

Arbeitsberichte des Instituts für Wirtschaftsinformatik

Herausgeber: Prof. Dr. J. Becker, Prof. Dr. H. L. Grob, Prof. Dr. K. Kurbel,
Prof. Dr. U. Müller-Funk, Prof. Dr. R. Unland

Arbeitsbericht Nr. 13

Reengineering

Stefan Eicker, Thomas Schnieder

Institut für Wirtschaftsinformatik der Westfälischen Wilhelms-Universität Münster,
Grevener Str. 91, 4400 Münster, Tel. (0251) 83-9750, Fax (0251) 83-9754

August 1992

Inhalt

1	Einführung	4
1.1	Die Situation im Wartungsbereich	4
1.2	Begriffsbestimmungen	7
2	Nutzen des Reengineering	9
3	Vorarbeiten für einen Reengineering-Prozeß	12
3.1	Die Auswahl von Kandidaten für Reengineering-Prozesse	12
3.2	Einarbeitung in ein Anwendungssystem	13
4	Reengineering-Werkzeuge	15
4.1	Die Bedeutung von Reengineering-Werkzeugen	15
4.2	Anforderungen an Reengineering-Werkzeuge	15
4.3	Verfügbare Reengineering-Werkzeuge	17
4.3.1	Structuring Engine	17
4.3.2	Structuring Facility	18
4.3.3	Amelio	18
4.3.4	Re-Spec und Epos	19
4.3.5	Datatec	20
4.3.6	Puns	20
4.4	Weitere für das Reengineering nutzbare Werkzeuge	20
4.4.1	Case-Tools	20
4.4.2	Pretty-Printer	21
4.4.3	Debugger	21
5	Fallbeispiele	21
5.1	DST Systems	21
5.2	Rona-Dismat Gruppe	23
6	Ausblick	23

Zusammenfassung

In Theorie und Praxis der betrieblichen Datenverarbeitung gewinnen Projekte zur systematischen Überarbeitung von (insbesondere alten) Anwendungssystemen immer mehr an Bedeutung. Als Bezeichnung für das entsprechende Wissens-/Forschungsgebiet hat sich der Begriff *Reengineering* etabliert. Der vorliegende Arbeitsbericht motiviert das Reengineering und weist auf seinen möglichen Nutzen hin. Darüber hinaus wird eine Übersicht einerseits über Anforderungen an Reengineering-Werkzeuge und andererseits über die zur Verfügung stehenden Tools gegeben. Außerdem werden zwei erfolgreiche Reengineering-Projekte vorgestellt.

1 Einführung

1.1 Die Situation im Wartungsbereich

Im Laufe der vergangenen Jahrzehnte hat sich in den großen und mittelständischen Unternehmen eine Vielzahl von Anwendungssystemen angesammelt. Die Wartung der Systeme bindet einen erheblichen Teil der Personalressourcen in den DV-Abteilungen: Bereits Ende der 70er Jahre kamen zwei Untersuchungen zu dem Ergebnis, daß etwa 50% des DV-Personals für Wartungsarbeiten eingesetzt werden¹⁾, wobei der Anteil in größeren Unternehmen tendenziell höher ist als in kleinen. Der heutige Wartungsanteil an den Gesamtaufwendungen für die Datenverarbeitung wird von der Gartner Group sogar auf über 80% veranschlagt (vgl. Abbildung 1):

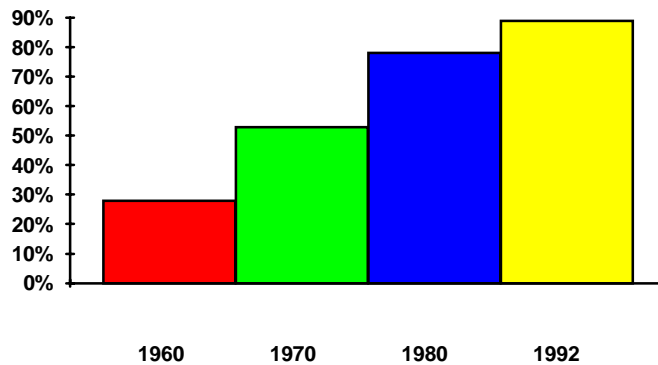


Abb. 1: Anteil der Wartung an den Gesamtaufwendungen für den DV-Bereich²⁾

Ein wesentlicher Faktor für den hohen und weiter wachsenden Aufwand, der in die Pflege von Altsystemen investiert werden muß, liegt in der schlechten softwaretechnischen Qualität der Altsysteme. Nach einer Umfrage sind zum Beispiel 82% der in der Bundesrepublik Deutschland installierten Cobol-Programme monolithisch, d.h. ohne Verwendung der Unterprogrammtechnik, und 77% unstrukturiert programmiert³⁾. Ihren Ursprung hat die mangelnde Qualität der Altsysteme im wesentlichen darin, daß zur Zeit ihrer Entwicklung Entwurfsmethoden und -techniken noch nicht verfügbar waren oder noch nicht angewendet wurden. Das "Verstehen" eines *unstrukturierten* Programms ist aber wesentlich schwieriger und zeitaufwendiger als das Verstehen eines *strukturierten* Programms. Hinzu kommt, daß eine Dokumentation in Gestalt formaler oder textueller Beschreibungen, die das Verständnis erleichtern könnten, in vielen Fällen nicht zur Verfügung steht, unvollständig ist oder aber

1) Vgl. Fjeldstad, Hamlet (1979) bzw. Lientz, Swanson (1980).

2) CAP debis GEI (1992), S. 2.

nicht mehr zu der aktuellen Programmversion paßt. Darüber hinaus kann auf die Entwickler eines Altsystems in der Wartungsphase häufig nicht mehr zurückgegriffen werden. Die Wartungsprogrammierer müssen sich somit mühsam in fremde Programme einarbeiten.

Die Arbeiten, die im Rahmen der Wartung an einem Anwendungssystem zu leisten sind, können in verschiedene Kategorien eingeteilt werden⁴⁾. Dazu gehören im wesentlichen:

1. die korrigierende Wartung, d.h. die Eliminierung von Entwurfs- und/oder Implementierungsfehlern;
2. die adaptive Wartung, d.h. die Anpassung des Systems an neue Hardware und/oder neue Basissoftware;
3. die optimierende Wartung, d.h. die Verbesserung bzw. Erstellung der Programmdokumentation, die Speicher- und Laufzeitoptimierung sowie allgemein die Verbesserung der Qualität des Systems (z.B. Erhöhung der Testbarkeit, der Sicherheit, der Portabilität, der Änderbarkeit, der Benutzerfreundlichkeit etc.);
4. die erweiternde Wartung, d.h. die Anpassung an Benutzerwünsche (in bezug auf die Erweiterung/Änderung existierender bzw. das Hinzufügen neuer Funktionen).

Abbildung 2 zeigt, wie sich der Wartungsaufwand auf die vier Kategorien verteilt.

Anpassungen eines Systems im Rahmen der erweiternden Wartung werden aus verschiedenen Gründen durchgeführt. Zu möglichen Auslösern zählen:

- Marktveränderungen (z.B. Individualisierung von Verträgen im Bereich der Lebens- und Berufsunfähigkeitsversicherungen auf dem EG-Binnenmarkt);
- Veränderungen in der Ablauforganisation (die Systeme, die die Bearbeitung entsprechender Vorgänge unterstützen, sind an die geänderte Organisation anzupassen)⁵⁾;
- Veränderungen in der Aufbauorganisation (die Zugriffsrechte auf Daten und Anwendungen müssen den Veränderungen angepaßt werden)⁶⁾;

3) Vgl. Sneed (1992), S. 20.

4) Vgl. Lientz, Swanson (1980), S. 68.

5) Vgl. Thurner (1990), S. 13.

- Veränderungen der technologischen Basis (d.h. der Basissysteme wie Betriebssystem, Datenhaltungssystem und Kommunikationssystem, der Entwicklungsumgebung, der Programmiersprache, der Mensch-Maschine-Schnittstellen etc.);
- Probleme durch die fehlende Integration von Anwendungssystemen (Stichworte "Datenredundanz" und "Funktionsredundanz");
- Gesetzesänderungen⁷⁾;
- Benutzerwünsche⁸⁾.

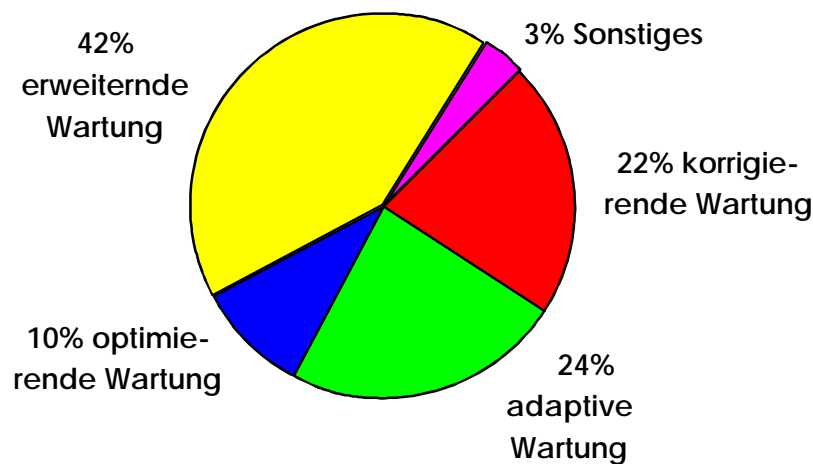


Abb. 2: Anteile der Wartungskategorien am Gesamtwartungsaufwand⁹⁾

Eine Studie im Rahmen des Esprit-Metkit-Projekts kommt dabei zu dem Ergebnis, daß die Anpassung eines Systems, die nicht auf eine Qualitätsverbesserung abzielt, stets die Wartbarkeit des Systems weiter verschlechtert¹⁰⁾. In konkreten Zahlen heißt dies, daß die Komplexität eines Programms nach einer Korrektur durchschnittlich um 4%, nach einer Änderung um 17% und nach einer Erweiterung um 26% steigt. Das bedeutet, daß der für die nächste Modifikation notwendige Wartungsaufwand stetig zunimmt; am Ende steht schließlich ein Softwareprodukt, das praktisch nicht mehr wartbar ist.

6) Vgl. ebenfalls Thurner (1990), S. 13.

7) Vgl. Kurbel (1983), S. 96.

8) Vgl. ebenfalls Kurbel (1983), S. 96.

9) Lientz, Swanson (1980), S. 73.

10) Vgl. Sneed, Kaposi (1990).

Zu beachten ist in diesem Zusammenhang, daß die Notwendigkeit zur Ablösung oder qualitativen Verbesserung eines Altsystems von den Verantwortlichen häufig erst dann eingesehen wird, wenn der Betrieb des Systems nur noch mit größtem Aufwand sichergestellt werden kann. Der Grund dafür liegt im wesentlichen darin, daß nicht - wie z.B. bei Betriebsmitteln - mit einem Verschleiß argumentiert werden kann.

1.2 Begriffsbestimmungen

Definition: (Reengineering)

Reengineering umfaßt alle Aktivitäten, die mit der Analyse und/oder Überarbeitung von Software in Verbindung stehen. Der Programmcode kann dabei unter Beibehaltung seiner Funktionalität verändert werden.

Reengineering-Aktivitäten lassen sich im wesentlichen (vgl. unten) zwei Schwerpunkten zuordnen, einerseits dem Reverse-Engineering, andererseits der Restrukturierung.

Definition: (Reverse-Engineering)

Reverse-Engineering umfaßt alle Aktivitäten, die darauf abzielen, aus einem Programmcode eine Dokumentation des zugehörigen (Grob- oder Fein-)Entwurfs oder der zugehörigen Spezifikation abzuleiten. Der Programmcode selbst bleibt unverändert.

Reverse-Engineering beinhaltet damit eine reine Dokumentationstätigkeit: Es werden nachträglich die Dokumente erstellt, die heute in einem Entwicklungsprozeß standardmäßig erstellt werden bzw. erstellt werden sollten. Abbildung 3 verdeutlicht diesen Sachverhalt: In der Mitte sind von oben nach unten Dokumente aufgeführt, die die Ergebnisse der jeweiligen Phase (bei dem zugrundeliegenden Phasenmodell) darstellen. Die linke Seite zeigt den Ablauf eines Software Engineering-Prozesses - im Zusammenhang mit dem Reverse-Engineering auch als Forward-Engineering bezeichnet; Input für eine Phase sind jeweils die Ergebnisse der vorhergehenden Phasen. Auf der rechten Seite ist demgegenüber die Vorgehensweise des Reverse-Engineering dargestellt, das - ausgehend vom Quellcode und der eventuell existierenden Dokumentation - den Entwicklungsprozeß eines Anwendungssystems bzw. seine Ergebnisse in umgekehrter Richtung nachzuvollziehen versucht.

Definition: (Restrukturierung)

Unter dem Begriff Restrukturierung werden Tätigkeiten zusammengefaßt, die versuchen, einem unstrukturierten Programm eine Struktur aufzuprägen. Die Funktionalität des Programms bleibt dabei unverändert.

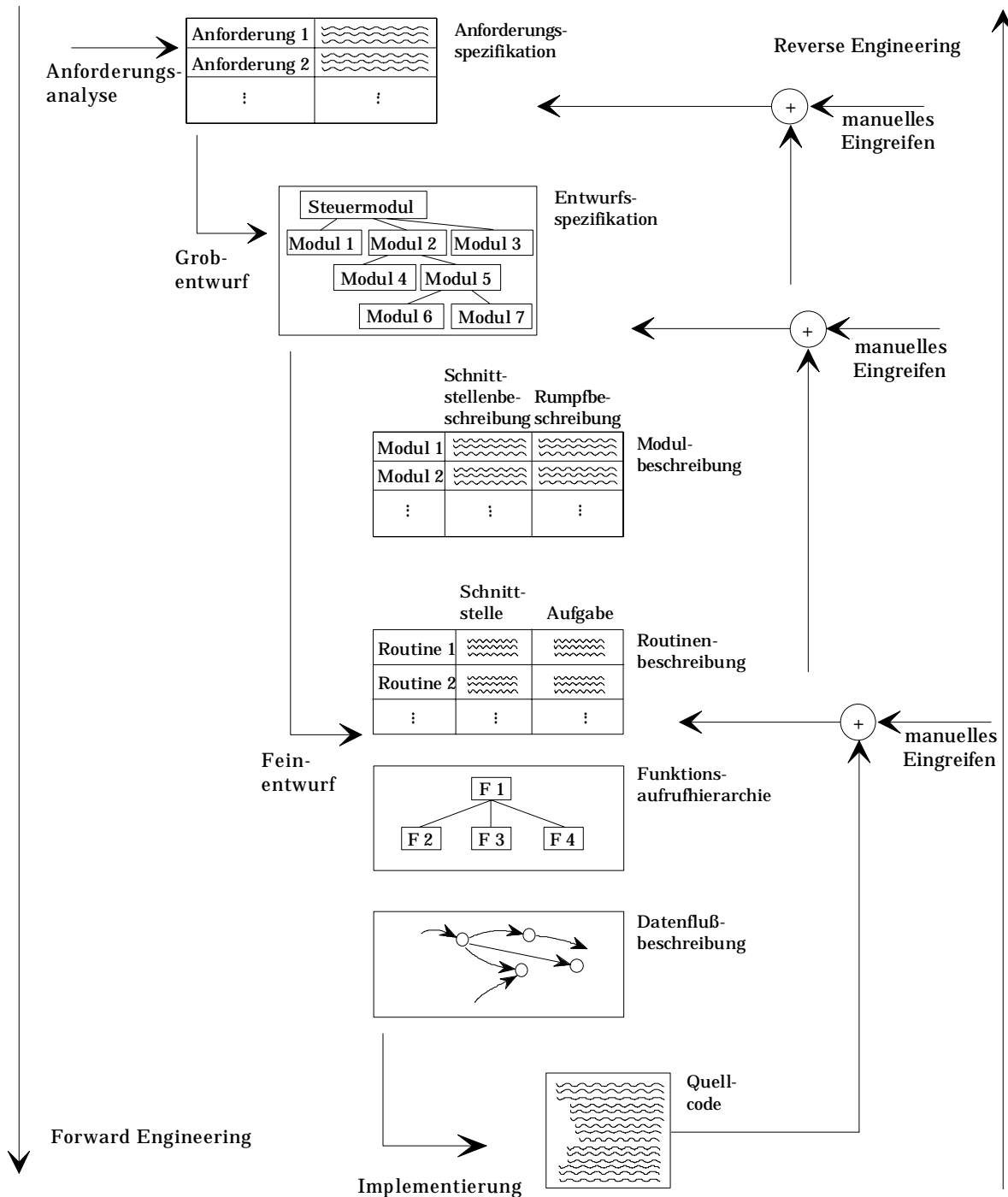


Abb. 3: Forward-Engineering und Reverse-Engineering

Ziel der Restrukturierung ist es vor allem, das Programm für den Leser verständlich zu machen und auf diese Weise die Wartbarkeit des Programms zu erhöhen. Teilziele bei einem Restrukturierungsprozeß sind vor allem die Eliminierung von Goto-Anweisungen, die Strukturierung des Systems durch Auslagerung von entsprechenden Anweisungsfolgen in Unterprogramme sowie allgemein die Reduzierung der Ablauf- und der Datenkomplexität.

2 Nutzen des Reengineering

Reengineering als Alternative zum unveränderten Weiterbetrieb

Altsysteme bilden häufig tragende Säulen der Datenverarbeitung eines Unternehmens. Entsprechend dieser ausgezeichneten Stellung wird auf die Systeme im allgemeinen das Prinzip der Risikovermeidung und davon abgeleitet das Prinzip der geringsten Veränderung angewendet. Weiterhin ist in die Entwicklung von Altsystemen und in die Fehlerbeseitigung viel Zeit investiert worden. Erfahrungen und Wissen um unternehmensspezifische Sachverhalte wurden in den Systemen abgebildet, Ablauf- und Aufbauorganisation mit den Systemen abgestimmt¹¹⁾. Im Laufe der Zeit sind für sich isoliert betrachtet relativ verlässliche Systeme entstanden, die den Anwendern vertraut sind. Eine Neuentwicklung kommt aus diesen Gründen in vielen Fällen nicht in Frage. Zum unveränderten Weiterbetrieb bildet aber das Reengineering eine echte Alternative. Es erlaubt,

- die in die Programme getätigten Investitionen zu erhalten;
- das im Programm angesammelte Wissen zu retten;
- das Wissen über die Anwendung zu vertiefen; insbesondere werden Dokumente erstellt, die die Wartung erleichtern¹²⁾;
- die Komplexität eines Programmes zu verringern; dadurch erreichte Designverbesserungen helfen, den Aufwand für alle genannten Wartungsaktivitäten zu verringern¹³⁾;

11) Vgl. Kurbel (1992), S. 36.

12) Vgl. Wagner (1992b).

13) Vgl. Lientz, Swanson (1980), S. 68.

- alte Vorgaben oder Systembeschränkungen aus dem Programmcode abzuleiten; eine Überprüfung dieser Randbedingungen auf ihre Aktualität kann Änderungen nahelegen, die die Wartbarkeit und/oder das Laufzeitverhalten der Software verbessern.

Reengineering als Alternative zur Neuentwicklung

Wie auch aus den am Ende dieses Berichts skizzierten Reengineering-Projekten ersichtlich, kann das Reengineering von Programmen - abhängig von den Programmen und der Zielsetzung - mit weniger personellem und finanziellem Aufwand auskommen als entsprechende Neuentwicklungen. Das Reengineering eines Anwendungssystems bzw. von bestimmten Systemteilen stellt somit grundsätzlich eine überlegenswerte Alternative zur Neuentwicklung dar.

Reengineering zur Verbesserung der Portabilität

Eine wesentliche Voraussetzung für die langfristige Nutzung von Anwendungssystemen ist ihre Portabilität. Portabilität ist dabei nicht nur von der verwendeten Programmiersprache, sondern auch von der Systemstruktur sowie davon abhängig, inwieweit bei der Programmierung Standards eingehalten wurden¹⁴). Zur Steigerung der Portabilität kann ein (partielles) Redesign eines Altsystems durchgeführt werden. Ergebnis der Überarbeitung sollte ein portables Quellprogramm mit virtuellen Schnittstellen für Hardware, Betriebssystem und gegebenenfalls für andere Anwendungssysteme sein. Spezielle (umgebungsabhängige) Module stellen dann die Verbindung zum jeweiligen realen Umfeld her. Bei einer Portierung müssen "nur" diese Module verändert werden.

Die Transformation von Programmquellcode in eine andere Programmiersprache oder einen anderen Dialekt wird in der Literatur ebenfalls als Ziel von Reengineering-Tätigkeiten genannt¹⁵). Durch eine solche (möglichst weitgehend automatisch durchgeführte) Transformation kann ebenfalls die Portabilität eines Anwendungssystems erreicht werden.

Reengineering zur Durchsetzung von Unternehmensstandards

Reengineering-Aktivitäten können mit dem Ziel durchgeführt werden, für Altsysteme einheitliche und/oder benutzerfreundlichere Oberflächen zu realisieren. Auch lassen sich durch die Erstellung entsprechender Dialogsoftware mehrere Applikationen zu einer einzigen

¹⁴) Vgl. Henselmann (1992), S. 16.

¹⁵) Vgl. Wagner (1992a), S. 96 ff.

zusammenfassen. Zunächst werden jeweils die Eingaben und die Ausgaben der Systeme analysiert. Die neue Oberfläche bzw. die Dialogsoftware wird dann jeweils so in ein System eingebunden, daß sie das ursprüngliche Ein-/Ausgabe-Verhalten simuliert.

Reengineering zur Verwirklichung der Daten- und der Funktionsintegration

Sowohl in bezug auf die Daten als auch in bezug auf die Funktionen bestehen in den Unternehmen starke Redundanzen. Nach einer entsprechenden Untersuchung in der Bundesrepublik Deutschland sind etwa 93% der Daten, die von Cobol-Programmen verarbeitet werden, überflüssig, da sie entweder redundant sind oder aus Unwissenheit oder Unsicherheit nicht gelöscht werden¹⁶⁾. Jones schätzt, daß höchstens 15% der 1983 erstellten Programme aus neuen Routinen besteht¹⁷⁾. Reengineering-Prozesse sollten deshalb auch das Ziel verfolgen, die Datenintegration und die Funktionsintegration zu unterstützen. Ein solcher Reengineering-Ansatz wird auch als *integrationsorientiertes Reengineering* bezeichnet¹⁸⁾.

Reengineering zur Aufwandschätzung

Die Analyse eines Anwendungssystems kann zur Aufwandschätzung bestimmter Anpassungs- oder Erweiterungsmaßnahmen dienen. Solche Aufwandschätzungen werden insbesondere benötigt, um zwischen den Handlungsalternativen "Neuentwicklung/Kauf" und "Reengineering" entscheiden zu können. Desweiteren eignet sich das gewonnene Wissen als Ausgangspunkt für die Zeitplanung. In der Praxis geht man teilweise sogar soweit, den Aufwand für die Analyse eines Altsystems nicht dem (nachfolgenden) Reengineering-Prozeß für das System zuzuordnen, sondern dem Entscheidungsfindungsprozeß bezüglich der Handlungsalternativen¹⁹⁾.

Reengineering zur Ermittlung von Wissen

Wie bereits angesprochen, sammelt sich im Laufe der Zeit in den Anwendungssystemen Entwickler- und Anwenderwissen an, das häufig ausschließlich durch den Programmcode selbst repräsentiert wird. Mit Hilfe eines Reengineering-Prozesses kann dieses Wissen ermittelt werden, um es dann in eine Neuentwicklung einfließen zu lassen oder zur Anpassung von Standardsoftware an die speziellen Anforderungen des Unternehmens zu nutzen.

¹⁶⁾ Vgl. Sneed (1992), S. 20.

¹⁷⁾ Vgl. Jones (1984), S. 488.

¹⁸⁾ Vgl. Eicker et al. (1992).

¹⁹⁾ Vgl. Rochester, Douglass (1991), S. 7.

Reengineering als Software-Recycling-Maßnahme

Ein noch weiter gehender Ansatz besteht darin, durch Reengineering-Maßnahmen wiederverwendbare Teile des Entwurfs und des Quellcodes eines Programms zu identifizieren, zu isolieren und in anderen Programmen zu benutzen²⁰). In der Literatur wird in diesem Zusammenhang auch von Software-Recycling gesprochen. Bezogen auf eine Neuentwicklung gelangt man so zu einem Bottom-up-Ansatz, in dessen Mittelpunkt die durch Analyse gewonnenen Teile der bereits existierenden Software stehen. Eine solche Vorgehensweise kann dazu beitragen, Entwicklungskosten und -risiken zu reduzieren.

Mit der Form der Softwarekomponenten, die sich für eine Wiederverwendung eignen, setzt sich Standish auseinander²¹). Er berichtet in seiner Arbeit auch von Beispielen für Software-Recycling-Projekte und von dem jeweils erzielten Nutzen.

3 Vorarbeiten für einen Reengineering-Prozeß

3.1 Die Auswahl von Kandidaten für Reengineering-Prozesse

Zur Unterstützung der Auswahl der Systeme, für die ein Reengineering-Prozeß sinnvoll wäre, hat das US National Bureau of Standards elf Kriterien herausgearbeitet²²):

- Das Anwendungssystem ist wegen Fehlern häufig außer Betrieb.
- Der Programmcode ist älter als sieben Jahre.
- Die Programmstruktur bzw. die Ablauflogik überschreitet ein "gewisses" Maß an Komplexität.
- Das System wurde für eine ältere Maschine geschrieben.
- Die für das System notwendige Umgebung wird emuliert.
- Einzelne Module des Systems sind "zu groß" geworden.
- Der Ressourcenbedarf des Systems - Laufzeit und Speicher - ist "zu hoch" geworden.
- Fest im System eingebaute Parameter bzw. Entwurfskonstanten wurden umgestoßen.
- Die für die Wartung des Systems notwendige Ausbildung wird zu teuer.
- Die technische Dokumentation des Systems ist unbrauchbar geworden.

²⁰) Vgl. ebenfalls Rochester, Douglass (1991), S. 6.

²¹) Vgl. Standish (1984).

²²) Vgl. NBS (1983).

- Die Spezifikation des Systems ist mangelhaft, unvollständig oder inkonsistent.

Weitere mögliche Auswahlkriterien nennt Sneed²³):

- die Zufriedenheit der Benutzer,
- die strategische Bedeutung des Systems für das Unternehmen,
- die Zukunftsperspektive des Systems (wann wird es ersetzt ?) sowie
- die Höhe der aktuellen Wartungskosten.

In den Entscheidungsprozeß, welche Systeme bzw. welche Teile von welchen Systemen tatsächlich einem Reengineering-Prozeß unterzogen werden sollen, müssen ökonomische Überlegungen einbezogen werden. Dazu muß prinzipiell jeweils die Wirtschaftlichkeit der Handlungsalternativen ("unveränderter Weiterbetrieb", "Neuentwicklung/Kauf", "Reengineering") bewertet werden. Aufwand und Nutzen der Alternativen zu quantifizieren ist allerdings schwierig, auf der Nutzen-Seite praktisch unmöglich. Systeme zur Unterstützung der Wirtschaftlichkeitsbetrachtung bezogen auf das Reengineering oder zur Unterstützung des gesamten Entscheidungsprozesses stehen noch nicht zur Verfügung.

3.2 Einarbeitung in ein Anwendungssystem

Vor der Durchführung des eigentlichen Reengineering-Prozesses müssen sich die Projektmitarbeiter in die Programme des Anwendungssystems einarbeiten, das überarbeitet werden soll. Nach Corbi existieren drei Möglichkeiten, sich in ein Programm einzuarbeiten²⁴):

1. das Studium der Dokumentation;
2. das Lesen des Quellcodes;
3. die Beobachtung des dynamischen Verhaltens.

Die Dokumentation eignet sich dabei besonders gut, weil das Programm in ihr in einer abstrakten, dem menschlichen Denken eher entsprechenden Form repräsentiert wird. Bei Alt-systemen ergibt sich allerdings das oben angesprochene Problem, daß eine Dokumentation überhaupt nicht existiert oder aber unvollständig bzw. nicht mehr aktuell ist. Das (teilweise) Lesen des Quellcodes kann deshalb (und weil für die Durchführung von Änderungen ein tieferes Verständnis des Programmcodes notwendig ist) als unerläßlich angesehen werden.

²³) Vgl. Sneed (1992), S. 129 ff.

²⁴) Vgl. Corbi (1989), S. 299.

Corbi nennt drei Theorien, wie Wartungsprogrammierer sich in einen Programmcode einarbeiten²⁵⁾:

1. Die Bottom-up-Theorie

Der Programmierer liest den Quelltext und abstrahiert im Laufe seiner Tätigkeit immer stärker vom eigentlichen Programm. In Gedanken entsteht dabei ein High-Level-Gerüst des Softwareprodukts, in dem größere Teile durch Begriffe charakterisiert werden.

2. Die Top-down-Theorie

Der Programmierer versucht, sein Verständnis des Problems in die Welt der Programmierung abzubilden. Die Erschließung des Programms beinhaltet die Rekonstruktion von Teilen dieser Abbildung. Der Rekonstruktionsprozeß wird von Erwartungen getrieben, die zur Aufstellung, Validierung und Verfeinerung von Thesen führen. Beispielsweise wird der Programmierer in einem Lohnbuchhaltungssystem nach Routinen suchen, die die Aufnahme und das Löschen von Arbeitnehmern realisieren, oder nach Routinen, die mit Hilfe gewisser Attribute den Nettolohn aus dem Bruttolohn ermitteln.

3. Die opportunistische Theorie

Die Vorgehensweise des Programmierers besteht aus einer Mischung des Bottom-up- und des Top-down-Ansatzes.

Corbi kommt dabei zu dem Schluß, daß derselbe Programmierer in Abhängigkeit von der zu erfüllenden Aufgabe verschiedene Ansätze wählt.

Das dynamische Verhalten eines Programms wird in erster Linie betrachtet, wenn der Programmcode unverständlich ist. Insbesondere die Reaktion des Programms auf bestimmte Eingaben kann Hinweise auf den zugrundeliegenden Algorithmus geben.

²⁵⁾ Vgl. Corbi (1989), S. 301 ff.

4 Reengineering-Werkzeuge

4.1 Die Bedeutung von Reengineering-Werkzeugen

Die Wirtschaftlichkeit des Reengineering kann durch Tools verbessert werden, die den Programmierer so weit wie möglich von automatisierbaren Tätigkeiten entlasten. Die Verfügbarkeit von Tools beeinflusst somit die Entscheidung zwischen den Handlungsalternativen. Allerdings darf die Bedeutung von Werkzeugen im Reengineering-Prozeß nicht überschätzt werden. Zwar können einige (insbesondere vorbereitende) Reengineering-Maßnahmen von Tools übernommen werden (z.B. Restrukturierungen, Dokumentation von Funktionsaufrufhierarchien, Aufstellung von Datenflußdiagrammen etc.); "intelligente" Reengineering-Maßnahmen hingegen erfordern stets den Eingriff des Menschen.

4.2 Anforderungen an Reengineering-Werkzeuge

Um über Anforderungen an Werkzeuge, die den Reengineering-Prozeß unterstützen, Aussagen machen zu können, muß man sich über den Aufwand für verschiedene Tätigkeiten im Bereich der Weiterentwicklung und Pflege existierender Systeme Gedanken machen. Abbildung 4 zeigt die Aufwandsverteilung, die die Gartner Group in der oben bereits angesprochenen Studie ermittelte. Die Firma IBM kommt in einer ähnlichen Untersuchung ebenfalls zu dem Ergebnis, daß etwa 50% des Aufwands auf Analysetätigkeiten entfallen²⁶⁾. Daraus ergeben sich zwei prinzipielle Forderungen an Tools, die das Reengineering unterstützen:

- Sie müssen das Programmverständnis erleichtern.
- Sie müssen Anpassungsprozesse so weit wie möglich unterstützen bzw. selbständig durchführen; damit entfällt auch ein Teil des Testaufwands.

Brooks beschreibt den Kern eines Softwaresystems als ein komplexes Konstrukt von zusammenhängenden Konzepten: Daten, Beziehungen zwischen Daten, Algorithmen, Routinen und Modulen²⁷⁾. Der Wartungsprogrammierer muß diese Konzepte und ihre Beziehung zueinander verstehen. Zu typischen Informationen, die zur Erschließung des Programms beitragen und somit von Werkzeugen zu erheben sind, gehören:

²⁶⁾ Vgl. Andresen (1990), S. 95.

²⁷⁾ Vgl. Brooks (1987).

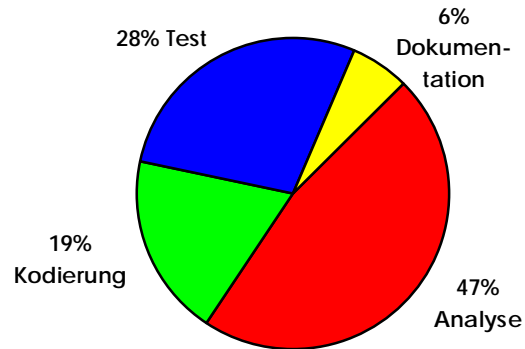


Abb. 4: Verteilung des Wartungsaufwands auf die verschiedenen Wartungstätigkeiten

- Informationen über Format, Nutzung und Definition von Variablen,
- Informationen über Schnittstellen eines Blocks zu seinem Umfeld,
- Listen von Daten, die von Modulen, Routinen und Blöcken importiert und exportiert werden,
- Hinweise auf globale Daten,
- Angabe von Stellen, an denen (bestimmte) Routinen aufgerufen werden,
- Auskunft über den Gültigkeitsbereich von Variablen,
- Liste der Routinen, in denen bestimmte Daten gelesen und/oder manipuliert werden,
- Angaben zu Datei- und Datenbankzugriffen sowie
- Informationen über Beziehungen zwischen den Dateien, die von einer Anwendung verwendet werden.

Darüber hinaus ist zu fordern, daß die Werkzeuge auf Mängel im Programm (einfaches Beispiel: eine Return-Anweisung in einem Cobol-Hauptprogramm) und mit Hilfe von Metriken auf "kritische" Programmteile (hohe Verschachtelungstiefen, viele Verzweigungen etc.) aufmerksam machen.

Auch die Kenntnis der jeweiligen Systemumgebung spielt bei der Bewältigung der Reengineering-Aufgaben eine zentrale Rolle²⁸⁾. Dem Reengineering-Programmierer sollten daher von unterstützenden Werkzeugen in übersichtlicher Form Informationen über die Basissoftware und die Hardware zur Verfügung gestellt werden. Informationen über benutzte Techniken und Tricks der Entwicklungs- und Wartungsprogrammierer sind zwar sehr

²⁸⁾ Vgl. Andresen (1990), S. 91.

nützlich für das Programmverständnis und damit für den Reengineering-Prozeß, können jedoch kaum von Werkzeugen erhoben werden. Allein die Einhaltung von Namenskonventionen und von Richtlinien kann im Prinzip weitgehend automatisch überprüft werden. Hier besteht auch grundsätzlich die Möglichkeit, den Quellcode durch einen weitestgehend automatisch ablaufenden Transformationsprozeß an solche Konventionen und Richtlinien anzupassen.

Eingesetzte Werkzeuge sollten auch die Fähigkeit besitzen, die gewonnenen Informationen in übersichtlicher Weise darzustellen. Insbesondere sollten Programmstruktur, Ablaufstruktur- und Datenflußdiagramme oder Datenbeschreibungstabellen generierbar sein.

Schließlich benötigt der Reengineering-Programmierer Hilfe bei der Durchführung von Änderungen. Eine solche Hilfestellung kann von einfachen Suche- und Ersetze-Möglichkeiten eines Editors zur konsistenten Änderung von Daten- und Routinennamen bis hin zur Überprüfung von aktuell getroffenen Entwurfsentscheidungen gehen. Zum Beispiel sollte ein Werkzeug nach der Einführung eines neuen Unterprogramms automatisch überprüfen, ob das Unterprogramm nur einen Eingang und einen Ausgang besitzt.

4.3 Verfügbare Reengineering-Werkzeuge

4.3.1 Structuring Engine

Structuring Engine gilt als das erste Werkzeug für die automatische Restrukturierung von Programmen²⁹⁾. Das Tool wurde bereits 1975 fertiggestellt; lauffähig ist es auf IBM-Mainframe-Rechnern. Auslöser für die Entwicklung von Structuring Engine war die Verfügbarkeit eines neuen Fortran-Derivats, das die strukturierte Programmierung unterstützt und die Lesbarkeit erhöht. Ein Precompiler übersetzt in dem Derivat geschriebene Programme in eine Fortran IV-Kodierung. Um auch die Lesbarkeit bereits existierender Programme zu verbessern, wurde mit Structuring Engine ein Werkzeug geschaffen, das Fortran IV-Programme in den entwickelten Dialekt übersetzt.

Das Tool zerlegt das Programm in Prozedurblöcke und ersetzt vorwärtsgerichtete Goto-Anweisungen durch If-then-else-Konstrukte, rückwärtsgerichtete Goto-Anweisungen durch Do-until-Schleifen sowie Goto-Anweisungen mit Sprungzielen außerhalb der Blöcke durch

²⁹⁾ Vgl. Sneed (1992), S. 317.

Prozeduraufrufe. Der Restrukturierungsprozeß läßt den Objektcode um 10% - 40% und auch die Laufzeit des Programmes um bis zu 8% anwachsen. Dies ist als ein wesentlicher Grund dafür anzusehen, daß sich das Tool in der Praxis nicht durchgesetzt hat.

4.3.2 Structuring Facility

Structuring Facility (SF) ist ein von IBM entwickeltes Tool, das Cobol-74- in Cobol-85-Programme konvertiert. Während der Konvertierung werden Vorwärts-Goto- in If-then-Ise-, Goto-depending on- in Evaluate-when-, Rückwärts-Goto- in Perform with test after- und Alter- in Perform-Anweisungen umgewandelt. SF dokumentiert außerdem in einem Graphen die Programmstruktur und macht Vorschläge zur Programmodularisierung. Weiterhin erstellt das Werkzeug Programmetriken; beispielsweise wird jeweils die Anzahl der Goto-, der Perform-, der Alter- und der If-Statements sowie die Tiefe von Verzweigungen erfaßt.

4.3.3 Amelio

Amelio ist ein von der Delta AG entwickeltes Tool, das unstrukturierte Cobol-Programme in strukturierte Delta/Cobol-Programme umsetzt. Das Tool fertigt folgende Dokumente an:

- eine Verbindungsliste, die Sprünge und Sprungziele visualisiert;
- Statistiken über die Anzahl der Vorwärts- und Rückwärts-Goto-Anweisungen, über die Anzahl der Lines of code in den Divisions, Sections und Paragraphs, über die Anzahl der Kommentarzeilen sowie über die Anzahl von Cobol-Verben und ihre Verteilung;
- eine Strukturbeschreibung in Form der "program architecture chart";
- eine eingerückte Version des Programms in einer Pseudokodierung;
- eine Programmdarstellung in Gestalt von Nassi-Shneiderman-Diagrammen.

Desweiteren unterstützt Amelio das Pfadtesten durch Angabe des ausgeführten Pfads und ermöglicht einen automatischen Vergleich zwischen der neuen und der alten Version des Programms, um Veränderungen zu dokumentieren. Mit Hilfe einer Sprache, die die Definition von Dokumentstrukturen erlaubt, können die erstellten Unterlagen zu einem Schriftstück zusammengefaßt werden.

Ein Beispiel für die Bearbeitung eines Programmes durch Amelio gibt Lauber³⁰⁾.

³⁰⁾ Vgl. Duckwitz (1990), Hirsch (1992), Sneed (1992).

4.3.4 Re-Spec und Epos

Epos (Entwicklungs- und Projektmanagement-orientiertes Spezifikationssystem) ist ein von der Gesellschaft für Prozeßrechnerprogrammierung mbH (GPP), München, entwickeltes Werkzeugsystem zur Rechnerunterstützung von Entwicklungs- und Projektmanagementtätigkeiten³¹). Epos enthält Sprachmittel zur Beschreibung des Programmablaufs, der Systemstruktur, der Datenstrukturen und der Datenflüsse. Der Pseudocode kann analysiert und in verschiedene Programmiersprachen (C, Ada, Cobol etc.) umgesetzt werden. Auch die Dokumentation ist aus dem Pseudocode ableitbar.

Darüber hinaus unterstützt Epos:

- die Wiederverwendung von Software,
- die Portierung von Software auf andere Hardware-/Basissoftware-Plattformen,
- die Übersetzung von Programmen in andere Programmiersprachen,
- die Restrukturierung von Programmen sowie
- das Re-Design von Systemen.

Für ältere Programme fehlt häufig die Spezifikation. Sie muß im Rahmen eines Reverse-Engineering-Projekts nachträglich erzeugt werden. Dazu wurde von der GPP das Tool Re-Spec entwickelt. Re-Spec analysiert vorhandene Software und erzeugt automatisch eine programmiersprachenunabhängige Spezifikation, die Informationen über die Systemstruktur, die Funktionen, den Ablauf, den Datenfluß und die Datenstrukturen enthält. Die Spezifikation wird in der Epos-Projektdatei gespeichert und kann so verarbeitet werden, als wäre das Programm mit Epos entworfen worden. Verfügbar ist Re-Spec für die Sprachen Ada, C, CMS-2, Fortran, Pascal und für verschiedene Assemblersprachen.

Verfügbar sind Re-Spec und Epos für eine Reihe von Hardware-Plattformen in der DEC-, in der HP- und in der Sun-Welt sowie für IBM-kompatible Personal Computer.

Ein Beispiel für die Bearbeitung eines C-Programmes durch Re-Spec und Epos beschreibt Lahm³²).

³¹) Vgl. Lauber (1990), S. 5.

³²) Vgl. Lahm (1992), S. 48 ff.

4.3.5 Datatec

Das Tool Datatec wurde von der XA Systems Corporation entwickelt. Es unterstützt den Programmierer bei der Anpassung von Datendefinitionen an unternehmensinterne Standards, insbesondere bei der Vergrößerung von Wertebereichen.

4.3.6 Puns³³⁾

Puns (Program Understanding Support Environment) erleichtert das Verständnis von Programmen, die in IBM 370-Assembler geschrieben wurden. Das Werkzeug besteht aus zwei Komponenten:

- einem Repository einschließlich Routinen zum Laden des Repository und zur Verarbeitung von High-Level-Queries sowie
- einer Schnittstelle, die Informationen graphisch präsentiert und den Programmierer beim "Durchgang" durch den Programmcode unterstützt.

Für das Programm, das bearbeitet wird, können verschiedene Views definiert werden. Der Wechsel zwischen den Views ist ebenso einfach möglich wie das Hinterfragen von Informationen innerhalb der Views.

4.4 Weitere für das Reengineering nutzbare Werkzeuge

4.4.1 Case-Tools

Verschiedene Case-Tools versuchen, auch die Wartung von Anwendungssystemen zu unterstützen. Die angebotenen Funktionen können häufig für das Reengineering genutzt werden. Ein Beispiel ist die InterCASE-Workbench der InterPort Software Corporation; das Werkzeug umfaßt Funktionen:

- zur Identifizierung und Eliminierung von nicht genutzten Anweisungen,
- zur Analyse der Laufzeit,
- zur Analyse der Aufrufverbindungen zwischen Programmen sowie

³³⁾ Vgl. Cleveland (1989).

- zur Verwaltung von Programmversionen.

4.4.2 Pretty-Printer

Als hilfreich erweist sich beim Reengineering auch der Einsatz eines Werkzeugs zur strukturierten Ausgabe von Quellcode ("Pretty-Printer"). Dies gilt vor allem, wenn der Benutzer auf das Ausgabe-Layout Einfluß nehmen kann.

4.4.3 Debugger

Um das dynamische Verhalten eines Programms zu verfolgen (vgl. oben), bietet es sich an, Debugger zu verwenden. Diese Werkzeuge wurden zur Unterstützung des Tests von Programmen entwickelt. Sie erlauben es u.a., Datenwertveränderungen zu verfolgen, Unterbrechungsmarken (Breakpoints) zu setzen und Programme Anweisung für Anweisung auszuführen.

5 Fallbeispiele

5.1 DST Systems³⁴⁾

DST Systems in Kansas City, Missouri, ist eines der führenden Unternehmen in den Vereinigten Staaten für die Verwaltung von Wertpapierkonten und -fonds. 1986 stellte man in dem Unternehmen fest, daß die in den Anwendungssystemen festgelegten Wertebereiche den aktuellen Anforderungen nicht mehr gerecht wurden und die Abwicklung von Geschäften behinderten. DST konnte beispielsweise nur 99 Fonds pro System verwalten. Auch die Beschränkung von 99 Millionen Aktien pro Wertpapierdepot erwies sich als problematisch. Schließlich konnte das Speichervermögen von 999 Transaktionen für jedes Depot Wissen über verarbeitete Vorfälle nicht lange genug aufbewahren, da auf einer Reihe von Konten über 100 Transaktionen am Tag ausgeführt wurden.

Nach Untersuchung des Marktes kam DST zu dem Schluß, daß der Kauf von Standardsoftware als Handlungsalternative ausscheiden mußte, denn es war kein Softwarepaket verfügbar, das den Wünschen von DST entsprach. Die Kosten für eine Neuentwicklung wurden von

³⁴⁾ Vgl. Rochester, Douglass (1991), S. 1 ff.

DST mit \$50 Millionen und die Dauer des Entwicklungsprojekts mit drei Jahren veranschlagt. Da für einen Reengineering-Prozeß nur Kosten in Höhe von \$10 Millionen und ein Aufwand von weniger als einem Jahr geschätzt wurden, entschied sich DST für diese Alternative.

Das zu bearbeitende "Volumen" stellte sich wie folgt dar:

- 5000 Programme,
- 350 Dateien in Datenbanken oder als VSAM Dateien,
- 3000 temporäre Dateien,
- 2500 Reports,
- 800 Bildschirme sowie
- 3000 Datenstrukturen.

Zu beachten war, daß eine Erweiterung von Wertebereichen eine Änderung von Masken- und Report-Layouts erforderlich machen kann. Für die korrekte Ausführung von Operationen sind eventuell in der Folge Änderungen weiterer Wertebereiche notwendig. Selbst Standardalgorithmen müssen eventuell geändert werden; beispielsweise gehen Hashing-Algorithmen häufig von bestimmten Größen des zugrundeliegenden "Universums" aus, d.h. sind ebenfalls von Wertebereichen abhängig.

Für die Vorgehensweise beim Reengineering wurden zwei Ansätze diskutiert:

1. der Big-Bang-Ansatz, d.h. paralleles Reengineering aller betroffenen Systeme und Zusammenführung in einem Schritt;
2. der Trickle-down-Ansatz, d.h. schrittweises Überarbeiten und Einbinden der Systeme.

Die Vor- und Nachteile der beiden Alternativen lagen auf der Hand: Der Big-Bang-Ansatz erlaubt eine Beschleunigung des Reengineering-Prozesses, ist aber wesentlich risikoreicher als der Trickle-down-Ansatz. Da die mit dem System verbundenen Probleme sich immer drängender darstellten, entschied man sich für den Big-Bang-Ansatz. Als Werkzeug wurde das Tool Datatec (vgl. oben) ausgewählt.

Das Projekt Flex wurde im Juli 1989 gestartet und lief bis zum ersten Quartal 1990. Die Laufzeittests für das "neue" System konnten schnell erfolgreich abgeschlossen werden. Bis zum November 1990 wurde das alte System vollständig durch das neue ersetzt. Die Projektkosten betragen insgesamt \$12 Millionen; \$1 Million davon wurden für Synchronisierungsaufgaben verbraucht. Als vorteilhaft erwies sich im nachhinein, daß während des Reengineering-

Projekts das System - wenn auch zeitweise nur eingeschränkt - weiterbetrieben werden konnte: "This is a trick similar to changing the engines on a 747 while it is flying"³⁵⁾.

5.2 Rona-Dismat Gruppe³⁶⁾

Rona-Dismat ist ein Anbieter für Werkzeuge aller Art mit 560 Geschäften in der Provinz Quebec, Kanada. Innerhalb der Gruppe wurden seit 1975 5000 Cobol-Programme mit insgesamt zwei Millionen Lines of code entwickelt; allerdings waren einige Programme nie in Betrieb genommen worden. Da außerdem eine Vielzahl von Programmen ihre Aufgabe nicht mehr länger erfüllen konnte, wurde eine systematische Überprüfung und Überarbeitung der Programme durchgeführt. Dazu wurden die Programme vom Mainframe auf den PC transferiert und dort mit Hilfe der InterCASE-Workbench (vgl. oben) analysiert. Anschließend wurden nicht benutzte Programmteile gelöscht sowie laufzeitintensive Teile verbessert und dann die Programme auf den Mainframe zurückgespielt.

Durch das geschilderte Verfahren und den Einsatz von InterCASE für die Wartung konnte eine Produktivitätsverbesserung im Wartungsbereich von 25% bis 30% erreicht werden; die Anzahl der Programme konnte von 5000 auf 4500 gesenkt werden. Bei der Durchführung weitergehender Analysen erscheint Rona-Dismat eine Reduktion um weitere 500 Programme nicht ausgeschlossen.

6 Ausblick

Das Reengineering bildet grundsätzlich einen erfolgversprechenden Ansatz für die Bewältigung der Software-Altlasten-Problematik. Um die Potentiale, die der Ansatz beinhaltet, nutzen zu können, müssen weitere Konzepte und Methoden sowie unterstützende Werkzeuge entwickelt werden. Zu beachten ist allerdings, daß das Reengineering eines Systems wirtschaftlich sinnvoller als der Kauf von Standardsoftware/die Neuentwicklung sein kann, aber nicht sein muß. Teilweise kommt zum Beispiel, da Altsysteme häufig auf veralteten Hardware-/Basissoftware-Plattformen aufsetzen, zum eigentlichen Reengineering-Aufwand der Aufwand für eine Portierung hinzu. Auch kann man von einem Reengineering-Prozeß keine Wunder erwarten: Aus einem Programmcode können nur die Informationen gewonnen

³⁵⁾ Rochester, Douglass (1991), S. 3.

³⁶⁾ Vgl. Rochester, Douglass (1991), S. 9.

werden, die in ihm stecken. Folglich ist es unmöglich, aus einem Programm, das ohne ein eigentliches Entwurfskonzept in den Rechner eingegeben wurde, direkt einen wohlstrukturierten Entwurf ableiten. Dazu kommt, daß auch ein wohlstrukturierter Entwurf nicht in jedem Fall vollständig aus dem Programmcode rekonstruiert werden kann. Implizite Annahmen, implizit verwendete Namenskonventionen u.ä. sind zum Beispiel im Programmcode kaum zu ermitteln. Entsprechende Analyse-Tools können dabei stets nur eine Hilfestellung geben; wesentlich für das Reengineering sind letztlich die Fähigkeiten und die Erfahrung des zuständigen Reengineering-Programmierers.

Literatur

- Andresen, T: Nach dem Fehlverhalten bei der Assembler-Programmierung zum 4.-GL-Mißbrauch, in: Thurner, R. (Hrsg.), Reengineering - Ein integrales Wartungskonzept zum Schutz von Software-Investitionen, Strategie - Methoden - Werkzeuge, Hallbergmoos 1990, S. 85-100.
- Brooks, F.P.: No silver bullet: Essence and accidents of software engineering, IEEE Computer 20 (1987) 4, S. 10-19.
- CAP debis GEI: ESW - Existing Systems Workbench, Produktinformation 1992.
- Cleveland, L.: A program understanding support environment, IBM Systems Journal 28 (1989) 2, S. 324-344.
- Corbi, T.A.: Program Understanding: Challenge for the 1990s, IBM Systems Journal 28 (1989) 2, S. 294-306.
- Duckwitz, S.: Werkzeuge zur Restrukturierung und Nachdokumentation von Cobol-Programmen, in: Thurner, R. (Hrsg.), Reengineering - Ein integrales Wartungskonzept zum Schutz von Software-Investitionen, Strategie - Methoden - Werkzeuge, Hallbergmoos 1990, S. 137-154.
- Eicker, S., Kurbel, K., Pietsch, W., Rautenstrauch, C.: Einbindung von Software-Altlasten durch integrationsorientiertes Reengineering, Wirtschaftsinformatik 34 (1992) 2, S. 137-145.
- Fjeldstad, R.K., Hamlen W.T.: Application Program Maintenance Study, in: Proc. of GUIDE 48, Philadelphia 1979.
- Henselmann, G.M.: Portabilität als Ziel des Reengineering, in: Wagner, B., et al. (Hrsg.), Reverse Engineering, Sanierung, Dokumentation und Strukturierung vorhandener Software, Ehningen (bei Böblingen) 1992, S. 16-38.
- Hirsch, K.: Werkzeugunterstützte Restrukturierung von Cobol-Programmen, in: Wagner, B. et al. (Hrsg.), Reverse Engineering, Sanierung, Dokumentation und Strukturierung vorhandener Software, Ehningen (bei Böblingen) 1992, S. 63-95.
- Jones, C.: Reusability in Programming: A Survey of the State of the Art, IEEE Transactions on Software Engineering 10 (1984) 5, S. 488-493.

- Kurbel, K.: Reengineering - Verjüngungskur für Alt-Software, *Wirtschaftsspiegel* 47 (1992) 2, S. 32-36.
- Kurbel, K.: *Software Engineering im Produktionsbereich*, Wiesbaden 1983.
- Lahm, R.: Respezifikation vorhandener Software und Einsatz von CASE-Werkzeugen, in: Wagner, B., et al. (Hrsg.), *Reverse Engineering, Sanierung, Dokumentation und Strukturierung vorhandener Software*, Ehningen (bei Böblingen) 1992, S. 39-62.
- Lauber, R.: *EPOS - Kurzbeschreibung, Darstellung der wichtigsten Eigenschaften des Entwicklungs- und Projektmanagementorientierten Spezifikationssystems EPOS*, Oberhaching 1990.
- Lientz, B.P., Swanson, E.B.: *Software Maintenance Management, A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations*, London et al. 1980.
- National Bureau of Standards: *Guidance on Software Maintenance, Special Publication No. 500-106*, National Bureau of Standards, Washington D.C. 1983.
- Rochester, J.B., Douglass, D.P.: *Re-Engineering Existing Systems*, *I/S Analyser* 29 (1991) 10.
- Sneed, H.: *Softwarewartung und -wiederverwendung, Bd. II: Softwaresanierung (Reverse und Re-engineering)*, Köln 1992.
- Sneed, H., Kaposi, M.: *A Study on the Effect of Reengineering upon Software Maintainability*, in: *Proceedings of 6th International Conference on Software Maintenance*, San Diego, Nov. 1990.
- Standish, T.A.: *An Essay on Software Reuse*, *IEEE Transactions on Software Engineering*, 10 (1984) 5, S. 494-497.
- Turner, R.: *Beherrschung des Technologie-Wechsels mit Reengineering*, in: Turner, R. (Hrsg.), *Reengineering- Ein integrales Wartungskonzept zum Schutz von Software-Investitionen, Strategie - Methoden - Werkzeuge*, Hallbergmoos 1990, S. 11-20.
- Wagner, B: *Transformation in andere Programmiersprachen*, in: Wagner, B., et al. (Hrsg.), *Reverse Engineering, Sanierung, Dokumentation und Strukturierung vorhandener Software*, Ehningen (bei Böblingen) 1992a, S. 96-109.
- Wagner, B: *Dokumentation als Basis effizienter Softwarewartung*, in: Wagner, B., et al. (Hrsg.), *Reverse Engineering, Sanierung, Dokumentation und Strukturierung vorhandener Software*, Ehningen (bei Böblingen) 1992b, S. 152-170.

Arbeitsberichte des Instituts für Wirtschaftsinformatik

- Nr. 1 Bolte, Ch.; Kurbel, K.; Moazzami, M.; Pietsch, W.: Erfahrungen bei der Entwicklung eines Informationssystems auf RDBMS- und 4GL-Basis; Februar 1991.
- Nr. 2 Kurbel, K.: Das technologische Umfeld der Informationsverarbeitung - Ein subjektiver "State of the Art"-Report über Hardware, Software und Paradigmen; März 1991.
- Nr. 3 Kurbel, K.: CA-Techniken und CIM; Mai 1991.
- Nr. 4 Nietsch, M.; Nietsch, T.; Rautenstrauch, C.; Rinschede, M.; Siedentopf, J.: Anforderungen mittelständischer Industriebetriebe an einen elektronischen Leitstand - Ergebnisse einer Untersuchung bei zwölf Unternehmen; Juli 1991.
- Nr. 5 Becker, J.; Prischmann, M.: Konnektionistische Modelle - Grundlagen und Konzepte; September 1991.
- Nr. 6 Grob, H.L.: Ein produktivitätsorientierter Ansatz zur Evaluierung von Beratungserfolgen; September 1991.
- Nr. 7 Becker, J.: CIM und Logistik; Oktober 1991.
- Nr. 8 Burgholz, M.; Kurbel, K.; Nietsch, Th.; Rautenstrauch, C.: Erfahrungen bei der Entwicklung und Portierung eines elektronischen Leitstands; Januar 1992.
- Nr. 9 Becker, J.; Prischmann, M.: Anwendung konnektionistischer Systeme; Februar 1992.
- Nr. 10 Becker, J.: Computer Integrated Manufacturing aus Sicht der Betriebswirtschaftslehre und der Wirtschaftsinformatik; April 1992.
- Nr. 11 Kurbel, K., Dornhoff, P.: A System for Case-Based Effort Estimation for Software-Development Projects; Juli 1992.
- Nr. 12 Dornhoff, P.: Aufwandsplanung zur Unterstützung des Managements von Softwareentwicklungsprojekten; August 1992.
- Nr. 13 Eicker, S., Schnieder, T.: Reengineering; August 1992.