24 Automatisierte Behebung von Strukturproblemen in objektorientierten Systemen

Adrian Trifu
Olaf Seng
Thomas Genssler
FZI Forschungszentrum Informatik, Karlsruhe, Germany
{trifu,seng,genssler}@fzi.de

24.1 Einführung

In diesem Papier stellen wir einen integrierten, qualitätsgetriebenen und werkzeugunterstützten Ansatz zur Evolution objektorientierter Systeme vor, der dazu beiträgt die existierende Lücke zwischen der Suche von Strukturproblemen und ihrer Behebung zu schließen. Unser Ansatz basiert auf dem neuen Konzept der "Correction Strategies" [TD03], und existierenden Arbeiten zur Problemerkennung und zur Codetransformation [Ciu99], [GK03].

24.2 Ansatz

Unser Ansatz besteht aus drei Phasen. In der ersten Phase - der Problemerkennung und -analyse - wird zuerst ein Qualitätsziel angegeben, und Strukturprobleme, die dieses Ziel negativ beeinflussen können werden vorgeschlagen. Um die Strukturprobleme im Quelltext zu suchen und zu finden, benutzen wir bereits existierende Methoden und Werkzeuge. Sobald ein oder mehrere Strukturprobleme gefunden und bestätigt wurden, wird in der Lösungsanalyse-

phase mit Hilfe der “Correction Strategies” ein Restrukturierungsplan erstellt, der in der abschließenden Restrukturierungsphase umgesetzt wird. Im Folgenden werden wir auf diese drei Phasen näher eingehen.

24.2.1 Problemerkennung und -analyse

Die Auswahl der Typen von Strukturproblemen, nach denen gesucht werden soll, ist von einem vom Benutzer vorgegebenen Qualitätsziel abhängig. Ein Qualitätsziel besteht dabei aus einer Gewichtung vorgegebener Qualitätsfaktoren wie z.B. Erweiterbarkeit, Flexibilität, Verständlichkeit ...

Für jeden Strukturproblemtyp, den wir erkennen können, wurde von uns vorgegeben, welchen Einfluss ein Auftreten eines solchen Problems auf die Qualitätsfaktoren hat. Qualitätsfaktoren und Einfluss der Strukturprobleme auf diese wurden vorerst einmal ad hoc bestimmt. Genauere Werte könnte man durch die Durchführung von Fallstudien und Experimenten und Recherchen in der Literatur gewinnen, was momentan im Rahmen des Forschungsprojektes QBench[1] geschieht.

Oft ist es nicht möglich, alle gefundenen Strukturprobleme zu entfernen, da die Behebung jedes einzelnen zwar werkzeuggestützt aber nicht vollautomatisch erfolgen kann, und somit relativ viel Aufwand benötigen kann. Die Gewichtung der Strukturprobleme macht es möglich, zuerst diejenigen zu beheben, die den größten Einfluss auf das Qualitätsziel haben.

Abhängig vom Kontext der Software (z.B. Echtzeitsystem, Prototyp, ...) kann dem Benutzer ein vordefiniertes Qualitätsziel vorgeschlagen werden, das im allgemeinen für Anwendungen des gewählten Kontexts geeignet ist. Der Kontext hat zudem Einfluss auf die Auswirkungen der Problemstrukturen auf die Qualitätsfaktoren und die Schwellwerte, die zur automatischen Erkennung der Strukturprobleme verwendet werden. Die automatisch gefundenen Strukturprobleme müssen manuell als solche bestätigt werden, damit im nächsten Schritt eine Lösung für jedes einzelne gefunden werden kann.

24.2.2 Lösungsanalyse

Ziel der Lösungsanalyse ist es, eine Menge von Restrukturierungen abzuleiten, die das vorhandene Strukturproblem entfernen und die gewählten Qualitätsfaktoren verbessern. Hierzu werden sogenannte “Correction Strategies” eingesetzt. Eine “Correction Strategy” ist eine strukturierte Beschreibung, die einem gegebenen Strukturproblem eine Menge von möglichen Lösungen zuordnet. Beispielsweise kann eine *Gottklasse* sowohl durch ein Aufteilen der Klasse in eine oder mehrere Klassen als auch durch reines Verschieben von Methoden und Attributen entfernt werden. Jede dieser potentiellen Lösungen kann dabei aus mehreren Teillösungen und Entscheidungen zwischen diesen aufgebaut sein.

Da für jede der möglichen (Teil)Lösungen die Auswirkungen auf die Qualitätsfaktoren bekannt ist, kann Schritt für Schritt die Lösung berechnet werden, die einen möglichsten guten Einfluss auf die Qualitätsfaktoren hat. “Correction Strategies können mit einem Expertensystem verglichen werden.

24.2.3 Reorganisation

Zur Reorganisation wird ausgehend vom Quelltext ein zweischichtiges Modell aufgebaut. Die obere Schicht bietet eine konzeptuelle Sicht auf objektorientierte Sprachen. Diese Sicht enthält typische Entwurfseinheiten wie Klassen und Methoden und Beziehungen zwischen diesen wie z.B. Methodenaufrufe und Attributzugriffe. Auf dieser Ebene sind eine Menge von Transformations- und Analyseoperationen definiert. Mit Hilfe der Analyseoperationen können Modellelemente mit bestimmten Eigenschaften ausgewählt und Vor- und Nachbedingungen von Transformationen spezifiziert und überprüft werden.

Konzeptuelle Transformationen sind Grundoperatoren zur Modifikation eines Modells. Sie reichen von einfachen Veränderungen (z.B. ein Attribut zu einer Klasse hinzufügen) bis zu komplexeren Refactorings (z.B. eine Methode von einer Klasse in eine andere zu verschieben). Diese konzeptuellen Transformationen sind atomar und garantieren Korrektheit in Bezug auf Syntax und statische Semantik der zugrunde liegenden Programmiersprache. Es können weitere Vor- und Nachbedingungen angegeben werden, mit denen man zusätzliche Nebenbedingungen formulieren, die z.B. Verhaltensbewahrung garantieren.

Die untere Schicht stellt die Verbindung zur Semantik der tatsächlichen Sprache her. Die konzeptuellen Einheiten werden auf die entsprechenden Einheiten der Sprache abgebildet, und es werden sprachabhängige Vor- und Nachbedingungen bereitgestellt. Momentan ist die untere Schicht für JAVA spezifiziert und implementiert, eine Unterstützung von C#[2] ist in Arbeit.

Zur Spezifikation komplexer Refactorings kann eine Skriptsprache namens Inject/J benutzt werden. Diese Skriptsprache erlaubt die einfache Navigation in einem Modell, die Auswahl bestimmter Modellelemente anhand ihrer Eigenschaften, und die Ausführung von Transformationen.

24.3 Werkzeugunterstützung

Jede der Phasen unseres Ansatzes wird durch Werkzeuge unterstützt. Der Advanced Refactoring Wizard (ARW) dient als Integrationsplattform für unsere Problemerkennungs- und Refactoring-Technologie und führt den Nutzer durch den gesamten Prozess.

Zur Problemsuche benutzen wir ein Opensource-Werkzeug namens jGoose[3], mit dem momentan nach 18 Strukturproblemen wie z.B. Gottklassen oder Flaschenhalsklassen gesucht werden kann. Die Lösungsanalyse er-

[1] `http://www.qbench.de`
[2] `http://recoder-cs.sf.net`
[3] `http://jgoose.sf.net`

folgt mit einem prototypischen Werkzeug, das momentan "Corrections Strategies" für vier Strukturprobleme unterstützt. Zur Behebung der Strukturprobleme verwenden wir ein Werkzeug namens Inject/J[4]. Alle drei Werkzeuge wurden am FZI entwickelt. Zur Zeit können wir mit unserer Werkzeugkette den kompletten Prozess für Java abdecken. Die Problemsuche und die Lösungsanalyse sind sprachunabhängig für objektorientierte Programmiersprachen gehalten, vorausgesetzt ein entsprechender Faktenextraktor existiert. Momentan unterstützen wir hier JAVA und C++. Um die Reorganisation für mehrere Sprachen zu unterstützen ist natürlich viel mehr Aufwand erforderlich. Zur Zeit arbeiten wir an einer Unterstützung von C#.

24.4 Fallstudie

Unser Werkzeug konnte auf eine in JAVA geschriebene Fallstudie kleiner bis mittlerer Größe (63 Klassen, 34000 LOC) erfolgreich angewandt werden. Eine der in diesem System gefundenen Gottklassen erwies sich als besonders problematisch. Sie enthielt 71 Methoden und ca. 1400 LOC, und stand in direkter zyklischer Abhängigkeit zu einer anderen Klasse. Bei der Lösungsanalyse musste zwischen den Möglichkeiten Klasse aufteilen und Methoden/Attribute verschieben entschieden werden. Da zusätzlich Methoden der Gottklasse als deplatzierte Methoden identifiziert werden konnten, und das Anlegen einer neuen Klasse die Komplexität erhöht hätte, wurde entschieden einen Teil der Methoden zu verschieben. In diesem Fall konnte das Verschieben mit Inject/J vollautomatisch durchgeführt werden, da es sich bei den zu verschiebenden Methoden um statische Methoden handelte. Die direkte zyklische Abhängigkeit konnte damit ebenfalls beseitigt werden.

24.5 Zusammenfassung und Ausblick

In diesem Papier haben wir eine integrierte, qualitätsgetriebene Methode zur Evolution objektorientierter Softwaresysteme vorgestellt. Ein wichtiger Bestandteil unserer Methode sind die sogenannten "Correction Strategies", mit denen die sichere Behebung von Strukturproblemen unter Berücksichtigung eines vorgegebenen Qualitätsziels geplant werden kann.

Einer unserer nächsten Schritte besteht in der Ausarbeitung von "Correction Strategies" für weitere Strukturprobleme, und eine Evaluation anhand weiterer realer Fallstudien. Zudem arbeiten wir an einer verbesserten, feingranulareren Version unseres Qualitätsmodells. Ein weiterer Punkt ist die Frage nach einer globalen Optimierung. Momentan betrachten wir jedes Strukturproblem getrennt. Es stellt sich jedoch die Frage, ob mit isolierter Behebung eines oder mehrerer Strukturprobleme eine globale Verbesserung eines Softwaresystems erreicht werden kann.

Literaturverzeichnis

[Ciu99] CIUPKE, OLIVER: *Automatic Detection of Design Problems in Object-Oriented Reengineering*. in *Proceedings of Technology of Object-Oriented Languages and Systems - TOOLS 30*, pp. 18–32, 1999.

[GK03] GENSSLER, THOMAS and VOLKER KUTTRUFF: *Source-to-Source Transformation In The Large*. in *Proceedings of the Joint Modular Language Conference*. Springer LNCS, August 2003.

[TD03] TRIFU, ADRIAN and IULIAN DRAGOŞ: *Strategy Based Elimination of Design Flaws in Object-Oriented Systems*. in DEMEYER, SERGE, STÉPHANE DUCASSE and KIM MENS eds.: *Proceedings of the Fourth Workshop on Object–Oriented Reengineering, in Conjunction with ECOOP 2003*, pp. 55–61, 2003.

[4] http://injectj.sf.net