Software-Reengineering

Analyse, Restrukturierung und Reverse-Engineering von Anwendungssystemen

Von
Dr. Achim H. Kaufmann
Professor für Wirtschaftsinformatik

R. Oldenbourg Verlag München Wien

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Kaufmann, Achim H.:
Software-Reengineering : Analyse, Restrukturierung und
Reverse-Engineering von Anwendungssystemen / von Achim H.
Kaufmann. – München ; Wien : Oldenbourg, 1994
ISBN 3-486-23073-5

Gesamtherstellung: R. Oldenbourg Graphische Betriebe GmbH, München

ISBN 3-486-23073-5

Inhaltsverzeichnis

Vorwort

Wird man zum ersten Male mit einer konkreten Reengineering-Aufgabe konfrontiert, z.B. mit der Umstellung eines Programmsystems von FORTRAN IV nach FORTRAN 77 oder einer Großrechner- zu einer PC-Umgebung, so besteht die große Versuchung, sich dieser Problemstellung intuitiv zu nähern. Dies bedeutet für die genannte Programmumstellung das Übersetzen des Altsystems mit dem in der Zielumgebung verfügbaren Compiler und das anschließende sukzessive Beseitigen der formalen und logischen Fehler. Bei meinen ersten Reengineering-Projekten zeigte sich jedoch, daß diese Vorgehensweise aufwendig, demotivierend, fehleranfällig und unkalkulierbar ist. Als Konsequenz wurde sehr schnell offensichtlich, daß andere Lösungswege erforderlich sind. Bei der Suche nach besseren Vorgehensweisen erwiesen sich die wenigen in der Literatur angebotenen Ansätze für den unmittelbaren praktischen Einsatz als auch nicht besonders hilfreich.

Davon ausgehend war es das Ziel dieser Arbeit, eine systematische und kalkulierbare Erschließung und Nutzung des in vorhandenen Software-Systemen existierenden Wissens zu ermöglichen. Dieses Ziel wurde durch die Entwicklung einer allgemein einsetzbaren Vorgehensweise für das Software-Reengineering erreicht, die schrittweise bis auf die Ebene konkreter Vorhaben verfeinert werden kann.

Mit dieser Arbeit werden nicht nur Studenten der Informatik respektive Wirtschaftsinformatik, sondern auch Praktiker angesprochen, die sich mit Wartung, Migration und Standardisierung im Software-Bereich beschäftigen. Beiden Zielgruppen wird ein modernes methodisches Instrumentarium angeboten, das dazu beitragen soll, das negativ belegte Image des Wartungsbereichs wesentlich zu verbessern, zukünftigen Studentengenerationen ein attraktives Arbeitsfeld zu eröffnen und den heute häufig anzutreffenden "Frust" bei den mit diesen Aufgaben Beschäftigten abzubauen.

Achim H. Kaufmann

Einführung

Die Lösung eines seit langem bekannten jedoch nur unzureichend beachteten Problems der betrieblichen Informationsverarbeitung wird zunehmend dringlicher. Es geht um die Lösung des Zielkonfliktes, bei konstanten oder nur langsam wachsenden Entwicklungskapazitäten die hohe und noch steigende Nachfrage nach neuen Informationssystemen[1] bedarfsgerecht zu befriedigen und gleichzeitig eine ausreichende Wartung der eingesetzten Anwendungssysteme sicherzustellen. Zur Deckung der Nachfrage nach neuen Systemen existieren eine Reihe von ausgiebig diskutierten Konzepten, deren Umsetzung jetzt in der betrieblichen Praxis mit Nachdruck betrieben wird (4GL, CASE, RAD, OOP, IDV). Im Gegensatz dazu stellt sich die Wartungsproblematik wesentlich schwieriger dar. Die rasant fortschreitende technologische Entwicklung im Hard- und Software-Bereich führt zu einer ebenso schnellen technischen Alterung bestehender Systeme. Jedes zusätzliche Software-System, das entwickelt und in Betrieb genommen wird, vergrößert den "Berg" an Altsystemen. Es fehlen vielfach die Ressourcen, notwendige Anpassungen von Altsystemen vorzunehmen und gleichzeitig neue Systeme zu entwickeln. Es ist daher nicht verwunderlich, daß heute Wartung, Migration, Integration und Ersetzung bestehender Informationssysteme bei vielen Anwendern immer mehr an Ressourcen verschlingen. Werden keine wirksamen Maßnahmen ergriffen, diese Entwicklung zu stoppen, so steht bereits in naher Zukunft für viele Anwender die Entscheidung an, entweder keine Neuentwicklungen mehr vorzunehmen oder die Wartung für große Teile der Altsysteme einzustellen.[2]

Lösungen für diese "Wartungsmisere" werden dem ratsuchenden Anwender unter dem Modewort Software-Reengineering in aktuellen Beiträgen der Fachpresse und in der Angebotspalette von Unternehmensberatern offeriert. Ursprünglich wurde Reengineering zum einen als Teilaspekt von Software-Wiederverwendbarkeit in Form der Wiederverwendung von Design-Komponenten (reuse of design), verbunden mit der Offenlegung dieser Strukturen (design recovery), und zum anderen als Teilaspekt von Software-Wartung in Form eines Hilfsmittels zur Programm-Nachdokumentation und

---

[1] Die Wachstumsrate in den USA wird auf über 12% pro Jahr geschätzt [Business Week/88].

[2] Auch der Einsatz von Standardanwendungssoftware hat nur eine begrenzte Entlastung bewirkt, da hier durch Versionspflege, Integration und Betreuung ebenfalls langfristig Entwicklungsressourcen gebunden werden.

-Restrukturierung eingeordnet. Neuere definitorische Abgrenzungen lassen jedoch deutlich erkennen, daß Reengineering auf dem besten Wege ist, als Sammelbegriff für Software-Wartung [McClure/92, S. 23 f.] und zunehmend auch für Software-Wiederverwendbarkeit [Richter/92, S. 128 f.] etabliert zu werden.

Die unterschiedlichen Methoden und Werkzeuge des Reengineering können ergebnisorientiert in die Teilgebiete Programmanalyse[3], Programmrestrukturierung[4] und Programm-Reverse-Engineering[5] eingeteilt werden, wobei ersichtlich wird, daß die meisten Reengineering-Maßnahmen unmittelbar auf Quellprogrammen als Ausgangsbasis aufsetzen und diese in eine neue oder modifizierte Systemrepräsentation überführen. Andere Informationen, die über ein Software-System vorhanden sind, z.B. Entwicklungsdokumente, Benutzerhandbücher und Organisationsunterlagen, sind i.allg. weder Gegenstand von Reengineering-Vorhaben noch ist eine explizite methodische Integration dieser Systemdarstellungen als zusätzliche Informationsquelle ersichtlich. Darüber hinaus sind die dargestellten Reengineering-Ansätze mit unterschiedlichsten Methoden formuliert und auch realisiert, so daß man sich des Eindrucks eines begrifflichen und methodischen Sammelsuriums nicht entziehen kann.

Ausgehend von der geschilderten Problemlage und dem unzureichenden Stand der Problemlösung war es das Ziel, eine systematische und kalkulierbare Erschließung und Nutzung des in vorhandenen Software-Systemen existierenden Wissens zu ermöglichen. Dies spiegelt sich auch in der hier zugrundeliegenden allgemeinen Definition von Software-Reengineering wider: Software-Reengineering ist die Nutzung bestehender Altsysteme als Informationsquelle zur Wartung und Neuentwicklung von Anwendungen. Da weder die Struktur des Ausgangsmaterials "Software" noch die Art und Weise der Nutzung explizit festgelegt oder eingeschränkt wurde, mußte zur Erreichung des Ziels eine weitgehende, verallgemeinerte methodische Durchdringung dieses Gebietes vorge-

---

[3] Untersuchung existierender Programmsysteme, um qualitative und quantitative Aussagen über deren Komponenten und Strukturen, Arbeitsweise und Wartbarkeit treffen zu können.

[4] Transformation der Software-Repräsentationsform, z.B. Datennamen, Definitionen und Quellkode, ohne Veränderung der Funktionalität, um die Programmqualität zu erhöhen oder Programme an andere oft neue Basistechnologien anzupassen.

[5] Transformation der Software-Repräsentationsform von der Ebene des Programmkodes auf eine höhere Abstraktionsebene - meist die Designebene -, um Programmsysteme nachzudokumentieren und damit die Verständlichkeit zu erhöhen.

nommen werden. Zusätzlich sollten bei der Festlegung eines allgemeinen methodischen Ansatzes folgende wichtige Rahmenbedingungen beachtet werden:

- Bei alternativen Methoden sind diejenigen mit dem höchsten praktischen Bekanntheits- und Verbreitungsgrad zu verwenden, um die Akzeptanz zu erhöhen und den Lernaufwand zu reduzieren.
- Es müssen Formalismen enthalten sein, die eine weitgehende Integration des bisher vorhandenen methodischen Wissens aus diesem Bereich ermöglichen, so daß die Vielfalt der Spezifikationsverfahren und damit ebenfalls der Lernaufwand eingeschränkt werden können.
- Die ausgewählten Methoden sollen eine inkrementelle und prototypische Vorgehensweise ermöglichen, so daß im konkreten Fall zwischen Reengineering und manueller Neuerhebung ein nach Wirtschaftlichkeitskriterien optimales Vorgehen gewählt werden kann.
- Der Ansatz muß ein hohes und leicht zu erschließendes Automatisierungspotential aufweisen, so daß die Erstellung und der Einsatz von Werkzeugen möglichst effizient erfolgen können; dies schließt auch die weitgehende Verwendung bereits vorhandener Werkzeuge ein.

Die Umsetzung dieses Anforderungskatalogs erscheint auf den ersten Blick sehr schwierig oder sogar unmöglich, jedoch können Parallelen zur Problemstellung bei der Entwicklung fachlicher Systeme (z.B. von betrieblichen Abrechnungs- und Informationssystemen) gezogen werden. Auch hier liegen unterschiedlich strukturierte fachliche Informationen vor, die nach bestimmten Regeln umgeformt und damit genutzt werden. Die Methoden zur allgemeinen Erhebung und Beschreibung fachlicher Zusammenhänge werden unter dem Begriff "Systemanalyse" zusammengefaßt; es bot sich an, die Eignung systemanalytischer Methoden auch für die vorliegende Problemstellung zu überprüfen. Dabei stellte sich heraus, daß im Gegensatz zur Analyse allgemeiner fachlicher Zusammenhänge hier bereits eine reduzierte und damit vereinfachte Grundstruktur vorlag: Reengineering-Maßnahmen transformieren Informationsbestände nach dem klassischen EVA-Prinzip der elektronischen Datenverarbeitung; die Datenbestände besitzen im Regelfall eine formalisierte bzw. formalisierbare Form und liegen in vielen Fällen bereits in elektronischer Speicherform vor oder sind verhältnismäßig leicht elektronisch zu erfassen. Damit sind auch besonders gute Voraussetzungen für einen weitgehenden und wirtschaftlichen Werkzeugeinsatz gegeben. Schwierigkeiten entstehen durch die höhere Abstraktionsebene, auf der man die Erhebung und Spezifikation der Informationsbestände und Transformationsprozesse durchführen muß. Werden bei der Systemanalyse konkrete fachliche Zusammenhänge erfaßt und beschrieben, so

sind hier unterschiedliche Modelle und Methoden durch eine Metamethode (Metamodellierung) abzubilden.

Auf der Grundlage dieser Überlegungen wurde eine allgemein einsetzbare Vorgehensweise für das Software-Reengineering entwickelt, bei der Ziel- und Ausgangssystem einheitlich mit Hilfe eines erweiterten Entity-Relationship-Ansatzes (ER-Ansatz) und die Transformationsprozesse mit Hilfe der Datenflußanalyse modelliert werden. Dadurch ist es möglich, nicht nur Software-Systeme im engeren Sinne (Programme), sondern alle Informationen, die für Anwendungssysteme verfügbar sind, in den Prozeß zu integrieren. Die Vorgehensweise wird exemplarisch auf verschiedene konkrete Reengineering-Vorhaben übertragen und führt zu speziellen, unmittelbar praktisch anwendbaren Vorgehensmodellen. Im einzelnen werden Vorgehensmodelle für die Analyse von Programmsystemen im Hinblick auf ihre Wartbarkeit, die Überführung von unstrukturierten Programmen in strukturierte Programme, die Restrukturierung von Datenbeständen sowie die Wiedergewinnung fachlicher Modelle aus Programmquellen und -dokumentationen dargestellt.

Im Hinblick auf die oben aufgestellten Anforderungen kommt man zu folgenden Resultaten: Durch die Verwendung von Metamodellen ist eine weitgehende Verallgemeinerung und damit breite Anwendbarkeit dieser Methode auf spezielle Reengineering-Vorhaben gewährleistet. Die verwendeten Basismethoden (ER- und Datenfluß-Modellierung) sind weit verbreitet und allgemein akzeptiert. Durch Anpassung der Spezifikationsform können bereits vorhandene Reengineering-Methoden gemäß dem dargestellten Ansatz vereinheitlicht und integriert werden. Die Vorgehensweise ist schrittweise entwickelbar sowie einsetzbar und enthält keine Vorgaben für die Realisierungsform (z.B. manuell oder automatisiert). Aufgrund der einheitlichen Modellierung ist ein hoher Standardisierungsgrad vorhanden, der die Entwicklung von Basiswerkzeugen mit breiter Einsatzfähigkeit fördert und leicht an spezielle Gegebenheiten anpaßbar ist. Unterstützt wird diese Möglichkeit durch das große Angebot an modernen Software-Entwicklungssystemen (z.B. CASE), die auf der Basis von ER-Modellen arbeiten und somit entweder die Entwicklung von Transformationsprogrammen wesentlich erleichtern oder sogar eine Konfigurierung mit Hilfe frei definierbarer Metamodelle anbieten. Dies entspricht der methodischen Grundlage der dargestellten Vorgehensweise zum Reengineering, so daß eine Integration von Computer-Aided-Software-Engineering (CASE) und Computer-Aided-Software-Reengineering (CARE) im gleichen Paradigma erfolgen kann [Martin/91].

Da große Bestände an Altsystemen in der Programmiersprache FORTRAN implementiert sind, wird im weiteren dargestellt, wie die speziellen Vorgehensmodelle auf entsprechende Anwendungssysteme übertragen werden können. Ausgangsbasis sind Analyse und Beschreibung der historisch gewachsenen, strukturellen und stilistischen Besonderheiten der Sprache im Hinblick auf Reengineering-Maßnahmen. Die daraus resultierenden Darstellungen sind ebenfalls in Form erweiterter Entity-Relationship-Modelle spezifiziert, so daß eine unmittelbare, methodisch einheitliche Einbettung der speziellen Vorgehensmodelle vorgenommen werden konnte.

Ist eine Altanwendung zu ersetzen oder in ihrer Darstellungsform zu ändern, muß eine geeignete Vorgehensweise zwischen "Neuentwicklung ohne Berücksichtigung des Altsystems" und "Reengineering des Altsystems" ausgewählt werden. Eine unter Wirtschaftlichkeitsaspekten zu treffende Wahl einer geeigneten Vorgehensweise muß sich am Aufwand für eine Neuentwicklung als Obergrenze orientieren, mit der Folge, daß nur dann Reengineering-Maßnahmen eingesetzt werden dürfen, wenn ihr Aufwand geringer ist. Ob Entwicklungstätigkeiten manuell durchgeführt oder durch automatisierte Systeme unterstützt werden sollen, ist eine weitere Entscheidung, die ebenfalls nur nach Wirtschaftlichkeitskriterien getroffen werden darf. Dies bedeutet grundsätzlich ein Abwägen zwischen dem Aufwand zur Programmerstellung und den Einsparungen, die bei der DV-Unterstützung gegenüber einer manuellen Vorgehensweise erreicht werden können. Zur Erhöhung der Wirtschaftlichkeit bietet es sich sogar an, daß nicht jeder Anwender eine Individuallösung anstrebt, sondern daß Standardisierungspotentiale in Form verallgemeinerter Reengineering-Ansätze genutzt werden.

Ausgehend von dem Umstand, daß ca. 70% der Entwicklungskapazitäten für Wartung, Migration und Ersetzung von bestehenden Anwendungssystemen eingesetzt werden müssen, liegt eine zentrale Erkenntnis dieser Arbeit in der Notwendigkeit, diesen Bereich zukünftig in der Ausbildung angemessen zu berücksichtigen. Das z.Zt. beobachtbare Defizit könnte durch ein qualifiziertes Angebot von Veranstaltungen zum Thema Reengineering behoben werden, so daß Erfahrungssicherung und ein wirtschaftlich fundierter Investitionsschutz nicht nur leere Worthülsen im Informatikbereich bleiben. Reengineering ist darüber hinaus keine einmalige Angelegenheit, sondern alles, was heute produziert wird, ist bereits morgen als Altsystem anzusehen. Reengineering kann vermeiden, daß aus Altsystemen Altlasten werden.

1. Kapitel

Grundlagen des Software-Reengineering

Der Begriff Software-Reengineering entstand als Antwort auf das Anfang der siebziger Jahre bekannte und sich ständig verschärfende Problem der Wartung des stetig wachsenden "Bergs" an vorhandenen und eingesetzten Anwendungssystemen. Ähnlich wie bei anderen populären Schlagwörtern aus dem Software-Entwicklungsbereich, z.B. Computer-Aided-Software-Engineering (CASE), Rapid-Application-Development (RAD) und Object-Oriented-Programming (OOP), handelte es sich anfangs um einen sehr diffusen Begriff, in dem sich Hoffnungen und Wünsche der DV-Anwender und potentielle Marktchancen für Anbieter aus diesem Bereich trafen.

Ganz allgemein bedeutet Software-Reengineering die Nutzung bestehender Altsysteme (Programmkode und Dokumentation) als Informationsquelle zur Wartung und Neuentwicklung von Anwendungen. Im einfachsten Fall kann es sich dabei um die Nachdokumentation von Programmsystemen und im schwierigsten Fall um die Extraktion fachlicher Zusammenhänge aus Programmen und Dokumentationen handeln [Bischoff/92, S. 125].

1.1 Klassen von Methoden

Ebenso wie beim Software-Engineering kann bei der grundsätzlichen methodischen Vorgehensweise zwischen der Experimentier- und Modellmethode unterschieden werden. Experimentieren bedeutet in diesem Zusammenhang die unmittelbare Veränderung im Einsatz befindlicher Anwendungssysteme, was meist mit "trial and error" gleichzusetzen ist. Diese Methodik führt jedoch bei komplexen Systemen nur zu begrenzten systematischen Erkenntnissen und Vorgehensweisen; darüber hinaus sind die Zielsysteme von Reengineering-Maßnahmen vielfach ihrerseits Modelle (z.B. Fachmodell, Benutzermodell), so daß im weiteren die Modellmethode zugrundegelegt wird.

1.2 Modellmethode der Systementwicklung

Voraussetzung für die folgenden Ausführungen ist daher die explizite Festlegung des Software-Entwicklungsprozesses über verschiedene Abstraktionsebenen[1] hinweg, die im Hinblick auf ihre Ergebnisse sowohl semantisch als auch methodisch eindeutig definiert sind. Im allgemeinen werden diese Ergebnisse bestimmten Phasen eines Software-Life-Cycle zugeordnet, so daß sich die begriffliche Unterscheidung in Tätigkeiten, Ergebnisse und Abstraktionsebenen verwischt. Beim Entwurf eines entsprechenden Reengineering-Modells bietet es sich an, auf Vorgehensmodelle des Software-Engineering zu referenzieren, da zum einen die ursprüngliche Systementwicklung in dieser Art und Weise durchgeführt wurde (bzw. hätte durchgeführt werden können) und zum anderen die einzelnen Zwischenergebnisse dieses Entwicklungsprozesses auch Ziel eines Reengineering-Vorhabens sein können. Im folgenden wird das in Abb. 1.1 dargestellte Vorgehensmodell für den Software-Entwicklungsprozeß zugrundegelegt.[2] Es besteht aus den Phasen Anforderungsanalyse, Fachentwurf, DV-Entwurf und Implementierung. Innerhalb der Anforderungsanalyse wird ein Modell der fachlichen Komponenten und Zusammenhänge entworfen, während nach der Implementierung das konkrete Anwendungssystem mit seinen statischen (Programmkode, Dokumentation, Dateibeschreibungen usw.) und dynamischen (Datenbestände, Ablaufverhalten, Benutzer-, Programminteraktion) Strukturen vorliegt.

---

[1] Der Begriff "Abstraktionsebenen" soll die zunehmende Spezialisierung der Zwischenergebnisse im Hinblick auf die Entwicklung eines konkreten DV-Systems ausdrücken.

[2] Dieses Phasenmodell beeinflußt mit seinen Abstraktionsebenen die gesamte Formulierung des folgenden Reengineering-Prozesses; die Allgemeingültigkeit wird jedoch nicht eingeschränkt, da in analoger Weise jedes andere Phasenmodell verwendet werden könnte.

Phase (Tätigkeit)	Ergebnis	Abstraktionsebene
Anforderungsanalyse	Beschreibung der organisatorischen (fachlichen) Komponenten und Zusammenhänge, losgelöst von Implementierungsaspekten	Organisationsmodell
Fachentwurf	Beschreibung der zu automatisierenden fachlichen Komponenten und Zusammenhänge sowie deren Interaktionen mit der Außenwelt (Benutzerschnittstellen usw.)	Anwendungsmodell
DV-Entwurf	Beschreibung der programmtechnischen Komponenten und Zusammenhänge (Daten-, Programmstrukturen usw.)	Implementierungsmodell
Implementierung	Realisierung der Beschreibungen des DV-Entwurfs in Form von Programmen, Datenbanken usw.	Anwendungssystem

Abb. 1.1: Vorgehensmodell zur Software-Entwicklung

1.3 Formen des Software-Reengineering

Da ein Anwendungssystem auf jeder der oben dargestellten Abstraktionsebenen eine spezifische Ausprägung besitzen kann, können Software-Reengineering-Maßnahmen grundsätzlich auf einer beliebigen Ebene aufsetzen, um eine alte (Ausgangssystem) in eine neue (Zielsystem) Systemrepräsentation derselben oder einer anderen Ebene zu überführen. Werden existierende Strukturen neuen Gegebenheiten oder Bedürfnissen angepaßt, wird also das Ausgangs- in ein Zielsystem derselben Ebene transformiert, spricht man von Restrukturierung. Befinden sich dagegen Ausgangs- und Zielsystem auf unterschiedlichen Ebenen, verwendet man den Begriff Redesign bzw. Reverse-Engineering [Lano/94, S. 5]. Tätigkeiten der Restrukturierung, des Redesigns und auch solche, die bei der Systemneuentwicklung anfallen, werden bei komplexeren Reengineering-Prozes-

sen kombiniert,[3] was dann zum Zusammenhang in Abb. 1.2 führt, bei der die Kreise die möglichen Repräsentationen eines Anwendungssystems darstellen.

Der Reengineering-Zyklus teilt sich in die zwei Hauptbereiche Forward-Engineering und Reverse-Engineering [Richter/92, S. 128 ff.]. Der Forward-Engineering-Bereich beinhaltet alle Arbeitsschritte, die auch bei einer Neuentwicklung anfallen, während Reverse-Engineering alle Aktivitäten umfaßt, die aus vorhandenen Strukturen den Informationsgehalt höherer Abstraktionsebenen extrahieren und in einer eigenen Repräsentation darstellen. Aus Abb. 1.2 wird ebenfalls deutlich, daß sich Reengineering-Prozesse über mehrere Abstraktionsebenen als Zwischenstufen erstrecken können. Eine Restrukturierung auf der Anwendungssystemebene muß z.B. nicht unmittelbar zu einer geänderten Version auf derselben Ebene führen; es kann auch zuerst eine Redesign-Maßnahme durchgeführt werden, um ein Implementierungsmodell zu erhalten, das dann in einem zweiten Schritt zu einem restrukturierten Anwendungssystem implementiert wird. Ebenso kann ein Redesign-Prozeß erst eine Restrukturierung des Ausgangssystems voraussetzen.

Innerhalb dieser Arbeit wird Reengineering als Oberbegriff für die anderen dargestellten Formen verwendet. Dies widerspricht der Definition von Chikofsky [Chikofsky/90, S. 15 f.], bei der Reengineering als eigenständige Form beschrieben wird, die aus der Folge von Reverse-Engineering und Forward-Engineering bzw. Restrukturierung besteht und zur Implementierung eines neuen Software-Systems führt. Reengineering wird jedoch zunehmend - wie in dieser Arbeit - in der Praxis, aber auch in Veröffentlichungen [Bischoff/92, S. 125, McClure/92, S. 23 f., Richter/92, S. 128] als generalisierter Begriff verwendet.

---

[3] Welche konkrete Maßnahme oder welches Maßnahmenbündel angewendet wird, hängt ausschließlich von der Zielsetzung eines Reengineering-Prozesses ab. Dazu zählt i.allg. die Effizienzverbesserung bei Wartung (Fehlerbehebung, Funktionserweiterung, Systemintegration und Migration) sowie Neuentwicklung bestehender Anwendungssysteme. Darüber hinaus können jedoch auch noch weitere Ziele existieren, wie z.B. Reengineering zur Informationsgewinnung beim Aufbau von Unternehmensmodellen (unternehmensweites Daten- bzw. Funktionsmodell).

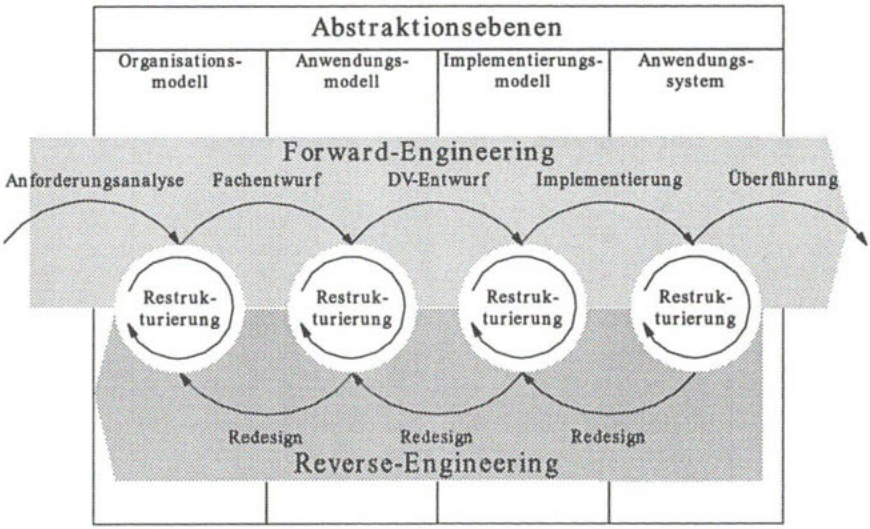

Abb. 1.2: Reengineering-Zyklus

1.4 Software-Engineering und Software-Reengineering

Aus der Darstellung des Reengineering-Zyklus' wird ersichtlich, daß Software-Engineering und -Reengineering auf derselben methodischen Basis definiert werden können. Beachtet man darüber hinaus auch, daß häufig das Wissen um fachliche Zusammenhänge durch den zunehmenden DV-Einsatz verlorengegangen ist[4] und daß Neuentwicklungen meistens nicht auf der "grünen Wiese" beginnen, sondern mit bestehenden Anwendungssystemen integriert werden müssen, so ergibt es sich zwangsläufig, daß Reengineering-Maßnahmen bereits heute Bestandteil eines "normalen" Software-Entwicklungsprozesses sind.

Wäre es nicht konsequent, Reengineering weder als Voraussetzung noch als Ergänzung, sondern als festen Bestandteil von Software-Engineering zu betrachten? Um diese Frage zu beantworten, ist eine Rückbesinnung auf die grund-

[4] Bei hoher DV-Durchdringung sind mit zunehmender Tendenz dem einzelnen Sachbearbeiter die fachlichen Zusammenhänge nicht mehr geläufig und transparent, da immer mehr an Funktionalität durch die eingesetzten DV-Systeme übernommen und damit verdeckt wird. Somit ist es auch bei einer Neuentwicklung nicht mehr möglich, die fachlichen Strukturen in Form der "klassischen" Systemanalyse, d.h. als Kommunikationsprozeß zwischen Fachbereich und Systementwicklung, zu erheben.

legenden Intentionen erforderlich, die mit dem Begriff Software-Engineering verbunden sind. Entwicklungsprinzipien, Vorgehensmodelle und Beschreibungsmethoden - so massiv sie auch in den Vordergrund des Interesses gerückt sind - dürfen nicht zum Selbstzweck werden, sondern stellen Mittel dar, um Software-Entwicklung planbar und damit kalkulierbar zu machen. Fordert man darüber hinaus, daß Software als Investitionsgut zu betrachten ist, so kommt man nicht umhin, die Wirtschaftlichkeit als wichtigsten Maßstab bei der Entscheidung für oder gegen eine Entwicklung zu akzeptieren. Wichtigstes Ziel des Software-Engineering im betrieblichen Umfeld ist somit, die Wirtschaftlichkeit von Software-Investitionen zu gewährleisten. Setzt man dieses Ziel konsequent auf die einzusetzenden Methoden (Mittel) um, so müssen die Methoden zur Anwendung kommen, mit denen ein bestimmtes Ergebnis möglichst effizient erreicht wird.

Reengineering stellt dann - ebenso wie Neuaufnahme - eine Methode dar, um bei der Software-Entwicklung Informationsquellen zu erschließen und den Informationsbedarf zu decken. Beide sind in Teilbereichen substituierbar, sie können sich aber auch ergänzen. Liegt eine Austauschbarkeit vor, so muß der jeweilige Aufwand zur Deckung des Informationsbedarfs als Entscheidungsgrundlage für die eine oder andere Methode herangezogen werden. Damit wird deutlich, daß Reengineering eine den anderen Methoden des Software-Engineering gleichberechtigte Maßnahme ist. Reengineering bildet folglich einen festen Bestandteil von Software-Engineering.

1.5 Automatisierung des Software-Reengineering

Unter dem Begriff Computer-Aided-Software-Engineering" (CASE) sind in den letzten Jahren eine Vielzahl von DV-gestützten Werkzeugen zur Anwendungssystementwicklung auf den Markt gekommen. Die meisten CASE-Produkte unterstützen die "traditionellen" Methoden der Software-Entwicklung (SA/SD, ERM, HIPO) in einer sehr weitgehenden und auch aufeinander abgestimmten Art und Weise (ICASE), so daß von dieser Seite einem effizienten praktischen Einsatz dieser Systeme nichts mehr im Wege steht. Die Software-Entwicklung durch die z.Zt. wichtigsten CASE-Systeme (AD/Cycle, PREDICT CASE, ADW, EXCELERATOR usw.) erfolgt ebenfalls mit Hilfe verschiedener Abstraktionsebenen, d.h., ausgehend von der Erhebung und Modellierung fachlicher (organisatorischer) Zusammenhänge, wird über mehrere Abbildungsstufen hinweg ein Programmsystem mehr oder weniger automatisiert realisiert. Neben dem originären Ziel der automatisierten Software-Entwicklung werden CASE-Systeme zunehmend zum Aufbau und zur Pflege von Unternehmensmodellen

verwendet, welche die Voraussetzung für eine effektive Nutzung und Integration betrieblicher Informationsbestände bilden und denen darüber hinaus im gesamten organisatorischen Bereich eine zentrale Bedeutung zukommen wird.

Mit dem Begriff Computer-Aided-Software-Reengineering" (CARE) wird der Eindruck erweckt, daß der allgemeine Entwicklungsstand der verfügbaren DV-gestützten Werkzeuge für Reengineering mit dem des CASE-Bereichs vergleichbar ist. Das trifft jedoch nicht zu. Die DV-Unterstützung beim Reengineering beschränkt sich z.Zt. im wesentlichen auf folgende Maßnahmen [Wagner/91; Sneed/91, S. 2.1-2]:

- Sanierung von Programmkode: Dazu zählt die Ersetzung unstrukturierter Kontrollanweisungen durch Konstrukte der Strukturierten Programmierung, Beseitigung gemeinsamer Speicherbereiche (COMMON-Blöcke) durch explizite Parameterübergabe, Entfernung "toter" Kodestrecken usw.
- Nachdokumentation und graphische Aufbereitung von Programmen: Darstellung der Programmstrukturen mit Hilfe von Nassi-Shneiderman-, Datenfluß-, Datenstruktur-, Programmablauf- und Programmaufruf-Diagrammen, Erzeugung von Cross-Reference-Tabellen, tabellarischen Datenbeschreibungen, Gesamtbeschreibungen usw.
- Transformation in eine andere Programmiersprache: Umsetzung von FORTRAN in C, COBOL in eine 4GL-Sprache usw.

Angeboten werden diese Werkzeuge noch vielfach als Insellösungen, jedoch sind die CASE-Hersteller dabei, ihre Produkte um entsprechende Funktionalitäten zu ergänzen.[5] Über die dargestellten Maßnahmen hinaus, z.B. in Richtung Reverse-Engineering, existieren nur noch vereinzelte - oft sehr allgemein gehaltende - Lösungsansätze, die nur in seltenen Fällen durch Werkzeuge unterstützt werden [Boetticher/91; Dörfel/91, S. 6.1-4].

Allgemein kann man feststellen, daß für die vorhandenen Reengineering-Ansätze bisher keine allgemein akzeptierte methodische Grundlage existiert, sondern daß es sich bei den einzelnen Maßnahmen um jeweils spezielle Lösungstechniken handelt. Aus dieser unzureichenden methodischen Durchdringung der Reengineering-Problematik erklärt sich auch die im Verhältnis zu anderen

---

[5] Der große Vorteil einer Integration von Reengineering-Funktionen in CASE-Systeme liegt in der Übernahme bestimmter Objekte (Daten, Masken, Funktionen) aus den Altsystemen in die zentralen CASE-Entwicklungsdatenbanken (repositories) und der damit verbundenen Möglichkeit zur Weiterbearbeitung.

Entwicklungsmethoden spärliche Rechnerunterstützung, die darüber hinaus vielfach nur Insellösungen darstellt. Damit bestätigt sich die Aussage, daß ein Werkzeug nie besser sein kann als die zugrundeliegende Methode.

2. Kapitel

Entwicklungsstand des Software-Reengineering

In der aktuellen Diskussion ist der Begriff des Software-Reengineering sehr eng mit zwei weiteren Begriffen, denen der Software-Wiederverwendbarkeit (software reusability) und Software-Wartung (software maintenance), verknüpft [Biggerstaff/89b; Richter/92; McClure/92]. In entsprechenden Veröffentlichungen sind oft begriffliche Abgrenzungen bzw. Zuordnungen entweder als eigener Publikationsgegenstand oder als Grundlage weitergehender Ausführungen enthalten; es fällt jedoch auf, daß die definitorischen Festlegungen von Autor zu Autor sehr unterschiedlich sein können und daß in der historischen Entwicklung eine breit akzeptierte Verschiebung bei den Abgrenzungen der Begriffe eingetreten ist, die sich vermutlich noch fortsetzen wird. Ursprünglich wurde Software-Reengineering zum einen als Teilaspekt von Software-Wiederverwendbarkeit in Form der Mehrfachnutzung von Design-Komponenten (reuse of design), verbunden mit der Offenlegung dieser Strukturen (design recovery), und zum anderen als Teilaspekt von Software-Wartung als Hilfsmittel zur Programm-Nachdokumentation und -restrukturierung eingeordnet. Neuere definitorische Abgrenzungen lassen deutlich erkennen, daß Reengineering auf dem besten Weg ist, als Sammelbegriff für Software-Wartung [McClure/92, S. 23 f.] und zunehmend auch für Software-Wiederverwendbarkeit [Richter/92, S. 128 f.] etabliert zu werden. Dies schließt auch Managementaspekte wie Konfigurations- und Änderungsmanagement (change/configuration management) und Anwendungssystemmanagement sowie Automatisierungsmöglichkeiten (CASE, repository) mit ein.[1]

Trotz dieser Tendenz, den Reengineering-Begriff in seiner Bedeutung immer mehr auszuweiten, tritt auch bei neueren Publikationen - sobald konkrete Problemstellungen und -lösungen behandelt werden - der originäre instrumentelle Charakter von Reengineering im Hinblick auf Software-Wartung wieder in den Vordergrund.

---

[1] Die zunehmende Beliebtheit des Begriffs Software-Reengineering ist auf die - wohl beabsichtigte - phonetische Verwandtschaft mit dem Begriff Software-Engineering zurückzuführen, der sich in der gesamten DV-Welt mit breit akzeptierten positiven Implikationen durchgesetzt hat. Die allgemeine Erwartung - insbesondere von Software-Anbietern - ist nun, daß sich dieses positive Image auf Reengineering überträgt und zu entsprechend guten Vermarktungschancen führt.

2.1 Software-Wartung und Software-Reengineering

Die in der Einleitung dargestellte Wartungsproblematik produktiv eingesetzter Programmsysteme ist heute wohl die bedeutendste Herausforderung des praktischen Informationsmanagements. Wird von den meisten Anwendern ein aktives Planen, Steuern und Kontrollieren des Software-Erstellungs- und -Einführungsprozesses mit Hilfe von Methoden und Werkzeugen des Software-Engineering anerkannt und aktiv angestrebt bzw. betrieben, so ist bei der zeitlich sich anschließenden Software-Wartung eher ein reaktives Verhalten statt eines aktiven Verhaltens zu beobachten.[2] Von der durch namhafte Experten aus Wissenschaft und Praxis aufgestellten Forderung nach einem aktiven Anwendungssystemmanagement über den gesamten Lebenszyklus eines Software-Produktes [Heinrich/88, S. 149 ff.] ist man heute in der DV-Realität noch weit entfernt. Aktives Handeln in den Phasen der Systemnutzung, d.h. bei Wartung und Ersetzung bzw. Aussonderung von Anwendungssystemen, ist jedoch die Voraussetzung, den sich ständig erhöhenden Bedarf an Wartungsressourcen zu Lasten der Neuentwicklung begrenzen zu können.

Maßgebliche Einflußgröße für den Wartungsaufwand ist neben der personellen, organisatorischen und technischen Wartungsumgebung die Wartbarkeitseignung der zu betreuenden Software-Systeme.[3] Wartbarkeit setzt sich zum einen aus den Wartungsanforderungen, d.h. Anzahl und zeitliche Verteilung, mit denen Wartungsfälle auftreten, und zum anderen aus der Wartungsfreundlichkeit des Software-Produktes zusammen. Der Eintritt eines Wartungsfalls ist abhängig von einer von außen determinierten Änderungsdynamik der Produktionsumgebung (Änderung fachlicher und technischer Anforderungen) und der dem Software-Produkt inhärenten Qualität hinsichtlich Fehlerfreiheit, funktionaler Vollständigkeit, Zuverlässigkeit, Robustheit und Flexibilität. Wartungsfreundlichkeit als zweite Komponente der Wartbarkeit ist die Eignung der Soft-

---

[2] Aktivitäten wie Fehlerbehebung, Anpassung an neue fachliche und technische Gegebenheiten oder das Ersetzen bzw. Aussondern von Anwendungssystemen werden von den DV-Zuständigen meistens erst dann in Angriff genommen, wenn konkrete, unausweichbare Anforderungen vorliegen.

[3] Wartung beschränkt sich im folgenden nicht nur auf Programmquellen, sondern umfaßt zusätzlich die Aktualisierung und Fortschreibung der gesamten Programmdokumentation, die zum Betrieb und für zukünftige Wartungsmaßnahmen notwendig sind [Sneed/91 S. 36], sowie auch die Änderungen angrenzender eigenständiger Software-Systeme und des organisatorischen Umfeldes, in das die Software eingebettet ist.

ware, aufgrund ihrer qualitativen und quantitativen Beschreibungsformen den zur Durchführung einer Wartungsmaßnahme[4] notwendigen Aufwand gering zu halten [Asam/86, S. 91]. Eine Wartungsmaßnahme läßt sich in folgende, ggf. auch iterativ durchführbare Arbeitsschritte aufteilen:

- Analysieren: Ausgehend von der Wartungsanforderung, müssen die betreffenden Stellen innerhalb des Software-Systems lokalisiert und die notwendigen Änderungen festgelegt werden. Dabei sind auch die Auswirkungen auf andere Systembestandteile[5] zu untersuchen und die notwendigen Folgeänderungen zu bestimmen; dieser Prozeß wird solange wiederholt, bis keine weiteren Seiteneffekte mehr vorhanden sind. Auf der Grundlage der notwendigen Wartungsarbeiten kann eine Abschätzung der weiteren Schritte vorgenommen und ins Verhältnis zum Nutzen der Wartungsanforderung gesetzt werden, so daß es möglich ist, über die Fortsetzung der Arbeiten nach wirtschaftlichen Kriterien zu entscheiden.
- Ändern: Sind die Änderungsmaßnahmen aufgrund des Analyseschrittes festgelegt, müssen sie konkret umgesetzt werden. Wesentlich dabei ist, daß die Änderungen fehlerfrei durchgeführt und auch keine Modifikationen von nicht zu ändernden Systemteilen mittelbar bewirkt werden. Um die zukünftige Wartbarkeit des Gesamtsystems nicht zu verschlechtern, ist ebenfalls sicherzustellen, daß durch die Wartungsaufgabe nicht die Systemarchitektur verzerrt, überlagert oder sogar zerstört wird[6] und daß die übrigen Software-Qualitätsmerkmale nicht negativ verändert werden. Bei der Durchführung der Änderungen wird man in vielen Fällen inkrementell vorgehen, d.h., eine unabhängige Teiländerung wird komplett realisiert und anschließend geprüft/getestet und ggf. auch bereits überführt, bevor mit der nächsten begonnen wird. Diese Vorgehensweise ist gemäß

---

[4] Wartungsmaßnahme wird hier als Sammelbegriff für Fehlerbehebung, Erweiterung, Anpassung und Optimierung verwendet.

[5] Hierzu zählen nicht nur Änderungen in anderen Quellprogrammteilen, sondern auch Auswirkungen auf die gesamte Systemdokumentation, andere in Verbindung stehende Software-Systeme, vorhandene Datenbestände und die organisatorische Umgebung (Benutzer, Arbeitsabläufe und Hilfsmittel).

[6] Bei Änderungsmaßnahmen kann es unter bestimmten Umständen sinnvoll sein, die Gesamtstruktur der Systemarchitektur zu ändern. Solange dies - ausreichend dokumentiert - komplett und konsistent erfolgt, ist es nicht zu verwerfen. Sobald jedoch bewußt oder auch unbewußt Bestandteile, die nach anderen Kriterien als dem Ausgangssystem strukturiert sind, in das System eingefügt werden, wird dieses zunehmend fragil und immer aufwendiger zu warten.

dem Lokalitätsprinzip des Software-Engineering grundsätzlich positiv einzustufen, jedoch darf sie nicht mit einem Trial-and-error-Vorgehen, bei dem die Folgewirkungen von Veränderungen nicht im voraus zu überblicken sind, verwechselt werden. Bei Änderung eines Produktionssystems sind auch die Auswirkungen auf bestehende Datenbestände und Schnittstellen zu anderen Systemen hin zu beachten.

- Prüfen/Testen: Prüfen beinhaltet die formale und logische Überprüfung der statischen Software-Bestandteile, während Testen das Verhalten beim Systemablauf zum Inhalt hat. Die Einhaltung der Qualitätsvorgaben und die vollständige Umsetzung der Änderungsaufgaben können in Form von automatischen Verfahren zur Qualitätssicherung und mit Hilfe von manuellen Verfahren wie "Reviews" und Inspektionen durchgeführt werden.[7] Das Testen der Änderungen besteht aus der Erstellung von Testdaten, der Testdurchführung und der Auswertung der Ergebnisse. Wird der Test nur auf die geänderten Systemteile bezogen, so ist der Aufwand verhältnismäßig gering. Dies birgt jedoch die Gefahr, daß nicht erkannte Fernwirkungen von Änderungen auch im Test unentdeckt bleiben. Besser wäre in diesem Fall ein Gesamttest wie bei einer Neuentwicklung, was oft nur wirtschaftlich zu vertreten ist, wenn man auf eine während der Systementwicklung vorgenommene Formalisierung und Automatisierung des Testablaufs aufsetzen kann.
- Überführen: Nachdem die Änderung durchgeführt und geprüft wurde, muß die Übertragung aus der Wartungs- in die Produktionsumgebung erfolgen. Dazu ist es notwendig, daß das ggf. noch in Produktion befindliche ungeänderte System entfernt und die Produktionsumgebung angepaßt wird, d.h. Anpassung der betroffenen Datenbestände, Schnittstellen zu anderen Systemen und Organisation des Anwendungsbereiches und des Rechenbetriebs. Zuvor sollte jedoch eine Sicherungs- und Katastrophenplanung vorgenommen werden, die angemessene Korrektur- und Rücksetzungsmaßnahmen ermöglicht. Der Abschluß der Überführung wird durch die fachliche Verantwortungsübernahme durch den Anwender und die technische durch den Rechenbetrieb markiert.

---

[7] Diese Prüfmethoden entsprechen denen, die auch bei der Software-Entwicklung eingesetzt werden, jedoch sind manuelle Verfahren bei der Wartung wesentlich aufwendiger als bei der Neuentwicklung, da außer dem betreffenden Wartungsingenieur oft keine weiteren Personen Kenntnisse über das System besitzen. Häufig werden daher bei der Wartung die Prüfungen und Tests aus kurzfristigen "praktischen" Gesichtspunkten auf ein Mindestmaß reduziert oder sogar ignoriert.

Die ersten drei Wartungsschritte setzen direkt auf den verschiedenen Repräsentationsformen des Software-Systems auf und werden in ihrem Aufwand unmittelbar durch dessen Wartungsfreundlichkeit beeinflußt. Von zentraler Bedeutung ist dabei die Erkenntnis, daß die Verständlichkeit (kognitive Komplexität) der Software-Strukturen[8] das wichtigste Aufwandskriterium bei der Durchführung von Wartungsarbeiten ist.[9] Folgende Eigenschaften des Software-Systems sind für dessen Verständlichkeit ausschlaggebend:

- Verfügbarkeit aller relevanten Systeminformationen: Dazu zählt die Vollständigkeit der programmtechnischen, benutzerspezifischen und organisatorischen Dokumentationen, wobei es unerheblich ist, ob die Beschreibungen physisch vorliegen oder ob sie ohne großen zeitlichen und kostenmäßigen Aufwand hergestellt (generiert) werden können.
- Eignung der verfügbaren Systembeschreibungen in bezug auf die jeweilige Wartungsaufgabe: Dazu zählen die Aktualität, Beschreibungsform (textlich, tabellarisch, graphisch usw.) und -qualität. Für die Beschreibungsqualität werden in der Literatur eine Reihe allgemeingültiger Merkmale genannt wie selbstbeschreibend, einfach, konsistent, strukturiert, einheitlich [Asam/86, S. 98 ff.].

Da bei vielen Wartungsaufgaben die benötigte Systemdokumentation nur unvollständig oder überhaupt nicht vorliegt und auch die Eignung vorhandener Unter-

---

[8] Verständlichkeit im Sinne von Effizienz, mit der die Komponenten und Beziehungen des Software-Systems vom Wartungspersonal auf der Basis von Programmquellen und Systemdokumentationen verstanden werden können.

[9] Der Wartungsaufwand verteilt sich auf "Verstehen des Programms" zu 47%, "Durchführen der Änderung" zu 25% und "Testen der Änderung" zu 28% [McClure/92, S. 36], dabei ist noch nicht berücksichtigt, daß auch das Durchführen und Testen von Änderungen durch die Programmverständlichkeit im Aufwand signifikant beeinflußt wird.

lagen oft sehr eingeschränkt ist, versucht man insbesondere in der Praxis,[10] Methoden und Werkzeuge zu entwickeln, um diese Mängel zu beseitigen und damit den bisher notwendigen Wartungsaufwand drastisch zu reduzieren. Teilt man nun die Methoden und Werkzeuge nach den genannten Kriterien, so können folgende Klassen unterschieden werden:

- Methoden und Werkzeuge, um die Vollständigkeit und Eignung der vorhandenen Systeminformationen zu beurteilen, z.B. Kennzahlen, Metriken, Checklisten.
- Methoden und Werkzeuge, um die Vollständigkeit der Systeminformationen zu erreichen, z.B. Nachdokumentation.
- Methoden und Werkzeuge, um die Eignung der Systeminformationen zu erhöhen, z.B. Transformation von Beschreibungen.

Diese Methoden und Werkzeuge haben gemeinsam, daß sie auf vorhandenen Informationen über das Software-System aufsetzen und aus diesen neue Sichten und Darstellungen erzeugen. Über eine Verbesserung der Systemdokumentation hinaus können sie auch bei bestimmten Wartungsmaßnahmen unmittelbar zur Durchführung des Änderungsschrittes eingesetzt werden. Dies gilt insbesondere für die Anpassung von Programmen an andere DV-technische Gegebenheiten[11] (moderne Benutzeroberflächen, andere Programmiersprachen, neue Systemsoftware-Umgebungen usw.), bei denen ebenfalls aus einer bestehenden eine neue Systemrepräsentation erzeugt wird. In Anbetracht der aktuellen und noch drastisch wachsenden zukünftigen Bedeutung der Wartung - verbunden mit außergewöhnlich attraktiven Marktchancen für Anbieter in diesem Bereich - lag es

---

[10] Im wissenschaftlichen Bereich hat die Auseinandersetzung mit der Wartungsproblematik bei weitem nicht den Stellenwert wie in der Praxis. Dies läßt sich zum einen darauf zurückführen, daß im Hochschulbereich normalerweise nicht so viele umfangreiche DV-Systeme eingesetzt werden, deren reibungsloser Betrieb zur Aufrechterhaltung der lfd. Forschung und Lehre unbedingt notwendig ist. Ergeben sich aufwendigere Wartungsarbeiten, so werden die Altsysteme durch Neuentwicklungen ersetzt, zumal i.d.R. auch die Entwicklungskapazitäten vorhanden sind. Da somit das Erfahrungsobjekt fehlt, dürfte die Sensitivität für diese Problemstellung nicht so hoch sein wie in der Praxis. Zum anderen setzt eine systematische Bearbeitung des Wartungskomplexes neben theoretischen vor allem auch praktische Kenntnisse sowohl in der alten als auch in der neuen DV-Welt voraus, was durch die Novität und hohe Änderungsdynamik der Disziplin Informatik nur selten anzutreffen ist.

[11] Die Anpassung eines Programms an andere Basistechnologien wird auch als Migration bezeichnet.

nahe, diese Methoden und Werkzeuge unter einen gemeinsamen marketingwirksamen Begriff, den des Reengineering, zu subsumieren.

2.2 Software-Wiederverwendung und Software-Reengineering

Nach Biggerstaff und Perlis [Biggerstaff/89b, S. XV] liegt Software-Wiederverwendung vor, wenn das Wissen oder Wissensteile über ein System zur Verringerung des Entwicklungs- und Wartungsaufwands wieder bei einem anderen ähnlichen System eingesetzt werden. Sie ist somit keine primäre Aufgabe innerhalb des Software-Lebenszyklus', sondern es handelt sich um Methoden und Werkzeuge (Hilfsmittel) zur Unterstützung von Neuentwicklung und Wartung von Software-Systemen. Dabei wird unterstellt, daß durch Mehrfachnutzung von Systemteilen der Entwicklungsaufwand, der meist höher ist als bei Einmalnutzung, nur einmal entsteht und anschließend der Aufwand zum Wiederauffinden und Anpassen relativ gering ist, so daß insgesamt der Erstellungsaufwand bezogen auf eine Nutzungseinheit drastisch sinkt. Durch den höheren Entwicklungsaufwand - und damit verbunden die gesteigerte Programmqualität - sinkt die Anzahl der Programmfehler, die auch wesentlich schneller erkannt werden und so in ihren Folgewirkungen beschränkt bleiben. Bei einem hohen Wiederverwendungsgrad ist daher auch mit einer wesentlichen Reduzierung des Wartungsaufwands zu rechnen.[12]

Software-Wiederverwendung im engeren Sinne beschränkt sich auf die Mehrfachnutzung von Programmquellen (reuse of code). Sie ist seit Beginn der Datenverarbeitung in Form von Unterprogrammbibliotheken bekannt und bis heute Gegenstand umfangreicher Entwicklungsanstrengungen, z.B. die Klassenhierarchien objektorientierter Programmiersprachen. Wird der Begriff weiter gefaßt, so beinhaltet er alle Informationen, die zu einem Software-System vorhanden sind (reuse of design), d.h. Informationen über fachliche, anwendungssystem- und programmsystem-bezogene Strukturen. Insbesondere dieser erweiterten Art der Wiederverwendung wird ein großes Potential zur Steigerung der Entwicklungs- und Wartungsproduktivität sowie Software-Qualität vorausgesagt [Biggerstaff/89c, S. 9 f.], sofern u.a. geeignete Repräsentationsformen für Design-Informationen verfügbar sind. Grundsätzlich sind folgende Arten der Software-Wiederverwendung zu unterscheiden [Endres/88]:

---

[12] Empirische Untersuchungen und Prognosen schätzen den möglichen Grad der Software-Wiederverwendung auf 30 bis 60% des gesamten Programmkodeumfangs [McClure/92, S. 227].

- Programmportierung: Bei ihr werden Programme ohne Quellkode-Modifikation von einem zum anderen Rechnersystem übertragen. Dies setzt weitgehend standardisierte Umgebungen, z.B. SAA- oder UNIX-Systeme, und strikte Einhaltung der entsprechenden Richtlinien bei der Programmierung, z.B. keine Nutzung systemspezifischer Besonderheiten und Verwendung standardisierter Sprachen wie COBOL und SQL, voraus.
- Programmadaptierung: Bei ihr werden während der Programmentwicklung bereits Möglichkeiten vorgesehen, die Funktionalität ohne großen Aufwand spezifischen Gegebenheiten anzupassen. Je geringer der Anpassungsaufwand, um so wirtschaftlicher ist die Programmadaptierung im Verhältnis zur Neuentwicklung. Maßnahmen zur Adaption sind tabellen- oder parametergesteuerte Abläufe bzw. Funktionalitäten sowie dezidierte Programmtextteile, die anwendungsspezifisch programmiert werden können (user exit). Diese Wiederverwendungsart ist besonders häufig bei Anwendungsstandardsoftware anzutreffen, z.B. bei Finanzbuchhaltungs-, Lohnabrechnungs- und Auftragsbearbeitungssystemen.
- Generierungstechnik: Bei ihr werden auf der Grundlage von Schablonen, Rahmen oder allgemein standardisierten Informationsstrukturen sowie anwendungsspezifischen Zusatzinformationen mit Hilfe von Generatoren Programme erzeugt.
- Baukastentechnik: Hier werden in sich abgeschlossene Einheiten (Unterprogramme, Objekte) durch geeignete Kompositionstechniken zu größeren Einheiten zusammengesetzt, bis vollständige Programme entstanden sind.

Wiederverwendung ist Standardisierung von komplexen Komponenten und weist damit auch alle Vor- und Nachteile sowie Hindernisgründe auf, die mit Standards verbunden sind. Je häufiger eine Komponente eingesetzt werden soll, um so allgemeiner (abstrakter) muß sie spezifiziert sein und um so schwieriger ist sie zu verstehen und anzupassen. Mit zunehmender Verallgemeinerung von Standardkomponenten steigt auch der "Overhead" an Funktionalität, der bei einer spezifischen Nutzung nicht benötigt wird, verbunden mit den Nachteilen Unübersichtlichkeit, Leistungseinbußen und Akzeptanzbarrieren. Weiterhin hinken Standards im Informatikbereich oft neuen Entwicklungsrichtungen und aktuellen Erfahrungswerten hinterher, so daß sie leicht als veraltet beiseite geschoben werden. Dieser nicht vollständigen Aufzählung von Nachteilen steht ein großes Potential an Einsparungen bei Entwicklung, Test und Wartung von Software gegenüber.

Vergleicht man den Zusammenhang zwischen Software-Wiederverwendung und Reengineering, so wird deutlich, daß beide die Erhöhung der Produktivität von Neuentwicklung und Wartung zum Ziel haben. Mittels Reengineering werden aus bestehenden Systemen wiederverwendbare Teile erkannt, extrahiert und einer erneuten Nutzung verfügbar gemacht, ohne daß damit Bedingungen an Systemkomponenten und -strukturen verbunden sind (ungeplante Wiederverwendung). Software-Wiederverwendung geht davon aus, daß vor der Entwicklung eines Programms die mehrfach verwendbaren Teile bereits standardisiert vorliegen und zum Aufbau des Systems verwendet werden (geplante Wiederverwendung). Damit ist die Abgrenzung offensichtlich; Reengineering ist von seinem Charakter her eine Erhebungsmethode, während Wiederverwendung eine Konstruktionsmethode darstellt, was jedoch auch einschließt, daß Reengineering zur Identifikation wiederverwendbarer Systemteile eingesetzt werden kann und die Verwendung von standardisierten Komponenten aufgrund ihres Formalismus' die Effizienz der Durchführung einer Reengineering-Maßnahme sehr positiv beeinflussen kann.

2.3 Abgrenzung und Teilbereiche des Software-Reengineering

Wie aus den vorangegangenen Abschnitten ersichtlich, spielt Reengineering als Mittel zur Verbesserung des Wartungsprozesses und Erhöhung der Software-Produktqualität mit Hilfe von entsprechenden Methoden und Werkzeugen eine zentrale Rolle. Ebenso ermöglicht es eine Produktivitätssteigerung bei der Entwicklung neuer Anwendungen durch das Erkennen und Bereitstellen wiederverwendbarer Teile existierender Systeme. Nach McClure [McClure/92, S. 23 f.] versteht man folglich unter Reengineering die Analyse und Modifikation existierender Software-Systeme unter Verwendung automatisierter Werkzeuge, um deren Wartbarkeit zu verbessern, sie neuen Technologien anzupassen, ihre potentielle Einsatzdauer zu verlängern, die Wartungsproduktivität zu erhöhen sowie deren Komponenten und Strukturen in einer Entwicklungsdatenbank (repository) zu erfassen und zu verwalten. Dabei vertritt die Autorin die Auffassung, daß der Einsatz von Werkzeugen ein wesentliches Merkmal für Reengineering ist. Die unterschiedlichen Methoden und Werkzeuge des Reengineering können ergebnisorientiert in folgende Teilgebiete aufgeteilt werden [McClure/92, S. 25 ff.]:

- Programmanalyse: Untersuchung existierender Programmsysteme, um qualitative und quantitative Aussagen über deren Komponenten und Strukturen, Arbeitsweise und Wartbarkeit treffen zu können. Als Hilfsmittel können dabei automatische Programmanalysatoren (data/logic tracers,