

Zehn Jahre WSR – Zwölf Jahre *Bauhaus*

Rainer Koschke
Universität Bremen
koschke@informatik.uni-bremen.de

Abstract: Das zehnjährige Jubiläum des deutschsprachigen Reengineering-Workshops WSR in Bad Honnef gibt Anlass, auch auf die eigene Arbeit zurück zu blicken, die thematisch mit dem WSR so eng verbunden ist. Unsere Forschergruppe *Bauhaus* beschäftigt sich schon seit zwölf Jahren mit dem Thema Software-Reengineering und verwandten Themen. Und seit Bestehen des WSR sind wir regelmäßige Teilnehmer am WSR. Dieser Artikel gibt einen historischen Rückblick auf unsere zwölfjährige Arbeit und fasst unsere Forschungsarbeiten auf diesem Gebiet zusammen.

1 Einführung

Unser Ziel im Projekt *Bauhaus* ist es, Programmierern durch analytische Methoden und Werkzeuge das Programmverstehen auf Code- und Architekturebene zu erleichtern, um damit die Qualität und Effizienz der Wartung zu verbessern [RVP06].

Dieses Ziel ist motiviert durch den hohen Aufwand für Wartung und Weiterentwicklung von Software, der insbesondere bei der Analyse der existierenden Software entsteht, bevor sie geändert und getestet werden kann [FH79]. Die Wartungsphase unterscheidet sich von der Erstentwicklung dadurch, dass in der Wartung mit bereits existierendem Code umgegangen werden muss. Nicht vorhandene, unvollständige oder veraltete Dokumentation oder Mängel im Code können das Verständnis deutlich erschweren.

Hier setzt das Forschungsprojekt *Bauhaus* an. Die Techniken des *Bauhaus* sollen die Produktivität der Wartungsingenieure erhöhen, indem sie diese mit einem breiten Spektrum an Programmanalysen unterstützen und leiten.

Bauhaus ist zugleich ein Projekt, eine Gruppe von Wissenschaftlern und eine Infrastruktur beziehungsweise ein Werkzeugkasten für die Programmanalyse.

Das Projekt *Bauhaus*: Das Projekt wird getragen von drei Partnern – von den beiden Universitäten Bremen und Stuttgart sowie vom Spin-Off Axivion GmbH¹, die *Bauhaus* gemeinsam voran treiben.

Dankbar sind wir unseren Förderern, ohne deren Unterstützung wir heute nicht so weit wären. Das Projekt *Bauhaus* wurde vom Land Baden-Württemberg, der zentralen Forschungsförderung der Universität Bremen, mehrfach von der Deutschen Forschungsgemeinschaft (DFG), dem Bundesministerium für Bildung und Forschung (BMBF) sowie der T-Nova Deutsche Telekom Innovationsgesellschaft mbH und Xerox Research gefördert.

¹<http://www.axivion.com>

Die Menschen am Bauhaus: Insgesamt 25 Wissenschaftler, drei Programmiererinnen und schätzungsweise mehr als 70 Studenten haben in der zwölfjährigen Geschichte im *Bauhaus* mitgewirkt. Gegenwärtig forschen und entwickeln acht Wissenschaftler an der Universität Bremen und sechs an der Universität Stuttgart im *Bauhaus*. Es zählt damit sowohl national als auch international zu den größten universitären Forschergruppen, die sich dem Thema Wartung widmen.

Die Infrastruktur des Bauhaus: Ein Grund für den Erfolg des *Bauhaus* sind sicherlich – neben der Fokussierung auf ein identitätsstiftendes und relevantes Forschungsthema – die Entwicklung einer tragfähigen Infrastruktur, von der jeder *Bauhaus*-Wissenschaftler profitieren kann und zu der jeder beiträgt. Auf diese Weise ist über die Jahre eine mächtige Infrastruktur für die Analyse von Programmen entstanden, auf deren Basis weiterführende Analysen entwickelt werden können. Zu Anfang hatten wir das selbe Problem wie jeder andere Forscher in unserem Gebiet, der die Wartung von Programmen mittels Werkzeugen unterstützen möchte: die Programme müssen erst einmal analysiert werden. Dazu benötigt man entsprechende Analysatoren, die in ihrer Funktionsweise den Frontends von Compilern gleichen (lexikalische, syntaktische und semantische Analyse). Die Entwicklung oder Einbindung solcher Frontends ist sehr zeitaufwendig und doch nur Mittel zum Zweck. Wer heute in *Bauhaus* forschen möchte, kann zurückgreifen auf Frontends für C, C++, Java und Ada sowie Byte-Code-Analysatoren für Java und C#. Außerdem stehen ihr oder ihm unter Anderem weiterführende Kontroll- und Datenflussanalysen, Zeigeranalysen, Metrikberechnungen sowie Visualisierungswerkzeuge zur Verfügung.

Im Folgenden stelle ich die zwölfjährige Geschichte des *Bauhaus* und seiner Forschungsbeiträge weitgehend chronologisch dar (zum Teil verliefen einige Entwicklungen parallel).

2 Die Gründerjahre und das Clustering

Das Projekt *Bauhaus* wurde 1996 als ein Forschungsprojekt des Instituts für Softwaretechnologie (ISTE) der Universität Stuttgart in Kooperation mit dem Fraunhofer Institut für experimentelles Software-Engineering (IESE) ins Leben gerufen². Die Gründer waren auf Seiten von ISTE Erhard Plödereder und Rainer Koschke und auf Seiten von IESE Jean-Marc DeBaud und Jean-François Girard. Warum die Abteilung *Programmiersprachen und Compilerbau* am ISTE die Wartung als zentrale Forschungsaufgabe für sich entdeckte, liegt auf der Hand: Compiler wissen sehr viel über ein Programm. Dieses Wissen kann nicht nur für die Generierung optimierten Codes genutzt werden, sondern auch, um Programmierern mehr über ihr Programm zu erzählen.

Dem neu geborenen Projekt gaben wir den Namen *Bauhaus*. Die Gründe für diesen Namen waren der Wunsch, einem deutschen Projekt einen deutschen Namen zu geben, der aber auch international tauglich ist. Zudem hatten Architekten des Original-Bauhaus maßgeblich zur zeitgenössischen Architektur beigetragen, national wie international. Da wir uns auch Software-Architekturen auf die Fahne schreiben wollten und das Prinzip *Form follows Function* bei der Gestaltung von Software für uns handlungsbestimmend sein sollte,

²<http://www.bauhaus-stuttgart.de>

passte der Name gut zu unserem Vorhaben. Ebenso teilen wir die Grundsätze des Original-Bauhaus, die wir auf die Wissenschaft übertragen: Einheit von Wissenschaft und Handwerk sowie Zusammenwirken von Wissenschaft und Industrie.

Die ersten Entwicklungen fanden prototypisch mit dem Analyse- und Transformationswerkzeug Refine der Firma Reasoning-Systems statt. ISTE und IESE pflegten hierzu ein gemeinsames Code-Repository. Dieses Rahmenwerk nahm uns die lästige Arbeit der Analyse von C-Programmen ab. Wir wollten uns zunächst auf die Analyse der Sprache C konzentrieren, die damals dominierend war. Zwar war (und ist) COBOL immer noch weiter verbreitet, aber hierzu fehlte uns der Zugriff auf Beispiel-Code und entsprechende Analyseunterstützungen.

Unser Hauptforschungsinteresse galt damals dem so genannten Software-Clustering, bei dem statische Software-Einheiten – wie Funktionen, Variablen und Typen – aufgrund unterschiedlicher Ähnlichkeitsmetriken zu abstrakten Datentypen oder Klassen zusammen gruppiert wurden. Dies war zu dieser Zeit ein aktives Forschungsgebiet. Diese Einheiten bilden die kleinsten für die Architektur noch relevanten Sinneinheiten. Ihre Gruppierung zu größeren Komponenten der Software-Architektur war unser längerfristiges Ziel.

Zunächst evaluierten wir existierende Ansätze des Software-Clusterings [GK97, GKS97a]. Um die aufgedeckten Schwächen existierender Verfahren zu verbessern, entwickelten wir dann eigene Ansätze [GKS97b, GKS99, GK00, KCC06]. Diese Forschung mündete schließlich in zwei Dissertationen: die meines Freundes und Mitstreiters Jean-François Girard [Gir06] und meine eigene [Kos99].

3 Technologische Fortschritte

Die Analyseumgebung Refine, die wir verwendeten, erleichterte es uns, Prototypen zu entwickeln. Schwächen in der Performanz und Interoperabilität und die proprietäre Sprache drohten uns jedoch in eine technologische Sackgasse zu führen. Aus diesem Grunde wagten wir 1997 einen technologischen Neuanfang. Wir reimplementierten unsere Analysen in der Programmiersprache Ada, entwickelten eine erste Infrastruktur für die Programm-analyse und schafften eine Integration mit dem Grapheneditor Rigi³, einem Visualisierungswerkzeug speziell für das Reengineering, für den uns dessen „Vater“ Hausi Müller freundlicherweise den Quellcode überließ. Die Wahl der Programmiersprache Ada erschien uns unter dem Gesichtspunkt der langfristigen Wartbarkeit günstiger als etwa C oder C++, insbesondere im universitären Umfeld, in dem Studenten sich in kurzer Zeit in ein System einarbeiten müssen, um ihre Diplomarbeit damit entwickeln zu können. Darüber hinaus war Ada die erste Programmiersprache, die Studenten an der Universität Stuttgart erlernten. Wir konnten also auf viele Studenten mit den notwendigen Programmiersprachenkenntnissen zurückgreifen. Der Advent von Java hat das Blatt leider gewendet, so dass wir Studenten insbesondere in Bremen häufig erst in die Programmiersprache einarbeiten müssen. Ein Stück weit geht es uns hier wie den Firmen, die COBOL als Programmiersprache einsetzen. Dennoch hat sich die Sprache unter den Gesichtspunkten

³<http://www.rigi.cs.uvic.ca/>

Wartbarkeit und Performanz bewährt. Bauhaus hat heute eine Größe von circa 1,5 MLOC erreicht. Mit anderen Sprachen wie C oder C++, die damals zur Debatte standen, wäre uns das System in dieser Größenordnung unter den besonderen Umständen der Mitwirkung von Studenten in Form von sehr kurzen studentischen Arbeiten vermutlich längst um die Ohren geflogen.

Unsere Gruppe in Stuttgart wuchs weiter. Nicht nur lieferte Georg Schied, der schon vor Start von *Bauhaus* Mitarbeiter am ISTE war, bei unserer Clustering-Forschung wichtige Beiträge, sondern es wurden auch alle neuen Mitglieder unserer Arbeitsgruppe in Stuttgart in das Projekt *Bauhaus* integriert. Dazu zählten Thomas Eisenbarth, der bereits als Student entscheidende Beiträge zum Projekt lieferte, Hartmut Keller, Daniel Simon, Jörg Czeranski, Yan Zhang und Holger Kienle. Holger Kienle wechselte später aus persönlichen Gründen in die Forschergruppe von Hausi Müller nach Victoria, was unsere Verflechtung mit der University of Victoria weiter vertiefte. Auch die Gruppe am IESE wuchs. Martin Würthner, der als Diplomand bei uns wichtige Beiträge für unsere Programmrepräsentation lieferte, wurde wissenschaftlicher Mitarbeiter am IESE.

Das Projekt erfuhr auch zum ersten Mal Anklang in der Industrie, was zu einem Forschungsprojekt mit der Deutsche Telekom Innovationsgesellschaft mbH führte.

Die Grundlagen unserer heutigen Infrastruktur wurden in dieser Phase geschaffen. Zentrale Elemente darin sind unsere zwei unterschiedlichen programmiersprachenübergreifenden Programmrepräsentationen [KGW98]: Die eine ist unsere *Intermediate Language (IML)*, mit der wir Programme feingranular in Form verallgemeinerter abstrakter Syntaxbäume (AST) darstellen [Wür96, Roh98]. Die andere ist der *Resource-Flow-Graph (RFG)*, der globale Programmelemente und ihre Abhängigkeiten als Graph darstellt [Eis98].

Die Zwischendarstellung IML erfüllt zwei, auf den ersten Blick sich widersprechende Anforderungen: (1) die vereinheitlichte Darstellung verschiedener Programmiersprachen, um nachfolgende Analysen für verschiedene Sprachen wiederverwenden zu können und (2) die Quellennähe, um Analyseergebnisse dem Programmierer quellennah darstellen zu können. Wir erreichen beide Design-Ziele, indem wir einerseits unterschiedliche Konstrukte auf einen Kern primitiver programmiersprachenübergreifender Konstrukte zurückführen (d.h. den Code normalisieren) und andererseits in Form von Spezialisierung der AST-Knotentypen für die jeweilige Programmiersprache und semantischer Annotation der AST-Knoten alle Informationen unterbringen, die wir benötigen, um den Code in seiner Originalform wieder ausgeben zu können [KGW98].

Die IML enthält alle Details eines Programms, die wir für weiterführende Detailanalysen benötigen, wie zum Beispiel die Klonerkennung, die Erhebung von Metriken sowie unsere Kontroll- und Datenflussanalysen. Sie ist jedoch zu detailliert, um als Medium für die Interaktion mit dem Benutzer auf Architekturebene zu dienen. Für diesen Zweck überführen wir die IML in einen RFG, der den Code in den Funktions- und Methodenrümpfen auslöst und nur die globalen Abhängigkeiten darstellt. Dieser Graph konnte in Rigi visualisiert und manipuliert werden.

Die Fortschritte in der Technologie gingen leider einher mit einer zunehmenden Entkopplung zwischen ISTE und IESE. Die Ada-Entwicklung wurde allein von ISTE voran getrieben, während das IESE weiter an Refine festhielt. Der Entkopplung auf technischer

Ebene folgte ein Stück weit auch die Entkopplung in der engen Zusammenarbeit auf wissenschaftlicher Ebene. Mit dem IESE blieben wir weiterhin freundschaftlich verbunden (nicht zuletzt weil später auch unser Diplomand Jens Knodel noch ans IESE wechselte), aber die Entwicklung nahm nun zwei getrennte Pfade.

4 Neuorientierung: die Suche nach Merkmalen

Die Jahrtausendwende stellte einen Höhepunkt in der Aufmerksamkeit dar, die das Thema Wartung auch in der breiten Öffentlichkeit genoss. Sicherlich war auch das ausschlaggebend für den ersten Workshop Software-Reengineering in Bad Honnef im Jahre 1999. Dieses Jahr war auch gleichzeitig das Jahr, in dem ich meine Promotion zum Thema Software-Clustering abschloss. Danach wollte ich mich auch neuen Forschungsthemen widmen. Inspiriert wurde ich hierbei durch Arbeiten von Gregor Snelling und Christian Lindig, die die formale Begriffsanalyse auf verschiedene Reengineering-Probleme anwandten. Mit dieser Technik hatte ich mich zunächst nur in der Lehre in meiner Vorlesung Software-Reengineering auseinandergesetzt [Kos00]. Zusammen mit meinen Kollegen Thomas Eisenbarth und Daniel Simon wandte ich die formale Begriffsanalyse zunächst auf die Analyse von Features und Komponenten in Software-Produktlinien an [EKS00]. Die Idee übertrugen wir dann auf die Merkmalsuche, das heißt, das Problem, die Menge von Komponenten zu identifizieren, die eine gesuchte Funktionalität (Merkmal, Feature) implementieren. Daraus folgte eine Reihe von Publikationen [EKS01b, EKS01a, EKS01c, EKS02, EKS03, KQ05, BLE⁺08], mit denen wir die Technik schrittweise ausbauten und verschiedene Facetten untersuchten. Für die Arbeit bei der ICSM 2001 zu diesem Thema erhielten wir erstmalig einen Best-Paper-Award. Diese Forschung führte unter anderem auch zur Promotion von Daniel Simon [Sim06]. Andere Forscher haben unsere Arbeiten aufgegriffen und weiterentwickelt, was durch mehr als 160 Zitierungen unserer Arbeit [EKS03] unterstrichen wird⁴.

5 Mehr Dynamik: Von der Struktur zum Verhalten

In meiner Dissertation ging es um Verfahren, zusammengehörige globale Deklarationen in C wie Funktionen, Typen und Variablen zu finden und zu abstrakten Datentypen und -objekten zu gruppieren. Hat man solche Komponenten gefunden, schließt sich unmittelbar daran die Frage an, wie man die Komponente richtig benutzt, mit anderen Worten, was das Protokoll der Komponente ist. Meist ist die Benutzung leider nicht vollständig spezifiziert. Das Protokoll beschreibt, in welcher Reihenfolge Operationen aus der Schnittstelle angewandt werden dürfen und welche Vor- und Nachbedingungen zu den Parametern gelten. Dieser Frage widmeten wir uns parallel zur Merkmalsuche. Die prinzipiellen Ideen hatten wir dabei im Jahre 2001 zum ersten Mal auf dem WSR vorgestellt. Die Idee war es, für jede Instanz einer Komponente (abstrakter Datentyp oder Klasse) zu bestimmen,

⁴<http://scholar.google.com>

in welcher Reihenfolge und unter welchen Bedingungen Operationen der Schnittstelle der Komponente auf die Instanz angewandt werden. Das Resultat ist ein Objektprozessgraph, der eine Projektion des Kontrollflussgraph darstellt, die nur die Operationen auf einem bestimmten Objekt (einer Instanz) und alle Bedingungen enthält, von denen die relevanten Anweisungen kontrollabhängig sind. Anschließend vereinigt man die einzelnen Objektprozessgraphen zum Protokoll der Komponente, unter der Voraussetzung, dass jedes Objekt korrekt verwendet wurde. Das Verfahren lernt und verallgemeinert also aus Beispielverwendungen.

Zunächst gingen wir (Gunther Vogel, Thomas Eisenbarth und ich) das Problem mit rein statischen Analysen an [EKV02, EKV05, Vog06]. Hierzu braucht man möglichst genaue Zeiger- und Seiteneffektanalysen, die treffsicher alle und am besten nur die Anweisungen ermitteln, die sich auf ein relevantes Objekt beziehen.

Das Jahr 2004 brachte dann einige Veränderungen mit sich. Nach meinem Wechsel an die Universität Bremen entstand ein neuer Standort im Nordwesten Deutschlands. *Bauhaus* ist damit sowohl im Süden als auch im Norden vertreten (ich vergleiche dies gerne mit der Aufteilung in Aldi-Nord und Aldi-Süd). Der Standort in Bremen musste erst aufgebaut werden. Der erste wissenschaftliche Mitarbeiter war Jochen Quante, der sich auch für die Protokollerkennung interessierte. Komplementär zum statischen Ansatz von Gunther Vogel in Stuttgart wandte er dynamische Analysen an [QK06, QK08, QK07, Qua07]. Die erste Arbeit hierzu wurde zu unserem zweiten *Best-Paper-Award* [QK06].

Die Distanz von ca. 640 km, die zwischen Bremen und Stuttgart liegen, überbrückten wir so gut es ging mit den Mitteln moderner Informationstechnologie. Der Code zu *Bauhaus* befindet sich in einem Repository, auf das alle Partner zugreifen. Wir benutzen einen Chat-Raum und ein Wiki für die Kommunikation und übertragen Vorträge über das Internet. Natürlich können diese Mittel eine gemeinsame Tasse Kaffee oder – wie im Norden etwas üblicher – Tee nicht ersetzen. Aus diesem Grund besuchen wir uns gegenseitig von Zeit zu Zeit, um den persönlichen Draht aufrecht zu erhalten.

6 Die Liebe zum Detail

Während die Marktaufteilung bei Aldi-Nord und -Süd geographisch verläuft, stellen sich Bauhaus-Nord und -Süd inhaltlich unterschiedlich auf. Die Stuttgarter nehmen sich der quellcodenahen semantischen Programmanalysen an und zeichnen sich verantwortlich für die Zwischendarstellung IML. Die Bremer wenden sich den Architekturanalysen zu, basierend auf der Zwischendarstellung RFG. So wenigstens der grobe Plan. Tatsächlich sind die Grenzen oft unscharf und bei der Protokollrekonstruktion arbeiten wir am selben Problem aus zwei unterschiedlichen Perspektiven.

Die Stuttgarter haben die IML immer weiter ausgebaut für weitere Programmiersprachen wie C++, Java und Ada und entsprechende Frontends dafür integriert. Für die IML entwickelten sie Zeigeranalysen der unterschiedlichsten Form. Diese sind eine wichtige Voraussetzung für die weiteren statischen Analysen in Bauhaus. Eduard Wiebe forscht an der Verbesserung der Präzision unserer Zeigeranalysen. Die gewonnenen Abschätzungen

erlauben die Bestimmung interprozeduraler Datenflussinformationen, die in IML mittels der *Interprocedural Static Single Assignment Form (ISSA)* explizit repräsentiert werden [SVKW07].

Gunther Vogel erforscht statische Verfahren zur Gewinnung von Objektprozessgraphen und zur Extraktion und Verifikation von Software-Protokollen [EKV02, EKV05, Vog06]. Die weiteren Forschungsschwerpunkte der Stuttgarter liegen in der Analyse von grafischen Benutzeroberflächen sowie der Analyse von nebenläufigen Programmen. Stefan Staiger arbeitet an der Erkennung und hierarchischen Einbettung von Widgets [Sta07b, Sta07a]. Aoun Raza und Steffen Keul entwickeln Verfahren zur Analyse nebenläufiger Programme [Raz06]. Nachdem dieses Thema nahezu gleichzeitig sowohl von einer Großbank als auch einem Produzenten von Flugüberwachungssoftware an uns herangetragen worden war, begannen wir uns mit den spannenden und höchst komplexen Fragen zu interessieren, die für nebenläufige Software zu beantworten sind. Seit 2005 bauen wir die IML-Strukturen für die Analyse dieser Software aus. Inzwischen ist eine erste Implementierung einer Analyse zur Erkennung von *Race-Conditions* in Bauhaus integriert und wird gegenwärtig in industriellem Umfeld erprobt.

Unsere Infrastruktur hat sich für diese Art von Analysen als sehr tragfähig erwiesen. Das Design-Ziel der IML war es, Programmanalysen für eine größere Menge unterschiedlicher Programmiersprachen zu ermöglichen. Sie war jedoch nicht dazu gedacht, Code-Transformationen zu unterstützen. In einem zweijährigen Projekt an der Universität Bremen hatten Studenten den Versuch unternommen, Refactorings auf Basis der IML zu implementieren – und zwar ambitioniert auch für die Sprachen C und C++, bei denen man mit der Präprozessorproblematik zu kämpfen hat. Die Refactorings selbst waren schon nicht einfach zu implementieren, aber auf noch größere Schwierigkeiten trafen wir beim Unparsing, das heißt, der Rückgewinnung des Quelltextes aus der modifizierten Zwischendarstellung. Die IML ist im Kern ein abstrakter Syntaxbaum, der von vielen Details, die keine Relevanz für die Analyse haben, abstrahiert. Viele der abstrahierten Details spielen jedoch bei der Quelltextausgabe eine Rolle. Zum Beispiel sollte eine Rückübersetzung jedes Semikolon wieder an die richtige Spalte setzen. Solche Details sind aber nicht in der IML repräsentiert.

7 Selbstreflexion

Clustering ist eine Form des unüberwachten automatischen Klassifizierens. Meine Evaluation existierender Clustering-Verfahren in meiner Dissertation zeigte jedoch, dass die Ergebnisse des Clusterings nicht die Klassifikation von Entwicklern reflektieren, die eher semantische Kriterien anlegen, die aber automatischen Verfahren nicht zugänglich sind. So sehr wie die Vorwärtsentwicklung geprägt ist von Intuition, Erfahrung und Kreativität und damit nur schwer automatisierbar ist, so wenig kann man erwarten, dass die Umkehrung dieses Prozesses – Reverse-Engineering genannt – völlig automatisierbar ist. Aus diesem Grund erschien mir die Reflexionsmethode von Murphy et al. [MNS01] attraktiver für die Rekonstruktion statischer Architektursichten. Die Idee dieser Methode ist sehr einfach. Zunächst stellt man eine Hypothese über die erwartete Architektur auf und bildet dann

die Implementierungskomponenten (Klassen, Dateien, Funktionen etc.) auf die Architekturkomponenten ab. Dank der Verbindung von Architektur- und Implementierungsmodell können deren Unterschiede durch einen automatischen Vergleich bestimmt werden. Nach Analyse der Unterschiede können Architektur, Implementierung oder Abbildung angepasst werden, um die Unterschiede zu beseitigen. Man iteriert diese Schritte solange, bis die beiden Modelle ausreichend ähnlich sind.

Dieses Verfahren haben wir schrittweise ausgebaut. Zunächst erweiterten wir das Architekturmodell um Hierarchien [KS03, KS04]. Danach versuchten wir Unterstützungen für die Abbildung zu finden, die in der Praxis den aufwendigsten Schritt darstellt. Die Vergangenheit holt einen ja bekanntermaßen immer wieder ein. Und so versuchten wir durch Ähnlichkeitsmetriken von Clustering-Techniken dem Re-Architekten Vorschläge für die Abbildung zu unterbreiten. Die Idee war es – ausgehend von einer partiellen Abbildung – für noch nicht abgebildete Elemente zu bestimmen, wohin deren Nachbarn abgebildet sind. Deren Abbildungsziele und das Bestreben, mögliche Diskrepanzen zur Architektur zu minimieren, liefern dann Hinweise, wohin ein noch nicht abgebildetes Element abgebildet werden könnte [CKS05, CKS07]. Auf diese Weise entstand ein Verfahren, das zugleich top-down und bottom-up vorging, wie es auch dem Vorgehen von Programmierern beim Programmverstehen entspricht [vMV95].

Das Schöne an diesen Arbeiten war auch, dass sie zusammen mit Margaret-Anne Storey von der University of Victoria statt gefunden haben, die uns auch schon mit ihren Arbeiten zur Visualisierung in Rigi inspiriert hatte. Rigi war allerdings mittlerweile im *Bauhaus* abgelöst worden. Wir waren architektonisch an seine Grenzen gestoßen. Durch die Verwendung unterschiedlicher Programmiersprachen (Ada für *Bauhaus* und C++ für Rigi) und doppelter Datenhaltung in den unterschiedlichen Programmiersprachen wurde das System nicht mehr handhabbar. Außerdem stießen wir auch in Bezug auf die Performanz an die Grenzen von Rigi. Ein Studentenprojekt hat 2001 einen Nachfolger für Rigi implementiert. Die Herkunft des Nachfolgers ist ihm jedoch auch heute noch anzusehen.

Im Jahre 2002 organisierten Arie van Deursen, Rick Kazman und ich das Dagstuhl-Seminar *Software Architecture: Recovery and Modelling*, in dem wir uns unserem Lieblingsthema, der Rekonstruktion von Software-Architekturen, widmeten. Im Musikzimmer in Schloss Dagstuhl arbeitete einer unserer parallelen Arbeitskreise einen Methodenrahmen für die Architekturrekonstruktion aus. Weil dazu eine hohe Orchestrierung verschiedener Rekonstruktionsverfahren notwendig ist und nicht zuletzt weil wir im Musikzimmer darüber nachgedacht hatten, nannten wir diesen Rahmen *Symphony* [vDHK⁺04a]. Eine Darstellung verschiedener Architektursichten und ihrer Rekonstruktionstechniken für *Symphony* präsentierten wir dann wenig später beim WSR 2006 [vDHK⁺04b]. Diese Vorstellung gab den Anstoß zu meiner systematischen und umfassenden Literaturübersicht zu Sichten und Rekonstruktionstechniken [Kos05], einer Veröffentlichung, die direkt aus dem WSR hervorging.

Den Sichten und Architekturrekonstruktionstechniken widmen wir uns auch in einem BMBF-Projekt zusammen mit unserem Partner der ersten Stunde, dem IESE, und mehreren Industriepartnern. Im Projekt ArQuE betten wir die Thematik ein in das Quality-Engineering. Hierzu verknüpfen wir Methoden und Software-Engineering-Bausteine (Produkt- und Prozessmetriken, Architekturvalidierung und -analyse, Reverse-Engineering,

Qualitätssicherung) zur Erstellung von zielgerichteten Qualitätsmodellen, die architekturzentriert konzipiert, kontinuierlich validiert und zur Vorhersage der Entwicklung von Qualitätseigenschaften bei Weiterentwicklungen von existierenden Produkten herangezogen werden können.

8 Vom Hobby zum wissenschaftlichen Schwerpunkt: Klonerkennung

In der Vorlesung Software-Reengineering, die ich in Stuttgart zum ersten Mal im Wintersemester 1999/00 hielt, stellte ich unter anderem verschiedene Techniken zur Erkennung duplizierten Codes (auch als Software-Klone bekannt) vor. Unbefriedigend war, dass ich den Studenten am Ende nicht sagen konnte, welche der Techniken unter welchen Umständen den anderen vorzuziehen ist. Leider gab es hierzu keinerlei vergleichende quantitative Untersuchungen. Aus diesem Grund nahm ich Kontakt zu den Forschern in der Klonerkennungsszene auf und fragte sie, ob sie an einem quantitativen Vergleich interessiert wären. Das Interesse war groß und alle Forscher wollten sich beteiligen. Ich fand in Stefan Bellon einen ausgezeichneten Diplomanden, der an der geplanten Untersuchung Gefallen fand. Heraus kam eine Diplomarbeit, die vermutlich zu den meistzitierten deutschsprachigen Diplomarbeiten in der Softwaretechnik gehört [Bel02] – und zwar in englischsprachigen Konferenzen und Zeitschriften.

Dieses Experiment war auch deshalb besonders spannend, weil es sechs internationale Forschergruppen auf drei Kontinenten integrierte. Diese Vielfältigkeit machte es aber auch nicht leicht, die Ergebnisse dann zeitnah in einer geeigneten Zeitschrift zu publizieren. Für unseren ersten Plan eines *Special Issues* zum Thema Klonerkennung in der Zeitschrift *Transactions on Software Engineering (TSE)* erhielten wir eine Abfuhr mit der Aussage, dass das kein interessantes Thema sei. So sehr kann man sich bei Forschungstrends irren. Schon davor und auch insbesondere danach ist eine ganze Reihe Artikel zu Software-Klonen in TSE erschienen. Davon unbeirrt reichten wir dann drei Artikel zu dem Experiment mit verschiedenen Autoren aus dem Experiment bei TSE ein, von denen dann schließlich zwei veröffentlicht wurden, darunter unser Artikel über das Experiment selbst und die Ergebnisse [BKA⁺07].

Aus dem, was eher als Hobby begann, wurde dann ein echter Forschungsschwerpunkt unserer wachsenden Arbeitsgruppe in Bremen. Eines der Ergebnisse des Experiments war, dass syntaktische Verfahren zwar eine hohe Präzision aufweisen, jedoch Probleme bei der Laufzeit haben. Umgekehrt verhielt es sich mit den token-basierten Verfahren, die in linearer Zeit arbeiten, sich jedoch an keine syntaktischen Grenzen halten können. Auf Basis unserer Erkenntnisse aus dem Experiment entwickelten wir deshalb ein Verfahren, das syntaktisch abgeschlossene Klone in linearer Zeit finden konnte [KFF06, FKF08]. Als weitere Folge dieses Experiments organisierte ich 2006 ein Dagstuhl-Seminar mit Andrew Walenstein, Ettore Merlo und Arun Lakhotia zum Thema Software-Klone. In einem Übersichtsvortrag stellte ich den Stand der Forschung in diesem Gebiet dar und zählte die aus meiner Sicht offenen Forschungsfragen auf [Kos07] (eine noch detailliertere Darstellung erscheint als ein Kapitel in einem Buch über Software-Evolution [Kos08]). Die Seminarteilnehmer bestätigten mich in diesen Punkten und so formulierte ich ein Forschungspro-

gramm, das eine Reihe der offenen Fragen adressiert. Die DFG fördert dieses Forschungsprogramm nun mit zwei wissenschaftlichen Mitarbeitern, und wir sind gespannt, welche Ergebnisse aus einem ehemaligen Hobby erwachsen werden.

9 Wiederverwendung aller Ebenen: Produktlinien

Die Beschäftigung mit Klonerkennungstechniken hatte auch Auswirkung auf noch ein anderes Thema, das uns zum Forschungsschwerpunkt wird, und zwar der Frage, wie man Software-Varianten, die durch Copy&Paste im großen Stil entstanden sind, zu organisierten Software-Produktlinien konsolidieren kann. Hierzu können wir eine Reihe von Techniken zusammenführen, mit denen wir uns schon länger in der Forschung beschäftigen. Die Ideen stellten wir 2006 beim WSR vor [Kos06]. Die Klonerkennung benutzen wir, um Gemeinsamkeiten zwischen Varianten auf Code-Ebene zu identifizieren. Die Merkmalsuche erweitern wir so, dass wir Merkmale auf Komponenten von Varianten abbilden und auch Unterschiede in den Implementierungen der Varianten finden können. Die Reflexionsanalyse verwenden wir, um Unterschiede und Gemeinsamkeiten auf der strukturellen Architektur verschiedener Varianten zu bestimmen. Die Protokollerkennung erweitern wir, um Gemeinsamkeiten und Unterschiede der Verwendungen gleicher oder ähnlicher Komponenten in verschiedenen Varianten zu identifizieren. Hier nutzen wir Synergieeffekte verschiedener Forschungsrichtungen aus, mit denen wir uns beschäftigen. Die Herausforderung ist, diese Techniken so zu erweitern und geschickt zu kombinieren, dass man möglichst viel bereits ermitteltes Wissen von den bisher untersuchten Varianten inkrementell auf die als nächstes zu untersuchenden Varianten übertragen kann.

Erste Erweiterungen der Reflexionsmethode sowie ermutigende Fallstudien mit industriellen Partnern haben wir bei der *Working Conference on Reverse Engineering 2007* veröffentlicht, bei der wir mit diesem Beitrag unseren dritten *Best-Paper-Award* erhalten haben [FKBA07]. Bei der *Conference on Software Maintenance and Reengineering 2008* haben wir eine Fallstudie veröffentlicht, in der wir mit Klonerkennungstechniken in einer industriellen Produktlinie kopierte Anteile zwischen produktspezifischen Erweiterungen lokalisieren, die man dann in den Produktlinienkern verlegen kann [MBKM08].

Im Umfeld des Bauhaus fand die Begriffsanalyse ihren Weg in ein weiteres Thema der Produktlinien. Felix Loesch erarbeitete in seiner Promotionsforschung in Stuttgart praktikable Ansätze, die Variabilität in Produktlinien begreifbar machen und gleichzeitig Vereinfachungen und Optimierungen der Produktlinien vorschlagen. Seine Studien zeigten recht überraschende Ergebnisse in der Anwendung der Techniken auf reale Systeme des Automobilbaus [LP07a, LP07b]. Mittlerweile haben weitere Fallstudien diese erstaunlichen Verbesserungen immer wieder bestätigt, so dass es sich wohl nicht um Eintagsfliegen handelte.

In der Vorlaufphase wird diese Forschung von der zentralen Forschungsförderung der Universität Bremen unterstützt. Mittlerweile fördert auch die DFG dieses Vorhaben.

10 Unsere Reifeprüfung: Technologietransfer

Was als Forschungsprojekt an der Universität Stuttgart begonnen und in Kooperation mit der Universität Bremen weiterentwickelt wurde, hatte vor ein paar Jahren einen Reifegrad erreicht, der uns motiviert hat, über Technologietransfer ernsthaft nachzudenken. Von den ersten Ideen und dem festen Entschluss, den wir nächstens bei unserem Dagstuhl-Seminar zur Architekturekonstruktion getroffen haben, war es dann doch noch ein langer und steiniger – weil bürokratischer – Weg, bis unser Vorhaben Wirklichkeit wurde. Den Übergang von der Wissenschaft zum Unternehmen federten wir zunächst etwas ab durch eine Interimslösung unter dem Dach der Technologietransferinitiative (TTI) der Stuttgarter Universität. Just einen Tag vor meinem Geburtstag im Jahre 2006 war es dann schließlich so weit und wir gründeten daraus ein eigenes Unternehmen, die Axivion GmbH⁵. Unser Spin-Off entwickelt und vermarktet die Forschungsprototypen der Universitäten als marktreife Produkte. Dabei bilden die drei Partner – die Universitäten und Axivion – ein eng zusammenarbeitendes Team, durch das Forschungsergebnisse in die Praxis transferiert und umgekehrt neue Forschungsthemen aus der Praxis aufgegriffen werden können. Für die Universitäten ist ein solcher Spin-Off von großem Nutzen, weil wir Firmen viel besser vom Nutzen einer Kooperationen mit uns überzeugen können. Die Universitäten sind kaum in der Lage, den professionellen Support für Werkzeuge zu leisten, Benutzerhandbücher zu schreiben und den Aufwand zu investieren, mehr als nur Prototypen zu bauen. Außerdem haben wir auf diese Weise ein besseres Ohr für reale Probleme in der Industrie, die für uns zu einem Forschungsthema werden können. Aber nicht zuletzt ist es auch einfach die Freude und Motivation, die wir daraus schöpfen, dass unsere Forschung auch eines Tages bei den Programmierern in der Praxis ankommt. Die ersten beiden sehr erfolgreichen Jahre von Axivion haben das Potential dieser Zusammenarbeit demonstriert.

11 WSR und Bauhaus

Die Entwicklung von *Bauhaus* ist eng an die vom WSR geknüpft. Mit dem WSR haben die Organisatoren vor zehn Jahren eine feste Institution für den Austausch zum Thema Reengineering im deutschsprachigen Raum etabliert. Wir haben jedes Jahr die Gelegenheit genutzt, unsere Forschung dort zu diskutieren. Erfreulich ist, dass der WSR kein reiner Wissenschaftlerzirkel geworden ist. Ein großer Teil der Teilnehmer kommt aus der Industrie sowohl von Unternehmen, die Methoden und Werkzeuge für die Software-Evolution anbieten, als auch von Unternehmen, die sich in ihrer Praxis dem Problem der Software-Evolution aktiv stellen. Auf diese Weise kommen wir mit den Menschen ins Gespräch, deren Probleme wir lösen wollen. Die Organisatoren des WSR haben es verstanden, alle zu Wort kommen zu lassen und den Raum für Diskussionen zu schaffen. Die Gespräche im Hörsaal des ehemaligen Heims für Damen höherer Stände finden ihre Fortsetzung bei den festen Ritualen, die zum WSR gehören: dem gemeinsamen Spaziergang und dem Besuch des Eiscafés. Das stetige Wachstum, das der WSR erfahren hat, lässt erwarten, dass wir diesen Ritualen auch in den kommenden zehn Jahren nachkommen können.

⁵<http://www.axivion.com>

Literatur

- [Bel02] Stefan Bellon. Vergleich von Techniken zur Erkennung duplizierten Quellcodes. Diplomarbeit Nr. 1998, Universität Stuttgart, Institut für Softwaretechnologie, September 2002.
- [BKA⁺07] Stefan Bellon, Rainer Koschke, Giulio Antoniol, Jens Krinke und Ettore Merlo. Comparison and Evaluation of Clone Detection Tools. *IEEE Computer Society Transactions on Software Engineering*, 33(9):577–591, September 2007.
- [BLE⁺08] Jim Buckley, Andrew LeGear, Chris Exton, Ross Cadogan, Trevor Johnston, Bill Looby und Rainer Koschke. Encapsulating Targeted Component Abstractions Using Software Reflexion Modelling. *Journal on Software Maintenance and Evolution*, 2008. Accepted for publication.
- [CKS05] Andreas Christl, Rainer Koschke und Margaret-Anne Storey. Equipping the Reflexion Method with Automated Clustering. In *Working Conference on Reverse Engineering*, Seiten 89–98. IEEE Computer Society Press, November 2005.
- [CKS07] Andreas Christl, Rainer Koschke und Margaret-Anne Storey. Automated Clustering to Support the Reflexion Method. *Journal of Systems and Software*, 49(3):255–274, 2007. Special Issue on WCRE 2005.
- [Eis98] Thomas Eisenbarth. GropiusSE, Eine Resource Flow Graph Bibliothek in Ada95 für das Speichern und Aufbereiten von Reengineeringinformationen. Studienarbeit Nr. 1663, Universität Stuttgart, 1998.
- [EKS00] T. Eisenbarth, R. Koschke und D. Simon. Herleitung der Feature-Komponenten-Korrespondenz mittels Begriffsanalyse. In *Proceedings of 1. Deutscher Software-Produktlinien Workshop (DSPL-1)*, Kaiserslautern, Seiten 63–68, November 2000.
- [EKS01a] Thomas Eisenbarth, Rainer Koschke und Daniel Simon. Aiding Program Comprehension by Static and Dynamic Feature Analysis. In *International Conference on Software Maintenance*, Seiten 602–611. IEEE Computer Society Press, November 2001.
- [EKS01b] Thomas Eisenbarth, Rainer Koschke und Daniel Simon. Derivation of Feature Component Maps by means of Concept Analysis. In *European Conference on Software Maintenance and Reengineering*, Seiten 176–180. IEEE Computer Society Press, März 2001.
- [EKS01c] Thomas Eisenbarth, Rainer Koschke und Daniel Simon. Feature-Driven Program Understanding Using Concept Analysis of Execution Traces. In *International Workshop on Program Comprehension*, Seiten 300–309. IEEE Computer Society Press, Mai 2001.
- [EKS02] Thomas Eisenbarth, Rainer Koschke und Daniel Simon. Incremental Location of Combined Features for Large-Scale Programs. In *International Conference on Software Maintenance*, Seiten 273–282. IEEE Computer Society Press, Oktober 2002.
- [EKS03] Thomas Eisenbarth, Rainer Koschke und Daniel Simon. Locating Features in Source Code. *IEEE Computer Society Transactions on Software Engineering*, 29(3):210–224, März 2003.
- [EKV02] Thomas Eisenbarth, Rainer Koschke und Gunther Vogel. Static Trace Extraction. In *Working Conference on Reverse Engineering*. IEEE Computer Society Press, 2002.

- [EKV05] Thomas Eisenbarth, Rainer Koschke und Gunther Vogel. Static Object Trace Extraction for Programs with Pointers. *Journal of Systems and Software*, 77(3):263–284, September 2005.
- [FH79] R.K. Fjeldstad und W.T. Hamlen. Application Program Maintenance Study: Report to our Respondents. In *Proceedings of the GUIDE 48*, Philadelphia, PA, 1979.
- [FKBA07] Pierre Frenzel, Rainer Koschke, Andreas P. J. Brey und Karsten Angstmann. Extending the Reflexion Method for Consolidating Software Variants into Product Lines. In *Working Conference on Reverse Engineering*. IEEE Computer Society Press, Oktober 2007.
- [FKF08] Raimar Falke, Rainer Koschke und Pierre Frenzel. Empirical Evaluation of Clone Detection Using Syntax Suffix Trees. *Empirical Software Engineering*, 2008. Accepted for publication.
- [Gir06] Jean-François Girard. *ADORE-AR: Software Architecture Reconstruction with Partitioning and Clustering*. Phd theses in experimental software engineering, Fraunhofer-Institut Experimentelles Software Engineering IESE, Kaiserslautern; Univ. of Kaiserslautern, Computer Science Department, AG Software Engineering, Band 17, Fraunhofer IRB Verlag, 2006.
- [GK97] Jean-François Girard und Rainer Koschke. Finding Components in a Hierarchy of Modules: a Step towards Architectural Understanding. In *International Conference on Software Maintenance*. IEEE Computer Society Press, 1997.
- [GK00] J.-F. Girard und R. Koschke. A Comparison of Abstract Data Type and Objects Recovery Techniques. *Journal Science of Computer Programming, Elsevier Science*, 6(2–3):149–181, März 2000.
- [GKS97a] Jean-François Girard, Rainer Koschke und Georg Schied. Comparison of Abstract Data Type and Abstract State Encapsulation Detection Techniques for Architectural Understanding. In *Working Conference on Reverse Engineering*, Seiten 66–75. IEEE Computer Society Press, 1997.
- [GKS97b] Jean-François Girard, Rainer Koschke und Georg Schied. A Metric-based Approach to Detect Abstract Data Types and State Encapsulations. In *International Conference on Automated Software Engineering*. IEEE Computer Society Press, 1997.
- [GKS99] Jean-François Girard, Rainer Koschke und Georg Schied. A Metric-based Approach to Detect Abstract Data Types and State Encapsulations. *Journal on Automated Software Engineering, Kluwer Academic Publishers*, 6(4), 1999.
- [KCC06] Rainer Koschke, Gerardo Canfora und Jörg Czeranski. Revisiting the Delta-IC Approach. *Journal of Science of Computer Programming*, 60(2):171–188, 2006.
- [KFF06] Rainer Koschke, Raimar Falke und Pierre Frenzel. Clone Detection Using Abstract Syntax Suffix Trees. In *Working Conference on Reverse Engineering*, Seiten 253–262. IEEE Computer Society Press, 2006.
- [KGW98] Rainer Koschke, Jean-François Girard und Martin Würthner. Intermediate Representations for Reverse Engineering. In *Working Conference on Reverse Engineering*, Seiten 241–250. IEEE Computer Society Press, 1998.
- [Kos99] Rainer Koschke. *Atomic Architectural Component Recovery for Program Understanding and Evolution*. Dissertation, Universität Stuttgart, Institut für Softwaretechnologie, 1999.

- [Kos00] Rainer Koschke. Vorlesungen zum Thema Software-Reengineering. In 2. *Workshop Software-Reengineering*, Seiten 3–7, Bad Honnef, Mai 2000. Universität Koblenz-Landau. Fachberichte Informatik, Nr. 8/2000.
- [Kos05] Rainer Koschke. Rekonstruktion von Software-Architekturen: Blickwinkel, Sichten, Ansichten und Aussichten. *Informatik – Forschung und Entwicklung, Springer Verlag*, 19(3), April 2005.
- [Kos06] Rainer Koschke. Konsolidierung von Software-Varianten in Software-Produktlinien. *Softwaretechnik Trends*, 26(2), Mai 2006. Beiträge des 8. Workshops 'Software Reengineering' (WSR 2006).
- [Kos07] Rainer Koschke. Survey of Research on Software Clones. In Rainer Koschke, Ettore Merlo und Andrew Walenstein, Hrsg., *Duplication, Redundancy, and Similarity in Software*, number 06301 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. Dagstuhl.
- [Kos08] Rainer Koschke. *Software Evolution*, Kapitel Identifying and Removing Software Clones, Seiten 15–39. Springer Verlag, 2008. Editors: Serge Demeyer und Tom Mens.
- [KQ05] Rainer Koschke und Jochen Quante. On Dynamic Feature Location. In *International Conference on Automated Software Engineering*, Seiten 86–95. ACM Press, 2005.
- [KS03] Rainer Koschke und Daniel Simon. Hierarchical Reflexion Models. In *Working Conference on Reverse Engineering*, Seiten 36–45. IEEE Computer Society Press, November 2003.
- [KS04] Rainer Koschke und Daniel Simon. Symphony Fallstudie: Hierarchische Reflexion Modelle. *Softwaretechnik Trends*, 24(2), 2004.
- [LP07a] Felix Lösch und Erhard Plödereder. Optimization of Variability in Software Product Lines. In *Proceedings of the 11th International Software Product Line Conference (SPLC)*, September 2007.
- [LP07b] Felix Lösch und Erhard Plödereder. Restructuring Variability in Software Product Lines using Concept Analysis of Product Configurations. In *European Conference on Software Maintenance and Reengineering*, Seiten 159–168. IEEE Computer Society Press, März 2007.
- [MBKM08] Thilo Mende, Felix Beckwermert, Rainer Koschke und Gerald Meier. Supporting the Grow-and-Prune Model in Software Product Lines Evolution Using Clone Detection. In *European Conference on Software Maintenance and Reengineering*. IEEE Computer Society Press, 2008. Accepted for publication.
- [MNS01] Gail C. Murphy, David Notkin und Kevin J. Sullivan. Software Reflexion Models: Bridging the Gap between Design and Implementation. *IEEE Computer Society Transactions on Software Engineering*, 27(4):364–380, April 2001.
- [QK06] Jochen Quante und Rainer Koschke. Dynamic Object Process Graphs. In *European Conference on Software Maintenance and Reengineering*, Seiten 81–90. IEEE Computer Society Press, 2006.
- [QK07] Jochen Quante und Rainer Koschke. Dynamic Protocol Recovery. In *Working Conference on Reverse Engineering*, Seiten 219–228. IEEE Computer Society Press, Oktober 2007.

- [QK08] Jochen Quante und Rainer Koschke. Dynamic Object Process Graphs. *Journal of Systems and Software*, 81(4):481–501, März 2008. Special Issue for IEEE Conference on Software Maintenance and Reengineering (CSMR) 2006.
- [Qua07] Jochen Quante. Online Construction of Dynamic Object Process Graphs. In *European Conference on Software Maintenance and Reengineering*, Seiten 113–122. IEEE Computer Society Press, 2007.
- [Raz06] Aoun Raza. A Review of Race Detection Mechanisms. In *International Computer Science Symposium in Russia CSR*, Seiten 534–543, 2006.
- [Roh98] Jürgen Rohrbach. Erweiterung und Generierung einer Zwischendarstellung für C-Programme. Studienarbeit Nr. 1662, Universität Stuttgart, 1998.
- [RVP06] Aoun Raza, Gunther Vogel und Erhard Plödereder. Bauhaus - A Tool Suite for Program Analysis and Reverse Engineering. In *Reliable Software Technologies, Ada-Europe*, number 4006 in LNCS, Seiten 71–82. Springer Verlag, Juni 2006.
- [Sim06] Daniel Simon. *Lokalisierung von Merkmalen in Softwaresystemen*. Dissertation, Universität Stuttgart, Logos Verlag Berlin, 2006.
- [Sta07a] Stefan Staiger. Reverse Engineering of Graphical User Interfaces Using Static Analyses. In *Working Conference on Reverse Engineering*, Seiten 189–198. IEEE Computer Society Press, 2007.
- [Sta07b] Stefan Staiger. Static Analysis of Programs with Graphical User Interface. In *European Conference on Software Maintenance and Reengineering*, Seiten 252–261. IEEE Computer Society Press, 2007.
- [SVKW07] Stefan Staiger, Gunther Vogel, Steffen Keul und Eduard Wiebe. Interprocedural Static Single Assignment Form. In *Working Conference on Reverse Engineering*, Seiten 1–10. IEEE Computer Society Press, 2007.
- [vDHK⁺04a] Arie van Deursen, Christine Hofmeister, Rainer Koschke, Leon Moonen und Claudio Riva. Symphony: View-Driven Software Architecture Reconstruction. In *IEEE/IFIP Working Conference on Software Architecture*, Seiten 122–132. IEEE Computer Society Press, Juni 2004.
- [vDHK⁺04b] Arie van Deursen, Christine Hofmeister, Rainer Koschke, Leon Moonen und Claudio Riva. Viewpoints in Software Architecture Reconstruction. *Softwaretechnik Trends*, 24(2), 2004.
- [vMV95] Anneliese von Mayrhauser und A. Marie Vans. Program Comprehension During Software Maintenance and Evolution. *IEEE Computer*, 28(8):44–55, 1995.
- [Vog06] Gunther Vogel. Statische Extraktion von Protokollen. In *Workshop on Software Reengineering*, Mai 2006.
- [Wür96] Martin Würthner. Entwurf und Implementierung einer Interndarstellung für die Analyse von Ada-Programmen. Studienarbeit Nr. 1567, Universität Stuttgart, 1996.