Zwei Ansätze zur automatischen modellbasierten Generierung von Testfällen für variantenreiche Systeme

Stephan Weißleder and Hartmut Lackner
Fraunhofer-Institut FOKUS
Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany
stephan.weissleder / hartmut.lackner @ fokus.fraunhofer.de

7. Januar 2013

Zusammenfassung

Heutige Systeme werden immer komplexer. Damit werden auch die Entwicklung und der Test immer aufwändiger. Entsprechende Komplexitätstreiber gibt es viele. In diesem Artikel fokussieren wir auf den Komplexitätstreiber der ständig steigenden Anzahl von Produktvarianten, wie er insb. im mobilen Umfeld häufig anzutreffen ist. Wir beschreiben zwei Ansätze, wie man für die Qualitätssicherung variantenreicher Systeme automatische Testdesignmethoden wie das modellbasierte Testen anwenden kann, und vergleichen diese kurz. Alle Erläuterungen werden mit einem Beispiel untermauert.

1 Einleitung

Verbraucher erwarten heutzutage, dass sich Produkte an ihre Bedürfnisse anpassen. Hersteller reagieren darauf mit einer Vielzahl von konfigurierbaren Produktmerkmalen (Features), die sich auf das Erscheinungsbild und den Preis des Produktes niederschlagen [7]. Typische Beispiele für variantenreiche Produkte finden sich im Automobilbereich und im mobilen Umfeld wie z.B. den Smartphones. Die wirtschaftliche Entwicklung einer Produktlinie setzt einen hohen Wiederverwendungsgrad voraus, da die separate Entwicklung der Varianten zu aufwändig ist. Die Nutzung von Modellen wie z.B. Feature-Modelle unterstützen diese Art der Systementwicklung.

Die Qualitätssicherung und insb. das Testen heutiger Systeme ist ein umstrittener Aspekt der Systementwicklung und zentraler Bestandteil des Risikomanagements. Sofort zu Buche schlagende Kosten und die nicht sofort sichtbaren Effekte der Qualitätssicherung stehen möglichen Risiken und Folgekosten nicht entdeckter Fehler gegenüber. Mit dem Testen ist man also niemals wirklich fertig und es gibt sehr oft noch nicht getestete Aspekte. Jede Form der Effizienzsteigerung des Testens ist also willkommen. In der Praxis hat sich das modellbasierte Testen als effizienz- und effektivitätssteigernde Form des Testdesgins herausgestellt [9].

Der Fokus des Artikels liegt in der Verknüpfung der beiden zuvor genannten Themen: Wir präsentieren zwei verschiedene Ansätze, das Testdesign auch für variantenreiche Systeme zu automatisieren. Dazu ist dieser Artikel wie folgt strukturiert: In Abschnitt 2 präsentieren wir die Grundlagen für unsere Arbeit anhand eines Onlineshop-Beispiels. Im Anschluss daran beschreiben wir zwei Vorgehensweisen zur Testfallherleitung für variantenreiche Systeme in Abschnitt 3 und vergleichen sie in Abschnitt 4. Abschließend fassen wir den Artikel in Abschnitt 5 zusammen und skizzieren unsere geplanten weiteren Arbeiten.

2 Grundlagen

In diesem Abschnitt beschreiben wir kurz die Grundlagen des Artikels. Wir präsentieren eine Produktlinie von Onlineshops als Beispiel für ein variantenreiches System, für das modellbasiert automatisch Testfälle generiert werden sollen. Für dieses Beispiel nehmen wir an, dass eine Firma Onlineshops verkauft. Der Preis richtet sich nach den unterstützten Features. Für unser Beispiel bieten alle Onlineshops einen Katalog, in dem die erhältlichen Produkte ausgewählt werden können. Weiterhin bietet jeder Shop mindestens eine der drei Bezahlmethoden Einzug per Bankverbindung, ECoins und Kreditkarte. Jeder Shop kann unsichere oder sichere Kommunikation anbieten. Für die Bezahlung per Kreditkarte ist eine sichere Kommunikation notwendig. Weiterhin optional ist eine komfortable Suchfunktion, die das Finden von Produkten nach verschiedenen Kriterien erlaubt. Gerade im mobilen Bereich gibt es dutzende weiterer Unterscheidungsmerkmale zwischen verschiedenen Verkaufsplattformen. Unser Beispiel beschränkt sich aber auf die beschriebenen Aspekte.

Zur Beschreibung des angeführten Beispiels nutzen wir Feature-Modelle wie im oberen Teil von Abbildung 1 gezeigt. Feature-Modelle sind Graphen, die die Features einer Produktlinie auf abstraktem Niveau beschreiben. Das oberste Kästchen beschreibt hier den Namen der Produktlinie. Alle weiteren Kästchen bezeichnen Features. Sie können durch verschiedene Relationen verbunden sein. Die Relationen mit dem ausgefüllten Kreis am Ende

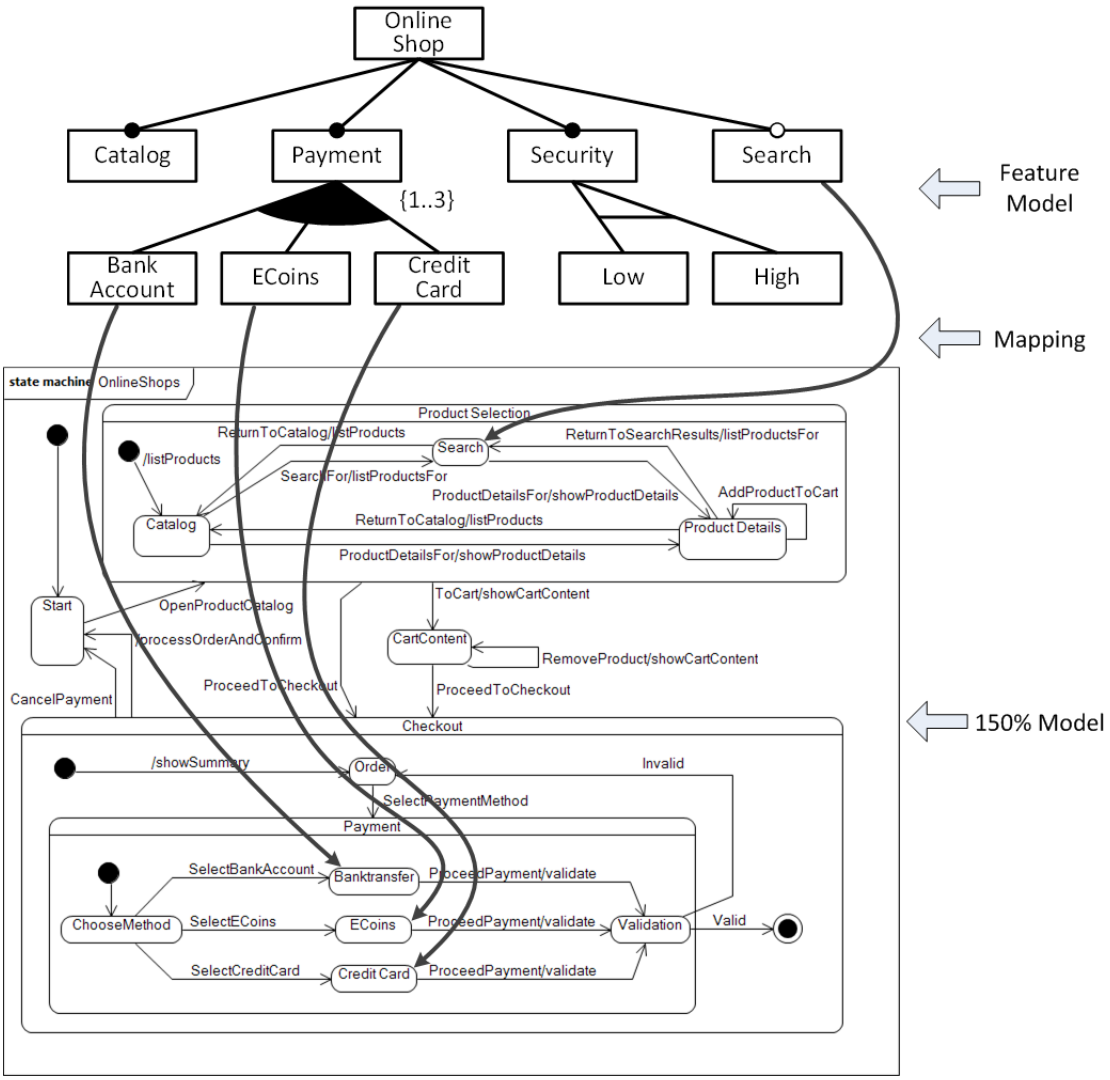

Abbildung 1: Mapping von Feature-Modell zu 150%-Zustandsmaschine.

bedeuten eine obligatorische Integration in jede Produktvariante (mandatory), sodass in unserem Beispiel die Features *Catalog*, *Payment* und *Security* in jeder Variante enthalten sind. Die Relation mit offenem Kreis am Ende bedeutet die optionale Integration dieses Features in eine Variante, sodass hier *Search* optional ist. Eine Auswahl von verschiedenen Features kann so wie für die Features *Bank Account*, *ECoins* und *Credit Card* dargestellt werden. Hier sind eins bis drei dieser Bezahlmöglichkeiten pro Variante erlaubt. Alternativen werden z.B. wie für die Features *Low* und *High* dargestellt. Feature-Modelle sind eine leicht verständliche Beschreibung der Features einer Produktlinie. Entsprechende Werkzeuge sind ebenfalls erhältlich [6]. Um für die Systementwicklung aber wirklich von Nutzen zu sein, müssen die Feature-Modelle mit tatsächlich genutzten Entwicklungsartefakten wie z.B. mit Anforderungen in DOORS oder mit Modellen verknüpft werden. In diesem Artikel werden wir Feature-Modelle mit Modellen für die automatische Testerzeugung verknüpfen.

Das Testen ist eine der wichtigsten qualitätssichernden Maßnahmen. Um die Aufwände für das Testen zu reduzieren, bietet sich u.A. die Automation von Testdesign und Testausführung an. Das automatische Testdesign unter Verwendung von Modellen ist auch als modellbasiertes Testen bekannt. Hierbei werden aus Modellen automatisch Testfälle abgeleitet. In unserem Artikel konzentrieren wir uns auf Testgenerierung unter Verwendung von Zustandsautomaten der Unified Modeling Language (UML) [4] wie unten in Abbildung 1 gezeigt.

Die Idee, modellbasierte Testgenerierung für variantenreiche Systeme anzuwenden, ist relativ eingängig: Wie weiter oben beschrieben, sind Feature-Modelle nur dann sinnvoll einzusetzen, wenn sie mit anderen Entwicklungsartefakten verknüpft werden. In unserem Fall verknüpfen wir die Features des Feature-Modells also mit Elementen von UML Zustandsmaschinen, die für den Test genutzt werden. Man verwendet dazu eine sogenannte 150%-Zustandsmaschine, die das Verhalten aller Produkte der Produktfamilie beschreibt [10]. Durch die Selektion einzelner Features werden die damit verknüpften Elemente der 150%-Zustandsmaschine eben-

falls ausgewählt. In der Zustandsmaschine für die auf Feature-Modell-Ebene ausgewählten Produktvariante sind dann nur die für diese Produktvariante relevanten Bestandteile enthalten. Man spricht dann von einer 100%-Zustandsmaschine. Abbildung 1 zeigt ein solches Mapping für unser Beispiel.

3 Zwei Ansätze zur Testgenerierung für Produktfamilien

In den vorherigen Abschnitten haben wir die allgemeine Verwendung von Feature-Modellen, das automatische Ableiten von Testfällen aus Modellen und die Verknüpfung beider Modellarten skizziert. In diesem Abschnitt untersuchen wir, auf welche Weise wir diese Verknüpfungen für die Testfällerzeugung für variantenreiche Systeme ausnutzen können. Für dieses Ziel gibt es viele verschiedene Vorgehensweisen. Wir fokussieren uns im Folgenden auf zwei davon, die wir *Top-Down* und *Bottom-Up* nennen. Die Abbildungen 2 und 3 skizzieren beide Ansätze.

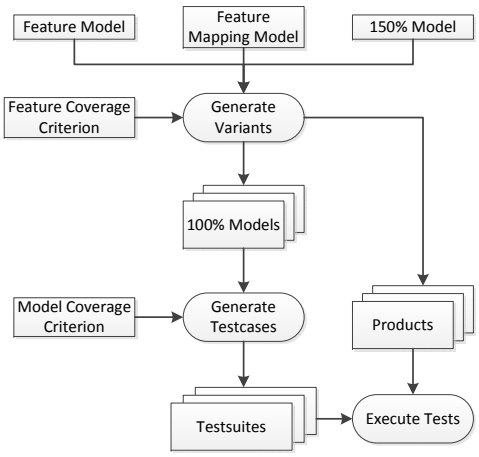

Abbildung 2: Top-Down-Ansatz.

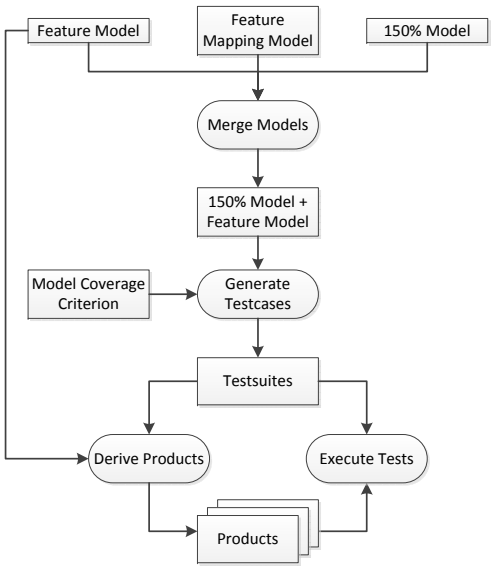

Abbildung 3: Bottom-Up-Ansatz.

Die Idee des Top-Down-Ansatzes besteht aus zwei Schritten: Zuerst wird eine repräsentative Menge von Varianten auf Basis des Feature-Modells augewählt. Die kann z.B. unter Verwendung von Überdeckungskriterien auf Feature-Modell-Ebene geschehen [3]. Im nächsten Schritt wird über die Verknüpfung zwischen Feature-Modell und 150%-Zustandsmaschine zu allen erzeugten Varianten die jeweils zugehörige 100%-Zustandsmaschine erzeugt. Für alle diese Zustandsmaschinen werden dann bekannte Ansätze wie die Anwendung von Überdeckungskriterien auf Zustandsmaschinen [8] angewendet. Die Motivation für diesen Ansatz liegt in der Einfachheit des Ansatzes. Allerdings vermuten wir, dass hiermit Teile der 150%-Zustandsmaschine, die für alle Varianten gültig sind, auch für alle erzeugten Varianten immer wieder getestet werden.

Die Idee des Bottom-Up-Ansatzes ist es, die Informationen des Feature-Modells zuerst in der 150%-Zustandsmaschine direkt mit aufzunehmen. Dazu werden z.B. zusätzliche Wächterbedingungen eingeführt oder bestehende erweitert, um die Abhängigkeiten zwischen den Variationspunkten im Feature-Modell auf die Variationspunkte in der Zustandsmaschine zu übertragen. Dazu können z.B. neue Variablen eingeführt werden, deren Wert jeweils der Selektion oder Deselektion eines Features entsprechen. Der Testgenerator erzeugt dann Testfälle basierend auf der erweiterten 150%-Zustandsmaschine. In diesem Generierungsprozess werden auch die zusätzlich eingeführten Variablen mit Werten belegt, sodass die zu den einzelnen Testfällen zugehörigen Varianten gleich mit erzeugt werden. Die Motivation für diesen Ansatz besteht darin, das mehrfache Testen von Verhalten, die in allen Varianten vorkommen, zu verhindern und eine effizientere Testmenge zu generieren.

4 Vergleich

In diesem Abschnitt vergleichen wir die beiden genannten Ansätze anhand unseres Beispiels kurz. Dazu verwenden wir zwei Betrachtungsweisen: Zuerst betrachten wir die möglichen Testergebnisse die ein menschlicher Tester (in dem Falle: wir) erzeugen würde. Im zweiten Schritt haben wir die Modelle zu den zwei Ansätzen erzeugt und zumindest die automatische Testgenerierung auf Basis der 100%-Modelle (für Top-Down) und des erweiterten 150%-Modells (für Bottom-Up) mit dem Conformiq Designer [2] durchgeführt.

Für den Top-Down-Ansatz haben wir 2 Varianten erzeugt (eine für Credit Card und High Security und eine für Bank Account, ECoins und Low Security). Für die beiden entsprechenden 100%-Modelle haben wir jeweils

einen Testfall erzeugt. Diese Testfälle hätten eine Länge von 19 bzw. 24 eingehenden Aufrufen. Conformiq liefert für die Modelle einmal sieben Testfälle mit 28 Aufrufen und einmal 11 Testfälle mit 42 Aufrufen.

Für den Bottom-Up-Ansatz erzeugen wir das erweiterte 150%-Modell. Manuell erzeugen wir einen Testfall, der 22 Aufrufe enthält. Die automatische Testgenerierung liefert 12 Testfälle mit insgesamt 59 Aufrufen.

In beiden Fällen wurden alle Transitionen der 150%-Zustandsmaschine überdeckt. Der Bottom-Up-Ansatz führte bei manuellem und automatischem Testdesign zur effizienteren Testmenge. Aufgrund der in Conformiq implementierten Breitensuche war der Unterschied zwischen manuellen und automatisch erzeugten Ergebnissen erwartet. Die gemessenen Zahlen sind ein Indiz für die anfangs dargestellte Vermutung, dass der Bottom-Up-Ansatz zu einer effizienteren Testmenge liefert.

5 Zusammenfassung, Diskussion und Ausblick

In diesem Artikel haben wir zwei Ansätze vorgestellt, um Testfälle für variantenreiche Systeme abzuleiten. Wir haben Messergebnisse zu beiden Ansätzen auf Grundlage manueller und automatischer Testfallerzeugung erstellt und verglichen. Für unser Beispiel führt der Bottom-Up-Ansatz zu einer effizienteren Testmenge.

Es gibt einige weitere Prozesse für den modellbasierten Test variantenreicher Systeme. So z.B. nutzen Olimpiew und Gomaa [5] Feature-Modelle und Sequenzdiagramme zur Testfallherleitung. Der Ansatz unterscheidet sich jedoch deutlich von unserem, zumal Sequenzdiagramme weniger Ausdrucksstärke besitzen als Zustandsdiagramme. Lochau et al. [3] wiederum fokussieren sich stark auf Abdeckungskritierien für Feature-Modelle. In dem von uns präsentierten Top-Down Ansatz benötigen wir genaus solche Kritierien zur Vorauswahl der Varianten.

Cichos et al. [1] arbeiten an einem Verfahren, dass unserem Bottom-Up Ansatz sehr ähnlich ist. Jedoch benötigen sie eine vordefinierte Menge von Varianten um aus dem 150%-Modell tests generieren zu können, die sie dem Testgenerator als zusätzliche Parameter übergeben. Die Autoren geben an, dass dieses Verfahren bislang von keinem kommerziellen Testgenerator unterstützt wird. Im Gegensatz dazu sind unsere Ansätze beide mit kommerziell verfügbaren Testgeneratoren realisierbar, wie in unserem Beispiel anhand Conformiq [2] gezeigt.

Die in diesem Artikel vorgestellten Ergebnisse wurden teils manuell und teils automatisch erzeugt. Momentan arbeiten wir an einer Werkzeugkette, die beide Ansätze unterstützen soll. Sie wird das Verbindungsstück zwischen existierenden Werkzeugen für Feature-Modellierung und Testgenerierung sein. Damit sind dann auch weiterführende Fallstudien möglich. Weiterhin gibt es einige Punkte, die in den weiterführenden Fallstudien zu berücksichtigen sein werden: Zum Beispiel ist die Frage, wie stark die Wiederverwendung gleicher Komponenten oder Features auch auf Quellcode-Ebene umgesetzt wurde. Wenn stattdessen des Öfteren Copy&Paste eingesetzt wurde, dann reicht es nicht, die Überdeckung auf Modellebene zu betrachten. Dann wird die Beziehung zwischen Features und Quellcode-Elementen ebenfalls einfließen müssen.

Literatur

[1] H. Cichos, S. Oster, M. Lochau, and A. Schürr. Model-based Coverage-Driven Test Suite Generation for Software Product Lines. In *Proceedings of the ACM/IEEE 14th International Conference on Model Driven Engineering Languages and Systems*, volume 6981 of *Lecture Notes in Computer Science (LNCS)*, pages 425–439, Heidelberg, 2011. Springer Verlag.

[2] Conformiq. Designer 4.4. http://www.conformiq.com/.

[3] Malte Lochau, Sebastian Oster, Ursula Goltz, and Andy Schürr. Model-based pairwise testing for feature interaction coverage in software product line engineering. *Software Quality Journal*, 20(3-4):567–604, 2012.

[4] Object Management Group. Unified Modeling Language (UML), version 2.4. `http://www.uml.org`, 2011.

[5] Erika Mir Olimpiew and Hassan Gomaa. Model-Based Testing for Applications Derived from Software Product Lines. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–7, 2005.

[6] pure systems. pure::variants. `http://www.pure-systems.com`, 2012.

[7] Carnegie Mellon University. Software product lines. `http://www.sei.cmu.edu/productlines/`, 2012.

[8] Mark Utting and Bruno Legeard. *Practical Model-Based Testing: A Tools Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[9] Stephan Weißleder, Baris Güldali, Michael Mlynarski, Arne-Michael Törsel, David Faragó, Florian Prester, and Mario Winter. Modellbasiertes Testen: Hype oder Realität? *OBJEKTspektrum*, 2011.

[10] Stephan Weißleder, Dehla Sokenou, and Holger Schlingloff. Reusing State Machines for Automatic Test Generation in Product Lines. In Axel Rennoch Thomas Bauer, Hajo Eichler, editor, *Model-Based Testing in Practice (MoTiP)*. Fraunhofer IRB Verlag, June 2008.