

Konfiguration von Produktlinien

Haiko Strotbek
Universität Stuttgart
Institut für Softwaretechnologie
Universitätsstraße 38
D-70569 Stuttgart
deppdepp@gmx.net

Zusammenfassung

In dieser Ausarbeitung wird ein Einblick in die Anforderungen und Probleme gegeben, die entstehen, um Produktlinien flexibel zu halten. Es wird auch näher betrachtet, welche unterschiedlichen Typen von Variationen es gibt und wie man sie realisieren könnte. Außerdem wird auf die höheren Anforderungen an das Configuration Management in einer Produktlinie und die Risiken, die ein schlechter Configuration Management Prozess birgt, eingegangen und wie man ihnen begegnen kann.

1. Einleitung

Flexibilität ist eine der Hauptanforderungen an Produktlinien, denn der Grundgedanke einer Produktlinie ist, möglichst viele Software-Artefakte wiederverwenden zu können, um so einen möglichst effizienten Entwicklungsprozess zu haben.

Um jedoch den verschiedenen Anforderungen durch verschiedene Produkte für verschiedene Kunden gerecht zu werden, müssen Grundlagen zur Schaffung und Verwaltung einer flexiblen Architektur geschaffen werden.

Um diese Flexibilität zu erreichen, müssen Variationspunkte und Konfigurationsmöglichkeiten in der Software realisiert werden.

Diese Variationspunkte sind im Grunde nichts weiter als offen gelassene Entwurfsentscheidungen: Bei der Entwicklung werden diverse Punkte noch nicht fest entschieden, um dort später festlegen zu können, wie und womit solch eine Lücke gefüllt werden kann.

2. Variationspunkte und Varianten

Dabei lassen sich drei Stufen der Variationsmöglichkeiten unterscheiden: implizite, entworfene und gebundene. Wird eine Variationsmöglichkeit neu eingeführt, so wird sie implizit genannt. Erst, wenn die

Realisierung festgelegt ist, ist sie entworfen worden. Wenn man für ein Produkt schließlich die passende Variante wählt, so ist sie gebunden. Dies kann wiederum zu verschiedenen Zeiten in der Entwicklung passieren, z. B. im Entwurf festgehalten werden oder beim Kompilieren fest eingebaut oder während der Laufzeit umkonfiguriert werden.

Wichtig ist hierbei noch, dass Variationspunkte früh eingebaut werden, denn je früher ein Variationspunkt eingeführt wird, desto größer ist die resultierende Vielfalt an Möglichkeiten.

Betrachtet man diese Variationsmöglichkeiten genauer, stellt man fest, dass diese im eigentlichen Sinne nichts anderes als Merkmale und Funktionalitäten der späteren Produkte sind. Um nun überhaupt einen Überblick über die Produktlinie zu haben, ist es unerlässlich, diese Merkmale zu dokumentieren. Unterschieden werden dabei drei verschiedene Typen von Merkmalen: Die obligatorischen oder verbindlichen Merkmale repräsentieren die Grundfunktionalität, welche alle späteren Produkte aufweisen. Optionale Merkmale hingegen treten nur in bestimmten, z. B. den teureren Produkten auf. Eine Variante wiederum repräsentiert Alternativen für dieselbe Funktionalität, von denen eine oder mehrere in ein Produkt integriert werden können.

Häufigste und bisher geeignetste Lösung zur Darstellung ist ein sogenannter Feature-Graph. Leider gibt es noch kein Tool, das explizit zur Erstellung solcher Graphen gedacht ist, jedoch lassen sich gut UML-Tools dazu verwenden. Die Aufgabe eines Feature-Graphen ist, zu zeigen, welche Merkmale und Funktionalitäten verfügbar sind und in welcher Abhängigkeit sie zueinander stehen, denn leider sind die meisten Funktionen nicht unabhängig, sondern stehen in einer direkten Beziehung zueinander. Auch hier gibt es verschiedene Sorten zu unterscheiden:

Während es bei einfachen oder exklusiven Varianten mehrere Möglichkeiten gibt, die sich gegenseitig ausschließen, z. B. die Systemumgebung, in der das Programm später läuft, gibt es auch mehrfache Varianten, bei denen man mehrere Möglichkeiten auswählen

kann. Ein Beispiel hierfür wären die unterstützten Protokolle bei einem Filesharing-Programm. Darüberhinaus gibt es noch optionale Varianten, bei denen man die Wahl hat, sie einzubauen oder nicht, wobei dies natürlich auch in Kombination mit einer Einfach- oder Mehrfachvariante auftreten kann.

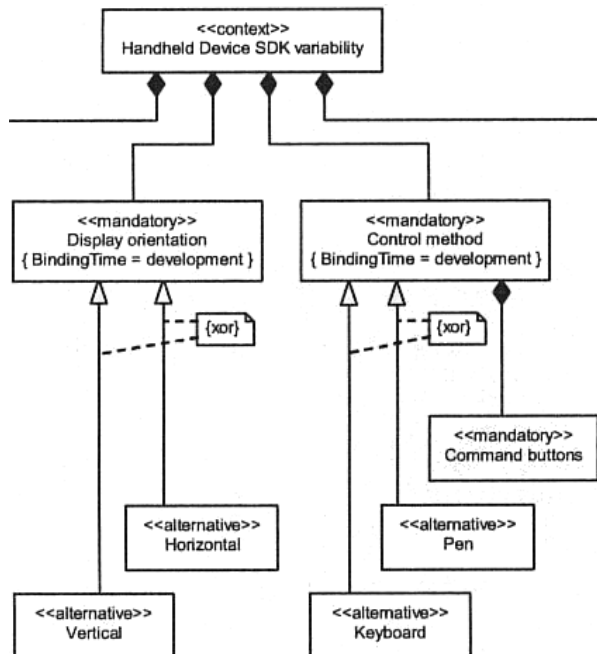


Abbildung 1: Ausschnitt eines Feature-Graphen

Leider ergibt sich aber aus dem Streben nach Variationsmöglichkeiten auch ein Zielkonflikt: Flexibilität kostet Geld! Werden viele Variationsmöglichkeiten in eine Produktlinie eingebaut, so ist der Aufwand der Entwicklung höher. Zum einen wegen der Quantität, da so auch mehr Varianten entwickelt werden müssen, aber auch qualitativ wird das Entwickeln schwerer, da Variationsmöglichkeiten immer Wechselwirkungen untereinander haben. Deshalb sind mehr Eventualitäten zu berücksichtigen, was mitunter schwierig werden kann.

Auf der anderen Seite muss eine Produktlinie, wie schon gesagt, eine gewisse Flexibilität bieten, um einen gewissen Kundenkreis anzusprechen.

3. Technische Realisierungen

Es gibt viele Möglichkeiten, um Variationsmöglichkeiten zu verwirklichen. Je nach Anwendungsgebiet und Programmiersprache ist eine Möglichkeit besser oder schlechter geeignet, falls die Sprache diese Technik überhaupt unterstützt. Hier ist eine Auswahl an Möglichkeiten, Variationspunkte in Software einzubauen:

Die Aggregation ist eine Technik objekt-orientierter Sprachen, die das Weiterleiten von Funktionsaufrufen an andere Objekte unterstützt. Durch Implementierung variabler Funktionalität in getrennten Klassen, deren Funktionen durch Weiterleitung der Aufrufe benutzt werden, können Variationsmöglichkeiten geschaffen werden.

Auch die Vererbung ist eine Technik der objekt-orientierten Sprachen. Um mit Hilfe dieser Technik Variationsmöglichkeiten zu realisieren, kann man gemeinsam genutzte Funktionalität in Oberklassen und variable Zusatzfunktionen in Unterklassen implementieren.

Parametrisierung bedeutet, dass Klassen oder Funktionen eine Funktionalität für verschiedene Datentypen bereithalten und dynamisch angepasst werden. Beispiele für diese Technik sind die Templates in C++ oder der generische Typ in Ada.

Das Überladen ist eine der ältesten und bekanntesten Techniken: Durch Wiederverwendung bereits existierender Namen kann man die Funktionalität von Prozeduren, Funktionen und Operatoren überschreiben.

Makros wiederum lassen sich auf zwei verschiedene Arten verwenden: Einerseits kann man gemeinsame Funktionalität als Makro definieren. Durch den Aufruf des Makros an den entsprechenden Stellen und Übergabe von Parametern lassen sich so Variationen erreichen. Andererseits kann man Makroaufrufe als Teil des Codes implementieren. Durch verschiedene Makrodefinitionen können verschiedene Programme realisiert werden.

Bedingte Kompilierung wird oft mit den Präprozessor-Anweisungen unter C/C++ verwirklicht. Durch Setzen verschiedener Werte kann Code in die Kompilierung ein- oder von der Kompilierung ausgeschlossen werden.

Bei der Konfigurations-Technik werden die einzelnen Varianten in jeweils verschiedene und getrennte Dateien geschrieben. Mit Tools des Configuration Management wird dann zwischen den Dateien gewählt.

Statische Bibliotheken stellen Funktionen für das Hauptprogramm bereit. Sie können vor dem Linken ausgetauscht werden und halten so Variationsmöglichkeiten bereit.

Dynamisch gelinkte Bibliotheken können noch bei der Laufzeit eines Programms getauscht werden und bieten so eine bessere Grundlage für Variationen. Jedoch muss das Betriebssystem diese Bibliotheken unterstützen, was vor allem im Embedded-Bereich nicht immer der Fall ist.

Die Java Virtual Machine beherrscht das dynamische Laden von Klassen. Dieses Verhalten kann im Quellcode erweitert und kontrolliert werden, um zur Laufzeit zu entscheiden, welche Klassen geladen werden.

Diese Liste ist jedoch keinesfalls vollständig. Es gibt auch kein einheitliches Konzept, Software variabel zu

halten, nur mehr oder weniger bewährte Techniken. Systematische Beschreibungen oder Standards, die festlegen, wie man bestimmte Arten von Flexibilität durch Verwendung einer Kombination verschiedener Techniken erreicht, wären sinnvoll und hilfreich. Auch eine bessere Unterstützung durch die Programmiersprachen oder Tools, um so etwas zu automatisieren, würden sich rasch bezahlt machen.

4. Das Configuration Management

Schwierigkeiten, die bereits bei einem herkömmlichen Software-Projekt im Configuration Management (CM) entstehen, sind bei einer Produktlinie um ein Vielfaches größer.

Bisher mußte das CM bereits die vielen und häufigen Änderungen in der Software verwalten, ohne dass dabei ein Chaos entsteht. Außerdem war es wichtig zu überwachen, welche Fehler in welcher Version von Software vorhanden oder behoben sind.

In einer Produktlinie kommt noch erschwerend hinzu, dass es viele Varianten von Dateien, Komponenten und Produkten gibt, die jeweils ihre eigene Konfiguration und Dokumentation erfordern. Gerade auch das Verwalten der verschiedenen Variationsmöglichkeiten ist ein aufwendiger Prozess. Man spricht hierbei öfters von einem „Variation Management“.

Darüberhinaus werden die Core Assets, und damit auch indirekt alle Produkte, nur von einem einzigen CM-Prozess verwaltet, während beim herkömmlichen Entwicklungsprozess noch jedes Produkt sein eigenes CM besitzt. Ebenso werden beim bisherigen Prozess die Komponenten der Software auch gleichzeitig von den Produktentwicklern geschrieben und benutzt, bei Produktlinien jedoch benutzen die Komponenten andere Entwickler, als die, die sie schreiben.

Das vorrangige Ziel des CM bei Produktlinien ist dabei, eine schnelle Reproduzierbarkeit aller Produkte zu ermöglichen.

4.1. Risiken

Gerade durch die erhöhten Anforderungen entstehen auch Risiken, die das CM bei Produktlinien betreffen.

CM-Prozess ist nicht robust genug:

Die Anforderungen an das CM in einer Produktlinie sind weitaus höher und dementsprechend ist der Prozess an sich auch komplexer.

Erweist sich der Prozess als nicht robust genug, wird er in irgendeiner Weise scheitern und ineffizient arbeiten.

CM erscheint zu spät:

Hat ein Unternehmen kein CM bevor das erste Produkt ausgeliefert wird, werden neue Produktversionen oder

die Reproduktion alter Produktversionen zeitraubend und teuer.

„Mutierende“ Assets:

Es besteht die Möglichkeit, dass Assets sich in verschiedene Richtungen entwickeln. Das kann einerseits passieren, wenn ein Asset für viele verschiedene Umgebungen angepasst wird oder wenn sich ein Asset in einem spezifischen Produkt weiterentwickelt.

Falls der zweite Fall eintritt und nicht anderweitig verhindert werden kann, wird die Funktionalität der Assets hinsichtlich der universellen Einsetzbarkeit vermindert.

Inkonsequente CM-Durchführung:

Wird das CM nicht konsequent durchgesetzt, kann das in einem kompletten Chaos enden.

Schlechte Tool-Unterstützung:

Tools, die zur Verwaltung von Produktlinien gedacht sind, sind noch nicht am Markt vorhanden. Deshalb werden zur Zeit mehrere Tools für diverse Teilprobleme benutzt oder gar zweckentfremdet.

Das Finden und Einrichten solcher Werkzeuge ist jedoch eine schwierige Aufgabe, die kompetentes Personal erfordert.

4.2. Multi-dimensionale Anforderungen

Die eigentliche Herausforderung des CM besteht darin, dass Änderungen in Dateien, Komponenten und Produkten verwaltet werden müssen – und das, während sie weiterentwickelt oder neue Varianten entwickelt werden über die gesamte Produktlinie bzw. deren Produkte hinweg.

Man spricht, wegen verschiedener Größenordnungen (Datei, Komponente, Produkt) und verschiedenen Kategorien (sequentiell, parallel, an verschiedenen Stellen), von einem multi-dimensionalen CM-Problem.

4.3. Der Divide & Conquer – Ansatz

Wie schon erwähnt, lassen sich die verschiedenen Schwierigkeiten unterteilen: Zum einen in verschiedene Ebenen: die Datei-Ebene, die Komponenten-Ebene und die Produkt-Ebene, und zum anderen in verschiedene Kategorien: Änderungen im Laufe der Zeit, parallele Entwicklung von Varianten und Änderungen an verschiedenen Orten.

Die daraus resultierenden neun Teilprobleme lassen sich wie folgt benennen:

1. Version Management: Versionen von Dateien, die sich im Laufe der Zeit ändern
2. Branch Management: Unabhängige Varianten der Dateien

3. Baseline Management: Ansammlung von Dateien in verschiedenen Versionen
4. Branched Baseline Management: Unabhängige Varianten von Komponenten
5. Composition Management: Ansammlung von Komponenten in verschiedenen Versionen
6. Branched Composition Management: Unabhängige Varianten von Produkten
7. Variation Point Management: Die verschiedenen Varianten von Dateien in der Produktlinie
8. Customization Management: Konsistente Zusammenstellungen der Dateien
9. Customization Composition Management: Zusammenstellung der konfigurierten Komponenten

	Sequential	Parallel	Domain space
Files	Version management	Branch management	Variation point management
Components	Baseline management	Branched Baseline management	Customization management
Products	Composition management	Branched Composition management	Customization Composition management

Abbildung 2: Die neun Teilprobleme des CM

Darüberhinaus lassen sie sich zu drei Kategorien klassifizieren:

Einerseits sind die ersten vier genannten Probleme Teil des klassischen CM, während Composition Management und Branched Composition Management zur Komponentenkomposition gehören.

Die übrigen drei sind Probleme der Massenanfertigung.

Für jedes Teilproblem gilt es nun, eine befriedigende Lösung zu finden.

4.4. Aufgaben des klassischen CM

Probleme dieser Kategorie sind Probleme des herkömmlichen CM und lassen sich daher auch gut mit den altbekannten Mitteln handhaben.

So ist auf Dateiebene eine Versionierung unentbehrlich. Änderungen müssen dokumentiert werden und die Dokumentation abrufbar sein, um eventuelle Pro-

bleme, Fehler oder deren Ursache zu erkennen. Durch Versionierung lässt sich so oft gut zurückverfolgen, wann ein Fehler behoben wurde oder ein neuer hinzugekommen ist.

Um die parallele Entwicklung von Varianten verwalten zu können, ist auch eine Unterstützung von Verzweigungen notwendig. Hat man nun eine neue Variante einer Datei geschrieben, z. B. indem man sie an eine andere Umgebung angepasst hat, bekommt sie keine neue Version, sondern einen eigenen Versionszweig, der dann weiterentwickelt werden kann.

Darüberhinaus könnte sich die „joining“-Fähigkeit als nützlich erweisen, da es mit ihr leichter möglich ist, Änderungen in verschiedenen Versionen und Versionszweigen in neue Versionen und Varianten zu übertragen.

Auch bei Komponenten, die nur ein Zusammenschluss mehrerer Dateien sind, sind Versionierungs- und Verzweigungsmöglichkeiten notwendig, um Änderungen festzuhalten und nachvollziehen zu können. Da Komponenten jedoch nur aus Dateien bestehen und nicht Dateien repräsentieren, muss für eine Komponente ein Konfigurations-Identifikationsdokument angelegt werden. In dieser wird aufgelistet, mit welchen Dateien in welchen Versionen und Varianten die Komponente arbeitet. Anhand dieser Konfigurationsdokumente kann dann eine Komponente weiterentwickelt werden, indem neue Versionen bzw. Varianten von Dateien die alten ersetzen.

4.5. Komponentenkomposition

Probleme bei der Komponentenkomposition oder Produktgenerierung entstehen durch die Vielzahl von Variationsmöglichkeiten, die es in einer Produktlinie gibt. Denn oft setzen bestimmte optionale Funktionalitäten andere Funktionen voraus oder bestimmte Features arbeiten nicht mit anderen zusammen.

Dieser Problematik versucht man oft mit Hilfe des bereits angesprochenen „Feature-Graphen“ beizukommen. Doch auch diese Graphen werden bei großen Systemen unüberschaubar oder können nicht aufgestellt werden, da man an einigen Stellen schlecht oder gar nicht die Abhängigkeiten zwischen den Variationspunkten dokumentiert hat.

Um Produkte letztendlich zusammensetzen, haben sich zwei verschiedene Ansätze hervor getan:

1. „shelf“: Hier wird von jeder benötigten Komponente eine Version ausgewählt und so Stück für Stück ein Produkt zusammengesetzt. Dabei sind die angesprochenen Abhängigkeiten und Unverträglichkeiten zu beachten.
2. „context“: Beim context-Ansatz wird eine funktionierende Kombination von Komponenten ständig weiterentwickelt. Entsteht eine neue Version einer

Komponente, wird sie geprüft und gegebenenfalls in den context mit aufgenommen.

Bezogen auf das Composition Management ist der context-Ansatz sehr empfehlenswert, da sonst für jede neue Produktversion ein großer Aufwand bei der Neugenerierung anfallen würde.

Denn beim context-Ansatz verteilt sich der Aufwand über einen längeren Zeitraum und ist, da nur relativ kurze Validierungstests anfallen, überschaubar.

Für das Branched Composition Management bedeutet der context-Ansatz, dass für die context-Datei mehrere Varianten entstehen.

4.6. Software Massenanfertigung

Um die Reproduzierbarkeit von Produkten aus der Produktlinie zu gewährleisten, ist es wichtig, auch die Anfertigung unter das CM zu stellen.

So muss in der Software Massenanfertigung auf jeden Fall sichergestellt werden, dass die nötigen Variationsmöglichkeiten zur Verfügung stehen, oder es muss bei Bedarf nachgebessert werden.

Aus mehreren Dateien werden dann logische Einheiten, also konfigurierte Komponenten gebildet, aus denen schließlich Produkte generiert werden können.

Jedoch sollte es nach Möglichkeit vermieden werden, die entsprechenden Variationen durch produktspezifischen Code zu realisieren. Dieser wächst erfahrungsgemäß sehr schnell und verbraucht bei der Entwicklung zusätzliche Ressourcen. Außerdem kann solch ein Code, da er nur für ein bestimmtes Produkt gedacht ist, nicht wiederverwendet werden. Trotzdem ist er in den meisten Fällen notwendig, er sollte sich dann jedoch auf sogenannten „glue code“ beschränken, der das Produkt zusammenhält und lauffähig macht.

Darüberhinaus hat es sich als vorteilhaft erwiesen, wenn man die Produkte sozusagen an die kurze Leine nimmt.

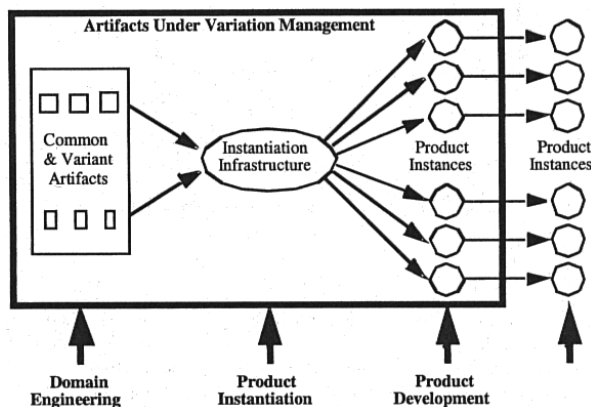


Abbildung 3: Produkte im CM

Lässt man sie sich unabhängig weiterentwickeln, so schafft gerade dies die Basis für produktspezifischen Code. Andere ungewünschte Nebenwirkungen sind dabei, dass weniger Rückmeldungen über nützliche Weiterentwicklungen an das Core-Asset-Entwicklerteam stattfinden. Im Umkehrschluss heißt das wiederum, dass Änderungen, die in den Core-Assets gemacht worden sind, nicht direkt auf Produkte übertragbar sind.

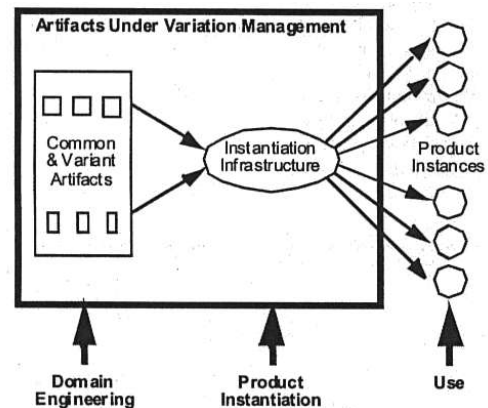


Abbildung 4: Empfohlen: Produkte außerhalb des CM

Deshalb sollten die Produktlinienarchitekten darauf achten, dass Produkte möglichst direkt aus der Produktlinie heraus generiert werden können.

5. Schlussfolgerung

Wie wir gesehen haben, ist die Technologie und der Wissensstand bei Produktlinien noch stark am Anfang der Entwicklung. Eine Produktlinie kann viel Geld einsparen und dabei bessere Produkte abwerfen als der bisherige Softwareentwicklungsprozess, doch werden andererseits Umschulungen und aufwendigere Prozesse notwendig.

Gerade die Tool-Unterstützung steckt noch vollkommen in den Kinderschuhen: Man bräuchte Tools zur Darstellung von Feature-Graphen oder Produktlinienarchitekturen, Tools zur Unterstützung der Produktgenerierung, Verwaltung und Dokumentation von Konfigurationen von Komponenten wie Produkten, sogar Tools zur Schaffung von Variationsmöglichkeiten in der Software wären denkbar.

Es gibt zwar bereits einige Ansätze, von der effizienten Anwendung solcher Software ist man jedoch noch weit entfernt. Aber gerade mit dem Erscheinen entsprechender Software könnte man die Idee der Produktlinien stärker ins Gespräch bringen und so die Verbreitung dieser vielversprechenden Technologie unterstützen.

6. Literatur

1. Clements, P., Northrop, L.: Software Product Lines – Practices and Patterns. SEI Series in Software Engineering. Addison-Wesley (2001) 152-160
2. Myllymäki, T.: Variability Management in Software Product Lines. Institute of Software Systems, Tampere University of Technology (2002)
<http://practise.cs.tut.fi/pub/papers/VarMgnFinal.pdf>
3. Krueger, C.W.: Variation Management for Software Product Lines. In Chastek, G.J., ed.: Proceedings of the 2nd Software Product Line Conference, San Diego, CA, USA, Springer (2002) 37-48
4. Jaring, M., Bosch, J.: Representing Variability in Software Product Lines: A Case Study. In Chastek, G.J., ed.: Proceedings of the 2nd Software Product Line Conference, San Diego, CA, USA, Springer (2002) 15-36
5. Van Gorp, J., Bosch, J., Svahnberg, M.: On the Notion of Variability in Software Product Lines. (2001)
(leider nicht mehr auffindbar)
6. Balzert, H.: Lehrbuch der Softwaretechnik. Spektrum (2000) 234-248