Architekturgetriebenes Pairwise-Testing für Software Produktlinien

Sebastian Oster, Andy Schürr

{oster, schuerr}@es.tu-darmstadt.de

Abstract: Software-Produktlinien-Entwicklung ermöglicht eine systematische Wiederverwendung von Software. Aufgrund der Variabilität innerhalb von Software-Produktlinien (SPL) kann eine sehr hohe Anzahl von verschiedenen Produkten erzeugt werden. Daher ist es unerlässlich Testverfahren zu entwickeln, die zum einen eine möglichst vollständige Abdeckung von allen zu generierenden Produkten sicherstellen und zum anderen weniger aufwendig sind als alle Produkte individuell zu testen. Dieser Beitrag beschreibt ein Konzept, mit dem Produktlinien-Variabilität in Bezug auf Systemtests repräsentiert, gehandhabt und vereinfacht werden kann.

1 Einleitung

SPL-Entwicklung ist eine Methode zur Umsetzung systematischer Wiederverwendung von Software. Dabei entsteht eine Gruppe von Software-Produkten, die sich gemeinsame Eigenschaften teilen und damit spezielle Anforderungen erfüllen [PBvdL05]. In der Praxis werden SPLs häufig dadurch getestet, dass jedes generierte Produkt individuell überprüft wird [TTK04]. Dies ist die sicherste und gründlichste Testmethode, jedoch ist der Aufwand für ein solches Testverfahren enorm und in der Regel nicht umsetzbar. Im Automotive-Bereich kann das Motorsteuergerät als eins von bis zu 70 Steuergeräten in einem PKW allein in mehr als 30.000 Varianten konfiguriert werden. Zum Zeitpunkt der Produktinstanziierung am Bandende ist es zeitlich nicht möglich alle Produkte individuell zu testen. Ebenso kann nur ein Bruchteil der möglichen Kombinationen während der Entwicklung überprüft werden. Erstrebenswert ist deshalb die Bestimmung einer repräsentativen Menge von Produkten der SPL. Das erfolgreiche Testen dieser Menge soll mit einer hohen Wahrscheinlichkeit garantieren, dass der Test aller weiteren Produkte der SPL ebenfalls erfolgreich ist. Diese repräsentative Menge soll während der Entwicklungszeit getestet werden. Der hier vorgestellte Ansatz wird im Rahmen des BMBF-Projektes feasiPLe entwickelt [fC06].

2 Verwandte Arbeiten

Kombinatorisches Testen: Kombinatorisches Testen wird bislang meist dazu verwendet, beim Testen EINER Produktinstanz die Anzahl der zu testenden Wertekombinationen von

Testparametern zu reduzieren [CDKP94]. Die Anzahl der entstehenden Testfälle wächst nur so schnell, wie der 2er Logarithmus der Anzahl der Eingabewerte [SM98]. Pairwise-Testing ist eine der bekanntesten Methoden und realisiert laut [CDKP94] eine Block-Coverage von 75% bei einem Aufwand von $O(bk)$, wobei b der Anzahl der Parameter und k der Anzahl der Testfälle entspricht [SM98]. Durch AETG, eine kommerzielle Realisierung des Pairwise-Testings, konnte in Fallstudien sogar eine Block-Coverage von bis zu 92% erreicht werden [CDKP94]. Die Generierung eines Produktes einer SPL ähnelt in vielen Fällen der Bildung einer Software, die alle Features enthält und zur Laufzeit durch Auswahl geeigneter Parameterwerte konfiguriert wird. Dem zufolge können die Ergebnisse des kombinatorischen Testens auf die Auswahl von Produkten einer SPL übertragen werden [McG01].

Architekturgetriebenes Testen: Durch Analyse der Architektur und der Implementierung innerhalb einer SPL werden Features identifiziert, deren Implementierungen nicht miteinander interagieren. Diese können dann unabhängig voneinander getestet werden [Sch07]. Es zeigt sich trotzdem deutlich, dass die Menge der ausgewählten Produkte größer ist als die beim kombinatorischen Testen generierte. Der vorgestellte Ansatz skaliert deshalb ohne weitere Einschränkungen nicht für praxisrelevante Szenarien. Jedoch werden Informationen aus Code und Architektur extrahiert und in den Testprozess eingebunden, was in einer höheren Testabdeckung als beim kombinatorischen Testen resultiert.

3 Architekturgetriebenes Pairwise-Testing

Die Grundidee ist die Kombination des kombinatorischen und des architekturgetriebenen Testens. Nachteil des kombinatorischen Ansatzes ist, dass nicht die minimale Menge von Produkten bestimmt wird, sondern es werden zu viele, sich zudem oft auch äquivalent verhaltende Produkte erzeugt [Sch07]. An dieser Stelle soll der architekturgetriebene Ansatz ins Spiel kommen. Durch Ausnutzung von Informationen aus Architektur und Code sollen möglichst nur sich unterschiedlich verhaltende Produkte gewählt werden. Dabei ist die Definition „sich unterschiedlich verhaltender“ Produkte ein offenes Forschungsthema. Basis des kombinierten Ansatzes bildet das Featuremodell (FM) der Produktlinie. Ein FM wurden erstmals in [KCH+90] vorgestellt und beschreibt die Features einer SPL hierarchisch und präsentiert die Gemeinsamkeiten und Variabilität. Das FM nach [KCH+90] durchlief diverse Anpassungen und Verbesserungen. Ausgangspunkt dieser Arbeit ist ein FM mit „optionalen“, „und“, „xor“ und „oder“-Notationen und require- und exclude-Beziehungen zwischen den Features, wie z.B. in [GFM98] beschrieben. Es ist möglich, das fol-

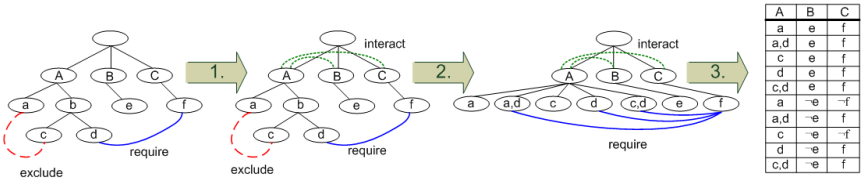

Abbildung 1: Architekturgetriebenes Pairwise-Testing

gende Konzept auf das Orthogonale Variabilitätsmodell zu übertragen, das, je nach Entwicklungsstufe, Anforderungen, Design oder Implementierung repräsentiert [PBvdL05]. Ebenso kann es auf „Feature-Based Model Templates“ von [CAK+05] angewendet werden. Die zusätzlichen Informationen, neben dem FM, unterstützen den Testprozess.

4 Vorgehensweise des kombinierten Ansatzes

Die Vorgehensweise wird durch Abbildung 1 skizziert. Ausgangspunkt ist das FM der entsprechenden SPL. Die Notation der einzelnen Features wird vorerst ignoriert.
1. Erweiterung des Featuremodells: Die bestehenden Abhängigkeiten zwischen den einzelnen Features basieren in der Regel auf den Wünschen des Kunden, auf dem Systemwissen und Managemententscheidungen. Sie schränken die Kombinationsmöglichkeiten der Features bzw. die Menge der zulässigen Produkte ein. Es wird eine weitere Art der Abhängigkeit ergänzt, die aus (Datenfluss-) Analysen der Architektur und des Codes der betrachteten SPL ermittelt werden [Sta00]. Sie bringen zum Ausdruck, dass die Implementierung einer Gruppe von Features Auswirkungen auf das Verhalten der Implementierung einer anderen Gruppe unabhängig davon auswählbarer Features hat. Nur in diesem Fall folgt, dass alle möglichen Kombinationen dieser beiden Featuregruppen getestet werden müssen. Daran anschließend wird das FM für das kombinatorische Testen vorbereitet.
2. Vereinfachung des Featuremodells: Pairwise-Algorithmen benötigen für die Ausführung Testparameter mit Wertebereichen, die aus einer unstrukturierten Aufzählung von Elementen bestehen, die dann kombiniert werden. Standardmäßig sind FMs jedoch zu komplex strukturiert, um entsprechende Algorithmen darauf anzuwenden. Entweder muss ein Pairwise-Algorithmus an die Komplexität des FM angepasst werden, oder das FM wird so vereinfacht, dass es die Voraussetzungen des Pairwise-Testings erfüllt. Zur Zeit werden beide Alternativen verfolgt. Es wird **(1.)** ein neuer rekursiver Pairwise-Testing-Ansatz entwickelt, der hierarchisch aufgebaute FMs mit zusätzlichen Abhängigkeiten als Eingabe akzeptiert. Des Weiteren wird aber auch die Alternative verfolgt, **(2.)** komplexe FMs mit Modelltransformationen in äquivalente nichthierarchische so zu übersetzen, dass man ein Produkt allein durch die Auswahl möglicher Alternativen (Features) aus n Wertebereichen (Variationspunkten) festlegt.
3. Pairwise-Testing: Das vereinfachte FM dient als Basis für die Ausführung des Pairwise-Algorithmus. Zur Realisierung des Pairwise-Testing wird der In-Parametric-Order (IPO)-Algorithmus verwendet [LT98]. Der IPO-Algorithmus wird erweitert, so dass er nur Kombinationen von Features bildet, die auf Grund von require- und exclude-Abhängigkeiten zulässig sind. Ebenso wird nur zwischen den Variationspunkten, die voneinander abhängige Feature-Gruppen enthalten, eine vollständige paarweise Abdeckung realisiert. Diese werden in Abbildung 1 durch interact-Kanten angedeutet. Zwischen Variationspunkten, die miteinander interagieren, soll jede paarweise Kombinationsmöglichkeit ausgeführt werden, da die enthaltenen Features auch in späteren Produkten in Kombination auftreten. Variationspunkte, die nicht miteinander über eine interact-Kante verbunden sind, werden unabhängig voneinander kombiniert. Das Resultat ist eine Menge von Produkten, welche repräsentativ für die SPL ist.

5 Zusammenfassung und Ausblick

Das vorgestellte Konzept kombiniert die Vorteile des kombinatorischen und des architekturgetriebenen Testens. Erwartet wird eine kleinere Produktmenge mit einer höheren Testabdeckung, als beide Ansätze einzeln für sich erzielen. Das FM wurde für die Verwendung des kombinatorischen Testens angepasst und der IPO-Algorithmus erweitert, so dass er Abhängigkeiten beachtet. In zukünftigen Veröffentlichungen wird in Fallstudien untersucht, ob der hier skizzierte Ansatz tatsächlich auch in der Praxis die erhofften Vorzüge besitzt. Evaluiert wird aktuell innerhalb des feasiPLe Projekts [fC06] und mit einer Lego-Auto-SPL in der Lehre. Darüber hinaus ist die Evaluierung an einer realen Automotive-SW-SPL mit einem Industrie-Partner aus dem Rhein-Main-Gebiet geplant.

Literatur

[CAK+05] Krzysztof Czarnecki, Michal Antkiewicz, Chang Hwan Peter Kim, Sean Lau und Krzysztof Pietroszek. Model-driven software product lines. In *OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, Seiten 126–127, New York USA, 2005. ACM.

[CDKP94] D. M. Cohen, S. R. Dalal, A. Kajla und G.C. Patton. The Automatic Efficient Tests Generator. *Fifth Int'l Symposium on Software Reliability Engineering*, IEEE:303–309, 1994.

[fC06] feasiPLe Consortium. www.feasiple.de. 2006.

[GFM98] Martin L. Griss, John Favaro und Case Methodologist. Integrating feature modeling with the RSEB. In *In Proceedings of the Fifth Int'l Conference on Software Reuse*, Seiten 76–85, 1998.

[KCH+90] K. Kang, S. Cohen, J. Hess, W. Nowak und S. Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. 1990.

[LT98] Yu Lei und K. C. Tai. In-Parameter-Order: A Test Generation Strategy for Pairwise Testing. *High-Assurance Systems Engineering, IEEE Int'l Symposium*, 1998.

[McG01] J. D. McGregor. Testing a Software Product Line. *Technical Report CMU/SEI-2001-TR-022*, Software Engineering Institute Carnegie Mellon University, 2001.

[PBvdL05] Klaus Pohl, Günter Böckle und Frank J. van der Linden. *Software Product Line Engineering : Foundations, Principles and Techniques*. Springer, September 2005.

[Sch07] Kathrin Scheidemann. Verifying Families of System Configurations. *Doctoral Thesis*, TU Munich, 2007.

[SM98] Brett Stevens und Eric Mendelsohn. Efficient Software Testing Protocols. In *CASCON '98: Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research*, Seite 22. IBM Press, 1998.

[Sta00] Judith A. Stafford. A Formal, Language-Independent, and Compositional Approach to Interprocedural Control Dependence Analysis, 2000.

[TTK04] A. Tevanlinna, J. Taina und R. Kauppinen. Product Family Testing: a Survey. *ACM SIGSOFT Software Engineering Notes.*, 29, 2004.