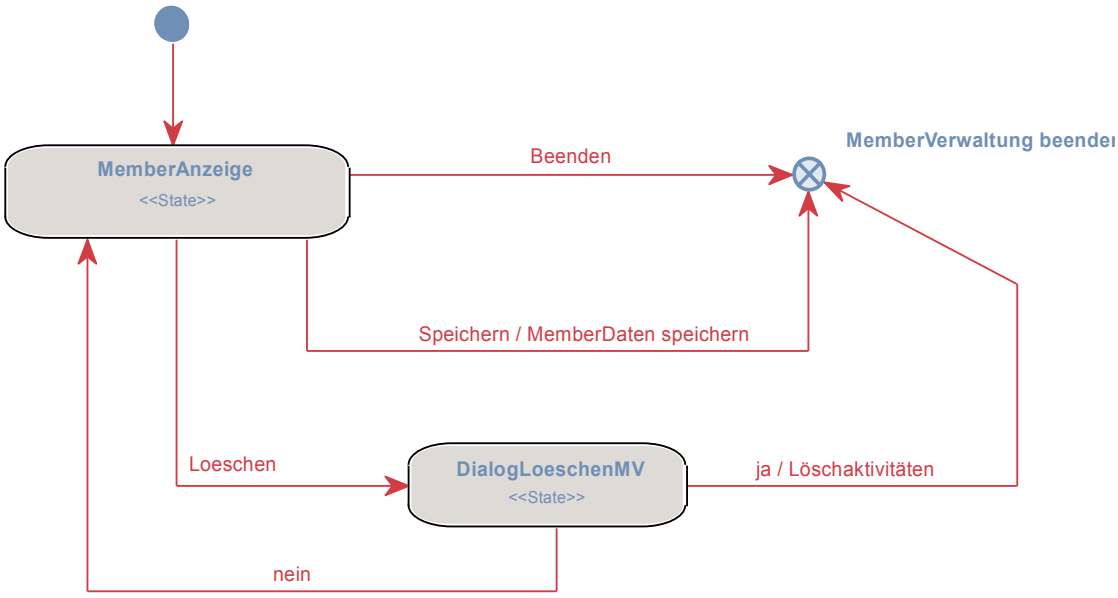


Abbildung E.5: Fallbeispiel – Verfeinerung von Zustand Member-Verwaltung

DateiVerwaltung

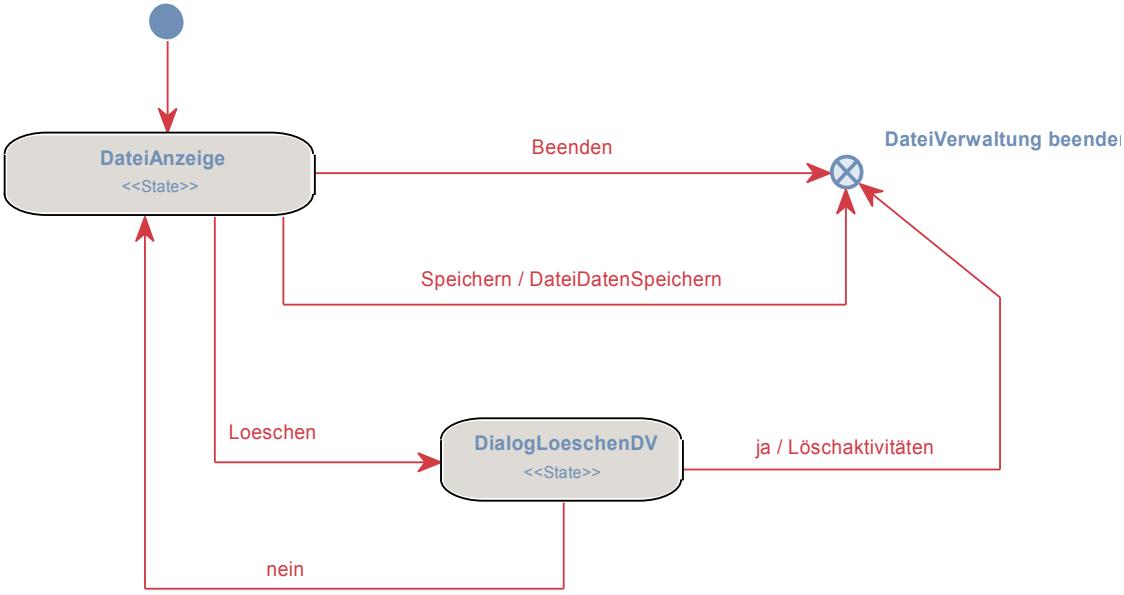


Abbildung E.6: Fallbeispiel – Verfeinerung von Zustand Dateiverwaltung

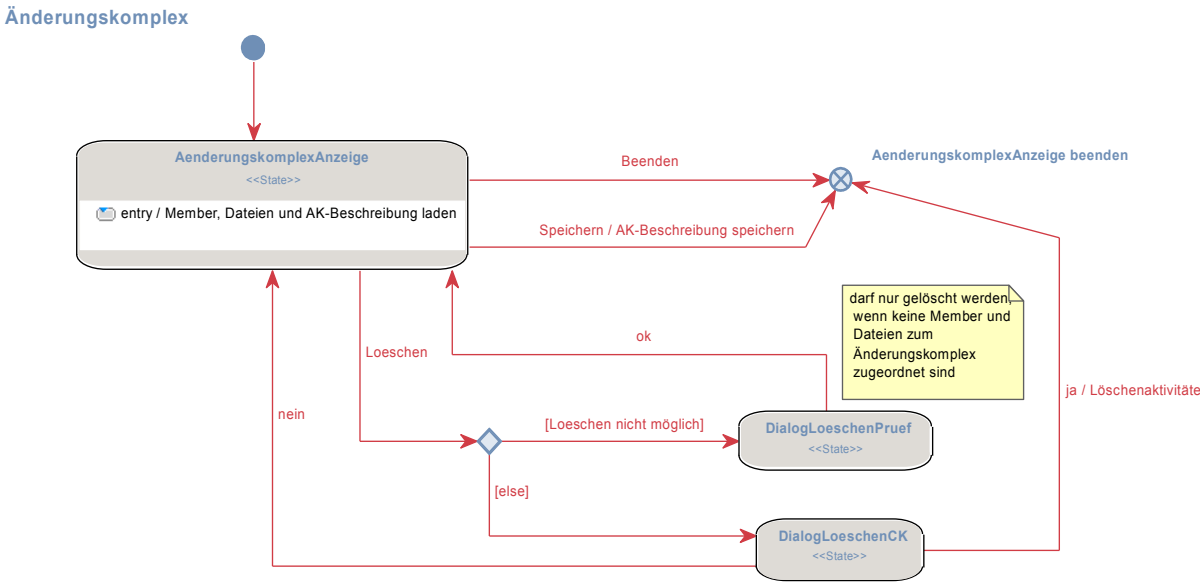


Abbildung E.7: Fallbeispiel – Verfeinerung von Zustand Änderungskomplex

F Tests für den Prototypen

F.1 Ausgewählte Testfälle

/TF1/ Ein Benutzer meldet sich mit Benutzername und Passwort über die Benutzeranmeldung an und bekommt die Verwaltungskomplexanzeige mit allen verfügbaren Membern und Änderungskomplexen angezeigt.

/TF2/ Ein Benutzer wählt in der Verwaltungskomplexanzeige einen verfügbaren Member aus und drückt den vorgesehenen HTML-Knopf, um alle Daten des ausgewählten Members in der Member-Verwaltung angezeigt zu bekommen.

/TF3/ Ein Benutzer stellt den Bearbeitungsstatus eines Members auf 6 (Vorübergabe) und speichert den Member durch Drücken des HTML-Knopfes Speichern ab. Das Datum für Vorübergabe wird durch das System automatisch gesetzt.

/TF4/ Ein berechtigter Benutzer speichert alle Informationen über einen Member, in dem der HTML-Knopf Speichern gedrückt wird.

/TF5/ Ein Benutzer, der als Bearbeiter für einen Member berechtigt ist, bearbeitet die Informationen über einen Member.

/TF6/ In der Verwaltungskomplexanzeige gibt ein Benutzer im Eingabefeld Neu einen neuen Member-Name ein und drückt den HTML-Knopf, um alle geforderten Informationen über den neuen Member in der Member-Verwaltung einzugeben.

/TF7/ In der Member-Verwaltung löscht ein berechtigter Benutzer einen Member durch Drücken des HTML-Knopfes Löschen. Der Löschvorgang muss in einem anschließenden Dialog bestätigt und protokolliert werden.

/TF8/ Ein Benutzer wählt in der Verwaltungskomplexanzeige einen verfügbaren Änderungskomplex aus und drückt den vorgesehenen HTML-Knopf, um alle zugehörigen Member bzw. Dateien oder auch die Beschreibung des ausgewählten Änderungskomplexes in der Änderungskomplexanzeige angezeigt zu bekommen.

/TF9/ In der Änderungskomplexanzeige speichert ein Benutzer die Beschreibung eines Änderungskomplexes durch Drücken des HTML-Knopfes Speichern.

/TF10/ Ein Benutzer bearbeitet die Beschreibung eines Änderungskomplexes in der Änderungskomplexanzeige oder fügt sie hinzu.

F.2 Ausgewählte Testszenarien

/TS8/ Einen Member bearbeiten und abspeichern

1. /TF1/
2. /TF2/
3. /TF5/
4. /TF4/

/TS9/ Einen neuen Member anlegen, bearbeiten und abspeichern

1. /TF1/
2. /TF6/
3. /TF4/

/TS10/ Die Beschreibung eines Änderungskomplexes hinzufügen bzw. ändern und abspeichern

1. /TF1/
2. /TF8/
3. /TF10/
4. /TF9/

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe.

_____ _____

Ort, Datum Unterschrift