Reverse Requirement Engineering

Harry M. Sneed & Wolfgang Prentner
Ziviltechnik Prentner Wien
Harry.Sneed@T-Online.de

Abstrakt: Software ist mehr als nur Code. Laut Harlan Mills besteht Software aus verschiedenen Elementtypen wie Source-Code-Dateien, Maschinencode-Komponenten, Steuerungsprozeduren, Testfällen und Dokumenten [Mills80]. Die Dokumente, e.g. Anforderungsdokumente, Entwurfsdokumente, Benutzungsdokumente, usw., gehören ebenso zum Softwareprodukt wie der Code selbst. Vor allem spielen die Anforderungsdokumente eine wichtige Rolle. Bei der Evolution des Softwareproduktes müssten sie neben dem Code fortgeschrieben werden. Sonst driften Code und Dokumentation immer weiter auseinander. Die Dokumentation wird immer chaotischer und schwieriger zu verstehen. Wie beim Code wird sie sanierungsreif.

Diese Sanierung der Anforderungs-dokumentation wird als „Requirement-Reengineering" bezeichnet. Sie kann nur hilfsweise automatisiert werden. Auf die „Requirement-Reengineering" folgt die „Requirement-Reverse-Engineering" bzw. die Umsetzung der Prosa-Anforderungen in ein Anforderungsmodell. In dem folgenden Beitrag wird ein Verfahren geschildert um die Dokumentation zu re-strukturieren und dem Code wieder anzupassen.

Schlüsselwörter: Reverse Engineering, Reengineering, Requirement-Dokumentation, Anforderungsspezifikation, Anforderungssanierung, Anforderungsmodellierung,.

1. Der Bedarf für Requirement Reengineering

Wie auch der Code werden die Anforderungsdokumente durch die ständigen Veränderungen alt und brüchig. Anderseits ist es wichtig sie aktuell zu halten. Sie werden gebraucht, um weitere Aufwände zu schätzen, um Code-änderungen zu spezifizieren, um den Fortschrift der Evolution zu verfolgen und als Basis für den modell-basierten Test [Jackson82]. Eigentlich müssten die Anforderungsdokumente im Gleichschritt mit dem Code fortgeschrieben werden. Wenn schon die Anforderungen in einem separaten Dokument in einer anderen Sprache als die des Codes gepflegt werden, dann sollte es jederzeit möglich sein, sie mit dem Code im Sinne eines Soll/Ist Vergleichs abzugleichen. Der Code ist hier der „Ist" die Dokumentation das „Soll".

Da es unmöglich ist, alle Anforderungen im Voraus zu erkennen, wird es immer Änderungsanträge geben. Änderungsanträge sind nichts anderes als zusätzliche Anforderungen. Ein neues Release ist eine Auswahl anstehender Anforderungen, die in der zur Verfügung stehender Zeit für das Release hineinpassen. Es ist immer nur ein Ausschnitt aus der gesamten Anforderungsdokumentation.

Umso wichtiger ist es, das die Anforderungs-dokumentation stets auf dem gleichen Stand ist wie der Code – sowohl quantitativ als auch qualitativ, denn auch die Qualität der Dokumentation soll erhalten bleiben. Das spricht dafür, dass die Anforderungsdokumente regelmäßig überarbeitet bzw. „reengineered" werden. Je nach Änderungshäufigkeit könnte der Requirement-Reengineering-Prozess jeden Monat, alle drei Monate, oder alle Jahre wieder stattfinden [ChiCros90].

2. Der Requirement Reengineering Prozess

Der Requirement-Reengineering vollzieht sich in drei Schritte:

- Im ersten Schritt wird das aktuelle Requirement-Dokument gemessen
- im zweiten Schritt wird das Dokument restrukturiert
- im dritten Schritt wird das Dokument markiert.

2.1 Erste Messung des Dokuments

Der erste Schritt erfolgt voll automatisch. Hier werden nicht nur die Anforderungen, sondern auch die Seiten, die Zeilen, die Abbildungen, die Abschnitte und alle erkennbare Anforderungsentitäten gezählt. Mit dieser ersten Messung gewinnt der Requirement-Engineer einen Überblick über den Umfang des Dokuments. Damit kann er den Aufwand für das „Reengineering" grob abschätzen.

2.2 Restrukturierung des Dokuments

Der zweite Schritt erfolgt manuell durch den Requirement-Engineer. Die Mehrzahl der heutigen Anforderungsdokumente sind monolithische Fließtexte ohne systematische Untergliederung. z.B. User Stories. Die Autoren dieser Texte haben einfach eine Menge Anwenderwünsche zusammengetragen ohne Rücksicht auf deren Anordnung, oder der Autor erzählt eine Geschichte darüber, wie er sich die Nutzung des

geplanten Systems vorstellt. Falls der Autor die Wünsche der Anwender registriert, werden diese normalerweise in nummerierten Abschnitten und Unterabschnitten aufgeteilt. Die Abschnittstexte werden oft durch Tabellen und Abbildungen ergänzt. Meistens beginnen die Dokumente mit der Zielsetzung. Aus den Zielen werden die Funktionen abgeleitet. In dem ersten Dokument, dem Lastenheft, wird das „was“ behandelt, i.e. was will der Benutzer eigentlich haben. In dem zweiten Dokument, i.e. dem Pflichtenheft, wird das „wie“ beschrieben. Wie gedenkt die ausführende Partei die Wünsche des Anwenders umzusetzen [Broy05]. Falls es bereits automatisierte Prozesse und elektronisch gespeicherte Daten gibt, wird auf diese hingewiesen.

Neugliederung des Dokuments: Die erste Aufgabe der Restrukturierung ist daher eine passende Gliederung der Anforderungstexte zu finden und die Texte durch „cut and paste“ anzupassen.

Zerlegung der Textabschnitte: Die Textabschnitte in einem herkömmlichen Anforderungsdokument sind unterschiedlich groß. Manche sind zu klein, andere zu groß geraten. Es hängt vom Inhalt ab. Bei der Restrukturierung sollten die zu kleinen zusammengelegt und die zu großen aufgeteilt werden. Die maximale Absatzgröße soll der Analytiker selber festsetzen. Auf keinen Fall darf ein Textabschnitt länger als eine Seite sein, d.h. maximal 25 Zeilen.

Neuordnung der Abschnittsreihenfolge: Was die Reihenfolge der Textabschnitte anbetrifft, empfiehlt es sich mit dem Kontext der Applikation zu beginnen und dann zu den Zielen des Projektes überzugehen. Auf die Ziele des Projektes sollten die für die Zielerfüllung erforderlichen Geschäftsprozesse und die von diesen Geschäftsprozessen benutzten Geschäftsobjekten und Geschäftsschnittstellen folgen. Über die Geschäftsobjekte und Geschäftsprozesse kommt man zum eigentlichen Produkt und über die Benutzeranforderungen an das Projekt. Eine gute Praxis ist die Datenwelt zuerst darzustellen weil sie die Grundlage für die Vorgänge ist. Nach dem Objekt- und Vorgangsmodell folgen die Geschäftsregeln. Hier zieht sich die Grenze vom allgemeinen Geschäftsmodell zu dem projektspezifischen IT-Modell. Nach den allgemeingültigen Regeln die für alle Vorgänge im Betrieb gelten, kommen die funktionalen Anforderungen. Das sind jene Funktionen, die nur durch dieses Projekt zu erfüllen sind.

Verweise auflösen: Sollte es Verweise von einer Anforderung auf eine andere geben, sollte der Anforderungstext, auf den verwiesen wird, in der verweisenden Anforderung hineinkopiert werden. Auf dieser Weise werden Abhängigkeiten zwischen Anforderungen abgebaut und die Lesbarkeit erhöht..

2.3 Markierung des Dokuments

Ein Entity/Relationship Modell, sowie von Prof. Peter Chen vorgeschlagen, besteht aus Entitäten und Beziehungen zwischen den Entitäten [Chen78]. Die Entitäten sind die Hauptwörter im Text. Die Beziehungen können Zeitwörter oder Eigenschaftswörter sein. Das Subjekt einer Anforderung tut etwas mit dem Objekt der Anforderung. Der Buchungsvorgang bucht den Geldbetrag. Buchungsvorgang und Geldbetrag sind die Entitäten, buchen ist das Prädikat, das sie verbindet.

In einem IT-Anforderungsdokument gibt es verschiedene Entitätentypen wie die, die oben aufgelistet wurden, manche physisch, anderswo abstrakt. Zur Bildung eines E/R Modells gehört die Auswahl der Entitäten, physische Entitäten wie z.B. Bildschirmoberflächen, Dateien, Nachrichten und Listen, sowie abstrakte Entitäten wie Prozesse, Anwendungsfälle, Auslöser und Funktionen. Datengruppen gehören ebenfalls zu den physischen Entitäten. Datenelemente einer Gruppe sind Attribute dieser Entitäten.

Ein Scanner-Tool durchsucht die Anforderungstexte nach physischen Entitäten, nämlich die Hauptwörtern. Jedes Hauptwort wird als Datenelement behandelt und in eine Hauptworttabelle eingetragen. Anschließend erfolgt eine manuelle Suche nach abstrakten Entitäten durch den Requirement-Engineer. Er versucht die vordefinierten Entitäten im Text zu erkennen. Wenn er eine findet wird das Kennwort für diese Entität am Anfang der Zeile, in der die Entität vorkommt, als Tag eingetragen. Sollten zwei oder mehrere Entitäten in der gleichen Zeile vorkommen, wird die Zeile dupliziert. Die Tags können vom Modellierer beliebig benannt werden, müssen jedoch konsequent verwendet werden. Diese Tätigkeit wird als „Document-Markup“ bezeichnet [Briand17].

Für das Anforderungsmodell einer elektronischen Gesundheitsakte wurden 32 Entitätentypen identifiziert. Jeder Entitätentyp wurde mit einem einmaligen Kennwort gekennzeichnet. Dieses Kennwort wurde am Anfang der Zeile, in der die Entität vorkommt, als Tag eingetragen. Wo zwei oder mehrere Entitäten in der gleichen Zeile vorgekommen sind, wurde die Zeile dupliziert. Die Tags konnten vom Modellierer beliebig benannt werden, müssten jedoch konsequent verwendet werden. Diese Tätigkeit wird als „Document Markup“ bezeichnet. Das Kennwort kann durch eine laufende Nummer ergänzt werden, um diese Ausprägung der

Entität von den anderen zu unterscheiden. Für die Zählung der Entitäten zwecks der Aufwandsschätzung ist dies nicht erforderlich, wohl aber für die Erkennung der Entitäten im Anforderungsmodell. Der Syntax einer Entitätsmarkierung ist:

<Entitättyp - [nr]> <Entitätbezeichner> <Entitättext>;
Als Beispiel einer Anforderungsmarkierung wird folgender Ausschnitt aus dem Anforderungsdokument der Austrian Gesundheitsakte herangezogen.

&FREQ_3_4 Beifügen_Nachhol_Impftermine
Auf Grundlage des gültigen Österreichischen Impfplanes muss bei der Abfrage des e-Impfpasses eines Teilnehmers das Datum der nächste(n) fälligen Impfungen und etwaige Nachhol-Impftermine beigefügt werden. Von GDA manuell eingefügte Impftermine müssen unterstützt werden und diese dürfen von der Fachlogik nicht überschrieben werden.

Die Markierung der Modellentitäten ist absichtlich einfach gehalten, damit der Modellierer möglichst viel Text in einer möglichst kurzen Zeit markieren kann. Es muss möglich sein, mindestens 100 Seiten Anforderungstext in einer Arbeitswoche zu re-strukturieren und zu markieren. Sonst wird das Ganze zu teuer. Deshalb wird hier auf jeglichen sprachlichen Schnick-Snack verzichtet. Der Modellierer sollen sich auf das notwendigste beschränken, nämlich ein Entitäten-Typ Kennzeichen und ein Entitäten-Bezeichner. Damit werden die Modellentitäten für den Text-Parser erkenntlich gemacht. Der Rest ist dem Textanalyse-Tool überlassen.

3. Der Requirement Reverse-Engineering-Prozess

In dem dritten Schritt des Requirement-Reengineering Prozess wird das Textdokument automatisch verarbeitet [Fermer16]. Die Hauptwörter werden auch ohne Markierung durch einen Text-scan erkannt und ausgewiesen. Sie bilden ein Hauptwortverzeichnis.

0172 07 NOUN Network
0173 21 NOUN Netzwerkkonfiguration
0174 12 NOUN Organisation
0176 22 NOUN Organisationseinheiten
0177 09 NOUN Parameter
0178 07 NOUN Partner
0179 09 NOUN Patienten
0180 14 NOUN Patientendaten
0181 11 NOUN Platzhalter

Der Benutzer hat jetzt die Möglichkeit, die irrelevanten Hauptwörter zu entfernen. Hauptwörter, die übrig bleiben, werden als systemrelevante Entitäten betrachtet. Es folgt ein automatischer Scan des Texts, Alle Sätze, die ein relevantes Hauptwort enthalten, kommen in eine Referenztabelle, wo die Einträge nach Hauptwort geordnet sind. Auf dieser Weise entsteht eine Querverweisliste der relevanten Hauptwörter. An jedem Hauptwort hängen die Sätze, in denen es verwendet wird.

Im gleichen Durchlauf werden alle Schlüsselwörter erkannt und in eine Schlüsselwort-Tabelle übertragen. Der Text, der auf ein Schlüsselwort folgt, wird bis zum nächsten Schlüsselwort bzw. bis zum Ende des betreffenden Absatzes herausgeschnitten und dem aktuellen Schlüsselwort zugewiesen. Dieser Vorgang ist gleich einer Zerschneidung des Textes. Es ist als ob man mit einer Schere den Text in vielen Schnipsel zerschneiden würde. Die Schnittstellen sind die Schlüsselwörter. Überall, wo ein Schlüsselwort steht wird der Text zerschnitten. Zum Schluss gibt es eine Menge Textausschnitte, einer für jede Nutzung eines Schlüsselwortes. Die Einzeltexte sind durch den Entitäten-Typ und den Entitäten-Bezeichner eindeutig identifiziert und können nach Entitäten-Typ, z.B. Datenobjekt oder Anwendungsfall – geordnet werden.
0001 07 ACT &Akteur
0009 05 CASE &Anwendungsfall
0010 01 DATA #
0011 05 GOAL &Ziel
0012 08 INPT &Eingabe
0014 07 OBJT &Objekt
0015 01 INTR @

Die zwei Ordnungsmerkmale – Hauptwörter und Schlüsselwörter – werden durch die ursprüngliche Zerlegung des Textes durch Titel und Untertitel ergänzt, so dass wir jetzt drei Sichten auf den Text haben:

- eine Sicht nach Hauptwörter bzw. Datenobjekte,
- eine Sicht nach Schlüsselwort bzw. Entitäten-Typ und
- eine Sicht nach Titel und Untertitel.

Diese übergeordneten Ordnungskriterien werden durch weitere untergeordnete Texttypen ergänzt. Zum Beispiel können Aufzählungen von Daten und Funktionen über ein Präfix erkannt werden und literale Texte sowie numerische Konstanten werden ohnehin erkannt. Es werden also Datenstrukturen, Datenvariabel und Datenkonstante erkannt und aus dem Text herausgeschnitten.

Die Bedingungen im Text werden ohnehin durch die Verwendung bestimmter Prädikate, z.B. „wenn“, „falls“, „solange“, „oder“, erkannt. Das Tool hat eine Tabelle solcher Bedingungswörter in deutscher Sprache. Wenn

eines erkannt wird, wird der Satz in dem es vorkommt als Bedingung markiert und kommt in eine Bedingungstabelle. Das Herausschälen der Bedingungen ist eine wichtige Voraussetzung für die Erkennung der Testfälle sowie für die Berechnung der logischen Komplexität der spezifizierten Aufgabe.

Das Endergebnis ist ein semiformales Anforderungsmodell, das vollautomatisch verarbeitet werden kann. Das Modell kann dazu dienen, die Entwicklungskosten zu schätzen und logische Testfälle für einen modellbasierten Test zu generieren [Sneed07]. Mit einem Modell ist es möglich zu prüfen, ob die Anforderungen formal erfüllt sind, auch ohne Test. Die funktionalen Anforderungen können nämlich auf die Code-Funktionen und die Testfälle verfolgt werden. Die Kosten dieser Nutzen sind relative niedrig. Wenn die Anforderungen nicht vom Anfang an ordentlich erfasst wurden, können sie später aus dem Dokumentationstext gewonnen werden. Der Analytiker muss nur bereit sein sich mit den Prosatext auseinander zu setzen. Der Reengineer muss kein Fachspezialist für das Thema der Anforderungen sein. Es genügt wenn er die Sprache versteht und die Regel der Modellierung kennt.

4. Erfahrungen mit diesem Verfahren

Die Nutzung der Requirement-Reengineering Technologie ist relative jung. Sie wurde bisher in drei Projekten verwendet um die Anforderungen nachträglich zu modellieren. Das erste Projekt war die Modellierung der Anforderungen an die Österreichische Sozialversicherung für ein neues Krankenversicherungssystem mit 229 funktionalen Anforderungen, 12.957 Hauptwörter, 949 Datenentitäten, 1.454 Geschäftsregel und 267 Anwendungsfällen. Mit dem Modell konnten 4.656 Function-Points, 30.392 Data-Points und 16.450 Object-Points gezählt werden. Das zweite Projekt war die Modellierung einer Erweiterung der österreichischen elektronischen Gesundheitsakte mit 331 Anforderungen, 4318 Hauptwörter, 1.104 Datenentitäten, 200 Geschäftsregel in 22 Anwendungsfälle. Mit dem Modell konnten 1.134 Function-Points, 4.371 Data-Points und 2.043 Object-Points gezählt werden. Das dritte Projekt war die Modellierung eines elektronischen Kassen-Abrechnungssystem für eine Lebensmittelkette mit 50 funktionalen Anforderungen, 2.285 Hauptwörter, 12 Datenentitäten, 68 Geschäftsregel und 6 Anwendungsfällen. Mit dem Modell konnten 456 Function-Points, 2.563 Data-Points und 800 Object-Points gezählt werden.

5. Zusammenfassung

Der Bedarf für Requirements Engineering ist definitiv vorhanden. IT-Anwender, die modell-basiert schätzen, testen, entwickeln und warten, brauchen Modelle [Selic2003]. Eine Möglichkeit Modelle zu konstruieren ist aus der vorhandenen Textdokumentation. Dafür ist Requirement-Reengineering gedacht. Ob der hier geschilderte Ansatz sich durchsetzt, kann nur die Zukunft zeigen. Einige Fragen sind noch zu klären, z.B. wie zwischen Datenobjekten und Datenelementen, wie zwischen Regeln und Anforderungen zu unterscheiden ist. Die Semantik der Anforderungssprache muss exakter definiert sein, um daraus sinnvolle Modelle abzuleiten. Bis dahin müssen wir uns mit fuzzy Modellen abfinden oder auf Modelle verzichten. Es fragt sich, was besser ist – ein Fuzzy Modell oder gar kein Modell.

6. Literaturhinweise

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

[BeRa00] Bennet, K., Rajlich, V. „Software Maintenance and Evolution – A staged Model" in Proc. On the Future of Software Eng., ICSE-2000, IEEE Press, Limmerick, 200, S. 73

[Briand17] Briand, L.: „Analyzing Natural Language Requirements – The not-too-sexy and yet curiously difficult Research that Industry needs", 23rd International Working Conference on Requrements Eng., Foundation for Software Quality (REFSQ 17), 2017

[Broy05] Broy, M./ Rausch, A.: „Das neue V-Modell-XT", Informatik Spektrum, Nr. 22, Juni 2005, S. 220

[Chen78] Chen, P.: "The Entity-Relationship Model – toward a unified view of Data", *ACM Trans. Database Syst*. Band 1, Nr. 1, März 1976, S. 9–36

[Fermer16] Fermer, H., Hauptmann, B., Widere, A.: „Automatische Unterstützung bei der Qualitätssicherung von Anforderungsdokumenten" Objektspektrum, Nr. 2/2016, p. 14

[Mills80] Mills, H.: „Principles of Software Engineering", IBM Systems Journal, Vol. 19, Nr. 4, 1980

[Jackson82] Jackson, M./McCracken, D.: "Life cycle Model considered harmful", SE-Notes, Vol. 7, No. 1, April 1982, p. 11

[Selic2003] Selic, B.: "The Pragmatics of Model-Driven Development", IEEE Software, Sept. 2003, p. 19

[Sneed2007] Sneed, H.: "Testing against natural language Requirements" *7th Int. Conference on Software Quality (QSIC2007)*, Portland, Oct. 2007