Universität Passau

Fachbereich Informatik/Mathematik

Bachelorarbeit

im Studiengang Business Administration and Economics

Thema:	Klassifizierung von Methoden zur Qualitätsbeurteilung in der Software-Produktlinienentwicklung
eingereicht von:	Semah Senkaya ‹semah_senkaya@web.de›
eingereicht am:	23. September 2011
Betreuer:	Dr.-Ing. Sven Apel Universität Passau Fakultät für Informatik und Mathematik Innstraße 33, D–94032 Passau

Senkaya, Semah:
Klassifizierung von Methoden zur Qualitätsbeurteilung in der Software-Produktlinienentwicklung
Bachelorarbeit, Universität Passau, 2011.

Inhaltsverzeichnis

1 EINLEITUNG & ZIELSETZUNG

Um sich heutzutage besser am Markt behaupten und zeitgleich von der Konkurrenz abzuheben zu können geht der Trend in der Softwareentwicklung immer mehr zur individualisierten Massenproduktion sogenannte Software Produktlinien (SPL). Aufgrund dieser Entwicklung verändern sich jedoch auch die Anforderungen an die Entwickler sowie an die Software selber.

Zum einen stehen die Entwickler mit diesem Ansatz vor neuen Herausforderungen, um das Ziel der individualisierten Massenproduktion und der damit verbundenen Variabilität bei gleichbleibender Qualität zu realisieren (z. B. organisatorische und strategische Veränderungen im Unternehmen). Zum anderen kann durch diese neue Organisation auch die Wirtschaftlichkeit und Geschwindigkeit vor allem durch das Konzept der Wiederverwendung von Artefakten gesteigert werden.

Um all das realisieren zu können, müssen auch Qualitätsstandards überdacht werden. Im Gegensatz zur „klassischen“ Softwareentwicklung bei Stand-Alone-Produkten steht nicht mehr nur das Endprodukt im Vordergrund, jetzt ist vor allem jedes qualitativ hochwertige Artefakt, das später auch in anderen Softwaresystemen wiederverwendet werden kann, von Bedeutung. Durch die Möglichkeit der Wiederverwendung dieser Artefakte kann die Gesamtqualität einer Produktlinie gesteigert werden.

Es wurden schon einige Forschungsarbeiten geschrieben, um diese Qualitätsstandards zu bewerten oder Modelle vorgestellt, die diese bewerten können. Dabei werden Bereiche der Software-Produktlinienentwicklung, wie die Entwicklungsphasen, Artefakte, sowie die an der Entwicklung beteiligten Personen (*Stakeholder*) mit einbezogen. Auch die strukturierte Organisation der Bereiche und ihre Wechselwirkungen wird immer mehr thematisiert.

Die hier vorgestellte Arbeit stellt die oben beschriebene Thematik vor und definiert einige Begriffe und Bereiche der SPL-Entwicklung. Das Ziel dieser Arbeit ist es verschiedene Forschungsarbeiten kurz vorzustellen und diese anhand der oben genannten Kriterien (Entwicklungsphasen, Artefakte und Stakeholder) zu klassifizieren.

Visualisiert werden die Klassifizierungen anhand von Tabellen, einer Graphik und letztendlich einer Webseite, die zusätzlich Möglichkeiten bietet auf dieser Arbeit aufzubauen und die Webseite weiterzuentwickeln.

1.1 Aufbau

Die Arbeit ist wie folgt aufgebaut:
Im Abschnitt 2 werden die Grundlagen und Hintergründe einer SPL vorgestellt, um dann in Abschnitt 3 auf die für die Klassifizierungen wichtigsten Termini detaillierter einzugehen. Hier wird auch die Methodologie für die Klassifizierung beschrieben.
In Abschnitt 4 wird die eigentliche Klassifizierung vorgestellt und anhand von Tabellen und der Webseite visualisiert. Im Anhang A werden die Paper vorgestellt, klassifiziert und diese Klassifizierung kurz begründet.

2 SOFTWARE-PRODUKTLINIEN - GRUNDLAGEN

2.1 Vorgeschichte

Schon seitdem es die Softwareentwicklung gibt, existieren Ideen oder Ansätze wie man bereits bestehende Softwarekomponenten so miteinander verknüpfen kann, so dass daraus wieder „neue“ Produkte gebildet werden. M.D. McIlroy [Mci69] war 1969 einer der ersten, der offiziell davon sprach Softwaresysteme aus wiederverwendbaren Softwarekomponenten zu bilden. Seitdem wurde die Thematik immer wieder aufgegriffen, bis in den frühen 90ern das Konzept der Produktlinien eingeführt wurde und sich bis heute immer stärker auf dem Markt etabliert.
Folgende Tabelle bietet einen kleinen Auszug von Ansätzen und Entwürfen, die seit den 60er Jahren der Software-Produktlinienentwicklung die Richtung geben:

1969	M.D.McIlroy	**„Mass Produced Software Components.“** Report on A Conference Sponsored by the NATO Science
1976	D. Parnas	**„On the design and development of program families.“** IEEE Transactions on Software Engineering, 2(1):1-9
1986	J. Neighbors	**„The draco approach to constructing software from reusable components.“** IEEE Transactions on Software Engineering, 10(5):564-573
1988	R. Johnson, B. Foote	**„Designing Reusable Classes.“** Journal of Object Oriented Programming, 22-30.
1990	K. Kang, S. Cohen, J. Hess, W. Novak and S. Peterson	**„Feature-oriented domain analysis (FODA) feasibility study.“** Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University
1995	F. van der Linden and J. Müller	**„Creating architectures with building blocks.“** IEEE Software, 12(6):51-60
1996	R. R. Macala, L. D. Stuckey, D.C. Gross	**„Managing Domain-Specific Product-Line Development.“** IEEE Software, pp. 57-67
1997	D. Dikel, D. Kane, S. Ornburn, W. Loftus and J. Wilson	**„Applying Software Product-Line Architecture.“** IEEE Computer, pp. 49-55
1998	F. Van der Linden	**„Development and Evolution of Software Architectures for Product Families.“** Proceedings of the Second International ESPRIT ARES Workshop
2000	J. Bosch	**„Design & Use of Software Architectures - Adopting and Evolving a Product Line Approach.“** Wesley, ISBN 020167494-7
2000	SPLC	**SPLC 2000, First Software Product Line Conference** August 28-31, 2000, Denver, Colorado `http://splc.net/history.html`
2001	P. Clements, L. Northrop	**„Software Product Lines - Practices and Patterns“**, Pearson Education (Addison-Wesley), ISBN 0-201-70332-7

Tabelle 1: Ansätze, die die Entwicklung zur heutigen SPL beeinflusst haben

Parnas führte 1976 das Konzept der Produktfamilien ein und definierte diese wie folgt:
„Program families are defined (analogously to hardware families) as sets of programs whose common properties are so extensive that it is advantageous to study the common properties of the programs before analyzing individual members.“

1986 spricht J. Neighbors [Nei86] als einer der ersten von einer spezifischen Software-Domäne die als Basis dienen soll, um wiederverwendbare Komponenten zu entwickeln.
Erst in den frühen 90ern wird das Konzept der Software-Produktlinie eingeführt. Eines der ersten Beiträge war die der Feature-Orientierten-Domänen-Analyse (FODA).

Die restlichen in der *Tabelle 1* vorgestellten Arbeiten sollen einen groben Überblick verschaffen. Es gibt einige wichtige Arbeiten in diesem Bereich, die ich hier nicht genannt habe, da es sonst den Rahmen sprengen würde.

In [vGBS00] gehen die Autoren etwas weiter und sprechen von der intra-organisatorischen Wiederverwendung, wie z.B. die Wiederverwendung von Software zwischen den verschiedenen Softwareprodukten oder Systemen die von der Firma erstellt bzw. benutzt wird. Sie heben hervor, dass es zwar schon verschiedene Ansätze gibt, z. B. objekt-orientierte Frameworks [JF88] und komponentenbasiertes Softwareengineering [Szy97], aber der Durchbruch in der intra-organisatorischen Wiederverwendung erst mit Aufkommen der Software-Produktlinien erreicht wurde (Siehe [Lin98] und [Bos00a]). Zusammenfassend erklären sie, dass da, wo sich die früheren Ansätze noch auf die technologische Aspekte der Softwarewiederverwendung konzentriert haben, der neue Ansatz der SPL auf einer ganzheitlichen Herangehensweise beruht, in der die Punkte Geschäft, Organisation, Prozesse und Technologie gleichzeitig behandelt werden. Das wiederum führt, wie im nächsten Abschnitt beschrieben, dazu, dass sich der Nutzen aus der Einführung einer SPL auch auf all diese Bereiche ausweitet.

2.2 Vorteile

Unternehmen jeglicher Art und Größe haben herausgefunden, dass eine Produktlinie, wenn sie effektiv eingeführt und angewandt wird, eine Menge Nutzen liefert und dem Unternehmen einen Wettbewerbsvorteil bringt. Das gilt auch bei der Umstellung von einer klassischen Softwareproduktion auf eine Software-Produktlinienproduktion. Nicht nur im Entwicklungsbereich kann die Umstellung einen großen Nutzen mit sich bringen, sondern auch strategische Wettbewerbsvorteile für das gesamte Unternehmen.

Charles W. Krueger[1] beschreibt diese Vorteile wie folgt:

Im Entwicklungsbereich:

- Verringerung der durchschnittlichen Zeit ein neues Produkt zu erstellen und zu verteilen
- Verringerung der durchschnittlichen Fehler pro Produkt
- Verringerung des durchschnittlichen Entwicklungsaufwands für Verteilung und Wartung eines Produktes und somit die Verringerung der durchschnittlichen Entwicklungskosten pro Produkt
- Erhöhung der absoluten Anzahl an Produkten die effektiv verteilt und gemanagt werden können

Die daraus resultierenden strategischen Geschäftsvorteile:

- Verringerte „time-to-market“ und „time-to-revenue“ für neue Produkte
- Erhöhter Wettbewerbsvorteile des Produktes
- Höhere Profitmargen
- Verbesserte Reaktion, um Möglichkeiten des Marktes auszunutzen
- Bessere Produktqualität und erhöhte Reputation für Qualitätsprodukte
- Erhöhte Flexibilität, um schneller in neue Märkte vorzudringen
- Verringertes Risiko bei Produktplatzierungen

SPL-Techniken fassen explizit die Gemeinsamkeiten einer kompletten Produktlinie zusammen und profitieren davon. Durch die Anwendung dieser Techniken können Variationen von Produkten (Variabilität) innerhalb einer Produktlinie gemanagt und kontrolliert und jeglicher Mehraufwand an Entwicklungsarbeit eliminiert werden.

[1] Charles W. Krueger, PhD, CEO, BigLever Software, zitiert im September 2011 auf `http://www.softwareproductlines.com/benefits/benefits.html`

Je höher die Qualität einer SPL-Entwicklung ist, desto höher wird der Nutzen sein, der sich daraus ergibt.

Qualitätsvorteile

Höhere Produktqualität der SPL hat einen Einfluss auf die Geschäftsstrategie und die technische Entwicklung der SPL:

- höhere Kundenzufriedenheit führt zu einem festen Kundenstamm und steigert das Interesse am Produkt für Neukunden.
- weniger Produktfehler bedeuten weniger Gemeinkosten während des Lebenszyklus

Die folgende *Abbildung 1* wurde von Carl Shaulis of Salion[2] für die Software-Produktlinien-Webseite zur Verfügung gestellt:

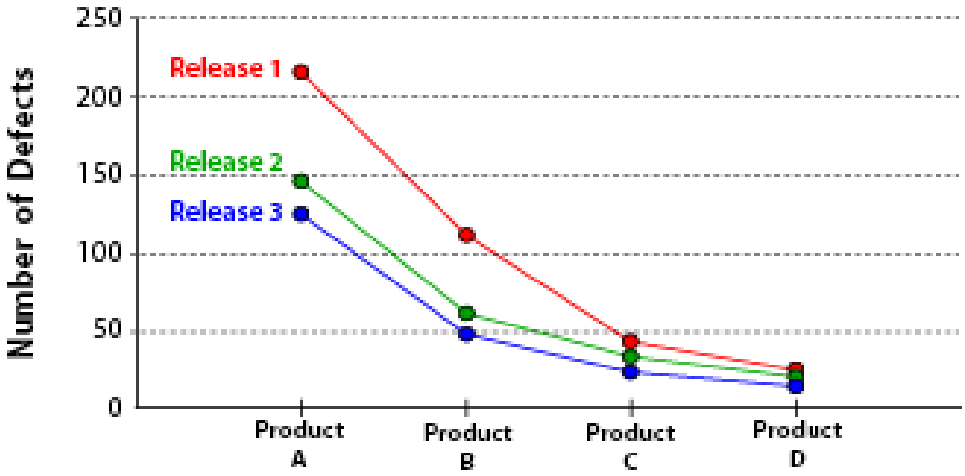

Abbildung 1: Fehlerreduzierung durch die Einführung einer SPL

Die *Abbildung 1* wird in Qualitätsvorteile einer SPL[3] wie folgt erklärt:

„Der Graph zeigt den Abwärtstrend von gefundenen Fehlern wenn eine Reihe von Produkten innerhalb einer SPL getestet wird. Dieser verläuft jeweils innerhalb einer Einzelfreigabe eines Produktes und über mehrere Freigaben (Release) des Produktes. Jede farbige Kurve stellt eine Reihe von Produkttest dar, für zeitgleich veröffentlichte Produkte einer Produktlinie. Jede abfallende Kurve zeigt, dass die Fehler die in frühen Tests (z. B. Product A) gefunden wurden, die Anzahl der Fehler die in den nachfolgenden Tests (z. B. Product B) gefunden wurden, reduzieren. Der Abwärtstrend der gesamten Fehler von Release 1 über Release 2 bis zu Release 3 zeigt, dass hochqualitative Software Artefakte durch den hohen Grad an Gemeinsamkeiten und sorgfältig gemanagten Variationen von Release zu Release entstehen.“

[2] Carl Shaulis of Salion, zitiert in `http://www.softwareproductlines.com/successes/salion.html` im September 2011
[3] `http://www.softwareproductlines.com/benefits/quality.html`, September 2011

2.3 Qualität

Welchen Qualitätsmerkmalen muss eine SPL entsprechen, um den oben genannten Nutzen voll ausschöpfen zu können? Um das beantworten zu können ist zunächst eine Definition von Qualität bei der Softwareentwicklung nötig. Im Report von Fitzpatrick [Fit96] bezieht sich dieser auf ein Qualitätsmodell zur klassischen Softwareentwicklung. Er übernimmt dabei die Definitionen von McCall [MRW77], die die Qualitätsfaktoren der Einfachheit halber auf die folgenden elf zusammengekürzt haben:

Effizienz	Das effiziente Nutzen von Code, um Prozesse ausführen und die Speicherressourcen effizient ausnutzen zu können.
Integrität	Das Vertrauen des Benutzers, dass Programme und Daten nicht unrechtmäßig verändert werden können, und dass Zugriffskontrollen erstellt werden können, die das verhindern.
Reliabilität / Funktionsfähigkeit	Die Fähigkeit eines Produktes oder eine Komponente, seine Aufgabe über einen bestimmten Zeitraum unter zuvor definierten Konditionen erfüllen zu können.
Benutzerfreundlichkeit	Ist die das Produkt für die vorgesehen Benutzergruppe erlernbar und bedienbar.
Genauigkeit	Oder auch Korrektheit. Wenn sich das Programm nach den Spezifikationen seiner Funktionen verhält.
Wartbarkeit	Wenn am Produkt Kosten und Fehler gesucht und bei Bedarf entsprechend behandelt werden können.
Testfähig	Wenn auf das Programm zugegriffen werden kann, um sicherzustellen dass die spezifischen Anforderungen erfüllt werden.
Flexibilität	In der aktuellen Auffassung von Flexibilität wird sie immer mehr mit Anpassungsfähigkeit verknüpft (z.B. Produkt an Kundenwünsche anpassen können).
Interface facility	Die Produktentwicklung wird so unterstützt, dass sie Produkte entwickelt die wiederum mit anderen Produkten interagieren können.
Wiederverwendbarkeit	Wenn das Programm in eine andere Anwendung transferiert werden kann, ohne dass zu hohe Kosten dadurch entstehen.
Übertragbarkeit	Der moderne Ausdruck für diesen Qualitätsfaktor ist *portability*. Portability ist die Strategie, die Software so zu entwickeln, dass diese mit minimalem Aufwand unter einem anderen Betriebssystem oder auf anderer Hardware laufen kann.

Tabelle 2: Qualitätsfaktoren der Softwareentwicklung nach McCall [MRW77]

Die einzelnen Qualitätsfaktoren sind nicht als eine isolierte Eigenschaft zu betrachten, nach der gesucht und die danach auch einzeln bewertet werden kann. Es gibt viele Fälle, in denen es ein Zusammenspiel und Abhängigkeiten zwischen den einzelnen Faktoren gibt. Diese Qualitätsfaktoren beziehen sich auf die Entwicklung eines klassischen Softwareprodukts und sollten somit auch grundlegende Qualitätsmerkmale für eine SPL sein.

Enrico Johansson and Martin Höst definieren in [JH02] den Begriff Qualität bei Software-Produktlinien wie folgt:

„Die Qualität wird angesehen als die Gesamtheit aller Feature (Merkmale) und Charakteristiken einer Softwareplattform, welche sich auf seine Fähigkeit stützt, festgelegten oder implizierten Bedarf einer Produktlinie (ISO 9000:2000, 2000) zu bedienen.“

Vor allem aber muss eine hohe Qualität, wie im nächsten Abschnitt beschrieben, bei der Identifizierung und Entwicklung der Variabilität und des Variabilitätsmanagements innerhalb einer SPL gewährleistet sein.

2.3.1 Variabilität

Mit der Variabilität und dem Variabilitätsmanagement steht und fällt eine Software-Produktlinie. Jilles van Gurp, Jan Bosch und Mikael Svahnberg beschreiben in [vGBS00] die Variabilität als die Fähigkeit, ein System zu verändern oder anzupassen, zu ermöglichen bestimmte Arten von Änderungen einfach und schnell vorzunehmen. Bei der Entwicklung einer Software-Produktlinie wird die Architektur eines Systems früh festgelegt, aber die Details einer tatsächlichen Produktimplementierung werden erst bei der Implementierungsphase benötigt. Sie bezeichnen diese verschobenen Designentscheidungen als Variationspunkte.

In [BFG+02] werden die verschiedenen Möglichkeiten, einen Variationspunkt zu implementieren beschrieben, um daraus in den späteren Phasen der SPL Produkte abzuleiten Die Autoren erklären, dass wenn der Architekt oder Designer sich entscheidet, die Designentscheidung zu verschieben, dieser dann einen Variationspunkt erstellen muss Dabei können sie sich für folgende entscheiden:

- **implizit**: In den Lebenszyklusphasen vor dem Design des Variationspunktes werden die Variationspunkte als nicht unentwickelt (implizit) betrachten
- **explizit**: In der oder nach der Phase, wenn der Variationspunkt designt ist, wird er als eindeutig (explizit) betrachtet.

Ein eindeutiger Variationspunkt kann an eine bestimmte Variante gebunden sein. Für jeden Variationspunkt kann die Menge an Variationen offen sein, d. h. es können Varianten hinzugefügt werden oder es kann geschlossen sein, d. h. es können keine weiteren Varianten hinzugefügt werden. Innerhalb dieser bestehenden Menge können verschiedenen Varianten verbunden werden. Schlussendlich kann ein Variationspunkt permanent verbunden werden, d. h. eine Variante kann permanent verknüpft werden.

In der folgenden Abbildung aus [BFG+02] stellen die Autoren dar, dass sich das Variabilitätsmanagement über alle Phasen des Domain und Application Engineering erstreckt.

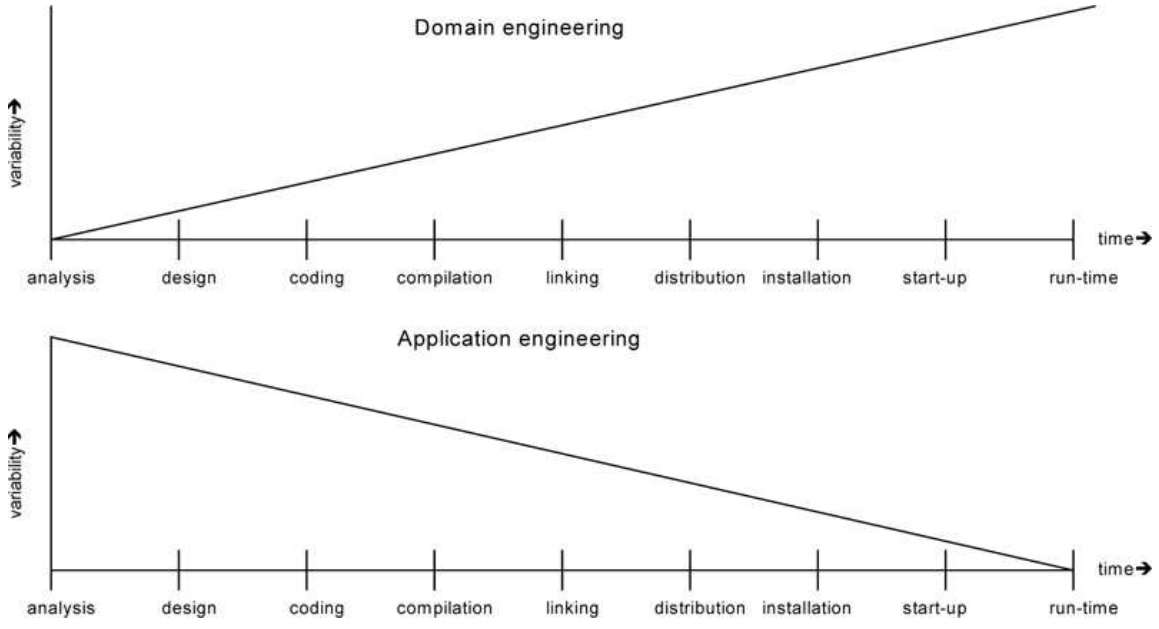

Abbildung 2: Variabilitätsmanagement über alle Phasen des Domain und Application Engineering hinweg

Eine kurze Zusammenfassung aus [BFG+02], die die *Abbildung 2* erläutert:
Softwareentwicklung in SPL kann als zweistufige Organisation betrachtet werden, d. h. Domain Engineering und Application Engineering. Das **Domain Engineering** behandelt vor allem die Identifizierung

von Gemeinsamkeiten und Varianten für die Produkte in der Produktlinie und implementiert die gemeinsam genutzten Artefakte in der Art und Weise, dass die Gemeinsamkeiten ausgenutzt werden können und gleichzeitig die nötige Variabilität erreicht wird. Während des **Application Engineerings** werden die einzelnen Produkte entwickelt, indem die gemeinsam genutzten Artefakte ausgewählt und angepasst, und wenn es notwendig, ist produktspezifische Erweiterungen hinzugefügt werden. D. h. die Variabilität der Software Artefakte entwickelt sich während des Domain Engineering (hier werden neue Variationspunkte eingeführt) und während des Application Engineerings (hier werden diese Variationspunkte an ausgewählte Varianten gebunden).

Variabilitätsmanagement:
Um die oben beschriebene Variabilität gewährleisten zu können, ist ein sorgfältiges Management der Variationspunkte sehr wichtig. Die Variationspunkte müssen flexibel genug gemanagt werden, um in Zukunft neue Anforderungen an weitere Produkten schnell und kostengünstig einbinden zu können. In [vGBS01] wird ein Modell vorgestellt, das die Aufgaben des Managements von Variationspunkten beschreibt. Ich fasse dieses Modell sehr vereinfacht zusammen:

Identifizieren der Varianten	Der erste Schritt in dem Prozess ist es, zu identifizieren wo Variabilität benötigt wird.
Einschränkung der Varianten	Sobald die Variationspunkte identifiziert wurden, müssen sie eingeschränkt werden. Es geht nicht darum unendliche Flexibilität zu bieten, sondern genau so viel, wie momentan und später in einer kosteneffektiven Weise benötigt werden.
Implementierung der Varianten	Basierend auf dem vorgegangenen Punkten, muss eine passende Technik zur Realisierung ausgewählt und ausgeführt werden.
Managen der Varianten	Abhängig davon, ob ein Variationspunkt offen ist oder nicht, wird ein Variantenmanagement nötig sein. In einigen Fällen, können Varianten per Hand hinzugefügt werden. Aber es ist in modernen System eher üblich, neue Varianten über das Internet runterzuladen und anschließend zu installieren.

Tabelle 3: Variabilitätsmanagementmodell nach Jilles van Gurp, Jan Bosch und Mikael Svahnberg beschreiben in [vGBS01]

2.4 Definitionen zur SPL

Als Abschluss zum Abschnitt Grundlagen will ich noch verschiedene Definitionen für Software-Produktlinien vorstellen und letztendlich mich auch an einer eigenen versuchen.

Das Software Engineering Institute (SEI) der Carnegie Mellon University in Pittsburgh, führte als erster den Begriff Software-Produktlinie ein und definiert diesen wie folgt[4]:
„A software product line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.“

Also nach dem SEI ist eine SPL eine Gruppe von softwareintensiven Systemen, die sich eine gemeinsame Menge an Features teilen, um spezifische Bedürfnisse eines bestimmten Marktes zu befriedigen oder eine bestimmte Aufgabe zu erfüllen. Diese Gruppe wird aus einer gemeinsamen Menge an Kernbestandteilen nach einer vorgeschriebenen Methode erstellt.

Definition für Software-Produktlinien in [vdHDM03]:

„Software product line engineering is a paradigm to develop software applications (software-intensive systems and software products) using platforms and mass customization.“

[4] `http://www.sei.cmu.edu/productlines` (September 2011)

Die Autoren beschreiben hier sehr kompakt eine Software-Produktlinienentwicklung als ein Modell, das Softwareanwendungen (softwareintensive Systeme und Softwareprodukte) durch die Verwendung einer Plattform und die Anwendung der individualisierten Massenanfertigung entwickelt.

Einen weiteren Ansatz Software-Produktlinien so kurz, aber auch so umfassend wie möglich zu beschreiben liefert Jan Bosch in [Bos00b]:

„. . . there has been a shift from world-wide reuse of components to organization-wide reuse. Parallel to this development, the importance of an explicit design and representation of the architecture of a software system has become increasingly recognized. The combinsation of these two insights lead to the definition of software product lines.
A software product line consists of a product line architecture, a set of reusable components and a set of products derived from the shared assets.“

Jan Bosch beschreibt hier, dass es eine Entwicklung gegeben hat, in der von der weltweiten Wiederverwendung von Komponenten auf eine unternehmensweite Wiederverwendung umgestellt wurde. Parallele dazu entstand ein Bewusstsein dafür, dass auf eine detaillierte Gestaltung und Repräsentation der Softwarearchitektur ein großer Wert gelegt werden muss. Diese beiden Entwicklungen lassen ihn die folgende Definition für eine SPL schlussfolgern:

Eine SPL besteht aus einer Produktlinienarchitektur, einer Menge von wiederverwendbaren Komponenten und einer Anzahl von Produkten, die aus den gemeinsamen Artefakten abgeleitet werden.

Wie ich eine Software-Produktlinie sehe:

Ein Grundstock an entwickelter Software, in dem alle Gemeinsamkeiten implementiert werden und an dem es viele „Andockmöglichkeiten“ (Variationspunkte) für Erweiterungen gibt, um Variabilität bei der Produktion zu gewährleisten. Diese Variationspunkte werden offen gelassen, damit durch das „Andocken“ von unterschiedlichen Komponenten verschiedene Produkte abgeleitet werden können. Das Ganze geschieht unter dem Aspekt der Wiederverwendbarkeit von bestehenden und neu zu entwickelnden Komponenten, um letztendlich eine schnellere, billigere und bessere Produktionsmöglichkeit in Form einer SPL zu entwickeln.

Eine Software-Produktlinie ist so komplex, dass sie durch ein oder zwei Sätze nicht annähernd beschrieben werden kann. In den folgenden Abschnitten werde ich einen Überblick über die einzelnen Phasen und Teilprozesse, die Artefakte und die an den Entwicklungsprozessen einer SPL beteiligten Personen schaffen, indem ich die die wichtigsten Termini definiere.

3 TERMINOLOGIE & METHODOLOGIE

In diesem Abschnitt werde ich die für die Klassifizierung wichtigsten Begriffe definieren und gegebenenfalls kurz erklären, um dann zum Schluss meine Vorgehensweise zu erläutern. Ich habe größtenteils die Definitionen aus den von mir zur Klassifizierung verwendeten Forschungsarbeiten entweder übernommen oder zusammengefasst übersetzt. Das ist im Folgenden durch die Quellenangaben gekennzeichnet.

3.1 Die Entwicklungsphasen der SPL

3.1.1 Domain Engineering

Den Begriff *Domain Engineering* werde ich während dieser Bachelorarbeit aus dem Englischen übernehmen.

John D. McGregor beschreibt in [McG09] das Domain Engineering als einen Bereich der SPL-Entwicklung, in dem die Fähigkeiten eines Unternehmens, Produkte aus ihrer bestehenden SPL herzuleiten realisiert wird (Realisierung der Entwicklungsfähigkeiten einer SPL). Dieser Bereich entwickelt die wiederverwendbaren Artefakte (z. B. Komponenten), die für die endgültige Herstellung eines Produktes benötigt werden.

Die Personen, die für die Realisierung dieser Aufgaben zuständig sind, arbeiten entweder in den Teams, die tatsächlich die Produkte bilden oder in Teams für die Herstellung der einzelnen Produktteile. Die Outputs tragen zur Entwicklung der Produktlinienarchitektur und der Softwarekernbestandteile bei. Sie sind ein wichtiger Bestandteil der SPL und decken einen umfangreichen Aufgabenbereich ab. In diesem Bereich werden alle Möglichkeiten für Produktableitungen analysiert und festgehalten.

Unterteilung des Domain Engineering in einzelne Entwicklungsphasen

Die Definitionen für die folgenden Begriffe entnehme ich aus dem Buch [PBL05] und fasse sie kurz zusammen.

Domänenanforderung
Während der Produktlinienphase der Domänenanforderung werden die Anforderungen bezüglich aller Gemeinsamkeiten und Variabilität der Produktlinie analysiert, erfasst und für die folgenden Phasen dokumentiert.
Der **Input** für diesen Teilprozess besteht aus dem Produktplan (roadmap). Der **Output** bildet sich aus den wiederverwendbaren, dokumentierten und modellbasierten Anforderungen und insbesondere aus dem Variabilitätsmodell der Produktlinie. Demzufolge deckt der Output nicht die Anforderungsspezifikationen einer bestimmten Applikation ab, sondern die gemeinsamen und variablen Anforderungen für alle absehbaren (alle die beinhaltet sind oder es noch werden) Applikationen der Produktlinie.

Domänendesign
Diese Teilphase der SPL umfasst alle Tätigkeiten, die die Referenzarchitektur der Produktlinie definieren. Diese Referenzarchitektur bestimmt die Struktur (statische und dynamische Zerlegung, die für alle Applikationen gültig ist) und die Textur (Sammlung von allgemeingültigen Richtlinien bezüglich Design und Umsetzung) aller Applikationen der SPL. D. h. die Referenzarchitektur bietet eine allgemeingültige, hochgradige Struktur für alle Applikationen der Produktlinie.
Der **Input** für diesen Teilprozess besteht aus den Domänenanforderungen und während der Phase der Domänenanforderungen definierten Variabilitätsmodells. Der **Output** umfasst die Referenzarchitektur und ein verfeinertes Variabilitätsmodell, das die sogenannte interne Variabilität (d. h. Variabilität die aus technischen Gründen notwendig ist) umfasst.

Domänenrealisierung
Die Domänenrealisierungsphase befasst sich mit dem detailliertem Design und der Implementierung von wiederverwendbaren Softwarekomponenten.
Der **Input** für diesen Teilprozess besteht aus dem Variabilitätsmodell der Referenzarchitektur, die eine

Liste aller wiederverwendbaren Artefakte beinhaltet, die während der Domänenrealisierung entwickelt werden sollen. Der **Output** umfasst in allen Einzelheiten die Artefakte für das Design und die Implementierung der wiederverwendbaren Softwarekomponenten.

Domänenvalidierung
Diese Phase ist für die Validierung und Kontrolle der wiederverwendbaren Komponenten verantwortlich. Während der Domänenvalidierung werden die Komponenten darauf getestet, ob sie den Spezifikation, z. B. Anforderungen, Architektur und Designartefakte entsprechen. Zusätzlich werden hier wiederverwendbare Testartefakte entwickelt, um den Aufwand für das Testen während der Applikationsphase zu verringern.
Der **Input** für diesen Teilprozess besteht aus den Domänenanforderungen, der Referenzarchitektur, dem Interface Design und den implementierten wiederverwendbaren Komponenten. Der **Output** umfasst die Testergebnisse der erfolgten Tests und die wiederverwendbaren Testartefakte.

3.1.2 Application Engineering

Den Begriff *Application Engineering* werde ich während dieser Bachelorarbeit aus dem Englischen übernehmen.

John D. McGregor beschreibt in [McG09] das Application Engineering als einen Bereich, in der SPL-Entwicklung, in dem die während des Domain Engineering entwickelten Outputs (siehe oben) zum Ableiten der Produkte angewandt werden. Also die endgültige Produktion der Produkte.

Unabhängig von der Art einer SPL ist das Ziel eines SPL-Unternehmens die Produktableitung so effizient wie möglich zu gestalten. Um das zu erreichen, arbeitet das Team des Application Engineering mit den vorab entwickelten Kernbestandteilen und Methoden des Domain Engineering. Diese Kernbestandteile und Methoden sollen garantieren, dass die Produkte den Anforderungen und Bedürfnissen des Unternehmens genügen.

Natürlich gibt es immer einen Teil des Produktes, der neu und anders ist und der nicht direkt aus den bestehenden Kernbestandteilen gebildet werden kann. Die Produktionsfähigkeiten des Unternehmens müssen deshalb so flexibel sein, dass jeder Zeit neue Kernbestandteile entwickelt und mit den bestehenden schnell verknüpft werden können.

Unterteilung des Application Engineering in die einzelnen Entwicklungsphasen

Die Definitionen für die folgenden Begriffe entnehme ich aus dem Buch [PBL05] und fasse sie kurz zusammen.

Applikationsanforderung
Diese Phase umfasst alle Tätigkeiten, um die Spezifikationen für die Applikationsanforderungen zu entwickeln. Der erreichbare Grad an wiederverwendbaren Domänenartefakten hängt stark von den Applikationsanforderungen ab. Deshalb ist einer der größten Anliegen dieser Entwicklungsphase die Unterschiede zwischen dem Ist- und dem Sollzustand der Applikationsanforderungen und der abrufbaren Leistungsfähigkeit der Plattform zu identifizieren.
Der **Input** für diesen Teilprozess besteht aus den Domänenanforderungen und dem Produktplan (roadmap) mit allen wichtigen Features der entsprechenden Applikation. Zusätzlich kann es noch spezifische Anforderungen (z. B. Kundenwünsche) für diese bestimmte Applikation geben, die noch nicht während der Domänenanforderungsphase erfasst wurde. Der **Output** ist die Anforderungsspezifikation dieser bestimmten Applikation.

Applikationsdesign
Während der Phase des Applikationsdesigns wird die Applikationsarchitektur entwickelt. Das Applikationsdesign verwendet die Referenzarchitektur, um die Applikationsarchitektur zu instanziieren. Es wählt die benötigten Teile der Referenzarchitektur aus, konfiguriert diese und bezieht applikationsspezifische Anpassungen ein. Die Variabilitätsbindung im Applikationsdesign bezieht sich auf die gesamte Struktur

des berücksichtigten Systems (d. h. die spezifischen Hardware-Geräte die in dem System vorhanden sind). Der **Input** für diesen Teilprozess besteht aus der Referenzarchitektur und der Spezifikationen aus der Applikationsanforderungsphase. Der **Output** erstellt die Applikationsarchitektur für die aktuelle Applikation.

Applikationsrealisierung
Die Hauptaufgabe hier ist die Auswahl und die Konfiguration der wiederverwendbaren Softwarekomponenten sowie die Erstellung der applikationsspezifischen Artefakte. Wiederverwendbare und applikationsspezifische Artefakte werden dann zusammengesetzt, um die Applikation zu erstellen.
Der **Input** besteht aus der Applikationsarchitektur und der wiederverwendbaren Realisierungsartefakten der Plattform. Der **Output** ist eine lauffähige Applikation mit detaillierten Designartefakten.

Applikationsvalidierung
Dieser Teilprozess umfasst alle Tätigkeiten, die nötig sind, um eine Applikation auf ihren Deckungsgrad mit der Spezifikation zu prüfen und für gültig zu erklären.
Der **Input** besteht aus allen möglichen Applikationsartefakten die als Testreferenz verwendet werden können, der implementierten Applikationen und den wiederverwendbaren Testartefakten, die durch die Domänenvalidierung entstanden sind. Der **Output** ist ein Testreport der die Ergebnisse aller Tests, die durchgeführt wurden, umfasst. Zusätzlich wurden alle entdeckten Fehler in detaillierter Form in einem Problemreport dokumentiert.

3.1.3 Die Phasen der Organisation innerhalb SPL-Entwicklung

Bei dem Begriff „Organisation“ geht es um das Managen der SPL-Phasen während des Domain- und Applikation Engineering, aber auch das Managen der Prozesse (Definition Prozesse siehe unten) dieser Phasen. Da ich die einzelnen Phasen der SPL bereits oben schon begrifflich erklärt habe, werde ich nicht weiter auf den Begriff der Organisation eingehen.
Der Versuch alle Zusammenhänge, Verantwortlichkeiten und Aufgaben, also Ablaufprozesse innerhalb einer SPL-Entwicklung wiederzugeben, würde den Rahmen sprengen.

Es gibt unterschiedliche Organisationsmodelle, die wie z. B. in [Bos02] eine Möglichkeit darstellen, die Entwicklungsprozesse einer SPL zu managen.

3.1.4 Zeitlicher Ablauf der Phasen

Einen festen zeitlichen Ablaufplan über alle SPL-Phasen hinweg vorzugeben oder zu erkennen ist sehr schwierig. Während der Einführungsphase einer SPL, wird zunächst damit begonnen die Anforderungen zu analysieren und zu definieren, auf denen dann alles aufgebaut wird. Was normalerweise als nächstes folgen würde, wäre das Design dieser Anforderungen. Die Stakeholder dieser Phasen müssen, um eine hohe Qualität gewährleisten zu können, regelmäßig Rücksprache halten, Deswegen kann es durchaus vorkommen, dass neue oder zusätzliche Anforderungen definiert werden müssen und damit wieder in die Domänenanforderungsphase zurückgekehrt wird. Das geschieht über alle Phasen der SPL hinweg. Da nicht alle Artefakte oder Komponenten für die Produktableitung vorherbestimmt werden können, bleiben über den gesamten Lebenszyklus der SPL die einzelnen Teilprozesse bestehen. Diese werden weiter entwickelt und ausgeführt, um bestehende Artefakte oder Komponenten anzupassen oder neue zu entwickeln. Also können Abläufe von mehreren Teilprozessen ineinander übergehen oder parallel ablaufen.

3.2 Artefakte

Anforderungen

Definition of Requirements in [IEEE 1990][5]

A requirement is:

1. A condition or capability needed by a user to solve a problem or achieve an objective.
2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.
3. A documented representation of a condition or capability as in (1) or (2).

Also ist laut IEEE eine Anforderung:

1. Eine Bedingung oder Fähigkeit, die von einem Anwender benötigt wird, um ein Problem zu lösen oder ein Ziel zu erreichen.
2. Eine Bedingung oder Fähigkeit, die das System oder einer Systemkomponente entweder erfüllen oder besitzen muss, um einen Vertrag, eine Norm, eine Spezifikation oder ein anderes formell eingeführtes Dokument zu erfüllen.
3. Eine dokumentierte Darstellung einer Bedingung oder Fähigkeit wie in (1) oder (2).

Architektur

In [vdHDM03] diskutieren die Autoren, dass Softwarearchitekturen hochgradige Abstraktionen sind, die Struktur, Verhalten und Schlüsselbestandteile eines Softwaresystems abbilden.

Diese Abstraktionen beinhalten:

1. Beschreibungen der Elemente, aus denen Systeme gebildet werden
2. die Wechselbeziehungen zwischen diesen Elementen
3. Modelle die ihre Zusammensetzung anleiten
4. Nebenbedingungen für diese Modelle

Im Allgemeinen ist ein System als eine Gruppe von Komponenten, ihren Wechselwirkungen (Bindeglieder) und die all umfassende Organisation (Konfiguration) dieser Komponenten und Bindeglieder definiert. Während eine „reguläre“ Architektur die Struktur eines einzelnen Produktes definiert, definiert eine Produktlinienarchitektur (PLA) die gemeinsame Architektur für eine Gruppe von verwandten Produkten.

Eine PLA spezifiziert explizit:

1. Elemente die in allen Produkten vorhanden sind
2. Elemente die optional sind
3. Variationspunkte unter diesen Produkten

[5] IEEE; IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990), IEEE Computer Society, 1990.

Besondere Produktinstanzen werden durch die Auswahl von gewünschten, optionalen Komponenten und einem Element pro Variationspunkt ausgesucht. Perry [Per98] hat den Möglichkeitsraum für die Modellierung einer PLA umschrieben und beobachtete, dass die PLA-Modellierungstechnik zwei Bedingungen entsprechen sollte. Zum einen sollte die PLA allgemein genug entwickelt werden, um alle Teile einer Produktlinie zu umfassen und gleichzeitig spezifisch genug sein, um den Entwicklern genügend Unterstützung für die Instanziierung und Implementierung von individuellen Produkten zu bieten.

Komponente

Definition of a component by Szyperski [Szy97]:

A unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.

Nach [Szy97] ist somit eine Komponente eine Kompositionseinheit, die aus vertraglich spezifizierten Schnittstellen und eindeutiger Kontextabhängigkeit besteht. Sie kann unabhängig verwendet werden und kann von Dritten zum weiteren Gebrauch beansprucht werden.
Oder in Wikipedia[6] anschaulich formuliert:
Eine Komponente zeichnet sich dadurch aus, dass sie ein Element einer komponentenbasierten Anwendung darstellt und definierte Schnittstellen zur Verbindung mit anderen Komponenten besitzt.

Testen
In [PBL05] werden Testartefakte definiert als Produkte des Testprozesses, der aus Plänen, Spezifikationen und Testergebnissen besteht. Eine umfassende Übersicht über Testartefakte wird im IEEE Standard für Softwaretestdokumentation [IEEE 1998][7] aufgeführt.
Von mir hier zusammengefasst:

Im **Testplan** wird die Ressourcenallokation bestimmt. Es werden logische und detaillierte Testfälle entwickelt, die unter anderem unterschiedliche Möglichkeiten aufweisen sollen die vorgegebenen Ziele zu erreichen.
Ein **Testfallszenario** beschreibt eine spezifische Abfolge von Aktionen. Die Ausführung dieses Szenarios endet mit dem Erreichen des Testziels. Jede Aktion ist als eine Szenario-Maßnahme definiert.
Eine **Szenario-Maßnahme** bei Testfällen beinhaltet Anweisungen für den Tester bestimmte Aktionen auszuführen und optional das erwartete Ergebnis für diesen Testabschnitt.
Eine **Test-Zusammenfassung** enthält einen Überblick über die Ergebnisse von bestimmten Testausführungen.

Prozesse
Prozesse stellen hier als Artefakt die Wechselwirkung aller Aufgaben, Zusammenhänge und Verantwortlichkeiten dar, die bei der Software-Produktlinienentwicklung entstehen.

3.2.1 Die Stakeholder

Den Begriff *Stakeholder* werde ich während dieser Bachelorarbeit aus dem Englischen übernehmen.

Es sind mehr Stakeholder direkt und indirekt an der Entwicklung einer SPL und der Ableitung der einzelnen Produkte beteiligt, als ich hier angebe. Bei den unten aufgeführten gebe ich die Aufgabenbereiche und Verantwortlichkeiten so kompakt wie möglich wieder. Sie sind die für die Klassifizierung in dieser Arbeit wichtigsten Stakeholder.

[6] Wikipedia:Softwarekomponente, September 2011
[7] IEEE Computer Society; IEEE Standard for Software Test Documentation, IEEE STD 829-1998, IEEE Press, IEEE Computer Society, Los Alamitos, 1998

Entwickler
Der Softwareentwickler arbeitet an der Planung, dem Design und der Implementierung der Software, also an den Kernbestandteilen (hier vor allem Komponenten) der SPL.

Architekt
Der Architekt arbeitet an der technischen Planung und der Entwicklung der Softwarearchitektur. Die einzelnen Verantwortlichkeiten können aus der Definition für die Architektur (siehe oben) herausgelesen werden.

Projektmanager
Der Projektmanager ist für die operative Gestaltung und Ausführung des Projektes zuständig. Er ist dafür verantwortlich, dass alle Anforderungen an das Projekt effizient durchgeführt werden, um die Ziele des Unternehmens zu erreichen.

Designer
Der Designer arbeitet an der Planung und erstellt Entwürfe für Software-Lösungen.

Produktmanager
Der Produktmanager muss dafür sorgen, dass das zu entwickelnde Produkt allen Anforderungen und Bedürfnissen, die von den Kunden und dem Unternehmen erwartet werden entspricht. Er ist dafür verantwortlich, dass alle Anforderungen an das Produkt effizient durchgeführt werden, um die Ziele des Unternehmens zu erreichen.

Tester
Der Softwaretester prüft, ob die einzelnen Softwarebestandteile ihren vordefinierten Anforderungen genügen. Der Tester ist dafür verantwortlich, dass alle Fehler behoben und die Qualitätsstandards eingehalten werden.

Kunde
Der Kunde äußert seine Bedürfnisse und Anforderungen an das Produkt. Durch Rücksprache mit dem Kunden sollen Verbesserungsvorschläge für bereits bestehende Produkte und Anregungen für neue Produkte angenommen werden.

3.3 Methodologie

Die im Abschnitt der Terminologie definierten Begriffe fügen sich in *Abbildung 3* [PBL05] zu einen Gesamtkonzept zusammen. Auf diesem basierend werden in [PBL05] alle oben aufgeführten Termini detailliert beschrieben und auch ihre Zusammenhänge erklärt.

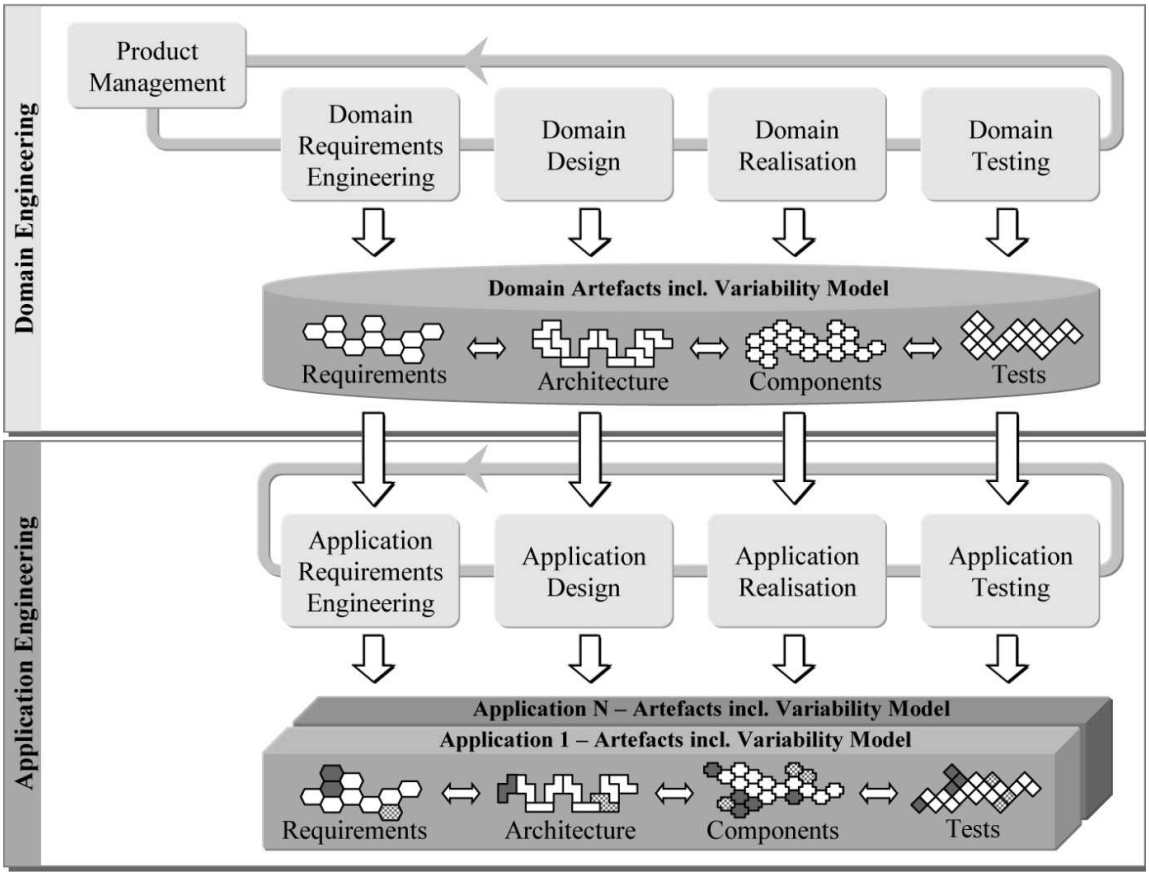

Abbildung 3: Eine Grundstruktur für die Software-Produktlinienentwicklung

Um die von mir ausgewählten Paper bestmöglich zu klassifizieren habe ich mich für ein dreidimensionales Koordinatensystem entschieden, dass die in der *Abbildung 3* zu sehenden Bereiche umfasst. Die Achsen des Koordinatensystems:

x-Koordinate: Die Phasen der Software-Produktlinien (**SPL-Phasen**).
y-Koordinate: Die an der SPL-Entwicklung beteiligten **Stakeholder**.
z-Koordinate: Die **Artefakte** einer SPL.

In *Abbildung 4* habe ich aus Gründen der Übersichtlichkeit die einzelnen Stakeholder auf der y-Achse weggelassen, sowie das Domain Engineering, Application Engineering und die Organisation in ein Koordinatensystem eingefügt. Eine detaillierte Ansicht der Klassifizierungen ist auf der von mir erstellten Webseite zu finden. Die zu den Nummerierungen gehörenden Paper, sowie die detaillierte Klassifizierung, gebe ich im Abschnitt der Klassifizierung in Tabelle 4 an.

An dieser Stelle soll lediglich ein grober Überblick verschafft werden.

In der von mir erstellten Webseite habe ich für jeden Bereich (Domain Engineering, Application Engineering und Organisation) ein eigenes Koordinatensystem erstellt, um eine detaillierte Klassifizierung anschaulich darzustellen. Auf die Webseite werde ich im nächsten Abschnitt näher eingehen.

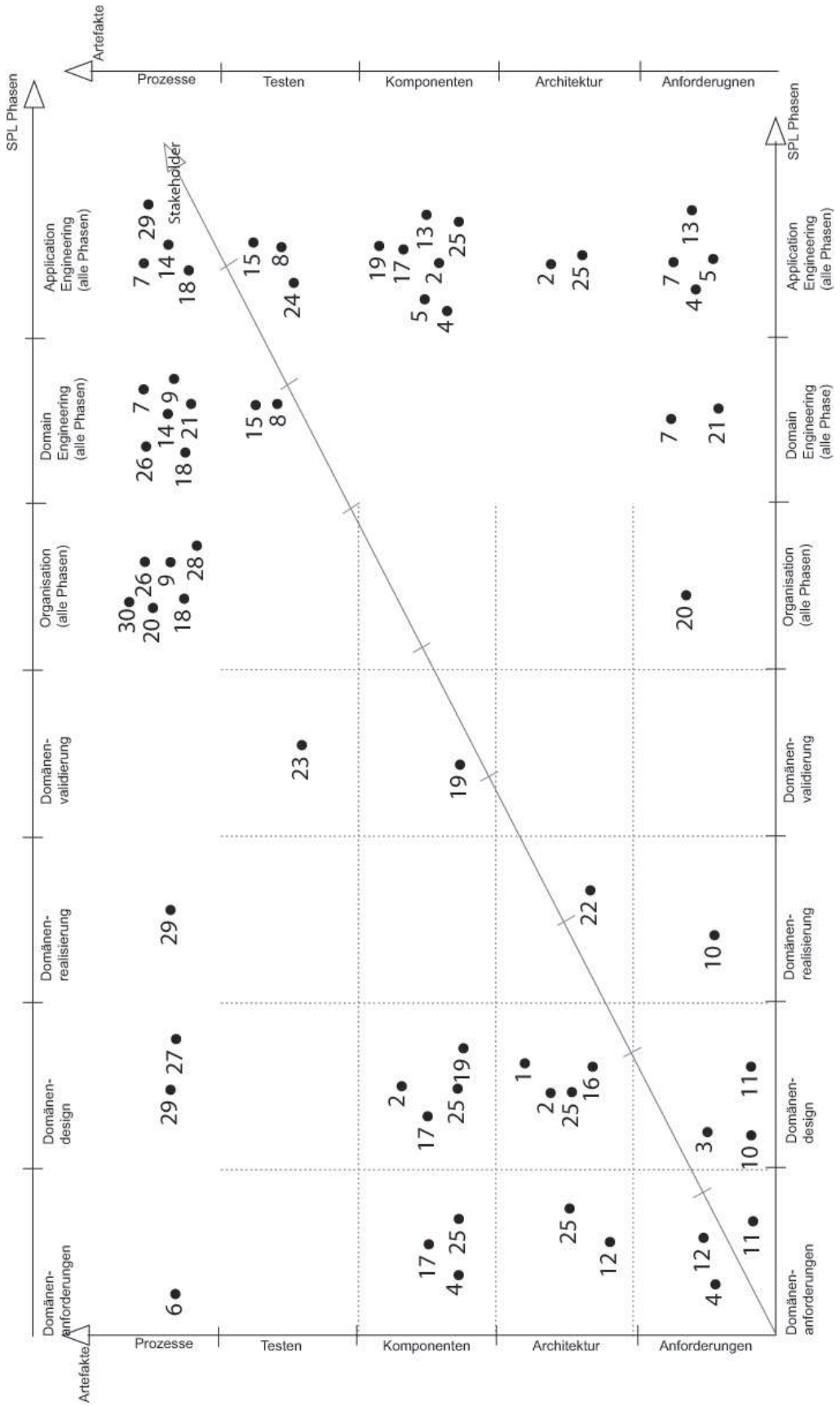

Abbildung 4: Überblick über die Klassifizierungen der Paper

4 KLASSIFIZIERUNG & VISUALIERUNG

4.1 Klassifizierung der Paper

In der folgenden Tabelle werde ich die Paper nummerieren, einen Verweis auf das Literaturverzeichnis und den Titel hinzufügen, sowie die einzelnen SPL-Phasen, Artefakte und Stakeholder angeben. Durch das Anklicken der Titel gelangen sie auf den Anhang, in dem jedes Paper einzeln kurz beschrieben und die Klassifizierung kurz begründet ist.

Werden bei den Klassifizierungen die Begriffe Domain Engineering, Application Engineering und Organisation genannt, so bezieht sich die Klassifizierung auf alle Phasen dieses Bereichs.

Nr.	Quelle	Titel	Klassifizierung
1	[vdHDM03]	Using Service Utilization Metrics to Assess the Structure of Product Line Architectures	Domänendesign, Architekt, Architektur
2	[Bos02]	Maturity and Evolution in Software Product Lines: Approaches, Artefacts and Organization	Domänendesign, Applikationsdesign, Architekt, Architektur, Komponenten
3	[JH02]	Tracking Degradation in Software Product Lines through Measurement of Design Rule Violations	Domänendesign, Architekt, Anforderungen
4	[SB00]	Issues Concerning Variability in Software Product Lines	Domänenanforderungen, Applikationsanforderungen, Architekt, Entwickler, Anforderungen, Komponenten
5	[BH01]	Product Instantiation in Software Product Lines: A Case Study	Applikationsanwendung, Applikationsdesign, Architekt, Entwickler, Anforderungen, Komponenten
6	[vdLBK+04]	Software Product Family Evaluation	Domain Engineering, Entwickler, Architekten, Produktmanager, Projektmanager, Anforderungen, Prozesse
7	[BFG+02]	Variability Issues in Software Product Lines	Domain Engineering, Application Engineering, Entwickler, Architekten, Anforderungen, Prozesse
8	[Mus96]	Software-Reliability-Engineered Testing	Domain Engineering, Application Engineering, Tester, Testen

Continued on next page

Tabelle 4 – continued from previous page

9	[Bos00b]	Organizing for Software Product Lines	Domain Engineering, Organisation, Projektmanager, Architekten, Entwickler, Prozesse
10	[vGBS00]	Managing Variability in Software Product Lines	Domänendesign, Domänenrealisierung, Architekten, Entwickler, Anforderungen
11	[vGBS01]	On the Notion of Variability in Software Product Lines	Domänenanforderungen, Domänendesign, Entwickler, Anforderungen
12	[Sch02]	A Comprehensive Product Line Scoping Approach and Its Validation	Domänenanforderungen, Produktmanager, Entwickler, Anforderungen, Architektur
13	[YR06]	A Configuration Management Model for Software Product Line	Application Engineering, Produktmanager, Projektmanager, Entwickler, Anforderungen, Komponenten
14	[DKMT96]	Analytical and Empirical Evaluation of Software Reuse Metrics	Domain Engineering, Application Engineering, Entwickler, Prozesse
15	[CDS06]	Coverage and Adequacy in Software Product Line Testing	Domain Engineering, Application Engineering, Entwickler, Tester, Anforderungen, Testen
16	[FCS+08]	Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability	Domänendesign, Designer, Entwickler, Architekten, Architektur
17	[CB91]	Identifying and Qualifying Reusable Software Components	Domänenanforderungen, Domänendesign, Applikationsanforderungen, Applikationsdesign, Entwickler, Komponenten
18	[Böc05]	Innovation Management for Product Line Engineering Organizations	Domain Engineering, Application Engineering, Organisation, Alle am Projekt beteiligten, Prozess

Continued on next page

Tabelle 4 – continued from previous page

19	[MSC03]	Measuring Component Reliability	Domänendesign, Domänenvalidierung, Applikationsdesign, Applikationsvalidierung, Entwickler, Designer, Komponenten
20	[CJNM05]	Project Management in a Software Product Line Organization	Organisation, Projektmanager, Anforderungen, Komponenten, Prozesse
21	[TP03]	Quality Modeling for Software Product Lines	Domain Engineering, Alle Beteiligten, Anforderungen, Prozess
22	[KAK08]	Granularity in Software Product Lines	Domänenrealisierung, Entwickler, Architektur, Komponenten
23	[WSB+08]	Automated Diagnosis of Product-line Configuration Errors in Feature Models	Domänenvalidierung, Entwickler, Testen
24	[UGKB08]	Testing Software Product Lines Using Incremental Test Generation	Applikationsvalidierung, Entwickler, Tester, Testen
25	[JJ03]	Coping with variability in product-line architectures using component technology	Domänenanforderungen, Domänendesign, Applikationsanforderungen, Applikationsdesign, Architekten, Entwickler, Architektur, Komponenten
26	[McG08]	Agile Software Product Lines, Deconstructed	Domain Engineering, Application Engineering, Alle Beteiligten, Prozesse
27	[NOQ+08]	Visualisation of Variability in Software Product Line Engineering	Domänendesign, Applikationsdesign, Alle Beteiligten, Prozesse
28	[BBM05]	Tracing Software Product Line Variability - From Problem to Solution Space	Domain Engineering, Alle Beteiligten, Prozesse
29	[McG09]	Goal-driven Product Derivation	Domain Engineering: Anforderungen und Design, Application Engineering: Anforderungen und Design, Entwickler, Designer, Prozesse

Continued on next page

Tabelle 4 – continued from previous page

30	[BCK+03]	Exploring the Context of Product Line Adoption	Organisation, Alle Beteiligten, Prozesse

Tabelle 4: Liste der klassifizierten Paper

4.2 Klassifizierung und Beurteilung von drei ausgewählten Papers

In diesem Abschnitt habe ich drei Paper ausgewählt, um deren Inhalt detailliert wiederzugeben und dadurch einen Einblick zu schaffen, wie ich bei der Klassifizierung der Paper vorgegangen bin. Der Grund für die Auswahl dieser Paper steht jeweils in den Ausführungen

- **Paper 20:**
  Hier wird erklärt wie Vorteilhaft, aber auch Komplex die Einführung einer SPL sein kann. Anhand eines Fallbeispiels wird für den Leser nochmal bewusst gemacht was vor allem das Management dafür an Organisation berücksichtigen muss. Da dies ein essentieller Teil der SPL-Entwicklung darstellt, habe ich mich entschieden näher darauf einzugehen.

- **Paper 21:**
  Damit eine SPL einwandfrei funktioniert und durch sie schnell, billig und problemlos ein neues Produkt entwickelt werden kann, muss diese bestimmte Qualitätsstandards erfüllen. In diesem Paper wird bezüglich dieser Thematik ein guter Überblick geschaffen, weshalb ich es zur detaillierten Bewertung ausgewählt habe.

- **Paper 28:**
  In dem vorgestellten Modell sollen über alle Phasen der Software Produktlinienentwicklung hinweg Variabilität und Variationspunkte aufgenommen, modelliert und für alle Beteiligten nachvollziehbar gemacht werden. Mit der Variabilität steht und fällt eine SPL. Da hier ein anschauliches und gut verständliches Variabilitätsmodell vorgestellt wird, habe ich mich entschieden es detaillierter zu beurteilen.

Sicherlich gibt es noch Themen, die für den ein oder anderen wichtiger erscheinen, als die die ich hier vorstelle. Mir haben gerade diese Paper sehr geholfen einen guten Überblick für das große Ganze zu bekommen. Themen im Bereich der SPL-Entwicklung gibt es viele, die nicht nur für sich alleine genommen sehr wichtig, sondern einen großen Beitrag für das Gesamtkonzept SPL leisten. Auf all diese detailliert einzugehen und sie vorzustellen würde mehrere Regale füllen.

4.2.1 Paper 20

Project Management in a Software Product Line Organization

Paul C. Clements, Lawrence G. Jones, and Linda M. Northrop, (Software Engineering Institute) and
John D. McGregor
Paper [CJNM05]

Die Autoren diskutieren die Frage, wie ein Projekt und die Projektmanagementmaßnahmen erweitert werden müssen, damit von einer Einzelproduktplanung in die SPL-Planung über gegangen werden kann. Vor allem in Bezug auf allgemeine Richtlinien, Strategien und Prozesse, die essentielle Bestandteile der SPL-Entwicklung sind.

Dabei verschaffen drei grundlegende Aufgabenbereiche einen guten Überblick über die oben genannten Punkte (*Abbildung 5*):

- *core asset developement:*
  Das Entwickeln von wiederverwendbaren Kernbestandteilen und vor allem deren **Organisation**. Das können sowohl Code als auch andere Artefakte sein. **Organisation** der wiederverwendbaren **Komponenten**
- *product developement:*
  Aus dem Output des *core asset development* und den Anforderungen werden neue Produkte innerhalb der SPL entwickelt
- das organisationale *Management* dieser Punkte

Mit dem Ziel allen **Anforderungen** an die Organisation gerecht zu werden und die **Prozesse**, die sich bei der SPL-Entwicklung bilden zu überblicken und zu koordinieren (Anforderungen, Prozesse).

Im Vordergrund stehen die Projekte, die aus den oben genannten Aufgabenbereichen resultieren und die Aufgaben des Projektmanagements. Die **Projektmanager** sollen über alle Phasen der Organisation der Produktlinie und die daraus zu entwickelnden Produkte hinweg die im Paper genannten Aufgaben ausführen.

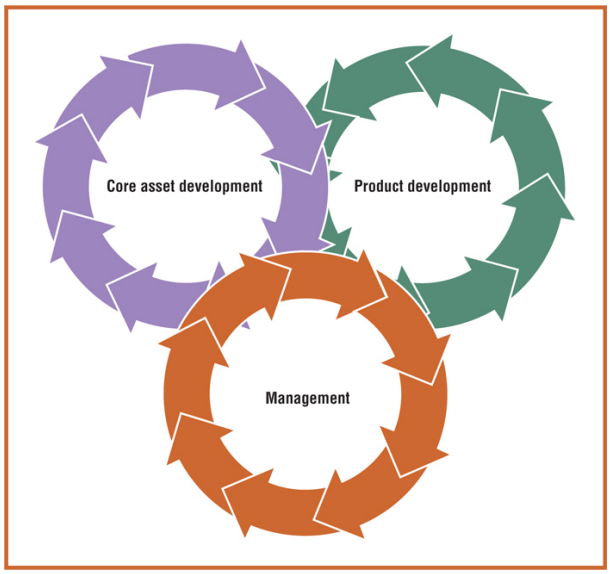

Abbildung 5: Die drei grundlegenden Aufgabenbereiche der Produktlinienentwicklung

Management Strategien
Die Autoren setzen in ihrem Paper voraus, dass die Projektmanager der einzelnen Hauptaufgabenbereiche sich untereinander verstehen, d. h. jeder weiß, was der Aufgabenbereich des anderen ist und wie die Umsetzung funktioniert. Also eine gute Koordination und Kommunikation zwischen den einzelnen Projekten.

Die Projektmanager, die in mehreren Aufgabenbereichen tätig sind, müssen dafür sorgen, dass sie mit ihrer Tätigkeit die Unternehmensziele unterstützen. Dies kann nur erreicht werden, wenn sowohl das *core asset development* als auch das *product development* in Form von einzelnen oder mehreren Projekten organisiert, koordiniert und aufeinander abgestimmt werden.

Das Hauptziel der **product development Projekte** ist es, Produkte zu entwickeln. Dies soll vor allem durch den bestmöglichen Gebrauch der Endprodukte des *core asset development* und die Umsetzung der Produktanforderungen, des Scoping, der Komponenten und des Produktplans geschehen.

Das Hauptziel der **core asset development Projekte** ist es diese genannten Punkte zu identifizieren, zu entwickeln und dem *product development* Projekt bereitzustellen.

Desweiteren heben sie hervor, dass sich SPL-Projekte von denen der Einzelproduktion unterscheiden, und dass das Management dies antizipieren muss, um erfolgreich die Unternehmensziele zu erreichen. In diesem Zusammenhang werden die drei grundlegenden Aufgabenbereiche als Projekte näher beschrieben und ihre Wechselbeziehung erläutert. Das vorgestellte Fallbeispiel soll diese Aussagen bekräftigen.Sie stellen ein neuartiges Projekt vor, bei dem es sich um ein organisationales Management von Software-Produktlinien handelt, dass sich hauptsächlich durch eine lückenlose Koordination der *core asset development* und der *product development* Projekte erfolgreich durchführen lässt. Dafür setzen sie ein starkes und visionäres Management voraus, gehen detaillierter auf deren Aufgabenbereiche ein und weisen hier auch wieder daraufhin, dass sich die Managementaufgaben von denen der Einzelproduktion unterscheiden.

In der ersten Tabelle stellen die Autoren die Unterschiede zwischen einem Produktlinienprojekt und einem klassischen Projekt dar, diskutiert anhand der drei grundlegenden Aufgabenbereiche *core asset development*, *product development* und *organisational management*. Die zweite Tabelle ist eine Übersicht über die Management Praktiken, die für alle Projektarten relevant sind.

In diesem Abschnitt wird über das **Projektmanagement** diskutiert, wie diese ihre Aufgaben organisieren und bestmöglich koordinieren sollen, um eine effiziente Software Produktlinie zu entwickeln und durchzuführen. Das Ziel ist die bestmögliche **Organisation** des Projektmanagements zur Ein- und Durchführung einer SPL. Im Vordergrund stehen die **Anforderungen** an die SPL, sowie die daraus resultierenden Projekte. In diesen Projekten soll eine stabile Basis an Kernbestandteilen, vor allem wiederverwendbare **Komponenten**, entwickelt und jegliche **Prozesse**, die entstehen, bestmöglich koordiniert werden. Das darauf folgende Fallbeispiel soll die beschriebene Vorgehensweise zur Produktlinieneinführung und -durchführung inhaltlich wiedergeben und untermauern.

Das Fallbeispiel:
Durch die Erfahrungen mit unterschiedlichen Kunden inspiriert, verdeutlichen sie die oben genannten Projekte anhand einer fiktiven Produktlinie in einem fiktiven Unternehmen:

AGM produziert drei Computerspiele (Games) für drei unterschiedliche Märkte (Markets). Für die unterschiedlichen Märkte bietet das Unternehmen Produktvariationen (Plattform- Variation) an. Die Dimensionen der Produktmatrix (*Abbildung 6*) lassen einige Produktvariationen naheliegend erscheinen: Spielregeln, Schnittstellen, durch den Markt auferlegte Grenzen, wie z. B. Speichergröße, Schnelligkeit des Prozessors und Plattformunterschiede.

Um mit den schnell wachsenden Produktvarianten und der immer geringeren Zeit zur Produktentwicklung mithalten zu können, führt AGM die SPL ein. Dazu wird ein organisationales Managementprojekt gegründet, aus dem wiederum Teilprojekte wie unter anderem das *core asset development* Projekt und

product development Projekt entstehen, die durch ein organisationales Management entwickelt und wie oben beschrieben durchgeführt werden.

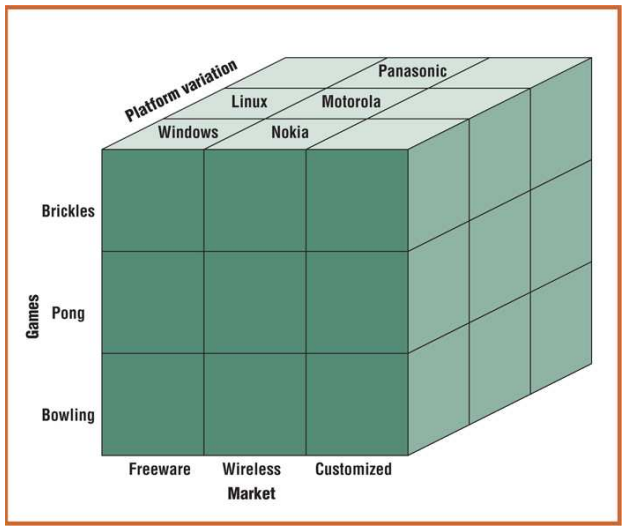

Abbildung 6: AGM Produktmatrix

Bei dem *core asset development* Projekt stellen die Entwickler fest, dass sie zu wenige Informationen über die Kernbestandteile (*Core Assets*) haben, um diese effektiv nutzen zu können. Daraufhin wird ein Produktionsplan entwickelt, der helfen soll die Kernbestandteile besser auszuwählen, die dazu gehörige Variabilität anzuwenden und diese für die Produktentwicklung optimal zu nutzen. Dieser Plan wird wiederum als Projekt von den product development und von den Archtitekturprojekt-Mitarbeitern durchgeführt.

Während des *product development* Projekts werden verschiedene Ansätze durchgeführt, die AGM vor allem folgende Vorteile ermöglichen:

- Das *product development* wird so entwickelt und ausgeführt, dass währenddessen die Basis für zukünftige Produkte gelegt wird.
- Die Komplexität und die Risiken (z. B. das Scheitern) eines jeden Projekts werden reduziert, um den Vorgaben des Ablaufplans, des Budgets und des Qualitätsstandards gerecht zu werden.
- Das reduziert AGMs Anfangsinvestitionen, führt aber zu einer komplizierteren Koordination des inzwischen größeren Projektportfolios.
- Durch extreme Programmierpraktiken, die im Paper kurz aufgelistet werden, wird dem Koordinationsproblem entgegen gewirkt
- Durch die regelmäßige Bestätigung, dass die Koordination erfolgreich verlaufen ist, wird Kommunikationsproblemen zwischen den einzelnen Projekten vorgebeugt.
- Im Verlauf der Projekte hat sich eine solide Basis an Kernbestandteilen entwickelt, so dass die Anzahl an Konkurrenzprojekten, die zur Entwicklung von zusätzlichen Produkten notwendig sind, stark gesunken ist.
- Das *Refactoring* der Architektur und das daraus folgende Refactoring der anderen Kernbestandteile (Core Assets) wurde stabilisiert.

(**Refactoring**: bezeichnet in der Software-Entwicklung die manuelle oder automatisierte Strukturverbesserung von Programm-Quelltexten unter Beibehaltung des beobachtbaren Programm-Verhaltens. Dabei sollen die Lesbarkeit, Verständlichkeit, Wartbarkeit und Erweiterbarkeit verbessert werden, mit dem Ziel, den jeweiligen Aufwand für Fehleranalyse und funktionale Erweiterungen deutlich zu senken.

Wikipedia:Refactoring)

In diesem Paper wird erklärt, wie vorteilhaft, aber auch komplex die Einführung einer SPL sein kann. Im Fallbeispiel wird für den Leser nochmal bewusst, was vor allem das Management dafür an Organisation berücksichtigen muss, deswegen habe ich mich entschieden näher darauf einzugehen. So kann ich meine Entscheidung, das Paper in den Bereich Organisation zu klassifizieren nochmal verdeutlichen.

4.2.2 Paper 21

Quality Modeling for Software Product Lines

Adam Trendowicz and Teade Punter
Paper [TP03]

Die Autoren definieren Anforderungen an ein Qualitätsmodell, um in den frühen Phasen der SPL-Entwicklung Qualität zu gewährleisten. Sie untersuchen bestehende Qualitätsmodelle auf diese Anforderungen, um dann das von ihnen entwickelte Modell Prometheus vorzustellen, das bis dahin als einziges Qualitätsmodell allen Anforderungen genügt. Dabei spielen bei der Erfüllung der Qualität die Verantwortlichkeiten, Zusammenhänge und Beziehungen über alle Phasen des *Domain Engineering* eine große Rolle (**Prozesse**).
Während der Vorstellung des Qualitätsmodells *Prometheus* wird ersichtlich, dass alle an der SPL Beteiligten, d. h. vom Kunden bis zum Architekten, in diesem Modell eine Rolle spielen (Alle Stakeholder).

Eigenschaften eines Qualitätsmodels für Software-Produktlinien
In diesem Abschnitt werden die Hauptanforderungen an ein angemessenes Qualitätsmodell vorgestellt. Diese sind Flexibilität, Wiederverwendbarkeit und Transparenz.

Flexibilität
Hier ist das Gewährleisten von Flexibilität aufgrund der Kontextabhängigkeit bei der Qualität der Software gemeint. Dabei werden die Zusammenhänge innerhalb des Unternehmens, der einzelnen Projekte und der **Prozesse** in Bezug auf Qualität näher erläutert.

Ein flexibles Qualitätsmodell sollte:

- von verschiedenen Unternehmen angewandt werden können
- für jede Projektdomäne anwendbar sein und die Betrachtungsweisen (an Qualitätscharakteristika und deren Zusammenhänge) von **allen** am Projekt beteiligten **Stakeholder** berücksichtigen.
- keinen starren Prozess voraussetzen. Es sollte ermöglichen ein Modell zu entwickeln, das an unternehmensspezifische Charakteristiken des Entwicklungsprozesses angepasst werden kann.
- in den Anfangsphasen einer SPL eingeführt werden, um stabile Qualitätsstandards zu gewährleisten.
- alle Charakteristiken des Umfelds einer Softwareprojekts, die einen Einfluss auf die Softwareprodukte haben, mit einbeziehen (**Prozesse**)

Wiederverwendbarkeit
Die Wiederverwendung der Erfahrungen aus bereits bestehenden Qualitätsmodellen, sowie die Wiederverwendung von Messdaten und Qualitätscharakteristiken und deren Zusammenhänge stehen hier im Vordergrund. Dadurch werden Zeit und Kosten bei der Qualitätssicherung gespart und die Genauigkeit und Effizienz von Qualitätsbewertungen verbessert. Es kann gewährleistet werden, dass qualitativ hochwertige Produkte entwickelt werden können.

Transparenz
Ein Qualitätsmodell sollte Grundprinzipien von bestimmten Charakteristiken, deren Zusammenhänge zu anderen und die Untercharakteristiken offenlegen und klar definieren. Dadurch sollen Redundanzen und Kontradiktionen zwischen Qualitätscharakteristiken verstanden und identifiziert werden. Zusätzlich sollten Projektstakeholder jeder Zeit auf das Modell zugreifen und, wenn nötig, es modifizieren können.

Übersicht über bestehende Qualitätsmodelle
Die Autoren stellen unterschiedliche Qualitätsmodelle vor und bewerten diese in Bezug auf die im vorherigen Abschnitt besprochenen **Anforderungen** an ein effizientes Qualitätsmodell.

Sie kommen zu dem Schluss, dass alle vorgestellten Qualitätsmodelle ein wichtiges Problem gemeinsam haben:

- Ein Mangel an umfassenden Regelwerken darüber, wie eine einheitliche Darstellung von Qualitätscharakteristiken und ihren Zusammenhängen erstellt werden kann.
- Die Unfähigkeit, Erfahrungen aus anderen Qualitätsmodellen, die über unterschiedlich Projekte und Unternehmen hinweg gemacht wurden, wiederzuverwenden, um so die Effizienz von Qualitätsbewertungen zu verbessern.

Laut den Autoren decken die hier vorgestellten Qualitätsmodelle kaum die besprochenen Anforderungen ab, stattdessen reproduzieren sie die Schwächen und die Mängel der anderen Modelle.

PROMETHEUS: Ein Ansatz für ein Qualitätsmodell in Software-Produktlinien
Das von den Autoren entwickelte Qualitätsmodell Prometheus soll im Gegensatz zu bis dahin existierenden Modellen allen Anforderungen die in den vorigen Abschnitten diskutiert wurden genügen.

Prometheus besteht aus drei Phasen:

- Spezifikationsphase (Entwicklung des Qualitätsmodells)
- Applikationsphase (Bewertung von spezifizierten nicht funktionalen Anforderungen)
- Packaging Phase (Diese Phase dient zum Erfassen von Erfahrungen, die während der Modellapplikation gemacht wurden, um das Modell zu verbessern und die Möglichkeit der Wiederverwendung zu gewährleisten)

Das Qualitätsmodell erfüllt folgende **Anforderungen** (*Abbildung 7*):

- Qualitätsziele definieren (durch Befragung der Kunden),
- Qualitätsmerkmale spezifizieren (Spezifikation der Inhalte des Modells von **allen Beteiligten** des *Domain Engineering*),
- Beziehungen spezifizieren (Spezifikation der Modellstruktur von allen Beteiligten des *Domain Engineering*),
- das Modell bewerten (durch alle Beteiligten des *Domain Engineering*),
- das Modell operationalisieren (Modellbewertung, vor allem von Domänenexperten).

Hier wird ersichtlich, dass alle an der Software Produktlinie Beteiligten, d. h. vom Kunden bis zum Architekten, in diesem Modell eine Rolle spielen.

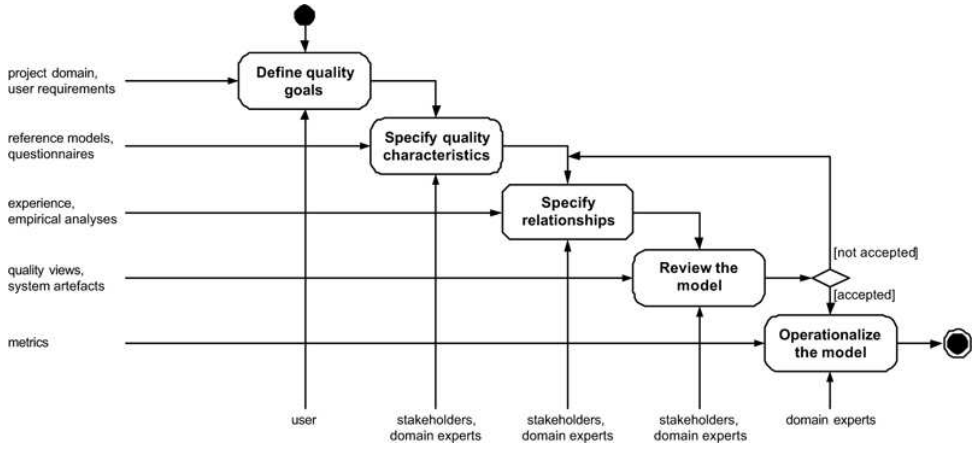

Abbildung 7: Aufgabenbereiche während der Spezifikationsphase des Prometheus Modells

Anhand einer weiteren Abbildung und Tabelle wird die Applikationsphase näher beschrieben, die wiederum auf der Spezifikationsphase aufbaut und deswegen keine weiteren Ansätze zur Klassifizierung dieses Papers bietet.
Die Autoren empfehlen Prometheus bereits in den frühen Phasen des Domain Engineering einer Software Produktlinie einzusetzen, um allen Qualitätsanforderungen besser gerecht werden zu können. Angewandt wird Prometheus, wie schon im Abstract des Papers angemerkt, über **alle Phasen** des *Domain Engineering.*

4.2.3 Paper 28

Tracing Software Product Line Variability - From Problem to Solution Space

KATHRIN BERG and JUDITH BISHOP,DIRK MUTHIG
Paper [BBM05]

Die Autoren stellen hier ein konzeptionelles Variabilitätsmodel vor, um Nachvollziehbarkeit von Variabilität in Software-Produktlinien zu gewährleisten und vergleichen dieses Modell mit dem klassischen Feature-Modell. Unter Nachvollziehbarkeit ist hier die Beschreibung von Beziehungen und Abhängigkeiten zwischen zwei Artefakten gemeint, die während den unterschiedlichen Phasen der Softwareentwicklung entwickelt werden (alle Phasen des *Domain Engineering*, **Prozesse**).

Das in diesem Paper vorgestellte Modell erlaubt eine eins-zu-eins Abbildung der Variabilität zwischen dem Problemraum und dem Lösungsraum und entspricht gleichzeitig den Anforderungen eines vereinheitlichten Ansatzes.

- **Problemraum**: Beschreibung und Festlegung des Systems, vor allem während der Analyse und der Entwicklung der Anforderungen im Domain Engineering.
- **Lösungsraum**: Das konkrete System, das vor allem während der Architektur-, Design- und Realisierungsphase entwickelt wird.

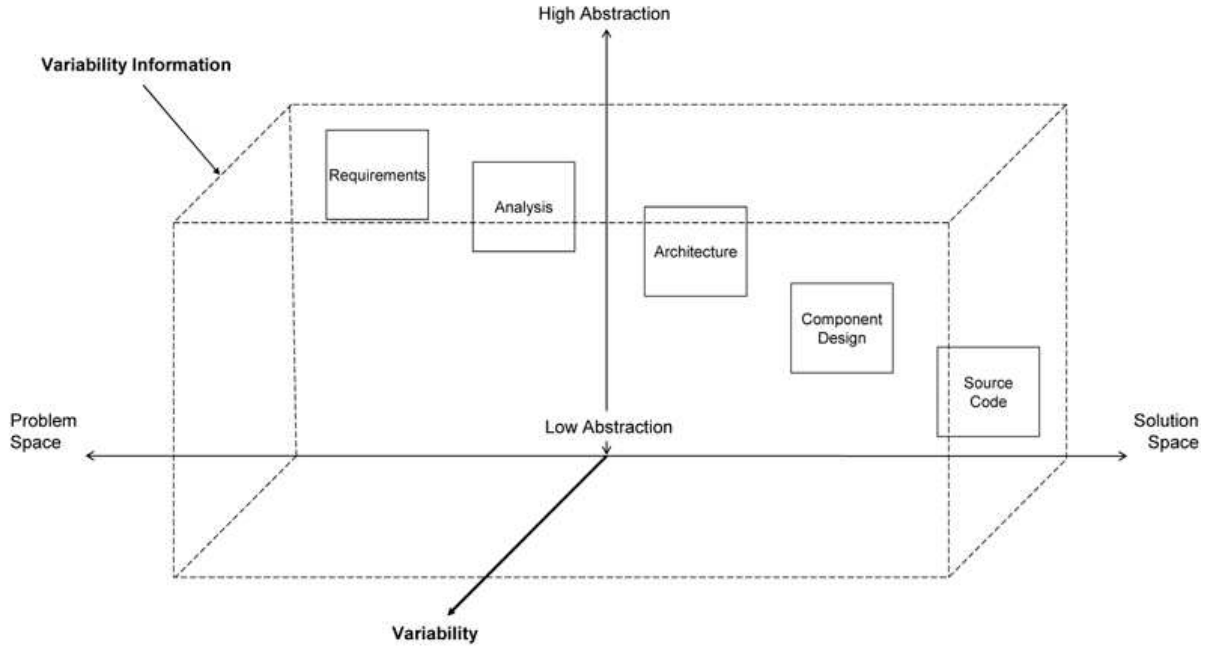

Abbildung 8: Die drei Stufen der Software Produktlinienentwicklung

Für die Softwareentwicklung von Ein-Produktsystemen sind laut den Autoren zwei Dimensionen ausreichend. Eine Dimension stellt die Entwicklungsphasen und die andere die Abstraktionsebenen dar, in denen alle Artefakte der Entwicklung einen Platz finden. Die Autoren führen für die SPL-Entwicklung eine zusätzliche dritte Dimension ein: die Variabilität (*Abbildung 8*).

Diese Dimension beinhaltet explizit alle Informationen über die Variabilität zwischen den einzelnen Teilen der SPL vor allem bezüglich des Managements der Variabilität.

Der hauptsächliche Beitrag dieses Papers ist, ein konzeptionelles Variabilitätsmodel zu entwickeln, um Nachvollziehbarkeit von Variabilität in Software-Produktlinien zu gewährleisten. Dies soll erreicht werden durch:

1. die detaillierte Modellierung von Variabilität und der damit verbunden Informationen in einer dritten Dimension. Die Informationen werden getrennt von Artefakten anderer Entwicklungsphasen gesammelt und in der dritten Dimension wiedergegeben.
2. die Erfüllung aller Anforderungen für den Ansatz eines vereinheitlichten Variabilitätsmanagements.

Im zweiten Abschnitt wird der vereinheitlichte Ansatz für ein Variabilitätsmanagement vorgestellt. Die Autoren behaupten hier, dass dieser Ansatz vier charakteristische Merkmale erfüllen muss: Konsistenz, Skalierbarkeit, Nachvollziehbarkeit und Visualisierung. Der Ansatz soll konsistent und skalierbar sein, Nachvollziehbarkeit aller Variationspunkte auf unterschiedlichen Abstraktionsebenen und über alle Entwicklungsphasen hinweg gewährleisten und Hilfsmittel bereitstellen, um Variabilität zu visualisieren [BM05].

Das konzeptionelle Variabilitätsmodell für Nachvollziehbarkeit
In den folgenden zwei Unterabschnitten werden die Konzepte des Variabilitätsmodells und das Modell an sich vorgestellt.

Die Konzepte
Als Konzepte werden der Problem- und Lösungsraum vorgestellt, die schon von [CE00] eingeführt wurden (*Abbildung 9*).

Problemraum: Beschreibung und Festlegung des Systems, vor allem während der Analyse und der Entwicklung der Anforderungen im *Domain Engineering*.

Lösungsraum: Das konkrete System, das vor allem während der Architektur-, Design- und Realisierungsphase des *Domain Engineering* entwickelt wird.

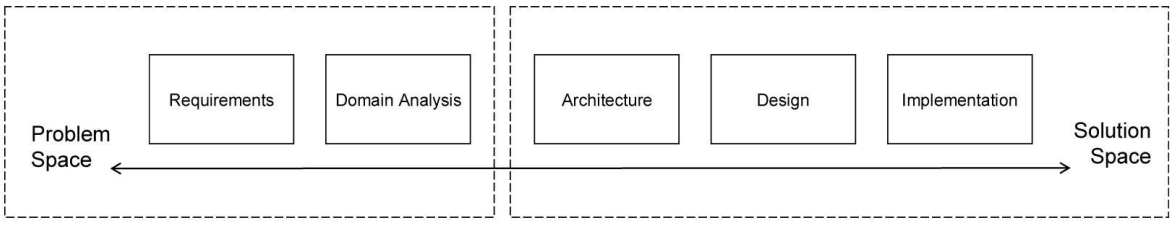

Abbildung 9: Der Problem- und der Lösungsraum

Hier können wir gut erkennen, dass sich die Konzepte dieses Modells über **fast alle Phasen** des *Domain Engineering* erstrecken.

Variabilität wird in allen generischen Artefakten als Variationspunkte realisiert. Nach [CE00] ermöglichen Variationspunkte alternative Implementierungen für funktionale und nicht-funktionale Features. Die Zusammenhänge und Abhängigkeiten zwischen den Variationspunkten sollten explizit erfasst und angemessen organisiert werden, um die Vorteile der Nachvollziehbarkeit auszuschöpfen (**Prozesse**).

Idealerweise sollte die Nachvollziehbarkeit ein Bindeglied sein, das die Zusammenhänge und Abhängigkeiten zwischen zwei Artefakten beschreibt, die in verschiedenen Phasen des Softwareengineering

entwickelt wurden [SPR04]. Diese Art von Bindeglied würde ein (1:1)-Mapping voraussetzen, d. h. *scattering* und *tangling* würden so vermieden werden.
Eine (n: n)-Nachvollziehbarkeit zwischen dem Problem- und Lösungsraum und die sich daraus ergebenden Zusammenhänge werden als scattering und tangling beschrieben.
In [TOHS99] wie folgt definiert:

Scattering: Unter Code *scattering* versteht man, dass sich eine Anforderung (Concern) auf mehrere verschiedene Module und Entwürfe auswirkt.

Tangling: Unter *tangling* versteht man, dass Code, der mehrere Anforderungen (Concerns) betrifft, in einem Modul vermischt wird.

Das Modell
Die Autoren stellen das konzeptionelle Modell für Variabilitätsmanagement vor (Abbildung 3), dass das oben beschriebene (1:1)-Mapping zwischen Variabilität erfüllen soll. D. h. dieses Modell (eingefügt als dritte Dimension) ermöglicht letztlich ein geeignetes Mapping zwischen allen Variationspunkten des zweidimensionalen Raums (Entwicklungsphasen und Abstraktionsebenen).

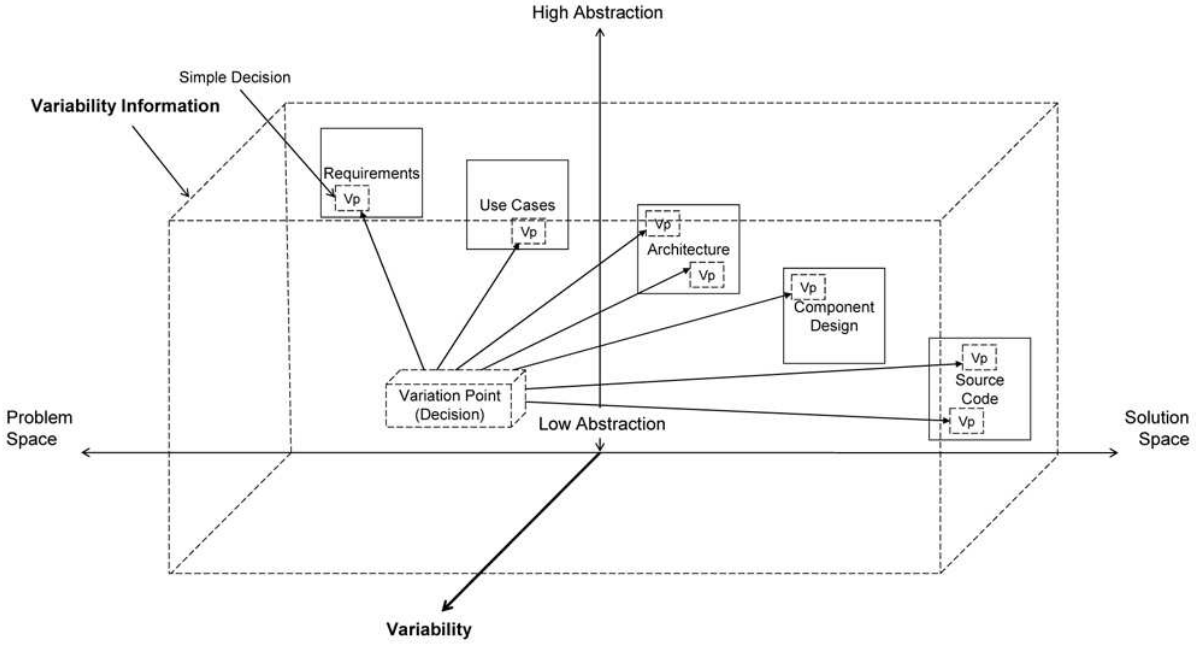

Abbildung 10: Das konzeptionelle Modell für Nachvollziehbarkeit

Nachdem sie das Modell detailliert erklärt haben, wird im folgenden Abschnitt die Vorgehensweise des Feature-Modells, das Nachvollziehbarkeit unterstützen soll, anhand eines Beispiels kurz erklärt und analysiert. In einem weiteren Abschnitt werden diese Erkenntnisse mit denen des oben eingeführten Variabilitätsmodells bezüglich Nachvollziehbarkeit verglichen (*Tabelle 5*).

Die Arbeit in diesem Paper bezieht sich, wie in *Abbildung 8* zu sehen, auf fast alle Phasen der Software Produktlinie und die damit verbundenen **Prozesse** (*Abbildung 10*). Da es sehr viele Variationspunkte in einer Software Produktlinie geben kann, ist es in diesem Modell wichtig, die Nachvollziehbarkeit ihrer Zusammenhänge und Abhängigkeiten immer wieder zu überprüfen. Deswegen beziehe ich hier die Testphase mit ein und klassifiziere das Paper somit in **alle Phasen** des *Domain Engineering*.

	Feature Model Conceptual	Variability Model
Consitstency	-	+
Scalability	-	+
Traceability	-	+
Visualization	+	+

Tabelle 5: Vergleich zwischen dem Feature-Modell und dem konzeptionellen Variabilitätsmodell

In dem vorgestellten Modell sollen über alle Phasen der Software Produktlinienentwicklung hinweg Variabilität und Variationspunkte aufgenommen, modelliert und für alle Beteiligten nachvollziehbar gemacht werden (**Prozesse, alle Stakeholder**).

4.3 Veranschaulichung der Klassifizierungen anhand einer Webseite

Dieser Arbeit ist eine Webseite beigefügt. Um diese zu öffnen klicken Sie bitte im Ordner „Webseite“ auf den Ordner „Hauptseite“ und dann auf index.html.
Bitte verwenden Sie bei der Ansicht der Webseite den Mozilla Firefox Browser in der Version 6.0.2. Ich werde im Ordner diese Version zum downloaden bereitstellen.

An dieser Stelle möchte ich noch darauf aufmerksam machen, dass die Webseite eine beispielhafte Darstellung und auf jeden Fall noch ausbaufähig ist. Darauf werde ich im nächsten Abschnitt 4.4 eingehen. Hier geht es mir in erster Linie um die bessere Veranschaulichung meiner Arbeit.

In diesem Abschnitt werde ich den Aufbau der Webseite kurz erläutern und auf die Darstellung der Koordinatensysteme und damit der Klassifizierungen eingehen.

Abbildung 11: Side bar der Webseite

Auf der **linken Sidebar** gibt es unterschiedliche Links (*Abbildung 11*). Diese beinhalten Teile die bereits in den vorherigen Abschnitten formuliert wurden. Unter dem Link Klassifizierung ist dieselbe Tabelle anzufinden wie im Abschnitt 4.1 - Tabelle 4.
In dieser Tabelle kann durch das anklicken der Titel ein PDF runtergeladen werden, dass zusätzlich zu Titel und Autor eine kurze Zusammenfassung, die Klassifizierung und eine kurze Begründung zur Klassifizierung, sowie eine Verlinkung auf die Quellenangeben enthält.

Die in der Webseite vorkommenden Texte sind teilweise mit Quellenverweisen gekennzeichnet.

Die *Abbildung 12* veranschaulicht die **Navigationsleiste**. Der interne Link „SPL“ verweist auf die Anfangsseite in der der Existenzgrund der Webseite und die Methodik zur Klassifizierung beschrieben werden. Bei den internen Links „Domain Engineering“, “Application Engineering“ und „Organisation“ poppen jeweils in einem neuen Fenster das **zugehörige Koordinatensystem** auf.

Abbildung 12: Navigationsleiste der Webseite

Abbildung 13 zeigt das dreidimensionalen Koordinatensystem „Domain Engineering“ mit folgenden drei Koordinatenachsen:

- **x-Koordinate:** Die Entwicklungsphasen einer Software-Produktlinie (SPL-Phasen):
  Anforderungen, Design, Realisierung, Validierung Domain Engineering umfasst dabei alle SPL-Phasen in diesem Bereich.
- **y-Koordinate:** Die an der SPL-Entwicklung beteiligten Stakeholder:
  Entwickler, Architekt, Projektmanager, Produktmanager, Tester, Kunde Stakeholder umfasst dabei alle gerade aufgezählten Stakeholder.
- **z-Koordinate:** Die Artefakte einer SPL:
  Anforderungen, Architektur, Komponenten, Testen, Prozesse.

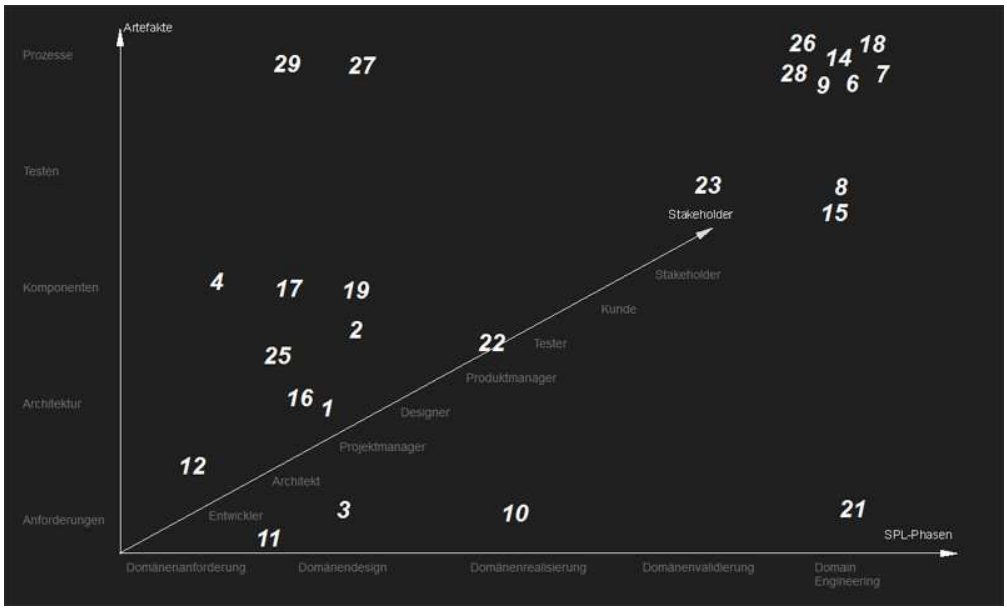

Abbildung 13: Koordinatensystem „Domain Engineering“)

Dieselbe Grafik und Beschreibung von gerade eben gilt auch für die Koordinatensysteme „Application Engineering“ und „Organisation“. Die Definitionen und Erklärungen zu den einzelnen Begriffen werden

in Abschnitt 3 abgehandelt.

In *Abbildung 14* ist ein Teilausschnitt des Koordinatensystems „Domain Engineering“ zu sehen. Wenn sich diese Seite öffnet sind zunächst alle Zuordnungsbereiche grau (siehe *Abbildung 13*). Sobald der Cursor über einer der Zahlen steht (also auf eins der Paper zeigt), erscheinen die von mir gewählten Klassifizierungen in orange. Zusätzlich poppt ein Tooltip mit Titel und Autor des jeweiligen Papers auf. Die einzelnen Nummern können auch direkt angeklickt werden, dadurch öffnet sich in einem neuen Browserfenster ein PDF, das zusätzlich zu Titel und Autor eine kurze Zusammenfassung, die Klassifizierung und eine kurze Begründung zur Klassifizierung beinhaltet. Das eben beschriebene gilt auch wieder für die beiden anderen Koordinatensysteme.

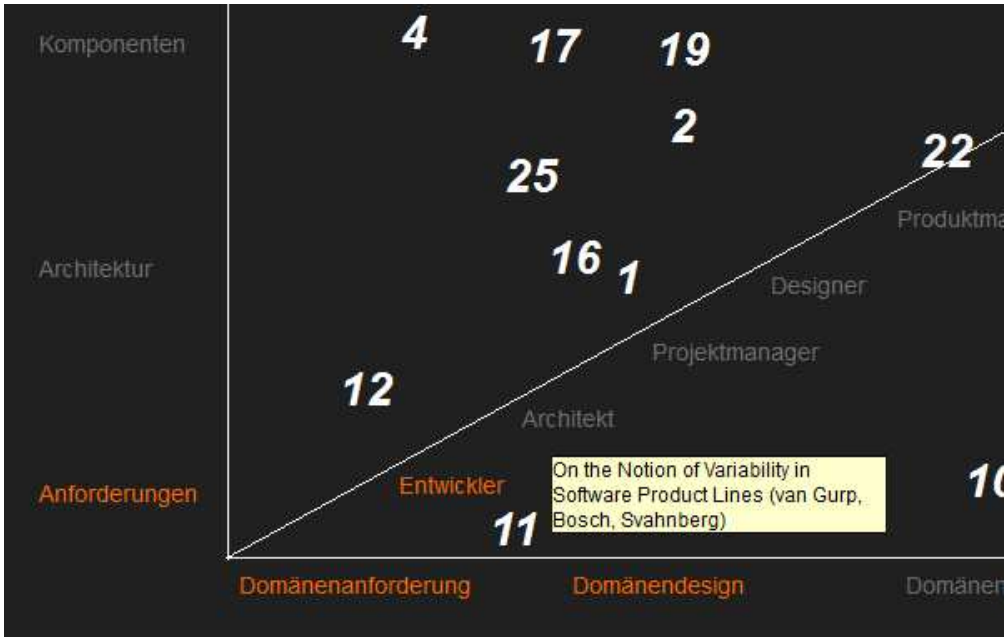

Abbildung 14: Teilausschnitt des Koordinatensystems „Domain Engineering“, wenn der Cursor über der Nr.11 steht

Textlich ist die Webseite nichts Neues zu meiner Arbeit aber grafisch kann ich die Klassifizierungen viel besser verdeutlichen. Inhaltlich interessant wird die Webseite vor allem dann, wenn sie (wie im nächsten Abschnitt beschrieben) erweitert werden würde.

4.4 Ausbaumöglichkeiten der Webseite

So wie die Webseite jetzt aufgebaut ist, kann sie als Informationsseite über Software-Produktlinienentwicklung dienen und sie stellt die Klassifizierung von 30 ausgewählten Papern grafisch gut dar. Zusätzlich stellt sie eine Übersicht über bestehende Forschungsarbeiten dar und ermöglicht es dem Anwender die für ihn wichtigen Themengebiete direkt anzuwählen.

Würde man nun diese Webseite über die 30 Paper hinaus anbieten, könnten Anwender (SPL-Unternehmen im F&E-Bereich oder andere Wissenschaftler) Forschungsarbeiten im Bereich des Domain Engineering, Application Engineering oder der Organisation nur das Koordinatensystem „Domain Engineering“ anzuklicken und sich aus der Übersicht das passende Paper raussuchen.

Letztendlich kann diese Übersicht auch sehr gut dazu dienen, um zu besser zu veranschaulichen in welchen Gebieten schon verstärkt geforscht wurde und in welchen noch ein Forschungsbedarf besteht. Um dazu eine empirisch Aussage machen zu können müssten weitaus mehr Paper oder Forschungsarbeiten klassifiziert und dargestellt werden.

Um diese Punkte auf der Webseite anbieten zu können müssen folgende Fragen beantwortet werden:

1. Wie bewege ich Forscher dazu ihre Arbeiten auf der Webseite zu repräsentieren?
2. Wie „fülle“ ich die Webseite mit Inhalt, also Paper, Klassifizierung dieses Papers und alles was dazu gehört auf, ohne dabei zu viel Zeit zu verlieren und einen zu großen Aufwand zu betreiben?

Die erste Frage ist nicht so leicht zu beantworten. Um die Webseite bekannt genug zu machen, müsste diese sämtlichen Kriterien der Suchmaschinenoptimierung entsprechen. So wäre die Seite schnell über Google & Co. zu finden. Eine bessere und auch effektivere Alternative wären die altbekannten „Beziehungen“. Dazu braucht es zusätzlich zu den Kontakten auch „Überzeugungskraft“. Das wiederum kann nur durch Qualität der Webseite und durch hohe Sicherheitsansprüche gewährleistet werden.
Hier die wichtigsten Qualitätskriterien:

- Die Seite muss technisch einwandfrei funktionieren
- Sie muss visuell so ansprechend sein, so dass ein hoher Wiedererkennungswert gewährleistet ist
- Sie sollte leicht und schnell zu bedienen sein

Mit Sicherheit ist vor allem gemeint, dass persönliche Daten sehr gut gesichert sind und nicht an Dritte weitergegeben werden. Denn für das hochladen, beschreiben und klassifizieren ihrer Arbeiten durch ein Formular (wie unten beschrieben) sollten die Forscher einen Account haben. Das ist nicht nur wichtig, um die Forschungsarbeiten intern besser zuordnen zu können, sonder auch wichtig um Zeit zu sparen.
Die Zeit wird gespart, indem im Hintergrund z. B. in der Datenbank die Arbeiten nach den Forschern eingeordnet werden. Haben diese Autoren bzw. Forscher schon mehrere Arbeiten hochgeladen und waren diese auch immer relevant und von ihnen angemessen klassifiziert, könnte das in der Datenbank markiert werden und diesen Autoren ein Status vergeben werden. Steht der Status z. B. auf „sehr gut“ muss nicht jede zweite in der Zukunft hochgeladene Arbeit gesichtet werden.

Die zweite Frage kann vor allem durch technische Umsetzung beantwortet werden. Jede zur Verfügung gestellte Arbeit durch einen Zuständigen zu klassifizieren und in die Webseite einfügen zu lassen würde sehr viel Zeit und ab einer bestimmten Anzahl an Forschungsarbeiten auch Geld kosten (Da z. B. mehrere Personen diese Arbeiten übernehmen müssten). Gestaltet man die Webseite allerdings so, dass der Autor seine Arbeit anhand eines Formulars klassifizieren und den Inhalt zusammenfassen kann, auf Wunsch auch die ganze Arbeit hochladen kann, dann würde man die oben genannten Problematiken umgehen können.
Wie könnte dieses Formular aussehen?
Das Formular in *Abbildung 15* ist bis auf ein paar Punkte selbsterklärend. Die anderen Punkte sind wie folgt zu erklären: Im SPL-Bereich kann zwischen „Application Engineering“, „Domain Engineering“, „Organisation“ und „Alle Bereiche“ ausgewählt werden, d. h. hier beginnt die erste Unterteilung der Arbeit. Bei der Wahl „Alle Bereiche“ ist die Arbeit in alle drei Bereiche einzuordnen.
Darunter kann dann die Feinauswahl durchgeführt werden, indem die entsprechenden Bereiche die im Koordinatensystem zur Verfügung stehen angeklickt werden.

Ganz unten im Formular kann der Forscher oder die Forschungsgruppe eine PDF-Datei hochladen. Diese kann dann entweder zur öffentlichen Ansicht auf der Webseite oder aber nur zur Ansicht für den Zuständigen der Webseite zur Verfügung gestellt werden. Gibt der Autor die Forschungsarbeit „frei“, so könnte diese von unabhängigen Forschern oder ähnlichen Personen die auch auf dieser Seite angemeldet sind durchgelesen und mit den Angaben verglichen werden. Befinden diese Forscher die Arbeit als relevant und korrekt klassifiziert, könnte durch ihr O.K. die Arbeit zum „Erscheinen“ auf der Webseite freigegeben werden. Dazu würden diese ausgewählten Forscher oder Personen z. B. einen ähnlichen Status wie ein Administrator brauchen.

Durch dieses Formular kann viel Zeit gespart werden. Natürlich müsste immer mal wieder ein Zuständiger ein oder zwei von diesen Arbeiten durchlesen und mit den angegebenen Klassifikationen vergleichen, um Qualität zu gewährleisten.
Die Webseite müsste dementsprechend umgebaut werden. Eine Datenbank mit den einzelnen Papers wäre sinnvoll. Auch der Code müsste angepasst werden, um die Eingaben im Formular so umzusetzen,

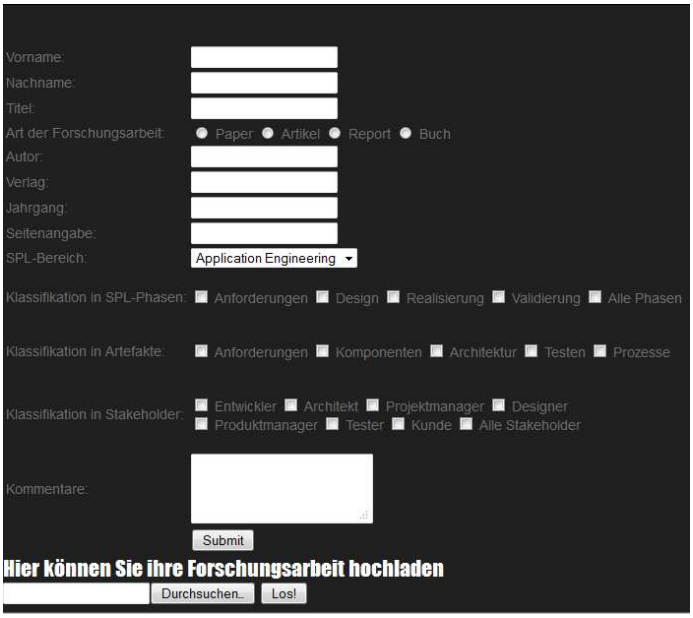

Abbildung 15: Formular zum beschreiben, klassifizieren und hochladen der Forschungsarbeit

dass die Arbeiten automatisch nummeriert und im entsprechenden Koordinatensystem eingefügt.

Das in diesem Abschnitt beschriebene wäre ein guter Anfang. Das „große“ Ziel könnte beispielsweise sein, dass diese Webseite ein Portal für alle Unternehmen, Forscher, Entwickler, Designer, etc., nicht nur ein Informations- sondern auch ein Austauschportal für alle Beteiligten untereinander wird. D. h. neue Forschungsarbeiten im Bereich SPL werden hier hochgeladen, für die gewünschten Personen sichtbar gemacht und von diesen studiert und inhaltlich beurteilt werden.

A Klassifizierung aller Paper

A.1 Using Service Utilization Metrics to Assess the Structure of Product Line Architectures

André van der Hoek, Ebru Dincel, Nenad Medvedovic [vdHDM03]

Zusammenfassung:
Metrics have long been used to measure and evaluate software products and processes. Many metrics have been developed that have lead to different degrees of success. Software architecture is a discipline in which few metrics have been applied, a surprising fact given the critical role of software architecture in software development. Software product line architectures represent one area of software architecture in which we believe metrics can be of especially great use. The critical importance of the structure defined by a product line architecture requires that its properties be meaningfully assessed and that informed architectural decisions be made to guide its evolution. To begin addressing this issue, we have developed a class of closely related metrics that specifically target product line architectures. The metrics are based on the concept of service utilization and explicitly take into account the context in which individual architectural elements are placed. In this paper, we define the metrics, illustrate their use, and evaluate their strengths and weaknesses through their application on three example product line architectures.

Klassifizierung:
Prozessphase:
Domänen Design
Stakeholder:
Architekt
Artefakt:
Architektur (Produktlinienarchitektur)

Kurze Begründung:
Die Autoren stellen Metriken vor, die externe und interne Eigenschaften eines Softwaresystems messen und die Qualität des Aufbaus einer Produktlinienarchitektur bewerten. Zusätzlich können diese Metriken die Variabilität auf den Stufen der Produktlinienarchitekturen analysieren.
Diese Aspekte können von Domänenarchitekten gemessen und in der Produktlinienarchitektur umgesetzt werden. Das geschieht vor allem in den Phasen des Domänendesigns (unter anderem: Aufbau, Qualität, Variabilität der PLA).
Folgende Metriken werden vorgestellt und diskutiert:

- **Provided Service Utilization (PSU) und Required Service Utilization (RSU)**
  Die kontextabhängigen PSU- und RSU-Werte sind nützlich, um ausgewählte Teile bzw. Komponenten der Produktlinienarchitektur zu messen. Diese Werte erfassen die Aufgabe jeder einzelnen Komponente in der Produktlinienarchitektur.
- **Componend Provided Service Utilization (CPU) und Component Required Service Utilization (CRSU)**
  Um jede einzelne Komponente bewerten zu können, ist es sinnvoll, auch die Produktlinienarchitektur als ein Ganzes zu analysieren. Die CPSU und CRSU sollen die internen Zusammenhänge der Architektur bewerten.

Link zum Paper:
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.2171&rep=rep1&type=pdf`

A.2 Maturity and Evolution in Software Product Lines: Approaches, Artefacts and Organization

Jan Bosch [Bos02]

Zusammenfassung:
Software product lines have received considerable adoption in the software industry and prove to be a very successful approach to intra-organizational software reuse. Existing literature, however, often presents only a single approach towards adopting and evolving a software product line. In this paper, we present an overview of different approaches to architecture-centric, intraorganizational reuse of software artefacts. We relate these to maturity levels for product line artefacts and organizational models.

Klassifizierung:
Prozessphase:
Domänendesign
Applikationsdesign
Stakeholder:
Architekt (Domänenarchitekt)
Artefakt:
Architektur
Komponenten

Kurze Begründung:
Jan Bosch definiert in diesem Paper sogenannte „Maturity Levels“. Das sind einzelne Stufen, die den Aufbau und den Design der Architektur sowie die Wiederverwendung einzelner Komponenten beschreiben. Dabei liegt das Hauptaugenmerk auf Variabilität, Gemeinsamkeitsgrad und die Rolle der genannten Artefakte, um effizient Software-Produktlinien und deren Produkte entwickeln zu können. Er bezieht sich in den genannten Punkten sowohl auf *Domain Engineering* als auch auf das *Application Engineering*. Wobei sein Hauptaugenmerk dem Design der Produktlinienarchitektur, dem Design der Komponenten sowie die organisatorische Gestaltung dieser Aspekte und deren Beziehungen untereinander gilt.
Der Aspekt der organisatorischen Planung und Durchführung einer Produktlinienarchitektur wird kurz umrissen. Die Organisationsmodelle werden aufgelistet und kurz beschrieben und für die Vertiefung in die Thematik verweist Jan Bosch auf eine andere Arbeit von ihm, weswegen ich hier die Klassifizierung in eine Organisationsphase unterlasse.

Link zum Paper:
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.3163&rep=rep1&type=pdf`

A.3 Tracking Degradation in Software Product Lines through Measurement of Design Rule Violations

Enrico Johansson and Martin Höst [JH02]

Zusammenfassung:
In order to increase reuse, a number of product versions may be developed based on the same software platform. The platform must, however, be managed and updated according to new requirements if it should be reusable in a series of releases. This means that the platform is constantly changed during its lifecycle, and changes can result in degradation of the platform. In this paper, a measurement approach is proposed as a means of tracking the degradation of a software platform and consequently in the product line. The tracking approach is evaluated in a case study where it is applied to a series of different releases of a product. The result of the case study indicates that the presented approach is feasible.

Klassifizierung:
Prozessphase:
Domänendesign
Stakeholder:
Architekt (Domänenarchitekt)
Artefakt:
Anforderungen

Kurze Begründung:
Die Autoren bestimmen und bewerten ein Maß, das die Funktionsminderung einer SPL verfolgt, die durch das Verletzen der Design-Richtlinien auftreten kann. D. h. das Maß erkennt wann eine Verletzung der Regeln beginnt und wann diese zur Schwächung oder gar zur Zerstörung der Produktlinie führt.

Die Autoren definieren hier die Design-Richtlinien wie folgt:
„Design Richtlinien beschreiben den Nutzen und das Verhalten von Komponenten, Mechanismen und Interfaces. Sind die Richtlinien bereits auf die Komponenten, Mechanismen, Interfaces und deren Wechselbeziehungen anwendbar, dann müssen sie auch strikt eingehalten werden. Gibt es für bestimmte Aspekte noch keine Richtlinien sind diese auch nicht anwendbar.“
Die Autoren diskutieren das Einhalten und Verletzen dieser Design-Richtlinien und die Folgen für die SPL. Diese Design-Richtlinien stellen beim Gestalten und Entwickeln der Produktlinienarchitektur einen essentiellen Teil dar. Letztendlich diskutieren sie die Frage, welchen Anforderungen an die Komponenten und die Architektur (bzgl. Neugestaltung aber auch Wiederverwendung) diese Richtlinien gerecht werden müssen. Das ist in erster Linie die Aufgabe des Architekten.

Link zum Paper:
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.112.6108&rep=rep1&type=pdf#page=59`

A.4 Issues Concerning Variability in Software Product Lines

Mikael Svahnberg and Jan Bosch [SB00]

Zusammenfassung:
Product-line architectures, i.e. a software architecture and component set shared by a family of products, represents a promising approach to achieving reuse of software. Several companies are initiating or have recently adopted a product-line architecture. However, little experience is available with respect to the evolution of the products, the software components and the software architecture. Due to the higher level of interdependency between the various software assets, software evolution is a more complex process. In this paper we discuss issues regarding variability that may help or cause problems when designing solutions for managing variability.

Klassifizierung:
Prozessphase:
Domänenanforderungen
Applikationsanforderungen
Stakeholder:
Architekt (Domänenarchitekt),
Entwickler
Artefakt:
Anforderungen
Komponenten

Kurze Begründung:
In diesem Paper werden Charakteristiken diskutiert die eine SPL von einem einzelnen Produkt unterscheiden. Die Gewährleistung von Variabilität innerhalb einer SPL und bezüglich der Komponenten spielt dabei eine sehr große Rolle. Dazu werden unterschiedliche Techniken vorgestellt und diskutiert.
Es wird vor allem diskutiert, welchen Anforderungen an die Komponenten entsprochen werden muss, um eine effektive Variabilität gewährleisten zu können. Die Ergebnisse dazu haben sie vor allem aus früheren Studien, die sie im Paper kurz vorstellen. Desweiteren wird die Anwendbarkeit der vorgestellten Techniken (die einzelnen Variabilitätsebenen) diskutiert und welchen Anforderungen diese Techniken an die erwartete Variabilität in den einzelnen Ebenen genügen müssen, um aus der SPL ein Produkt zu entwickeln.
Diese im Paper besprochenen Anforderungen können für die SPL während des *Domain Engineering* und für die Produkte der SPL während des *Application Engineering* von Architekten und/oder Entwicklern bearbeitet und umgesetzt werden.

Link zum Paper:
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.22.1727&rep=rep1&type=pdf`

A.5 Product Instantiation in Software Product Lines: A Case Study

Jan Bosch and Mattias Högström [BH01]

Zusammenfassung:
Product instantiation is one of the less frequently studied activities in the domain of software product lines. In this paper, we present the results of a case study at Axis Communication AB on product instantiation in an industrial product line, i.e. five problems and three issues. The problems are concerned the insufficiency of functional commonality, features spanning multiple components, the exclusion of unwanted features, the evolution of product line components and the handling of initialization code. The issues discuss architectural compliance versus product instantiation effort, quick-fixes versus properly engineered extensions and component instantiation support versus product instantiation effort. The identified problems and issues are based on the case study, but have been generalized to apply to a wider context.

Klassifizierung:
Prozessphase:
Applikationsanwendung
Applikationsdesign
Stakeholder:
Architekt (Domänenarchitekt),
Entwickler
Artefakt:
Anforderungen
Komponenten

Kurze Begründung:
Die Autoren diskutieren Probleme und Ursachen, die bei der Instanziierung von Produkten in einer bestehenden SPL vorkommen können. Die Wiederverwendung von Komponenten und Features führt zu unterschiedlichen Problemen, die wiederum unterschiedliche Ursachen haben können. Dabei stehen die Variabilität und der Gemeinsamkeitsgrad zwischen den Produkten und deren wiederverwendeten Komponenten im Vordergrund.

Im Abschnitt der Produktinstanziierung werden die Softwarearchitektur und die Prozesse beschrieben und Variabilitätsmechanismen diskutiert. Es wird intensiv auf die Probleme anhand von Beispielen eingegangen und was deren Ursachen sind.

Im nächsten Abschnitt werden dann die aus den Problemen und Ursachen resultierenden Fragen diskutiert. Diese Überlegungen können in der Phase der Applikationsanwendung und des Applikationsdesigns von Architekten und Entwickler angewandt werden. Sie entscheiden, welche Anforderungen an die wiederverwendeten Komponenten und an die Variabilität bestehen, so dass weitere Produkte aus dieser Produktfamilie entwickelt werden können.

Link zum Paper:
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.27.766&rep=rep1&type=pdf`

A.6 Software Product Family Evaluation

Frank van der Linden, Jan Bosch, Erik Kamsties, Kari Känsälä, Lech Krzanik and Henk Obbink [vdLBK+04]

Zusammenfassung:
This paper proposes a 4-dimensional software product family engineering evaluation model. The 4 dimensions relate to the software engineering concerns of business, architecture, organisation and process. The evaluation model is meant to be used within organisations to determine the status of their own software family engineering. Evaluation results may be used for benchmarking or improvement plans.

Klassifizierung:
Prozessphase:
Domain Engineering
Stakeholder:
Entwickler,
Architekten,
Produktmanager,
Projektmanager
Artefakt:
Anforderungen
Prozesse

Kurze Begründung:
In diesem Paper wird ein 4-dimensionales Model vorgestellt, das den Status der SPL bestimmen und bewerten soll. Die Ergebnisse können dann genutzt werden, um Kosten zu sparen oder Verbesserungspläne zu erstellen. Die vier Dimensionen beeinflussen sich gegenseitig, d. h. bei Verbesserungen an einer kann es zu Veränderungen (nicht zwingend Verbesserungen) bei der anderen Dimension kommen.

Die Autoren geben selber an, dass dieses Bewertungsmodell innerhalb von Unternehmen genutzt werden kann, um den Status ihrer eigenen SPL-Entwicklung zu bestimmen und die daraus resultierenden Ergebnisse z. B. zum Leistungsvergleich oder für Verbesserungsvorschläge zu nutzen.

Bei den vier Dimensionen handelt es sich um das Business (wie können Gewinne erwirtschaftet werden), die Architektur (technische Mittel, um die Software zu entwickeln), Prozesse (Funktionen, Verantwortungsbereiche und Beziehungen innerhalb der Softwareentwicklung) und die Organisation (Wiedergabe des Istzustands von Aufgaben und Verantwortungsbereichen der Organisationsstrukturen). Also erstreckt sich das Modell über alle Produktlinienphasen und da das Ziel diese Modells die Entwicklung einer SPL ist über alle Phasen des *Domain Engineering*. Dabei spielen die Prozesse, also die Interdependenzen dieser Phasen bezüglich der Softwareentwicklung, eine große Rolle. An der Planung wären die Projekt- und Produktmanager direkt beteiligt und an der Umsetzung die Entwickler und Architekten. Dabei müssen viele Anforderungskriterien überdacht und evtl. neu gestaltet werden.

Link zum Paper:
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.87.4972&rep=rep1&type=pdf`

A.7 Variability Issues in Software Product Lines

Jan Bosch, Gert Florijn, Danny Greefhorst, Juha Kuusela, J. Henk Obbink, and Klaus Pohl [BFG+02]

Zusammenfassung:
Software product lines (or system families) have achieved considerable adoption by the software industry. A software product line captures the commonalities between a set of products while providing for the differences. Differences are managed by delaying design decisions, thereby introducing variation points. The whole of variation points is typically referred to as the variability of the software product line. Variability management is, however, not a trivial activity and several issues exist, both in general as well as specific to individual phases in the lifecycle. This paper identifies and describes several variability issues based on practical experiences and theoretical understanding of the problem domain.

Klassifizierung:
Prozessphase:
Domain Engineering
Application Engineering
Stakeholder:
Entwickler,
Architekten
Artefakt:
Anforderungen,
Prozesse

Kurze Begründung:
Die Autoren diskutieren in diesem Paper das Thema Variabilitätsmanagement und gehen dabei auf alle Phasen der Software Produktlinienentwicklung ein, sowohl im *Domain Engineering* als auch im *Application Engineering*. Sie behaupten und diskutieren, dass Variabilität innerhalb einer SPL alle Phasen dieser SPL und der Produktentwicklung betrifft. Dabei werden vor allem die Anforderungen an die einzelnen Phasen besprochen und wie Variabilität in und zwischen den Phasen über die gesamte Entwicklungsphase hinweg gewährleistet werden kann. Zusätzlich wird diskutiert, wie dies von Entwicklern und Architekten angewandt und umgesetzt werden kann. Da hier auch unter anderem Zusammenhänge und Verantwortlichkeiten während der Softwareentwicklung hinsichtlich Variabilität besprochen werden, ist eine Einordnung unter Prozesse durchaus sinnvoll.

Link zum Paper:
`http://www-sst.informatik.tu-cottbus.de/~db/doc/People/LNCS/papers/22900013.pdf`

A.8 Software-Reliability-Engineered Testing

John D. Musa [Mus96]

Zusammenfassung:
Software testing often results in delays to market and high cost without assuring product reliability. Software reliability engineered testing (SRET), an AT&T best practice, carefully engineers testing to overcome these weaknesses. The article describes SRET in the context of an actual project at AT&T, which is called Fone Follower. The author selected this example because of its simplicity; it in no way implies that SRET is limited to telecommunications systems. SRET is based on the AT&T Best Current Practice of Software Reliability Engineering, approved in May 1991. Qualification as an AT&T best current practice requires use on typically eight to 10 projects with documented large benefit/cost ratios, as well as a probing review by two boards of high level managers. Some 70 project managers also reviewed this particular practice. Standards for approval as a best current practice are high; only five of 30 proposed best current practices were approved in 1991.

Klassifizierung:
Prozessphase:
Domain Engineering
Application Engineering
Stakeholder:
Tester
Artefakt:
Testen

Kurze Begründung:
In diesem Paper wird die Software SRET vorgestellt, mit der Tests bei unterschiedlichen Systemen durchgeführt werden können. Bei Softwareproduktlinien erstrecken sich diese Tests über den gesamten Lebenszyklus, d. h. sowohl im *Domain Engineering*, als auch im *Application Engineering*.

Der Name SRET steht übersetzt für das Testen von Software auf deren Zuverlässigkeit bzw. Fehleranfälligkeit. Zwei Testvarianten werden vorgenommen, einmal das *developement testing* (Fehlerfindung und -behebung) und das *certification testing* (Akzeptieren oder Verwerfen der Software). Das *development testing* geht dem *certification testing* voraus, das wiederum als Generalprobe für die Kundenzufriedenheitstests dient. In diesen beiden Phasen spielen zusätzlich Entwickler, Architekten und Kunden (Kundenzufriedenheit) eine Rolle, allerdings sind die Tester die Hauptanwender, die mit den Entwicklern und Architekten zusammenarbeiten müssen und die Kundenzufriedenheit direkt gewährleisten müssen. Also unterliegt den Testern die Hauptaufgabe des Testens.

Link zum Paper:
`http://itu.dk/people/nicki/test/Musa_Reliability_q.PDF`

A.9 Organizing for Software Product Lines

Jan Bosch [Bos00b]

Zusammenfassung:
Software product lines have received increasing amounts of attention within the software engineering community, especially from industry. Most authors focus on the technical and process aspects and assume an organizational model consisting of a domain engineering unit and several application engineering units. In our cooperation with several software development organizations applying software product line principles, we have identified several other organizational models that are employed. This article presents a number of organizational models, organized in four main approaches, i.e. development department, business units, domain engineering units and hierarchical domain engineering units. For each approach, its characteristics, applicability and advantages and disadvantages are discussed, as well as an example.
Klassifizierung:
Prozessphase:
Domain Engineering
Organisation
Stakeholder:
Projektmanager,
Architekten,
Entwickler
Artefakt:
Prozesse

Kurze Begründung:
Jan Bosch stellt hier Organisationsmodelle vor, die durch vier Hauptansätze organisiert werden:

- Die Entwicklungsabteilung arbeitet an *Domain Engineering* Projekten, um wiederverwendbare Artefakte zu schaffen und weiterzuentwickeln, die letztlich die Produktlinie ausmachen.
- Die Geschäftseinheiten, die Projekte im *Domain Engineering* anregen, in denen entweder gemeinsame neue Artefakte geschaffen oder bestehende geändert werden sollen.
- Einheiten des *Domain Engineering* sind verantwortlich für die Schaffung, das Design und die Entwicklung wiederverwendbarer Artefakte, d. h. für die Softwarearchitektur und die Komponenten, die den wiederverwendbaren Teil für die SPL bilden.
- Hierarchische Einheiten des *Domain Engineering*, die benötigt werden, wenn hierarchische Strukturen innerhalb der Domäneneinheiten notwendig sind.
- Und weitere Faktoren, die das Organisationsmodell beeinflussen.

Diese genannten Hauptansätze werden über die Phasen des *Domain Engineering* hinweg diskutiert, wobei die Beziehungen, Rollen und Verantwortlichkeiten in und zwischen diesen Phasen besprochen werden (Prozesse). Hierbei spielen Projektmanager, Entwickler und Architekten eine übergeordnete Rolle. Da hier Organisationsmodelle vorgestellt werden, ist eine zusätzliche Einordnung mit den eben genannten Stakeholdern und Artefakten in die gesamten Prozessphasen der Organisation sinnvoll.

Link zum Paper:
`http://www.cs.rug.nl/search/uploads/Publications/bosch2000osp.pdf`

A.10 Managing Variability in Software Product Lines

Jilles van Gurp, Jan Bosch, Mikael Svahnberg [vGBS00]

Zusammenfassung:
Software Product Lines are large systems intended for reuse in concrete products. As such these large systems provide reusable architecture and implementation that the individual products have in common. The differences between the product are left open. We refer to these open spots as variability points. In this article we provide a conceptual framework of terminology for the concept of variability and we discuss how variability can be managed in Software Product Lines.

Klassifizierung:
Prozessphase:
Domänendesign,
Domänenrealisation
Stakeholder:
Architekten,
Entwickler
Artefakt:
Anforderungen

Kurze Begründung:
In diesem Paper werden Definitionen und Konzepte vorgestellt, um eine bessere Kommunikation zwischen den Entwicklern bezüglich Variabilität gewährleisten zu können. Zusätzlich wird eine Methode für das Managen von Variabilität in umfangreichen Systemen vorgestellt, die durch die Definition und Beschreibung von *Features* erklärt wird. Dabei ist die Rede von Abstraktionsstufen: unter anderem Architektur, Beschreibung, Funktionsbeschreibung, Designdokumentation, Quellkode und laufendes System. Die Autoren diskutieren darüber, dass Variabilität in den Abstraktionsstufen auch als eine Veränderung in und/oder zwischen den zusammenhängenden *Feature*-Gruppen gesehen werden kann. Diese Diskussion vertieft sich vor allem in den Phasen des Domänendesigns und der Domänenrealisation in der *Features* geplant, entwickelt und verändert werden, um Variabilität zu gewährleisten. Es werden die dazu nötigen Anforderungen besprochen, die von Entwicklern und Architekten diskutiert und angewendet werden sollen.
Features: Werden hier als eine Art Abstraktion der Anforderungen betrachtet. *Features* zu modellieren ist hier der erste Schritt Anforderungen auszuwerten und zu bestimmen.

Link zum Paper:
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.1087&rep=rep1&type=pdf`

A.11 On the Notion of Variability in Software Product Lines

Jilles van Gurp, Jan Bosch, Mikael Svahnberg [vGBS01]

Zusammenfassung:
In this paper, we discuss the notion of variability. We have experienced that this concept has so far been underdefined. Although, we have observed that variability techniques become increasingly important. A clear indication of this trend is the recent emergence of software product lines. Software product lines are large, industrial software systems intended to specialize into specific software products. Our contribution in this paper is that we provide the reader with a framework of terminology and concepts regarding variability. In addition, we present three recurring patterns of variability. Finally, we suggest a method for managing variability in software product lines.

Klassifizierung:
Prozessphase:
Domänenanforderungen,
Domänendesign
Stakeholder:
Entwickler
Artefakt:
Anforderungen

Kurze Begründung:
Die Autoren stellen Definitionen und Konzepte bezüglich Variabilität vor und diskutieren eine Methode für das Managen von Variabilität. Die Zusammenhänge und wichtige Aspekte werden mit *Features* erklärt und begründet, um vor allem eine frühe Identifikation von Variabilität in einer Software Produktlinie zu gewährleisten.
Features hier: werden als eine Art Abstraktion der Anforderungen betrachtet. Features zu modellieren, ist der erste Schritt Anforderungen auszuwerten und zu bestimmen.
Die Autoren richten ihre Ausarbeitungen größtenteils an Softwareentwickler, die diese Ansätze für die Anforderungen in der Frühen Phase des Designs anwenden und mit anderen Entwicklern diskutieren können, ohne dabei zu sehr ins Detail bezüglich der Implementierung gehen zu müssen.

Link zum Paper:
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.8217&rep=rep1&type=pdf`

A.12 A Comprehensive Product Line Scoping Approach and Its Validation

Klaus Schmid [Sch02]

Zusammenfassung:
Product Line Engineering is a recent approach to software development that specifically aims at exploiting commonalities and systematic variabilities among functionally overlapping systems in terms of large scale reuse. Taking full advantage of this potential requires adequate planning and management of the reuse approach as otherwise huge economic benefits will be missed due to an inappropriate alignment of the reuse infrastructure. Key in product line planning is the scoping activity, which aims at focussing the reuse investment where it pays. Scoping actually happens on several levels in the process: during the domain analysis step (analysis of product line requirements) a focusing needs to happen just like during the decision of what to implement for reuse. The latter decision has also important ramifications for the development of an appropriate reference architecture as it provides the reusability requirements for this step. In this paper, we describe an integrated approach that has been developed, improved, and validated over the last few years. The approach fully covers the scoping activities of domain scoping and reuse infrastructure scoping and was validated in several industrial case studies.

Klassifizierung:
Prozessphase:
Domänenanforderungen
Stakeholder:
Produktmanager,
Entwickler
Artefakt:
Anforderungen,
Architektur

Kurze Begründung:
Klaus Schmid stellt einen neuen Ansatz des *Scopings* vor. *Scoping* findet vor allem während des Planens der Domänenanforderungen statt mit einem Fokus auf Entscheidungen für die Implementierungsphase bezüglich der Wiederverwendbarkeit. Schmids Ansatz (PuLSE-Eco V2.0) deckt zwei Bereiche des *Scoping* ab:

- *Domain Scoping* (Bewerten von Nutzen und Risiken bezüglich systematischer Wiederverwendung in den Relevanten Phasen der Produktlinie)
- *Asset Scoping* (Ermitteln einer wiederverwendbaren Infrastruktur, um die Unternehmensziele so effizient wie möglich zu erreichen) PuLSE-Eco V2.0 basiert auf drei Hauptkomponenten
- *product line mapping* (Die Darstellung auf hoher Ebene von Produktlinie, Domänen, Features und die damit verbundenen Zusammenhänge)
- *domain potential assessment* (Das Bewerten von möglichen Chancen und Risiken bei der Wiederverwendung und die Auswahl der Domänen)
- *reuse infrastructure scoping* (Identifizierung der Artefakte für die Wiederverwendung)

Seine vorgestellten Ansätze und deren Analysen sind eine Hilfestellung für Produktmanager und Entwickler in der Planungsphase einer Software Produktlinie mit einem Hauptaugenmerk auf die Anforderungen und die Architektur.

Link zum Paper:
`http://www.esi.es/Projects/Cafe/pdf/cr-icse-eco2.pdf`

A.13 A Configuration Management Model for Software Product Line

Liguo Yu and Srini Ramaswamy [YR06]

Zusammenfassung:
Software Product Line has proved to be an effective approach to benefit from software reuse. Configuration management, an integral part of any software development activity, takes on a special significance in software product line context. This is due to the special property of software product line, in which the core assets are shared by all products. In this paper, we compare the existing configuration management models and analyze the artifacts that need to be configuration managed in software product line. We then present an evolution-based configuration management model for software product line, in which, the configuration management is divided into two domains, the production domain and the product domain. In this model, the evolution propagation of corrective changes and enhancement changes on different configuration artifacts follow different paths. The advantages and the constraints of this model are also discussed.

Klassifizierung:
Prozessphase:
Application Engineering
Stakeholder:
Produktmanager,
Projektmanager,
Entwickler
Artefakt:
Anforderungen,
Komponenten

Kurze Begründung:
Die Autoren diskutieren bestehende CM-Modelle (*Configuration Management Model*: Entscheidungen wie Artefakte aufgebaut sein sollen, Entwicklungsplanung und die Abwendung von einem Produktlinienverfall).
Das sind Managementmodelle, die organisatorische Regeln aufstellen, um diese am Produktlebenszyklus einer Konfigurationseinheit anzuwenden. Diese Regeln beziehen sich auf das Definieren, Designen, Implementieren und Betreiben eines Produktes innerhalb einer SPL (das entspricht allen Phasen des *Application Engineering*).
Desweiteren stellen sie ein entwicklungsbasiertes CM-Modell vor und vergleichen dieses mit den bestehenden Modellen. Das von ihnen vorgestellte Modell konzentriert sich auf *production domain* (das Managen und Entwickeln von hauptsächlich Komponenten, aber auch Artefakten allgemein) und *product domain* (das Managen der Entwicklung eines Produktes) und die daraus resultierenden Anforderungen, um ein effizient entwickeltes Produkt innerhalb einer SPL zu entwickeln. Dabei sollen vor allem Produkt- und Projektmanager, sowie Entwickler angesprochen werden.

Link zum Paper:
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.144.4005&rep=rep1&type=pdf`

A.14 Analytical and Empirical Evaluation of Software Reuse Metrics

Prem Devanbu, Sakke Karstu, Walcelio Melo and William Thomas [DKMT96]

Zusammenfassung:
How much can be saved by using existing software components when developing new software systems? With the increasing adoption of reuse methods and technologies, this question becomes critical. However, directly tracking the actual cost savings due to reuse is difficult. A worthy goal would be to develop a method of measuring the savings indirectly by analyzing the code for reuse of components. The focus of this paper is to evaluate how well several published software reuse metrics measure the 'time, money and quality' benefits of software reuse. We conduct this evaluation both analytically and empirically. On the analytic front, we introduce some properties that should arguably hold of any measure of 'time, money and quality' benefit due to reuse. We assess several existing software reuse metrics using these properties. Empirically, we constructed a toolset (using GEN++) to gather data on all published reuse metrics from C++ code; then, using some productivity and quality data from 'nearly replicated' student projects at the University of Maryland, we evaluate the relationship the known metrics and the process data. Our empirical study sheds some light the applicability of our different analytic properties, and has raised some practical issues to be addressed as we undertake broader study of reuse metrics in industrial projects.

Klassifizierung:
Prozessphase:
Domain Engineering,
Application Engineering
Stakeholder:
Entwickler
Artefakt:
Prozesse

Kurze Begründung:
In diesem Paper werden indirekte Messmethoden bezüglich des Nutzens von wiederverwendbarer Software beurteilt. Es werden Fünf Metriken analytisch und empirisch bewertet, ob diese nützliche Voraussagen bezüglich Qualität, Zeit und Geld über eine wiederverwendbare Software machen können. Die Autoren kommen zu dem Schluss, dass die unterschiedlichen Messmethoden Voraussagen über unterschiedle Qualitätsmerkmale der SPL machen können. Diese Erkenntnisse können sowohl in den gesamten Phasen des *Domain Engineering* als auch in denen des *Application Engineering* sehr gut von Entwicklern für die Prozesse während der Softwareentwicklung in Bezug auf Wiederverwendung genutzt werden.

Link zum Paper:
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.65&rep=rep1&type=pdf`

A.15 Coverage and Adequacy in Software Product Line Testing

Myra B. Cohen, Matthew B. Dwyer and Jiangfan Shi [CDS06]

Zusammenfassung:
Software product line modeling has received a great deal of attention for its potential in fostering reuse of software artifacts across development phases. Research on the testing phase, has focused on identifying the potential for reuse of test cases across product line instances. While this offers potential reductions in test development effort for a given product line instance, it does not focus on and leverage the fundamental abstraction that is inherent in software product lines - variability. In this paper, we illustrate how rich software product line modeling notations can be mapped onto an underlying relational model that captures variability in the feasible product line instances. This relational model serves as the semantic basis for defining a family of coverage criteria for testing of a product line. These criteria make it possible to accumulate test coverage information for the product line itself over the course of multiple product line instance development efforts. Cumulative coverage, in turn, enables targeted testing efforts for new product line instances. We describe how combinatorial interaction testing methods can be applied to define test configurations that achieve a desired level of coverage and identify challenges to scaling such methods to large, complex software product lines.

Klassifizierung:
Prozessphase:
Domain Engineering
Application Engineering
Stakeholder:
Entwickler,
Tester
Artefakt:
Anforderungen,
Testen

Kurze Begründung:
Die Autoren zeigen, wie vielseitig die Darstellung der SPL-Modelle auf das darunterliegende Relationsmodel in den möglichen Produktinstanzen abgebildet werden kann. Dabei dient das Relationsmodell als semantische Basis für die Definition von Kriterien, um eine Produktlinie testen zu können.
Sie gehen davon aus, dass zunächst Abgrenzungskriterien festgelegt werden müssen. Innerhalb dieser Abgrenzungskriterien soll dann eine SPL entwickelt werden. Getestet wird letztendlich, ob die SPL in allen Phasen sowohl des *Domain Engineering* als auch des *Application Engineering* sich innerhalb dieser Abgrenzungskriterien bewegt und ab wann Abweichungen vorkommen. Sie stellen ein orthogonales Variabilitätsmodell vor, das diesen Punkten entsprechen soll. Im Großen und Ganzen sollen zunächst Entwickler die Anforderungen für eine SPL festlegen, damit die Tester diese im Laufe der Entwicklungsphasen einer SPL testen können.

Link zum Paper:
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.132.1151&rep=rep1&type=pdf`

A.16 Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability

Eduardo Figueiredo, Nelio Cacho, Claudio Sant'Anna, Mario Monteiro, Uira Kulesza, Alessandro Garcia, Sergio Soares, Fabiano Ferrari, Safoora Khan, Fernando Castor Filho and Francisco Dantas [FCS+08]

Zusammenfassung:
Software product lines (SPLs) enable modular, large-scale reuse through a software architecture addressing multiple core and varying features. To reap the benefits of SPLs, their designs need to be stable. Design stability encompasses the sustenance of the product line's modularity properties in the presence of changes to both the core and varying features. It is usually assumed that aspect-oriented programming promotes better modularity and changeability of product lines than conventional variability mechanisms, such as conditional compilation. However, there is no empirical evidence on its efficacy to prolong design stability of SPLs through realistic development scenarios. This paper reports a quantitative study that evolves two SPLs to assess various design stability facets of their aspect-oriented implementations. Our investigation focused upon a multi-perspective analysis of the evolving product lines in terms of modularity, change propagation, and feature dependency. We have identified a number of scenarios which positively or negatively affect the architecture stability of aspectual SPLs.

Klassifizierung:
Prozessphase:
Domänendesign
Stakeholder:
Designer,
Entwickler,
Architekten
Artefakt:
Architektur

Kurze Begründung:
In diesem Paper werden Möglichkeiten diskutiert, Designstabilität zu messen und bestimmte Aspekte in einer SPL bezüglich der Stabilität ihrer Architektur zu identifizieren. Dabei werden die positiven und negativen Einflüsse des *aspect oriented programming* untersucht, vor allem die daraus resultierenden Unterschiede im Design der Kernarchitektur und den variablen Features von Software-Produktlinien. Unter anderem die Analyse von Modularität und Feature-Abhängigkeiten. Die Ergebnisse werden an Hand von einigen Entwicklungsszenarien von zwei heterogenen Software-Produktlinien erforscht. Letztendlich werden dabei die sich kontinuierlich weiterentwickelnden Software-Produktlinien analysiert (Das betrifft das *Domain Engineering*).
In Bezug auf die betreffenden Phasen und angesprochenen Artefakte können sowohl Architekten und Entwickler als auch Designer diese Erkenntnisse antizipieren. Modularität: Aufteilen eines Ganzen in Teile/Module.
Aspect Oriented Programming: Ist ein Programmierparadigma für die objektorientierte Programmierung, um generische Funktionalitäten über mehrere Klassen hinweg zu verwenden. (Wikipedia, August 2011)

Link zum Paper:
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.119.9423&rep=rep1&type=pdf`

A.17 Identifying and Qualifying Reusable Software Components

Gianluigi Caldiera and Victor R. Basili [CB91]

Zusammenfassung:
Identification and qualification of reusable software based on software models and metrics is explored. Software metrics provide a way to automate the extraction of reusable software components from existing systems, reducing the amount of code that experts must analyze. Also, models and metrics permit feedback and improvement to make the extraction process fit a variety of environments. Some case studies are described to validate the experimental approach. They deal with only the identification phase and use a very simple model of a reusable code component, but the results show that automated techniques can reduce the amount of code that a domain expert needs to evaluate to identify reusable parts (IEEE, August 2011)

Klassifizierung:
Prozessphase:
Domänenanforderungen,
Domänendesign,
Applikationsanforderungen,
Applikationsdesign
Stakeholder:
Entwickler
Artefakt:
Komponenten

Kurze Begründung:
In diesem Paper wird eine Projektorganisation vorgestellt die Anforderungen analysieren und sowohl Spezifikationen als auch ein hochwertiges Systemdesign entwickeln soll. Entwickler generieren diese Spezifikationen und designen das System so, dass diese Anforderungen umgesetzt werden (Anforderungen an die Komponenten).
Nachdem sie die Systemkomponenten ausgewählt und identifiziert haben, fordern sie Komponenten von der *Component Factory* an, um diese in das System, das sie designt haben, zu integrieren. Danach folgen die Qualitätskontrolle und die Freigabe.
Die Komponenten werden aus einem bereits bestehenden Pool ausgesucht, damit diese dann wiederverwendet und gegebenenfalls angepasst werden können. Das Ziel ist auf diese Weise eine SPL zu designen oder aber auch ein neues Produkt innerhalb der SPL generieren zu können (*Domain Engineering* und *Application Engineering*). Dabei müssen alle Anforderungen an die SPL (hier vor allem an die Komponenten) analysiert und berücksichtigt werden (Domänendesign und Applikationsdesign).
Component Factory: Diese Abteilung besitzt einen „Katalog“ an Komponenten die sie auf Anfrage freigeben. Dabei müssen die Komponenten den angefragten Anforderungen entsprechen.

Link zum Paper:
`https://www.cs.umd.edu/~basili/publications/journals/J41.pdf`

A.18 Innovation Management for Product Line Engineering Organizations

Günter Böckle [Böc05]

Zusammenfassung:
Active innovation management is performed by companies to create an environment that fosters innovation. In a product line environment, platform and predefined variability restrict innovation because the development artifacts in the platform and the variation are prescribed. An analysis of innovation projects in literature shows that moderate innovations like introducing a new member of a product line yield only a small return on investment. This paper introduces a series of measures that can help to prevent a lock-in of a product line organization with respect to innovation. We take a look at various aspects of innovation - personnel, customer and market, technology and engineering, organization and process. Organizations may pick the best-suited measures for their current situation.

Klassifizierung:
Prozessphase:
Domain Engineering,
Application Engineering,
Organisation
Stakeholder:
Alle am Projekt beteiligten.
Artefakt:
Prozess

Kurze Begründung:
Günter Böckle diskutiert das Problem des *log-in* in einer SPL-Organisation bezüglich Innovation. Unter *log-in* ist hier eine Einschränkung der Innovationsfähigkeit in einer SPL-Umgebung durch die Plattform oder die vordefinierte Variabilität gemeint.
Er diskutiert Aspekte der Innovation, wie z. B. personelle, kunden- und marktspezifische oder technische Aspekte (Organisation).
Diese Innovationen können sowohl während den Entwicklungsphasen der SPL als auch während der Entwicklungsphasen der einzelnen Produkte innerhalb einer SPL entstehen (*Domain Engineering* und *Application Engineering*).
Die Innovationsmöglichkeiten erstrecken sich über alle Phasen der Entwicklung und ihrer Aufgaben, Verantwortlichkeiten und Zusammenhänge untereinander (Prozesse). Dabei kann jeder einen Beitrag leisten, der innerhalb dieser Entwicklungen in der Lage ist, Ideen zu äußern und durchzuführen (Also alle Stakeholder, die am Entwicklungsprozess der SPL und ihrer Produkte beteiligt sind.), vor allem durch *cross-functional teams*.
cross-functional teams : Eine Gruppe von Leuten, die aus unterschiedlichen Bereichen stammen und an einer gemeinsamen Aufgabe arbeiten.

Link zum Paper:
`http://www.springerlink.com/content/7dcl7tbgh9duqxbe/`

A.19 Measuring Component Reliability

John D. McGregor, Judith A. Stafforda and Il-Hyung Cho [MSC03]

Zusammenfassung:
Much of the research on component-based software engineering assumes that each component has a single service. This simplifies analyses but it is a significant departure from actual components. This paper reports on an investigation of the feasibility of using design constructs as a means of treating several methods as a single unit. Grouping the services of a component into a few sets satisfies the goal of simplicity while still providing the designer with a more usable model of component reliability.

Klassifizierung:
Prozessphase:
Domänendesign,
Domänenvalidierung,
Applikationsdesign,
Applikationsvalidierung
Stakeholder:
Entwickler,
Designer
Artefakt:
Komponenten

Kurze Begründung:
In diesem Paper wird untersucht, in welchem Maße ausreichende Informationen über die Funktionsfähigkeit von Komponenten an Interessenten angeboten werden können. Je mehr Informationen die Interessenten über eine Komponente erfahren, desto besser können sie diese für ihre eigenen Zwecke nutzen und desto interessanter wird es diese auch zu kaufen. Die Autoren nehmen an, dass Komponenten designt wurden, um bestimmte Funktionen in verschiedenen Standards oder Entwurfsmustern erfüllen zu können. Dabei wird in diesem Paper eine Methode diskutiert, die die Funktionsfähigkeit der Komponenten misst. Diese Funktionsfähigkeit der Komponenten soll auf Basis dieser Messungen so beschrieben werden, dass andere Entwickler diese Komponenten für ihre Zwecke nutzen können und wollen.
Die Funktionsfähigkeit einer Komponente ist sowohl im *Domain Engineering* als auch im *Application Engineering* ein wichtiger Aspekt, wobei hier ein Hauptaugenmerk auf das Design und die Validierung der Komponente gelegt wird. Nutzen soll dies laut den Autoren vor allem den Entwicklern und den Designern.

Link zum Paper:
`http://cse.spsu.edu/jwang/research/cbsd/References/CBSE6/p13.pdf`

A.20 Project Management in a Software Product Line Organization

Paul C. Clements, Lawrence G. Jones, and Linda M. Northrop, (*Software Engineering Institute*) and John D. McGregor [CJNM05]

Zusammenfassung:
In traditional software engineering project management, managers provide focused guidance to a team responsible for producing a specific result in a specified amount of time. Today, however, organizations are increasingly taking a product line approach to software to exploit product commonalities. Software product line organizations have unique practices and project definitions. These unconventional features offer new challenges and directions for traditional project management. How does the traditional concept of a project - a temporary endeavor aimed at creating a unique product or service - hold up under this new paradigm? In this article, we discuss this question, along with how the idea of a „project“ and project management techniques must expand to fit a product line context. In particular, we show how the „overall guidelines, policies, and procedures“ that Thayer and Pyster spoke of some years ago remain crucially important in product line organizations today. IEEE, August 2011)

Klassifizierung:
Prozessphase:
Organisation
Stakeholder:
Projektmanager
Artefakt:
Anforderungen,
Komponenten,
Prozesse

Kurze Begründung:
Die Autoren diskutieren die Frage, wie ein Projekt und die Projektmanagementmaßnahmen erweitert werden müssen, damit von einer Einzelproduktplanung in die SPL-Planung übergegangen werden kann. Vor allem in Bezug auf allgemeine Richtlinien, Strategien und Prozesse, die essentielle Bestandteile der SPL-Entwicklung sind.
Dabei werden drei grundlegende Aufgabenbereiche vorgestellt und diskutiert:

- *core asset developement*
  Das Entwickeln von wiederverwendbaren Kernbestandteilen und vor allem deren Organisation -> Organisation der wiederverwendbaren Komponenten
- *product developement*
  Aus dem Output des core asset development und den Anforderungen werden neue Produkte innerhalb der Software-Produktlinie entwickelt
- sowie das Management dieser Punkte.

Mit dem Ziel, allen Anforderungen an die Organisation gerecht zu werden und die Prozesse, die sich bei der SPL-Entwicklung bilden zu überblicken und zu koordinieren (Anforderungen, Prozesse). Im Vordergrund stehen die Projekte, die aus den oben genannten Aufgabenbereichen resultieren und die Aufgaben des Projektmanagements. Die Projektmanager wenden über alle Phasen der Organisation der Produktlinie und die daraus zu entwickelnden Produkte hinweg die beschriebenen Projekte inhaltlich an.

Link zum Paper:
`http://sites.google.com/site/ingangel/projectmanagementsofwtware.pdf`

A.21 Quality Modeling for Software Product Lines

Adam Trendowicz and Teade Punter [TP03]

Zusammenfassung:
In today's embedded software systems development, non-functional requirements (e.g., dependability, maintainability) are becoming more and more important. Simultaneously the increasing pressure to develop software in less time and at lower costs pushes software industry towards product line's solutions. To support product lines for high quality embedded software, quality models are needed. In this paper, we investigate to which extent existing quality modeling approaches facilitate high quality software product lines. First, we define several requirements for an appropriate quality model. Then, we use those requirements to review the existing quality modeling approaches. We conclude from the review that no single quality model fulfills all of our requirements. However, several approaches contain valuable characteristics. Based upon those characteristics, we propose the Prometheus approach. Prometheus is a goal-oriented method that integrates quantitative and qualitative approaches to quality control. The method starts quality modeling early in the software lifecycle and is reusable across product lines.

Klassifizierung:
Prozessphase:
Domain Engineering
Stakeholder:
Alle Beteiligten
Artefakt:
Anforderungen,
Prozesse

Kurze Begründung:
Die Autoren definieren Anforderungen an ein Qualitätsmodell, um den frühen Phasen der SPL-Entwicklung Qualität zu gewährleisten. Sie untersuchen bestehende Qualitätsmodelle auf diese Anforderungen, um dann letztendlich das von ihnen entwickelte Modell *Prometheus* vorzustellen, das bis dahin als einziges Qualitätsmodell allen Anforderungen genügt. Dabei spielen bei der Erfüllung der Qualität die Verantwortlichkeiten, Zusammenhänge und Beziehungen über alle Phasen des *Domain Engineering* eine große Rolle (Prozesse). *Prometheus* erfüllt folgende Anforderungen:

- Qualitätsziele definieren (durch Befragung der Kunden)
- Qualitätsmerkmale spezifizieren (Spezifikation der Inhalte des Modells von allen Beteiligten des *Domain Engineering*)
- Beziehungen spezifizieren (Spezifikation der Modellstruktur von allen Beteiligten des *Domain Engineering*)
- das Modell bewerten (durch alle Beteiligten des *Domain Engineering*)
- das Modell operationalisieren (vor allem Domänenexperten)

Hier wird ersichtlich, dass alle an der Software-Produktlinie Beteiligten, d. h. vom Kunden bis zum Architekten, in diesem Modell eine Rolle spielen (Alle Stakeholder).

Link zum Paper:
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.109.2295&rep=rep1&type=pdf`

A.22 Granularity in Software Product Lines

Christian Kästner, Sven Apel and Martin Kuhlemann [KAK08]

Zusammenfassung:
Building software product lines (SPLs) with features is a challenging task. Many SPL implementations support features with coarse granularity - e.g., the ability to add and wrap entire methods. However, fine-grained extensions, like adding a statement in the middle of a method, either require intricate workarounds or obfuscate the base code with annotations. Though many SPLs can and have been implemented with the coarse granularity of existing approaches, fine-grained extensions are essential when extracting features from legacy applications. Furthermore, also some existing SPLs could benefit from fine-grained extensions to reduce code replication or improve readability. In this paper, we analyze the effects of feature granularity in SPLs and present a tool, called Colored IDE (CIDE), that allows features to implement coarse-grained and fine-grained extensions in a concise way. In two case studies, we show how CIDE simplifies SPL development compared to traditional approaches.

Klassifizierung:
Prozessphase:
Domänenrealisierung
Stakeholder:
Entwickler
Artefakt:
Architektur,
Komponenten

Kurze Begründung:
Die Autoren untersuchen die Auswirkungen auf die Granularität von unterschiedlichen Ansätzen bei der SPL-Entwicklung und beweisen, dass die bereits erforschten Ansätze nicht in der Lage sind, Aussagen über genau granulierte Ergänzungen bei Features zu machen oder Ideen, diese zu implementieren, bieten.
Dazu stellen sie das von ihnen entwickelte Tool CIDE vor, das den oben genannten Anforderungen genügt. D. h. CIDE beschränkt Features auf einen konstruktiven Code-Bestandteil, um so Entwicklern das Nutzen dieser Features zu vereinfachen, dabei wird Feingranularität gewährleistet. Das Tool unterstützt die Entwickler, das Feature mit Navigations- und Projektionsmöglichkeiten zu verstehen und erschafft Möglichkeiten die Software-Produktlinie in unterscheidbare Feature-Module zu exportieren.
CIDE soll vor allem eine Hilfe darstellen eine Software-Produktlinie (mit Hilfe der oben genannten veränderten Feature) effektiv und effizient wie möglich zu entwickeln, um die Alternative der komplett neuen Implementierung von allen notwendigen Features sinnvoll zu umgehen. Das Hauptaugenmerk liegt dabei auf der Realisierungsphase im *Domain Engineering*, bezüglich der Architektur und der Komponenten, durchgeführt von Entwicklern.

Link zum Paper:
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.135.6475&rep=rep1&type=pdf`

A.23 Automated Diagnosis of Product-line Configuration Errors in Feature Models

J. White and D. C. Schmidt, D. Benavides and P. Trinidad and A. Ruiz-Cortés [WSB+08]

Zusammenfassung:
Feature models are widely used to model software product-line (SPL) variability. SPL variants are configured by selecting feature sets that satisfy feature model constraints. Configuration of large feature models can involve multiple stages and participants, which makes it hard to avoid conflicts and errors. New techniques are therefore needed to debug invalid configurations and derive the minimal set of changes to fix flawed configurations.
*This paper provides three contributions to debugging feature model configurations: (**1**) we present a technique for transforming a flawed feature model configuration into a Constraint Satisfaction Problem (CSP) and show how a constraint solver can derive the minimal set of feature selection changes to fix an invalid configuration, (**2**) we show how this diagnosis CSP can automatically resolve conflicts between configuration participant decisions, and (**3**) we present experiment results that evaluate our technique. These results show that our technique scales to models with over 5,000 features, which is well beyond the size used to validate other automated techniques.*

Klassifizierung:
Prozessphase:
Domänenvalidierung
Stakeholder:
Entwickler
Artefakt:
Testen

Kurze Begründung:
In diesem Paper steht vor allem die Fehlersuche und -beseitigung in großen Feature-Modellen, die bei stufenweiser oder Multistakeholder-Konfiguration entstanden sind, im Vordergrund. Dabei werden ungültige Feature-Modelle in ein CSP transformiert, um dann die Randbedingungen aufzulösen. Somit kann die kleinstmögliche Anzahl an Feature-Modellen mit den dazugehörigen Modifikationen abgeleitet werden, wodurch die Konfiguration in einen gültigen Zustand versetzt wird.
Desweiteren wird die CURE-Technik vorgestellt und erklärt, die die CSPs nutzt, um Fehler und Konflikte in Konfigurationen zu diagnostizieren. Das Hauptaugenmerk dieser Arbeit liegt vor allem auf der Fehlersuche und -beseitung in Feature-Modellen (während der Validierungsphase testen Softwareentwickler diese Modelle), diese werden stark bei der Entwicklung von Software-Produktlinien genutzt (*Domain Engineering*).
Multistakholder: Mehrere Personen aus unterschiedlichen Fachbereichen arbeiten zusammen an der Lösung eines Problems.
CURE (**C**onfiguration **U**nderstanding and **RE**medy): Die Lösung basiert auf dem Entwerfen von Varianten Diagnose Tools in automatisierten Software-Produktlinien
CSP (**C**onstraint **S**atisfaction **P**roblem): Eine CSP ist eine Gruppe von Variablen und den dazu gehörigen Bedingungen.

Link zum Paper:
`http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4626856`

A.24 Testing Software Product Lines Using Incremental Test Generation

Engin Uzuncaova, Daniel Garcia, Sarfraz Khurshid and Don Batory [UGKB08]

Zusammenfassung:
We present a novel specification-based approach for generating tests for products in a software product line. Given properties of features as first-order logic formulas, our approach uses SAT-based analysis to automatically generate test inputs for each product in a product line. To ensure soundness of generation, we introduce an automatic technique for mapping a formula that specifies a feature into a transformation that defines incremental refinement of test suites. Our experimental results using different data structure product lines show that incremental approach can provide an order of magnitude speed-up over conventional techniques.

Klassifizierung:
Prozessphase:
Applikationsvalidierung
Stakeholder:
Entwickler,
Tester
Artefakt:
Testen

Kurze Begründung:
Die Autoren stellen einen zu diesem Zeitpunkt neuartigen Ansatz für effiziente Testgenerierung für Produkte in Software-Produktlinien vor und diskutieren diesen (*Application Engineering*). In diesem Ansatz werden Ideen aus der SPL-Entwicklung und des *specifictaion-based testing* kombiniert (Validierungsphase).
Neuartig sind dabei zwei Punkte:

1. Jedes Produkt wird als eine Komposition von Features spezifiziert, wobei jedes Feature als ein *Alloy Formel* festgelegt ist (Vor allem Aufgabe der Entwickler).

2. In diesem Paper wird der *Alloy Analyser* genutzt, um Tests stufenweise zu generieren (in erster Linie Aufgabe der Tester). Im konventionellen Gebrauch geschieht dies in einem gesamten Schritt.

Es werden automatisch erzeugte Tests generiert (Testen), bei bereits bestehenden Features, die zusammen kombiniert ein Produkt ergeben sollen. Dies geschieht während der Validierungsphase des *Application Engineering*.

Specification-based testing: Das Programm wird entsprechend seinen Bedingungen auf das damit verbundene Verhalten getestet.
Alloy: ist eine deklarative Beschreibungssprache (formale Sprache), um komplexe strukturelle Einschränkungen und Verhalten in einem Softwaresystem darzustellen.

Link zum Paper:
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.148.8463&rep=rep1&type=pdf`

A.25 Coping with variability in product-line architectures using component technology

Bo Nørregaard Jørgensen and Wouter Joosen [JJ03]

Zusammenfassung:
Since any successful software product is continuously evolving in order to fulfill market requirements, it needs an architectural basis that can sustain the necessary variability to meet changing requirements. In this paper we presents a component-oriented approach for developing product-line architectures that can accommodate variability through separation of architectural, functional and non-functional concerns. The approach is based on our experiences from developing a product-line architecture for building experimental robot controllers that can control arbitrary robotic manipulators.

Klassifizierung:
Prozessphase:
Domänenanforderungen,
Domänendesign,
Applikationsanforderungen,
Applikationsdesign
Stakeholder:
Architekten,
Entwickler
Artefakt:
Architektur,
Komponenten

Kurze Begründung:
Die Autoren stellen einen systematischen Ansatz für die Entwicklung einer komponentenbasierten Produktlinie vor, aus der später Produkte abgeleitet werden sollen (*Domain Engineering* und *Application Engineering*). Dieser beinhaltet die Möglichkeit, Variabilität anzupassen, indem die Belange der Architektur sowie die funktionalen und nichtfunktionalen Belange voneinander getrennt betrachtet werden. Sie diskutieren vor allem den Umgang mit Variabilität innerhalb der SPL-Architektur, die auf Komponentensystemen basiert (Architektur, Komponenten).
Es wird hervorgehoben und diskutiert, dass eine gut designte Komponente sehr früh eine wesentliche Rolle bei der Entwicklung einer SPL spielt (Design). Zusätzlich werden über Richtlinien diskutiert, die beschreiben wie geerbte Flexibilität bei diesem Ansatz genutzt werden sollte, um Komponentensysteme zu bilden und wie diese Richtlinien zu einem grundlegenden und ganzheitlichen Bestandteil der Architektur dieses Komponentensystems gemacht werden kann (Anforderungen).

Sie definieren die Aufgaben der Beteiligten, die für die Umsetzung ihres Ansatzes zuständig sind:

- Systemarchitekt (definiert die Richtlinien für die Architektur und für die Zusammenstellung der Komponenten)
- Komponentenentwickler (implementiert die Komponenten anhand der vorgegebenen Richtlinien vom Systemarchitekt)
- Applikationsentwickler (stellt die Komponenten nach den Richtlinien des Systementwicklers zusammen, um ein Produkt zu entwickeln)

Link zum Paper:
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.148.8451&rep=rep1&type=pdf`

A.26 Agile Software Product Lines, Deconstructed

John D. McGregor [McG08]

Zusammenfassung:
There was much interest at this year's Software Product Line Conference in how to combine agile and product line techniques. Agile teams seek to address change one product at a time while product line organizations take an investment view by addressing change among a set of products. On the surface there are some seeming contradictions between the methods, but they may not be as different as they are sometimes portrayed. In this issue of Strategic Software Engineering I want to deconstruct product line and agile practices, compare the pieces, and make some suggestions about how to re-construct a hybrid method. I will do this in part by treating agility as a quality attribute of processes.

Klassifizierung:
Prozessphase:
Domain Engineering
Application Engineering
Stakeholder:
Alle Beteiligten
Artefakt:
Prozesse

Kurze Begründung:
John D. McGregor diskutiert darüber, wie flexibel Software-Produktlinien bei Veränderungen (z. B. bei kurzfristigen Kundenwünschen) sein sollten, wobei hier die Veränderungen die gesamte SPL betreffen und auch Gruppen von Produkten (*Domain Engineering* und *Application Engineering*). Dabei werden unterschiedliche Teile aus der *agile production* und aus der Produktlinienproduktion analysiert, um dann die die zusammenpassen könnten zu kombinieren. Die Teile, die nicht gut zusammenpassen, werden verworfen ohne dabei grundsätzliche Charakteristiken beider Methoden zu verändern. Dabei soll eine „agile" Produktlinie entstehen.
Dieser Ansatz bezieht sich auf die Prozesse (Aufgaben, Verantwortlichkeiten und Zusammenhänge) in allen Phasen des *Domain Engineering* und *Application Engineering*. Dabei spielen alle beteiligten Stakeholder eine Rolle.

Agile production: Schnelle und effiziente Anpassung der Produktion auf sich laufend verändernde Kundenbedürfnisse. Vor allem ein Begriff in der klassischen Einzelproduktion.

Link zum Paper:
`http://www.jot.fm/issues/issue_2008_11/column1/`

A.27 Visualisation of Variability in Software Product Line Engineering

Daren Nestor, Luke O'Malley, Aaron Quigley, Ernst Sikora and Steffen Thiel [NOQ+08]

Zusammenfassung:
Using a product line approach allows companies realize significant improvements in time-to-market, cost, productivity, and quality. One fundamental problem in software product line engineering is related to the fact that a product line of industrial size can easily incorporate several thousand variation points. This makes variability management and product derivation tasks extremely difficult. This paper elaborates on the idea of using visualisation techniques to support and improve the effectiveness of these tasks. A reference model that helps to frame the visualisation research and important areas that affect the use of visualisation techniques in software product line engineering are presented.

Klassifizierung:
Prozessphase:
Domänendesign,
Applikationsdesign
Stakeholder:
Alle Beteiligten
Artefakt:
Prozesse

Kurze Begründung:
Die Autoren arbeiten in diesem Paper Visualisierungstechniken aus, um vor allem Variabilität und Variationspunkte einer komplexen SPL so darzustellen, dass diese überschaubar bleibt und bei Bedarf trotzdem in die Tiefe gehen kann. Die Effektivität von signifikanten Produktlinienabläufen, wie Variabilitätsmanagement und Produktableitung ist ein Hauptziel dieser Forschungsarbeit. D. h. das Hauptaugenmerk liegt auf den Prozessen innerhalb des *Domain Engineering* und *Application Engineering*. Dabei werden zu den bisherigen Techniken drei weitere wichtige Forschungsrichtungen abgegrenzt:

- Darstellung von Informationen (Es sollen so Kommunikations- und damit Informationsprobleme überwunden werden.)
- Task Support (direkte Unterstützung für die Stakeholder einer Produktlinie bezüglich Variabilitätsmanagement und Produktableitung, z. B. Bestimmen der Aufgaben)
- Interaktion der Nutzer (im Sinne von Benutzerfreundlichkeit, d. h., dass jeder Nutzer abgefragte Informationen individuell gefiltert bekommt, um so das Meiste aus der Visualisierung rauszuholen.)

Im Großen und Ganzen hilft eine Visualisierungstechnik den Überblick und somit das Verständnis für eine sich schnell ausdehnende SPL beizubehalten. Zusätzlich kann diese Technik die Komplexität bezüglich des Verwaltens der Dokumentationen, Anwendungen und der wiederverwendbaren Artefakte reduzieren.
Hier sollten alle gerade genannten Punkte über alle Prozesse hinweg aufgenommen, modelliert und für alle Beteiligten nachvollziehbar sein und einen guten Überblick verschaffen (Design, alle Stakeholder).

Link zum Paper:
`http://www.vamos-workshop.net/2007/files/VAMOS07_Paper_7_Slides.pdf`

A.28 Tracing Software Product Line Variability - From Problem to Solution Space

Kathrin Berg and Judith Bishop, Dirk Muthig [BBM05]

Zusammenfassung:
The management of variability plays an important role in successful software product line engineering. There is a need for a universal variability management approach to be consistent and scalable; it should provide traceability between variations at different levels of abstraction and across various generic development artifacts; and there should be a means for visualizing variability. Focusing specifically on the aspect of traceability in the context of such an approach, we define a conceptual variability model that captures variability information in a third dimension, and allows a 1-to-1 mapping of variability between the problem space and the solution space. Decision models, of which the feature model is most popular, are commonly used for, amongst others, managing traceability of variation. These, however, usually reside in a two dimensional space. We analyze the feature model in a small case study with regards to our conceptual variability model, and present our findings.

Klassifizierung:
Prozessphase:
Domain Engineering
Stakeholder:
Alle Beteiligten
Artefakt:
Prozesse

Kurze Begründung:
Die Autoren stellen hier ein konzeptionelles Variabilitätsmodel vor, um Nachvollziehbarkeit von Variabilität in Software-Produktlinien zu gewährleisten und vergleichen dieses Modell mit dem klassischen Feature-Modell. Unter Nachvollziehbarkeit ist hier die Beschreibung von Beziehungen und Abhängigkeiten zwischen zwei Artefakten gemeint, die während der unterschiedlichen Phasen der Softwareentwicklung entwickelt werden (alle Phasen des *Domain Engineering*, Prozesse).
Das in diesem Paper vorgestellte Modell erlaubt eine eins-zu-eins Abbildung der Variabilität zwischen dem Problemraum und dem Lösungsraum und entspricht gleichzeitig den Anforderungen eines vereinheitlichten Ansatzes.
Problemraum: Beschreibung und Festlegung des Systems, vor allem während der Analyse und der Entwicklung der Anforderungen im *Domain Engineering*.
Lösungsraum: Das konkrete System, das vor allem während der Architektur-, Design- und Realisierungsphase entwickelt wird.
Hier sollen über alle Prozesse der SPL-Entwicklung hinweg Variabilität und Variationspunkte aufgenommen, modelliert und für alle Beteiligten nachvollziehbar gemacht werden (Prozesse, alle Stakeholder).

Link zum Paper:
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.107.9241&rep=rep1&type=pdf`

A.29 Goal-driven Product Derivation

John D. McGregor [McG09]

Zusammenfassung:
The purpose of a software product line organization is to produce products. Some organizations adopt the software product line approach because they want better quality products, some want the products faster, and some want to produce the products using fewer resources. There are many different ways to produce products and which way is the best depends on which of many goals are the ultimate objective. In this issue of Strategic Software Engineering I will discuss how the specifications of goals for the production system during strategic planning affect how products are derived in a product line organization. (By the way, remember to visit splc.net to see what is happening at SPLC 2009)

Klassifizierung:
Prozessphase:
Domain Engineering: Anforderungen und Design
Application Engineering: Anforderungen und Design
Stakeholder:
Entwickler,
Designer
Artefakt:
Prozesse

Kurze Begründung:
McGregor diskutiert hier Produktableitung innerhalb einer SPL. Genauer noch, wie diese Ableitung bzw. Entwicklung des Produktes aus der gegebenen SPL durch die strategische Planung der spezifischen Ziele in der Produktionsplanung beeinflusst wird.
McGregor behauptet und diskutier hier, dass diese Entwicklung des Produktes schon im *Domain Engineering* beginnen muss, damit eine spätere Produktableitung effizient gebildet werden kann. Das bedeutet, dass aus den zusammenhängenden Features und den später zugefügten Features neue Produkte effizient abgeleitet werden können und deswegen muss hier die Planung frühzeitig geschehen. Aus den, im *Domain Engineering* entwickelten Methoden und Artefakten, werden dann im *Application Engineering* die neuen Produkte abgeleitet.
Alle eben genannten Entwicklungsschritte werden durch den im Paper vorgestellten Produktionsplan (Produktionsstrategie, -methode und der endgültige -plan) erklärt, d. h. die Entwicklung bezieht sich vor allem auf die Planungsphasen der Software-Produktlinie und später der Produktableitung (Anforderungen und Design aller Prozesse innerhalb der Entwicklungsphasen -> Entwickler, Designer).

Link zum Paper:
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.159.5448&rep=rep1&type=pdf`

A.30 Exploring the Context of Product Line Adoption

Stan Bühne, Gary Chastek, Timo Käkölä, Peter Knauber, Linda Northrop and Steffen Thiel [BCK+03]

Zusammenfassung:
To successfully adopt a product line approach an organization needs to define its adoption goals, conceive a strategy, and implement a plan to achieve those goals. This process is repeated for each business unit and individual affected by the product line adoption. This paper describes how the characteristics of the market, organization, business unit, and individual influence product line adoption goals, strategies, and plans.

Klassifizierung:
Prozessphase:
Organisation
Stakeholder:
Alle Beteiligten
Artefakt:
Prozesse

Kurze Begründung:
Die Autoren diskutieren in diesem Paper die erfolgreiche Einführung einer Software-Produktlinie und wie die damit verbundenen Ziele, Strategien und Pläne von den charakteristischen Merkmalen des Marktes, des Unternehmens, dessen Geschäftseinheiten und den Individuen, die am Entwicklungsprozess beteiligt sind, beeinflusst werden.
Sie behaupten, dass jede Organisationsstufe seine Ziele bezüglich der Einführung einer Software-Produktlinie setzen, seine Strategien konzipieren, und die Pläne durchsetzen muss, um diese Ziele zu erreichen.
D. h. ein Unternehmen, das eine SPL einführen will, muss über alle Entwicklungsphasen hinweg seine Organisationsschritte anpassen bezüglich der Prozesse in der Softwareentwicklung und während der Organisation seiner Ziele, Strategien und Pläne (Organisation, Prozesse). Das soll von allen beteiligten Stakeholdern während der Entwicklungsphasen berücksichtigt und gegebenenfalls angewandt werden.

Link zum Paper:
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.1201&rep=rep1&type=pdf`

B Dateien

Diese Arbeit beinhaltet eine CD-ROM mit folgendem Inhalt:

Im Ordner Webseite:

- AE_Matrix.html
- DE_Matrix.html
- ORGA_Matrix.html
- style_de.css

Im Ordner Hauptseite:

- auswertung.htm
- einfuhrung.html
- Formular.html
- INDEX.html
- klassifizierung.html
- quali.html
- quelle.html
- style.css
- tabelle.html
- TERM.html
- varibil.html
- vorteile.html

Literatur

[BBM05] Kathrin Berg, Judith Bishop, and Dirk Muthig. Tracing software product line variability: from problem to solution space. In *Proceedings of the 2005 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, SAICSIT '05, pages 182–191. South African Institute for Computer Scientists and Information Technologists, 2005.

[Böc05] Günter Böckle. Innovation management for product line engineering organizations. In J. Henk Obbink and Klaus Pohl, editors, *Software Product Lines*, volume 3714 of *Lecture Notes in Computer Science*, pages 124–134. Springer Berlin / Heidelberg, 2005.

[BCK+03] Stan Bühne, Gary Chastek, Timo Käkölä, Peter Knauber, Linda Northrop, and Steffen Thiel. Exploring the context of product line adoption. In *Proc. 5th Int. Workshop on Product Family Engineering*, pages 19–31. Springer-Verlag, 2003.

[BFG+02] Jan Bosch, Gert Florijn, Danny Greefhorst, Juha Kuusela, J. Henk Obbink, and Klaus Pohl. Variability issues in software product lines. In *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, PFE '01, pages 13–21. Springer-Verlag, 2002.

[BH01] Jan Bosch and Mattias Högström. Product instantiation in software product lines: A case study. In *Proceedings of the Second International Symposium on Generative and Component-Based Software Engineering-Revised Papers*, GCSE '00, pages 147–162. Springer-Verlag, 2001.

[BM05] Kathrin Berg and Dirk Muthig. A critical analysis of using feature models for variability management. Technical report, University of Pretoria, Computer Science Department, Pretoria, South Africa, 2005.

[Bos00a] Jan Bosch. *Design and use of software architectures: adopting and evolving a product-line approach*. ACM Press/Addison-Wesley Publishing Co., 2000.

[Bos00b] Jan Bosch. Organizing for software product lines. In *Proceedings of the International Workshop on Software Architectures for Product Families*, IW-SAPF-3, pages 117–134. Springer-Verlag, 2000.

[Bos02] Jan Bosch. Maturity and evolution in software product lines: Approaches, artefacts and organization. In *Proceedings of the Second International Conference on Software Product Lines*, SPLC 2, pages 257–271. Springer-Verlag, 2002.

[CB91] Gianluigi Caldiera and Victor R. Basili. Identifying and qualifying reusable software components. *Computer*, 24:61–70, February 1991.

[CDS06] Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. Coverage and adequacy in software product line testing. In *Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis*, ROSATEA '06, pages 53–63. ACM Press, 2006.

[CE00] Krysztof Czarnecki and Ulrich Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley Professional, 2000.

[CJNM05] Paul C. Clements, Lawrence G. Jones, Linda M. Northrop, and John D. McGregor. Project management in a software product line organization. *IEEE Softw.*, 22:54–62, September 2005.

[DKMT96] Prem Devanbu, Sakke Karstu, Walcélio Melo, and William Thomas. Analytical and empirical evaluation of software reuse metrics. In *Proceedings of the 18th international conference on Software engineering*, ICSE '96, pages 189–199. IEEE Computer Society, 1996.

[FCS+08] Eduardo Figueiredo, Nelio Cacho, Claudio Sant'Anna, Mario Monteiro, Uira Kulesza, Alessandro Garcia, Sérgio Soares, Fabiano Ferrari, Safoora Khan, Fernando Castor Filho, and Francisco Dantas. Evolving software product lines with aspects: an empirical study on design stability. In *Proceedings of the 30th international conference on Software engineering*, ICSE '08, pages 261–270. ACM Press, 2008.

[Fit96] Ronan Fitzpatrick. Software quality:definitions and strategic issues. Technical report, MSc Computing Science (ITSM), Staffordshire University, 1996.

[JF88] Ralph E. Johnson and Brian Foote. Designing reusable classes. *Object-Oriented Programming*, 1(2), 1988.

[JH02] Enrico Johansson and Martin Höst. Tracking degradation in software product lines through measurement of design rule violations. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, SEKE '02, pages 249–254. ACM Press, 2002.

[JJ03] Bo Nørregaard Jørgensen and Wouter Joosen. *Technology of Object-Oriented Languages, Systems and Architectures*, chapter Coping with variability in product-line architectures using component technology, pages 208–219. Kluwer Academic Publishers, 2003.

[KAK08] Christian Kästner, Sven Apel, and Martin Kuhlemann. Granularity in software product lines. In *Proceedings of the 30th international conference on Software engineering*, ICSE '08, pages 311–320. ACM Press, 2008.

[Lin98] Frank van der Linden, editor. *Development and Evolution of Software Architectures for Product Families*. Springer-Verlag, 1998.

[McG08] John D. McGregor. Agile software product lines, deconstructed. *Journal of Object Technology*, 7:7–19, November - December 2008.

[McG09] John D. McGregor. Goal-driven product derivation. *Journal of Object Technology*, 8:7–19, July - August 2009.

[Mci69] Doug Mcilroy. Mass-produced software components. In J. M. Buxton, P. Naur, and B. Randell, editors, *Proceedings of Software Engineering Concepts and Techniques*, pages 138–155. NATO Science Committee, January 1969.

[MRW77] Jim A. McCall, Paul K. Richards, and Gene F. Walters. *Factors in Software Quality Vol. I-III.* 1977.

[MSC03] John D. McGregor, Judith A. Stafford, and Il-Hyung Cho. Measuring component reliability. Technical report, Clemson University, 2003.

[Mus96] John D. Musa. Software-reliability-engineered testing. *Computer*, 29:61–68, November 1996.

[Nei86] J. Neighbors. *The Draco approach to constructing software from reusable components*, pages 525–535. Morgan Kaufmann Publishers Inc., 1986.

[NOQ+08] Daren Nestor, Luke O'malley, Aaron Quigley, Ernst Sikora, and Steffen Thiel. Visualisation of variability in software product line engineering. In *Proceedings of the 2nd Int. Workshop on Variability Modelling of Software-intensive Systems*, Essen, Germany, 2008.

[PBL05] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer-Verlag New York, Inc., 2005.

[Per98] Dewayne E. Perry. Generic architecture descriptions for product lines. In *Proceedings of the Second International ESPRIT ARES Workshop on Development and Evolution of Software Architectures for Product Families*, pages 51–56. Springer-Verlag, 1998.

[SB00] Mikael Svahnberg and Jan Bosch. Issues concerning variability in software product lines. In *Proceedings of the International Workshop on Software Architectures for Product Families*, IW-SAPF-3, pages 146–157. Springer-Verlag, 2000.

[Sch02] Klaus Schmid. A comprehensive product line scoping approach and its validation. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 593–603. ACM Press, 2002.

[SPR04] Periklis Sochos, Ilka Philippow, and Matthias Riebisch. Feature-oriented development of software product lines: Mapping feature models to the architecture. In Mathias Weske and Peter Liggesmeyer, editors, *Object-Oriented and Internet-Based Technologies*, volume 3263 of *Lecture Notes in Computer Science*, pages 23–42. Springer Berlin / Heidelberg, 2004.

[Szy97] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., 1st edition, 1997.

[TOHS99] Peri Tarr, Harold Ossher, William Harrison, and Stanley M. Sutton, Jr. N degrees of separation: multi-dimensional separation of concerns. In *Proceedings of the 21st international conference on Software engineering*, ICSE '99, pages 107–119. ACM, 1999.

[TP03] Adam Trendowicz and Teade Punter. Quality modeling for software product lines. In *7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'03*, 2003.

[UGKB08] Engin Uzuncaova, Daniel Garcia, Sarfraz Khurshid, and Don Batory. Testing software product lines using incremental test generation. In *Proceedings of the 2008 19th International Symposium on Software Reliability Engineering*, pages 249–258. IEEE Computer Society, 2008.

[vdHDM03] André van der Hoek, Ebru Dincel, and Nenad Medvidović. Using service utilization metrics to assess the structure of product line architectures. In *Proceedings of the 9th International Symposium on Software Metrics*, pages 298–. IEEE Computer Society, 2003.

[vdLBK+04] Frank van der Linden, Jan Bosch, Erik Kamsties, Kari Känsälä, Henk Obbink, and Philips Medical Systems. Software product family evaluation. In *Bosch, J. On the Development of Software Product-Family Components. SPLC 2004*, pages 110–129. SpringerVerlag, 2004.

[vGBS00] Jilles van Gurp, Jan Bosch, and Mikael Svahnberg. Managing variability in software product lines. Technical report, University of Groningen, Groningen, The Netherlands, 2000.

[vGBS01] Jilles van Gurp, Jan Bosch, and Mikael Svahnberg. On the notion of variability in software product lines. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture*, WICSA '01, pages 45–. IEEE Computer Society, 2001.

[WSB+08] J. White, D. C. Schmidt, D. Benavides, P. Trinidad, and A. Ruiz-Cortés. Automated diagnosis of product-line configuration errors in feature models. In *Proceedings of the 2008 12th International Software Product Line Conference*, pages 225–234. IEEE Computer Society, 2008.

[YR06] Liguo Yu and Srini Ramaswamy. A configuration management model for software product line. Technical report, Computer Science and Informatics Indiana University South Bend, South Bend, USA, 2006.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie dass ich diese Bachelorarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Passau, 22. September 2011

Semah Senkaya