Adaptive Erkennung von Entwurfsmängeln in objektorientierter Software

Jochen Kreimer, jotte@uni-paderborn.de
Universität Paderborn, Institut für Informatik
Fürstenallee 11, 33102 Paderborn

1 Einleitung

Qualität von Software. Die Qualität von Software kann je nach Anwendungsgebiet an unterschiedlichen Kriterien gemessen werden. Für große Software-Systeme spielen u. a. Kriterien wie Wartbarkeit, Verständlichkeit und Erweiterbarkeit eine wichtige Rolle [1].

Entwurfsmängel erkennen. Unser Ziel ist es, Fehler im Entwurf von Software-Systemen zu erkennen und somit „schlechte“ — unverständliche, schwer erweiter- und änderbare — Programmstrukturen zu vermeiden.

Diese Fehler nennen wir Entwurfsmängel. Sie stehen eng mit Eigenschaften der Programmstruktur in Beziehung und sind daher unterhalb der System-Architektur anzuordnen. Eine Vielzahl von Entwurfsmängeln finden sich in der Literatur als *„Design Heuristics“* [7], *„Design Characteristics“* [10] oder *„Bad Smells“* [3]. Die Autoren bezeichnen Entwurfsmängel i. d. R. durch sprechende Begriffe und erklären dem Software-Entwickler wie solche Entwurfsmängel erkannt und behoben werden können.

Prominente Entwurfsmängel sind z. B. „Große Klasse“ oder „Lange Methode“.

Da die Suche zeitaufwändig und vom persönlichen Empfinden und Erfahrungsschatz des Suchenden abhängig ist, bietet sich eine Werkzeugunterstützung an. Werkzeuge wie *„CodeCrawler“* [5] und *„Crocodile/CrocoCosmos“* [8] visualisieren quantitative und strukturelle Eigenschaften eines analysierten Programms und bieten dem Suchenden unterschiedliche Sichtweisen zur Analyse großer Programme an.

Unser Ziel ist es jedoch, dem Benutzer die Interpretation und Analyse der Programmeigenschaften abzunehmen und automatisiert Hinweise auf Entwurfsmängel zu liefern. Dabei soll ein flexibler Interpretationsmechanismus eingesetzt werden, der sich den speziellen Anforderungen und Konventionen eines Projektes, einer Entwicklergruppe oder eines Anwendungsgebietes anpasst.

2 Adaptive Erkennung

In diesem Ansatz kombinieren wir Metrik-basierte Ansätze und Maschinelle Lernverfahren zu einem adaptiven Verfahren.

Metriken. Wir folgen dem Ansatz aus [6] und ordnen jedem Entwurfsmangel eine Menge von Programmeigenschaften zu, die wir durch objektorientierte Entwurfsmetriken ausdrücken. Häufig werden Größen-, Komplexitäts-, Kopplungs- oder Kohäsionsmetriken verwendet.

Wir setzen statische Programmanalyse mit klassischen Methoden der Kontroll- und Datenflussanalyse sowie der abstrakten Interpretation ein. Die Analyseergebnisse führen u. a. zu einem Programmabhängigkeitsgraphen der als abstraktes Modell des Analysegegenstandes dient und zur Metrikberechnung genutzt wird.

Maschinelle Lernverfahren. Bei Lernverfahren [11, 4] werden statistische Methoden eingesetzt, um aus einer gegebenen Menge von Beispiel-Instanzen zu „lernen“. Man spricht hier von einer Trainingsphase.

Eine Instanz besteht aus einer Menge von Attributen, denen Werte zugeordnet sind. Ein spezielles Attribut, das Zielattribut, beschreibt das Lernziel.

Das Lernverfahren erstellt aus dem Trainingsinstanzen einen Klassifizierer, z. B. in Form eines Entscheidungsbaumes, der bereits erlernten und zukünftigen Instanzen ein Zielattribut zuordnet.

Adaption. Wir verwenden Lernverfahren, um für jeden Entwurfsmangel einen speziellen Klassifizierer zu erstellen. Die Menge von Metriken, die dem Entwurfsmangel zugeordnet sind, bilden die Instanzen für das Lernverfahren. Als Zielattribut wird die Information verwandt, ob ein Entwurfsmangel vorliegt oder nicht.

So lassen sich zunächst Ergebnisse der Metrikberechnung von Programmobjekten, die bekanntermaßen Entwurfsmängel enthalten, als Trainigsbeispiele verwenden.

Später können ganze Systemteile analysiert und nach Entwurfsmängeln durchsucht werden. Hierzu werden die Ergebnisse der Metrikberechnungen aller enthaltenen Programmobjekte den entsprechenden Klassifizierern übergeben, die dann im Zielattribut einen Entwurfsmangel anzeigen.

3 Werkzeug für Eclipse

Wir haben das prototypische Werkzeug *„IYC — It's Your Code“* als *Plugin* für die freie Entwicklungsumgebung *Eclipse* [2] entwickelt.

Das Werkzeug analysiert Java-Bytecode und erstellt, neben einigen spezialisierten Analysen, einen Programmabhängigkeitsgraphen.

Als maschinelles Lernverfahren setzen wir den „J48“-Klassifizierer aus der *WEKA*-Bibliothek [9] ein.

Der Benutzer kann in Konfigurationsdialogen Entwurfsmängel definieren, indem er vorgegebene Metriken zuordnet. Die Benutzeroberfläche von *Eclipse* erlaubt Programmobjekte in unterschiedlichen Sichten auszuwählen. In Kontextmenüs können Trainings- und Analysevorgänge angestoßen werden.

Gefundene Entwurfsmängel werden dem Benutzer präsentiert. Dieser kann einzelne Vorschläge abweisen oder akzeptieren und damit das Lernverfahren weiter trainieren.

4 Beobachtung und Ausblick

Diese Entwurfsmangelerkennung wird auf drei Ebenen beeinflusst:

1. Die Möglichkeiten einen Entwurfsmangel zu modellieren wird durch die Menge der Analysen bzw. Metriken, die zur Modellierung von Entwurfsmängeln bereitgestellt werden, beeinflusst.

2. Die Modellierung der einzelnen Entwurfsmängel durch Auswahl der zu benutzenden Metriken liegt in der Hand des Benutzers. Hier wird die grobe Charakteristik eines Mangels festgelegt.

3. Der Trainings- und Lernprozess erlaubt die feine Einstellung ohne dass der Benutzer eigene Grenzen für Metriken und deren Verknüpfung spezifizieren müsste.

Die Wirksamkeit des vorgestellten Verfahrens muss durch empirische Untersuchungen gezeigt werden. Hierzu suchen wir noch Projektpartner.

Literatur

[1] F. Abreu and R. Brito. Objectoriented Software Engineering: Measuring and Controlling the Development Process, 1994.

[2] Eclipse.org Consortium, `http://www.eclipse.org`. *Eclipse.org Main Page*, 2003.

[3] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Addison Wesley, 1999.

[4] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Stats. Springer, 2001.

[5] Michele Lanza. *Object-Oriented Reverse Engineering — Coarse-grained, Fine-grained, and Evolutionary Software Visualization*. PhD thesis, University of Berne, May 2003.

[6] Radu Marinescu. *Measurement and Quality in Object-Oriented Design*. PhD thesis, University of Timisoara, 2002.

[7] Arthur J. Riel. *Object-Oriented Design Heuristics*. Addison Wesley, 1996.

[8] Frank Simon, Frank Steinbruckner, and Claus Lewerentz. Metrics Based Refactoring. In *CSMR*, pages 30–38, 2001.

[9] Weka 3 — Data Mining with Open Source Machine Learning Software in Java, `http://www.cs.waikato.ac.nz/~ml/weka/`.

[10] Scott A. Whitmire. *Object Oriented Design Measurement*. John Wiley & Sons, Inc., 1997.

[11] Ian H. Witten and Eibe Frank. *Data Mining*. Morgan Kaufmann, Los Altos, US, 2000.