Quelltextanalyse eines mittelgroßen, neuen Produktivsystems

Daniel Vinke, Meik Teßmer, Thorsten Spitta
Universität Bielefeld
Fakultät für Wirtschaftswissenschaften
post@danielvinke.de, {mtessmer|thspitta@wiwi.uni-bielefeld.de}

Abstract

Es wurde ein System von 200.000 LOC auf OO-Maße hin analysiert und die Abhängigkeiten der Maße statistisch analysiert. Das System ist seit 2001 im Einsatz und wird auch zur Zeit noch (2007) evolutionär weiter entwickelt. Einige Befunde aus der neueren Literatur wurden bestätigt, einige präzisiert. Wartungsintensive Klassen lassen sich zuverlässig erkennen.

1 Einführung

Quelltextanalysen sind nichts Neues (einen guten Überblick gibt Zuse [91]), auch vor der Verbreitung objektorientierter Software. Für objektorientierte Systeme braucht man spezifische Maße, von denen die „CK-Maße“ von Chidamber und Kemerer [CK94] als allgemein akzeptiert gelten [SK03]. Unklar definiert ist das Maß LCOM [SK03, 300]. Es erwies sich bei einer Validation der CK-Maße als einziges als nicht signifikant [BBM96].

An mittleren bis großen Systemen finden sich nur wenige Untersuchungen, noch weniger an produktiven [Vin07, 35]. Neben einer umfangreichen Laboruntersuchung [DKST05] wurde nur eine Analyse der Browserfamilie Mozilla gefunden [GFS05]. Beide Systeme sind in C++ geschrieben.

2 Untersuchungsobjekt

Das von uns analysierte System BIS (*Bielefelder Informations-System*) nennen wir *mittelgroß*: 190.000 LOC, 1430 Java-Klassen, 135 Packages. Von den Anfängen als Stundenplan-Verwaltungssystem für Studierende hat es sich inzwischen zum Raumplanungs-, Veranstaltungsverwaltungs- bis zum Prüfungsabwicklungssystem evolutionär entwickelt. Die Prüfungsabwicklung ist derzeit (Sommer 2007) noch recht rudimentär ausgeprägt. Große Teile des Systems sind als Auskunftsystem im Internet sichtbar: http://eKVV.uni-bielefeld.de/.

Da das System seit Ende 2003 unter Eclipse mit CVS weiterentwickelt und gepflegt wird, sind wir im Projekt SQUARE (*Software Quality and Refactoring Environment*) in der Lage, die Systemzustände seit Anfang 2004 in Form von Zeitreihen festzustellen, um z. B. Aussagen über Art und Verlauf der Evolution während der Entwicklung zu machen. Als erster Schritt war dazu eine Zeitpunkt-bezogene Analyse notwendig, von der wir hier berichten.

3 Untersuchung

Analysiert wurde der Stand des Systems vom Mai 2005. Wir wollten die CK-Maße erheben, ihren Nutzen mittels statistischer Analyse beurteilen und für das analysierte System kalibrieren.

Als Analysewerkzeug wurde *Understand* (for Java) verwendet. Die Alternative *Metrics* (Sourceforge) schied aus, da wir Quell- und keinen Bytecode analysieren wollten. Mit *Understand* hatten wir bereits Erfahrungen durch die Analyse eines deutlich größeren Systems (ca. 1 Mio LOC), das in C++ geschrieben war [Teß04]. Mittels *Understand* wurden alle CK-Maße ermittelt:

- **CBO** Coupling between Objects
- **LCOM** Lack of Cohesion in Methods
- **RFC** Response for a Class
- **DIT** Depth of Inheritance Tree
- **NOC** Number of Children
- **WMC** Weighted Methods per Class
- **NOM** Number of Methods

Es zeigte sich, dass die Daten für RFC nicht verwendbar waren, weil das Maß von *Understand* falsch berechnet wird, und dass LCOM nur bedingt verwendbar ist. Es ist bereits in [CK, 488] unpräzise definiert und wird von *Understand* als Prozentsatz ausgeben. Dies waren jedoch keine Hindernisse für die Analyse, weil wir ohnehin über Regressionsanalysen ermitteln wollten, welche Maße für das Auffinden wartungsintensiver Klassen benötigt werden. Danach ist RFC entbehrlich, weil es stark mit CBO und WMC korreliert ist.

Mit den ermittelten Maßen (ohne RFC, teilweise mit LCOM) wurde eine umfassende statistische Analyse mit dem System ***R*** durchgeführt, und zwar univariat (Häufigkeiten, Verteilungseigenschaften), bivariat (Korrelationsanalyse und Vergleich von jeweils

zwei Maßen mit Toleranzgrenzen) und multivariat (Hauptkomponentenanalysen). Bild 1 zeigt einen Beispielplot von max(VG)[1] gegen WMC. Man sieht z. B. eine Klasse mit fast 200 Methoden links oben, die aber nicht komplex ist. Interessant in diesen Plots sind immer die Einzelpunkte links oben, vor allem aber die rechts oben. Dies sind die „Ausreißer", die als wartungsintensiv interpretiert werden. Wir zählen in Bild 1 davon acht.

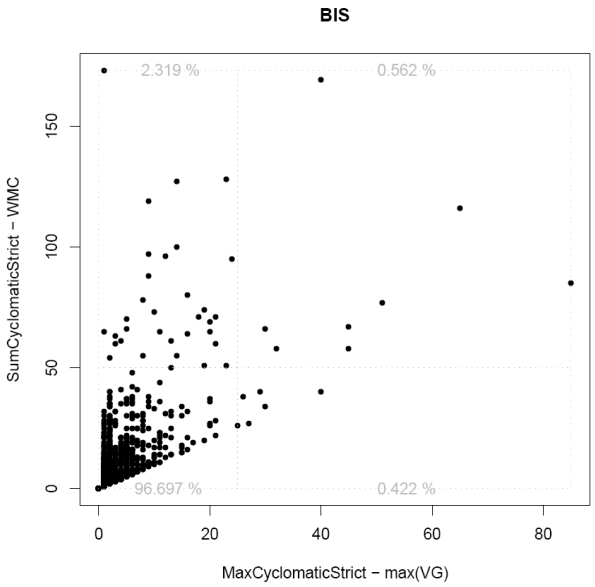

Bild 1: max(VG) korreliert mit WMC

Außerdem wurde Eclipse (mit 3.5 Mio LOC) analysiert, um die eigenen Kalibrierungen beurteilen zu können. Es enthält 12.000 Klassen, davon einige mit mehr als 10.000 LOC. Wir haben eine Klasse mit max(VG) > 1000 und drei mit VG > 500 gefunden. Strukturell sieht BIS in der Korrelationsanalyse zwischen WMC und CBO nicht schlechter aus als Eclipse.

4 Ergebnisse

Folgende Ergebnisse wurden ermittelt bzw. Angaben aus der Literatur bestätigt:

- LOC korreliert stark mit Komplexität
- Man findet wartungsintensive Klassen recht sicher mit mehreren Maßen. Diese sind miteinander korreliert.
- Diese Maße sind: LOC, max(VG), CBO und WMC (impliziert VG)
- Das Interaktionsmaß DIT*CBO bringt wichtige Zusatzinformationen
- Von den **OO-Maßen** braucht man **nur wenige**; es sind: WMC und DIT*CBO
- LCOM: Nach der Interpretation durch *Understand* haben recht viele Klassen einen **Kohäsionsmangel** von über 50%; oft **nahe 90%**.
- Im konkreten Fall des Systems BIS sind mindestens 70 (von rund 1400) Klassen wartungsintensiv. **5% halten wir für viel**.

5 Ausblick

Es besteht dringender Forschungsbedarf zur Kalibrierung der Maße, denn unsere Vorschläge beziehen sich nur auf einen Fall. Klar ist, dass in jedem Einzelfall kalibriert werden *muss*. Es sollte aber Toleranzgrenzen nach einer Klassifikation geben (etwa große / kleine oder einfache / komplexe Systeme).

Die erwähnten Längsschnittuntersuchungen sind in Arbeit.

Literatur

[BBM96] Basili, V. R.; Briand, L. C.; Melo, W. L.: A Validation of Object-Oriented Design Metrics as Quality Indicators. *IEEE Trans. Software Eng.* 22 (10): 751-761, 1996.

[CK94] Chidamber, S. R.; Kemerer, C. F.: A Metrics Suite for Object Oriented Design. *IEEE Trans. Software Eng.* 20 (6): 476–493, 1994.

[DKST05] Darcy, D. P.; Kemerer, C. F.; Slaughter, S. A.; Tomayko, J. E.: The Structural Complexity of Software: An Experimental Test. *IEEE Trans. Software Eng.* 31 (11): 982–995, 2005.

[GFS05] Gyimóthy, T.; Ferenc, R.; Siket, I.: Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. *IEEE Trans. Software Eng.* 31 (10): 897–910, 2005.

[Teß04] Teßmer, M.: Architekturermittlung aus Quellcode – Eine Fallstudie. Diplomarbeit, Universität Bielefeld, Technische Fak., Jan. 2004.

[Vin07] Vinke, D.: Quellanalyse eines mittelgroßen Produktivsystems – Eine Fallstudie zur Qualitätssicherung. Diplomarbeit, Universität Bielefeld, Fak.Wirtschaftswissenschaften, März 2007.

[Zus91] Zuse, H.: *Software Complexity*. Berlin et al., de Gruyter, 1991.

1 Die Komplexität nach McCabe. Die durchschnittliche Komplexität avg(VG) ist wenig aussagefähig.