EXPOSEE

Entwicklung einer Software-Produktlinie mit automatischer Code-Generierung der Varianten

Mistra Forest Kuipou Tchiendja (kuipoutc@th-brandenburg.de)

An der technischen Hochschule Brandenburg werden Bestellsysteme in der Lehre benötigt. Lehrende wollen nicht über die Semester hinweg dieselben Systeme, sondern veränderte Varianten anbieten. Das Ziel der Lehrenden ist einerseits, verschiedene Systeme aus unterschiedlichen Domänen von einem einzigen Grundsystem abzuleiten. Andererseits geht es um das Vermitteln von Konzepten, die sich in Code-Strukturen widerspiegeln und weniger um die Anwendung als solche oder den Domäneninhalt. In diesem Kontext hat Grigarzik (2020) in seiner Abschlussarbeit ein bestehendes Bestellsystem nachimplementiert. Weitere Nachimplementierungen werden benötigt, um möglichst viele Variante zu bilden. Die manuelle Nachimplementierung ist zeitaufwendig und fehleranfällig. Eine mögliche Lösung besteht darin, die Variantenbildung unter einem Dach als Software-Produktlinie (SPL) zu entwickeln und zu verwalten.
Diese Alternative erweist sich als sinnvoll, betonnen Siegmund et al. (2009) und McGregor, Monteith, & Zhang (2011), denn unter einer SPL wird eine Gruppe von Produkten verstanden, die eine gemeinsame wiederverwendbare Grundstruktur aufweisen, welche nur einmal für alle Produktvarianten entwickeln wird. Darüber hinaus werden Unterschiede unter den Varianten innerhalb einer SPL dadurch gekennzeichnet, dass Features separat entwickelt und konfiguriert werden, um spezifische Anforderungen einer bestimmten Domäne gerecht zu werden (vgl. Siegmund et al. 2009, S. 1–7, Stahl et al. 2012, S. 35 und Lopez-Herrejon & Batory, 2002).

Diese Arbeit beschäftigt sich mit der Frage, wie Variantenbildung von Bestellsystemen, unabhängig vom Diskursbereich, systematisch entwickelt werden. Das Hauptziel besteht darin, eine SPL so zu entwickeln, dass der Sourcecode einzelner Variante durch eine einfache Konfiguration automatisch generiert wird. Die zu vermittelnden Konzepten sollten sich in den generierten Codes wieder erkennbar sein.

SPL Engenieuring ist kein einfacher Prozess und kommt in vielen Bereichen der Industrie vor. Industrielle Produkte werden in verschiedenen Varianten serienweise hergestellt. Im Herstellungsprozess wird das Konzept der Wiederverwendbarkeit in jeder Produktlinie umgesetzt. Dubinsky et al. (2013, S. 25-34) haben in ihrer Forschung in sechs großen Unternehmen herausgefunden, dass die Wiederverwendbarkeit systematisch durch Klonen realisiert werden kann. Die Umwandlung geklonter Produktlinien in strukturierten SPL-Modellen spart viel Zeit. Dabei entstehen (Code-)Duplikate, die refaktoriert werden müssen (Dubinsky, et al., 2013, S. 25-34). Die herkömmliche Refaktorierungsmethode muss angepasst werden, um die Variabilität vor und nach der Refaktorierung zu erhalten (vgl. Schulze et al. 2012, S. 73–81). Fenske et al. (2017, S. 316-326) zeigen den Prozess der schrittweisen Umwandlung von geklonten Produkten zu Produktlinien durch sinnvolles Refactoring von Produktvarianten. Die Autoren heben in ihre Ausführung, eine wichtige Eigenschaft einer Refaktorierung hervor: Alle potenziellen (Software-)Produkte innerhalb einer SPL bleiben kompilierbar und behalten ihr bisheriges Verhalten bei. Während Setyautami & Hähnle (2021, S. 1–9) ihren Schwerpunkt darauf legen, Java-Code aus UML-Modellen zu generieren, zeigen Setyautami, Adianto & Azurat (2018, S. 274–278) in ihrer Untersuchung, wie aus UML übesetzte Java-Codes, Produkte als Java-Objekte mittels Gradle Task simuliert werden. Nach Stürmer & Conrad (2004,

S. 33-37) ist die Modellgetriebene Entwicklung eine gängige Praxis im automitiven Sektor. Demnach werden automatische Code effizient aus Software-Modellen generiet.

In dieser Arbeit wird der Schwerpunkt darauf gelegt, eine SPL und einen Generator durch Verwendung von Bibliotheken bzw. geeigneten Werkzeugen zu implementieren. Die von Stahl et al. (2012) aufgeführten Grundprinzipien der modellgetriebenen Softwarentwicklung (MDSD) im Allgemeinen und der SPL im Besonderen werden für den Denkanstoß dienen. Werkzeuge für SPL besitzen bereits einen Generator, durch welchen Codes automatisch generiert werden. Die generieten Code werden nicht immer explizit zur Veranschaulichung gespeichert, sondern nach dem Generiervorgang sofort ausgeführt und lassen sich nur mit sehr hohen Aufwand extrahieren. Es bedarf tiefere Kenntnisse der Architektur der Werkzeugen. Aus diesem Grund wird einen externen Generator angebunden. Dafür wird einen Template-basierte Generator, wie in Kolassa et al. (2016, S. 221-236) beschrieben, zum Einsatz kommen. Bei diesem Prozess sollten folegende Frage beantwortet werden: Was wird generiet? In welche Zielsprache soll generiert werden? Hierfür soll ein Template mit den notwendigen Informationen entwicklet werden, um mit dem Generator zieren zu können.
Die untersuchungen von Horcas, Pinto & Fuentes (2019) werden dabei helfen, die gängigen SPL Werkzeuge gegenüber zu stellen, um eine geeignetes Tool auszuwählen. Darüber hinaus werden die vermittelten Konzepte sowohl von FOSD (Feature-Oriented Software Development, vgl. Kästner & Apel, 2013), als auch von AOP (Aspect-Oriented Programming, vgl. Lopez-Herrejon & Batory, 2002 und Hecht at al. 2007) zur implementierung einer Produktlinie (PL) eingesetzt.
Analoge zu der Modellierung durch UML wobei, ein abstraktes Metamodell mithilfe der AST (Abtract Syntax Tree) generiert wird (vgl. Setyautami & Hähnle 2021, S. 1–9), wird bei der SPL ein Feature-Modell mittels einer Feature Modelling Syntax erstellt. Dabei wird nach der Methode der Bottom-up-Entwicklung vorgegangen. Das heißt das Feature-Modell soll vom bereits bekannten Zielcode entwickelt werden. Auf dieser Weise werden die zu generierenden Codes aus Grigarzik (2000) als Grundlage dienen, das Startmodell zu konstruieren. Methoden der FOP (Feature oriented programming) und Refactoring-Techniken aus Schulze et al. (2012 S. 73–81) werden dafür genutzt. Durch geeignete Mappings wird der Generator eingebunden, welcher für eine Modell-zu-Code-Transformation zuständig ist.

Darüber hinaus wird der Generator in der Lage sein, nicht nur die Anwendung, sondern auch den Code für die grafische User-Interface (GUI) sowie für Tests zu generieren. Die Erweiterung auf GUI und Tests sowie das Anbinden eines externen Generator war in den bereits zitierten Literatur im Zusammenhang mit SPL nicht betrachtet worden. Der Generator soll nicht nur auf Bestellsysteme beschränkt werden. Auch die Möglichkeit die Persistenz-Schicht künftig bei der Generierung anzubinden, soll berücksichtigt werden.

Voraussichtliche Gliederung

Mögliche Titel:

- **Entwicklung von Bestellsystemen als Software Produkt Linie mit Generierung der Productvarianten**
- **Entwicklung einer Software Produkt Linie mit Sourcecode-Generierung der Varianten**
- **Conception and implementation a software product line with automated generation of variants**

1. Einleitung

1.1. Einführung in das Thema
1.2. Aufgabenstellung
1.3. Abgrenzung des Themas
1.4. Ziele und Aufbau der Arbeit

2. Stand der Wissenschaft (Theorie/Grundlagen)

2.1. MDSD (Modellgetriebene Softwareentwicklung)
2.1.1. M-2-M transformation
2.1.2. M-2-C transformation
2.1.3. Top-Down Vs Bottom-Up Entwicklung
2.1.4. Template-basierte Codegenerierung

2.2. SPLE (Software Produktlinie Entwicklung)
2.2.1. Domain engineering
2.2.2. Application engineering
2.2.3. FOSD/ FOP
2.2.4. AOP (AspectJ)

3. Umsetzung/Implementierung (Praxis)

3.1. Architectur und Design
3.2. Bottom-up Entwiclung
3.3. Werkzeuge
3.4. Erstellung des Feature Modells
3.5. Anbindung der Template-basierte Generators

4. Zusammenfassung

Farzit
Diskussion
Kritik
Weiterführende Arbeiten, Ausblick

Literaturverzeichnis

Dubinsky, Y., Rubin, J., Berger, T., Duszynski, S., Becker, M., & Czarnecki, K. (2013). An Exploratory Study of Cloning in Industrial Software Product Lines. *2013 17th European Conference on Software Maintenance and Reengineering*, S. 25-34. doi:10.1109/CSMR.2013.13.

Fenske, W., Meinicke, J., Schulze, S., Schulze, S., & Saake, G. (20-24. Febuar 2017). Variant-Preserving Refactorings for Migrating Cloned Products to a Product Line. *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, S. 316-326. doi:10.1109/SANER.2017.7884632

Grigarzik, P. (7. Dezember 2020). Analyse eines Bestellsystems für zukünftige Variantenbildung in der Lehre. *Unveröffentlichte Bachelorarbeit am Fachbereich Informatik und Medien. Brandenburg an der Havel: Technische Hochschule Brandenburg.*

Hecht, M., Piveta, E., Pimenta, M., & Price, R. (January 2007). Aspect-oriented Code Generation.

Horcas, J.-M., Pinto, M., & Fuentes, L. (2019). Software Product Line Engineering: A Practical Experience. (A. f. Machinery, Hrsg.) *In Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A (SPLC '19)*, S. 164–176. Von https://doi.org/10.1145/3336294.3336304 abgerufen

Kästner, C., & Apel, S. (2013). Feature-Oriented Software Development: A Short Tutorial on Feature-Oriented Programming, Virtual Separation of Concerns, and Variability-Aware Analysis. (R. Lämmel, J. Saraiva, & J. Visser, Hrsg.) *Generative and Transformational Techniques in Software Engineering IV. GTTSE 2011. Lecture Notes in Computer Science, vol 7680*. Von https://doi.org/10.1007/978-3-642-35992-7_10 abgerufen

Kolassa, C., Look, M., Müller, K., Roth, A., Reiß, D., & Rumpe, B. (2016). TUnit – Unit Testing For Template-based Code Generators. (A. Oberweis, & R. Reussner, Hrsg.) *Modellierung 2016, Lecture Notes in Informatics (LNI), Gesellschaft f ̈ur Informatik*, S. 221-236.

Lopez-Herrejon, R. , & Batory, D. (2002). Using AspectJ to Implement Product-Lines: A Case Study. (C. Science, Hrsg.)

McGregor, J., Monteith, J., & Zhang, J. (2011). Quantifying Value in Software Product Line Design. (A. f. Machinery, Hrsg.) *Proceedings of the 15th International Software Product Line Conference, Volume 2 (SPLC '11)*, S. 1-7. doi:https://doi.org/10.1145/2019136.2019182

Schulze, S., Thüm, T., Saake, G., & Kuhlemann, M. (2012). Variant-Preserving Refactorings for Migrating Cloned Products to a Product Line. (A. f. Machinery, Hrsg.) *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS '12)*, S. 73–81. doi:DOI:https://doi.org/10.1145/2110147.2110156

Setyautami, M. R., & Hähnle, R. (February 2021). An Architectural Pattern to Realize Multi Software Product Lines in Java. (A. f. Machinery, Hrsg.) *15th International Working Conference on Variability Modelling of Software-Intensive Systems*(Article 9), S. 1–9. Abgerufen am 3. Mai 2021 von https://doi.org/10.1145/3442391.3442401

Setyautami, M. R., Adianto, D., & Azurat, A. (1. September 2018). Modeling Multi Software Product Lines using UML. (A. f. Machinery, Hrsg.) *Proceedings of the 22nd International Systems and Software Product Line Conference, Volume 1*(SPLC '18), S. 274–278. Abgerufen am 3. Mai 2021 von https://doi.org/10.1145/3233027.3236400

Siegmund, N., Pukall, M., Soffner, M., Köppen, V., & Saake, G. (2009). Using software product lines for runtime interoperability. (N. Y. Association for Computing Machinery, Hrsg.) *Proceedings of the Workshop on AOP and Meta-Data for Software Evolution*, S. Article 4, 1–7. doi: https://doi.org/10.1145/1562860.1562864

Stahl, T. , Völter, M., Efftinge, S., & Haase, A. (2012). *Modellgetriebene Softwareentwicklung : Techniken, Engineering, Management.* dpunkt.verlag.

Stürmer, I., & Conrad, M. (2004). Code Generator Testing in Practice. (P. Dadam, & M. Reichert, Hrsg.) *Informatik verbindet, Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Band 2*, S. 33-37.