Extract Method Refactoring-Vorschläge automatisch ableiten*

Roman Haas
Technische Universität München
Lichtenbergstr. 8, 85748 Garching
roman.haas@tum.de

Benjamin Hummel
CQSE GmbH
Lichtenbergstr. 8, 85748 Garching
hummel@cqse.eu

Zusammenfassung

Das Extract Method Refactoring ist eine gebräuchliche Art, zu lange Methoden im Code zu kürzen. Bevor aber Tool-Support für das Refactoring genutzt werden kann, müssen Entwickler zunächst geeignete Codezeilen identifizieren – ein zeitintensives und fehleranfälliges Unterfangen. Dieses Paper präsentiert einen Ansatz, der automatisch Vorschläge für Extract Method Refactorings generiert. Dazu werden zunächst alle gültigen Extract Method Refactorings berechnet und anschließend diejenigen vorgeschlagen, die die Komplexität des Codes am meisten verringern. Der Ansatz beruht auf einer Scoring-Funktion, deren Gewichtung durch Learning-to-Rank-Verfahren bestimmt wurde. Wir stellen in diesem Paper auch die wichtigsten Ergebnisse aus den Learning-to-Rank-Verfahren anhand von drei Forschungsfragen vor.

1 Einleitung

Eine lange Methode ist ein *Bad Smell* in einem Softwaresystem [1] und verschlechtert Lesbarkeit, Verständlichkeit und Testbarkeit des Codes. Um lange Methoden zu kürzen werden häufig Extract Method Refactorings genutzt.

Aktuelle IDEs unterstützen Entwickler bei der Durchführung von Extract Method Refactorings. Hierbei müssen nur die zu extrahierenden Codezeilen ausgewählt, die Refactoring-Funktion aufgerufen und ein Name für die neue Methode vergeben werden. Das Extrahieren der Methode wird dann automatisch durchgeführt. Dennoch sind diese Refactorings zeitintensiv und fehleranfällig, weil der Entwickler vorher infragekommende Kandidaten bestimmen muss.

Wir stellen in diesem Paper einen Ansatz vor, wie Extract Method Refactoring-Vorschläge für zu lange Methoden automatisch generiert werden können. Dabei werden zunächst alle möglichen Extract Method Refactorings generiert und die besten Vorschläge anhand einer Scoring-Funktion bestimmt. Die Parameter der Scoring-Funktion wurden durch die Anwendung von Learning-to-Rank-Verfahren ermittelt. In einem weiteren Schritt wurde die Scoring-Funktion auf Basis der Erkenntnisse aus dem Lernverfahren vereinfacht. Der Ansatz wurde in einem Tool zur kontinuierlichen Softwarequalitätsanalyse für die Sprachen Java und C# implementiert.

*Das diesem Artikel zugrundeliegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen Q-Effekt, 01IS15003A gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

2 Ansatz

Unser Ansatz besteht aus zwei wesentlichen Schritten: Zunächst werden alle validen Refactoring-Möglichkeiten ("Kandidaten") bestimmt und diese anschließend mittels einer Scoring-Funktion gerankt. Die besten Kandidaten werden dann für ein Extract Method Refactoring vorgeschlagen.

2.1 Kandidatengenerierung

Ein Kandidat besteht aus einer Sequenz von Codezeilen, die aus der ursprünglichen Methode in eine neue extrahiert werden kann. Wichtigste Einschränkung ist dabei, dass die zu extrahierende Methode beispielsweise bei Java oder C# nicht mehr als einen Rückgabeparameter benötigen darf. Um unnütze Vorschläge zu vermeiden, werden nur Kandidaten berücksichtigt, die mindestens drei Statements umfassen und bei denen wenigstens drei Statements (inklusive dem Aufruf der extrahierten Methode) in der ursprünglichen Methode verbleiben.

2.2 Scoring-Funktion

Die Scoring-Funktion bewertet alle gültigen Kandidaten hinsichtlich ihres Einflusses auf die Code-Komplexität der zu langen Methode. Kandidaten, die die Komplexität des Codes reduzieren, werden im Allgemeinen besser gerankt, als solche, die die Komplexität kaum senken. Als Komplexitätsindikatoren werden in der Scoring-Funktion die Reduktion der Länge und der Verschachtelung der einzelnen Methoden betrachtet. Außerdem wird die Zahl der benötigten Parameter bei der extrahierten Methode berücksichtigt und Zusatzpunkte auf Basis von strukturellen Informationen (Kommentare und leere Zeilen) vergeben.

Details zu unserem Ansatz finden sich auch in [2].

3 Learning to Rank

Um die Scoring-Funktion zu verbessern und den Einfluss der einzelnen Komponenten besser zu verstehen, wurden zwei Learning-to-Rank-Verfahren ge-

nutzt: SVM-rank von Tsochantaridis et al. [5] und ListMLE von Xia et al. [6].

Die Learning-to-Rank-Tools lernen Gewichtungen für die einzelnen Parameter der Scoring-Funktion. Es werden elf Komplexitätsindikatoren, die Zahl der benötigten Ein- und Rückgabeparameter und sieben strukturelle Informationen als Parameter verwendet.

3.1 Forschungsfragen

Im Rahmen dieses Papers wird auf die folgenden drei Forschungsfragen eingegangen:

- FF1: Welche Parameter sind auf Grundlage der Ergebnisse der Learning-to-Rank-Verfahren besonders wichtig?
- FF2: Wie stabil sind die gelernten Scoring-Funktionen in der 10-fold-cross Evaluierung?
- FF3: Kann die Scoring-Funktion vereinfacht werden?

3.2 Setup

Als Datengrundlage wurden 5-9 zufällig ausgewählte, gültige Refactoring-Vorschläge aus 177 wiederum zufällig ausgewählten zu langen Methoden aus 13 bekannten Open Source Systemen manuell hinsichtlich ihrer Komplexitätsreduktion gerankt.

Wir wendeten ein Kreuzvalidierungsverfahren (10-fold-cross) an, bei dem die Lerndaten in zehn etwa gleich große Mengen aufgeteilt und in zehn Durchläufen je neun der Mengen als Lerndaten und die verbleibende Menge als Testdaten genutzt werden (vgl. auch [4]).

Um die erlernten Scoring-Funktionen hinsichtlich ihrer Ranking-Fähigkeit vergleichen zu können, wurde die Normalized Discounted Cumulative Gain Metrik (NDCG, s. auch [3]) genutzt.

3.3 Ergebnisse und Diskussion

Im Folgenden werden die Ergebnisse erläutert.

FF1: Welche Parameter sind auf Grundlage der Ergebnisse der Learning-to-Rank-Verfahren besonders wichtig? ListMLE lernt eine Scoring-Funktion, die besonders viel Gewicht auf die Reduktion der Länge legt. Die Zahl der benötigten Eingabeparameter hat einen kleinen, aber negativen Einfluss auf den Score. Die strukturellen Informationen sind für das Ranking relevant, bei weitem aber nicht so einflussreich wie die Komplexitätsindikatoren.

SVM-rank zeichnet ein weniger klares Bild. Die Komplexitätsindikatoren und strukturellen Informationen haben einen stark variierenden Einfluss auf das Ranking. Die Zahl der Eingabe- bzw. Rückgabeparameter hat negativen Einfluss auf den Score.

FF2: Wie stabil sind die gelernten Scoring-Funktionen in der 10-fold-cross Evaluierung? Um die Stabilität der gelernten Scoring-Funktionen bewerten zu können, haben wir den Variationskoeffizienten für alle Parameter der Scoring-Funktion berechnet. Der Mittelwert liegt hier für ListMLE bei 0,0087 und für SVM-rank bei 22,522. Der schlechteste Variationskoeffizient bei ListMLE ist immer noch besser als der beste Variationskoeffizient bei SVM-rank.

Die einzelnen gelernten Scoring-Funktionen von SVM-rank lassen sich also kaum für einen Einsatz in der Realität verallgemeinern. Die Variationskoeffizienten für ListMLE sind nahe Null und deuten damit auf verlässlichere Ergebnisse hin.

FF3: Kann die Scoring-Funktion vereinfacht werden? In weiterführenden Experimenten konnten wir die Zahl der Parameter von zwanzig auf bis zu drei reduzieren, ohne die Ranking-Performance der Scoring-Funktion deutlich zu verringern.

4 Implementierung

In Teamscale[1], einem Werkzeug zur kontinuierlichen Softwarequalitätsanalyse, können Extract Method Refactoring-Vorschläge für zu lange Methoden aus Java- und C#-Projekten angezeigt werden.

Dabei werden die zu extrahierenden Zeilen hervorgehoben und die Variablen, die in der extrahierten Methode als Parameter dienen, farblich markiert.

5 Zusammenfassung

Zu lange Methoden lassen sich mittels Extract Method Refactorings kürzen. Um Zeit zu sparen und Fehler zu vermeiden, sind Refactoring-Vorschläge für Entwickler nützlich. In diesem Paper haben wir einen Ansatz hierfür vorgestellt, der eine gelernte Scoring-Funktion nutzt und in Teamscale, einem Tool zur Softwarequalitätsanalyse, implementiert ist.

Literatur

[1] M. Fowler. *Refactoring : Improving the Design of Existing Code*. Addison-Wesley object technology series. Addison-Wesley, Reading, PA, 1999.

[2] R. Haas and B. Hummel. Deriving extract method refactoring suggestions for long methods. In *SWQD*, 2016.

[3] L. Hang. A short introduction to learning to rank. *IEICE Transactions on Information and Systems*, 94(10):1854–1862, 2011.

[4] C. Sammut, editor. *Encyclopedia of machine learning*. Springer, New York, 2011.

[5] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

[6] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li. Listwise approach to learning to rank: Theory and algorithm. In *25th ICML*, 2008.

[1] `teamscale.eu`