Software Reengineering in der Praxis

Harry M. Sneed
SoRing Kft. H-1221 Budapest
Technische Universität Dresden, Institut für Softwaretechnik
Harry.Sneed@T-Online.de

Abstrakt: Der Stand der Software Reengineering Praxis wird hier zusammengefasst. Anhand seiner Projekterfahrung schildert der Autor welche Techniken sich bewährt haben und welche nicht. Obwohl es hier und da Erfolge gab, ist die Schlussfolgerung eher ernüchternd. Wir sind immer noch weit entfernt von den ursprünglich hochgesteckten Zielen. Die Erfolge die es gegeben hat wurden zu einem hohen Preis bezahlt. Die Ergebnisse sind bescheidend. Der Automatisierungsgrad lässt immer noch zu wünschen übrig. Zum Schluss folgt ein Ausblick auf die Zukunft der Reengineering Technologien.

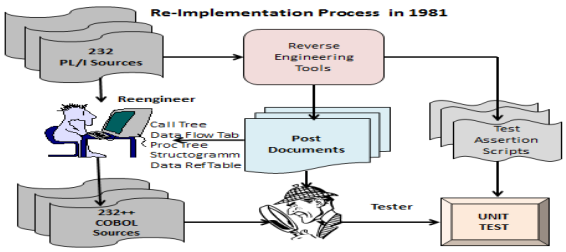
Schlüsselwörter: Reverse Engineering, Reengineering, Refactoring, Restructuring, Konversion, Retesting.

1 Die ursprünglichen Ziele

Reengineering als Teilgebiet von Software Engineering ist Anfang der 80er Jahre in den USA entstanden und zwar an der Universität Michigan. Professor Daniel Teichrow hat ein Forschungsprojekt für die Modellierung von Informationssystemen mit dem CASE Tool Set ISDOS. Das Ziel des Projektes war es lauffähigen Code aus einem Structured Analysis Model zu erzeugen. Einen der Projektmitarbeiter, nämlich Elliot Chikofsky kam auf die Idee, rückwärts zu gehen um aus dem Source Code ein Modell abzuleiten. Dies ist auch teilweise gelungen. Die Ergebnisse wurden veröffentlicht und die Industrie hat sie sofort aufgegriffen. In kurzer Zeit kamen zahlreiche Reverse und Re-Engineering Werkzeuge auf den Markt, Tools wie SuperStructure, ReCoder und Structured Programming Facility von IBM. In einem 1983 erschienen Buch von James Martin und Carma McClure „Software Maintenance-The Problem and its Solution“ wurden zwei Wartungsstrategien einander gegenübergestellt. Nach der einen Strategie lässt man den Code wie er ist und pflegt ihn in dem alten Zustand weiter. Nach der anderen Strategie saniert man den Code und Pflegt den neuen Zustand weiter. Die zweite Strategie verlangt eine gewisse Vorinvestition, sollte jedoch langfristig kostengünstiger sein. Also war vom Anfang die Reduzierung der Wartungskosten das Hauptziel. Andere Ziele waren die Migration der Software zu erleichtern und Software Komponente wieder zu verwenden – Reuse. Reengineering ist aus der Not geboren, bestehende Systeme wiederzuverwenden.

2 Das erste Pilotprojekt in Deutschland

Das erste große industrielle Reverse und Reengineering Projekt fand 1982 ausgerechnet in Deutschland statt. Die Bertelsmann AG hatte versucht mit einem großen Computerhersteller ein neues Versandsystem zu entwickeln. Nach zwei Jahren wurde das Projekt abgebrochen. Zurück blieb ein Haufen PLI Programme und ein veraltetes Datenbanksystem Die Firma des Autors wurde beauftragt den fertigen PLI Code zu sanieren und in COBOL zu migrieren. Der alte Code wurde zunächst automatisch nachdokumentiert und dann teils maschinell und teils manuell saniert und konvertiert. Das System wurde anschließend mit der Assertion-Technik gegen die alten Daten getestet. Nach anderthalb Jahr lief das neue COBOL System mit dem neuen Datenbanksystem. Dieses Projekt galt als Musterbeispiel für die Anwendung der neuen Reverse und Reengineering Technologie [Sned84].



3 Reverse vs. Re-Engineering

Vom Anfang an wurde zwischen Reverse und Re-Engineering unterschieden. In Reverse Engineering wird aus dem Code eine technische Dokumentation gewonnen. Der Code bleibt unangetastet. In Re-Engineering wird aus dem alten Code neuen Code erzeugt. Der neue Code soll leichter zu verstehen und leichter zu handhaben sein. Diese angestrebten Eigenschaften waren schon immer schwer zu messen. Ob ein Entwickler mit dem Code zurechtkommt, hängt weitestgehend vom Entwickler ab. Viele Reengineering Maßnahmen sind umstritten, sogar die Entfernung von GOTO Verzweigungen und die Refaktorierung zu tief verschachtelter Logik. Die Vision von Reengineering war immer die, einer Code-Wäscheautomat. Man schmeißt den alten dreckigen Code hinein, druckt auf einen Knopf und raus kommt einen sauberen, gebügelten Code und zwar fehlerfrei. Diese Vision ist zwar nie ganz erfüllt worden, aber einzelne Toolentwickler sind nah rangekommen.

4 Ein Automatisierungsansatz

Das Tool *CodeRedo* macht eine schrittweise Transformation des Codes.

- Im 1. Schritt wird der Code reformatiert.
- Im 2. Schritt werden unerwünschte Anweisungs- und Datentypen ersetzt.
- Im 3. Schritt werden nummerische Konstanten und Textliterale in Ressource Tabellen ausgelagert.
- Im 4. Schritt werden IO und DB Operationen in eine Datenzugriffsschale versetzt.
- Im 5. Schritt werden redundanter Codeblöcke – Clones – zusammengefasst.
- Im 6. Schritt wird der Code flach gebügelt – Else und GOTO Anweisungen werden entfernt
- Im 7. Schritt werden zu tief verschachtelte Codeblöcke ausfaktoriert.
- Im 8. Schritt werden Kommentare am Anfang jeder Funktion, bzw. Methode, eingefügt.
- Im 9. Schritt wird der Code in mehrere Module bzw. Klassen zerlegt.

Zum Schluss kommt ein Code heraus der schulbuchmäßig strukturiert ist. Der Reengineer hat sogar die Möglichkeit Datennamen auszutauschen. Der Reengineering Traum ist beinahe erfüllt. Dennoch wird das Ergebnis nicht ohne weiteres angenommen. Zu Unterschiedlich sind die Erwartungen was Codequalität anbetrifft.

Noch weiter auseinander liegen die Erwartungen bezüglich Reverse Engineering. Es fragt sich welche Modelle sich aus dem Code abzuziehen sind. Nicht alles was leicht zu gewinnen ist, ist nützlich und nicht alles was nützlich wäre, z.B. die Geschäftsregel, ist leicht zu gewinnen. Das Haupthindernis zur Verständigung fremder Programme ist die Benennung der Daten und Prozeduren. Die alten Namen bleiben in den neuen Modellen. Reverse Engineering wird meistens im Zusammenhang mit Re-Implementierung verwendet. Erst wird ein Modell aus dem Code abgeleitet und nachgebessert. Wenn das neue Modell steht wird daraus neuen Code generiert. Rich und Waters bezeichneten diesen Ansatz als „Abstraction and Reimplementation“ im Gegensatz zum Transliteration Ansatz, wonach der alte Code direkt in den neuen Code umgesetzt wird. Abstraktion setzt eine Entwurfssprache voraus um das Modell zu beschreiben. Das hat zu Folge, das der Reengineer neben der Programmiersprache auch noch eine Entwurfssprache lernen muss. In den 80er Jahren sind mehrere solcher Entwurfssprachen entstanden, z.B. PDL, Z, WSL und AST.

Das *Softredoc* Tool des Autors versucht möglichst vielen dieser Erwartungen entgegenzukommen. Folgende Diagramme werden erstellt

- einen Aufrufsbaum der Module, bzw. einen Vererbungsbaum der Klassen
- einen Prozedurbaum, bzw. Methodenbaum
- einen Datenflussdiagramm auf Prozedur-, bzw. Methodenebene
- einen Entscheidungslogikbaum
- eine Datenverwendungstabelle
- eine Testfalltabelle und
- ein Verzeichnis der Geschäftsregel

Zuletzt wird eine Entity/Relationship Repository aufgebaut mit der, der durch die Architektur navigieren kann. So wird jeden Informationswünsch befriedigt.

5 Sprachkonversion

Für die meisten Anwender ist die Konversion ihrer alten Programme in eine neue Programmiersprache das ultimatives Ziel. Notfalls kann er mit dem alten Code endlos weiter leben, solange wie die Technologie mit dem der alte Code implementiert ist weiter unterstützt wird und so lange er noch Personal findet, die sich mit der alten Technologie auskennen. Kritisch wird es allerdings wenn die Legacy Technologie nicht mehr unterstützt wird oder wenn er kein Personal mit den erforderlichen Legacy-Kennnisse findet. Ebenso kritisch wird es wenn die alte Technologie mit der neuen Umgebung nicht mehr kompatibel ist. Der Anwender ist gezwungen zu migrieren. Er muss die alte Sprache in eine neuere übersetzen.

Die einfachste Art der Übersetzung ist die Transliteration, die 1:1 Übersetzung von der einen Sprache in die Andere. Dies kann manuell oder maschinell erfolgen. Eine manuelle Transformation ist ähnlich einer manuellen Reengineering, nur wird der Code nicht nur saniert sondern auch gleich übersetzt. Angesichts der hohen Kosten dieser Arbeit werden solche Arbeitsintensive Projekte meistens in billig-Lohn Länder ausgeführt. Wer nicht outsourcen will muss Werkzeuge haben um den Code automatisch zu transformieren. Die Codequalität bleibt ein Problem. Der automatisch konvertierte Code ist selten so wie der Anwender ihn gerne hätte. Man muss ihn am Ende doch noch manuell nachbessern, auch wenn er formal korrekt und lauffähig ist. Das künftige Wartungspersonal ist damit selten zufrieden. Also gibt es keinen Königsweg zu einem neuen System. Aus einem Esel wird kein Rennpferd egal was einer anstellt. Das hat 40 Jahre Reengineering nachgewiesen.

6 Literaturhinweise

[ChiCros90] Chikofsky, E./ Cross, J.: “Reverse Engineering and Design Recovery”, IEEE Software, Jan. 1990, p. 13-18

[McCl83] McClure, C., Martin, J.: Software Maintenance – the Problem and its Solution, Prentice-Hall, 1983

[RiWa88] Rich, C., Waters, R.: “A Research Overview of Rengineering”, IEEE Software, July, 1988, p. 11

[Sned84] Sneed, H.: “Software Renewal – A Case Study”, IEEE Software Magazine, July 1984, p. 56

[Sned90] Sneed, H.: Softwaresanierung, Rudolf Müller Verlag, Köln, 1991

[Sned99] Sneed, H.M.: Objektorientierte Softwaremigration. Addison-Wesley, Bonn, 1999.

[SnWo10] Sneed, H., Wolf, E.: Softwaremigration in der Praxis, dpunkt.verlag, Heidelberg, 2010