Rückwärtsmodellierung bestehender Finanzdienstleistungssysteme

Reverse Engineering von Legacy Software

Harry Sneed[1], Sonja Schuster[2]

Abstract: Many IT user organizations are stuck with software systems from the past. These old systems often fulfill a mission critical purpose for financial institutions such as issuing credit, leasing goods or administering accounts. Should such a system fail to run for any reason this amounts to a super GAU for the institution involved. It could result in bankruptcy. Therefore, the continuity of service has the utmost priority. On the other hand these systems are implemented in ancient technologies which hardly anyone wants to deal with. Most young developers will run away when confronted with the unpleasant task of having to maintain such a system. That is the main motivation for outsourcing the maintenance operations. But outsourcing can only be a temporary solution. Eventually the software has to be renovated or replaced. To replace them is not so easy because the new system will seldom provide the same functionality. If users insist on keeping the current functions, the user company has no other choice but to renovate the existing system. This is no easy task. Either the code is converted automatically by tool and no one is really satisfied with the results, or the code is re-implemented by hand based on a model extracted from the code. Re-implementation requires that the system be documented down to a very detailed level. Those who rewrite the code need detailed descriptions of how the old code was implemented in order to be able to replicate it. This paper describes a project aimed at doing just that – extracting a documentation consisting of many models of the system – data models, architecture models, data flow models, control logic models and business rule models. The models extracted are intended to be used not only as a basis for re-implementation but also as a reference for system maintenance. This inverse modelling process is fully automated and can be realized with a minimum of effort.

Zusammenfassung: Viele IT Anwender sind mit alten Systemen ausgestattet. Diese veralteten „Legacy Systeme“ erfüllen für das Unternehmen einen lebenswichtigen Zweck. In der Finanzwirtschaft leisten sie unternehmenskritische Dienste wie Kreditvergabe, Investitionsverwaltung und Leasing von Wertgegenständen. Der Ausfall eines dieser Systeme würde das Ende des Finanzdienstleisters bedeuten. Die Kontinuität der Dienstleistung ist deshalb lebenswichtig für das Unternehmen. Dennoch sind viele dieser unentbehrlichen IT-Systeme in die Jahre gekommen. Sie sind mit alten Programmiersprachen implementiert und basieren auf überholten Datenbanksystemen. Sie zu erneuern ohne den Tagesbetrieb zu unterbrechen, ist eine große Herausforderung für die Software Reengineering Technologien. Sämtliche Teilbereiche der Technologie sind davon betroffen – Datenbankmigration, Code-Konvertierung, Oberflächenerneuerung und Prozessumgestaltung. Der Erneuerungsprozess darf nicht ad hoc passieren, er soll geplant und am besten modellbasiert sein. Zunächst muss die alte Lösung modelliert werden, um diese anschließend in ein Modell der neuen Lösung zu transferieren. Wenn nur der Source Code vorliegt, wie das üblicherweise der Fall ist, bleibt nur der Weg übrig, das

[1] SoRing Kft, Kapitanutca 6, H1123 Budapest, Ungarrn, Harry.Sneed@SoRing.hu
[2] ReqPool GmbH, Garnisonstrasse 19B, A4020 Linz, Austria,Sonja.Schuster@reqpool.com

Modell aus dem bestehenden Code wiederzugewinnen. Die Modellierung findet bottom-up statt, vom Code zum Modell. In diesem Beitrag wird anhand zweier Beispiele aus der Finanzwirtschaft gezeigt, wie werkzeuggestützte Modellierungstechniken diesen Modernisierungsprozess unterstützen können.

Keywords: Modellierung, Datenmodelle, Prozessmodelle, Reverse Engineering, Codeanalyse, Altsystemmodellierung, Modelltransformation, Systemarchitektur, Aufrufhierarchie, Systemablauflogik, Geschäftsregel, inkrementelle Komponentenablösung.

1 Einleitung

Viele Anwenderbetriebe stehen vor der Ablösung ihrer Kernapplikationen. Diese sogenannten Legacy Systeme, die mit der Technologie des letzten Jahrhunderts realisiert wurden, stehen der Einführung neuer Technologien im Wege. Sie müssen entweder migriert, neuentwickelt oder durch andere Systeme ersetzt werden. Die Zeit drängt.

2 Software Sanierungsprojekte

Die Aufgabe ein bestehendes IT-System zu modernisieren ist eine große Herausforderung für die Reengineering Forschung, vor allem wenn das System ununterbrochen weiter funktionieren soll. Das erste Projekt dieser Art fand in Deutschland bei der Bertelsmann AG im Jahre 1983 statt [Sn84]. Damals ging es um die Umsetzung eines unfertigen PL/I Systems mit einer DLI Datenbank in ein COBOL System mit einer ADABAS Datenbank. In dem Projekt war das System noch nicht in Betrieb. Das Entwicklungsprojekt wurde abgebrochen und die PL/I – DLI Lösung kam deshalb nie zum Einsatz. Zurück blieben nur große Teile ungetesteten Codes. Die ungarische Sanierungsmannschaft dokumentierte den Code nach, konvertierte und testete diesen neu.

Das letzte Projekt dieser Art war ein VisualAge/PL-I Abrechnungssystem, das durch ein Java System mittels inkrementeller Vorgehensweise abgelöst wurde. Der Code wurde nachdokumentiert, um daraus ein prozedurales Ist-Modell mit 196 Anwendungsfällen abzuleiten. Jeder Anwendungsfall bildete eine eigene Komponente, die danach als Java Paket implementiert wurde. Die neuen Java Pakete konnten neben den bestehenden VisualAge Komponenten eingesetzt werden [SV15]. Auf diese Weise wurde die Software Komponente für Komponente abgelöst. Der Vorteil dieser Vorgehensweise ist, dass die Datenbank unverändert bleibt. Es ist eine besondere Herausforderung, sowohl Daten als auch Programme und dessen Oberflächen in einem Projekt zeitgleich abzulösen. Je weniger verändert werden muss, desto größer sind die Erfolgschancen des Projektes [Go16].

3 Vorgehensweise eines Software-Sanierungsprojektes

Bei den letzten Sanierungsprojekten haben sich folgende Vorgehensweisen mit sieben Schritten bis zum neuen Code bewährt:

- Systembestandsaufnahme
- Systemverpackung
- Systemvermessung
- Systemnachdokumentation
- Systemmodellierung
- Systempartitionierung
- System Re-implementierung [SpL03].

3.1 Systembestandsaufnahme

Im ersten Schritt wird eine Bestandsaufnahme des bestehenden Systems vorgenommen. Alles, was zu dem System gehört – Source Code Dateien, Datenbankschemen, Jobsteuerungsprozeduren, Testfälle, Testprotokolle, Fehlerberichte, Benutzerdokumente und technische Dokumente – werden gesammelt und gezählt. Inwieweit diese Systemelemente verwertbar sind, wird zu einem späteren Zeitpunkt festgestellt. Es kommt in diesem Schritt darauf an, alle Informationen zentral an einer Stelle zu sammeln und festzuhalten. Dies wird auch als Systeminventur bezeichnet. Die Namen und Versionen der Elemente werden in Excel Tabellen erfasst, dabei wird eine Tabelle für jeden Elementtyp angelegt [Te10].

3.2 Systemverpackung

Im zweiten Schritt werden die Systemelemente bzw. Source-Dateien auf mehrere Bibliotheken aufgeteilt. Es gibt verschiedene Möglichkeiten der Aufteilung in Abhängigkeit der Art des Systems, z.B. nach Subsystemen, Transaktionen bzw. Anwendungsfällen, Jobprozessen oder nach Datenbankcluster. Online-Transaktionssysteme werden in der Regel nach zusammengehörigen Datentransaktionen aufgeteilt, d.h. man sammelt alle Source-Dateien, die zu einer Transaktion gehören, einschließlich Bildschirmmasken und Datenbanktabellen bzw. -ansichten. Batchsysteme werden hingegen nach Batchprozessen aufgeteilt. Zu einem Batchprozess gehören Jobprozeduren, Source-Programme und Listenbeschreibungen. Die Source-Elemente werden je nach Source-Typ den Unterbibliotheken zugeordnet. Es ist wichtig, dass alle Sourcen desselben Typs zusammen sind, da diese Sammlung eine Voraussetzung für die Vermessung ist [Te10].

3.3 Systemvermessung

Im dritten Schritt werden die messbaren Systemelemente vermessen. Die meisten Elementtypen lassen sich automatisch vermessen, jedoch erfordern einige manuelle Eingriffe, andere können überhaupt nicht automatisiert werden. Alle Elemente mit einer annehmbaren formalen Struktur können geparst bzw. gescanned werden. Alle Source-Typen, Datenbankschemen, Jobsteuerungsprozeduren, Testfälle und Entwurfsdokumente lassen sich mittels Werkzeuge automatisch analysieren und deren Inhalte zählen, z.B. Function Point Zählung des Source Codes. Das Tool SoftAudit verarbeitet etliche Programmier-, Datenbank- und Jobsteuerungssprachen, sowie Testfalltabellen und natursprachliche Dokumente, sofern diese markiert sind. Zu jedem Elementtyp wird ein Metrik-Bericht erstellt, in dem die Elemente gezählt werden. Die Anzahl der Elemente und deren Eigenschaften macht die Quantität bzw. die Größe des Systems aus. Anhand der Größenmaße kann mit ausgewählten Sofware-Metriken die Komplexität der Systemkonstruktion ermittelt werden. Mit denselben Maßen lässt sich durch ausgewählte Qualitätskriterien die Qualität der inneren Konstruktion des Systems ermitteln.

Das Ziel der Vermessung ist es, die Größe, Komplexität und innere Qualität des Zielsystems festzustellen. Diese Maße werden benötigt um:

a) die Kosten der Sanierung zu schätzen
b) den Sinn der Sanierung zu beurteilen.

Oft stellt sich die Frage, ob es sich lohnt sich mit einem Legacy-System zu befassen, da ein Ergebnis zeigen könnte, von dem alten System nichts wiederverwenden zu können. In diesem Fall gibt es für den Anwender keine andere Möglichkeit, als wieder von vorne anzufangen und ein neues System zu konzipieren bzw. ein neues Standardpaket zu beschaffen. Dieser Umstand muss aber erst festgestellt werden. Es muss möglich sein, Kosten und Nutzen einer Sanierung mit den Kosten und Nutzen einer Neuentwicklung zu vergleichen. Das ist ohne Messung aber nicht möglich[SBS11].

3.4 Systemnachdokumentation

Sollte entschieden werden das vorhandene System wiederzuverwenden, wird im nächsten Schritt die Systemkonstruktion dokumentiert bzw. modelliert. Dieser Schritt sollte soweit wie möglich automatisiert stattfinden. Es werden Werkzeuge benötigt, um aus den Source Code Dateien, Datenbankschemen, Jobsteuerungsprozeduren und Testskripten modellhafte Dokumente abzuleiten. Diese Dokumente sollen als Hilfestellung für die Entwicklung dienen, den Source Code, die Daten und Prozess-Steuerungen neu zu implementieren, zu konvertieren oder zu kapseln. Die Nachdokumentation soll somit als Basis für die Systemsanierung dienen. Das Ergebnis der Nachdokumentation sind Modelle des vorhandenen Systems [BRSK14]. Jedes Modell gibt eine andere Sicht auf das System:

- das Datenmodell = Sicht auf die Datenstruktur
- das Prozessmodell = Sicht auf den Prozessablauf
- das Codemodell = Sicht auf die Codekonstruktion
- das Testmodell = Sicht auf die Testfälle.

Alle vier Modelle können vollumfänglich aus den vorhandenen Systemelementen automatisch abgeleitet werden [GSL12].

3.5 Systemmodellierung

Die wiedergewonnene Dokumentation erlaubt dem Anwender das Zielsystem neu zu modellieren. Es werden Cluster bzw. Teilsysteme gebildet, die unabhängig voneinander eingesetzt werden können – „Architectural Reengineering“. Die Ablöse großer Systeme soll inkrementell erfolgen. Ein Teilsystem nach dem anderen wird ersetzt. Es werden neue Modelle geschaffen um die Architektur des sanierten Systems festzuhalten – ein neues Datenmodell, ein neues Prozessmodell, ein neues Codemodell und ein neues Testmodell. Jedes dieser Modelle ist zunächst eine Kopie des alten Modells, das manuell nachgebessert werden muss. Der Systemarchitekt ist hier aufgefordert das jeweilige Modell zu optimieren [PJW14]. Die Dateien werden neu aufgeteilt, die Codemodule werden in den neuen Diagrammen umstrukturiert, die Prozesse werden auf Papier optimiert bzw. neugestaltet. Am Ende entsteht eine Reihe neuer Modelle, die neben den alten Modellen aufgestellt werden kann. Die einzige Frage, die nun noch offen bleibt ist, wie man die alten Modellelemente den neuen Modellstrukturen zuordnen kann. [GFS16]

3.6 Systempartitionierung

Die neuen Systemmodelle bilden die Basis, auf der das alte System aufgeteilt werden soll. Es werden Cluster bzw. Teilsysteme gebildet, die unabhängig voneinander eingesetzt werden können. Brodie und Stonebraker beschreiben die Techniken der Systemaufteilung in ihrem Buch „Migrating Legacy-Systeme“ [BS95]. Es komme darauf an, belastbare Schnittstellen zwischen den einzelnen Teilsystemen zu schaffen. Voraussetzung dafür ist die Migration der Daten in eine gemeinsame Datenbank. Falls die alten Komponenten behalten werden, müssen diese neue Zugriffsschnittstellen erhalten. Es muss möglich sein, dass alte und neue Systemkomponenten auf die gleichen Daten zugreifen können. Die Datenbank ist das Kit, das das System zusammenhält. Sie ist die Konstante, die Teilsysteme sind variabel und sollten beliebig ausgetauscht werden können. Direkte Beziehungen zwischen Teilsysteme werden gekappt. Das ist die Aufgabe der Partitionierung. Sollten zwei oder mehr Teilsysteme die gleiche Komponente verwenden, wird diese Komponente in jedes der Teilsysteme eingebaut, d.h. Redundanz ist der Preis der Unabhängigkeit. Es muss möglich sein,

Teilsysteme ohne Auswirkung auf andere Teilsysteme zu ersetzen. Das ist das Ziel der Partitionierung [BRSK14].

3.7 Inkrementelle Systemablösung

Sind die Teilsysteme einmal voneinander unabhängig, kann der Anwender beginnen diese Stück für Stück abzulösen. Dabei hat er mehrere Möglichkeiten:

1. Konvertierung, z.B. von COBOL in Java automatisch übersetzen zu lassen [Sn10].
2. Re-Implementierung bzw. Neukodierung in derselben oder einer anderen Sprache [Sn14].
3. Kapselung bzw. ohne Veränderung hinter einem Wrapper implementieren.

Diese Migrationsmöglichkeiten werden in anderen Beiträgen des Autors, sowie in der einschlägigen Literatur geschildert. Ziel dieses Artikels ist, die Modellierungstechniken zu behandeln [SWH10].

4 Redokumentation des Altsystems

Um das Altsystem zu modellieren muss es erst dokumentiert werden. Diese Nachdokumentation sollte möglichst automatisiert werden. Ein Reverse Engineering Tool parst den vorhandenen Code, um daraus diverse Tabellen wiederzugewinnen:

- eine Tabelle der externen Schnittstellen,
- eine Tabelle der internen Prozeduren,
- eine Tabelle der Prozedur Ein- und Ausgaben,
- eine Tabelle der verwendeten Datenvariablen,
- eine Tabelle der logischen Entscheidungen,
- eine Tabelle der Verarbeitungsregeln und
- eine Tabelle der Testfälle.

4.1 Tabelle der externen Schnittstellen

Eine Codekomponente ruft andere Komponenten auf, greift auf Datenbanken zu, verarbeitet Dateien, sendet und empfängt Nachrichten. In dieser Tabelle werden die Namen der fremden Objekte und deren Parameter aufgelistet. Sie werden auch als Eingaben, Aufgaben oder beides gekennzeichnet.
<Prozedurname> <Schnittstellentyp> <Call/IO-Stmt> <Inputobjekte> <Outputobjekte>

4.2 Tabelle der internen Prozeduren

Eine Legacy-Komponente enthält in der Regel mehrere interne Prozeduren. In PL/I sind es Prozeduren, in COBOL sind es Absätze, in Natural sind es Subroutinen. Diese Codeblöcke können sich gegenseitig aufrufen oder zu einem definierten Eintrittspunkt springen. In dieser Tabelle werden die internen Prozeduren und ihre Verbindungen zueinander festgehalten, zum einen namentlich und zum anderen mit deren Zeilennummer. Damit ist es möglich den internen Programmablauf von Moduleingang bis Modulausgang zu verfolgen.
<Prozedur> <AufgerufeneProzeduren> <Einsprungmarken> <Einsprungbedingungen>

4.3 Tabelle der Prozedur Ein- und Ausgaben

Eine interne Prozedur setzt Eingaben bzw. Argumente in Ausgaben bzw. Ergebnisse um. In dieser Datenflusstabelle werden die Namen der Eingabedaten in einer Spalte und die Namen der Ausgabedaten in einer anderen Spalte registriert. In einer dritten Spalte werden die Namen der Prädikate bzw. Bedingungsoperanden aufgelistet. Diese Variabel ist besonders zu vermerken, weil sie den Programmablauf steuert. Der Ablauf durch die Prozeduren wird durch den Inhalt der Prädikate bestimmt. Diese sind besonders wichtig für die Spezifikation der Testfälle.
<Prozedurname> <Eingabenamen> <Prädikatnamen> <Ausgabenamen>

4.4 Tabelle der verwendeten Variablen

Die Tabelle der verwendeten Variablen dient als Datenverzeichnis für die jeweilige Komponente. Jede Komponente verwendet eine Untermenge der Daten, die ihr zur Verfügung gestellt werden. Manche Variablen werden errechnet, andere gesetzt. Sie sind die Ausgaben der jeweiligen Komponenten. Variablen, die abgefragt werden, sind Prädikate. Weitere Daten werden nur benutzt um die Ausgabewerte zu erzeugen. Viele Daten in der Legacy-Komponente werden überhaupt nicht benutzt. Sie sind nur in den verwendeten Datenstrukturen vorhanden und werden hier ausgeklammert.
<Strukturname> <Datenname> <Datentyp> <Datenlänge> <Erklärung>

4.5 Tabelle der logischen Entscheidungen

In einer Komponente können viele Entscheidungen vorkommen. Entscheidungen in der Codeebene werden von den Steuerungsanweisungen getroffen, z.B.: die Auswahl- und Wiederholungsanweisungen „if“, „select“, „until“, „while“ und „perform“. In der Tabelle der logischen Entscheidungen werden diese Anweisungen mit ihren Bedingungsoperanden eingetragen. Die Bedienungsoperanden verweisen auf die Daten und Datenflusstabellen und verknüpfen die Ablauflogik mit dem Dateninhalt. Die

Entscheidungen werden über ihre Prozedurnamen, sowie über ihre Zeilennummer identifiziert und geordnet.
<Prozedurname> <Zeilennr> <Anweisungstyp> <Vorbedingung> <Anweisung>

4.6 Tabelle der Verarbeitungsregel

Eine Verarbeitungsregel verbindet ein bestimmtes Ausgabedatum mit den Anweisungen in denen das Datum ausgegeben wird, sowie mit den Entscheidungen, die zur Ausführung jener Anweisung führen. Das Format ist

<Ergebnis> = <Zuweisung> if <Bedingung> [Sn17]

Da ein Ergebnis an verschiedenen Stellen erzeugt werden kann, werden hier alle Zuweisungen von allen Stellen mit deren Bedingungen gesammelt. Es gilt hier für jede Komponentenausgabe, die Regel zu formulieren und an den Namen der Ausgabe anzuhängen.

4.7 Tabelle der Modultestfälle

Ein Modultestfall entspricht einem Ablaufpfad durch die Komponente, die in dieser Tabelle dokumentiert wird. In der Tat ist die Komponente eine Sequenz aller Anweisungen, die von dem Testfall durchlaufen wird. Sie kann als gerichteter Graph, als Struktogramm oder als Entscheidungstabelle dargestellt werden. Wichtig ist, dass die Bedingungen, die zu einem Testfall gehören, in der Reihenfolge ihres Vorkommens im Source Code aufgelistet sind. Die Bedingungen enthalten die Prädikate, dessen Inhalte den Fortlauf des Falles bestimmen. Die Zuweisung künstlicher Werte kann von einem Testautomaten geleistet werden. Dadurch können die Module ohne menschlichen Eingriff zum Laufen gebracht werden.
<Testfallkz> <Eingangsadresse> <Pfad durch nummerierte Anweisungen> <Prädikate>

5 Ein Modell des bestehenden Systems – das Ist-Modell

Ein Modell ist eine abstrakte Darstellung eines Gebildes, in dem unwesentliche Details zu Gunsten der Übersicht versteckt werden. Die Frage ist, welche Details zu unterdrücken und welche zu betonen sind. In einem Datenmodell könnten z.B. die Attribute der Datenentitäten selbst und ihre Beziehungen angezeigt werden. In einem Codemodell könnten z.B. die Attribute der Datenentitäten ausgeblendet und nur die Entitäten selbst und ihre Beziehungen angezeigt werden. In einem Codemodell könnten z.B. die einzelnen Anweisungen ausgespart und nur die Struktur des Codes dargestellt werden, d.h. die Komponenten-, Modul- und Prozedurnamen und deren Beziehungen zueinander [MK88].

Das Inhaltsverzeichnis der Codeeinheiten wäre somit das Modell. Wenn man die Aufrufbeziehungen hinzunimmt, wird das Modell noch informativer aber gleichzeitig weniger übersichtlich. Die Modelldarstellung wird durch die vielen Datennamen überlastet. Es gibt keine statische Modelldarstellung, die alle Anforderungen befriedigt, deshalb muss das Modell dynamisch bleiben. D.h. je nach Bedarf können Details ein- und ausgeblendet werden. Das Modell besteht aus den gespeicherten Entitäten und Beziehungen, die man aber in ihrer Gesamtheit nie zu sehen bekommt. Es kann immer nur eine Sicht auf das Modell anzeigt werden, diese kann aber bei Bedarf abgespeichert werden. Das ist die Philosophie hinter dem Modellierungswerkzeug SoftRepo [KELN10].

In SoftRepo sind sämtliche Entitäten und Beziehung eines Systems in einer relationalen Datenbank gespeichert. Für jeden Entitätstyp gibt es eine eigene Tabelle. Die Tabellen sind miteinander über den Schlüssel bzw. Hash der Entitäten verknüpft. Man kann eine Entität auswählen und bekommt sämtliche Beziehungen der Entitäten als Baum zur Ansicht. Manche Beziehungen können ausgeblendet und andere betont werden. Wesentlich ist die uniforme Sicht als Baum, die beliebig nach oben oder unten ausgebaut werden kann. Jede Sicht ist ein Ausschnitt aus dem großen Systembaum, in dem sämtliche Entitäten als Knoten und Beziehungen als Kanten vorkommen. Es gibt keine Grenzen zur Anzahl der Entitäten und Beziehungen. In der Software kann man ohne weiteres neue Knoten und Kanten dem Baum hinzufügen. Der Hauptanteil wird allerdings aus dem Source Code, den Datenstrukturen, den Dokumenten, den Testfällen und den bestehenden Modellen abgeleitet.

Wenn man wissen will, was mit einem bestimmten Datenelement passiert, muss man dieses Element nur namentlich auswählen und die Beziehungstypen, die man sehen möchte, anklicken, z.B. dort, wo das Datum benutzt, gesetzt oder abgefragt wird. In der Software kann man überprüfen von welchen anderen Funktionen eine bestimmte Funktion aufgerufen wird und welche diese selbst aufruft. Es ist auch ersichtlich, zu welchem Modul bzw. zu welcher Klasse diese Funktion gehört und an welcher Stelle diese definiert wurde. Das bedeutet, dass das Modell der Software sich dynamisch vor den Augen des Betrachters entfaltet. Die Verkettung der Baumknoten bildet einen Pfad durch den Baum.

Alle Informationen, die für die Wartung des Systems relevant sind, werden in dem Ist-Modell bzw. dem Systembaum abgebildet. Impact Analysen können jederzeit gestartet werden um die Folgen einer Änderung zu schildern. Auch Abfragen über die Nutzung ausgewählter Systemelemente wie Daten, Funktionen und Schnittstellen sind jederzeit möglich. In SoftRepo kann jeder Knoten des Baummodells beliebig expandiert werden. Die dynamische Darstellung ist dem statischen Bild vorzuziehen.

6 Vom Ist-Modell zum Soll-Modell durch Code-Inversion

Eine unentbehrliche Voraussetzung für die Transformation des alten Modells ist die Veränderung der Daten- und Prozedurnamen. In alten Source Codes sind es meistens gekürzte mnemotechnische Bezeichnungen. Eine Eins-zu-eins-Übertragung der Namen in das neue Modell macht an dieser Stelle keinen Sinn. Diese Kurznamen erscheinen nicht nur in sämtlichen Diagrammen, sondern auch im Source Code selbst, in dem diese unbedingt durch sprechende Namen ersetzt werden müssen. Erst danach kann man mit der Transformation in ein neues Modell starten. [Sn15]

Das Soll-Modell hat den gleichen Aufbau wie das Ist-Modell, ist aber mit einer anderen Struktur aufgestellt. Auf der ICSM Konferenz in Montreal 2002 stellte der Autor einen automatisierten Ansatz vor, um ein prozedurales Systemmodell in ein objektorientiertes umzusetzen. Dafür wurden Software-Inversionstechniken eingesetzt, die das Prozedurmodell auf den Kopf stellten. Aus COBOL Unterprogrammen wurden Java Superklassen abgeleitet, von denen die früheren Hauptprogramme geerbt haben. An der Spitze der neuen Klassenhierarchie standen die Datenzugriffsroutinen als allgemein verfügbare Services. Das Modell diente als Spezifikation für ein neues, in Java implementiertes Bausparsystem. Dieser Beitrag gewann damals den Preis als bestes Industrie-Paper. Der Ansatz in der Industrie fand wenig Beachtung, da er zwischen den Fronten lag. Die COBOL Programmierer haben ihn nicht akzeptiert, weil die neue Systemarchitektur für sie zu objektorientiert war. Die Java Entwickler haben es abgelehnt, weil die Architektur für sie nicht objektorientiert genug war. Die Modelle lehnten sich zu stark an die alten prozeduralen Lösungen. Fazit: Das Soll-Modell soll am besten von denjenigen konzipiert werden, die das System übernehmen. [Sn02].

7 Eine Fallstudie aus der Finanzwirtschaft

Um diesen Modellierungsansatz zu illustrieren, werden zwei Systeme aus der deutschen Finanzwirtschaft zitiert – eins für Leasing und eins für Darlehen. Beide Systeme sind in den 80iger Jahren entstanden und sind heute noch im Betrieb ist. Es handelt sich um klassische Legacy-Systeme, das eine in der Sprache COBOL mit einer hierarchischen Datenbank von IBM, das andere in Natural mit einer vernetzten Datenbank von der Software AG. Auf die exakte Größe der Systeme darf an dieser Stelle nicht weiter eingegangen werden, aber es handelt sich bei beiden um vergleichsweise sehr große Systeme, die Daten und Codezeilen in Millionenhöhe haben.

Aus Anwendersicht wäre es wünschenswert die Software dieser Systeme zu erneuern. Zusätzlich sind alle finanziellen Institutionen in der EU verpflichtet, eine aktuelle Dokumentation ihrer Finanzsysteme an die EU-Bankaufsichtsbehörde abzuliefern. Ziel der Dokumentation sollte einerseits die Erfüllung der Anforderungen der Bankenaufsicht sein und anderseits die Grundlage für eine Systemerneuerung in Form einer Spezifikation des Nachfolgesystems. Anhand dieser Spezifikation soll es möglich

sein, die Funktionalität der alten Systeme zu kopieren bzw. einer Wiedergewinnung der Fachlichkeit zu erzielen.

Da die Vorgaben für die Dokumentation der Finanzsysteme aus fachlichen Anforderungen, Datenmodellen, Prozessablaufbeschreibungen und Verarbeitungslogik bestehen, ist es relativ einfach, diese Mindestanforderungen durch eine Systemdokumentation abzudecken. Dazu werden keine Repository benötigt, da die Standarddokumente aus dem SoftRedoc System genügen. Wenn es jedoch um eine Modellierung der Fachlichkeit geht, ist das klassische Reverse Engineering überfordert. Ein Automat kann nicht erkennen, welche fachliche Relevant hinter dem Source Code steht. Außerdem darf man nicht den Fehler machen, die vorhandene Lösung mit dem ursprünglichen Problem zu verwechseln. Oftmals ging die Fachlichkeit bereits vor längerer Zeit verloren und stellt nun nur mehr ein weiteres Problem im Source Code dar oder ist schlichtweg durch mehrere Codeschichten mit verschlüsselten Datennamen und technischen Hilfsfunktionen überdeckt. Hunderte von Änderungen und Verschiebungen verzerren somit das Bild der ursprünglichen Lösungen. Uralte Codesegmente, die vor der Jahrtausendwende in das System programmiert wurden, belasten den ursprünglichen Code und blähen die Datenstrukturen künstlich auf. Es ist unmöglich, die Fachlichkeit aus dieser Akkumulation diverser Korrekturen und Ergänzungen wiederzugewinnen. Besser gesagt, ist es nicht möglich, solange die Datenbezeichner noch verschlüsselt und die fachlichen und technischen Anweisungen noch vermischt sind. Im ersten Schritt müssten also die Anwender die Daten umbenennen und die fachlichen von den technischen Anweisungen trennen. Da diese Voraussetzungen für eine Wiedergewinnung des Fachmodells mit enormen Kosten verbunden wäre, wurde dieser Reengineering Ansatz in dem Fall unterlassen. Was bleibt, ist nur ein Modell der vorhandenen Codestrukturen.

7.1 Ein dynamisches Modell der Codearchitektur

In diesem Modellierungsprojekt wurde ein dynamisches Modell der Codearchitektur geliefert. Die Entitäten und Beziehungen aus der statischen Analyse des Codes, der Daten, der JCL und der Benutzeroberflächen wurden in die SoftRepo Repository übertragen. Dort wurde das Tool SoftRepo benutzt um verschiedene Baumstrukturen zu visualisieren u.a.

- ein Datenbaum,
- ein Prozessbaum,
- ein Prozedurbaum,
- ein Oberflächenbaum und
- ein Regelbaum.

Alle Bäume können durch das Hinzufügen neuer Zweige, sowie durch die Verlängerung bestehender Zweige, leicht erweitert werden. Bäume können beliebig wachsen, daher der Begriff „dynamisches Modell“.

7.2 Der Datenbaum

Der Datenbaum zeigt in der IMS-Darstellung die Segmenthierarchie, erweitert durch die Schlüssel und Attribute der jeweiligen Segmente. In der DB2-Darstellung ist der Datenbaum eine Hierarchie der Tabellen mit deren Schlüssel und Attributen. Die Beziehung zwischen den Tabellen wird durch Verbindungsknoten dargestellt. So wird die fremde Tabelle als Unterbaum der Haupttabelle angehängt. In der Software kann man die Verweise von der einen Tabelle in die andere durch Mausklicks verfolgen.

Wenn man sehen möchte, wie die Objekte verwendet werden, dann klickt man die Beziehung, Input, Output oder Query an. Daraufhin werden die Module eingeblendet, die das gewählte Segment bzw. die gewählte Tabelle lesen, schreiben oder abfragen. Wenn man sehen möchte, wie die Datenattribute verwendet werden, klickt man die gleichen Beziehungen an. Daraufhin werden die Funktionen bzw. Prozeduren eingeblendet, die das ausgewählte Attribut benutzen, überschreiben oder abfragen. Auf jeder Hierarchiestufe bekommt man eine Sicht darauf, was mit den Daten geschieht. Natürlich sind die Datennamen immer noch mnemotechnische Bezeichnungen, sodass eine Verbindung zum Fachmodell nur über eine Namenszuordnung erfolgen kann.

7.3 Der Prozessbaum

Der Prozessbaum zeigt für den Mainframe die Jobhierarchie. An der Spitze des Prozessbaumes steht der Geschäftsprozess, ein von dem Anwender erfundener Batchprozess oder Online-Dialog. Die Knoten in dem Baum sind die einzelnen Jobs bzw. Transaktionen. Unter dem Job ist ein Unterbaum mit den einzelnen Jobschritten ersichtlich. Hinter der Transaktion befinden sich die Module, die darin aufgerufen werden. Analog zur Datenbank-Darstellung, kann man einen Baumknoten anklicken, um eine erweiterte Sicht auf die Prozessbeziehung zu bekommen. Die Beziehungen eines Jobs verweisen auf die Dateien, die der Job verarbeitet hat – als Eingabe oder als Ausgabe. Das gleiche gilt für die Online-Transaktionen. Gezeigt werden die Benutzeroberflächen, die von den Anwendern des Legacy-Systems bedient werden und die Datenbanken, auf die zugegriffen wird. Eine Untermenge dieser Beziehungen wird für jeden Jobschritt angezeigt. Da die Jobschritte und Transaktionen auch Prozeduren sind, werden diese in dem Prozessbaum weiter aufgerollt.

7.4 Der Prozedurbaum

Der Prozedurbaum beginnt mit einem einzigen Programm. Dieses Programm könnte ein Batch-Jobstep oder ein Dialogprogramm in einer Online Transaktion sein. Die Knoten des Prozedurbaumes sind die Prozeduren, die zu dem Programm gehören. Ihre Beziehungen sind vielfältig. Zum einen kann eine Prozedur – in COBOL sind das Sections und Paragraphen, in Natural sind das Subroutinen – andere Prozeduren aufrufen oder von anderen Prozeduren aufgerufen werden. Prozeduren können auch Daten verarbeiten, zuweisen, benutzen oder abfragen. Es sind die Prozeduren, die die Daten verändern. Prozeduren beinhalten Aktionen und Bedingungen, die wiederum Daten verwenden. Die Bedingungen bestimmen, welche Aktion als nächste aufgeführt wird. Mit einem Mausklick kann man die Argumente, Ergebnisse und Prädikate jeder Funktion anzeigen lassen. Über dieses Fenster ist es möglich, anschließend in den Datenbaum hineinzuspringen. Auf diese Weise kann man zwischen Daten- und Funktionsbaum hin und her wechseln und somit beide Bäume miteinander vereinen. Denn Modellierung ist letzten Endes eine Frage der Sichtweise. Der Baum ist das optimale Darstellungsmittel um verschiedene Sichten zusammenzuführen.

7.5 Der Oberflächenbaum

Der Oberflächenbaum zeigt die Benutzeroberflächen und ihren Unterbaum an. Beide Programmiersprachen – COBOL wie auch Natural – haben sogenannte Masken, die Struktur und Inhalt der Benutzeroberflächen spezifizieren. In ihnen enthalten sind die Felder in der Maske und die Gestalt der Maske in Bezug auf Zeilen und Spalten. Masken verweisen auf den Dialogschritt, in dem sie sich befinden, sowie auf die Datenelemente, die sie anzeigen und erfassen. Diese Beziehungen werden als Querverweise zu dem Prozedurbaum sowie zu dem Datenbaum geschildert. Da die Maskenbeschreibung auch in einer Art Prototyp der Oberflächensicht umgesetzt wird, ist es möglich diese Sicht in den Oberflächenbaum einzublenden. Damit kann man sehen, wie die Oberfläche, die ein- und ausgegeben Daten und der Dialogschritt aussieht. Hier werden also drei Sichten miteinander kombiniert.

7.6 Der Geschäftsregelbaum

Der Regelbaum hat als Hauptknoten nur die geschäftsrelevanten Ausgabedaten. Das unterscheidet ihn von dem Datenbaum, in dem alle benutzten Daten enthalten sind. Hier werden die Datennamen einem Verzeichnis aller relevanten Geschäftsdaten gegenübergestellt. Um in den Baum aufgenommen zu werden, muss ein Datenelement eine Ausgabe sein und zu den geschäftsrelevanten Daten gehören. Technische Arbeitsdaten und Zwischenergebnisse werden dadurch ausgefiltert. Es obliegt dem Benutzer, die Liste der geschäftsrelevanten Daten zusammenzustellen.

Nachdem der Baum der geschäftsrelevanten Ausgabedaten (fachliche Ergebnisse) aufgebaut ist, werden den elementaren Daten bzw. dem Basisknoten ein Unterbaum mit weiteren Knoten hinzugefügt. Die erste Stufe der Unterknoten repräsentiert die Namen der Prozeduren, in denen das Ausgabedatum zugewiesen bzw. erzeugt wurde. In der zweiten Unterstufe sind die Anweisungen, die das Datum verändern. In der dritten Stufe folgen die Vorbedingungen jener Zuweisungen. Es entsteht folgende Datenhierarchie:

Ausgabe-Datenelement <X> vom Typ <X1>
- Wird in Modul <Y>.Prozedur_<Y1>
  - durch Zuweisungsanweisung <Y_99> gesetzt
    - wenn Bedingung <Z> hält.

Falls die Prozedur, in der die Variabel gesetzt ist, von einer anderen Prozedur aufgerufen wird, wird die Aufrufkette bis an ihr Ende weiter aufgerollt. Hier gibt es eine Überlappung mit dem Prozedurbaum. Der Unterschied ist, dass die Zuweisungen und deren Bedingungen auch gezeigt werden. Man bekommt den vollständigen Datenfluss von dem Eingang in die Komponente bis zur Ausgabe der relevanten Business-Datenwerte. Der gesamte Pfad, einschließlich Aufrufe, Zuweisungen und Bedingungen gilt als Geschäftsregel für diese Ausgabe. Die Geschäftsregel spielt eine besondere Rolle bei der Re-Implementierung des Codes. Es wurde überlegt, ob man die Regeln nicht in Gurkin transformieren sollte, um sie leichter verständlich zu machen. Am Ende wurde dagegen entschieden, weil die Bedingungen dadurch verzerrt sein würden [WH12].

7.7 Visualisierung des dynamischen Modells

Endziel der Modellierungs-Fallstudie war ein Word-File, das die Anwender bei Bedarf an die Aufsichtsbehörde übergeben können. Die Erstellung erfolgte in zwei Schritten, die im Folgenden näher erläutert werden:

Im ersten Schritt wurde aus dem Modell-Repository ein einziger großer HTML-Baum für jede Online-Transaktion und jeden Batchprozess generiert. Dieser Baum wurde durch ein Inhaltsverzeichnis und eine Textbeschreibung des Vorgangs ergänzt. Die Textbeschreibung wurde aus den Modulkommentaren zusammengestellt. Sie kann nach Bedarf von dem Ersteller der Dokumentation verändert werden. Auch das Inhaltsverzeichnis lässt sich editieren. Die anderen Inhalte, die aus der Modell-Repository entnommen wurden, sind fix.

Im zweiten Schritt wurde der HTML-Baum mit allen Inhalten in ein MS-Word Dokument umgesetzt. Das Word-Dokument ist sehr groß, aber lässt sich ohne weiteres ausdrucken. Somit bekommt man eine Textdatei mit mehreren hundert Seiten für jede Transaktion, die als revisionsfähige, technische Dokumentation gilt [Ri10]. Voraussetzung für das Verständnis sind dennoch technisches Know-How und Kenntnis

von dem Inhalt des Systems. Wesentlich ist, dass die Transaktionen und Prozesse dokumentiert sind. Wie dies geschieht, muss erst beschlossen werden. Ein Abnahmeverfahren der Aufsichtsbehörde wird entscheiden, ob die Dokumentation ausreichend ist oder nicht.

Die nachträgliche Dokumentation dient nicht nur der Aufsichtsbehörde, sondern soll auch dem Service und Support helfen, den bestehenden Code schneller und besser zu verstehen. Zu diesem Zweck erhält das zuständige Personal den HTML-Baum und das dahinterliegende Repository. In JavaDoc kann es durch das Systemmodell navigieren und die Stellen suchen, die geändert werden müssen oder die von Änderungen betroffen sind – Impact Analyse [LFR13]. Der HTML-Baum könnte als Spezifikation für eine Neuentwicklung oder Re-Implementierung des Systems dienen. Alle Informationen, die ein Entwickler braucht um die vorhandene Geschäftslogik in eine andere Sprache zu versetzen, sind darin enthalten.

8 Zusammenfassung

Der SofRedoc Ansatz, der hier geschildert wird, ist einer von vielen Versuchen mit alten Legacy-Systemen fertig zu werden. Andere Ansätze sind der Rigi Ansatz von Hausi Müller [MK88], der AMELIO Ansatz der Firma Delta Technologies [Sc17] und der Bauhaus Ansatz der TU Stuttgart [RVP06]. SofRedoc wurde schon mehrfach eingesetzt, um Legacy-Systeme zu dokumentieren. Ergänzt wird er durch das Repository Tool SoftRepo, um die Systemarchitektur zu modellieren.

Hier wurden diese Tools für die Modellierung bestehender Finanzsysteme eingesetzt. Das Ziel war, eine technische Beschreibung der Applikation aus dem bestehenden Source Code abzuleiten. Die Mittel dazu bestanden aus der statischen Codeanalyse, gekoppelt mit einem dynamischen Architekturmodell, und der HTML-Bäume als Darstellungstechnik. Das Architekturmodell vereinigt verschiedene Sichten auf das System – eine Datensicht, eine Prozesssicht, eine Prozedursicht und eine Regelsicht – um die allumfassende Beschreibung des Codes zu erstellen. Sämtliche Sichten werden automatisch aus dem Code gewonnen. Dieser Modellierungsprozess ist besonders preiswert. Eine komplexe IMS-Transaktion kann in einem Tag modelliert und dokumentiert werden. Falls der Code verändert wird, kann der Prozess für die betroffene Transaktion jederzeit wiederholt werden. Für die rasche und preiswerte Nachdokumentation vorhandener Legacy-Systeme hat sich der Ansatz bewährt. Die Zukunft wird zeigen, ob die Dokumentation zur Erhaltung der hier erwähnten Finanzapplikationen dient. Jedenfalls ist die Forderung nach Revisionsfähigkeit abgedeckt. Die automatisch erzeugte Dokumentation erfüllt diesen Zweck.

Literaturverzeichnis

[BS95] Brodie, M., Stonebraker, M.: Migrating Legacy Systems, Morgan-Kaufmann Pub., San Francisco, 1995

[BRSK14] Becker, S., Riebisch, M., Sauer, S., Klatt, B., Ruhroth, T.: „Modellbasierte und Modellgetriebene Softwaremodernisierung“, GI Softwaretechnik Trends, Band 34, Heft 2, Mai 2014, S. 30

[GFS16] Grieger, M., Fazal-Baqaie, M., Sauer, S.: „A Method Base for the Situation Specific Development of Model-Driven Transformation Methods“, GI Softwaretechnik Trends, Band 36, Heft 3, August 2016, S. 67

[Go16] Goll, M.: „Modellbasierte Migration von Unternehmensanwendungen durch Kombination eines Top-down und Bottom-up Ansatzes“, GI Softwaretechnik Trends, Band 36, Heft 3, August 2016, S. 63

[GSL12] Güldali, B., Sauer, S., Lohr, P.: „Entwicklung eines Softwarewerkzeuges für die modell-getriebene Migration betrieblicher Informationssysteme“, GI Softwaretechnik Trends, Band 32, Heft 2, Mai 2012, S. 5

[KELN10] Kuhn, A./ Erni, D./ Loretan, P./ Nierstrasz, O.: „Software Cartography – thematic software visualization with consistent Layout“ Journal of Software Maintenance and Evolution, Vol. 22, No. 3, April 2010, S. 191

[LFR13] Lehnert, S./Farooq, Q./Riebisch, M.: Rule-based Impact Analysis for heterogeneous Software Artifacts”, IEEE Proc.of CSMR2013, Genova, March 2013, S. 209

[MK88] Müller, H./Klashinsky, K.: „RIGI - A System for Programming in the Large“ in IEEE Proc. of 10th ICSE, Singapore, April 1988, p. 80

[PJW14] Pandey, G., Jelschen, J., Winter, A.: „Towards Quality Models in Software Migration“, GI Softwaretechnik Trends, Band 34, Heft 2, Mai 2014, S. 40

[Ri10] Rinn, J: „Einführung in JavaDoc“, (2010) http://java.sun.com/j2se/javadoc,

[RVP06] Raza, A., Vogel, G., Plödereder, E.: „Bauhaus – A Tool Suite for Program Analysis and Reverse Engineering“, Ada-Europe, Lecture Notes in Computer Science, Vol. 4006, Springer Verlag, Berlin, 2006, S. 71

[SBS11] Sneed, H., Baumgardner, M., Seidl, R.: Software in Zahlen, Hanser Verlag, München, 2011

[Sch17] Schilling, D.: „COBOL und PL/I Anwendungen automatisch wieder verstehen“, GI Softwaretechnik Trends, Band 37, Heft 2, Mai, 2017, S. 56

[Sn84] Sneed, H.: “Software Renewal – A German Case Study”, IEEE Software, Vol. 1, No. 3, 1984, S. 56

[Sn02] Sneed, H.: „Transforming Procedural Programm Structures to object-oriented Class Structures“, IEEE of 18th ICSM, Computer Society Press, Montreal, Okt. 2002, S. 286.

[Sn10] Sneed, H.: „Automatisierte Migration alter COBOL Programme in Java“, GI Softwaretechnik Trends, Band 30, Heft 2, Mai 2010, S. 62

[Sn14] Sneed, H.: „Reverse-Modellierung von Traceability zwischen Code, Test und Anforderungen“, GI Softwaretechnik Trends, Band 34, Heft 2, Mai 2014, S. 34,

[Sn15] Sneed, H.: “Namensänderung in einem Reverse Engineering Projekt“, GI Softwaretechnik Trends, Band 35, Heft 2, Mai 2015, S. 23

[Sn17] Sneed, H.: „Wiedergewinnung von Geschäftsregeln aus einem Legacy Anwendungssystem“, GI Softwaretechnik Trends, Band 37, Heft 2, Mai, 2017, S. 26

[SpL03] Seacord, R., Plakosh, D., Lewis, G.: “Modernizing Legacy Systems”, SEI Series in Software Engineering, Addison-Wesley, Boston, 2003

[SV15] Sneed, H., Verhoef, C.: „Reverse Engineering a Visual Age Application“, IEEE Proc of ICSM2015, Computer Society Press, Bremen, 2015, S. 201

[SWH10] Sneed, H., Wolf, E., Heilmann, H.: Softwaremigration in der Praxis, dpunkt.verlag, Heidelberg, 2010

[Te10] Teppe, W.: „Wiedergewinnung von Informationen über Legacy-Systeme in Reengineering Projekten“, GI Softwaretechnik Trends, Band 30, Heft 2, Mai 2010, S. 68

[Zu93] van Zuylen, H.J.: The REDO Compendium – Reverse Engineering for Software Maintenance, John Wiley & Sons, New York, 1993, S. 225-248