Spezifikation und Verifikation von Produktlinien mit Feature-orientierten Verträgen[1]

Thomas Thüm[2]

Abstract: Variabilität ist allgegenwärtig in der heutigen Softwareentwicklung. Während Techniken zur effizienten Implementierung von Software-Produktlinien seit Jahrzehnten eingesetzt werden, waren Verifikationstechniken ein Forschungsschwerpunkt in den letzten Jahren. Wir geben eine Übersicht darüber, wie existierende Verifikationstechniken auf Produktlinien angewandt worden. Mithilfe unserer Erkenntnisse beheben wir zwei Defizite früherer Forschungsarbeiten. Einerseits untersuchen wir systematisch wie Verträge zur Produktlinienspezifikation genutzt werden können. Unsere theoretische Diskussion und praktische Evaluierung führt uns zum Konzept der Feature-orientierten Verträge. Insbesondere haben wir beobachtet, dass viele aber nicht alle Feature-orientierten Verträge dem liskovschen Substitutionsprinzip folgen. Anderseits nutzen wir Feature-orientierte Verträge zum Vergleich verschiedener Verifikationstechniken und -strategien für dieselbe Produktlinienimplementierung und -spezifikation. Bei der Kombination von Theorembeweisern und Modellprüfern konnten wir Synergien für die Produktlinienverifikation messen.

1 Einführung

Heutige Software wird meist nicht von Grund auf neu entwickelt, sondern aufbauend auf existierender Software [RC13, LC13, XXJ12, HK12, An14]. Widersprüchliche funktionale und nicht-funktionale Anforderungen zwingen Entwickler Software in Varianten zu entwickeln. Die Ähnlichkeiten und Unterschiede zwischen diesen Varianten werden mit Software-Produktlinien explizit als Features modelliert [PBvdL05, CN01]. Mit generativer Programmierung kann jedes Software-Produkt für eine gegebene Feature-Auswahl automatisch generiert werden [CE00, BSR04, Ap13]. Ein typisches Beispiel für eine Produktlinie ist ein Betriebssystem [Pa76] wie z.B. Linux, aber es gibt viele weitere Anwendungsdomänen [We08] in denen Rentabilität, Entwicklungszeit und Qualität durch Produktlinien verbessert werden können [CN01, vdLSR07, Lu07].

Software-Produktlinien werden zunehmend für sicherheitskritische Systeme eingesetzt [We08]. Existierende Verifikationstechniken wie Theorembeweisen, Modellprüfen und Typprüfung können benutzt werden, um jedes Produkt einzeln zu verifizieren. Durch die Ähnlichkeiten der Produkte entsteht dabei jedoch redundanter Verifikationsaufwand. Zudem steigt die Anzahl der generierbaren Produkte häufig exponentiell in der Anzahl der Features, sodass die Verifikation von jedem Produkt einzeln nicht skaliert [Li13]. Effizientere Ansätze wurden in verschiedenen Forschungsbereichen vorgestellt, zuerst für Modellprüfung (engl. model checking) [FK01, NCA01], dann für Typprüfung (engl.

[1] Englischer Titel der Dissertation: “Product-Line Specification and Verification with Feature-Oriented Contracts” [Th15]

[2] TU Braunschweig, t.thuem@tu-braunschweig.de

type checking) [APB02] und Theorembeweisen (engl. theorem proving) [Po07]. Seither wurden viele weitere Ansätze für jede Verifikationstechnik vorgeschlagen. Die dabei verwendeten unterschiedlichen Terminologien erschweren das Verständnis ihrer Unterschiede und deren systematische Anwendung in Forschung und Industrie.

Beim Erstellen einer Übersicht der Forschungsliteratur zur Produktlinienverifikation konnten wir Schwächen bisheriger Ansätze feststellen, von denen wir zwei adressieren. Erstens fokussiert die Literatur auf effizienten Verifikationstechniken wobei Spezifikationstechniken oft nicht empirisch validiert werden. Allerdings ist die Spezifikation des gewünschten Verhaltens entscheidend für viele Verifikationstechniken [CGP99, Sc01, Sm85]. Zweitens wurden Verifikationstechniken bisher nicht miteinander kombiniert, um die gleichen Eigenschaften einer Produktlinie zu überprüfen. Solche Kombinationen können jedoch die Effektivität und Effizienz der Verifikation erhöhen. Unser langfristiges Ziel ist, dass Verifikationstechniken und -strategien basierend auf statischen Eigenschaften einer Produktlinie wie die Anzahl, Größe und Kohäsion von Features empfohlen werden können. Diese Doktorarbeit liefert drei Hauptbeiträge, um dieser Vision näher zu kommen:

- Wir schlagen eine Klassifikation von Techniken zur Produktlinienverifikation vor, welche die zu Grunde liegenden Strategien zur Implementierung, Spezifikation und Verifikation identifiziert. Basierend auf unseren Einsichten beim Klassifizieren von 137 Ansätzen, entwickeln wir eine Forschungsagenda.
- Wir diskutieren systematisch, wie Methodenverträge zur Spezifikation von Produktlinien genutzt werden können. Wir präsentieren eine Taxonomie für und Mechanismen zur Feature-orientierten Vertragskomposition und evaluieren diese Anhand von 14 Produktlinien.
- Wir stellen eigene Verifikationstechniken für Feature-orientierte Verträge auf und evaluieren diese. Insbesondere konnten wir Synergien bei der Kombination von Theorembeweisern und Modellprüfern in Bezug auf die Effizienz und Effektivität für Produktlinien messen.

2 Klassifikation von Verifikationstechniken für Produktlinien

Mit Hilfe von Software-Produktlinien werden ähnliche Software-Produkte effizient und basierend auf wiederverwendbaren Artefakten erstellt. Während es eine Vielzahl von effizienten Strategien zur Implementierung von Produktlinien gibt [CE00, Ap13], ist ein aktueller Forschungsgegenstand wie Verifikationstechniken, beispielsweise Typprüfung, Datenflussanalysen, Modellprüfung und Theorembeweisen, von einzelnen Programmen auf Produktlinien skaliert werden können.

Wir klassifizieren Ansätze zur Produktlinienverifikation in drei Hauptstrategien: produktbasierte (engl. product-based), Feature-basierte (engl. feature-based) und familienbasierte Analysen (engl. family-based analyses). Mit der produktbasierten Strategie werden alle oder eine repräsentative Teilmenge der Produkte generiert und verifiziert, wobei es zu vielen redundanten Berechnungen kommt. Hingegen wird bei der Feature-basierten Strategie

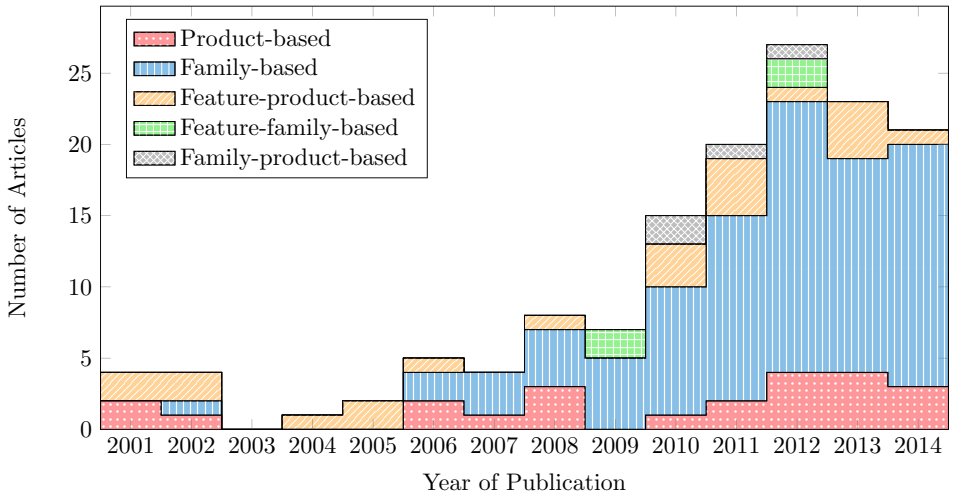

Abb. 1: Häufigkeit der Analysestrategien in der Forschungsliteratur [Th15].

jedes Features in Isolation analysiert, wodurch jedoch Fehler durch Feature-Interaktionen nicht aufgedeckt werden können. Alle Features und deren Abhängigkeiten werden bei der familienbasierten Strategie zusammen analysiert, was allerdings deutlich mehr Speicher benötigt als die anderen Strategien.

Abb. 1 zeigt, dass der Großteil der Forschungsliteratur familienbasierte und produktbasierten Strategien vorschlägt, aber die Feature-basierte Strategie in der reinen Form nie aufgetreten ist. Stattdessen konnten wir Kombinationen dieser drei Hauptstrategien ausfindig machen. Besonders häufig wurde dabei die Feature-basierte mit der produktbasierten Strategie zu einer Feature-produktbasierten Strategie kombiniert. Dabei werden erst Features in Isolation verifiziert und dann die Verifikationsergebnisse für die Verifikation aller Produkte genutzt, wo dann auch Feature-Interaktionen verifiziert werden können.

Die Klassifikation von 137 Ansätzen zur Verifikation von Produktlinien hat uns zu neuen Einsichten geführt. Erstens gibt es sehr vielfältige Ansätze die damit beworben werden, dass sie kompositional sind. Wir unterscheiden zwischen Feature-produktbasierten und Feature-familienbasierten Strategien, um zu enthüllen wie diese Ansätze mit inhärent nicht-kompositionalen Eigenschaften wie Feature-Interaktionen umgehen. Zweitens haben wir identifiziert, dass nicht alle Strategien auf alle Verifikationstechniken angewandt wurden, womit sich die Frage stellt, ob diese neuen Kombinationen prinzipiell möglich und effizient sind. Basierend auf diesen und weiteren Einsichten haben wir eine Forschungsagenda entwickelt, für die wir auf die eigentliche Doktorarbeit verweisen [Th15]. Wir hoffen, dass unsere Klassifikation die Bedeutung und Herausforderungen von Produktlinienverifikation verdeutlicht, Diskussionen mit einer gemeinsamen Terminologie vereinfacht, und Forscher motiviert neue Kombinationen zu explorieren. Wir verweisen interessierte Leser auch auf unsere Webseite wo wir neue Ansätze klassifizieren.[3]

[3] `http://fosd.net/spl-strategies/`

```
public class Graph {                                          feature module Base
  private Collection<Node> nodes;
  private Collection<Edge> edges;
  /*@ requires edge != null && nodes.contains(edge.first)
    @   && nodes.contains(edge.second);
    @ ensures hasEdge(edge); @*/
  public void addEdge(Edge edge) {
    edges.add(edge);
  }
  [...]
}
```

```
public class Graph {                                          feature module MaxEdges
  private static Integer MAXEDGES = new Integer(10);
  /*@ requires \original && MAXEDGES != null;
    @ ensures \old(edges.size()) < MAXEDGES ==> \original; @*/
  public void addEdge(Edge edge) {
    if(countEdges() < MAXEDGES)
      original(edge);
  }
}
```

Abb. 2: Beispiel für Feature-orientierte Verträge mit expliziter Vertragsverfeinerung [Th15].

3 Spezifikation mit Feature-orientierten Verträgen

Die meisten auf Produktlinien angewandten Verifikationstechniken benötigen eine Spezifikation des gewünschten Verhaltens für jedes Produkt. Während es einige Spezifikationsansätze für Produktlinien gibt, sind diese häufig nur Mittel zum Zweck und nicht empirisch validiert. Um dies zu ändern, erstellen wir die erste tiefgreifende systematische Diskussion und Evaluierung, wie Produktlinien spezifiziert werden können und sollten. Damit stellt unsere Forschung die Grundlage für andere Forschungsbereiche dar, die auch auf Spezifikationen angewiesen sind, wie beispielsweise die formale Verifikation, Feature-Interaktionserkennung, und Testfallgenerierung für Produktlinien.

Unsere Betrachtungen stützen sich auf Verträge zur Spezifikation des gewünschten Verhaltens. Verträge können genutzt werden um direkt im Quelltext Spezifikationen anzugeben und ermöglichen die Schuldzuweisung (engl. blame assignment), welche hilft die Fehlerursache zu lokalisieren [Me88, Ha12]. Die obere Hälfte in Abb. 2 zeigt ein Beispiel für einen Vertrag, welcher der Methode `addEdge` zugeordnet ist. So besteht ein Vertrag aus einer Vorbedingung, die der Aufrufer einer Methode sicherstellen muss und aus einer Nachbedingung, welche die Methode selbst erfüllen muss. Zudem bieten Verträge zahlreiche Verifikationsmöglichkeiten, wie beispielsweise Theorembeweisen [Bu05, BHS07, Ba11, Ha12], Modellprüfung [Ro06], Datenflussanalysen [Bu05, Ha12], Überprüfung auf Laufzeitbedingungen [Me88, Bu05, Ba11, Ha12], sowie die Testfallgenerierung [Bu05, Ha12].

In Produktlinien verhalten sich Methoden jedoch nicht immer gleich und daher muss auch Variabilität in Verträgen modelliert werden können. In Abb. 2 geben wir ein Bei-

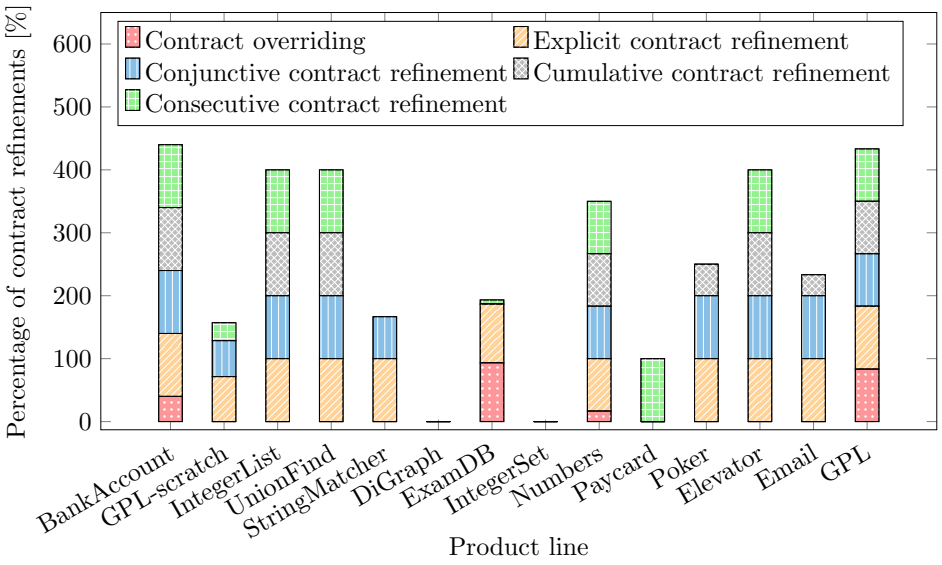

Abb. 3: Anwendbarkeit verschiedener Mechanismen für Variabilität in Verträgen [Th15].

spiel, wie die Basisimplementierung einer Klasse durch das Feature namens *MaxEdges* verändert werden kann. Dieses Beispiel bedient sich der Feature-orientierten Programmierung [Pr97, BSR04], in der Klassen auf Feature-Module aufgeteilt werden. Wenn das Feature *MaxEdges* gewählt ist, so wird die ursprüngliche Implementierung der Methode `addEdge` derart verändert, dass die maximale Anzahl an Kanten in einem Graph nie überschritten werden kann. Ziel der in der Dissertation vorgeschlagenen Feature-orientierten Verträge ist es nun, dieses veränderte Verhalten auch zu spezifizieren. Dazu wird ein neuer Vertrag aus Vorbedingung und Nachbedingung angegeben, wobei – ähnlich wie in der Implementierung – die ursprünglichen Vorbedingung und Nachbedingung mittels Schlüsselwort `original` referenziert werden können.

Unser Beispiel besteht nur aus zwei Features, *Base* ist notwendig und *MaxEdges* optional, wodurch auch nur zwei verschiedene Produkte generiert werden können. Generell erlauben jedoch n optionale, unabhängige Features die Generierung von bis zu 2^n verschiedenen Produkten. Mit Feature-orientierten Verträgen können wir für jedes der Produkte auch eine maßgeschneiderte Spezifikation generieren. Die im Beispiel gezeigte explizite Vertragsverfeinerung (engl. explicit contract refinement) ist jedoch nur einer der Mechanismen, die in der Dissertation vorgestellt werden. Daneben gibt es noch einige implizite, die ohne das Schlüsselwort `original` auskommen, da Vorbedingungen und Nachbedingungen immer in bestimmter Weise miteinander verbunden werden (z.B. per Disjunktion oder Konjunktion). Abb. 3 zeigt wie oft jeder dieser Mechanismen auf bestimmte Verträge in verschiedenen Produktlinien angewendet werden konnte. Insbesondere wird hierbei deutlich, dass das reine Überschreiben von Verträgen (engl. contract overriding) – also explizite Vertragsverfeinerung ohne das Schlüsselwort `original` – meist nicht ausreicht, obwohl es oft für die aspektorientierte Programmierung vorgeschlagen wird.

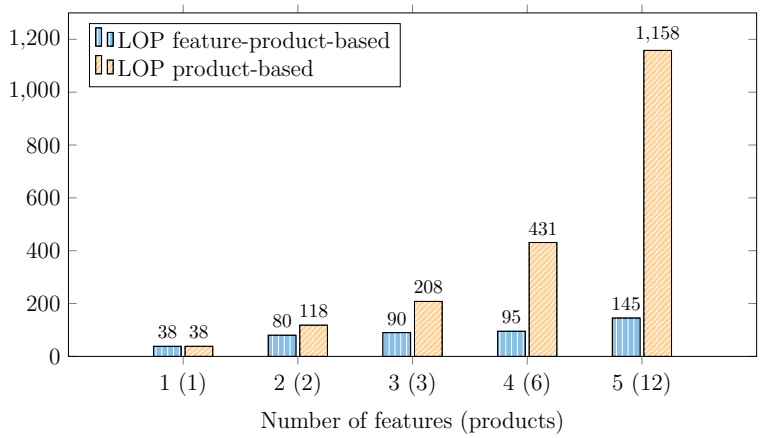

Abb. 4: Einsparung bei der Verifikation von Software-Produktlinien durch Beweiskomposition in Lines of Proof (LOP) [Th15].

4 Verifikation mit Feature-orientierten Verträgen

Da Feature-orientierte Verträge erlauben viele Produkte effizient zu spezifizieren, stellt sich natürlich auch unmittelbar die Frage, ob all diese Produkte auch ihre Spezifikation erfüllen. Wie oben erwähnt, ist jedoch die separate Verifikation von jedem einzelnen Produkt oft zu aufwändig. Insbesondere, wenn - wie bei der deduktiven Verifikation mit Theorembeweisern - Nutzerinteraktion nötig ist. Die Dissertation befasst sich im Wesentlichen mit zwei verbesserten Verifikationstechniken, die wir hier mit den Messergebnissen zusammen vorstellen wollen. Beide Techniken nutzen die deduktive Verifikation als Grundlage, da wir dies als neuen Bereich in unserer Literaturrecherche identifiziert haben.

Mit Feature-orientierten Verträgen haben wir die Feature-orientierte Programmierung um die Möglichkeit erweitert, nicht nur die Implementierung für jedes Produkt zu generieren, sondern auch die Spezifikation in Form von Verträgen. Mit der Beweiskomposition (engl. proof composition) führen wir diesen Gedanken fort und generieren auch maschinenlesbare Beweise für jedes Produkt. Dabei werden für jedes Feature Teilbeweise von Menschen geschrieben und dann zur Überprüfung für jedes Produkt kombiniert und von einem Beweisassistenten vollautomatisch überprüft. In Abb. 4 vergleichen wir die Ergebnisse der Verifikation einer Produktlinie mit Beweiskomposition, also einer Feature-produktbasierten Strategie, mit einer herkömmlichen produktbasierten Strategie. Dabei haben wir als Maß für den Aufwand zum Beweisen die Zeilen in Beweisskripten (engl. lines of proof) gezählt. Hierbei sind deutliche Einsparungen mit steigender Anzahl von Features ersichtlich; es wird somit der Aufwand beim Schreiben der Beweisskripte durch einen Menschen reduziert. Nichtsdestotrotz gibt es durch die Beweiskomposition keine Einsparung bei der Überprüfung der Beweisskripte für jedes Produkt durch ein Computerprogramm.

Um sowohl beim Beweisfinden als auch bei der Beweisüberprüfung von den Gemeinsamkeiten der Produkte zu profitieren, haben wir als weitere Technik Variabilitätskodierung

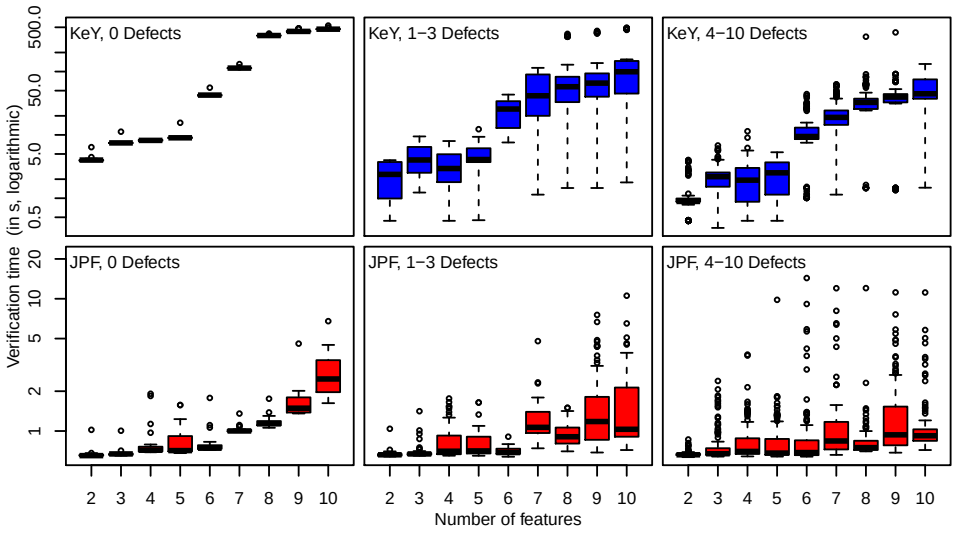

Abb. 5: Verifikationszeit mit Theorembeweisern und Modellprüfern für keine, wenig und viele Fehler in Software-Produktlinien [Th15].

(engl. variability encoding) erstmals auf deduktive Verifikation angewandt. Die Idee von Variabilitätskodierung ist es, zum Zweck der Verifikation die Variabilität zur Kompilierungszeit in Variabilität zur Laufzeit zu überführen. Dabei wird für jedes Feature eine neue boolesche Variable erzeugt, die repräsentiert ob das Feature gewählt ist oder nicht. Variabilitätskodierung ist somit eine Form der familienbasierten Analyse, bei der bestehende Verifikationswerkzeuge ohne Anpassungen für Produktlinien genutzt werden können.

In unserer Evaluierung haben wir den Theorembeweiser KeY und den Modellprüfer JPF erstmalig zur Überprüfung der gleichen Eigenschaft genutzt und verglichen. Dabei haben wir Produktlinien ohne Fehler, mit wenigen und mit vielen Fehlern in Form von Mutationen in Quelltext und Spezifikationen simuliert. In Abb. 5 zeigen wir die zur Verifikation benötigte Zeit und können dabei feststellen, dass unabhängig von der Größe der Produktlinie Fehler einen positiven Einfluss auf die Verifikationszeit haben. Für Modellprüfer entspricht das der Intuition, da die Verifikation beim ersten gefundenen Fehler abbricht und viele Fehler zu einem früheren Abbruch führen. Bei Theorembeweisern hätten wir vermutet, dass eine längere Zeit zum Beweisen nötig ist, aber intelligente Strategien zur Erkennung von Sackgassen in Beweisen resultieren ebenfalls in kürzerer Verifikationszeit.

5 Zusammenfassung

Der Erfolg von Software-Produktlinien hängt nicht nur von effizienten Techniken zur Programmgenerierung ab, sondern auch von effizienten und effektiven Verifikationsstrategien. Da Produktlinien immer mehr auch für sicherheitskritische Anwendungen eingesetzt werden, gewinnt die Verifikation von Produktlinien immer mehr an Bedeutung.

Während existierende Verifikationsansätze aus unterschiedlichen Forschungsbereichen stammen, schlagen wir eine einheitliche Terminologie vor und klassifizieren existierende Ansätze nach ihrer Strategie mit Variabilität in Implementierung, Spezifikation und bei der Analyse umzugehen. Eine interessante Einsicht ist, dass viele Arbeiten behaupten Kompositionalität zu erreichen, wobei wir zwischen kompositionaler Implementierung, kompositionaler Spezifikation und drei Arten kompositionaler Analyse unterscheiden. Unsere Klassifikation legt damit offen wie diese Ansätze mit den inhärent nicht-kompositionalen Feature-Interaktionen umgehen.

Bei unserer Literaturrecherche mussten wir feststellen, dass Spezifikationstechniken für Produktlinien bisher nur unzureichend betrachtet wurden. Daher haben wir mit Feature-orientierten Verträgen erstmals systematisch diskutiert und evaluiert wie wir Produktlinien spezifizieren können. Unsere wichtigste Einsicht ist, dass viele zur Verifikation genutzte Spezifikationstechniken zu restriktiv sind und daher oft nicht angewendet werden können. Mit Feature-orientierten Verträgen ermöglichen wir verschiedene Techniken für eine Produktlinie zu kombinieren und dabei Synergien zu erreichen.

Basierend auf Feature-orientierten Verträgen haben wir Beweiskomposition und Variabilitätskodierung als zwei Techniken betrachtet, um die Verifikation jedes Produktes einzeln zu vermeiden. Dabei reduziert die Beweiskomposition nur den Aufwand beim Schreiben der Beweisskripte, während Variabilitätskodierung auch die Überprüfung der Beweisskripte beschleunigt. In beiden Fällen konnten wir jedoch drastische Verbesserungen durch das Ausnutzen der Gemeinsamkeiten messen.

Literaturverzeichnis

[An14] Antkiewicz, Michał; Ji, Wenbin; Berger, Thorsten; Czarnecki, Krzysztof; Schmorleiz, Thomas; Lämmel, Ralf; Stănciulescu, Stefan; Wasowski, Andrzej; Schaefer, Ina: Flexible Product Line Engineering with a Virtual Platform. In: Proc. Int'l Conf. Software Engineering (ICSE). ACM, New York, NY, USA, S. 532–535, 2014.

[Ap13] Apel, Sven; Batory, Don; Kästner, Christian; Saake, Gunter: Feature-Oriented Software Product Lines: Concepts and Implementation. Springer, Berlin, Heidelberg, 2013.

[APB02] Aversano, Lerina; Penta, Massimiliano Di; Baxter, Ira D.: Handling Preprocessor-Conditioned Declarations. In: Proc. Int'l Working Conference Source Code Analysis and Manipulation (SCAM). IEEE, Washington, DC, USA, S. 83–92, Oktober 2002.

[Ba11] Barnett, Mike; Fähndrich, Manuel; Leino, K. Rustan M.; Müller, Peter; Schulte, Wolfram; Venter, Herman: Specification and Verification: The Spec# Experience. Comm. ACM, 54:81–91, Juni 2011.

[BHS07] Beckert, Bernhard; Hähnle, Reiner; Schmitt, Peter: Verification of Object-Oriented Software: The KeY Approach. Springer, Berlin, Heidelberg, 2007.

[BSR04] Batory, Don; Sarvela, Jacob N.; Rauschmayer, Axel: Scaling Step-Wise Refinement. IEEE Trans. Software Engineering (TSE), 30(6):355–371, 2004.

[Bu05] Burdy, Lilian; Cheon, Yoonsik; Cok, David R.; Ernst, Michael D.; Kiniry, Joseph; Leavens, Gary T.; Leino, K. Rustan M.; Poll, Erik: An Overview of JML Tools and Applications. Int'l J. Software Tools for Technology Transfer (STTT), 7(3):212–232, Juni 2005.

[CE00] Czarnecki, Krzysztof; Eisenecker, Ulrich: Generative Programming: Methods, Tools, and Applications. ACM/Addison-Wesley, New York, NY, USA, 2000.

[CGP99] Clarke, Edmund M.; Grumberg, Orna; Peled, Doron A.: Model Checking. MIT Press, Cambridge, Massachussetts, 1999.

[CN01] Clements, Paul; Northrop, Linda: Software Product Lines: Practices and Patterns. Addison-Wesley, Boston, MA, USA, 2001.

[FK01] Fisler, Kathi; Krishnamurthi, Shriram: Modular Verification of Collaboration-Based Software Designs. In: Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE). ACM, New York, NY, USA, S. 152–163, 2001.

[Ha12] Hatcliff, John; Leavens, Gary T.; Leino, K. Rustan M.; Müller, Peter; Parkinson, Matthew: Behavioral Interface Specification Languages. ACM Computing Surveys, 44(3):16:1–16:58, Juni 2012.

[HK12] Hemel, Armijn; Koschke, Rainer: Reverse Engineering Variability in Source Code Using Clone Detection: A Case Study for Linux Variants of Consumer Electronic Devices. In: Proc. Working Conf. Reverse Engineering (WCRE). IEEE, Washington, DC, USA, S. 357–366, 2012.

[LC13] Laguna, Miguel A.; Crespo, Yania: A Systematic Mapping Study on Software Product Line Evolution: From Legacy System Reengineering to Product Line Refactoring. Science of Computer Programming (SCP), 78(8):1010–1034, August 2013.

[Li13] Liebig, Jörg; von Rhein, Alexander; Kästner, Christian; Apel, Sven; Dörre, Jens; Lengauer, Christian: Scalable Analysis of Variable Software. In: Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE). ACM, New York, NY, USA, S. 81–91, August 2013.

[Lu07] Lutz, Robyn: Survey of Product-Line Verification and Validation Techniques. Bericht 2014/41221, NASA, Jet Propulsion Laboratory, La Canada Flintridge, CA, USA, Mai 2007.

[Me88] Meyer, Bertrand: Object-Oriented Software Construction. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1st. Auflage, 1988.

[NCA01] Nelson, Torsten; Cowan, Donald D.; Alencar, Paulo S. C.: Supporting Formal Verification of Crosscutting Concerns. In: Proc. Int'l Conf. Metalevel Architectures and Separation of Crosscutting Concerns. Springer, London, UK, S. 153–169, 2001.

[Pa76] Parnas, David L.: On the Design and Development of Program Families. IEEE Trans. Software Engineering (TSE), SE-2(1):1–9, Marz 1976.

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

[Po07] Poppleton, Michael: Towards Feature-Oriented Specification and Development with Event-B. In: Proc. Int'l Working Conf. Requirements Engineering: Foundation for Software Quality (REFSQ). Springer, Berlin, Heidelberg, S. 367–381, 2007.

[Pr97] Prehofer, Christian: Feature-Oriented Programming: A Fresh Look at Objects. In: Proc. Europ. Conf. Object-Oriented Programming (ECOOP). Springer, Berlin, Heidelberg, S. 419–443, 1997.

[RC13] Rubin, Julia; Chechik, Marsha: A Framework for Managing Cloned Product Variants. In: Proc. Int'l Conf. Software Engineering (ICSE). IEEE, Piscataway, NJ, USA, S. 1233–1236, Mai 2013.

[Ro06] Robby; Rodríguez, Edwin; Dwyer, Matthew B.; Hatcliff, John: Checking JML Specifications Using an Extensible Software Model Checking Framework. Int'l J. Software Tools for Technology Transfer (STTT), 8(3):280–299, Juni 2006.

[Sc01] Schumann, Johann: Automated Theorem Proving in Software Engineering. Springer, Berlin, Heidelberg, 2001.

[Sm85] Smith, Brian Cantwell: The Limits of Correctness. SIGCAS Comput. Soc., 14,15(1,2,3,4):18–26, Januar 1985.

[Th15] Thüm, Thomas: Product-Line Specification and Verification with Feature-Oriented Contracts. Dissertation, University of Magdeburg, Germany, Februar 2015.

[vdLSR07] van der Linden, Frank J.; Schmid, Klaus; Rommes, Eelco: Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer, Berlin, Heidelberg, 2007.

[We08] Weiss, David M.: The Product Line Hall of Fame. In: Proc. Int'l Software Product Line Conf. (SPLC). IEEE, Washington, DC, USA, S. 395, 2008.

[XXJ12] Xue, Yinxing; Xing, Zhenchang; Jarzabek, Stan: Feature Location in a Collection of Product Variants. In: Proc. Working Conf. Reverse Engineering (WCRE). IEEE, Washington, DC, USA, S. 145–154, 2012.

Thomas Thüm ist wissenschaftlicher Mitarbeiter am Institut für Softwaretechnik und Fahrzeuginformatik an der Technischen Universität Braunschweig. Er studierte Informatik mit Nebenfach Mathematik an der Otto-von-Guericke-Universität Magdeburg von 2004 bis 2010 und schloss dieses Studium mit Auszeichnung ab. Während des Studiums forschte er bereits fünf Monate an der University of Texas at Austin. Im Anschluss an das Studium promovierte er in Magdeburg im Bereich Softwaretechnik unter Anleitung von Prof. Gunter Saake. Die am 23. Februar 2015 verteidigte Doktorarbeit wurde mit dem Prädikat summa cum laude bewertet. Thomas Thüm erhielt mit dieser Arbeit den Preis für die beste Dissertation der Fakultät für Informatik und den mit 1.000 Euro dotierten Dissertationspreis der Otto-von-Guericke-Universität. Seine Diplomarbeit erhielt bereits den mit 2.000 Euro dotierten Software-Engineering-Preis der Ernst-Denert-Stiftung. Er hat im Zuge seiner Dissertation an mehr als 40 wissenschaftlichen Publikationen mitgewirkt, darunter sind auch zahlreiche Veröffentlichungen als Erstautor in hochrangigen Zeitschriften und Konferenzen wie z.B. ACM Computing Surveys oder der International Conference on Software Engineering. Zudem hat er sich eingesetzt für den Wissenstransfer von Forschung zu Lehre und industrieller Softwareentwicklung. Die Open-Source-Entwicklungsumgebung FeatureIDE hat er bereits seit 2007 mitgeprägt und darin auch zahlreiche seiner Forschungsprototypen integriert. FeatureIDE wird seit Jahren an Universitäten weltweit für Forschung und Lehre verwendet sowie seit einigen Monaten auch in der Industrie.