

Vorlesung Software-Reengineering

Prof. Dr. Rainer Koschke¹

¹Arbeitsgruppe Softwaretechnik
Fachbereich Mathematik und Informatik
Universität Bremen

Wintersemester 2005/06

Überblick I

① Einführung

1 Einführung

- Administrativa
- Lernziele
- Motivation
- Wichtige Begriffe
- Wartung
- Reverse Engineering
- Restrukturierung
- Reengineering
- Wrapping
- Business Process Reengineering
- Ziele und Aufgaben
- Unterschiede zur Vorwärtsentwicklung

Organisatorisches

- Vorlesung: montags, 10:15–11:45 Uhr, MZH 5210
- Vorlesung/Übung (alternierend): donnerstags, 8:15–9:45 Uhr, MZH 5210
- Erreichbar: OAS, Telefon 218-9671 koschke@tzi.de
- Sprechstunde nach Vereinbarung
- Literatur: Folien zur Vorlesung und verwendete Artikel
http://www.informatik.uni-bremen.de/st/lehredetails.php?id=&lehre_id=39
(Folien zur Vorlesung)
- Reengineering-Bibliographie:
<http://www.iste.uni-stuttgart.de/ps/reengineering/>

- Modulprüfung: mündliche Prüfung
- ansonsten (zählen jeweils gleich):
 - ① erfolgreiche Bearbeitung dreier praktischer Aufgaben
 - ② Fachgespräch (wie mündliche Prüfung)

Lernziele

- Grundlegende Begriffe
- Übersicht über die Gebiete des Reengineerings
- Abgrenzung zum traditionellen Software Engineering

Software-Evolution

- Gesetz des fortgesetzten Wandels
- Gesetz der ansteigenden Komplexität
- ...

⇒ ständige Anpassung erforderlich

⇒ Komplexität muss kontrolliert und begrenzt werden

Wunsch

- Gewählte Lösung antizipiert mögliche Änderungen.
- Änderungen werden auf der adäquaten Ebene vorgenommen.
- Dokumentation wird mitgeführt.

- Die Zukunft lässt sich nur begrenzt vorhersagen.
- Ursprüngliche Systemstruktur wird ignoriert.
- Dokumentation ist unvollständig oder obsolet.
- Mitarbeiter verlassen das Projekt (und mit ihnen verschwindet das ganze Wissen).

Legacy System

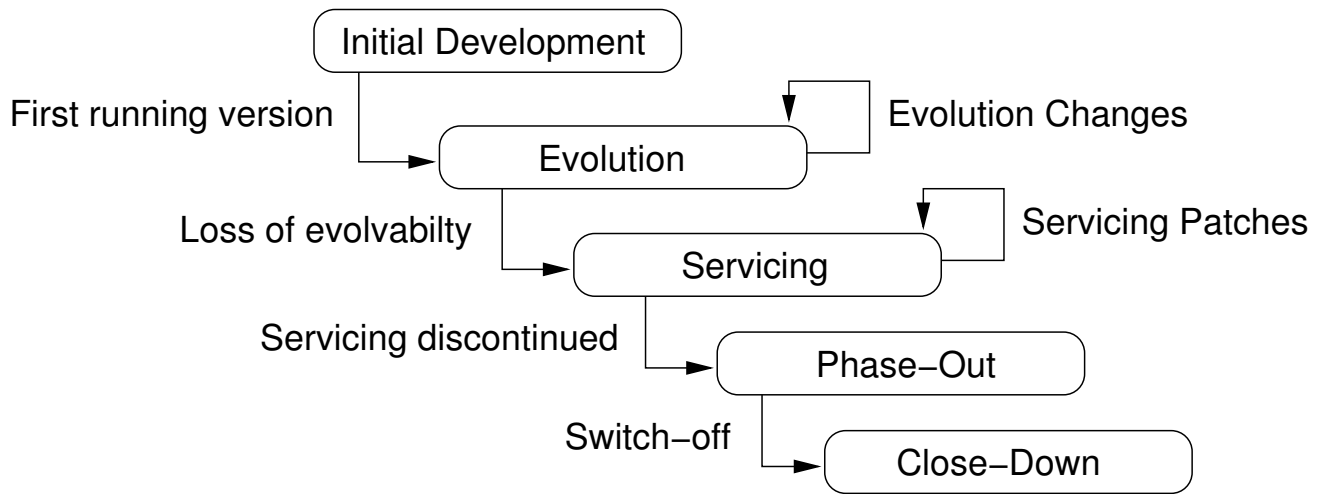
Legacy:

“A sum of money, or a specified article, given to another by will; anything handed down by an ancestor to a predecessor.”

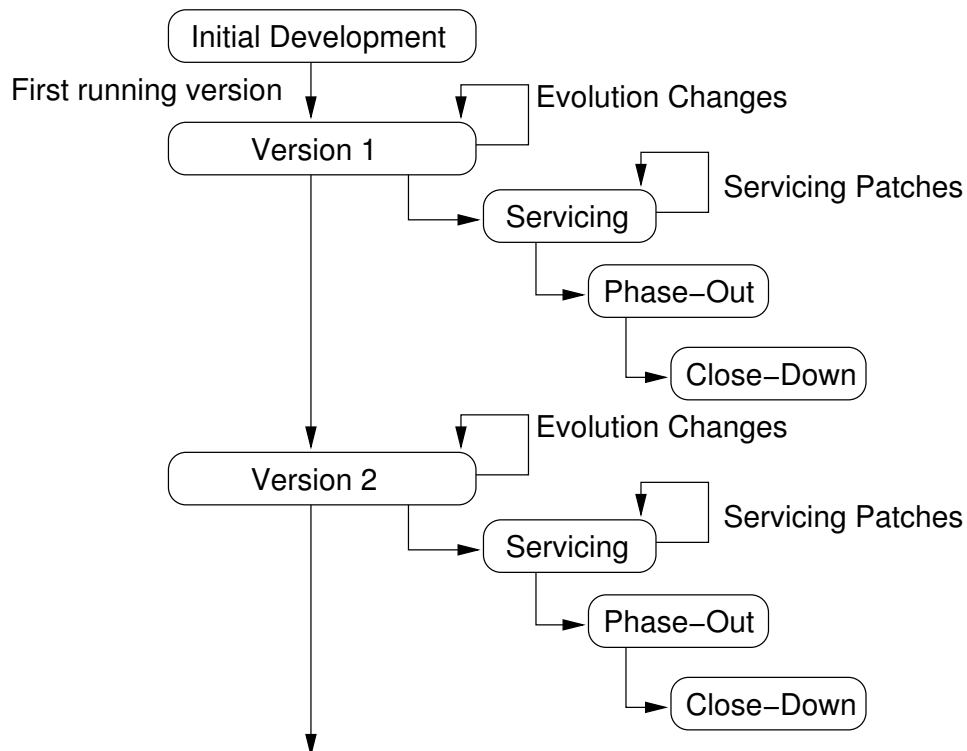
– Oxford English Dictionary

Legacy System: Software-System, das erbt wurde und einen Wert darstellt.

Staged Software Life Cycle Model



Versioned Staged Model



- Software-Wartung
- Software-Evolution
- Reengineering
- Software-Reengineering
- Business-Process-Reengineering
- Renovation
- Reclamation
- Reverse Engineering
- Restrukturierung
- Wrapping

Software-Wartung

ANSI/IEEE Standard 729-1983:

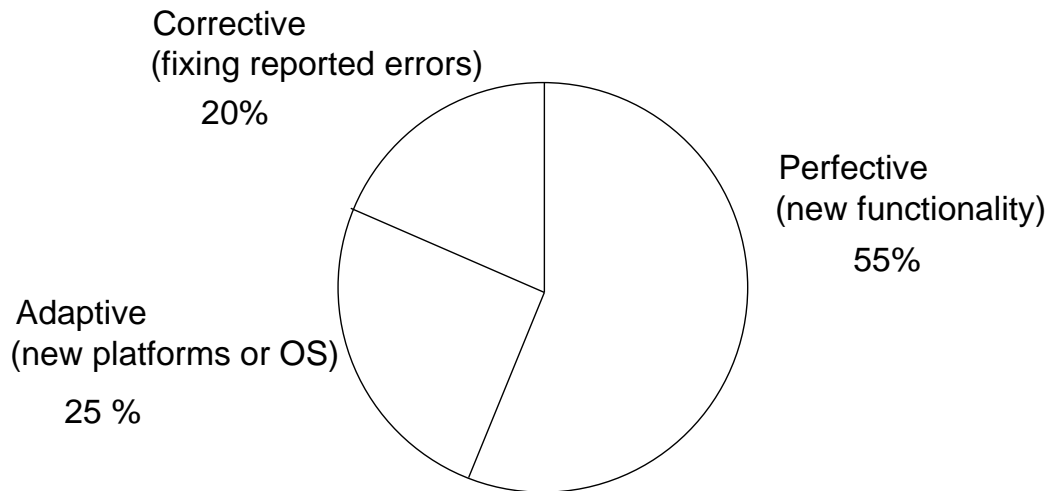
“Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment.”

Häufiger Sprachgebrauch: Änderungen am System nach dessen Auslieferung.

Schließt Anpassungen an neue Anforderungen ein.

Besserer Begriff hierfür: Software-Evolution.

Aufwand für Software-Wartung



– Lientz und Swanson (1980)

2005-10-19

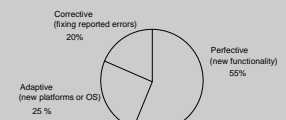
Vorlesung Software-Reengineering

└ Einführung

└─┬─ Wartung

└─┬─ Aufwand für Software-Wartung

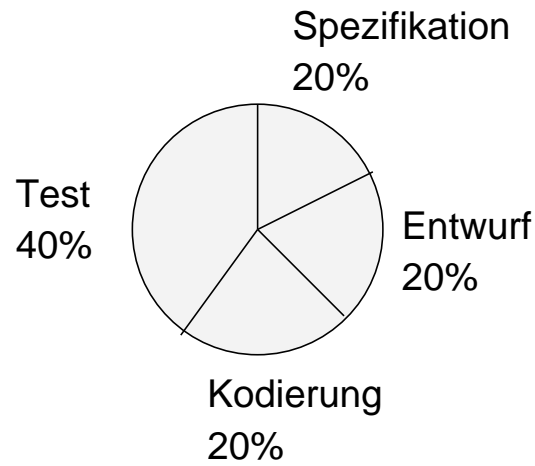
Aufwand für Software-Wartung



– Lientz und Swanson (1980)

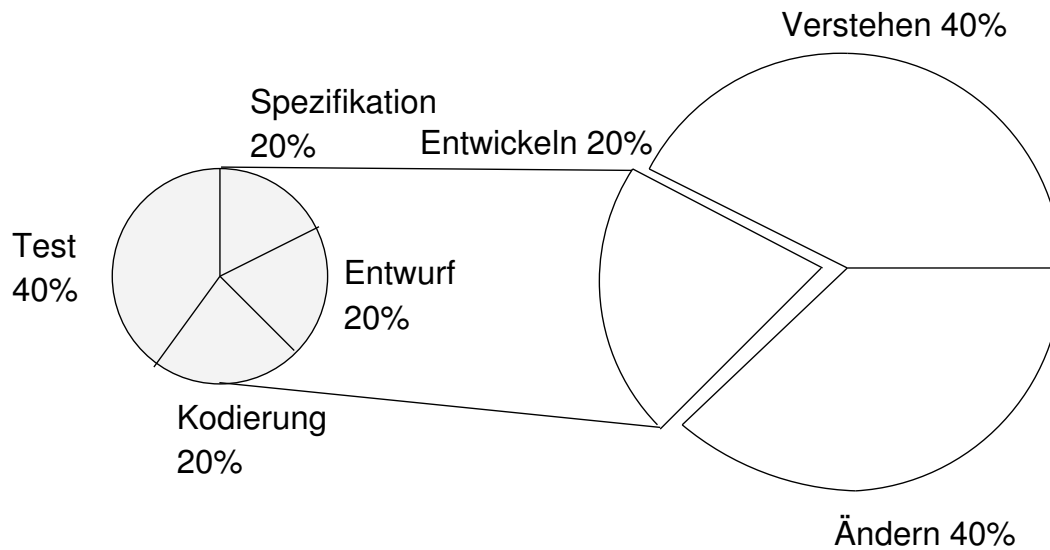
Lientz und Swanson (1980) haben den Aufwand für verschiedene Wartungsarten anhand von 487 Software-Organisationen näher untersucht und festgestellt, dass ca. 80% der so genannten Wartung tatsächlich Erweiterungen sind (neue Funktionalität bzw. Anpassungen an neue Hardware- oder Software-Plattformen).

Aufwand im Software-Lifecycle I



– Boehm (1981)

Aufwand im Software-Lifecycle II



– Fjedstad und Hamlen (1979)

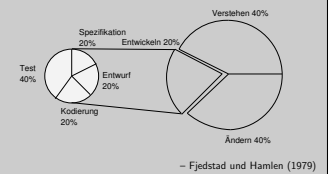
2005-10-19

Vorlesung Software-Reengineering

Einführung

Wartung

Aufwand im Software-Lifecycle



Eine typische Verteilung des Aufwands für Aktivitäten in der Erstentwicklung wurde von Boehm (1981) anhand groß angelegter empirischer Studien erhoben.

Der Aufwand für die Erstentwicklung ist jedoch vergleichsweise gering, wenn man ihn mit dem Aufwand für Wartung vergleicht. Arthur (1988) hat insgesamt sechs Untersuchungen aus den Siebzigern zum Anteil der Wartung am Software-Lifecycle zusammen getragen. Der Aufwand liegt in diesen Untersuchungen zwischen 60 und 80 Prozent. Die Garnter Group, eine große Unternehmensberatung, sagt für die Zukunft sogar einen ansteigenden Aufwand voraus, der bis zu 95% der Gesamtkosten für Software einnehmen wird (Moad 1990).

Fjedstad und Hamlen (1979) haben den Aufwand für die einzelnen Wartungsaktivitäten empirisch näher untersucht und dabei heraus gefunden, dass Wartungsprogrammierer ca. 50% ihrer Zeit allein mit der Analyse beschäftigt sind, bevor sie eine Änderung tatsächlich vornehmen und testen können. Bei korrektiver Wartung (also Fehlerbeseitigung) liegt der Aufwand für die Analyse gar bei 60%.

Aufwand für Software-Evolution

US Air Force System (Boehm, 1975):

- \$ 30 / Statement bei Erstentwicklung
- \$ 4000 / Statement in der Wartung

Beseitigung des Jahr-2000-Problems (geschätzt von Cassell, 1997)

500.000.000.000 - 1.000.000.000.000 DM

Reverse Engineering I

License Restrictions:

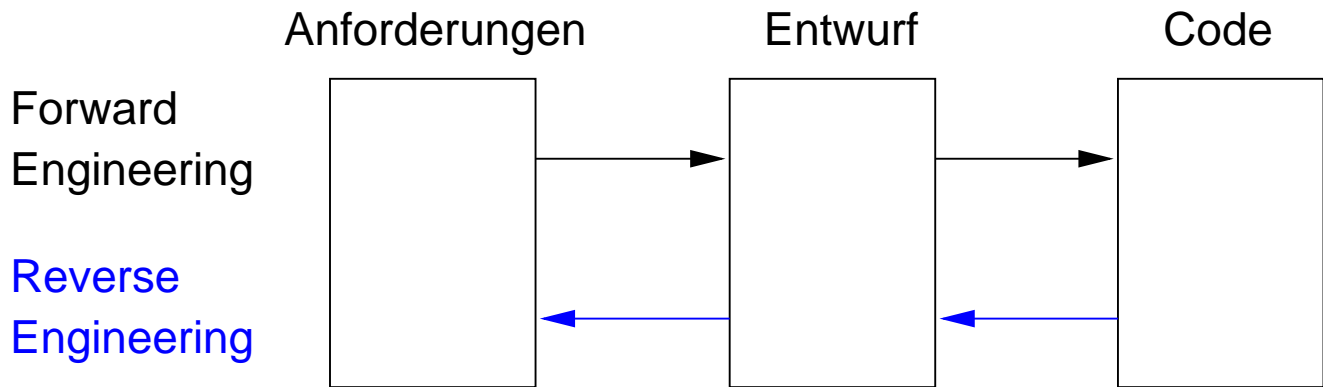
“Customer may not reverse engineer, disassemble, decompile, or translate the Software, or otherwise attempt to derive the source code of the Software.”

Reverse Engineering II

“To me the flow of time is irrelevant. You decide what you want. I then merely make sure that it has already happened.”

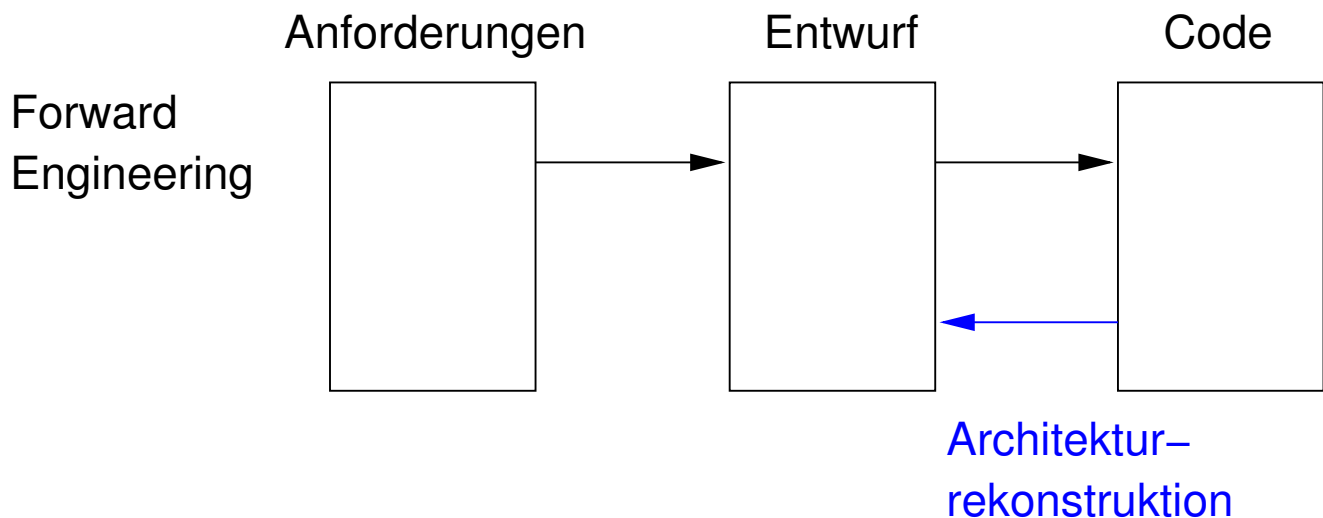
The Hitch Hiker’s Guide to the Galaxy

Identifikation der Systemkomponenten und deren Beziehungen.
Ziel ist die Beschreibungen des Systems in einer anderen Form oder auf höherem Abstraktionsniveau.



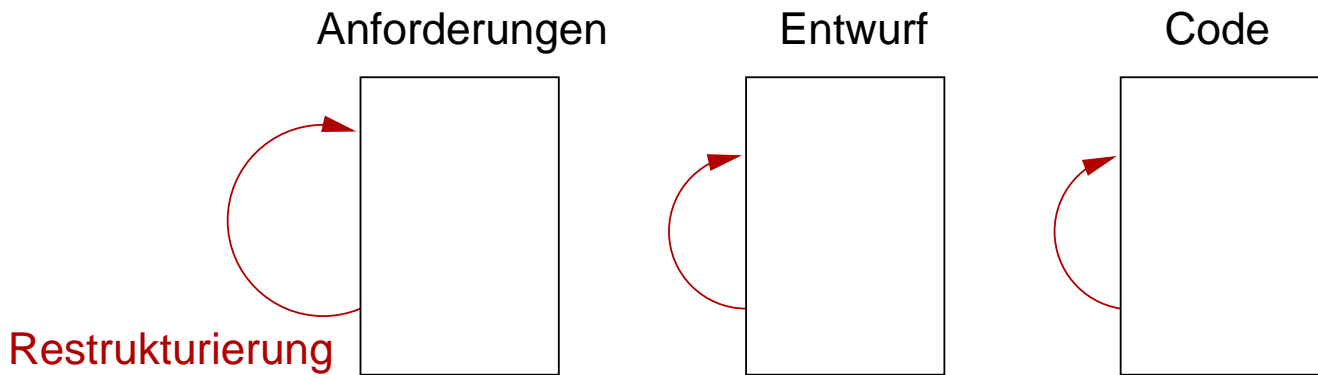
Architekturrekonstruktion

Reverse Engineering mit dem Ziel, eine Beschreibung der Architektur des Systems zu erstellen.



Restrukturierung (Chikofsky und Cross II. 1990)

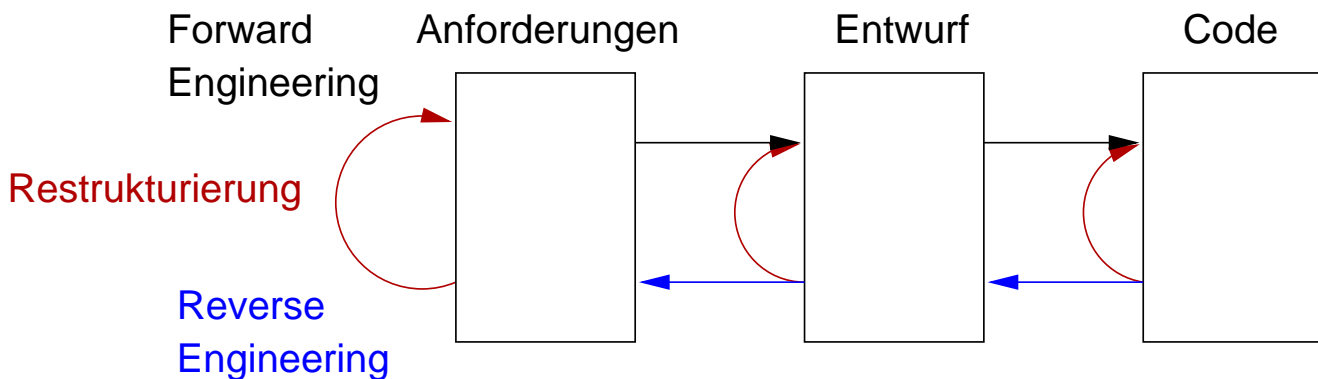
Transformation einer Repräsentation in eine andere auf derselben Abstraktionsebene, ohne Änderung der Funktionalität des Systems.



Reengineering (Chikofsky und Cross II. 1990)

Untersuchung (Reverse Engineering) und Änderung des Systems, um es in neuer Form zu implementieren.

Synonyme: Renovation, Reclamation.



Reines Reengineering:

- das System soll lediglich restrukturiert werden
- keine Funktionalität kommt hinzu / wird geändert

Erweiterndes Reengineering:

- System wird zunächst analysiert und/oder restrukturiert, um dann Funktionalität zu ändern oder hinzuzufügen

Wrapping

Das System erhält eine neue Schnittstelle, bleibt aber ansonsten unangetastet.

Interimslösung, wenn System bald ausgewechselt werden soll.

Notwendig, wenn das System Subsystem per Subsystem geändert werden muss.

Oft eingesetzt, um zeichenorientierte Anwendungen mit einer graphischen Benutzerschnittstelle zu versehen.

Organisatorische Gründe:

- “altes” Wartungspersonal behält Kontrolle über Wartung “ihres” Systems
- “junges” Wartungspersonal hat “moderne” Sicht

“Business process reengineering is the search for, and the implementation of, radical change in business process to achieve breakthrough results.”

– T.A. Stewart, Fortune Magazine’93.

Business Process Reengineering

Etwas sachlicher:

- Wiedergewinnung der tatsächlichen Abläufe der Geschäftsprozesse (Workflow) (z.B. Bestellweswesen, Auftragsabwicklung, etc.)
- Überarbeitung und Neudefinition der Abläufe

- Kontrolle der Komplexität
- Gewinnung alternativer Sichten
- Wiedergewinnung verlorener Information
- Erkennung von Seiteneffekten
- Schaffung höherer Abstraktionen
- Unterstützung von Wiederverwendung

Häufige Reengineering-Aufgaben

- Plattformanpassung
- Änderung der Programmiersprache
 - neuer Standard, neue Sprache, neues Paradigma
- Benutzerschnittstelle zeichenorientiert hin zu graphisch orientiert
- Mainframe hin zu Client-Server-Architektur
- Datenbankumstellung

Präventive Maßnahmen, wie z.B. Verbesserung des Information Hidings, Remodularisierung, sind eher selten.

Änderungen, die weite Teile des Codes bzw. sehr viele Systeme betreffen.

- Einführung des Euros
- Legacy to Internet Interoperability (Electronic Commerce)
- Änderung von Repräsentationsformen:
 - Y2K Problem
 - Erweiterung des Bar-Codes
 - Unix-Datum

Reengineering in der Praxis

Gegenwärtig zur Verfügung stehende Werkzeuge:

- grep
- symbolische Debugger
- Cross-Reference-Tools (fertige Parser, die Basisinformationen extrahieren; z.T. mit Visualisierung durch Graphen)
- programmierbare Analyseumgebungen basierend auf abstrakten Syntaxbäumen (z.B. Refine von Reasoning Systems, DMS von Semantic Designs, RainCode)

- Programmanalysen und -repräsentationen
- Program-Slicing
- Erkennung duplizierten Codes
- Refactoring und Transformationen
- Begriffsanalyse
- Analyse und Restrukturierung von Vererbungshierarchien
- Merkmalsuche
- Mustersuche
- Software-Visualisierung
- Reengineering-Projekte

Software Engineering

“Software Engineering is reengineering on the empty system.”

“Is it?”

Forward Engineering auf grüner Wiese

- Problem noch unklar
- Aussagen über Aufwand, Dauer, Zuverlässigkeit etc. sind schwierig
- System existiert nicht
 - Entwurf hat viele Freiheiten
 - im sauberen Entwurf gibt es keine versteckten Abhängigkeiten

Reengineering

- Problem weitgehend klar
- Idealerweise: Daten aus der Vergangenheit existieren, die Grundlage für Schätzungen darstellen
- System existiert
 - Genaue Struktur/Qualität bekannt?
 - Lösung ist durch existierendes System beschränkt
 - Änderungen können globale Auswirkungen haben (viele versteckte Abhängigkeiten)

Software Engineering & Reengineering

Reengineering beginnt oft bereits während der Erstentwicklung:

- neue Anforderungen treffen ein
- Missverständnisse und Unklarheiten werden sichtbar
- der Entwurf hat sich als unzureichend erwiesen
- Integration anderer Komponenten macht Umstrukturierungen notwendig

- Chikofsky und Cross II. (1990): "Reverse Engineering and Design Recovery: A Taxonomy", IEEE Software
definiert Terminologie; ist die begriffliche Grundlage
- Baumöl u. a. (1996): "Einordnung und Terminologie des Reengineering"
führt z.T. deutsche Begriffe ein

- 1 Arthur 1988** ARTHUR, L.J.: Software Evolution: The Software Maintenance Challenge. New York, NY : John Wiley & Sons, 1988
- 2 Baumöl u. a. 1996** BAUMÖL, Ulrike ; BORCHERS, Jens ; EICKER, Stefan ; HILDEBRAND, Knut ; JUNG, Reinhard ; LEHNER, Franz: Einordnung und Terminologie des Software Reengineering. In: Informatik Spektrum 19 (1996), S. 191–195
- 3 Boehm 1981** BOEHM, Barry: Software Engineering Economics. Englewood Cliffs, NJ : Prentice Hall, 1981
- 4 Chikofsky und Cross II. 1990** CHIKOFSKY, Elliot J. ; CROSS II., James H.: Reverse Engineering and Design Recovery: A Taxonomy. In: IEEE Software 7 (1990), Januar, Nr. 1, S. 13–17
- 5 Fjedstad und Hamlen 1979** FJEDSTAD, R.K. ; HAMLLEN, W.T.: Application Program Maintenance Study: Report to our Respondents. In: Proceedings of the GUIDE 48. Philadelphia, PA, 1979
- 6 Lientz und Swanson 1980** LIENTZ, B.P. ; SWANSON, E.B.: Software Maintenance Management. Reading, MA : Addison-Wesley, 1980

7 Moad 1990 MOAD, J.: Maintaining the Competitive Edge. In:
DATAMATION, 1990, S. 61–66