Sichere Produktlinien: Herausforderungen für Syntax- und Typ-Prüfungen

Christian Kästner[1], Sven Apel[2], and Gunter Saake[1]

[1] School of Computer Science, University of Magdeburg, ckaestne/saake@ovgu.de
[2] Dept. of Informatics and Math., University of Passau, apel@uni-passau.de

Softwareproduktlinien stellen neue Herausforderungen an Entwicklungswerkzeuge und Programmiersprachen. Während in der klassischen Anwendungsentwicklung genau ein Programm entwickelt wird, entwickelt man in einer Softwareproduktlinie gleich eine Vielzahl verwandter Programme in einer Domäne aus gemeinsamen, wiederverwendbaren Artefakten [15, 3, 5]. Die Gemeinsamkeiten und Unterschiede der Programme einer Software-Produktlinie werden durch Features modelliert [10, 5]. Durch Kombination von Features können (potentiell tausende) maßgeschneiderte Programme spezifiziert und entsprechend generiert werden. Typische Implementierungsansätze die eine automatisierte Generierung von Programmen aus einer Produktlinie erlauben sind Frameworks [9], Präprozessoren (#ifdef) [15, 7], Generatoren [8] oder spezielle Sprachen der Aspekt- oder Feature-Orientierten Programmierung [14, 16, 4, 2].

Software-Produktlinien erlauben die Maßschneiderung von Programmen auf bestimmte Anwendungsszenarien bei hoher Wiederverwendung, und versprechen somit schnellere Entwicklung von maßgeschneiderten Lösungen bei geringeren Kosten [15, 3, 5]. Diesen Vorteilen stehen aber auch neue Herausforderungen gegenüber: eines der wichtigsten Probleme bei der Implementierung von Softwareproduktlinien ist, dass Fehler sehr schwierig zu finden sind wenn sie nur in einzelnen generierten Programmen mit bestimmten Features oder Feature-Kombinationen auftauchen [6, 17, 11, 13]. So schleichen sich Syntax-Fehler (z. B. vergessene schließende Klammer), Typ-Fehler (z. B. ins leere zeigende Methodenaufrufe oder fehlende Klassen), und auch Semantik-Fehler (z. B. fehlerhaftes Verhalten bei bestimmten Featurekombinationen) bei vielen Implementierungsansätzen leicht in einem generierten Programm ein. Oft werden diese Fehler aber erst bemerkt wenn ein Kunde tatsächlich – ggf. Monate nach der ursprünglichen Entwicklung – eine problematische Feature-Kombination anfragt. Das Generieren und isolierte Prüfen aller potentiellen Programme skaliert für Produktlinien nicht.

In dem Vortrag geben wir einen Überblick zu aktuellen Forschungen zum Erkennen von Syntax- und Typ-Fehlern in Produktlinien, und zwar für die gesamte Produktlinie anstatt nur für einzelne Programme:

- Syntax-Fehler können etwa beim Präprozessoreinsatz vermieden werden, wenn man gewisse, leicht überprüfbare Regeln einhält (und dass sogar sprachübergreifend) [12, 13].
- Typ-Fehler können mit einem speziellen Typsystem für Softwareproduktlinien die gesamte Produktlinie statt einzelner Programme prüfen. Dazu haben wir mit FFJ und CFJ zwei auf Featherweight Java basierende Calculi entwickelt (und entsprechende Tools für Java implementiert), die zum Quelltext auch Informationen zu Features der Produktlinie berücksichtigen und somit garantieren können dass aus einer wohl-

getypten Produktlinie nur wohlgetypte Programme generiert werden können [11, 1].

Zu diesen Konzepten stellen wir konkrete Tools und Sprachen zur Produktlinienentwicklung vor, welche diese Prüfungen unterstützen, sowie Erfahrungen aus verschiedenen Fallstudien.

Literatur

1. S. Apel, C. Kästner, and C. Lengauer. Feature Featherweight Java: A Calculus for Feature-Oriented Programming and Stepwise Refinement. In *Proc. Int'l Conf. Generative Programming and Component Engineering (GPCE)*, pages 101–112. 2008.
2. S. Apel, T. Leich, and G. Saake. Aspectual Feature Modules. *IEEE Trans. Softw. Eng.*, 34(2):162–180, 2008.
3. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 1998.
4. D. Batory, J. N. Sarvela, and A. Rauschmayer. Scaling Step-Wise Refinement. *IEEE Trans. Softw. Eng.*, 30(6):355–371, 2004.
5. K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley, 2000.
6. K. Czarnecki and K. Pietroszek. Verifying Feature-Based Model Templates Against Well-Formedness OCL Constraints. In *Proc. Int'l Conf. Generative Programming and Component Engineering (GPCE)*, pages 211–220. 2006.
7. M. Ernst, G. Badros, and D. Notkin. An Empirical Analysis of C Preprocessor Use. *IEEE Trans. Softw. Eng.*, 28(12):1146–1170, 2002.
8. S. Jarzabek, P. Bassett, H. Zhang, and W. Zhang. XVCL: XML-based Variant Configuration Language. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 810–811. 2003.
9. R. E. Johnson and B. Foote. Designing Reusable Classes. *Journal of Object-Oriented Programming*, 1(2):22–35, 1988.
10. K. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, 1990.
11. C. Kästner and S. Apel. Type-checking Software Product Lines – A Formal Approach. In *Proc. Int'l Conf. Automated Software Engineering (ASE)*, pages 258–267. 2008.
12. C. Kästner, S. Apel, and M. Kuhlemann. Granularity in Software Product Lines. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 311–320. 2008.
13. C. Kästner, S. Apel, S. Trujillo, M. Kuhlemann, and D. Batory. Guaranteeing Syntactic Correctness for all Product Line Variants: A Language-Independent Approach. In *Proc. Int'l Conf. Objects, Models, Components, Patterns (TOOLS EUROPE)*. 2009.
14. G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *Proc. Europ. Conf. Object-Oriented Programming (ECOOP)*, pages 220–242. 1997.
15. K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, 2005.
16. C. Prehofer. Feature-Oriented Programming: A Fresh Look at Objects. In *Proc. Europ. Conf. Object-Oriented Programming (ECOOP)*, pages 419–443. 1997.
17. S. Thaker, D. Batory, D. Kitchin, and W. Cook. Safe Composition of Product Lines. In *Proc. Int'l Conf. Generative Programming and Component Engineering (GPCE)*, pages 95–104. 2007.