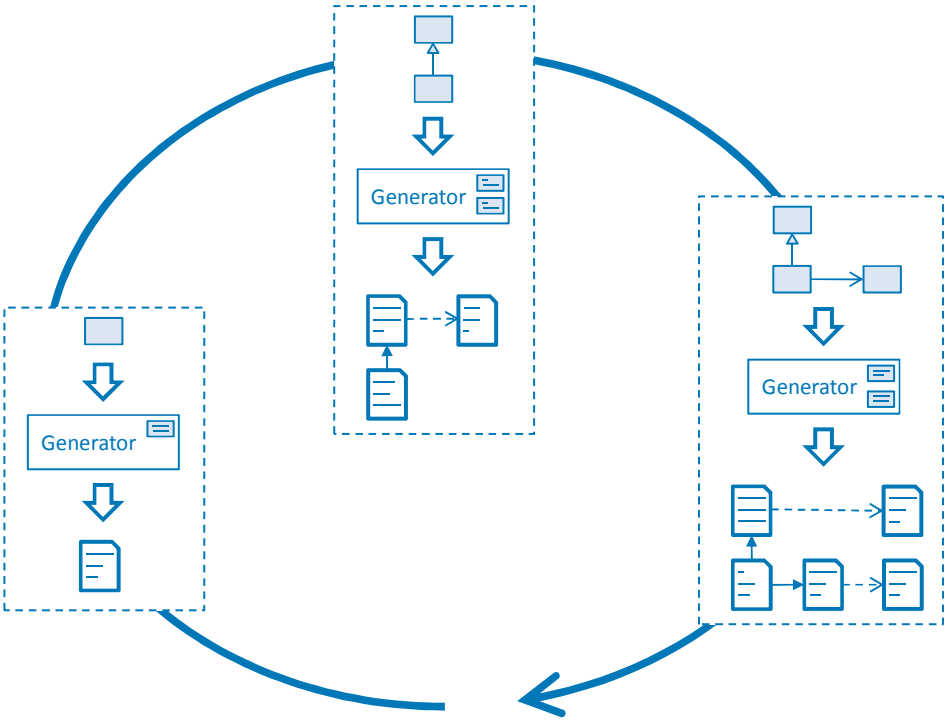
SE | RWTH AACHEN UNIVERSITY

Klaus Müller

Modellbasierte Unterstützung der Software Evolution im industriellen Kontext



Aachener Informatik-Berichte,
Software Engineering

Band 35

Hrsg: Prof. Dr. rer. nat. Bernhard Rumpe

Modellbasierte Unterstützung der Software Evolution im industriellen Kontext

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der
RWTH Aachen University zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

M.Sc.
Klaus Müller
aus Bernkastel-Wittlich

Berichter: Universitätsprofessor Dr. rer. nat. Bernhard Rumpe
Universitätsprofessor Dr. rer. nat. Udo Kelter

Tag der mündlichen Prüfung: 27. Juli 2017

Aachener Informatik-Berichte, Software Engineering

herausgegeben von
Prof. Dr. rer. nat. Bernhard Rumpe
Software Engineering
RWTH Aachen University

Band 35

Klaus Müller
RWTH Aachen University

Modellbasierte Unterstützung der Software Evolution im industriellen Kontext

Shaker Verlag
Aachen 2018

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Zugl.: D 82 (Diss. RWTH Aachen University, 2017)

Copyright Shaker Verlag 2018
Alle Rechte, auch das des auszugsweisen Nachdruckes, der auszugsweisen oder vollständigen Wiedergabe, der Speicherung in Datenverarbeitungsanlagen und der Übersetzung, vorbehalten.

Printed in Germany.

ISBN 978-3-8440-6034-8
ISSN 1869-9170

Shaker Verlag GmbH • Postfach 101818 • 52018 Aachen
Telefon: 02407 / 95 96 - 0 • Telefax: 02407 / 95 96 - 9
Internet: www.shaker.de • E-Mail: info@shaker.de

Kurzfassung

Software Evolution ist ein integraler Bestandteil der Softwareentwicklung, da jedes produktiv eingesetzte Softwaresystem kontinuierlich angepasst werden muss, damit es seinen Nutzen nicht verliert [Leh79]. Die Folge davon ist, dass die Komplexität der Softwaresysteme im Laufe der Zeit stetig ansteigt. Eine Möglichkeit um mit dieser steigenden Komplexität umzugehen besteht in dem Einsatz von Modellierungs- und Generierungstechniken [SVEH07, Sch12]. Diese führen allerdings zu weiteren Herausforderungen hinsichtlich der Software Evolution, da die eingesetzten Modelle und Generatoren ebenfalls kontinuierlich angepasst werden müssen. Eine besondere Herausforderung in diesem Kontext ist, dass jede Änderung eines Modells weitreichende Auswirkungen auf ein Softwaresystem haben kann, sobald auf Basis des Modells Generierungsaktivitäten durchgeführt werden. Eine wichtige Voraussetzung zur Unterstützung der Software Evolution besteht deshalb in der Ermittlung der Modelländerungen, die im Laufe der Zeit durchgeführt wurden.

Diese Arbeit zielt auf die modellbasierte Unterstützung der Software Evolution im industriellen Kontext ab. Der Fokus liegt auf Softwaresystemen, die generativ entwickelt und in denen Klassendiagramme zur Modellierung eingesetzt werden. Die wichtigsten wissenschaftlichen Beiträge lassen sich wie folgt zusammenfassen:

- Bei der automatischen Ermittlung von Modelländerungen kann im Allgemeinen nicht garantiert werden, dass stets die korrekten Modelländerungen erkannt werden. Zur Unterstützung der Modellierer wird in dieser Arbeit eine domänenspezifische Sprache (engl. domain specific language, DSL) bereitgestellt, die es Modellierern ermöglicht punktuell festzulegen, welche Modelländerungen für ausgewählte Modellelemente durchgeführt wurden. Diese Vorgaben werden in die automatische Differenzberechnung einbezogen, so dass Fehler in der Differenzberechnung vermieden werden können.

- Jede Modelländerung kann weitreichende Auswirkungen auf ein Softwaresystem haben. Aus diesem Grund wird in dieser Arbeit eine DSL bereitgestellt, über die es möglich ist, die Auswirkungen von Modelländerungen in Klassendiagrammen zu modellieren. Nachdem die durchgeführten Modelländerungen identifiziert wurden, wird auf Basis der modellierten Auswirkungen eine Checkliste generiert. Diese legt dar, welche Entwicklungsaktivitäten aufgrund der erkannten Modelländerungen durchgeführt werden müssen.

- Werden Datenmodelle zur Modellierung der Struktur der Daten eines Softwaresystems eingesetzt und generativ verarbeitet, können Änderungen der Datenmodelle die Durchführung von Datenmigrationen erfordern. Auf welche Weise diese Datenmigrationen vorgenommen werden müssen, richtet sich nach der Art der Modelländerungen. In dieser Arbeit werden die notwendigen Schritte zur Umsetzung von Datenmigrationen für verschiedenste Arten von Modelländerungen dargelegt. Darüber hinaus wird eine Vorgehensweise zur Generierung von Migrationsklassen vorgestellt, welche diese Schritte kapselt.

- Zur Implementierung eines Softwaresystems oder einer Variante eines Softwaresystems kann es hilfreich sein neue Varianten bestehender Generatoren zu erzeugen. Zur Entwicklung solcher Varianten werden in dieser Arbeit mehrere DSLs bereitgestellt. Über diese ist

es möglich eine neue Generatorvariante zu definieren, indem modelliert wird, auf welche Weise der Inhalt bestehender Generatorartefakte für die neue Variante angepasst werden soll. Als Ergebnis werden auf Basis der modellierten Operationen automatisch neue Generatorartefakte erzeugt, welche genau die modellierten Änderungen abbilden.

- Die Restriktionen, die festlegen, welche Komponenten eines Systems mit welchen anderen Komponenten interagieren dürfen, sind ein wichtiger Bestandteil der Softwarearchitektur eines Systems. In dieser Arbeit wird die Modellierung solcher Abhängigkeitsrestriktionen für Plugin-basierte Softwaresysteme durch eine DSL ermöglicht. Diese Restriktionen drücken erlaubte und verbotene Abhängigkeiten zwischen Plugins des Softwaresystems aus. Zudem wurde eine Prüfkomponente entwickelt, welche die Implementierung eines Softwaresystems kontinuierlich gegen die modellierten Restriktionen prüft.

Die genannten Teilgebiete der Software Evolution werden an einem als Herzstück eines Produkts im Diagnose-Bereich für Fahrzeuge eingesetzten Softwaresystem evaluiert. Die entwickelten Konzepte sind jedoch unabhängig von diesem Softwaresystem und lassen sich auf andere generativ entwickelte Softwaresysteme übertragen.

Abstract

Software evolution is an essential part of software development since every software system that is productively used needs to be adapted continuously or it becomes less useful over time [Leh79]. One possibility to cope with the increasing complexity arising from this is to use modeling and generation techniques [SVEH07, Sch12]. However, such techniques lead to further challenges when it comes to software evolution. One reason for this is that the used models and generators need to be adapted continuously too. One particular challenge in this context is that every change of a model can have severe impacts on the whole software system as soon as generation activities are performed based on the model. Thus, an important prerequisite to facilitate software evolution is to determine the model changes which are performed in the course of the time.

This work is targeted on supporting software evolution in an industrial context in a model-based way. In particular, the focus is on software systems that are being developed in a generative way and that use class diagrams for modeling purposes. The most important scientific contributions are summarized in the following:

- In general, an automatic approach to identify the performed model changes cannot guarantee that the reported model changes are always correct. To support the modelers in this context, a domain specific language (DSL) is provided. This DSL allows modelers to define for selected model elements which model changes have been performed. This knowledge is taken into account during the automatic identification of the model changes. In this way, errors in the automated identification of the model changes can be avoided.

- Every model change can have severe impacts on a software system. For this reason, a DSL is provided in this thesis that allows for modeling consequences of model changes in class diagrams. After having identified the performed model changes, a checklist is generated based on the modeled consequences. This checklist indicates which development activities need to be carried out due to the model changes.

- In case data models are used for modeling the structure of data in a software system, and these models are processed by a code generator, changes of data models can necessitate performing data migrations. In which way these data migrations need to be carried out depends on the types of model changes. In this thesis the steps that are necessary to perform data migrations for various types of model changes are explained. Moreover, a procedure to generate migration classes encapsulating these steps is presented.

- In order to implement a software system or a variant of a software system, it can be helpful to develop new variants of existing generators. For developing such variants multiple DSLs are provided in this thesis. These allow for defining a generator variant by modeling in which way the content of existing generator artifacts is to be adapted in the new variant. As a result, new generator artifacts are created automatically based on the modeled operations. These newly created artifacts conform to the modeled operations.

- The restrictions that indicate which component of a software system may interact with which other components are an important part of the software architecture of a system. To support software developers in developing a software system that conforms to such restrictions, a DSL is provided that makes it possible to model dependency restrictions for plugin-based software systems. These restrictions express allowed and forbidden dependencies between distinct plugins of the software system. Moreover, a checking component was developed to validate the implementation of a software system against the modeled restrictions continuously.

The stated areas of software evolution are evaluated by applying them to a software system that is at the core of a product in the area of vehicle diagnostics. However, the developed concepts are independent from this software system and can be transferred to other software systems that are developed in a model-driven way.

Danksagung

An dieser Stelle möchte ich mich bei all den Menschen bedanken, die mich während meiner Promotion unterstützt und so zum Erfolg dieser Dissertation beigetragen haben.

Mein ganz besonderer Dank gilt meinem Doktorvater Prof. Dr. Bernhard Rumpe für die Betreuung meiner Promotion. Die interessanten und konstruktiven Diskussionen mit dir sowie deine wertvollen Ratschläge haben maßgeblich zu dem Erfolg dieser Arbeit beigetragen. Zudem möchte ich mich dafür bedanken, dass ich die Möglichkeit hatte in einem interessanten Industrieprojekt bei der DSA Daten- und Systemtechnik GmbH zu arbeiten und die entwickelten Ideen dort anzuwenden.

Darüber hinaus danke ich Prof. Dr. Udo Kelter für die Bereitschaft zur Übernahme des Zweitgutachtens. Ebenso möchte ich mich bei Prof. Dr. Ulrik Schroeder für die Abnahme der Prüfung in der praktischen Informatik und bei Prof. Dr. Erika Ábrahám für die Abnahme der Prüfung in der theoretischen Informatik sowie die Leitung der Prüfungskommission bedanken.

Weiterhin gilt mein Dank der Geschäftsführung der DSA Daten- und Systemtechnik GmbH und hier insbesondere Dr. Ansgar Schleicher. Gemeinsam mit Bernhard hast du meine Promotion erst möglich gemacht. Bedanken möchte ich mich auch für die Möglichkeit die Dissertation parallel zur Arbeit bei DSA fertigzustellen.

Großer Dank gilt auch allen Kollegen am Lehrstuhl. Die vielfältigen gemeinsamen Freizeitaktivitäten haben mir immer viel Freude bereitet und waren eine gelungene Abwechslung in so manch stressigen Zeiten. Besonders bedanken möchte ich mich bei Dr. Arne Haber, Dr. Andreas Wortmann und Dr. Markus Look. Ihr habt mir nicht nur wichtiges Feedback im Laufe der Promotion gegeben, sondern auch dafür gesorgt, dass ich mich schnell in Aachen heimisch gefühlt habe. Ebenso möchte ich an dieser Stelle Christoph Schulze danken. Auch wenn wir beide durch die Industrieprojekte nicht so häufig am Lehrstuhl waren, warst du stets ein sehr angenehmer Bürokollege. Die gemeinsamen Poker Matches gegen dich bzw. mit dir werde ich vermissen. Weiterhin möchte ich Dr. Pedram Mir Seyed Nazari danken. Du hast mich zwar leider zu häufig im Tischtennis besiegt, aber das kann ich dir verzeihen. Danke für deine Hilfe bei der Prüfungsvorbereitung und für die vielen netten Gespräche. Weiterhin gilt mein besonderer Dank Dr. Alexander Roth. Wir haben so einige Stunden beim gemeinsamen Schreiben von Papern verbracht. Du hast nicht nur stets gutes Feedback gegeben, auf dich konnte man sich auch immer verlassen und du hattest auch immer ein offenes Uhr für sonstige Gespräche. Die Konferenzreise nach Angers mit dir und Pedram werde ich stets in guter Erinnerung behalten. Natürlich möchte ich mich auch bei allen bedanken, die ein Review von Teilen meiner Arbeit durchgeführt haben: Dr. Andreas Wortmann, Dr. Arne Haber, Robert Heim, Achim Lindt, Dr. Pedram Mir Seyed Nazari, Dr. Markus Look, Dr. Alexander Roth sowie Timo Greifenberg. Sollte ich jemanden vergessen haben, tut es mir leid.

Weiterhin möchte ich mich herzlich bei meinen Kollegen von der DSA bedanken. Mein Dank gilt unter anderem Christoph Briem, David Scheuren, Detlef Wilkening sowie Oliver Küsters. Die Diskussionen mit euch und euer Feedback haben mir nicht nur den Einstieg in die Arbeit bei DSA erleichtert, sondern auch zum Gelingen dieser Arbeit beigetragen. Weiterhin gilt mein Dank Gregor Skibka und Jochen Wollgarten. Durch euch habe ich einen schnellen Einstieg in die Modellierung und Generierung bei DSA erhalten. Die gemeinsame Zeit mit euch im Büro

war zudem sehr unterhaltsam. Hervorheben möchte ich zudem auch noch Jens Carstensen. Wir haben echt viele lustige Dinge erlebt und viele lustige Unterhaltungen geführt - auch wenn es für unsere Kollegen glaube ich nicht immer so lustig war. Darüber hinaus gilt mein großer Dank Sven Ewald. Gemeinsam mit Jens und dir habe ich meine Leidenschaft für das Bouldern entdeckt. An die vielen lustigen Unterhaltungen und Events mit dir denke ich immer sehr gerne zurück.

Ebenso möchte ich an dieser Stelle Prof. Dr. Rainer Oechsle von der Hochschule Trier danken. Du hast mir während meines Studiums in Trier früh die Möglichkeit gegeben erste Praxiserfahrungen in einem interessanten Projekt zu sammeln. Auch in den darauffolgenden Jahren hast du mir die Möglichkeit gegeben mich in verschiedensten Bereichen zu entfalten. Diese Erfahrungen haben mit dazu beigetragen, dass ich mich für eine Promotion entschieden habe.

Abschließend möchte ich ein besonders großes Danke an meine Eltern sowie meine Geschwister richten. Ihr wart immer für mich da und habt mich auf meinem gesamten Lebensweg stets unterstützt. Dadurch habt ihr es mir überhaupt erst möglich gemacht diesen Weg zu gehen. Zudem habt ihr immer Verständnis gezeigt, wenn ich durch das Studium bzw. die Promotion mal weniger Zeit hatte, was leider häufig vorkam. Ebenso möchte ich meiner Patentante Inge Sliwka herzlich danken. Du hast früh mein Interesse für Informatik geweckt und dieses Interesse sowie mich im Allgemeinen gefördert. Zu guter Letzt kann ich meiner Freundin Maria nicht genug danken. Du musstest wohl am meisten unter meiner Promotion leiden. Dessen ungeachtet hast du mich immer liebevoll unterstützt und motiviert. Du warst zudem immer verständnisvoll, wenn ich abends oder am Wochenende an der Dissertation oder an Papern gesessen habe. Ich weiß, dass das nicht selbstverständlich ist. Ohne deinen großen Rückhalt und deine Unterstützung wäre diese Arbeit nicht möglich gewesen.

Aachen, Juni 2018
Klaus Müller

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Motivation und Kontext der Arbeit 2
1.2 Wissenschaftliche Fragestellung 5
1.3 Aufbau der Arbeit 8

2 Kontext der Arbeit und Methodik **9**
2.1 Systemlandschaft bei DSA 9
2.2 Ermittlung von Modellunterschieden 11
2.3 Auswirkungsanalyse zur Generierung von Checklisten 13
2.4 Generierung von Migrationsfunktionalität zur Daten Evolution 15
2.5 Modellierung von Generatorvariabilität 19
2.6 Modellbasierte Prüfung der Architekturkonformität in Plugin-basierten Softwaresystemen 20

3 Grundlagen der DSL-Entwicklung **23**

4 Ermittlung von Modellunterschieden **27**
4.1 Grundlagen und Stand der Technik 27
4.2 Methodik zur Bestimmung von Modellunterschieden 31
4.2.1 Konvertierung von Eingabemodellen 33
4.2.2 Durchführung der Matchingphase 37
4.2.3 Integration von Delta-Benutzervorgaben in die Matchingphase 43
4.2.4 Durchführung der Differenzierungsphase 52

5 Auswirkungsanalyse zur Generierung von Checklisten **55**
5.1 Checkliste 56
5.2 Beispiel 57
5.2.1 Auswirkungsregel zur Analyse von objektrelationalen Mapping-Dateien (ORM-AR) 57
5.2.2 Checkliste zur Auswirkungsregel ORM-AR 59
5.3 Methodik zur Generierung von Checklisten 61
5.4 Ansatz zur regelbasierten Auswirkungsanalyse 64
5.4.1 Spezifikation von Auswirkungsregeln 64
5.4.2 Generierung und Erweiterung von Auswirkungsregeln 70
5.4.3 Beispiele für die Modellierung von Auswirkungsregeln 72

5.5 Diskussion . . . 88
5.5.1 Allgemeine Diskussion . . . 88
5.5.2 Diskussion zur Anwendbarkeit des Ansatzes in produktiv eingesetzten Softwaresystemen . . . 89
5.6 Verwandte Arbeiten . . . 91
5.7 Fallstudie in der Industrie . . . 93
5.7.1 Ziele und Forschungsfragen . . . 93
5.7.2 Aufbau und Gegenstand der Studie . . . 93
5.7.3 Ergebnisse und Diskussion . . . 95
5.7.4 Gültigkeit der Fallstudie . . . 98

6 Generierung von Migrationsfunktionalität zur Evolution von Daten 101
6.1 Grundlagen . . . 102
6.1.1 JSON . . . 102
6.1.2 Durchführung von Datenmigrationen . . . 104
6.2 Überblick zur Generierung von Migrationsfunktionalität . . . 104
6.2.1 Einsatzszenarien . . . 104
6.2.2 Methodik zur Generierung von Migrationsfunktionalität . . . 107
6.3 Ermittlung von Pfaden zu Modellelementen . . . 108
6.3.1 Ermittlung von Pfaden . . . 108
6.3.2 Konstruktion von Graphen zur Abbildung von Pfaden . . . 115
6.4 Vereinfachung des Differenzmodells . . . 117
6.4.1 Verschiebung eines Attributs . . . 118
6.4.2 Veränderung der Quellklasse einer Assoziation . . . 119
6.4.3 Veränderung der Zielklasse einer Assoziation . . . 121
6.4.4 Verschiebung eines Literals aus einer Enumeration . . . 122
6.4.5 Verschiebung eines Pakets oder einer Klasse . . . 123
6.4.6 Umbenennung eines Pakets . . . 123
6.4.7 Löschen eines Pakets . . . 123
6.4.8 Veränderung einer Klasse hin zu einer (nicht) abstrakten Klasse . . . 123
6.4.9 Hinzufügen einer Vererbungsbeziehung . . . 124
6.4.10 Nicht betrachtete Modelländerungen . . . 125
6.5 Verarbeitung von Modelländerungen . . . 126
6.5.1 Handhabung von Modelländerungen zu Attributen oder Assoziationen . 127
6.5.2 Handhabung von Modelländerungen zu Enumerationen . . . 140
6.5.3 Handhabung von Modelländerungen zu Klassen . . . 142
6.5.4 Zusammenfassung . . . 151
6.6 Generierung von Migrationsfunktionalität . . . 153
6.6.1 Minimierung der Graphen . . . 153
6.6.2 Generierung der Traversierungslogik . . . 155
6.6.3 Generierung der Migrationslogik . . . 158
6.6.4 Generierung der Migrationslogik für die Attributverschiebung . . . 161

6.6.5 Generierung der Migrationslogik für die Veränderung der Quellklasse einer Assoziation . . . 165
6.6.6 Durchführung manueller Anpassungen an generierter Migrationsfunktionalität . . . 166
6.7 Diskussion . . . 167
6.7.1 Allgemeine Diskussion . . . 167
6.7.2 Diskussion zur Anwendbarkeit des Ansatzes in produktiv eingesetzten Softwaresystemen . . . 167
6.8 Verwandte Arbeiten . . . 168
6.9 Fallstudie in der Industrie . . . 170
6.9.1 Ziele und Forschungsfragen . . . 170
6.9.2 Aufbau und Gegenstand der Studie . . . 171
6.9.3 Ergebnisse und Diskussion . . . 172
6.9.4 Gültigkeit der Fallstudie . . . 174

7 Modellierung von Generatorvariabilität 175
7.1 Grundlagen . . . 176
7.1.1 Feature-orientierte Programmierung . . . 176
7.1.2 Template-basierte Generierung . . . 177
7.2 Beispiel . . . 179
7.3 Methodik zur Modellierung von Generatorvariabilität . . . 181
7.4 Variabilitätskonzepte . . . 183
7.4.1 Variabilitätsregionen . . . 183
7.4.2 Definition von Kompositionsoperationen . . . 185
7.5 Konfiguration von Generatorvarianten . . . 187
7.5.1 Konfiguration von Generatorschichten . . . 187
7.5.2 Konfiguration einer Generatorvariante . . . 191
7.5.3 Automatische Ermittlung und Validitätsprüfung der benötigten CGLs . 192
7.6 Kompositionale Erzeugung von Generatorvarianten . . . 196
7.6.1 Überblick über die Schritte zur Durchführung der Komposition . . . 199
7.6.2 Ermittlung der Auswirkungen der im Schichtendefinitionsmodell modellierten Kompositionsoperationen . . . 200
7.6.3 Erzeugung der Generatorartefakte der Generatorvariante . . . 205
7.6.4 Adaption der Generatorartefakte zur Anpassung der Namen . . . 210
7.7 Diskussion . . . 215
7.7.1 Allgemeine Diskussion . . . 215
7.7.2 Diskussion zur Anwendbarkeit des Ansatzes in produktiv eingesetzten Softwaresystemen . . . 217
7.8 Verwandte Arbeiten . . . 218
7.9 Fallstudie in der Industrie . . . 219
7.9.1 Ziele und Forschungsfragen . . . 219
7.9.2 Aufbau und Gegenstand der Studie . . . 220
7.9.3 Ergebnisse und Diskussion . . . 221

7.9.4 Gültigkeit der Fallstudie 224

8 Modellbasierte Prüfung der Architekturkonformität in Plugin-basierten Softwaresystemen 225
8.1 Grundlagen 226
8.1.1 Begriffsdefinitionen 226
8.1.2 Motivation im Projektkontext und Anforderungen 228
8.2 Überblick über den Ansatz zur Prüfung der Architekturkonformität 230
8.3 Domänenspezifische Sprache zur Definition von Abhängigkeitsmodellen 232
8.3.1 Definition einer Feature und Plugin-Basis 233
8.3.2 Definition von Feature und Plugin-Gruppen 235
8.3.3 Definition von Abhängigkeitsrestriktionen 238
8.4 Semantik eines Abhängigkeitsmodells 243
8.5 Vorgehensweise zur Prüfung der Architekturkonformität 245
8.5.1 Implementierung 245
8.5.2 Beispiel 250
8.6 Anwendungsmethodik 254
8.7 Diskussion 255
8.7.1 Allgemeine Diskussion 255
8.7.2 Diskussion zur Anwendbarkeit des Ansatzes in produktiv eingesetzten Softwaresystemen 257
8.8 Verwandte Arbeiten 258
8.9 Fallstudie in der Industrie 261
8.9.1 Ziele und Forschungsfragen 261
8.9.2 Aufbau und Gegenstand der Studie 261
8.9.3 Ergebnisse und Diskussion 262
8.9.4 Gültigkeit der Fallstudie 269

9 Zusammenfassung und Ausblick 271
9.1 Zusammenfassung 271
9.2 Ausblick 274

Literaturverzeichnis 277

A Abkürzungsverzeichnis 311

B Notationen 313

C Checkliste 315

D Vordefinierte Bedingungen der DSL IRS 319

E Grammatiken 323
E.1 Grammatik der DSL DUPS 323
E.2 Grammatik der DSL IRS 324

E.3 Grammatik der DSL LDL 325
E.4 Grammatik der DSL PCL 326
E.5 Grammatik der DSL DepCoL 326

F Kontextbedingungen 331
F.1 Kontextbedingungen für IRS 331
F.2 Kontextbedingungen für LDL 332
F.3 Kontextbedingungen für PCL 333
F.4 Kontextbedingungen für DepCoL 333

G Fragebögen zu Fallstudien 337
G.1 Fragebögen bezüglich der Checklistenerzeugung 337
G.2 Fragebögen bezüglich der Architekturkonformität 339

H Glossar 341

I Curriculum Vitae 345