Bewertung automatisch erkannter Ausprägungen von Software-Mustern

Dietrich Travkin
Fachgebiet Softwaretechnik, Institut für Informatik, Universität Paderborn
travkin@uni-paderborn.de

1 Einleitung

Beim Reverse Engineering werden unter anderem Werkzeug-gestützte Verfahren zu Erkennung von Software-Mustern wie Design Patterns [GHJV95] eingesetzt, um Teile des ursprünglichen Designs aus dem Quellcode wiederzugewinnen. Aufgrund der meist unpräzisen Beschreibung von Software-Mustern können False Positives[1] in den Suchergebnissen nicht ausgeschlossen werden.

Da die meisten bisherigen Mustererkennungsverfahren nur absolute Aussagen machen (eine Musterausprägung ist entweder vorhanden oder nicht), stehen dem Reverse Engineer keine Informationen über die Qualität der Suchergebnisse zur Verfügung. Die Identifikation der relevanten Suchergebnisse kann bei großen Softwaresystemen und entsprechend vielen Ergebnissen mühsam und zeitraubend werden. Bei den wenigen Mustererkennungsverfahren, welche die Suchergebnisse bewerten [NWW03, Wen05], weist das Bewertungsverfahren erhebliche Mängel auf (Details in [Tra06]). Die Bewertungen haben nur eine geringe Aussagekraft.

Im Rahmen meiner Diplomarbeit [Tra06] habe ich ein neues Verfahren zur automatischen Bewertung der Suchergebnisse einer Mustererkennung entwickelt. Dieses ermöglicht eine Sortierung der Funde nach Relevanz und wurde prototypisch in dem CASE-Tool Fujaba [Fuj06] realisiert.

2 Vorgehen

An der Universität Paderborn wurde ein Mustererkennungsverfahren entwickelt, bei dem Software-Muster exemplarisch durch spezielle UML-Objektdiagramme beschrieben und im abstrakten Syntaxgraphen des zu untersuchenden Softwaresystems gesucht werden [NSW+02, NWW03]. Die Suchergebnisse werden nach dem in diesem Papier vorgestellten Verfahren bewertet.

[1] Stellen, die als Musterausprägungen erkannt werden, jedoch keine sind.

2.1 Mustererkennung und -spezifikation

Der Algorithmus zur Erkennung von Software-Mustern führt eine Teilgraphensuche auf dem abstrakten Syntaxgraphen der zu untersuchenden Software durch. Software-Muster werden durch spezielle Graphgrammatikregeln spezifiziert, die den zu suchenden Teilgraphen und die bei Erfolg zu erzeugende Annotation beschreiben.

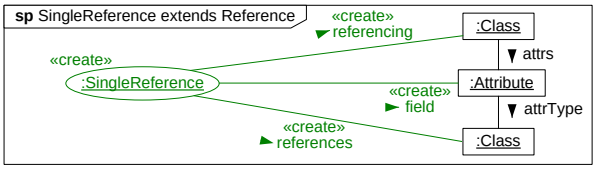

Abbildung 1: Spezifikation einer zu-1-Referenz

Ein Beispiel für die Spezifikation einer zu-1-Referenz ist in der Abbildung 1 zu sehen. Die Knoten stellen Objekte im abstrakten Syntaxgraphen dar. Werden Objekte der Typen Class und Attribute gefunden, die wie abgebildet miteinander verknüpft sind, so wird die Fundstelle durch Erzeugen des mit dem Stereotyp ≪create≫ markierten Objekts und zugehöriger Verknüpfungen annotiert. Neben den Objekttypen und Verknüpfungen können auch Attributeigenschaften angegeben werden.

Aufgrund der meist abstrakten und informellen Beschreibung von Software-Mustern ist es schwierig, die Muster formal zu spezifizieren und dabei alle Implementierungsvarianten jedes Musters zu berücksichtigen. Um die Anzahl der Spezifikationen klein zu halten, fassen Niere et. al. mehrere Implementierungsvarianten in einer Spezifikation zusammen [NWW03]. Dazu werden einige weniger relevante Informationen zu einem Muster und den zugehörigen Implementierungsvarianten weggelassen.

Eine zu-1-Referenz zum Beispiel kann auf mehrere verschiedene Weisen implementiert werden: Unter anderem mit einem privaten Attribut und öffentlichen Zugriffsmethoden für den Lese- und Schreibzugriff oder nur mit einem öffentlichen Attribut ohne Zugriffsmethoden. Die Spezifikation in Abbildung 1 deckt diese beiden Varianten ab, da keine Bedingungen an die Sichtbarkeit des Attributs oder die Existenz von Zugriffsmethoden gestellt werden.

Das Weglassen von Informationen resultiert in unpräzisen Musterspezifikationen. False Positives können nicht ausgeschlossen werden. Zum Beispiel könnte eine zu-n-Referenz als zu-1-Referenz erkannt werden, was auf Quellcode-Ebene (technisch) korrekt wäre, auf Design-Ebene jedoch nicht.

2.2 Bewertung von Fundstellen

Bei dem erarbeiteten Verfahren zur automatischen Bewertung der Suchergebnisse einer Mustererkennung werden die Musterspezifikationen um einige der Informationen erweitert, die bisher ungenutzt blieben. Zum Beispiel kann bei der Spezifikation einer zu-1-

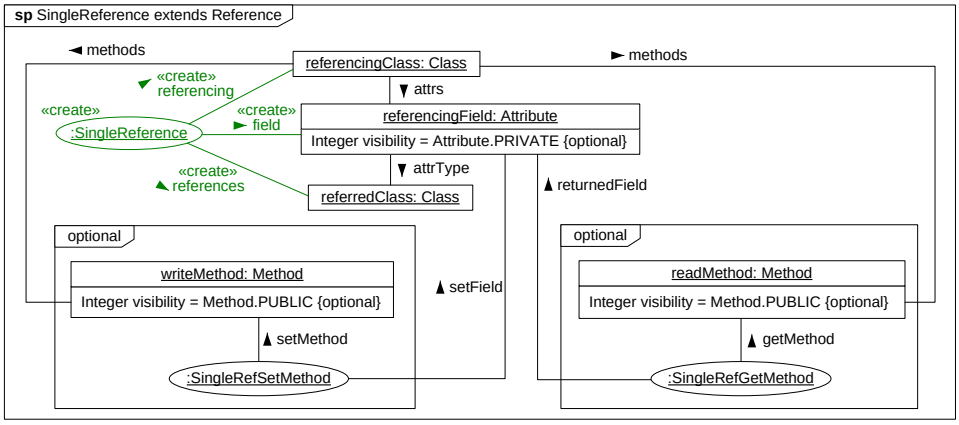

Abbildung 2: Spezifikation einer zu-1-Referenz mit optionalen Elementen

Referenz das Wissen verwendet werden, dass das referenzierende Attribut nach dem Information-Hiding-Prinzip gewöhnlicherweise private-Sichtbarkeit besitzt. Zusätzlich existiert eine meist öffentliche Zugriffsmethode für den Lesezugriff und häufig auch eine für den Schreibzugriff. Diese Eigenschaften treffen nicht auf jede Implementierungsvariante zu, wenn sie jedoch erkannt werden, geben sie einen zusätzlichen Hinweis auf das Vorkommen des gesuchten Musters. Ein False Positive wird somit unwahrscheinlicher.

Damit solche Informationen bei der Mustererkennung und der Bewertung der Suchergebnisse verwenden werden können, wurde die Musterspezifikationssprache um optionale Elemente erweitert. Die oben genannten Eigenschaften können wie in Abbildung 2 in Form von optionalen Teilgraphen und optionalen Attributbedingungen spezifiziert werden. Die Knoten der Typen SingleRefGetMethod und SingleRefSetMethod repräsentieren Zugriffsmethoden, die durch andere Musterspezifikationen beschrieben sind.

Im Gegensatz zu dem bisherigen Mustererkennungsverfahren, bei dem sämtliche in der Spezifikation angegebenen Bedingungen erfüllt sein müssen, sind bei dem neuen Verfahren auch unvollständige Funde möglich. Knoten, Teilgraphen und Bedingungen, die als „optional“ markiert sind, müssen nicht gefunden beziehungsweise erfüllt werden. Der Mustererkennungsalgorithmus versucht jedoch, die Spezifikationen möglichst vollständig zu erfüllen. Je mehr optionale[2] Knoten vorhanden und Bedingungen erfüllt sind, desto höher wird ein Fund bewertet.

Die Bewertung einer Fundstelle, die als Musterausprägung erkannt wurde, beschreibt, wie vollständig die Spezifikation eines Software-Musters erfüllt wird. Dazu werden Knoten, Attributbedingungen und andere Eigenschaften als Bedingungen angesehen und der Anteil der erfüllten Bedingungen berechnet. Vereinfacht kann die Bewertung einer Fundstelle a als der Quotient aus der Anzahl der von a erfüllten und der Anzahl der von a erfüllbaren Bedingungen beschrieben werden. Die Knoten und Bedingungen können gewichtet werden, um auszudrücken, dass bestimmte Eigenschaften besonders wichtig sind.

[2]Die Bedeutung von „optional“ weicht hier von seiner ursprünglichen Bedeutung ab und wird im Sinne von „nicht notwendig aber erwünscht“ verwendet.

3 Zusammenfassung

Es wurde ein Ansatz zur automatischen Bewertung der Suchergebnisse aus einer Mustererkennung vorgestellt. Bei diesem Verfahren können Musterspezifikationen um zusätzliche Informationen in Form von optionalen Knoten und Bedingungen erweitert werden. Die Bewertung einer Fundstelle gibt den Anteil der erfüllten Bedingungen wieder, was die Identifikation der relevanten Suchergebnisse erleichtert.

Rahmen der Arbeit

Meine Diplomarbeit ist an der Universität Paderborn im Fachgebiet Softwaretechnik bei Prof. Schäfer entstanden. Der Zeitrahmen der Arbeit betrug 3 Monate Voll- beziehungsweise 6 Monate Halbzeitarbeit. Im Rahmen der Arbeit war zu untersuchen, wie die Aussagekraft der Suchergebnisse eines (semi-) automatischen Mustererkennungsverfahrens erhöht werden kann. Hierzu sollte die Qualität der Suchergebnisse in Form eines automatischen Bewertungsverfahrens numerisch ausgedrückt werden, ähnlich dem Ranking der Ergebnisse einer Internet-Suchmaschine.

Nach einer Einarbeitung in das Themengebiet und einer anschließenden Recherche wurden existierende Bewertungsverfahren auf ihre Tauglichkeit untersucht und Anforderungen an ein Bewertungsverfahren formuliert. Da die existierenden Verfahren erhebliche Mängel aufwiesen, wurde ein neues Verfahren entwickelt, prototypisch realisiert und an Beispielen erprobt.

Literatur

[Fuj06] Fujaba Development Group. Fujaba Tool Suite. Online unter: http://www.fujaba.de, Stand: August 2006.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides. *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley, 1995.

[NSW+02] Jörg Niere, Wilhelm Schäfer, Jörg P. Wadsack, Lothar Wendehals und Jim Welsh. Towards Pattern-Based Design Recovery. In *Proc. of the* 24^{th} *International Conference on Software Engineering (ICSE), Orlando, Florida, USA*, Seiten 338–348. ACM Press, Mai 2002.

[NWW03] Jörg Niere, Jörg P. Wadsack und Lothar Wendehals. Handling Large Search Space in Pattern-Based Reverse Engineering. In *Proc. of the* 11^{th} *International Workshop on Program Comprehension (IWPC), Portland, USA*, Seiten 274–279. IEEE Computer Society Press, Mai 2003.

[Tra06] Dietrich Travkin. *Bewertung automatisch erkannter Instanzen von Software-Mustern*. Diplomarbeit, Universität Paderborn, Institut für Informatik, September 2006.

[Wen05] Sven Wenzel. Detection of Incomplete Patterns Using Fujaba Principles. In Holger Giese und Albert Zündorf, Hrsg., *Proc. of the* 3^{rd} *International Fujaba Days 2005, Paderborn, Germany*, Jgg. tr-ri-05-259, Seiten 33–40. Universität Paderborn, September 2005.