Nachdokumentation von Geschäftsregeln aus Quelltext

Rainer Schmidberger
Abteilung Software-Engineering, Institut für Softwaretechnologie
Universität Stuttgart
www.iste.uni-stuttgart.de/se

Kurzfassung
Viele Softwaresysteme haben ein Problem bei der Qualität der Dokumentation der funktionalen Anforderungen. Hiervon besonders betroffen sind Systeme, die sich bereits mehrere Jahre in der Wartung befinden. Vor allem eine Restrukturierung oder Neuentwicklung des Systems wird so erheblich erschwert oder gar unmöglich gemacht.
Neben den an der Entwicklung des Systems beteiligten Personen dient in der Regel der Quelltext als verbleibende Informationsquelle, wenn es um die Wiederherstellung einer Dokumentation einzelner funktionaler Anforderungen, wie z. B. einer Geschäftsregel, geht.
In diesem Artikel wird ein Verfahren vorgestellt, das systematisch den Programmcode nach Indikatoren für Geschäftsregeln durchsucht und eine methodische Hilfestellung bei der Nachdokumentation leistet.

1 Einführung

Für die wirtschaftliche Wartung oder Neuentwicklung eines Softwaresystems ist eine ausführliche Dokumentation der funktionalen Anforderungen ein großer Vorteil oder sogar eine Voraussetzung. Viele Projekte haben in diesem Punkt ein Defizit.
Wenn es in der Praxis um die Wiederherstellung der Dokumentation einzelner funktionaler Anforderungen – im folgenden als Merkmale bezeichnet – geht, dient üblicherweise der Quelltext als Informationsquelle. Die heute bekannten Verfahren zur Merkmallokalisierung [1], [2], also dem Auffinden des für ein bestimmtes Merkmal spezifischen Programmcodes, setzen aber voraus, dass man die Existenz eines Merkmals grundsätzlich kennt und nur die genaue Ausgestaltung rekonstruieren will. Programmteile, die spezifisch für ein bestimmtes Merkmal sind, werden aufgezeigt und können gezielt durch den Wartungsprogrammierer genauer analysiert werden. Der Nutzen dieser Verfahren ist, dass nicht der gesamte Programmcode, sondern nur ein kleiner Teil des Programmcodes auf das Merkmal hin untersucht werden muss.
Für die geplante Neuentwicklung eines Systems sind die Verfahren der Merkmallokalisierung aber nur bedingt geeignet, da in der Regel nicht alle Merkmale, die in der Software implementiert sind, überhaupt noch bekannt sind.

2 Definitionen

Geschäftsregeln gehören zu den funktionalen Anforderungen und spielen bei Informationssystemen in der Regel eine große Rolle. Selfridge et al. definieren in [3] Geschäftsregel wie folgt:

> *A requirement on the condition or manipulation of data expressed in terms of the business enterprise or application domain.*

Sneed definiert in [4] Geschäftsregeln so:

> *Business rules are a set of conditional operations attached to a given data*

In beiden Definitionen spielt die Bedingung, unter der die Regel angewendet wird, eine wichtige Rolle. In dieser Arbeit wird die folgende Definition verwendet: Geschäftsregeln sind funktionale Anforderungen, die in der Sprache der Anwendungsdomäne formuliert und nach dem Muster „Wenn x, dann y“ aufgebaut sind.
Die These dieser Arbeit ist, dass umgekehrt Verzweigungen mit ihren Bedingungen, also das „Wenn x“, gute Indikatoren für Geschäftsregeln im Programmcode darstellen.
Im Prozessmodell des in diesem Artikel beschriebenen Verfahrens spielt der Fachexperte die dominierende Rolle. Er muss über zwei entscheidende Fähigkeiten verfügen: Erstens muss er in einer Bottom-up-Betrachtung Quelltextzeilen einem fachlichen Kontext zuordnen können, zweitens muss er in der Lage sein, in einer Top-down-Betrachtung Prozesse der Anwendungsdomäne auf Quelltextabschnitte zu übertragen. Es ist auch denkbar, diese Rolle mit zwei Personen zu besetzen, die gemeinsam arbeiten: einem Spezialisten der Anwendungsdomäne und einem Entwickler.

3 Ablauf

Mit dem Verfahren soll festgestellt werden, ob für eine Verzweigung des Programmcodes fachliche Relevanz vorliegt oder nicht. Es werden aber a priori nicht alle Verzweigungen des Programmcodes betrachtet, sondern nur die, die bei der Programmausführung mit den ausgewählten Testdaten auch berührt werden.
Ausgangspunkt, wie in Abbildung 1 dargestellt, ist eine größere Menge an praxisüblichen Testdaten (z. B. aktuelle Produktionsdaten) sowie ein instrumentiertes, kompiliertes und betriebsbereites Programm. Das Programm wird nun mit den ausgewählten Testdaten ausgeführt, und die abgearbeiteten Verzweigungen werden in der Programmpfad-Datenbank protokolliert. Nach der Programmausführung erfolgt eine Auswertung der protokollierten Programmpfade.
Die abgearbeiteten Programmverzweigungen werden in zwei Gruppen unterteilt:

1. Verzweigungen vom Typ A, bei denen die Testfälle nicht alle enthaltenen Zweige abarbeiten. Beispielsweise wird bei einer IF-Verzweigung nur der IF- und nicht der ELSE-Zweig bearbeitet. In diesem Fall verbleibt ein nicht bearbeiteter Zweig, der im folgenden „dunkler Zweig“ genannt wird.
2. Verzweigungen vom Typ B, bei denen die Testfälle alle enthaltenen Zweige abarbeiten. Beispielsweise wird bei einer IF-Verzweigung sowohl der IF- als auch der ELSE-Zweig bearbeitet.

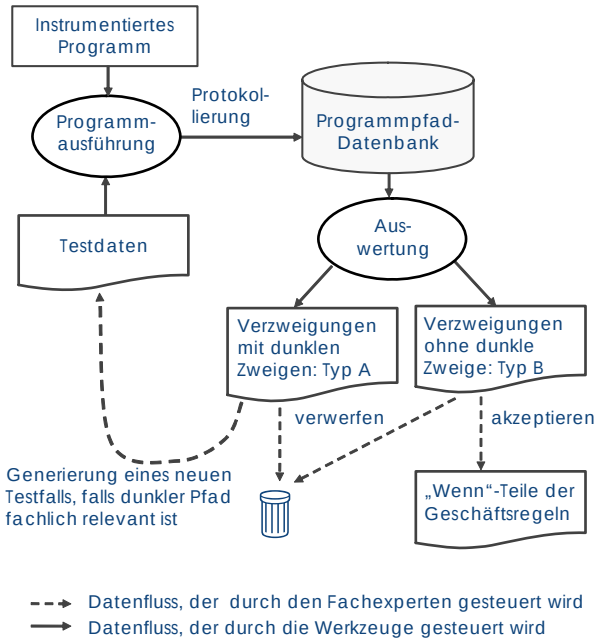

Abbildung 1: Prozessmodell

Verzweigungen vom Typ A, bei denen also ein dunkler Zweig verbleibt, prüft der Fachexperte, ob sinnvolle Eingabedaten für einen Testfall existieren, der zur Abarbeitung des dunklen Pfads führen würde. Anhaltspunkte, wie dieser Testfall beschaffen sein könnte, kann das Prädikat der Bedingung liefern. Die Testdaten werden dann um einen solchen Testfall erweitert. Gibt es keine solchen Testdaten, liegt ein sicheres Indiz dafür vor, dass die Verzweigung nicht Teil einer Geschäftsregel ist. Die Verzweigung wird dann verworfen, d. h. nicht weiter betrachtet.
Verzweigungen vom Typ B, bei denen also alle Zweige abgearbeitet werden, werden vom Fachexperten ebenfalls einzeln betrachtet. Der Fachexperte bewertet, ob die Verzweigung fachlich relevant ist oder nicht. Falls die Verzweigung fachlich relevant ist, wird sie akzeptiert, und eine fachliche Beschreibung in der Form „Wenn ... dann ...“ als Kommentar wird festgehalten. Fachlich nicht relevante Verzweigungen werden verworfen und nicht weiter betrachtet.
Der beschriebene Ablauf wird nun so lange wiederholt, wie aus den Verzweigungen mit dunklen Zweigen neue Testfälle entstehen. Das Resultat des Ablaufs sind die kommentierten, fachlich relevanten Verzweigungen.
Im praktischen Einsatz des Verfahrens hat sich gezeigt, dass es günstig ist, den Programmpfad durch vordefinierte Eintritts- und Austrittsstellen zu begrenzen. Es können so auch einzelne Module betracht werden. Ferner hat sich gezeigt, dass der Anteil der fachlich relevanten Verzweigungen unter den Typ-B-Verzweigungen sehr viel höher ist als unter den Typ-A-Verzweigungen. Ausnützen lässt sich diese Erkenntnis allerdings nicht, da trotzdem eine Durchsicht der Typ-A-Verzweigungen erfolgen muss.

4 Technische Umsetzung

Die technische Umsetzung der erforderlichen Programmcode-Instrumentierung wurde für IBM-COBOL und Java am Institut für Softwaretechnik (ISTE) der Universität Stuttgart implementiert und bei einem COBOL-System mit ca. 40 kLOC sowie einer Java-Anwendung mit 30 kLOC erprobt. Das System zur Auswertung der Programmpfade wurde ebensfalls am ISTE implementiert und befindet sich derzeit in der Erprobung.

5 Bewertung

Der Vorteil des Verfahrens liegt zum einen darin, dass der Fachexperte angeleitet wird systematisch fachlich relevante Verzweigungen des Programmcodes durch das Akzeptieren von nicht fachlich relevanten zu trennen. Die Generierung neuer Testdaten aus den Verzweigungen mit dunklen Zweigen bildet für den Fachexperten zudem eine methodische Hilfestellung: Sie erleichtert es, die fachliche Bedeutung eines Programmzweigs zu bewerten. Wenn sich keine Testdaten finden lassen, die zur Abarbeitung eines dunklen Zweiges führen, ist die fachliche Bedeutung der Verzweigung vermutlich vom Fachexperten noch nicht voll erfasst.
Der Nachteil des Verfahrens besteht darin, dass die Funktion des „dann“-Teils der Geschäftsregel nicht betrachtet wird. Auch hat das Verfahren keinen Vorteil gegenüber der vollständigen Betrachtung eines Programmcodes, wenn keine dunklen Zweige als fachlich unbedeutend festgestellt werden.

Literatur

[1] N. Wilde and M.C. Scully, “Software Reconnaissance: Mapping Program Features to Code,” J. Software Maintenance: Research and Practice. vol. 7, pp. 49-62, Jan. 1995.

[2] T. Eisenbarth, R. Koschke and D. Simon, “Locating features in source code” Software Engineering, IEEE Transactions on, Volume 29, Issue 3, March 2003 Page(s):210 - 224.

[3] Selfridge, P.G.; Waters, R.C.; Chikofsky, E.J.; Reverse Engineering, 1993., Proceedings of Working Conference on, 21-23 May 1993 Page(s):144 - 150

[4] Sneed, H.M.; Erdos, K.; Program Comprehension, 1996, Proceedings., Fourth Workshop on 29-31 March 1996 Page(s):240 - 247