Software-Reengineering
Überblick, Ausprägungen und Abgrenzung

Andre Ufer

a09008@hb.dhbw-stuttgart.de

Zusammenfassung. Diese Seminararbeit hat das Ziel, einen Überblick über die Disziplin des Software-Reengineering zu verschaffen. Um das zu erreichen, werden zum einen die einzelnen Ausprägungen erläutert, zum anderen eine Abgrenzung gegenüber Techniken beschrieben, die zwar ähnliche Ziele verfolgen jedoch auf einem höheren bzw. niedrigerem Abstraktionsniveau ansetzen. Außerdem soll der Begriff des Software-Reengineerings erläutert werden und auf seine Unterschiedlichen Bedeutungen hin untersucht werden. Im Rahmen dieser Arbeit werden ebenfalls Werkzeuge vorgestellt, mit denen sich das Reengineering automatisieren oder vereinfachen lässt. Viele dieser Werkzeuge sind dabei Ergebnisse aus der Forschung und somit im Rahmen von Projekten an Universitäten oder forschenden Unternehmen aus der Wirtschaft entstanden. Es wurde aber auch untersucht, ob es Werkzeuge aus dem Open-Source-Bereich gibt. Zusätzlich sollen Best-Practices beschrieben werden, mit denen bereits Reengineering-Projekte erfolgreich durchgeführt wurden. Eine wirtschaftliche Betrachtung wurde nicht durchgeführt, jedoch wurden einige Aspekte davon behandelt.

1 Einleitung

Die Idee des Software-Reengineerings ist keine neue Erscheinung. Am Anfang der Computergeschichte galt Software bis zum Ende der 1950er Jahre als notwendiges Beiwerk zur Hardware. Anfang der 1960er Jahre entstanden allerdings die ersten lösungsorientierten Hochsprachen wie FORTRAN und COBOL. Damit wurde die Entwicklung von Software zu einem eigenen Wirtschaftszweig, der sich unabhängig von der Hardware-Entwicklung ständig weiterentwickelte. Mit der zunehmenden Komplexität von Computerhardware vergrößerte sich auch das Einsatzfeld. Das hatte zur Folge, dass auch die Software immer größer und komplexer wurde. Mit der Zeit entstanden dadurch immer schwerer wart- und erweiterbare Programme.
Diese Entwicklung führte Mitte der 1960er-Jahre zur Software-Krise. Dieses Phänomen beschreibt die Problematik, dass die Entwicklung und Wartung von Software teurer wurde als die der Hardware. Um dem entgegenzuwirken und um die Entwicklung von Software unter wissenschaftlichen Aspekten betreiben zu können, gründete, definierte und etablierte man die Ingenieurs-Disziplin des Software-Engineering.[Ran69]

Mit der Disziplin des Software-Engineerings entstanden neue Programmiersprachen mit neuen Paradigmen. Hier ist vor allem die objektorientierte Programmierung (OOP) hervorzuheben, da sie sich bis heute als als eine der verbreitetsten Paradigmen durchgesetzt hat.
Einige dieser neuen Paradigmen und die Ergebnisse in der Forschung des Software-Engineerings haben es, zumindest in der Theorie, deutlich vereinfacht, modularere und wartbarere Software zu entwickeln. Neben der Neuentwicklung von Anwendungen stellt sich die Frage, ob es Wege gibt, alte und nahezu unwartbare Software wieder in einen "gesunden" Zustand zu bringen. Diese Überlegungen sind die Grundlage für das Software-Reengineering.

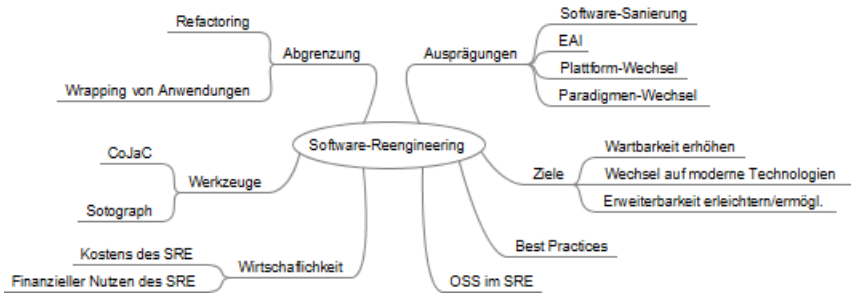

Abb. 1. Begriffs-Übersicht

2 Ziele des Reengineerings

Wie die Entwicklung von Software bereits in den 1960ern zeigte, wurde es immer teurer, Anwendungen zu warten und weiterzuentwickeln. Um diesen Problemen entgegenzuwirken gibt es jedoch nur zwei Wege.

- Neuentwicklung von bestehenden Anwendungen
- Überarbeiten bestehender Anwendungen

Natürlich bleibt auch noch die Möglichkeit, alte Software ("Legacy-Software") weiterzuentwickeln und die damit verbundenen Kosten hinzunehmen. Hierbei muss man jedoch das Risiko in Kauf nehmen, dass die Entwicklungs- und Wartungskosten noch weiter steigen, sodass man früher oder später aus wirtschaftlichen Gründen doch gezwungen ist, ein Reengineering der Software durchzuführen.
Neben der Wiederherstellung oder Erhöhung gibt es noch weitere Ziele, aus denen ein Reengineering durchgeführt werden kann. Einhergehend mit der Wartbarkeit ergibt sich auch das Problem der schlechten Erweiterbarkeit von Software.
Auch ein Wechsel auf moderne Technologien oder sich im Laufe der Zeit ändernde

Anforderungen können Anreiz und Grund für Software-Reengineering bieten.

Begriffe wie *Kopplung*, *Kohäsion* und *Schnittstelle* sind zwar vor allem in der objektorientierten Programmierung (OOP) gebräuchlich, werden hier aber paradigmenübergreifend verwendet, da auch prozedurale Programmiersprachen Modularisierung und damit Kopplung und Kohäsion enthalten, sowie Schnittstellen anbieten.

2.1 Erhöhen der Wartbarkeit

Die Wartbarkeit ist ein entscheidendes Kriterium für die Qualität von Software. Im Laufe des mitunter Jahrzehnte dauernden Lebenszyklus müssen ständig Änderungen vorgenommen werden. Die benötigten Änderungen haben dabei die verschiedensten Gründe haben. In erster Linie geht es um die Entfernung von Fehlern. Das können zum einen logische Fehler sein, die den Programmablauf beeinträchtigen (Bugs) oder z.B. einfache typografische Fehler sein. Dabei muss jedes Mal Hand an den Quellcode gelegt werden. Bei einem langen Lebenszyklus und/oder einem größeren Programmierteam arbeiten dabei ständig verschiedene Personen am selben Code. Neben den individuellen Programmierstilen und dem unter Umständen Nicht-Einhalten von Coding-Conventions führt auch oftmals Zeitdruck dazu, dass der Code immer schlechter wartbar wird. Sogenannter tt-oter Codetritt vermehrt auf und Zusammenhänge zwischen einzelnen Modulen sind immer schlechter zu Überblicken. Auch die Code-Kommentierung wird oft vernachlässigt und führt dazu, dass einzelne Algorithmen und Codezeilen nicht mehr Nachvollziehbar sind und sich die Folgen einer (unbedachte) Änderung nicht abschätzen lassen.
Die Folge davon ist, dass mit dem Code auch die Anwendung selbst immer mehr an Qualität verliert. Dieses Phänomen wird auch als *Softwarealterung* bezeichnet. Um dem entgegenzuwirken, sollte ein Reengineering in Betracht gezogen werden, um den Code wieder in einen "gesundenünd wartbaren Zustand versetzt. Das bedeutet, dass eine saubere Paketierung/Modularisierung modelliert und umgesetzt wird oder im Laufe der Zeit entstandene, zu konkrete Schnittstellen überarbeitet werden. Kurz: eine hohe Kohäsion und niedrige Kopplung sollen hergestellt werden. Der Code muss nachvollziehbar und gut dokumentiert bzw. kommentiert sein.

2.2 Erhöhen der Erweiterbarkeit

Einhergehend mit der einer schwierigeren Wartbarkeit verschlechtert sich auch die Erweiterbarkeit von Code. Eine Software bleibt nur selten in exakt dem Funktionsumfang, wie sie es bei Veröffentlichung war. Mit jeder neuen Versionen steht neben Fehlerbehebung auch die Erweiterung der Funktionalität an. Ein wie im Kapitel Ërhöhen der Wartbarkeit"beschriebener Quellcode macht auch das Erweitern immer schwieriger. Oft fehlen Schnittstellen zu bestehenden Modulen und die Anpassung dieser ist zu riskant. Somit neigt man dazu, für jede zusätzliche Funktion Ïndividuallösungenßu entwickeln, die das System mit

der Zeit immer komplexer und somit schwerer wart- und erweiterbar machen. Meistens gerät man so in eine Art Teufelskreis", dem man z.B. mit gezieltem Reengineering durchbrechen kann. Die konkreten Ziele sind dabei dieselben wie im vorherigen Kapitel: hohe Kohäsion, niedrige Kopplung, saubere Dokumentation, einheitliche Schnittstellen.

2.3 Wechsel auf moderne Technologien

Neben der Qualität des Codes spielen auch die eingesetzten Technologien eine Rolle bei der Qualität und Aktualität von Anwendungen. Zwar bedeuten moderne Technologien nicht zwangsläufig auch eine hohe Qualität von Software, jedoch bieten sie meist mehr Möglichkeiten, Software zu testen, zu betreiben, oder an größeren Gruppen daran zu arbeiten. Auch das Angebot an Nachwuchskräften, die die alten Technologien beherrschen, kann eine Weiterentwicklung unter Umständen schwierig bis unmöglich gestalten. Ein weiterer Punkt ist das Betreiben von Hardware-Umgebungen, die für diese benötigt werden. Hier kann ebenfalls das Problem auftreten, dass Personal mit Wissen über die eingesetzte Technik immer seltener und schwerer zu rekrutieren wird. Auch die Kosten für Wartung und Ersatzteile dieser Systeme kann sich auf lange Sicht als unwirtschaftlich herausstellen.
Aus diesem Grund stellt ein Reengineering mit dem Ziel, auf alte Technologien basierende Anwendungen auf neue zu Übertragen, eine Lösung dieser Probleme dar. Dafür muss ein Konzept erarbeitet werden, dass fachliche Anforderungen des alten Systems in ein neues System übernimmt. Ein Beispiel dafür wären z.B. alte, in Cobol entwickelte Großrechner-Verfahren, die in JavaEE-Anwendungen übertragen werden.

2.4 Anpassung an veränderte Anforderungen

Code wird im Laufe des Lebenszyklus einer Software regelmäßig geändert. Meistens handelt es sich dabei um Bugfixing oder funktionale Erweiterungen. In einigen Fällen sind es jedoch die fachlichen Anforderungen, die sich im Laufe der Zeit geändert haben. Diese Änderungen können verschiedene Ursachen haben. Beispiele für Änderungen von Anforderungen:

- **Änderung von Geschäftsprozessen**
  Eine Anwendung zur Abwicklung von Beschaffungsabläufen wurde anhand einer bestehenden Unternehmens- und der dazugehörigen Prozessstruktur erstellt. Nach einigen Jahren ändert sich die Unternehmensstruktur und damit die Zuständigkeiten und zugrunde liegende Prozesse. Die bisher verwendete Anwendung unterstützt diese neuen Strukturen nicht und muss entweder ersetzt oder überarbeitet werden.
- **Änderung von gesetzlichen Vorgaben**
  Vor allem in der öffentlichen Verwaltung gelten bei der Verarbeitung, Speicherung und Archivierung etliche Gesetze und Regelungen erfüllt werden.

Viele dieser Regelungen ändern sich im Abstand von ein paar Jahren, werden erweitert oder selten auch vereinfacht. Das bedeutet in der Regel, dass sich die Anforderungen regelmäßig in unterschiedlichem Umfang ändern.

Beide Beispiele erfordern Anpassungen am Quellcode. Je nach Umfang des Änderungsbedarfs kann bei der Bearbeitung von einem Reengineering auf Basis von veränderten Anforderungen gesprochen werden.

3 Ausprägungen des Reengineerings

Nachdem nun die wichtigsten Ziele des Software-Reengineerings erläutert wurden, sollen in diesem Kapitel die Mittel und Wege beschrieben werden, mit denen diese Ziele erreicht werden können. Bei den meisten Ausprägungen kommt es zu Überschneidungen. Beispielsweise kann bei der Umsetzung einer serviceorientierten Architektur die Plattform gewechselt werden oder die bestehenden Softwaremodule grundlegend saniert werden, um auf zukünftige Änderungen besser vorbereitet zu sein.

3.1 Software-Sanierung

Unter Softare-Sanierung versteht man die Grundlegende Überarbeitung der Module, Pakete und Komponenten einer bestehenden Software. Vereinfacht ausgedrückt stellt sie die abstraktere Version des Refactorings dar bzw. verwendet es nur als eines von mehreren Werkzeugen. Bei der Durchführung einer Sanierung bleiben Plattform und Paradigma in der Regel behalten. Der Schwerpunkt wird darauf gelegt, den Code selbst in einen gesunden Zustand zu überführen, auf dem dann ein regelmäßige Wartung vereinfacht wird. Damit stellt die Software-Sanierung den Vorgang dar, eine vernachlässigte Software-Wartung oder -Modellierung zu korrigieren und eine zukünftige Wartung möglich machen. Die Die Sanierung hat weiterhin den Zweck, eine Software nicht nur wart- und erweiterbar sondern auch testbar zu machen. Insgesamt verfolgt man also mehrere Ziele:

- Erhöhung der Wart- und Erweiterbarkeit durch Modularisierung, Restrukturierung
- Dokumentation des neuen Codes
- Sicherstellen der Korrektheit mittels (Unit-)Testing

3.2 Enterprise Application Integration und SOA

Unter Enterprise Application Integration versteht man die Integration von Ge schäftsprozess-unterstützender Software zu einem Konstrukt, dessen Komponenten in einer einheitlichen Erscheinung auftreten.[Kel02] Mit “Erscheinung” kann sowohl eine Benutzer-Schnittstelle (Frontend) als auch das interne Zusammenspielen der einzelnen Softwaremodule (Backend) bezeichnet werden. In der Regel werden findet eine Integration im Frontend- und Backend-Bereich statt.

SOA (Serviceorientierte Architektur) ist ein weiteres Konzept, um verschiedene Softwarekomponenten zu einem ineinandergreifenden Gesamtsystem zusammenzuführen. Es versteht sich als Erweiterung der EAI, indem es beim Entwurf einen serviceorientierten Lösungsweg vorgibt. Dieser lose Zusammenschluss ergibt sich über die Bereitstellung von einzelnen unabhängigen Softwarekomponten, die als *Services* bezeichnet werden. Jeder Service stellt also einen individuelle, funktionalen Kontext dar, der durch externe Programme über *Service-Verträge* ansprechbar ist und somit seine Funktionalität zur Verfügung stellt. [Erl08]

Beide Ansätze finden z.B. bei modernen Portallösungen Anwendung. Sowohl EAI als auch SOA lassen sich als Ausprägungen des Software-Reengineering bezeichnen, da mit beiden Techniken eine moderne Programmumgebung geschaffen werden kann, in der bestehende Anwendungen in der Art umgestaltet werden, dass sie Schnittstellen (EAI) oder Services (SOA) anbieten, um untereinander zu kommunizieren zu können (Backend-Integration) und um ihre Funktionen von einem gemeinsamen Frontend abrufen und die Daten darstellen lassen.
Diese Ausprägungen werden meist dann verfolgt, wenn Software an neue Anforderungen angepasst oder ein Wechsel auf moderne Technologien durchgeführt werden soll.

3.3 Plattform- und Paradigmenwechsel

Eine weitere Ausprägung des Software-Reengineerings ist der Wechsel der Plattform, auf der eine Anwendung läuft, bzw. der Wechsel auf ein neues Programmier-Paradigma. Bei dieser Ausprägung spielt wieder der Wunsch bzw. das Ziel, auf moderne Technologien zu wechseln, eine wichtige Rolle.

Der Begriff *Plattform* beschreibt in erster Linie die zugrunde liegende Infrastruktur sowohl hard- als auch softwareseitig. Oft bedingt ein Plattformwechsel auch einen Paradigmen- bzw. Sprachenwechsel. Umgekehrt ist es eher selten der Fall. Im folgenden werden die Plattformbeispiele genannt und auf die Auswirkungen auf Paradigma und Sprache eingegangen.

Großrechner Der Großrechner wird vor allem in Bereichen eingesetzt, in denen Zuverlässigkeit und Datendurchsatz gefragt sind und ist demnach vor allem für Banken, Versicherungen und auch in der öffentlichen Verwaltung attraktiv. Das Betreiben eines Großrechner ist eine sehr kostspielige Angelegenheit und die Rekrutierung von Nachwuchskräften wird schwieriger, da der aktuelle Trend in der Ausbildung sich in Richtung JavaEE und dem .NET-Bereich entwickelt und Großrechnersprachen wir Cobol immer seltener beherrscht werden. Somit stellt sich für Unternehmen und Einrichtungen die Frage, ob sich der weitere Betrieb des Großrechners wirtschaftlich lohnt, oder ob ein Reengineering der Verfahren hin zu einer alternativen Plattform mehr Sinn ergibt. Als Alternativen kommen hier je nach bisherigem Anwendungstyp Desktopanwendungen oder

Webservice-basierte Client-Server- oder Webanwedungen infrage. Am Markt existieren Werkzeuge, die z.B. eine Transformation von Cobol- in Java-Code vornehmen können, um so das Reengineering in begrenztem Umfang automatisiert durchführen zu lassen [Pro11]. Meistens wird beim Weggang vom Großrechner auf eine moderne, Objektorientierung unterstützende Sprache gesetzt. Infrage kommen hier z.B. Java in der Enterprise-Version oder ASP.NET MVC. Auch funktionale Programmiersprachen wie z.B. Haskell kommen als Alternative infrage.

Client-Server Eine Client-Server-Architektur (CSA) beschreibt ein Konzept, bei der verschiedene Teile des Funktionsumfangs einer Anwendung zentral zur Verfügung gestellt wird (Server) und von dort über die Clients in Anspruch genommen werden kann. Moderne Ansätze sind z.B. Rich- und Thin-Clint-Architekturen. Diese unterscheiden sich in erster Linie dadurch, wie viel Funktionalität jeweils vom Client bzw. dem Server bereitgestellt wird.
Für die Umsetzung dieser Architekturen gibt es mehrere Frameworks am Markt, wie z.B. die auf Java basierende Eclipse Rich Client Plattform [Ecl11].

Webanwendung Neben der Client-Server-Architektur bieten auch Webanwendungen eine Alternative zum Großrechner. Bietet jedoch einige Vorteile gegenüber der CSA. Aus Sicht der Entwickler entfallen hier vor allem die Wartung von zwei verschiedenen Anwendungen. Auch die Pflege der Schnittstellen fällt kaum bis gar nicht ins Gewicht. Bei der Auswahl zwischen CSA oder Webanwendung sollte vor allem auf die Komplexität der Datenbearbeitung geschaut werden. Eine komplexe Bearbeitung kann z.B. auf den vergleichsweise gering ausgelasteten Clientrechnern ausgeführt werden, während der Server für die Datenhaltung und Übertragung zuständig ist. Hier wäre also eine CSA sinnvoll eingesetzt. Bei weniger komplexen Bearbeitungen oder wenigen Begrenzungen in der Beschaffung und Bereitstellung von Hardwareinfrastruktur kann jedoch auch eine Webanwendung verwendet werden.
Ebenso wie bei der CSA können auch hier z.B. JavaEE, ASP.NET oder auch PHP inkl. eines Frameworks wie Zend eingesetzt werden.

4 Abgrenzung gegenüber anderen Techniken

Neben den bisher angesprochenen Methoden des Reengineering gibt es noch weitere Vorgehensweisen, mit denen sich ähnliche Resultate erzielen lassen. Davon sollen zwei etwas genauer betrachtet werden

4.1 Reengineering vs. Refactoring

Aufgrund des sehr niedrigen Abstraktionsniveaus des Refactoring, dass den Fokus weniger auf das System als Ganzes wirft, sondern eher direkt auf Klassen- und Modulebene arbeitet, gilt es selbst nicht als Reengineering-Ausprägung.

Dabei gehört das Refactoring zu den wichtigsten Mitteln, wenn es um die Verbesserung von Code-Qualität geht. Es ist Bestandteil der Softare-Sanierung. Die beim Refactoring verwendeten und bekannten Best-Practices lassen sich demnach auch auf die Sanierung übertragen. Beispiele dafür sind das Umsetzen von Paradigmen-spezifischen Eigenheiten wie das Geheimnisprinzip/Kapselung der OOP, das Aufteilen oder Zusammenfassen von Software-Modulen oder das Umbenennungen von Variablen, Methoden und Klassen.

4.2 Reengineering vs. Wrapping

Das Wrapping bezeichnet eine Form der Integration, bei der im Gegensatz zu einem "sauberen" EAI- bzw. SOA-Modell nur eine weitere Software-Schicht entwickelt wird, die nach außen hin dem EAI- bzw SOA-Prinzips folgt. Diese Schicht übernimmt dabei die Kommunikation zwischen den einzelnen Anwendungen und dem Fronten, während bei einem EAI-/SOA-Ansatz die Software-Komponenten ihre Daten untereinander und ohne Hilfe einer weiteren (Abstraktions-)Schicht auskommen. Bei der Umsetzung dieses Wrappings wird theoretisch kaum Code von exisitierenden Anwendungen verändert, sofern eine (beliebige) Schnittstelle vorhanden ist. Im besten Falle ließe sich das Wrapping als eine oberflächliche Ausprägung von Software-Integration bezeichnen, ein Reengineering und damit ein Überarbeiten von bestehenden Systemen stellt es jedoch nicht dar.

5 Werkzeuge im Reengineering

Dieses Kapitel stellt einzelne Werkzeuge vor, mit denen sich das Reengineering von Software sowohl hinsichtlich automatisierter Code-Erzeugung als auch mittels Code-Analyse leichter und computergestützt umsetzen lässt. Der Schwerpunkt liegt vor allem darin, die allgemeine Verwendbarkeit zu analysieren und herauszufinden, in welchem Umfang Software dazu in der Lage ist, semantische Zusammenhänge über Syntaxgrenzen hinweg erkennen und übertragen zu können. Eine Suche nach OpenSource Software war nicht ergiebig. Hier findet man vor allem Werkzeuge zur Unterstützung von Reverse-Engineering.

5.1 Sotograph

Der Sotograph (Software-Tomograph) der Firma hello2morrow ist ein Werkzeug zur Analyse und Visualiserung von Code-Struktur, -Qualität und -Abhängigkeiten. Er lässt sich dazu verwenden, Qualitätsaspekte im Laufe der Entwicklung von Software von Version zu Version zu bewerten und anzuzeigen [H2m11]. Damit lassen sich die Fortschritte eines Reengineering-Prozesses gut beobachten und nachvollziehen. Auch lassen sich Qualitäts-Defizite kurz nach Erscheinen beheben.

Vom Sotographen bewertete Qualitätskriterien sind beispielsweise Code-Duplizierung und Verstöße gegen Coding-Conventions. Auch Punkte wie Klassen-Kopplung

können analysiert und dargestellt werden. Vor allem bei der Sofware-Sanierung ist er ein vielversprechendes Werkzeug, um das Reengineering ständig überwachen, bewerten und korrigieren zu können. Der Sotogpraph unterstützt verschiedene Abstraktionslevel bei der Darstellung, sodass konkrete Kriterien für Entwickler genau so wie eine Übersicht z.B. für Management-Mitglieder visualisieren lassen.

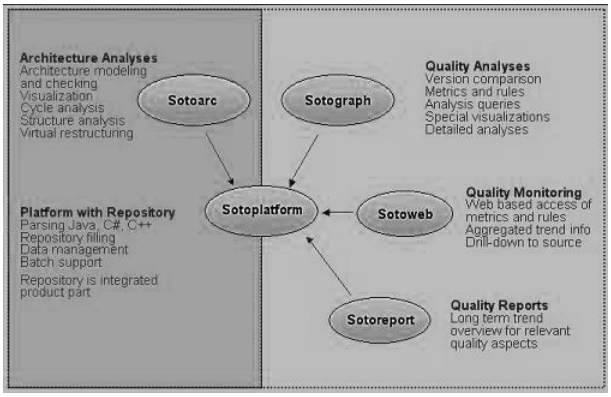

Abb. 2. Übersicht Soto-Toolchain

5.2 CoJaC

Die Firma *pro et con* stellt die Toolchain *COBOL to Java Converter* (CoJaC) bereit. Mit diesen Werkzeugen lässt sich eine COBOL-Legacy-Anwendung in eine funktional gleichwertige Java-Anwendung konvertieren. Von der Funktionsweise arbeitet CoJac nach dem klassischen Conpilermodell bestehend aus Scanner, Parser und Syntaxbaum-Bearbeitung. Der grobe Ablauf der Transformation ist in folgende Schritte aufgeteilt:

- **Präprozessor** Vorverarbeitung, Cobol-Module werden zusammengefügt.
- **Scanner** Erstellung einer Cobol-Tokenliste
- **Parser** Erstellen eines attributierten Cobol-Syntaxbaums
- **Transformator** Übersetzen in attributierten Java-Syntaxbaum
- **Postprozessor** Zerlegen in mehrere Teilbäume, Übernahme von Kommentaren
- **Generator** Erstellung von Java-Codedateien

Der entstandene Code verwendet intensiv eine von pro et con entwickelte Klassenbibliothek, in der vor allem Datentypen gekapselt werden. Auch die Ausführungslogik wurde in diesen Bibliotheken implementiert. Eine strikte Objektorientierung wurde hingegeben nicht umgesetzt. Laut pro et con sei dieser Ansatz nahezu unmöglich und wenig wünschenswert, da ehemalige Cobol-Programmierer unter Umständen auch den Java-Code weiterpflegen sollen.[Erd11]

Im Rahmen dieser Seminararbeit wurde eine Testkonvertierung eines Cobol-Batchprogramms beantragt und auch durchgeführt. Die resultieren Java-Dateien entsprechen Angaben von pro et con, ein Blick auf die Klassenbibliotheken war jedoch nicht möglich.

6 Ausblick

Mit seiner Jahrzehnte langen Geschichte hat das Software-Reengineering in seinen verschiedensten Ausprägungen immer wieder gezeigt, dass es durchaus Sinn ergibt, alte Software gründlich zu überarbeiten, auf neue Plattformen zu übertragen und den Einsatz von modernen Technologien und Sprachen in Erwägung zu ziehen. Durch die kostante Zusammenarbeit zwischen Industrie und Wissenschaft werden auf diesem Gebiet ständig neue Erkenntnisse gewonnen, Werkzeuge entwickelt und Vorgehensweisen entdeckt, mit denen sich Software in einen wirtschaftlich guten Zustand bringen lässt, die sowohl die Arbeit der Entwickler vereinfacht als auch die Zufriedenheit der Kunden sicherstellen kann.
Auch die Fortschritte im Bereich der künstlichen Intelligenz können Einfluss auf die Methoden des Reengineerings haben. Wenn Software in der Lage ist, nicht nur die Syntax sondern auch Semantik und damit die (fachlichen) Anforderungen an ein System analysieren kann, bietet das noch mehr Anreize, Legacy-Software oder andere, unwartbare Software zu sanieren, anstatt weiter Geld und Arbeitskraft in ein ständig teurer werdendes Programm zu investieren.

Literaturverzeichnis

[Ecl11] Eclipse Rich Client Platform. http://wiki.eclipse.org/index.php/Rich_Client_Platform Letzter Zugriff: 05.12.2011

[Erd11] Erdmenger, U: Ein Translator für die COBOL-Java-Migration. In: GI-Softwaretechnik-Trends, Band 31, Heft 2. ISSN: 0720-8928

[Erl08] Erl, T: SOA - Entwurfsprinzipien für serviceorientierte Architektur, Addison-Wesley, München, Deutschland, 2008. ISBN: 978-3-8273-2651-5

[Kel02] Keller, W: Enterprise Application Integration - Erfahrungen aus der Praxis, dpunkt.verlag, Deutschland, 2002. ISBN: 3-89864-186-4

[Pro11] pro et con - Innovative Informatikanwendungen GmbH http://www.proetcon.de/de/migCOBOL_d.html Letzter Zugriff: 05.12.2011

[Ran69] Randell, B; Naur, P: Software Engineering: Report of a conference sponsored by the NATO Science Comittee, Garmisch, Germany, 7-11 Oct. 1968. Scientific Affairs Division, Brüssel 1969.

[H2m11] Hello2morrow - Sotograph http://www.hello2morrow.com/products/sotograph Letzer Zugriff: 06.12.11