Anpassbare Software geschickt umgesetzt?!

Implementierungsmechanismen anpassbarer Software im Vergleich

Martin Becker und Bo Zhang,
Fraunhofer Institut für Experimentelles Software Engineering (IESE)

Bei der erfolgreichen Realisierung von wiederverwendbaren und konfigurierbaren Softwaremodulen ist einer der wichtigsten Schritte die Auswahl von geeigneten Variabilitätsmechanismen. Prominente Vertreter sind hier die bedingte Kompilierung mit dem Präprozessor, die Modulauswahl oder die Parametrierung in Verbindung mit bedingter Ausführung. Der Blick in den Entwicklungsalltag zeigt, dass die Mechanismen oftmals leichtfertig ausgewählt und willkürlich eingesetzt werden. Die Konsequenz sind dann oftmals schlecht verständliche und wartbare Software-Ungetüme. Der Beitrag gibt einen Überblick über gängige Variabilitätsmechanismen, zeigt Stolperfallen und Best-Practices anhand von Praxisbeispielen auf.

Variabilitätstreiber in eingebetteten Systemen

Der Bedarf an kundenspezifischen Software-Lösungen steigt ungebremst. Haupttreiber hinter den angebotenen Produktvarianten sind dabei in erster Linie die unterschiedlichen Kundenbedürfnisse, der technologische Wandel, z.B. neue Mehrkern-Prozessoren, und der Kostendruck. Diese Treiber lassen sich in der Regel nur begrenzt beeinflussen, wodurch die Vermeidung entsprechender Varianten oftmals schwer fällt.

Gerade für den Software-Anteil der eingebetteten Systeme herrscht die Meinung vor, dass dieser schnell und kostengünstig an neue Gegebenheiten angepasst werden kann, was im Vergleich zur Hardware aufgrund der äußerst geringen Produktionskosten zunächst auch richtig ist. Naheliegender weise versucht man die erforderlichen Anpassungen also in die Software zu verlagern. Dies führt dann im Laufe der Zeit recht schnell zur hochgradig anpassbaren, konfigurierbaren und parametrierbaren Softwarelösungen, die immer schwerer zu warten und zu konfigurieren sind. Viele Embedded Software Entwickler haben diese Präprozessor- oder Makefile-Hölle in der einen oder anderen Form schon am eigenen Leib gespürt.

Variabilitätsmechanismen im Vergleich

Die Unterschiede in den Produktvarianten lassen sich auf unterschiedliche Arten realisierten. Man spricht in diesem Zusammenhang von Variabilitätsmechanismen. Prominente Vertreter sind hier die bedingte Kompilierung mit dem Präprozessor, die Modulauswahl oder die Parametrierung in Verbindung mit bedingter Ausführung. Mit den Variabilitätsmechanismen lassen sich also die variablen Produktmerkmale im Quellcode und anderen Entwicklungsartefakten (z.B. Testfälle, Anforderungen, Design-Modelle) umsetzen.

Um eine wohlüberlegte Auswahl der Variabilitätsmechanismen zu fördern, stellen wir in der nachfolgenden Übersicht (siehe Tab. 1) typische Variabilitätsmechanismen vor und

charakterisieren diese entlang zentraler Eigenschaften. Der Vergleich basiert auf entsprechenden Übersichten [1][2][5][6], die in der Software Product Line Engineering Community über die letzten beiden Jahrzehnte entstanden sind.

Gemäß unserer industriellen Erfahrung und jüngsten Studien [3][4][7][8] werden im Kontext von Embedded Software insbesondere folgende Mechanismen zur Realisierung von Software-Varianten eingesetzt:

- Cloning. Softwarestände werden zwischen verschiedenen Projekten kopiert und anschließend modifiziert.
- Bedingte Kompilierung. Mit dem C-Präprozessor werden Code-Blöcke per Schalter ein-/ausgeblendet.
- Bedingte Ausführung. Über Kontrollkonstrukte der Programmiersprachen werden Code-Blöcke über Variablen (Parameter) gesteuert aktiviert/deaktiviert.
- Modulaustausch. Build-Systeme, wie zum Beispiel make, ermöglichen die Auswahl von Modulen, die beim Bauen zum Einsatz kommen.

Weiterhin finden sich noch folgende Mechanismen in der industriellen Praxis:

- Polymorphismus. Sprachkonstrukte wie Funktionszeiger, Überladung oder virtuelle Methoden können verwendet werden, um in der Laufzeit zwischen unterschiedlichen Realisierung auszuwählen.
- Aspekt Orientierung. Über entsprechende Weaver (für C, Java) lassen sich variable Aspekte nachträglich in Komponenten einweben.
- Frame Technology. Unter Verwendung dedizierter Präprozessoren lässt sich die zu bauende Software in einem Vorverarbeitungsschritt gezielt anpassen. Ein prominenter Vertreter ist hier die Frame Technology von Paul Basset.

Bei der Auswahl von Mechanismen sollten folgende generelle Aspekte berücksichtigt werden:

- Bindezeit: Wann wird die Variabilität gebunden, d.h. wann wird die konkrete Variante in den Artefakten gebildet?
- Granularität: Wie fein können die Änderungen sein, die zwischen den Varianten durchzuführen sind, z.B. einzelne Zeilen, Blöcke, Dateien, Module, Subsysteme?
- Explizite Variationspunkte: Lassen sich die Stellen, in denen sich die Varianten unterscheiden, einfach lokalisieren und mit den entsprechenden variablen Merkmalen in Verbindung bringen?
- Varianten-Isolation. Ist die Realisierung einer einzelnen Variante isoliert von den restlichen oder befinden sich alle Varianten innerhalb einer Datei?
- Offene Variation. Lassen sich neue Varianten hinzufügen, ohne dass dies einen Einfluss auf existierende Varianten hat?
- Nicht-Code-Artefakte. Kann der Mechanismus auch jenseits von Code eingesetzt werden?
- Default-Werte. Werden Default-Realisierungen unterstützt?

Die obigen Mechanismen lassen sich bezüglich ihrer Eigenschaften wie folgt charakterisieren.

Mechanismus	Techniken	Bindezeit	Granularität	Explizite Variationpunkte	Varianten Isolation	Offene Variation	Nicht-Code-Artefakte	Default-Werte
Cloning	Kopieren der Artefakte, Branches im Konfigurations management	Build time	alle	Implizit	ja	nein	ja	nein
Bedingte Kompilierung	Präprozessor	Build time	alle	Explizit	nein	nein	ja	ja
Bedingte Ausführung	bedingte Anweisungen	Run time	begrenzt	Implizit	nein	nein	nein	nein
Poly-morphismus	Funktionszeiger, Overloading, usw.	meistens Run time	begrenzt	Implizit	ja	ja	nein	nein
Modul-austausch	Build-System	meistens Run time	begrenzt	Implizit	ja	ja	ja	nein
Aspekt Orientierung	Aspect Weaver	meistens Run time	begrenzt	Implizit	ja	ja	ja	ja
Frame Technology	Frame-Anpassung	Build time	alle	Explizit	ja	ja	ja	ja

Tab. 1: Charakterisierung der Variabilitätsmechanismen

Praktische Erfahrungen mit Variabilitätsmechanismen

In diesem Abschnitt wird die Verwendung der Variabilitätsmechanismen in der Praxis näher beleuchtet sowie deren praktische Vorteile und Herausforderungen diskutiert und entsprechende Empfehlungen gegeben.

Vorteile, Herausforderungen und Empfehlungen sind in der folgenden Tabelle aufgelistet.

Mechanismen	Praktische Vorteile	Praktische Herausforderungen	Empfehlungen
Cloning	Schnelle Einführung mit geringem Aufwand. Unabhängigkeit von anderen Varianten. Gilt für jede Artefakt-Art und -Größe.	Hohe (Langzeit-) Wartungsaufwand .Unvollständige Ausbreitung von Änderungen und Fehlerkorrekturen, was zu niedrigerer Codequalität führt.	Geringer Aufwand von kurzlebigen Klonen. Änderungen sollten klar lokalisiert sein und gemanagt werden. Periodische Klon-Analyse und Refactoring auf andere Mechanismen bei Wartungsproblemen.
Bedingte Kompilierung	Einfach einzuführen. Alle Granularitäten. Explizite Variationspunkte. Keine Laufzeit-Auswirkung (Ressourcen-Bedarf).	#ifdef Code ist schwer zu verstehen wegen #ifdef Verschachtelung, Vermischung, Streuung usw. #ifdef Code neigt zur Erosion und ist oftmals in der Evolution schwer zu warten.	Namensrichtlinien ermöglichen einfache Identifikation und Analyse von Variationspunkten. Analysieren und die unnötige Komplexität des #ifdef Codes aktiv vermeiden / reduzieren. Umbau im Falle von Wartungsproblemen.
Bedingte Ausführung	Einfach einzuführen und zu verstehen. Unterstützt	Begrenzt auf kleine Granularität. Schwer zu unterscheiden	Erwägen Sie, ob die Bindung zur Laufzeit zwingend

	Bindung zur Laufzeit. Hohe Flexibilität.	zwischen der Variationslogik und Code-Funktionalität. Laufzeit-Overhead.	notwendig ist. Ggf. anderen Mechanismus, z.B. Bedingte Kompilierung einsetzen.
Poly-morphismus	Die Trennung von gemeinsamen und varianten Elementen. Hohe Flexibilität. Jede Variante Element kann in Isolation entwickeln.	Geringere Effizienz aufgrund der Fragmentierung der varianten Elemente. Erhöhtes Risiko von Softwarefehlern.	Falls sich Probleme ergeben, könnte Modulaustausch eine Alternative sein.
Modulaustausch	Vorteile von Polymorphismus. Keine Laufzeit-Auswirkung	Variationspunkte und variante Elemente sind schwer zu identifizieren und zu verstehen.	Verwalten von Variationspunkten und varianten Elemente mit dedizierten Modellen mit geeigneter Werkzeugunterstützung, z.B. Generierung der entsprechenden Makefiles.
Aspekt Orientierung	Vorteile von Polymorphismus. Klare Separierung der Änderungen von den Gemeinsamkeiten. Default-Support.	Lernaufwand . Erhöht die Codegröße. Effektiver Code wird erzeugt/modifiziert. Erfordert erneute Validierung. Niedrige Codeverständlichkeit.	Seien Sie vorsichtige bei der Verwendung. Im Falle von negativen Auswirkungen, sollten Sie auf andere Mechanismen wechseln.
Frame Technology	Vorteile von Polymorphismus. Keine begrenzte Granularität. Keine Laufzeit-Auswirkung. Explizite Variationspunkte.	Kaum bekannt. Erfordert spezielle Werkzeuge. Erfordert erneute Validierung.	Seien Sie vorsichtige bei der Verwendung. Im Falle von negativen Auswirkungen, sollten Sie auf andere Mechanismen wechseln.

Tab. 2: Praktische Erfahrungen mit Variabilitätsmechanismen

Generell lässt sich sagen, dass in der Vergangenheit aufgrund der Ressourcenbeschränkungen vermehrt auf Bedingte Kompilierung und Modulaustausch gesetzt wurde. Im zunehmenden Maße wird Variabilität aufgrund der höheren Flexibilität, neuer Geschäftsmodelle und des geringeren Management-Aufwandes in die Laufzeit verlagert. Dabei kommen neben Polymorphismus und Bedingter Ausführung vor allem auch entsprechende Lifecycle-Management-Frameworks wie beispielsweise OSGI zum Einsatz. Egal welche Mechanismen zum Einsatz kommen, achten Sie darauf, dass die Variationspunkte einfach lokalisieren lassen und sich auf die zugrunde liegenden Variabilitäten zurückverfolgen lassen. Damit erleichtern Sie das Varianten-/ Variabilitätsmanagement ungemein und ermöglichen toolgestützte Analysen und Verbesserungen.

Zusammenfassung

Unzulängliche Realisierung von Software-Varianten führt schnell zu unnötiger Komplexität und sollte nach Möglichkeit frühzeitig vermieden werden. In diesem Beitrag haben wir einen Überblick über gängige Variabilitätsmechanismen gegeben und Stolperfallen und Best-Practices aufgezeigt. Der Beitrag ist als Orientierungshilfe gedacht.

Literatur- und Quellenverzeichnis

[1] S. Apel, D. Batory, C. Kästner, G. Saake, “Feature-Oriented Software Product Lines”, Springer, 2013.

[2] F. Bachmann and P. Clements, "Variability in software product lines", Software Engineering Institute, Pittsburgh, USA, Tech. Rep. CMU/SEI-2005-TR-012, Sep. 2005.

[3] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, K. Czarnecki, 0“An Exploratory Study of Cloning in Industrial Software Product Lines”, Proceedings of the 17th European Conference on Software Maintenance and Reengineering, 2013

[4] S. Duszynski, “Analyzing Similarity of Cloned Software Variants using Hierarchical Set Models”, Dissertation, Fraunhofer IESE, Kaiserslautern, Germany, 2015.

[5] C. Gacek and M. Anastasopoulos, "Implementing product line variabilities," SIGSOFT Softw. Eng. Notes, vol. 26, no. 3, pp. 109-117, May 2001.

[6] T. Patzke, "Sustainable Evolution of Product Line Infrastructure Code," Dissertation, Fraunhofer IESE, Kaiserslautern, Germany, 2011.

[7] B. Zhang, M. Becker, T. Patzke, K. Sierszecki, and J. E. Savolainen, "Variability evolution and erosion in industrial product lines: a case study", 17th International Software Product Line Conference (SPLC 2013), pp. 168—177.

[8] B. Zhang, "VITAL - reengineering variability specifications and realizations in software product lines", Dissertation, Fraunhofer IESE, Kaiserslautern, Germany, 2015.

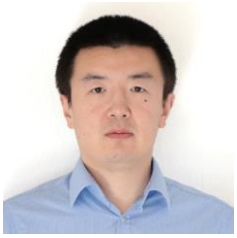
Autoren

Dr. Martin Becker ist Informatiker und leitet die Abteilung Embedded Systems Engineering am Fraunhofer IESE. In einer Vielzahl von Forschungs- und Industrieprojekten hat er in den letzten 15 Jahren vielfältige Erfahrungen mit Varianten-Management in verschiedenen Anwendungsgebieten gesammelt, sowie neue Methoden, Techniken und Werkzeuge auf diesem Gebiet entwickelt.

Kontakt

Internet: www.iese.fraunhofer.de/
Email: Martin.Becker@iese.fraunhofer.de

Bo Zhang ist ein wissenschaftlicher Mitarbeiter am Fraunhofer Institut für Experimentelles Software Engineering (IESE) in Kaiserslautern. Er arbeitet in der Embedded Systemen Entwicklung Abteilung und seiner Forschung konzentriert sich auf die modellbasierte Systemarchitektur Engineering und Variationsmangement. Er hat sich in vielen industriellen und akademischen Projekten gearbeitet. Er hat an der Technischen Universität Kaiserslautern im Jahr 2015 über Analyse und Verbesserung von Variabilität promoviert.



Kontakt
Internet: www.iese.fraunhofer.de/
Email: Bo.Zhang@iese.fraunhofer.de