Dissertationen

Bernhard Deifel: **Requirements Engineering komplexer Standardsoftware**

Promotion: Technische Universität München, Fakultät für Informatik
Erstgutachter: Prof. Dr. M. Broy
Zweitgutachter: Prof. Dr. J. Schlichter
Datum der Prüfung: 31.07.2001
Veröffentlichung: http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2001/deifel.html

Kurzfassung:

Die vorliegende Arbeit stellt eine Methodik für das Requirements Engineering komplexer Standardsoftware vor. Diese Software besteht aus einer Vielzahl von Funktionalitäten und Variationen, wodurch sie als ein aus mehreren Softwareprodukten bestehendes System auf einem Markt verkauft wird. Die Methodik berücksichtigt die Wechselwirkung firmeneigener Ziele, wie Kunden-, Zeit- und Kostenorientierung, mit dem Markteinfluss. Dieser äußert sich in erheblichen Unsicherheiten, ständigen Veränderungen und vielen unterschiedlichen Anforderungen. Hauptmechanismen hierfür sind die Entwicklung in Variationen und Versionen.

Kern der Arbeit sind ein Modell von Entwicklungsprodukten und ein Prozessmodell des Requirements Engineerings komplexer Standardsoftware. Das Prozessmodell beschreibt ein Vorgehen für das Herausarbeiten, die Verhandlung und die Spezifikation von Anforderungen, während das Modell der Entwicklungsprodukte zu erarbeitende Ergebnisse und methodische Anleitungen dokumentiert.

Ein eigenes Modellierungskonzept verbessert die Verfolgbarkeit von Anforderungen und von Entscheidungen einzelner Entwicklungsaktivitäten sowie die schnelle Reaktionsfähigkeit auf sich ändernde Anforderungen. Entwicklungsschritte werden als Zuarbeiten zu Entscheindungssituationen modelliert, worin Entwicklungsprodukte genau festgelegte Aufgaben übernehmen. Dadurch wird die Erarbeitung und die Verwendung aller Entwicklungsergebnisse und Entscheidungen stets nachvollziehbar. Inhalte und Zusammenhänge zwischen Entwicklungsprodukten sind genau dokumentiert, was eine durchgängige Verfolgbarkeit von Anforderungen ermöglicht. Durch die spezielle Strukturierung des Prozessmodells und des Modells der Entwicklungsprodukte kann stufenweise von roh gesammelten Anforderungen hin zu einer Spezifikation übergegangen werden. Die Hauptstufen sind die Marktsicht, die Systemsicht und die Entwicklungssicht. Erstere ist kundenorientiert und hilft, Variationen zu identifizieren. Die zweite Sicht zielt auf eine Kompromissbildung von Marketing- und Entwicklungszielen ab und fokussiert auf die Anforderungsverteilung auf Softwareprodukte. Schließlich konzentriert sich letztere auf die kosten- und zeitoptimale Versionsplanung einer komplexen Standardsoftware.

Zur methodischen Unterstützung des Prozessmodells werden darüber hinaus eine Bewertungsmethode für Entscheidungssituationen sowie ein Konzept zur Spezifikation von Entwicklungsprodukten vorgeschlagen.

James J. Hunt: **Erweiterbare, sprachenbewusste Differenz und Verschmelzung (Extensible, Language Aware Differencing and Merging)**

Promotion: Fakultät für Informatik, Universität Karlsruhe (TH)
Erstgutachter: Prof. Dr. W. F. Tichy
Zweitgutachter: Prof. Dr. S. Abeck
Datum der Prüfung: 09.11.2001

Kurzfassung:

Die Entwicklungszyklen für Softwareprojekte werden immer kürzer. Obwohl es bereits viele Hilfsmittel gibt, die den Softwareentwicklungsprozess vereinfachen, sind an der überwiegenden Mehrheit von Softwareprojekten mehrere Software-Ingenieure beteiligt, die eng zusammen arbeiten. Trotz des Einsatzes vieler Techniken, die Komponenten eines Projekts unabhängig halten sollen, gibt es oft Fälle, in denen zwei Personen gleichzeitig auf den selben Dateien arbeiten müssen. Als Resultat hat man zwei Versionen dieser Dateien, deren Änderungen man zusammenführen will.

Die Standardmethode zur Verschmelzung zweier Dateien basiert auf Zeilen und ist daher eher primitiv. Zeilenbasierte Verschmelzung erkennt weder Information über die Struktur noch über die Semantik des Programms. Obwohl dieses Vorgehen in vielen Fällen ausreichend ist, ist es nicht fehlersicher. Es gibt echte Konflikte, die nicht gefunden werden (falsche Positive). Man findet damit auch Konflikte, die keine tatsächlichen Konflikte sind (falsche Negative). Um auf der sicheren Seite zu bleiben, muss ein Entwickler, der diese Methode anwendet, sehr vorsichtig mit Änderungen sein. Sogar kleine Formatierungsänderungen können zeilenbasierte Verschmelzungen unbrauchbar machen. Je komplexer und grösser das Projekt wird, um so schwieriger wird es, diese Beschränkungen einzuhalten.

Bisherige Verfahren, die eine Verbesserung erzielt haben, setzen Zerteiler ein, um mehr Wissen über die Änderungen zu gewinnen. Obwohl sie bessere Ergebnisse liefern, konnte sich keines in der Praxis etablieren. Diese Verfahren erfordern entweder einen speziellen Programmeditor, der alle Änderungen protokolliert oder funktionieren nur mit bestimmten, eigens für die Verschmelzung eingeschränkten Programmiersprachen.

Erweiterbare, sprachenbewusste Verschmelzung (**ELAM**) ist eine neue Methode, die die obengenannten Probleme zum größten Teil löst. Dadurch wird ein Kompromiss zwischen praktischer Einsetzbarkeit und vollständiger Semantikanalyse hergestellt. Diese Methode hat einige Vorteile:

- Sie benötigt keinen speziellen Programmeditor.
- Sie berücksichtigt Kommentare.
- Sie ist leicht an eine große Klasse der Programmiersprachen anpassbar.
- Sie ist auch auf Dokumentsprachen wie HTML und TeX anwendbar.
- Sie hat einen generischen, wortbasierten Zerteiler für nicht unterstützte Sprachen, der immer noch besser als ein zeilenbasiertes Verfahren ist.
- Sie kann mehrere Dateien zusammen betrachten.
- Sie kann Namenskonflikte auch über Dateigrenzen hinaus erkennen und auflösen.

Obwohl nicht alle Verschmelzungskonflikte damit automatisch gelöst werden, ist sie eine deutliche Verbesserung gegenüber der Standardpraxis.

Wie fast alle anderen Verschmelzungsmethoden ist die erweiterbare, sprachenbewusste Verschmelzung ein Drei-Wege-Vergleich. Sie benötigt einen gemeinsamen Vorgänger, um zwei davon abgeleitete Varianten zu verschmelzen. Sie besteht aus vier Komponenten: einem Zerteiler, einem Differenzalgorith-

mus, einem Umbenennungsanalysierer und einem Differenzverschmelzer.

Der Zerteiler basiert auf dem Zerteilergenerator ANTLR[1]. ANTLR ermöglicht die Verwendung einer eigenen Regelmenge für Kommentare, die verschieden von der Regelmenge der Hauptsprache ist. Das bedeutet, dass das System leicht Programm und Kommentar auseinander halten und getrennt betrachten kann. Die verwendeten Regeln behalten auch alle Trenn- und Gruppierungszeichen bei, wodurch die Differenzberechnung vereinfacht wird.

Die Differenzberechnung verwendet ein neues Verfahren—einer der Hauptbestandteile dieser Doktorarbeit. Typischerweise werden für den Vergleich zweier Zerteilungsbäume Baumdifferenzalgorithmen eingesetzt. Die bekannten Baumdifferenzalgorithmen bauen auf dem LCS-Algorithmus (Longest Common Subsequence) auf. LCS wird auch für zeilenbasierte Differenzberechnungen verwendet, hat aber einen Aufwand von $O(n^2)$. Baumdifferenzalgorithmen haben sogar einen noch höheren Aufwand. In **ELAM** werden die Lexemsequenzen mit einem Suffixbaum indiziert. Damit werden alle zusammenpassenden Blöcke zwischen zwei Revisionen einer Datei gefunden. Die Bestimmung übereinstimmender Lexemsequenzen geschieht in linearer Zeit. Danach wird eine dynamische Programmierungsverfahren verwendet, um die längste aufsteigende Lexemsequenz zu finden. Ein letzter Schritt passt das Differenzergebnis an die syntaktische Struktur der Eingabe an. Obwohl dieser zweite Teil im schlimmsten Fall ebenfalls quadratischen Aufwand hat, ist dieser Aufwand quadratisch zur Anzahl der zusammenpassenden Blöcke anstatt zur Anzahl der Lexeme. Im typischen Fall ist der Aufwand eher linear.

Der Umbenennungsanalysierer erzeugt eine Symboltabelle. Anhand dieser Tabelle und des Resultats der Differenzberechnung werden Namens- und Signaturunterschiede zweier Revisionen derselben Dateimenge erkannt. Dadurch kann das System Umbenennungen zwischen den Revisionen erkennen. Drei verschiedene Informationen werden betrachtet: die Namensdeklaration, die Namensverwendungen und die Implementierung der Namensdeklaration, z.B. den Methodenrumpf. Bei dieser Analyse wird ein kleines Regelsystem eingesetzt, um Sprachanpassungen zu erleichtern. Das Resultat ist eine Umbenennungsliste, die für die Verschmelzung angewendet werden kann.

Die Differenzverschmelzung ähnelt dem Kernalgorithmus der zeilenbasierten Verschmelzung. Dieser wurde so erweitert, dass er die Informationen zur Konfliktfindung aus allen anderen Teilen des Systems verwendet, um Konflikte nach Möglichkeit aufzulösen. Durch globale Namens- und Signaturenanalyse können Konflikte in Namens- und Signaturenänderungen erkannt und gelöst werden. Zuweisungsanalyse wird auch verwendet, um eine präzisere Konflikterkennung zu ermöglichen.

Das Resultat ist ein praktisch einsetzbarer Verschmelzer, der erweiterbar ist und dateiübergreifend arbeitet. Der Systemkern ist schon mit Anpassungsmodulen für Java und Scheme ausgestattet. Um das Verschmelzungsverfahren zu testen, wird es in ein Revisionsverwaltungssystem eingebaut.

[1] Terence Parr: www.antlr.org

Alexander Knapp: **A Formal Approach to Object-Oriented Software Engineering**

Promotion: Ludwig-Maximilians-Universität München
Erstgutachter: Prof. Dr. Martin Wirsing (Ludwig-Maximilians-Universität München)
Zweitgutachter: José Meseguer, PhD (SRI International, California)
Datum der Prüfung: 20.07.2000
Veröffentlichung: Alexander Knapp. A Formal Approach to Object-Oriented Software Engineering. Shaker, Aachen, 2001

Kurzfassung:

Formale Softwareentwicklungsmethoden ergänzen die traditionellen Entwicklungsschritte und deren informelle oder semi-formale Modelle um verbindende Beweisverpflichtungen auf der Grundlage mathematischer Modelle. Dabei stellen die sog. offenen Ansätze die korrekte Softwareentwicklung und die verwendeten Formalismen eher heraus, die sog. verdeckten Ansätze lassen während der Entwicklung die formale Fundierung weiter in den Hintergrund treten.

Die Methode „fOOSE“ (Formal Object-Oriented Software Engineering) exemplifiziert eine formale Softwareentwicklungsmethode für einen integrativen Ansatz: Es wird mit einer gängigen, aber in ihren verwendeten Teilen semantisch gut untersuchten Entwicklungsnotation gearbeitet, diese diagrammatische Darstellung aber durch Anmerkungen, etwa Invarianten, in einer formalen, algebraischen Sprache optional ergänzt. Aus der Semantik der Diagramme und der Anmerkungen kann halb-automatisch eine algebraische, ablauffähige Spezifikation abgeleitet werden: Ein automatisch erstelltes Spezifikationsgerüst muß vom Benutzer um Detailinformationen ergänzt werden. Diese Spezifikationen unterstützen einerseits das Rapid prototyping und erlauben andererseits frühzeitige formale Verifikation. Schließlich bildet die semi-automatisch erstellte Spezifikation aus einem gegebenen, annotierten Softwaremodell auch die Grundlage für eine korrekte Implementierung dieses Modells.

Auf der technischen Seite werden für fOOSE folgende Untersuchungen angestellt: Als Softwareentwicklungssprache wurde die „UML“ (Unified Modeling Language) als der gängige Standard ausgewählt, als formale Anmerkungssprache die algebraische, objektorientierte Spezifikationssprache Maude, als Implementierungssprache die objektorientierte, nebenläufige Programmiersprache Java. Es wird ein geeigneter Ausschnitt der UML zusammengestellt (Use cases, Static structures, Collaborations) und semantisch analysiert. Ebenso wird eine mächtige Teilsprache von Maude (Object theories) näher studiert und als tragfähige Zielspezifikationssprache für die gewählte Teilsprache von UML erwiesen. Schließlich wird eine ereignisorientierte, strukturelle operationale Semantik von Java vorgestellt und die Nebenläufigkeitsstrukturen von Java eingehend betrachtet. Basierend auf den Semantiken von UML, Maude und Java werden Übersetzungen von UML Collaborations nach Maude und nach Java angegeben und die Maude-Übersetzung als korrekt bezüglich der UML-Semantik, die Java-Übersetzung als korrekt bezüglich der Maude-Übersetzung erwiesen.

Andreas Rausch: **Componentware: Methodik des evolutionären Architekturentwurfs**

Promotion: Institut für Informatik, Technische Universität München
Erstgutachter: Prof. Dr. Manfred Broy
Zweitgutachter: Hon.-Prof. Dr. Ernst Denert
Datum der Prüfung: 12.11.2001

Kurzfassung:

Heutige Softwaresysteme sind äußerst komplex. Erfahrungsgemäß können sie von einzelnen Softwareingenieuren kaum mehr vollständig erfasst werden. Die Entwicklung eines Systems findet meist in einem hochgradig dynamischen Umfeld statt. Der iterative, evolutionäre Entwurf einer Softwarearchitektur und frühes Prototyping können helfen, Risiken der Softwareentwicklung entscheidend zu minimieren. Die Softwarearchitektur beschreibt dabei eine geeignete Zerlegung des Systems in Komponenten und die Abhängigkeiten zwischen diesen Komponenten.

In dieser Arbeit wird eine Methodik des evolutionären Architekturentwurfs komponentenbasierter Systeme entwickelt. Sowohl textbasierte als auch UML- basierte, grafische Spezifikationstechniken für Komponenten, deren Abhängigkeiten und komponentenbasierte Systeme werden erarbeitet. Mit diesen Spezifikationstechniken kann die Softwarearchitektur eines komponentenbasierten Systems vollständig und präzise beschrieben werden. Nach der Veränderung einzelner Komponentenspezifikationen in Rahmen eines evolutionären Entwicklungsschrittes können die Abhängigkeiten zwischen den Komponenten erneut auf ihre Gültigkeit hin überprüft werden. Die Integration der einzelnen Komponentenspezifikationen zu einem konsistenten Architekturmodell erfolgt so bereits auf der Spezifikationsebene.

Grundlage der erarbeiteten Methodik ist ein Systemmodell, das die Menge aller komponentenbasierter Systeme charakterisiert. Die Komponenten eines solchen Systems werden nebenläufig und verteilt ausgeführt. Sie kommunizieren über asynchrone Nachrichten, greifen lesend und schreibend auf Attribute von Komponenten zu und verändern die Struktur des komponentenbasierten Systems.

Das Systemmodell ist die Basis für die formale Fundierung der erarbeiteten Spezifikationstechniken. Über eine prädikatenbasierte formale Semantik wird einer Spezifikation eine Menge von Eigenschaften zugewiesen. Ein komponentenbasiertes System ist genau dann eine korrekte Implementierung der Spezifikation, wenn es diese Eigenschaften besitzt.

Die Konzeption und prototypische Realisierung einer durchgängigen und weitreichenden Werkzeugunterstützung garantieren die effektive und praxisorientierte Umsetzung der erarbeiteten Methodik. Die vorgestellten Konzepte reichen von der Architekturspezifikation über die Konsistenz- und Integrationsüberprüfung bis zu Prototypengenerierung und Testverfahren.

Frank Simon: Meßwertbasierte Qualitätssicherung - Ein generisches Distanzmaß zur Erweiterung bisheriger Softwareproduktmaße -

Promotion: Fakultät für Mathematik, Naturwissenschaften und Informatik, Brandenburgische Technische Universität Cottbus
Erstgutachter: Prof. Dr. C. Lewerentz
Zweitgutachter: Prof. Dr. R. Dumke
Drittgutachter: Prof. Dr. H. Schmidt
Datum der Prüfung: 12.07.2001

Kurzfassung:

Die wachsende Komplexität heutiger Softwaresysteme erfordert sowohl für die konstruktiven Phasen des Entwicklungsprozesses von Software als auch für die verschiedenen Arten der Softwarewartung wie Fehlerbeseitigung, Anpassung und Perfektionierung spezielle Techniken. Mit der zunehmenden Verbreitung iterativer Softwareerstellungsprozesse, d.h. der möglichst frühzeitigen Erstellung von Teilfunktionalität mit anschließenden, wiederholten berarbeitungen, die dem fertigen Produkt immer ähnlicher werden, und mit der zunehmenden Dauer des Einsatzes eines Softwareprodukts gewinnen speziell die Wartungsaktivitäten immer mehr an Bedeutung. Deren Zweck, nämlich das Erreichen und Halten einer hohen Produktqualität, zielt sowohl auf das nach außen beobachtbare Verhalten als auch auf den internen Aufbau des Produkts. Dem letzteren kommt dabei insbesondere aus wirtschaftlichen Gesichtspunkten eine große Bedeutung zu, da der Aufwand für Anpassungen der nach außen sichtbaren Funktionalität maßgeblich von der Qualität des internen Aufbaus abhängt. Der Fokus dieser Arbeit liegt in der auf dem Konzept der Vermessung von Software basierenden Sicherung und Verbesserung dieser internen Qualität. Hierfür werden neue Ansätze zur meßwertbasierten Qualitätssicherung von Softwareprodukten für praxisrelevante Systemgrößen beschrieben und für einige reale Systeme angewendet. Dies umfaßt Hilfen zur spezifischen Definition von Qualität, Prozesse zur Qualitätssicherung, Werkzeugunterstützung sowie die Identifikation von Problemen bei der Anwendung vieler bisheriger Softwaremaße. Die pragmatische Anwendung der Meßtheorie erlaubt eine Erweiterung bisheriger Meßansätze in Form eines auf Ähnlichkeiten zwischen Softwareelementen basierenden generischen Distanzmaßes. Dessen unterschiedliche Instanziierungen ermöglichen die automatische, anpaßbare Gruppierung von Elementen, sowie die ausdrucksstarke, ausschließlich durch Meßwerte bestimmte Softwarevisualisierung. Die ebenfalls in dieser Arbeit wiedergegebene empirische Arbeit, in der die Effizienz und Effektivität dieses Ansatzes mehrfach belegt wird, demonstriert die mögliche Bedeutung der meßwertbasierten Qualitätssicherung bei der Analyse großer Systeme.