

Requirements Engineering

Michael Grosse
Universität Stuttgart
grosseml@informatik.uni-stuttgart.de

Zusammenfassung

Dieser Artikel beschäftigt sich mit verschiedenen Möglichkeiten, Requirements Engineering in Produktlinien zu ermöglichen. Er stellt sowohl das Requirements Engineering im Allgemeinen vor, geht danach auf die spezifischen Anforderungen für Produktlinien ein und stellt abschließend in kurzen Worten einen Überblick über vier verschiedene Techniken vor. Eine genaue Beschreibung der vorgestellten Techniken ist in den Quellen zu finden.

1. Einführung

Die Entwicklung ähnlicher Produkte als Produktlinie – oder Produktfamilie – bietet gegenüber der relativ teuren Einzelsystementwicklung viele Vorteile, die überwiegend darauf beruhen, dass alle Familienmitglieder auf einer gemeinsamen Infrastruktur – auch Plattform oder Architektur – aufbauen. Während in anderen Industriebranchen, wie dem Automobilbau oder der Schokoladenindustrie, die Vorteile der Produktlinienentwicklung schon lange systematisch genutzt werden, sind die meisten heutigen Softwaresystementwicklungen immer noch extrem teure Einzelstücke.

Dabei kann auch die Softwareentwicklung vom Prinzip der Produktlinienentwicklung profitieren. Ein Beispiel hierfür wäre die Zeit- und Kostenersparnis bei der Entwicklung ähnlicher Produkte oder durch erhöhte Produktqualität durch einen möglichst hohen Wiederverwendungsanteil existierender und damit erprobter und getesteter erfolgreicher Komponenten.

Auch die Anpassung von eigenen Standardprodukten an besondere Kundenwünsche wird durch vorab sorgfältig geplante Variabilität erleichtert. Produktlinien decken hier durch den gesamten Lebenszyklus der Software ab und integrieren hier durch viele Themenbereiche des herkömmlichen Software Engineerings wie Requirements Engineering, Softwarearchitektur und Reengineering.

Das Requirements Engineering spielt eine wichtige Rolle im Lebenszyklus von Software. Modellbasiertes Requirements Engineering könnte als Grundlage eines Produktfamilien Engineerings dienen.

2. Unterschiede Einzelstück vs Produktlinie

2.1. Das Einzelstück

Die verschiedenen Möglichkeiten und Probleme der Anforderungsanalyse für Softwareeinzelstücke ist zwar noch weit davon entfernt, hinreichend erforscht zu sein, jedoch herrscht hier weitestgehend Einigkeit über das prinzipielle Vorgehen bei der Analyse. Zuerst werden hier alle Anforderungen an das zu erstellende Produkt definiert. Hierfür werden die gängigen Methoden der UML, spezifisch Use Cases, häufig verwendet. Bei der Erfassung der gewünschten Anforderungen wird dabei natürlicherweise sehr genau auf die Kundenwünsche eingegangen und es wird notfalls bis an die Grenzen der Machbarkeit gegangen, um spezielle Kundenwünsche auf dessen System in seiner Umgebung nicht nur lauffähig zu bekommen, sondern auch gleichzeitig diese zu optimieren.

Eine weitere gern genutzte Möglichkeit das Einzelstück möglichst optimal an den Kunden anzupassen, ist die Definition der Schnittstellen an das spezielle System des Kunden. Hier werden unter Umständen Schnittstellen benutzt, die die vorhandene Hardware voll ausreizen oder Schnittstellen zu anderen Einzelstücken der vorhandenen Software ausnutzen.

Da die Anforderungen an jedes einzelne Projekt sich sehr stark unterscheiden können die allerwenigsten Anforderungen über verschiedene Projekte weiterverwendet werden. Dies macht jede einzelne Analysephase wieder zu einem teuren Einzelstück, das nach dem Projekt im Aktenschrank für immer abgelegt wird.

2.2. Die Produktlinie

Aufgrund der in den Einzelprodukten gesammelten Erfahrung und vor allem natürlich der hohen Kosten und des immens hohen Zeitaufwands war relativ schnell klar, dass bei der Planung einer Produktlinie die Anforderungsanalyse der einzelnen Produkte viel kleiner als bisher ausfallen muss, da in der Anforderungsanalyse der Produktlinie schon möglichst alle gemeinsamen Fakten und Zusammenhänge erkannt und analysiert sein müssen.

Die Hauptaufgabe der Anforderungsanalyse liegt dadurch im Sammeln der Gemeinsamkeiten. Unter der Voraussetzung, dass man aus den bisherigen Einzelprodukten eine Produktlinie bilden will, müssen zuerst sämtliche zum damaligen Zeitpunkt gesammelten Anforderungen zusammengetragen werden. Hierbei ist es erstmal unerheblich ob die Anforderungen redundant oder widersprüchlich sind, was in diesem ersten Schritt zählt ist die Informationsbasis, auf der man die später einheitlichen und gemeinsamen Anforderungen auswählt.

Im zweiten Schritt werden nun die gemeinsamen Anforderungen zusammengefasst und auf eine möglichst breite Basis gestellt, so dass sie für alle zu diesem Zeitpunkt denkbaren Anforderungen zutreffen können. Da dieser Schritt für die Kostenersparnis der Produktlinien eine wesentliche Rolle spielt, kommt ihm eine zentrale Bedeutung zu und er sollte daher nicht unterschätzt werden.

Der dritte Schritt der Anforderungsanalyse bezieht sich auf das Klären der Widersprüche der einzelnen Programme. Dieser Schritt ist besonders heikel, da Widersprüche, die in einer Produktlinienanalyse auftauchen, automatisch immense Kosten hervorrufen, wenn sie erst nach der Realisierung einiger Produkte herausgefunden werden und die Produkte unter Umständen einzeln kostspielig nachbearbeitet werden müssen.

Der vierte Schritt ist der bisher am wenigsten erforschte. Er beinhaltet die sinnvolle Darstellung der gemeinsamen Anforderungen. Klar ist bisher nur, dass es derzeit keine bekannte Lösung gibt, die die komplexen Anforderungen einer Produktlinie einheitlich und übersichtlich, vor allem aber überschaubar darstellen kann. Aus diesem Grund werden verschiedene angedachte Möglichkeiten im folgenden Kapitel vorgestellt und kurz erläutert.

3. Zweigeteilte Analyse

Da es naturgegebenmaßen unmöglich ist, sowohl eine allgemeingültige Abstraktion für viele Produkte

zu erreichen, gleichzeitig aber für eine konkrete Situation passende Anforderungen zu erstellen, muss der Analyseprozess aufgeteilt werden in die Domain Analyse und die Anwendungsanalyse. Auf diese beiden Analysearten soll in diesem Kapitel weiter eingegangen werden.

3.1. Domain Analyse

Zuerst eine kurze Definition des Begriffs Domain Analyse:

„Domain Analyse (DA) ist der Prozess der Erkennung, des Erwerbs und der Evolution von wieder- verwendbarer Information über einen Anwendungsbereich (,domain')“.[1]

Die Domain Analyse ist daher die für Produktlinien die wichtige Analyse um die gemeinsamen Anforderungen zu erkennen. Der Prozess zur Erstellung einer Domain Analyse ist in vielen Punkten eine Zusammenfassung der im letzten Kapitel genannten Schritte. Hier nun die Beschreibung der einzelnen Schritte:

Der erste Schritt der Domain Analyse besteht in der Identifikation von Kategorien und Problembereichen. Hier ist eine besondere Sorgsamkeit von Nöten um später nicht doppelte oder dreifache Arbeit durch Nachbesserungen oder redundante Anforderungen zu haben.

Der zweite Schritt besteht im Herausfinden ähnlicher Applikationen. Hierbei wird versucht, so weit wie möglich ganze Programmpakete zu erreichen, die möglichst vielen späteren Produkten gemein sein sollen.

Im dritten Schritt geht es um die Identifikation, sowie die Formalisierung der Konzepte, die den verschiedenen Applikationen gemein sind. Damit ist gemeint, dass versucht wird, durch systematisches Durchforsten der in den vorigen Schritten gesammelten Anforderungen und Applikationsteilen gemeinsame Konzepte zu finden und diese soweit möglich als Formale Spezifikation zu erfassen. Je formaler und damit eindeutiger diese gemeinsamen Anforderungen und Konzepte erfasst werden, desto mehr nutzt dieses Wissen später der gesamten Produktlinie.

Der fünfte und letzte Schritt ist das Organisieren der gemeinsamen Konzepte in Bibliotheken mit wieder verwendbaren Bausteinen. Ein weiterer wichtiger Faktor bei diesem Schritt ist die Organisation einer leichten Zugriffsmöglichkeit um sowohl das Auffinden der fertigen Teilkomponenten zu erhöhen, als auch den Zugriff zu regeln und zu erleichtern. Es muss klar gestellt werden, wer Änderungen unter welchen Umständen an den allgemeinen Komponenten vornehmen darf und wer nicht. Wird dies nicht gewährleistet droht eine

ständige Änderung und ein schwerer Schaden in der Stabilität der Produktlinie.

3.2. Anwendungsanalyse

Die Anwendungsanalyse ist für das konkrete zu realisierende Produkt wichtig. In ihr wird auf die Anforderungen eingegangen, die sich durch die Systemumgebung und spezielle Kundenwünsche ergeben. Die Anwendungsanalyse besteht aus einer Extraktion der Anforderungen aus der Produktlinie und eines Auslotens der Anforderungen des spezifischen Kunden.

Hierbei ist aus der Sicht des Produktdesigners darauf zu achten, welche Komponenten nun tatsächlich aus den Definitionen der Produktlinien entnommen werden können. Da nicht immer die Standardlösung die beste ist, muss von Fall zu Fall entschieden werden, ob diese oder jene gemeinsame Komponenten nun tatsächlich genutzt werden sollen. Der Vorteil der sich aus der schon bekannten Spezifikation der Produktlinie ergeben besteht darin, dass die Anforderungen nicht nur klar und so gut wie möglich spezifiziert sind, sondern auch schon umgesetzt sind, wodurch die Zeit und dadurch die Kosten des Projektes erheblich verkleinert werden.

Dadurch müssen die schon fertigen Teile nur noch minimal an das Kundensystem angepasst werden, sowie die vom Kunden hinzugefügten Wünsche und Features implementiert werden.

4. Verschiedene Ansätze

Ein bisher ungelöstes Problem des Requirements Engineering für Produktlinien ist, dass es bisher keine Standardisierung der Beschreibungssprache gibt. Herkömmliche Modelle wie FODA oder UML bieten jeweils nicht genügend Darstellungs- / Einstellungsmöglichkeiten, alle Aspekte und Problembereiche der für Produktlinien wichtigen Anforderungen abzudecken. Aus diesem Grund werde ich hier zwei bisher bekannte Technologien vorstellen und dazu noch weitere, die beim „International Workshop on Requirements Engineering for Product Lines“ im September 2002 in Essen entwickelt wurden.

4.1. Feature Oriented Domain Analysis (FODA)



Das FODA-Konzept [2] wurde 1990 von Kang entwickelt. Es basiert auf dem Versuch mehrere verschiedene Alternativen und Optionen leicht verständlich und vor allem übersichtlich in einem Diagramm darzustellen.

Das vorliegende Beispiel zeigt einen typischen Feature – Baum mit einigen möglichen Relationen. In diesem Beispiel muss ein Auto einen Motor und ein Getriebe haben, währenddessen eine Klimaanlage nur ein optionaler Bestandteil ist. Das Getriebe muss entweder ein Schaltgetriebe oder ein Automatikgetriebe sein, während die Klimaanlage automatisch oder manuell eingebaut werden kann.

Ein 8-Zylindermotor muss ein automatisches Getriebe haben. Ein 4-Zylindermotor kann jedoch kein Automatikgetriebe besitzen.

Der Produktentwicklungs – Produktfindungsprozess besteht in diesem Modell daraus, den Feature-Baum in einer geordneten Art und Weise abzuarbeiten und dabei die optionalen Features auszuwählen. Das Ergebnis ist eine Produktbeschreibung die alle Features des Produktes beinhaltet (Feature Konfiguration).

Das Problem von FODA ist, dass eine textuelle Beschreibung der einzelnen Features sehr schnell zu einem unübersichtlichen Diagramm führen kann.

Man ist sich jedoch weitgehend einig, dass Modelle wie FODA und sein Nachfolger FORM [3] (Kang 1998) eine nützliche Struktur für Anforderungslisten bietet, welche eine Grundlage für vollständige, konsistente und widerspruchsfreie Spezifikationen bieten kann. Näheres zu FODA in der Ausarbeitung über Domain Analysis und Scoping.

4.2. 1-Level / 2-Level Use Cases

Bsp.: (Beispiel Nokia Handys) [4]

... (Normale UseCase Header)

Hauptzenario:

- Modell 3310: Das System stellt Spiel1 und Spiel2 zur Auswahl
- Modell 3330: Das System stellt Spiel1, Spiel2 und Spiel3 zur Auswahl.
- Der Benutzer wählt ein Spiel

- Das System stellt das Logo des gewählten Spiels dar
 - Modell 3310:
 - drücke Auswahl
 - Wähle Level und drücke YES
 - Wähle Schwierigkeitsgrad und drücke OK
 - Modell 3330:
 - drücke Auswahl
 - wähle Optionen und drücke OK
 - scrolle zu Schwierigkeitslevel und drücke YES
 - wähle gew. Schwierigkeitslevel und drücke YES
 - Das System startet das Spiel und spielt, bis es vorüber ist.
- ...

- Der Benutzer wählt ein Spiel
 - Das System stellt ein Logo des gewählten Spieles dar.
 - Der Benutzer wählt den Schwierigkeitsgrad unter Berücksichtigung der ([V2] angebrachten) Prozedur und wählt YES
 - Das System startet das Spiel und spielt bis es zuende ist.
- ...
- V0: Alternative
 - V0: 1. Modell Nokia 3310
 - 2. Modell Nokia 3330
 - V1: Optional
 - Falls V0=1 dann Spiel1 oder Spiel2, ansonsten Falls V0=2 dann Spiel1, Spiel2 oder Spiel3
 - V2: Parametrisiert
 - Falls V0=1 dann ProzedurA :
 - o drücke Auswahl
 - o scrolle zu Optionen und drücke YES
 - o scrolle zu Schwierigkeitsgrad und drücke YES
 - o wähle den gewünschten Schwierigkeitslevel und drücke YES
 - Ansonsten falls V0=2 dann ProzedurB:
 - o ...

Der 1-Level Use Case ist vom Prinzip her eine normale Use Case Beschreibung, welche Variationen über eine fortlaufende Aneinanderreihung der verschiedenen Möglichkeiten bietet. Diese Methodik ist die derzeit geläufige, um kleinere Variationen in einer Use Case Beschreibung auszudrücken, ohne gleich einen neuen Use Case konzipieren zu müssen.

Der Vorteil dieser Methodik liegt an der Formlosigkeit, wie die Variationen dargestellt und beschrieben werden. Jedoch sind genau diese Punkte zugleich zwei der größten Nachteile. Wenn eine mögliche Variante fehlt, ist erstmal nicht ersichtlich, ob sie einfach vergessen wurde oder ob hier keine Variante eingeplant war. Auch ist die Wartung beziehungsweise die Erweiterung um weitere Alternativen extrem aufwändig, da an jede einzelne Stelle gesprungen und im gesamten Text überall Flickwerk betrieben werden muss, was sehr schnell zu Inkonsistenzen führen kann.

Eine mögliche Abhilfe bietet hier der 2-Level Use Case, der die Flickwerk-Problematik durch die Einführung von Variablen umgeht. Diese Variablen werden in drei Kategorien eingeteilt: Optional, Alternativ und Parametrisiert

Alternative Komponenten drücken die Möglichkeit aus, eine spezifische Anforderung durch die Auswahl aus einer vordefinierten Liste auszuwählen, die jeweils vom Auftreten bestimmter Bedingungen abhängen.

Parametrisierte Komponenten hängen vom aktuellen Wert eines Parameters in der Anforderung für ein spezifisches Produkt ab.

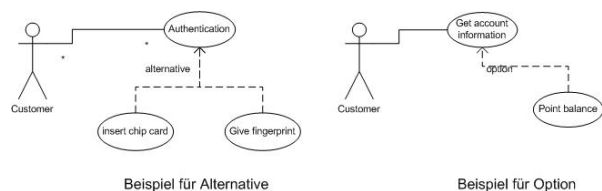
Optionale Komponenten können gewählt oder auch weggelassen werden. Sie sind nicht Bestandteil der „Muss“-Komponenten eines spezifischen Produktes.

- Bsp.: (Beispiel Nokia Handys) [5]
 Hauptaktor: Ein Handy der ([V0] 33-er Serie)
 ... (Normale UseCase Header)
 Hauptszenario:
 - Das System stellt eine Liste der ([V1]verfügbaren) Spiele dar

Der Vorteil dieser Methodik ist, dass zusätzliche Alternativen, Optionen und Parameter nur an einem zentralen Ort hinzugefügt werden müssen, wodurch der vorher erwähnte Flickenteppich in der Use Case Beschreibung nicht vorkommt.

Ein Nachteil dieser Methodik ist hier, dass eine gute Möglichkeit fehlt, diese Worte sinnvoll als UML Diagramm graphisch darzustellen.

4.3. Alternative / Option



Um das eben angesprochene Problem der übersichtlichen Darstellung zu umgehen wurde eine Use Case Diagrammerweiterung angedacht, die als Zusatzinformation zu jeder Beziehung noch Alternative und Option [6] dazuschreiben. Diese Möglichkeit bietet

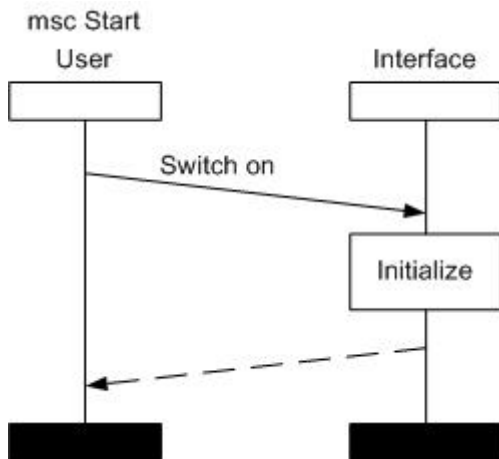
sich vor allem in einer Mischung mit den 2-Level Use Cases an.

Im vorliegenden Beispiel kann ein Kunde einer Bank entweder seine Chipkarte einführen oder seinen Fingerabdruck abgeben. Diese Wahl steht ihm als Alternative zur Verfügung. Auf der rechten Grafik hat der Benutzer die Auswahl ob er seinen Kontoauszug ausdrucken möchte. Dies ist eine Option seinen Kontoauszug abzufragen.

4.4. Erweiterte Message Sequence Charts

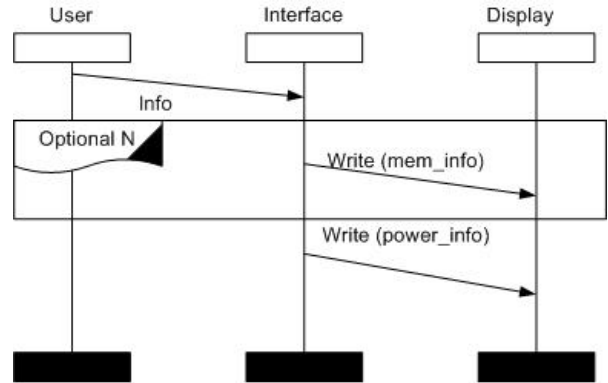
Zusätzlich zu den statischen Gesichtspunkten eines Produktes könnte der Benutzer zusätzlich noch Verhaltensalternativen definieren wollen. Eine Produktfamilie für Netzwerksoftware könnte verschiedene Protokolle verwenden und Komponenten eines Systems könnten unterschiedlich in verschiedenen Produkten zusammenarbeiten.

Message Sequence Charts [7] sind eine szenariobasierte Sprache die zuerst von ITU standardisiert wurde und deren derzeitige Version sehr nah an die von UML 2.0 herankommt.

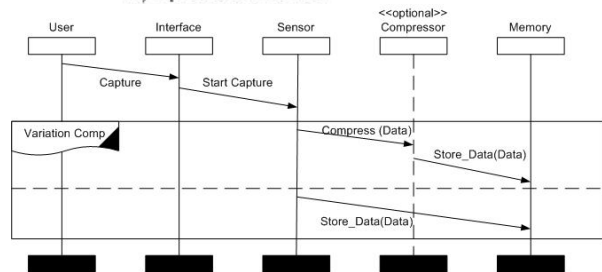


Dieser Diagrammtyp stellt die Nachrichten zwischen verschiedenen Objekten dar. In der Standardversion sind Variationen in Form von Optionen oder Alternativen nicht vorgesehen.

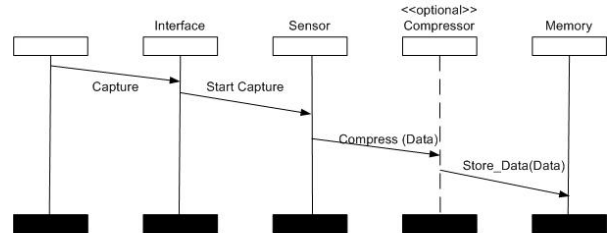
Daher wurden auch hier Erweiterungsmethoden angedacht, um diese beiden Arten auszudrücken.



A) Optional behaviour



B) optional instances and variation point



C) MSC with options and variation points fixed

Im linken oberen Beispiel ist die Möglichkeit ausgedrückt, nicht nur die Information über die Stromversorgung auszugeben sondern auch über den Inhalt des Speichers.

Im Beispiel rechts daneben sind die beiden Alternativen in einem zweigeteilten Prozess (komprimieren und dann speichern) oder einem einfachen Prozess der nur das Speichern ohne vorherige Komprimierung vorsieht.

Eine konkrete Realisierung in einem Produkt stellt das untere Bild dar, das die ausgewählte Variante (mit dem zweigeteilten Prozess) zeigt.

Diese Erweiterung stellt eine Erweiterung der UML dar, um verschiedene Verhaltensweisen ausdrücken zu können. Sie kann jedoch nur als Zusatz zu den anderen Diagrammformen gesehen werden, da sie schon extrem ins Detail geht und auf der Stufe der Anforderungsanalyse zuviel von einem möglichen Design festlegt.

5. Schlussfolgerung

Da die Idee der Produktlinien für Softwareprodukte bisher noch nicht hinreichend erforscht wurde, fehlen bisher noch gesicherte Erkenntnisse, wie speziell die Anforderungsanalyse für so komplexe Systeme wie eine Produktlinie übersichtlich und verständlich dargestellt werden kann.

Meiner persönlichen Meinung nach, ist zur Beschreibung einer Option oder Variante eine Vermengung von FODA und dem 2-Level Use Case am übersichtlichsten, da hierbei die Struktur sowohl grafisch als auch mit ausreichender Erklärung weitläufig beschrieben werden kann.

Alle hier von mir vorgestellten Techniken sind nur Ansätze dieses Problem zu lösen. Keine von ihnen kann es derzeitig alleine richten sondern bedarf immer die Unterstützung anderer Diagramm und Darstellungstypen. In diesem Bereich besteht daher noch erhöhter Forschungsbedarf.

6. Quellen

[1] Definition der Universität Wien, Institut der Wirtschaftsinformatik. [<http://www.pri.univie.ac.at/>]
Link zu den Folien:
[http://www.pri.univie.ac.at/~renatem/skripten/swe/280,23,Requirements Engineering – Erwerb](http://www.pri.univie.ac.at/~renatem/skripten/swe/280,23,Requirements%20Engineering%20-%20Erwerb)

[2] FODA - [Kang 90], Kang, K., et al. Feature-Oriented Domain Analysis (FODA)

[3] FORM - K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. *FORM: A feature-oriented reuse method with domain-specific reference architectures*. Annals of Software Engineering, 5:143--168, 1998.

[4] Nokia 1-Level - Papers des International Workshop on Requirements Engineering for Product Lines, Essen, Germany September 2002 *Use Case Description of Requirements for Product Lines*

[5] Nokia 2-Level - Papers des International Workshop on Requirements Engineering for Product Lines, Essen, Germany September 2002. *Use Case Description of Requirements for Product Lines*

[6] Alternative / Option - Papers des International Workshop on Requirements Engineering for Product Lines, Essen, Germany September 2002 *Modeling Variability by UML Use Case Diagrams*

[7] MSC - Papers des International Workshop on Requirements Engineering for Product Lines, Essen, Germany September 2002. *Modeling Variability by UML Use Case Diagrams*