Werkzeugunterstützung für evolutionäre Softwareentwicklung

Markus Bauer, Thomas Genßler, Volker Kuttruff und Olaf Seng

Forschungszentrum Informatik Karlsruhe (FZI)
Haid-und-Neu-Str. 10-14
76131 Karlsruhe
{bauer|genssler|kuttruff|seng}@fzi.de

Moderne Softwaresysteme zeichnen sich durch steigende Komplexität aus. Aus diesem Grund wird es immer schwieriger, Systeme von vornherein gut zu entwerfen. Softwaresysteme weisen daher von Anfang an gewisse Entwurfsmängel auf, zu denen sich im weiteren Verlauf des Entwicklungs- und Lebenszyklus weitere gesellen: Software muss allzu oft hastig an neue Anforderungen angepasst werden. Dabei geht die ursprüngliche Struktur der Systeme langsam verloren und es entstehen zerwartete Systeme. Deshalb müssen Softwaresysteme während ihres gesamten Lebenszyklus systematisch auf Schwachstellen in Entwurf und Implementierung untersucht werden. Die gefundenen Schwachstellen müssen mit Hilfe von Restrukturierungen möglichst unmittelbar behoben werden. Unser Ziel ist die durchgängige Methoden- und Werkzeugunterstützung bei der Durchführung dieser Aktivitäten, die durch den heutigen Stand der Technik nicht gewährleistet ist.

1 Einleitung

Moderne Softwareentwicklung ist heute wesentlich durch zwei Trends geprägt: Zum einen werden die entwickelten Systeme immer komplexer, zum anderen müssen Softwaresysteme flexibel an neue Marktsituationen und Anforderungen angepasst werden können, da sie in vielen Fällen das Rückgrat von Unternehmen darstellen – ohne geeignete Softwaresysteme kann heute kein Unternehmen mehr seine Geschäftsprozesse reibungslos abwickeln.

Dies hat zur Folge, dass sich moderne Softwaresysteme über lange Zeiträume hinweg im Einsatz befinden und dabei kontinuierlich erweitert, modifiziert und verbessert werden – wir sprechen in diesem Zusammenhang von *Softwareevolution.* Diese Tatsache wird z.B. in neuen Softwareprozessen wie *eXtreme Programming* berücksichtigt [BECK 1999].

Allerdings ist die Evolution von Softwaresystemen oftmals unverhältnismäßig teuer. Dies liegt vor allem daran, dass die Systeme im Laufe der Zeit zerwartet werden. Eine anfangs noch gute Struktur weicht dem sprichwörtlichen Spaghetti - oder im Zeitalter der Objektorientierung - dem Raviolicode. Ursachen für dieses Phänomen sind vor allem fehlendes Know-How bei den Entwicklern, die in späteren Phasen zum Entwicklungsteam stoßen, unvollständige oder schlechte Dokumentation des ursprünglichen Entwurfs, Nachlässigkeit sowie Zeitdruck während der Integration neuer Funktionalität.

2 Ziele

Diese Situation lässt sich verbessern, wenn Methoden und Techniken aus den folgenden Bereichen verzahnt und Softwareentwicklern zur Verfügung gestellt werden :

- Werkzeuggestützte Problemidentifikation: Nutzung von Analysen, Metriken und Heuristiken zur Identifikation von Schwachstellen im Design und der Implementierung von Softwaresystemen.
- Ableitung der notwendigen Restrukturierungsschritte ausgehend von den entdeckten Schwachstellen und unter Verwendung von Expertenwissen.
- Automatisierte Restrukturierung der Systeme zur Behebung der zuvor identifizierten Schwachstellen.

Um diese Techniken praktikabel zu machen, müssen folgende Gesichtspunkte beachtet werden:

- Durchgängige Methoden- und Werkzeugunterstützung, beginnend bei der Identifikation von Problemstellen, über die Ableitung von Restrukturierungsmaßnahmen bis hin zur Durchführung der Restrukturierungsmaßnahmen.
- Formalisierung des Expertenwissens, welches erforderlich ist, um aus Problemstellen geeignete Restrukturierungsmaßnahmen abzuleiten.

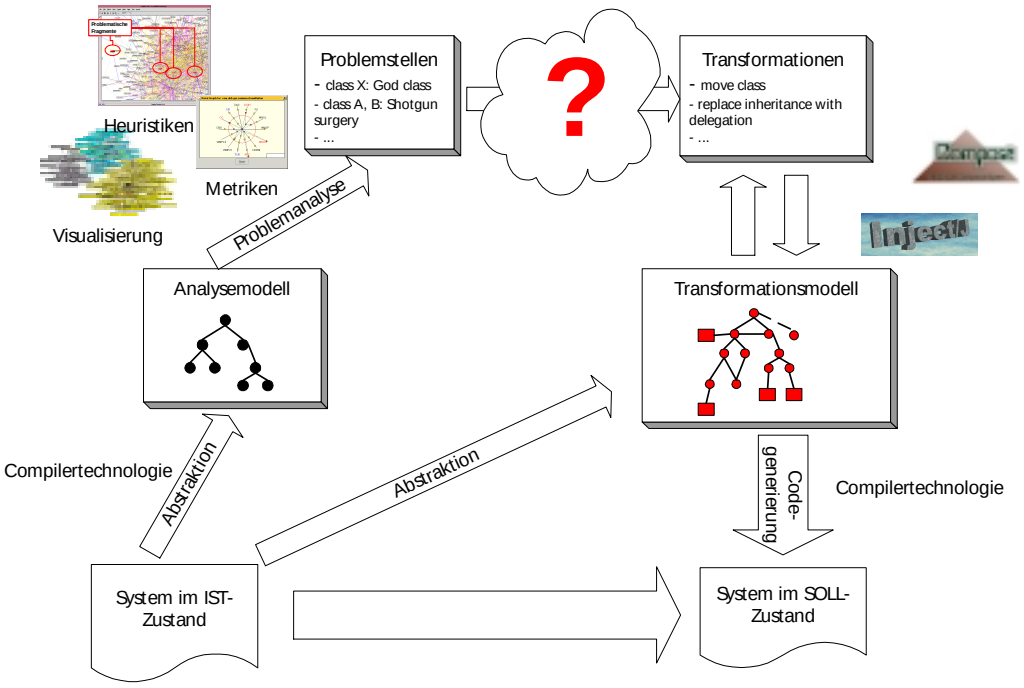

Abbildung 1: Vorgehensweise zur Behebung von Problemstellen in Softwaresystemen

- Integration von Nutzerinteraktionen während der einzelnen Prozessschritte. Eine vollständige Automatisierung des Gesamtvorgehens ist nicht möglich, weil die Befunde aus der Problemidentifikation immer von Fachexperten beurteilt werden müssen.
- Skalierbarkeit der Methoden und Werkzeuge für große Systeme. Dies erfordert geeignete Abstraktionsmechanismen, so dass Softwareingenieure Analysen und Restrukturierungen stets anhand geeigneter Sichten auf das System durchführen können, ohne dass sie dabei von irrelevanten Detailinformationen erschlagen werden.
- Erweiterbarkeit der Werkzeuge und Methoden hinsichtlich neuer Designprobleme

3 Bestandsaufnahme

Es existieren zahlreiche Arbeiten zu Techniken, die dazu herangezogen werden können, Schwachstellen in Softwaresystemen zu identifizieren. Allerdings liefern die meisten dieser Ansätze nur Symptome, die darauf hinweisen, dass etwas im Design bzw. der Implementierung der System nicht stimmt - beispielsweise Arbeiten aus dem Bereich der Qualitätsanalyse mit Hilfe von Metriken [KIDD 1994]. Diese sind bis auf einige wenige Ausnahmen ([LEWERENTZ 2001]) nicht in der Lage, konkrete Probleme zu benennen und die Ursache für schlechte Messwerte zu identifizieren. Dies ist jedoch die Voraussetzung dafür, solche Probleme werkzeuggestützt beseitigen zu können.

Im Bereich Software-Restrukturierung beschäftigen sich die meisten Arbeiten damit, wie problembehaftete Stellen (*bad smells*) in Softwaresystemen systematisch mit Hilfe elementarer Restrukturierungsoperationen *(Refactorings)* behoben werden können [FOWLER 1999]. Diese Ansätze verlassen sich allerdings weitestgehend auf die Intuition der Softwareentwickler, wenn es darum geht, problembehaftete Stellen zu entdecken. Zudem existieren bislang wenige Werkzeuge, die den Softwareentwickler bei der Durchführung der Restrukturierungsmaßnahmen unterstützen, eine größere Anzahl von *Refactorings* ist bislang fast ausschließlich SMALLTALK vorbehalten [ROBERTS 1997].

Nur einige wenige Arbeiten decken die Lücke zwischen der Identifikation potentieller Schwachstellen und deren Behebung durch Restrukturierungen ab [CASAIS 1992][LIEBERHERR 1988][NEUMANN 2000] und ermöglichen so eine durchgängige Vorgehensweise zur Behebung der Schwachstellen. Diesen Arbeiten ist jedoch gemeinsam, dass sie jeweils nur sehr spezialisierte Problemfälle identifizieren und beheben.

4 Beispiel

Unsere Vision der Verzahnung von Problemidentifikation, Ableitung der notwendigen Restrukturierungsschritte und Problembehebung durch Restrukturierungen demonstrieren wir an-

hand eines Beispiels: Wir illustrieren, wie mit Hilfe von Metriken ein bestimmtes Designproblem zunächst identifiziert und dann durch die Ableitung von Restrukturierungen und deren Anwendung eliminiert werden kann.

Wir suchen nach Klassen, die nicht genau ein Konzept implementieren, sondern mehrere verschiedene Abstraktionen in sich vereinen. Solche Klassen sind erstens relativ groß und besitzen eine hohe Komplexität. Zweitens können ihre Attribute und Methoden in Teilmengen zerlegt werden, die in sich eine hohe Kohäsion aufweisen, aber aus anderen Teilmengen keine Attribute benutzen und keine Methoden aufrufen.

Um Klassen mit diesen Eigenschaften zu bestimmen, können folgende Metriken angewandt werden [MARINESCU 2001]: Die großen, komplexen Klassen können mit der Metrik *Weighted Method Count (WMC)* bestimmt werden, *Tight Class Cohesion (TCC)* findet im nächsten Schritt die Klassen, die nicht in sich zusammenhängend sind.

Die gefundenen Klassen stehen für potentielle Problemstellen. Ob wirklich ein Problem vorliegt, kann nur der Benutzer endgültig entscheiden. Dazu müssen ihm zum einen die betroffenen Klassen mit den zusammengehörigen Attribute und Methoden und die Restrukturierungsvorschläge geeignet präsentiert werden (siehe Abbildung 2). Zum anderen benötigt er Informationen über den Kontext, in dem diese Klasse verwendet wird, d.h. er benötigt eine Art abstrakte Sicht auf das System. Werden die Teilmengen der Attribute und Methoden von jeweils verschiedenen, semantisch nicht oder nur schwach zusammenhängenden Stellen aus aufgerufen, dann ist es wahrscheinlich, dass wirklich zwei Abstraktionen in einer Klasse vereint sind.

Hat der Benutzer sich entschieden, eine Klasse in mehrere Klassen zu zerlegen, dann kann er den Lösungsvorschlag überprüfen und gegebenenfalls anpassen. Auf jeden Fall sollte er die neuen Klassen mit aussagekräftigen Namen versehen. Anschließend kann die nötige Transformation mit Hilfe der Restrukturierung *Klasse extrahieren* [FOWLER 1999] erfolgen. In einem ersten Schritt werden die neuen Klassen erzeugt. Falls eine Klasse Attribute und Methoden benötigt, die nach der Zerlegung in anderen Klasse liegen, so müssen neue Attribute erzeugt werden, mit denen die jeweiligen Klassen referenziert werden können. Als nächstes müssen die zu verschiebenden Attribute und Methoden in den neuen Klassen neu erzeugt werden, danach werden die Benutzungsstellen an diese neuen Attribute und Methoden angepasst. Abschließend können die ursprünglichen Methoden und Attribute entfernt werden, da sie nun nicht mehr benötigt werden.

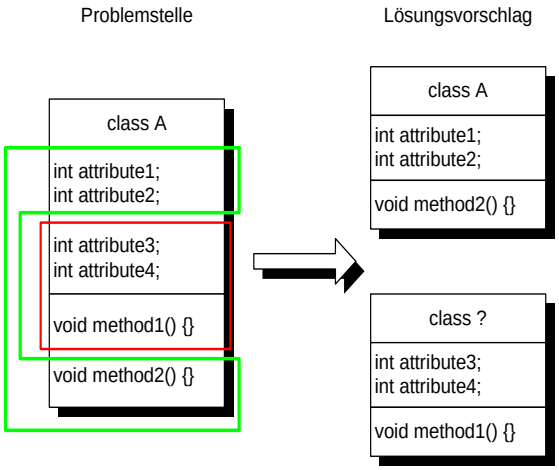

Abbildung 2: Restrukturierungsvorschlag

5 Erste Schritte, weitere Arbeiten

Bisher haben wir wesentliche Elemente einer durchgehenden Vorgehensweise zur Erkennung und Behebung von Entwurfsproblemen in objektorientierten Systemen realisiert:

Wir haben folgende Methoden und entsprechende Werkzeugprototypen zur Identifikation von Problemstellen in objektorientierten Systemen erarbeitet:

- GOOSE: Problemidentifikation mit Hilfe von Heuristiken in Form von Anfragen an eine Faktenbasis, welche die Struktur eines Softwaresystemes repräsentiert. Implementierung von Abstraktionstechniken [CIUPKE 1999][CIUPKE 2002][BAUER 2001].
- PRODEUS: Problemidentifikation mit Hilfe von Metriken. Das Werkzeug unterstützt die Identifikation typischer *Bad Smells* aus [FOWLER 1999] mit Hilfe geeigneter Kombinationen von Software-Produktmetriken [MARINESCU 2001].

Zudem haben wir die theoretische Basis zur automatisierten Restrukturierung von objektorientierten Systemen gelegt und in Form der Werkzeugumgebungen RECODER [RECODER 2002] und INJECT/J in Java [INJECTJ 2002][GENSSLER 2001][SENG 2001] implementiert: Diese Arbeiten umfassen ein Metamodell zur abstrakteren Repräsentation von OO-Systemen und die Spezifikation von Basisopera-

tionen anhand dieses Modells. Komplexere Restrukturierungen können mit Hilfe einer Skriptsprache komponiert werden. Wir arbeiten an der Umsetzung nahezu aller Refactorings aus [FOWLER 1999] und implementieren davon ausgehend komplexere Restrukturierungen.

Unsere nächsten Schritte sehen wir darin, die eben skizzierten Techniken und Werkzeuge zur Problembehebung sowohl auf methodischer, als auch auf implementierungstechnischer Seite zu verzahnen. Die Schwerpunkte liegen dabei in der Vereinheitlichung der zugrunde liegenden Metamodelle zur Darstellung und Manipulation der Implementierungsstrukturen der zu restrukturierenden Systeme und in der Formalisierung des Expertenwissens zur Ableitung von Restrukturierungen für die identifizierten Entwurfsmängel. Zusätzlich müssen dem Benutzer Informationen zugänglich gemacht werden, die zwar keine automatische Bewertung der Problemstellen liefern können, aber dem Benutzer den Kontext der vorgeschlagenen Problemstellen zeigen. Dies kann z.B. durch geeignete, problemorientierte Visualisierungen der Softwarestrukturen geschehen.

6 Zusammenfassung

Die Evolution großer Softwaresysteme kann durch geeignete Methoden und Werkzeuge zur Identifikation von Schwachstellen in Entwurf und Implementierung sowie zur systematischen Restrukturierung zur Behebung dieser Schwachstellen unterstützt werden. Praktikabel werden solche Methoden und Werkzeuge jedoch nur dann, wenn sie in der Lage sind, die komplette Prozesskette von der Problemidentifikation über die Ableitung geeigneter Restrukturierungsmaßnahmen bis hin zu ihrer Durchführung unterstützen. Dieses Papier zeigt, dass dazu jedoch noch eine „Lücke“ in der Prozesskette zwischen Problemidentifikation und -behebung geschlossen werden muss und gibt einen kurzen Überblick über unsere diesbezüglichen Forschungsarbeiten.

7 Literatur

[BAUER 2001] Markus Bauer und Oliver Ciupke, *Objektorientiertes Design unter der Lupe*, 3. German Workshop on Software Reengineering, Bad Honnef, 2001

[BECK 1999] Kent Beck, *Extreme Programming Explained: Embrace Changed*, Addison Wesley, 1999.

[CASAIS 1992] Eduardo Casais, *An Incremental Class Reorganization Approach*, Proceedings of *ECOOP: European Conference on Object-Oriented Programming*, LNCS 615, Springer Verlag, 1992

[CIUPKE 2002] Oliver Ciupke, *Problemidentifikation in objekt-orientierten Softwarestrukturen.* Dissertation an Universität Karlsruhe. Erscheint 2002.

[CIUPKE 1999] Oliver Ciupke, *Automatic Detection of Design Problems in Object-Oriented Reengineering*. In: Proceedings of the *TOOLS USA 99*, 1999.

[FOWLER 1999] Martin Fowler, *Refactoring – Improving the Design of Existing Code*. Addison-Wesley, 1999.

[GENSSLER 2001] Thomas Genssler und Volker Kuttruff, *Werkzeuggestützte Softwareadaption mit Inject/J*, 3. German Workshop on Software Reengineering, Bad Honnef, 2001

[INJECTJ 2002] Thomas Genssler und Volker Kuttruff, *Inject/J Homepage*, http://injectj.fzi.de , 2002

[KIDD 1994] J. Kidd und M. Lorenz. *Object-Oriented Software Metrics.* Prentice-Hall, Englewood Cliffs, NJ, 1994.

[LEWERENTZ 2001] Claus Lewerentz, Frank Simon und Frank Steinbrückner, *Metrics Based Refactoring*. In: *Proceedings of the 5th Conference on Conference on Software Maintenance and Reengineering,* 2001.

[LIEBERHERR 1988] Karl Lieberherr, Ian Holland und Arthur Riel, *Object-Oriented Programming: An Objective Sense of Style*, Proceedings of *OOPSLA: Conference on Object-Oriented Programming Systems, Languages and Applications*, ACM SIGPLAN Notices, ACM Press, 1988.

[MARINESCU 2001] Radu Marinescu, *Detecting Design Flaws Using Metrics*. In: Proceedings of the TOOLS USA 01, 2001.

[NEUMANN 2000] Rainer Neumann, *Vermeidung spezialisierungsbedingter Probleme in objektorientierten Systemen*, Dissertation an der Universität Karlsruhe, 2000.

[RECODER 2002] Andreas Ludwig, Recoder Homepage, http://recoder.sourceforge.net, 2002.

[ROBERTS 1997] Don Roberts, John Brant und Ralph E. Johnson, *A Refactoring Tool for Smalltalk*, In: *Journal of Theory and Practice of Object Systems (TAPOS)*, vol. 3, no. 4, 1997

[SENG 2001] Olaf Seng, Thomas Genssler, Volker Kuttruff und Benedikt Schulz, *Adaptives Programmieren mit Inject/J*, 3. German Workshop on Software Reengineering, Bad Honnef, 2001