Reverse Engineering von Datenbankanwendungen

Birgit Demuth

Technische Universität Dresden
Fakultät Informatik
demuth@inf.tu-dresden.de

Zusammenfassung

Das Reengineering von relationalen Altanwendungen basiert in der kommerziellen Praxis häufig auf der Architekturentscheidung, auch weiterhin relationale Datenbanken für die Datenverwaltung einzusetzen. Der erste Schritt für das Reengineering eines Altsystems ist dabei ein Reverse Engineering sowohl der Datenbank als auch der darauf bestehenden Anwendungen, mindestens im Sinne der Redokumentation der Architektur des bestehenden Gesamtsystems. Wir berichten über Reverse Engineering-Erfahrungen aus zwei Fallstudien und geben einen Ausblick auf zukünftige Arbeiten.

1 Motivation

Relationale Datenbankanwendungen spielen in der kommerziellen Praxis der Informationstechnologie eine immer größere Rolle. Objektorientierte Technologien gewinnen zwar im Bereich moderner Softwareentwicklung an entscheidender Bedeutung, konnten aber bislang im Bereich der Datenverwaltung die relationalen Datenbanken nicht verdrängen. Aufgrund der bekannten Vorteile relationaler Datenbankmanagementsysteme, wie z.B. Zuverlässigkeit, Funktionsmächtigkeit, Performance und Investitionssicherheit, ist gegenwärtig der Trend zur Integration von objektorientierter Anwendungsentwicklung und des Einsatzes relationaler Datenbanken zu beobachten. Für die in den letzten ein bis zwei Jahrzehnten entstandenen relationalen Anwendungssysteme ergibt sich häufig die Notwendigkeit, die Systeme zu renovieren, um künftigen Anforderungen gewachsen zu sein. Im allgemeinen werden folgende Szenarien in Erwägung gezogen:

- *Restrukturierung* des Altsystems auf gleichem Abstraktionsniveau mit ggf. vorsichtigem Übergang zu objektorientierten Technologien, z.B. durch Wrapping von C-Programmen in C++-Klassen und Redokumentation des Systems auf der Basis objektorientierter Notationen;
- *Reengineering* des Altsystems mit dem Ziel, echte objektorientierte Softwarearchitekturen zu realisieren.

Beide Wege basieren auf der Architekturentscheidung, auch weiterhin relationale Datenbanken für die Datenverwaltung einzusetzen. Das schließt das Reengineering der relationalen Altdatenbanken nicht aus. Im Gegenteil, oft ist es notwendig, die Altdatenbanken ebenfalls zu restrukturieren. Der erste Schritt sowohl für die Restrukturierung als auch für das Reengineering eines Altsystems ist ein *Reverse Engineering* (RE), mindestens im Sinne der Redokumentation der Architektur des bestehenden Gesamtsystems. Das beinhaltet in Abhängigkeit von den vorhandenen Software-Artifakten und dem angestrebten Ziel des Reverse Engineerings u.a.

- Redokumentation des relationalen Datenbankschemas (Tabellen, Sichten, Constraints, Trigger, Stored Procedures) sowie das Design Recovery eines Entity-Relationship-Modells
- Redokumentation der Programmstruktur sowie aller physischen Komponenten
- Redokumentation der Beziehungen zwischen Datenbank und Anwendung
- Erstellung eines Anwendungsfallmodells
- Redokumentation von Aufrufhierarchien
- Redokumentation von Datenflußdiagrammen

Der Lehrstuhl Softwaretechnologie an der Fakultät Informatik der TU Dresden führt im Rahmen von Diplomarbeiten und Großen Belegen (Studienarbeiten) Fallstudien zum Reverse Engineering relationaler Datenbankanwendungen durch. In Kooperation mit Softwareunternehmen werden Altsysteme mit verschiedenen CASE- bzw. CARE[1]-Tools untersucht und prototypisch ein Vorgehen zum Reverse Engineering des konkreten Altsystems entwickelt. Wir berichten in den Abschnitten 2 und 3 über Reverse Engineering-Erfahrungen aus zwei Fallstudien und geben anschließend einen Ausblick auf zukünftige Arbeiten.

2 Fallstudie EMBEDDED SQL&COBOL

Die erste Fallstudie hat eine reale Cobol-Anwendung mit eingebettetem SQL zum Gegenstand, welche für den Stapelbetrieb entwickelt wurde. Die Anwendung hat eine eher geringe Komplexität (100 Dateien, 17 Cobol-Programme, ca. 23000 LOC ohne Kommentare, Leerzeilen und Copy-Befehle, 28 Datenbanktabellen). Ziel der Fallstudie war es, Erfahrungen mit unterschiedlichen Softwarewerkzeugen bei der Redokumentation von Cobol-Programmen und der Erstellung eines objektorientierten Anwendungsmodells zu sammeln.

Vorgehen. Das EMBEDDED SQL&COBOL-System wurde im Rahmen eines Großen Beleges untersucht. Dabei wurde zunächst das COREM-Vorgehensmodell aus [10] in den frühen Phasen (Design Recovery) auf Cobol angepaßt und auf das konkrete Cobol-System in folgenden Schritten angewendet:

- *Systemüberblick.* In der Vorbereitungsphase für das Reverse Engineering werden der physische Systemaufbau, d.h. die existierenden Dateien (Cobol-Programme, Copy-Strecken, Batch-Programme, Dialogprogramme, Datenbankbeschreibungen, Datenbanktabellen, Daten-Dateien) und die zwischen ihnen bestehenden Beziehungen (z.B. Aufruf- und Copy-Beziehungen) erfaßt.
- *Programmstrukturanalyse.* Der erste Schritt im Design Recovery von COREM bezieht sich auf das Generieren von *Structure Charts*. Dazu müssen zunächst die Prozeduren identifiziert werden. Da es in Cobol kein echtes Prozedurkonzept gibt, werden die PERFORM-Anweisungen für eine Simulation von Prozeduren herangezogen und deshalb analysiert. PERFORM-Anweisungen führen einen Programmblock aus und springen dann in der Programmabarbeitung zurück. Eine Alternative für die Darstellung der Strukturierung eines Cobol-Systems ist die Aufteilung auf der Ebene der Programme, wie dies z.B. durch das SOFT-REORG Reengineering System [13] unterstützt wird. Im Rahmen

[1] CARE: Computer Aided Reverse Engineering

der Fallstudie wurde jedoch eine geringere Granularität angestrebt und deshalb auf Basis der PERFORM-Beziehungen ein Algorithmus zur Identifikation von Cobol-Prozeduren und deren Aufrufbeziehungen entwickelt. Die in [10] zur Darstellung der Sichtbarkeit von Variablen verwendeten *Nesting Trees* haben aufgrund der globalen Cobol-Variablen keine Bedeutung und werden deshalb weggelassen.

- *Datenstrukturanalyse.* Für die nachfolgende Datenflußanalyse werden die verwendeten Datenstrukturen ermittelt. Als Darstellungsmittel eignen sich aufgrund der hierarchischen Strukturierung der Cobol-Datenstrukturen die Datenstrukturdiagramme der Jackson-Methode (*Jackson-Diagramme* [1]. Dabei finden auch persistente Datenfelder Berücksichtigung (Felder aus Datenbanktabellen und Dateien).

- *Datenflußanalyse.* Eine detaillierte Datenflußanalyse und deren Darstellung in einem *Datenflußdiagramm* führt sehr schnell zu einer hohen Komplexität. Für die Fallstudie wurde ein generischer Algorithmus für GOTO-freie Cobol-Programme einschließlich der Berücksichtigung von Datenbankoperationen entworfen. Die Ergebnisse dieses Algorithmus werden sowohl in einer *Variablen-Prozedur-Matrix* als auch in einem *Datenflußdiagramm* dargestellt.

- *Generierung von Entity-Relationship-Diagrammen (ERD).* Die Generierung von ERD berücksichtigt entsprechend [10] *Data-Store-Entities (DSE)*, die persistente Daten repräsentieren und unmittelbar im ERD dargestellt werden. Für eine fundierte Analyse der DSE ist darüberhinaus die Analyse der *Non-Data-Store-Entities (NDSE)* notwendig, die in einem bestimmten Zusammenhang mit den DSE stehen.

Auf Basis eines zuvor erarbeiteten Anforderungskataloges wurden verfügbare Cobol-CARE-Tools evaluiert und als Unterstützung für das oben beschriebene Vorgehen eingesetzt.

Anforderungen an COBOL-Reverse-Engineering-Werkzeuge. Der Anforderungskatalog entstand sowohl aufgrund eigener Erfahrungen als auch im Ergebnis einer Literaturanalyse. Insbesondere sei dabei auf [2] verwiesen. Die umfangreichen Anforderungen werden in folgende Kategorien gruppiert:

Generelle Anforderungen : Werkzeugplattform, Mehrbenutzermodus, Speicherformate, Erweiterbarkeit, Umgebungsanpassungen

Unterstützung der Vorbereitungsphase für das Reverse Engineering : analysierbare Quellen, Importmethode, Fehlerbehandlung, unterstützte Verknüpfungen zwischen verschiedenen Quellenarten, Analyse fehlender, doppelter und ungenutzter Dateien, Systemüberblick

Unterstützung grundlegender Analysen :

- Untersuchung von Programmeigenschaften wie Kontrollfluß, Datenfluß, Abhängigkeiten zwischen Variablen, Datenstrukturen, Datenbankstrukturen, Datenbankzugriffe
- Darstellung dieser Eigenschaften in verschiedenen Diagrammen
- Möglichkeiten der manuellen Verfeinerung bzw. Korrektur dieser Darstellungen

Unterstützung der objektorientierten Modellbildung :

- Analyse von Entitäten, Beziehungen, dynamischen Aspekten von Entitäten sowie deren graphische Darstellung
- Möglichkeiten der Interaktion des Benutzers mit dem CARE-Tool

Erfahrungen. Ein Neueinsteiger in die Cobol-Problematik[2] sieht sich nicht nur mit Altsoftware, sondern auch mit einer unüberschaubaren Vielzahl von praktischer als auch theoretischer Altliteratur, Werkzeugen sowie einer gewissen Renaissance von Cobol-Werkzeugen im Kontext des Jahr-2000-Problems konfrontiert. In der zur Verfügung stehenden Zeit von einem halben Jahr konnte deshalb nur ein geringer Ausschnitt an vorhandenem Cobol-Wissen und vor allem Tools evaluiert werden. Zusammenfassend können folgende Aussagen formuliert werden:

- Das Reverse Engineering von Cobol-Altanwendungen ist in einem aufwendigen und schrittweisen Prozeß möglich.
- Es wurde kein Tool gefunden, welches eine durchgängige Unterstützung des beschriebenen Vorgehensmodells bietet. RE-Funktionen werden oft nur als Zusatz in CASE-Tools angesehen. Der Übergang zum objektorientierten Design Recovery wird von keinem Tool konsequent unterstützt. Bei dieser Einschätzung ist zu beachten, daß die verfügbare Auswahl evaluierter Tools aufgrund organisatorischer Randbedingungen klein war.
- Die evaluierten Tools gestatten nur einfache Analysen von Dokumenten in RE-Teilprozessen (z. B. Extraktion von Datenstrukturen).
- Die vielen Cobol-Dialekte erschweren das Reverse Engineering.
- Aus Sicht der Praxistauglichkeit hat *Revolve* (Microfocus) den besten Eindruck hinterlassen. Die relativ einfache Bedienbarkeit und der große Funktionsumfang bilden eine gute Basis für die Untersuchungen zu Beginn des RE-Prozesses. Das Repository von Revolve enthält umfangreiche Programmdetails, die durch einen integrierten COBOL-Parser geliefert werden. Erweiterungen wie Datenbankanbindungen werden erkannt und als Objekte gespeichert. Der Datenfluß ist nicht als explizite Darstellungsart vorhanden, kann aber über eine *Impact-Analyse* verfolgt werden. Das Problem mit dem Einsatz von Revolve nach dem beschriebenen Vorgehensmodell ist, daß Revolve lediglich vorgefertigte Analysen des Repositorys, wie z.B. die Erkennung ungenutzter Variablen, bereitstellt. Eine eigene Abfrage des Repositorys war nicht möglich. Der Übergang zur objektorientierten Modellierung wird nicht unterstützt.
- Aus der Sicht des Reverse Engineering von Datenbankanwendungen verdient das Forschungsprojekt und Werkzeug für den Datenbankentwurf *DBMAIN* [7] besondere Beachtung. DBMAIN ist ein mächtiges CASE-Tool, welches auch RE-Funktionen für Cobol-Programme mit eingebettetem SQL einschließt. Das Werkzeug erfordert einen erheblichen Einarbeitungs- und Anpassungsaufwand. Die RE-Funktionen für COBOL können noch nicht als ausgereift bezeichnet werden. Vor allem sollten mehr COBOL-Dialekte verarbeitet oder wenigstens unbekannte Programmkonstrukte fehlerfrei ignoriert werden. Das offene Repository und die flexible Skriptsprache von DBMAIN ermöglichen es auf der anderen Seite, eigene Import-Funktionen zu implementieren, so Unzulänglichkeiten auszugleichen sowie es als Plattform für ein experimentelles Cobol-Reengineering einzusetzen.

[2] noch dazu, wenn es ein Freund objektorientierter Denkweisen und der Java-Programmierung ist

3 Fallstudie PL/SQL&C++.

In einer zweiten Fallstudie wird ein großes Softwaresystem (Quellcode ca. 50 MB) einschließlich einer Oracle-Datenbank (ca. 400 Tabellen) untersucht. Das System wurde Anfang der 90er Jahre nach Prinzipien der strukturierten Analyse in C++ entwickelt. Es befindet sich in ständiger Weiterentwicklung und Anpassung an verschiedene Kunden. Im Rahmen der Fallstudie wird gegenwärtig daran gearbeitet, den Bedingungen des Softwareunternehmens angepaßte technische Empfehlungen für die Redokumentation und das Reverse Engineering des Softwaresystems zu erarbeiten.

Softwaretechnologische Probleme. Aus der angewendeten Entwicklungsmethode und dem firmenspezifischen Entwicklungsprozeß resultieren eine Reihe schwerwiegender softwaretechnologischer Probleme, die zum Teil typische Beispiele für *Anti-Patterns* [6] repräsentieren:

- Es existieren zwar viele Entwicklungsdokumente, diese aber nicht zu allen wichtigen Analyse- und Entwurfsaspekten. Die Dokumentationsqualität ist sehr unterschiedlich. Desweiteren gibt es zwischen den vorhandenen Software-Artifakten so gut wie keine dokumentierten Beziehungen.
- Eine Sicht auf die Architektur des Gesamtsystems fehlt. Die physische Sicht auf das System (Verwaltung mit CVS, Dateien in 19 Verzeichnissen organisiert) reflektiert keine Architektur- und Entwurfslogik.
- Die Konsistenz zwischen Quellcode bzw. Datenbank und existierenden Dokumentationen ist im Laufe der Jahre verlorengegangen. Ein Roundtrip-Engineering ist wünschenswert.
- Der C++-Code ist kaum kommentiert. Vorhandene Kommentare beziehen sich im wesentlichen auf die Fehlerbeseitigung.
- Das Reverse Engineering des C++-Quellcodes mit einem OOA/OOD-Tool[3] bringt typische *Blobs* [6] zum Vorschein. Blobs sind Klassen mit einer Vielzahl von Attributen und/oder Operationen, die häufig über nur eine geringe Kohäsion verfügen. Sie sind damit oft ein Symptom für fehlenden objektorientierten Entwurf. Ihre Komplexität erschwert die Wiederverwendung und das Testen.
- Erste Analysen der physischen Struktur des Systems und des vorhandenen Codes sowie der Versionierungsprozeß lassen viele tote Codefragmente, Variablen und Datenbankobjekte vermuten (*Lava Flow*-Anti-Pattern [6]).
- Sowohl der ursprüngliche Entwurf des Systems nach Prinzipien der strukturierten Analyse als auch die Inspektion des Codes lassen das Fehlen jeglicher objektorientierter Architektur und damit die Hoffnungslosigkeit auf Wiederverwendung von Klassen dieses so komplexen Systems erkennen.
- Die Organisation in der Softwareentwicklung nach dem *Mushroom-Management-Prinzip* [6] unterbindet den unmittelbaren Kontakt der eigentlichen Softwareentwickler (Programmierer) zu den Endnutzern (Kunden) des Systems. Die Programmierer erhalten damit Softwareanforderungen praktisch aus zweiter und manchmal aus dritter Hand. Das wirft Verständnis- und Realisierbarkeitsprobleme auf.
- Die Wartung und Weiterentwicklung des Systems ist derzeit nur eingeschränkt mit einem immensen Personalaufwand möglich.

[3] OOA: objektorientierte Analyse, OOD: objektorientiertes Design

Vorgehen. Ziel der Fallstudie (Gegenstand einer Diplomarbeit) ist das Reverse Engineering der Architektur des beschriebenen Systems und dessen Darstellung in UML-Notation [5], [12]. Zunächst war nur bekannt, daß das System in funktionale Module strukturiert ist, die jeweils grundlegende Geschäftsprozesse implementieren. Um Redokumentation und Reverse Engineering des komplexen Systems prototypisch zu realisieren, wurde ein einzelner Modul für die Untersuchungen ausgewählt. Da dieser jedoch mit dem (ebenfalls sehr komplexen) Kern des Systems (Basisklassen) in vielen Beziehungen steht, konnte die Komplexität der zu untersuchenden Software nur wenig eingeschränkt werden. Nach ersten Erfahrungen bei der Analyse der vorhandenen Software-Artifakten und des Entwicklungsprozesses, des Einsatzes von Werkzeugen und in Diskussionen mit Entwicklern wurde ein detaillierter Fragebogen ausgearbeitet, der zum einen die Situation in der Softwareentwicklung und zum anderen die spezifischen Dokumentationsprobleme systematisch erfassen soll. Die weiteren Schritte werden sein:

- Auswertung der Fragebögen und detaillierte Erfassung der spezifischen Anforderungen an die Dokumentation und das Roundtrip-Engineering des Systems
- Evaluation von bereits ausgewählten CASE/CARE-Tools unter den spezifischen Anforderungen (u.a. Rational Rose, Oracle Designer 2000, System Architect, Together/C++)
- Suche nach weiteren Tools, die die spezifischen Anforderungen unterstützen
- Auswahl geeigneter Tools für die hausspezifische Dokumentation und das Roundtrip-Engineering
- Erarbeitung von Empfehlungen und Präsentation der Ergebnisse anhand des Beispielmoduls

Die Ergebnisse der Fallstudie sollen die Wartung des existierenden Systems verbessern helfen. Ein Reengineering ist nicht vorgesehen. Parallel zum existierenden System wird ein neues Produkt nach objektorientierten Prinzipien entwickelt, welches langfristig das bestehende ablösen soll.

Bisherige Erfahrungen. Abgesehen vom Aufdecken der oben aufgelisteten grundlegenden softwaretechnologischen Probleme haben wir eine Reihe weiterer spezifischer Beobachtungen gemacht:

- Das bereits genannte Mushroom-Management-Prinzip in der Softwarefirma scheint ein teamorientiertes Arbeiten zu behindern, was sich wiederum negativ auf die Dokumentation der Software auswirkt.
- Es werden Dokumentationen des Systems benötigt, die die verschiedenen Sichten der Mitarbeiterprofile (Analytiker, Programmierer, Testingenieure, Dokumentationsautoren, Trainer, Administratoren) auf das System reflektieren.
- Die Erfassung der spezifischen Anforderungen gestaltet sich schwierig. Die Entwickler stehen unter großem Projektdruck und sind deshalb wenig für Interviews zu ihren spezifischen Problemen aufgeschlossen.
- Bei den Softwareentwicklern gibt es einen hohen softwaretechnologischen Qualifizierungsbedarf.

- Es gibt keine ausgesprochenen Entwurfsdokumente, lediglich Analysedokumente, die als Arbeitsgrundlage zwischen der Firma und den Kunden dienen. Diese sind mit wenigen technischen Details (vor allem DB-Aspekte, die den Kunden nicht interessieren) angereichert. Daraus ergeben sich gravierende Probleme für die Programmierer, die oft nicht wissen, wie sie die Anforderungen umzusetzen haben.

- Prozeßflüsse zur Darstellung der Geschäftsabläufe spielen im Entwicklungsprozeß eine entscheidende Rolle. Die Dokumentationsqualität dieser ist sehr unterschiedlich. Das sollte ein Ansatzpunkt für die weiteren Untersuchungen sein.

- Eine große Hilfe für die Programmierer wäre eine ausgefeilte Toolunterstützung zur Analyse und zum Reverse Engineering von *Stored Procedures* [11] (PL/SQL-Prozeduren), in denen der Hauptteil der Geschäftslogik implementiert ist.

- Die Dokumentation der Datenbankstrukturen ist vergleichsweise gut. Die Datenbanktabellen wurden ursprünglich mit CASE-Toolunterstützung aus Entity-Relationship-Diagrammen generiert. Allerdings besteht inzwischen das Konsistenzproblem. Nachträgliche Änderungen im Datenbankschema wurden in den ERDs nicht nachgezogen. Typische Probleme des Reverse Engineerings relationaler Datenbanken, wie sie auch in unserer Fallstudie auftreten sin in [4], [3] beschrieben.

4 Ausblick

In unseren Fallstudien verfolgen wir das Ziel, die Mächtigkeit derzeitiger CASE- und CARE-Tools im Hinblick auf das Reverse Engineering sowohl von Datenbanken als auch von darauf basierenden Anwendungen kennenzulernen. Wir suchen nach Ansätzen einer besseren Integration von Daten- und Programm-Reverse-Engineering. Auf Basis einer auf diese Weise erzielten aktuellen Redokumentation bzw. eines Design Recoverys von Datenbankanwendungen wäre in einem weiteren Schritt das Roundtrip Engineering aus der Sicht der Softwareentwickler äußerst hilfreich. Wir planen, unter Nutzung unserer eigenen XML- und UML-basierten Werkzeugplattform *Dresden UML Toolset* [8] experimentelle Implementierungen von RE-Funktionen durchzuführen. Damit möchten wir einen Beitrag zur Weiterentwicklung der in der Praxis der Softwareentwicklung so dringend benötigten CARE-Umgebungen leisten. Wichtige allgemeine Anforderungen, die von einer CARE-Umgebung erwartet werden [9], wie z.B.

- Erweiterbarkeit der Funktionalität
- Auswertung verschiedener Quellen
- Zusammenarbeit mit anderen CASE-Prozessen
- offene Architektur
- Unterstützung verschiedener Schematransformationstechniken

werden wir durch die Verwendung von XMI als Austauschformat von Modellen bzw. Metamodellen zwischen verschiedenen Tools sowie durch eine Plug-in-Architektur von Dresden UML Toolset sichern. Um in kurzer Anpassungszeit die verschiedenen Document Type Definitions (DTDs) für Metamodelle sowie deren Versionen (z.B. UML DTD 1.1, UML DTD 1.3, UML DTDs der verschiedenen CASE-Tool-Hersteller) und deren Instanziierungen durch XML-Dateien einlesen und verarbeiten

zu können, wurde im Rahmen eines Großen Beleges ein Tool zur Generierung von beliebigen Metadatenmodellen in Java-Repräsentation (DTD2J [8]) realisiert. Momentan wird dieses Tool verwendet, um interne Darstellungen von UML-Modellen zu generieren. Genauso kann es aber für generische Metadatenmodelle von prozeduralen Sprachen genutzt werden. Das wiederum erlaubt uns, Analyse- und Transformationsalgorithmen basierend auf Standardaustauschformaten zu implementieren. Insgesamt jedoch bleibt die Tatsache bestehen, daß die Toolunterstützung für RE-Prozesse ungleich schwieriger zu realisieren ist als die für das Forward Engineering.

Danksagung. Wir danken unseren Studentinnen und Studenten Anne Thomas, Axel Großmann, Sven Obermaier und Frank Finger für ihr Engagement und ihre konstruktiven Diskussionen bei der Bearbeitung der beschriebenen Fallstudien bzw. der Entwicklung von Dresden UML Toolset.

Literatur

[1] Balzert, H.: Lehrbuch der Softwaretechnik. Software-Entwicklung. Spektrum Akademischer Verlag, 1996

[2] Bellay, B., Gall, H.: A Comparison of Four Reverse Engineering Tools. in: Baxter, I. and others: Proceedings of the Fourth Working Conference of Reverse Engineering (WCRE97). IEEE Computer Society Press, 1997

[3] Blaha, M., Premerlani, W.: Observed Idiosynchracies of Relational Database Design. Second Working Conference on Reverse Engineering (WCRE95), Toronto, Ontario, 1995

[4] Blaha, M., Premerlani, W.: Object-Oriented Modeling and Design for Database Applications. Prentice Hall, 1998

[5] Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison-Wesley, 1999

[6] Brown, W. and others: Anti Patterns. Refactoring Software Architectures, and Projects in Crisis. John Wiley, 1998

[7] DBMAIN: A R&D Programme in Database Applications Engineering and CASE Technology, Deptartment of Computer Science, University of Namur

`http://www.info.fundp.ac.be/~dbm/`

[8] Dresden UML Toolset, Technische Universität Dresden, Fakultät Informatik

`http://www-st.inf.tu-dresden.de/UMLToolkit/`

[9] Hainaut, J.-L. and others: Database Reverse Engineering: From Requirements to CARE Tools. in: Wills, L., Newcomb, Ph.: Reverse Engineering. Kluwer Academic Publishing, 1996

[10] Klösch, R., Gall, H.: Objektorientiertes Reverse Engineering. Von klassischer zu objektorientierter Software. Springer, 1995

[11] Melton, J.: SQL's Stored Procedures. A Complete Guide to SQL/PSM. Morgan Kaufmann, 1998

[12] OMG UML Specification v. 1.3 draft

[13] Sneed, H.: Softwaresanierung (Reverse und Reengineering). Verlag Rudolf Müller (DV-Praxis-Online), 1992