Methodische Erkennung von Softwarearchitekturen unter Nutzung von Featuremodellen

Matthias Riebisch, Ilian Pashov
TU Ilmenau, FG Softwaresysteme / Prozessinformatik
Matthias.Riebisch@tu-ilmenau.de

Einleitung

Im Laufe von Veränderungen von Softwaresystemen im Rahmen von Software-Wartung kommt es zu Verlusten an Qualität der Softwarearchitektur. Diese Verluste werden häufig als Architektur-Drift oder als Architectural Decay bezeichnet. Diese Qualitätsverluste wirken sich vorrangig auf nichtfunktionale Qualitätsmerkmale wie Wartbarkeit, Zuverlässigkeit und Portabilität aus, sie werden deshalb vom Auftraggeber von Wartungsmaßnahmen häufig zu spät wahrgenommen. Der Verlust an Wartbarkeit führt relativ schnell dazu, dass das betroffene Softwaresystem nicht mehr weiterentwickelt werden kann. Bezogen auf das Phasenmodell Staged Model [RB00] hat dieses Softwaresystem den Zustand Evolution verlassen. Erfordert eine weitere Nutzung größere Veränderungen, muss durch Reengineering der Zustand Evolution wieder herbeigeführt werden. Hierzu ist es vor allem erforderlich, Strukturen und Zusammenhänge des betroffenen Softwaresystems zu verstehen. Dem Erkennen der Softwarearchitektur kommt besondere Bedeutung zu, weil erst daraufhin architekturelle Veränderungen möglich sind.

In den meisten Fällen stehen dabei die Entwickler vor dem Problem, dass die Softwaredokumentation für das Verständnis der Legacy-Architekturen unzureichend ist und das keine Kommunikation mit den ehemaligen Entwicklern mehr möglich ist. Existierende Ansätze für die Erkennung gehen davon aus, die benötigten Informationen aus dem Quellcode zu extrahieren. Für die Erkennung einer Softwarearchitektur sind die beiden Aspekte

- der Architektur zugrunde liegende funktionale Entwurfsziele,
- daraufhin getroffene Entwurfsentscheidungen

von besonderer Bedeutung, die aus dem Quellcode üblicherweise nicht ermittelt werden können.

In dem hier vorgestellten Ansatz wird vorgeschlagen, darüber hinaus das Wissen über die Problemdomäne zum Programmverständnis zu nutzen. Das ursprüngliche Domänenwissen wird dazu in Form eines Featuremodells beschrieben, welches mit Hilfe von Domänenexperten zu aktualisieren ist. Dabei werden außerdem Ergebnisse der Quellcodeanalyse einbezogen. Features werden dann als Mittel der Aufstellung und Verifikation von Hypothesen genutzt, um schrittweise Architekturbeschreibungen zu erstellen.

Nutzung von Featuremodellen

Featuremodelle stellen die Beziehungen zwischen Features dar. Bei Features handelt es sich um für den Kunden sichtbare Merkmale, die wichtige Abstraktionen von Anforderungen darstellen. Ein Feature ist dabei definiert als "a logical unit of behaviour that is specified by a set of functional and quality requirements representing an aspect valuable to the customer and system architect" entsprechend der Definitionen in [Bo00] und [Ri03]. Features unterstützen die Kapselung von Anforderungen, widerspiegeln die Domäne und ihre Variabilität und erlauben eine Steuerung der Weiterentwicklung. Darüber hinaus stellen sie ein wesentliches Kommunikationsmittel zwischen den an der Entwicklung und Nutzung Beteiligten dar [RSP03].

Featurebasierte Architekturerkennung

Die Features werden in der hier vorgestellten Methode als Mittler zwischen Anforderungs- und Architekturebene eingesetzt, um durch Überbrücken der unterschiedlichen Abstraktions- und Detaillierungsebenen beider Ebenen das Verstehen der oben genannten Aspekte und das Erkennen der Architektur zu unterstützen.

Dazu werden zwei unterschiedliche Typen von Features eingesetzt, die Entwurfsziele und Entwurfselemente beschreiben. Das Featuremodell der Entwurfsziele beschreibt dann den Problemraum und widerspiegelt Anforderungen. Das zweite Featuremodell der Entwurfselemente stellt den entwurfsorientierten Lösungsraum dar, es beschreibt Elemente der Architektur wie Strukturen, Komponenten und Pattern. Verknüpfungen beider korrespondierender Featuremodelle erlauben eine Kombination von Bottom-Up- und Top-Down-Vorgehen. Die Erstellung solcher Verknüpfungen in einem inkrementellen Vorgehen von Hypothese und Verifikation stellt den Kern der Methode dar (siehe Abb. 1).

Die Ausgangspunkte für das Bottom-Up-Vorgehen ist die Analyse des existierenden Quellcodes und weiterer Entwurfsdokumente, soweit vorhanden. Dabei gewonnene Erkenntnisse werden im Entwurfselemente-Featuremodell strukturiert. Dabei können bereits Vergleiche zum Entwurfsziele-Featuremodell hergestellt werden.

Das Top-Down-Vorgehen hat seinen Ausgangspunkt in der Analyse der noch vorhandenen Anforderungsdokumente wie Pflichtenheft, Szenario-Beschreibungen, Liste der Änderungswünsche und Handbücher. Daraus werden Features der Entwurfsziele abgeleitet und Use-Case-Beschreibungen entwickelt, die eine genauere Darstellung von Szenarien enthalten. Zusätzlich wird eine Domänenanalyse durchgeführt, um weitere Features sowie zusätzliche, in den Requirements-Dokumenten nicht enthaltene Szenarien zu ermitteln.

Architekturerkennung durch Hypothesenaufstellung und -verifikation

Die beiden Featuremodelle und die Use-Case-Beschreibungen dienen als Ausgangsbasis für die Aufstellung von Hypothesen zur vorliegenden Architektur. Eine Hypothese wird durch eine Referenz zwischen einem Entwurfsziel-Feature und einem Entwurfselement-Feature ausgedrückt, die in einer Cross-Referenz-Tabelle vermerkt wird. Andere Hypothesen, die sich auf Code-Fragmente beziehen, werden durch eine Referenz zwischen einem Entwurfsziel-Feature und einem Quellcodeelement abgebildet und in einer weiteren Cross-Referenz-Tabelle vermerkt. Dabei wird der Quellcode mit den üblichen Analysemethoden untersucht.

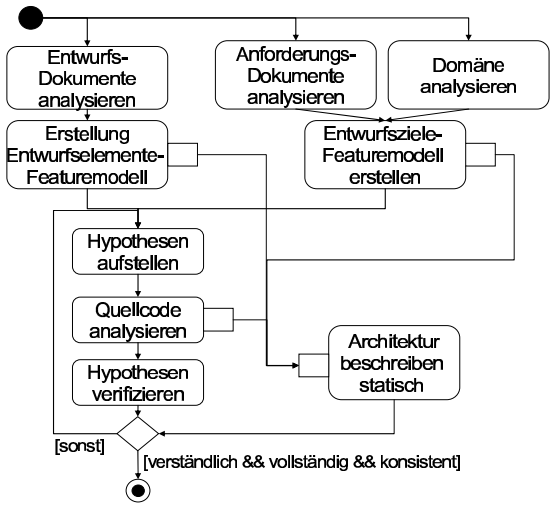

Abb. 1: Vorgehen bei der Erkennung von Software-Architekturen

Im Anschluss an die Aufstellung von Hypothesen werden diese verifiziert. Dazu werden Architekturdiagramme erstellt, die das vermutete Umfeld eines Architekturelements beschreiben. Erweist sich eine zunächst vermutete Beziehung zwischen Features, Architektur- und Quellcodeelementen als falsch, wird versucht, die betreffende Hypothese durch eine andere zu ersetzen. Häufig ergeben sich während der Untersuchungen neue Hypothesen für weitere Architekturelemente. Alle positiv verifizierten Hypothesen verbleiben in den Cross-Referenz-Tabellen. Aus den mit den Hypothesen erstellten Fragmenten entsteht Mosaik-artig eine Architekturbeschreibung, die zunächst die statischen Aspekte enthält. Die Featuremodelle und die Cross-Referenz-Tabellen helfen dem Bearbeiter, die häufig sehr schlecht verständlichen Legacy-Strukturen schrittweise zu erfassen und die Komplexität zu beherrschen.

Dynamische Architekturbeschreibung unter Verwendung der Use-Case-Szenarien

In der nächsten Phase werden die Use-Case-Beschreibungen als Ausgangspunkt für die Analyse des Verhaltens eingesetzt. Dabei werden ebenfalls die bekannten Verfahren der Code-Analyse eingesetzt. Aus den dabei gewonnenen Verhaltensinformationen werden neben den Architekturbeschreibungen auch Testfälle abgeleitet, die beim meist anschließend durchgeführten Refactoring verwendet werden können.

Erfahrungen und Werkzeugeinsatz

Der hier vorgestellte Vorgehensansatz wurde bisher bis zur Modellierung der statischen Architektur an einem größeren Projekt im industriellen Maßstab eingesetzt. Dabei konnte eine gute Akzeptanz der Methode erreicht werden, die sich unter anderem an der Aufnahme des Featuremodells in das unternehmensweite Vorgehensmodell des betreffenden Unternehmens zeigt. Eine detaillierte Beschreibung des Vorgehens nach der hier vorgestellten Methode ist in [Pashov 2003] zu finden. Die dynamische Architekturanalyse ist Gegenstand weiterer Untersuchungen und industrieller Projekte.

Für die Umsetzung der hier vorgestellten Methode wurde eine Kombination aus verschiedenen Werkzeugen verwendet. Für die Implementierung von Featureverwaltung und Crossreferenz-Tabellen wurde eine relationale Datenbasis verwendet. Die Visualisierung und Bearbeitung erfolgte mit XML-Editoren. Für die graphische Anzeige und Bearbeitung von Featuremodellen wurden keine Werkzeuge eingesetzt. Mittlerweile gibt es dafür jedoch erste Werkzeuge, sowohl Public Domain wie Captain Feature [CF05] als auch professionell vertriebene wie pure::variants [PV05].

Die vorgestellte Methode ermöglicht durch Werkzeugunterstützung eine effektive Analyse auch großer Legacy-Systeme. Sie ersetzt nicht den Sachverstand der Bearbeiter der Reverse-Engineering-Aufgaben, sondern unterstützt deren Beherrschung der Komplexität und ihr zielgerichtetes inkrementelles Vorgehen. Für die Akzeptanz in der Praxis hat sich als besonders förderlich gezeigt, dass die Nutzung von Crossreferenz-Tabellen neben der Verwaltung von Wahrscheinlichkeiten der Hypothesen auch eine Fortschrittsanalyse ermöglicht.

Referenzen

[Bo00] Bosch, J.: Design & Use of Software Architectures - Adopting and Evolving a Product Line Approach. Addison-Wesley, 2000.

[CF05] Captain Feature - Feature Modelling Tool, V 1.0. Online verfügbar unter https://sourceforge.net/projects/captainfeature/

[PV05] pure::variants. Feature Modelling Tool, V 1.2. Online verfügbar unter http://www.pure-systems.com/

[RB00] Rajlic, Bennett: A Staged Model for the Software Life Cycle. Computer, July 2000.

[Ri03] Riebisch, M.: Towards a More Precise Definition of Feature Models. In: M. Riebisch, J. O. Coplien, D, Streitferdt (Eds.): Modelling Variability for Object-Oriented Product Lines. BookOnDemand Publ. Co., Norderstedt, 2003. S. 64-76.

[RSP03] Riebisch, M.; Streitferdt, D.; Pashov, I.: Modeling Variability for Object-Oriented Product Lines. In: Buschmann, F.; Buchmann, A. P.; Cilia, M. (Eds.): Object-Oriented Technology. ECOOP 2003 Workshop Reader. Springer, Lecture Notes in Computer Science , Vol. 3013, 2004, pp. 165 - 178.