Technische Universität Chemnitz
Fakultät für Wirtschaftswissenschaften
Lehrstuhl für Wirtschaftsinformatik II
Prof. Dr. P. Loos

**Seminararbeit**

Patrick Langi

# Produktlinien-basierte Entwicklung betrieblicher Informationssysteme

**2002**

| | |
|---|---|
| Verfasser | Patrick Langi |
| | Köthensdorfer Strasse 8 |
| | 09114 Chemnitz |
| E-Mail: | patrick.langi@isym.tu-chemnitz.de |
| Studiengang | Wirtschaftsinformatik |
| Semester | 8. Hochschul- und Fachsemester |
| Abgabetermin | 23.08.2002 |

**Inhaltsverzeichnis**

**Abbildungsverzeichnis**

**Tabellenverzeichnis**

# 1 Einleitung

In der vorliegenden Arbeit soll das Konzept der *Produktlinien* erläutert werden. Dabei handelt es sich um ein Konzept der Wiederverwendung von Software, welches darauf abzielt, speziell für die Anwendungsdomäne erzeugte Ergebnisse wiederzuverwenden. Die Ergebnisse rekrutieren sich dabei aus zwei parallel ablaufenden Prozessen. Zum einen aus dem Softwareentwicklungsprozess und zum anderen werden sie in der Domain Analysis gewonnen. Ziel eines solchen Vorgehens, wie es im Rahmen der *Produktlinien* aufgezeigt wird, ist es, den Softwareentwicklungsprozess effizienter zu gestalten. Dabei sollen unter anderem Entwicklungskosten gespart und Entwicklungszeiten verkürzt werden. Diese Ziele der Wiederverwendung sowie die damit verbundenen Probleme werden im Folgenden kurz erläutert. Im zweiten Abschnitt wird das Konzept der *Produktlinien* gegenüber verwandten Konzepten abgegrenzt. Im Darauffolgenden wird das Konzept der *Produktlinie* dargelegt und bewertet. Motivation der vorliegenden Arbeit ist es, die Tauglichkeit des Konzeptes zur Entwicklung von betrieblichen Informationssystemen darzustellen. Wiederverwendung in der Softwaretechnik versucht, Ergebnisse aus einem Softwareprojekt beliebig oft wiederzuverwenden. Das heißt, dass einige Ergebnisse nicht erneut erzielt werden, sondern als Input in neue Softwareprojekte eingehen. Dadurch werden die notwendigen Prozesse durch das Ergebnis eines anderen Softwareentwicklungsprozesses substituiert. Ein derartiges Vorgehen ermöglicht die Aktivierung von Einsparungspotentialen. Zur Erreichung dieses Ziels haben sich vielerlei Formen der Wiederverwendung herausgebildet und werden in der Literatur diskutiert.[1]

Die Objekte der Wiederverwendung werden in der Literatur unterschiedlich bezeichnet. Die meisten dieser Bezeichnungen beziehen sich jedoch auf den Code, welcher wiederverwendet wird.[2] So lautet beispielsweise die Definition einer Komponente des Arbeitskreises 5.10.3 der Gesellschaft für Informatik wie folgt: „Eine Komponente besteht aus verschiedenartigen (Software-) Artefakten. Sie ist wiederverwendbar, abgeschlossen vermarktbar, stellt Dienste über eine wohl definierte Schnittstelle zur Verfügung, verbirgt ihre Realisierung und kann in Kombination mit anderen Komponenten eingesetzt werden, die zur Zeit der Entwicklung nicht unbedingt vorhersehbar ist.“[3] Softwareartefakte beziehungsweise Artefakte werden im selben Schriftstück als „[...] der ausführbare Code, darin verankerte Grafiken, Textkonstanten usw. [...]“ beschrieben. Eine solche Begrifflichkeit ist zur Charakterisierung des Produktlinienkonzeptes ungeeignet. Daher soll vorläufig[4] der Begriff *Ergebnis* verwendet werden. Dieser bezeichnet dabei allen, wie auch immer gearteten, Output eines Softwareentwicklungsprojektes.

Im Allgemeinen verspricht die Wiederverwendung von Software fünf signifikante Vorteile[5].

- Senkung des mengenmäßigen Einsatzes zur Erzielung des im Rahmen des Projektes definierten Erfolges.
- Sicherung der Qualität der verwendeten Ergebnisse.
- Aufwand zur Generierung des Architekturentwurfes sowie zur Generierung des Fachkonzeptes.
- Verkürzung der Entwicklungszeiten sowie
- eine Reduzierung der Entwicklungskosten.

---

[1] Vgl. Rost 1997. S. 357ff.
[2] Vgl. Rezagholi 1997, S. 222.
[3] Vgl. Ackermann et al. 2002, S. 1.
[4] Eine Präzisierung der Begrifflichkeit wird in Abschnitt 4 vorgenommen.
[5] Vgl. zur Vertiefung Balzert 1998, S. 639.

In der Literatur werden weiterhin 3 Klassen von Problemen der Wiederverwendung wiederkehrend genannt. Als erste Klasse ist die Unabhängigkeit verschiedener Projekte zu nennen. Diese führt dazu, dass Code nicht problemlos in unterschiedlichen Projekten genutzt werden kann. Die dafür in der Literatur aufgeführten Gründe sind vielfältig. In neuerer Zeit wird vielen Unternehmen bewusst, dass ein Vorgehen ohne Wiederverwendung von Softwareartefakten nicht sinnvoll ist und somit die Überwindung der Probleme angezeigt ist. Der Grund dafür ist vor allem in veränderten Marktstrukturen zu sehen, so sind die Unternehmen immer stärker darauf angewiesen schnell neue Produkte in den Markt einzuführen und vorhandenen an die Nachfrage anzupassen.[6] Daher ergibt sich eine erste These:

- Damit Unternehmen langfristig am Softwaremarkt überleben können ist es notwendig, dass einmal erzeugter Code wiederverwendet wird.[7]

Die zweite Klasse von Problemen ergibt sich aus der Frage, welche Teile eines Softwareentwicklungsprojektes wiederverwendet werden. Typischerweise ist das der erzeugte Code[8]. Problematisch an diesem Vorgehen ist, dass Wissen, welches während des Softwareentwicklungsprozesses kumuliert wurde, nur durch ein Ergebnis, den generierten Code, in zukünftige Projekte eingeht. Das ist insofern verwunderlich, als dass die meisten Vorgehensmodelle zur Softwareentwicklung vorsehen, eine Menge mehr als nur Code zu produzieren. Insbesondere sind hier Aktivitäten im Rahmen des Fachkonzeptes[9] zu nennen, da diese häufig nicht unerheblich sind. Daraus ergibt sich die zweite These:

- Zusätzlich zur Wiederverwendung von Code ist es ökonomisch sinnvoll, Ergebnisse anderer Phasen der Softwarenwicklung wiederzuverwenden.

Die dritte Klasse von Problemen lässt sich aus den Projektbeschränkungen ableiten. Diese sind oftmals von vielen zu beachtenden Randbedingungen gekennzeichnet. Dies führt dazu, dass diese Randbedingungen den Code für andere Projekte unbrauchbar werden lassen oder eine teure und zeitaufwendige Anpassung des Codes notwendig machen. Es wird deutlich, dass eine geplante Wiederverwendung von Code nicht allein durch das Hinzufügen einer Phase im Softwareentwicklungsprozess, welche die Verfügbarkeit von wiederverwendbarer Software prüft, erfolgen kann. Vielmehr ist es notwendig den gesamten Prozess der Softwareentwicklung neu auszurichten. Es darf nicht nur das Ziel verfolgt werden, vorhandene Codestücke in das Projekt einfließen zu lassen, sondern es müssen neuerliche Aktivitäten auf das Ziel der Wiederverwendbarkeit des erzeugten Codes ausgerichtet werden.

- Um eine Wiederverwendung von Software zu ermöglichen, ist es notwendig, den Softwareentwicklungsprozess neu zu überdenken.

Um nun diese drei Thesen zu untermauern ist es notwendig darzulegen wie ein Konzept der Wiederverwendung gestaltet sein soll, damit neben dem Code auch die Ergebnisse anderer Phasen projektübergreifend nutzbar gemacht werden können. Anhand dieses Konzeptes können dann die Thesen geprüft und die sich ergebenden Vorteile sichtbar gemacht werden. Ebendies soll mittels des Konzeptes der *Produktlinien* erreicht werden.

---

[6] Vgl. Bass et al. 1997, S. 1.

[7] Es wird hier in Anlehnung an Bass et al. 1997 die Formulierung „überleben“ genutzt. Auch wenn diese etwas hart erscheint, ist ihre Intention dennoch angebracht.

[8] Vgl. Rezagholi 1997, S. 222.

[9] Siehe Scheer 1998.

# 2 Abgrenzung zu verwandten Konzepten

## 2.1 Kriterien

Um die Abgrenzung zwischen den *Produktlinien* und verwandten Konzepten zu ermöglichen sollen verschiedene inhaltliche Kriterien herangezogen werden. Diese werden hier kurz erläutert. Wiederverwendung scheint nur sinnvoll, wenn die wiederverwendeten Ergebnisse Einsparungspotentiale aktivieren. Dabei wird ein notwendiger Aufwand (Prozess) durch sein Ergebnis ersetzt. Hierbei sind die wiederverwendeten Ergebnisse vielfältig. Gemein ist allen, dass sie die Lösung eines Problems innehaben und somit Wissen darstellen. An dieser Stelle soll nun in Wissen im Sinne einer dokumentierten Problemlösung und in Wissen in Form einer vorliegenden Implementierung unterschieden werden, wobei die Implementierung genau eine Komponente ist. Ein erstes Unterscheidungsmerkmal sind somit die *Objekte der Wiederverwendung*, wobei Komponenten und sonstiges Problemlösungswissen unterschieden werden.

Ein zweites Kriterium ist das Ausmaß oder der *Wirkungsbereich der Wiederverwendung*. Wie bereits weiter oben herausgestellt wurde, aktiviert die Wiederverwendung Einsparungspotentiale, indem Prozesse der Anwendungsentwicklung durch ihre Ergebnisse substituiert werden. Es besteht jedoch noch ein weiteres Einsparungspotential. Aktiviert wird dieses Einsparungspotential bei der Wartung von Anwendungen. So werden anstatt der Anwendungen die Komponenten gewartet. Deren Verbesserung kommt dann allen Anwendungen zu gute, welche aus diesen Komponenten zusammengesetzt sind. Unterschieden sollen an dieser Stelle die Phasen des Lebenszyklus eines Softwaresystems sein, in denen Einsparungspotentiale durch Wiederverwendung aktiviert werden. Hier wird unterschieden in Systementwicklung und Systembetrieb.

Ein drittes Kriterium ist in der *Vollständigkeit* zu sehen. Vollständigkeit meint, in welchem Umfang ein Konzept der Wiederverwendung eine Anwendung aus Softwarebausteinen zusammensetzt. Unterschieden wird hierbei in Konzepte, die eine Anwendung ausschließlich auf Basis von Softwarebausteinen entwickeln oder wenigstens die theoretische Möglichkeit einräumen. Die zweite Gruppe von Konzepten unterstützt dies nicht.

Ein weiteres Kriterium soll in der *Form der Wiederverwendung* gesehen werden. Dabei wird die Art und Weise untersucht, in der die wiederverwendeten Ergebnisse innerhalb des Konzeptes behandelt werden. Hierbei müssen die verschiedenen Phasen der Wiederverwendung berücksichtigt werden. Im Rahmen der Beschaffung von Ergebnissen ist nach den Annahmen der Modelle bezüglich der *Verfügbarkeit von Ergebnissen* zu sehen. Innerhalb der Verwendung ist die Anforderung an die *Granularität des Ergebnisses* zu berücksichtigen. Des Weiteren muss hier die *Behandlung der Ergebnisse* berücksichtigt werden. Dabei haben die genannten Kriterien folgende Ausprägungen:

- Verfügbarkeit von Ergebnissen
    - Umfassend, es wird davon ausgegangen, dass Ergebnisse für jeden Anspruch verfügbar sind.
    - Hoch, Annahme ist hier, dass Ergebnisse verfügbar sind.
    - Gering, dabei wird davon ausgegangen, dass Ergebnisse verfügbar sein könnten.
- Granularität des Ergebnisses
    - fachlich
    - problembezogen
    - implementierungsnah

- Behandlung der Ergebnisse
    - Black-box, die Ergebnisse werden an Stelle des Problemlösungsprozesses verwendet.
    - White-box, die Ergebnisse werden als solche innerhalb des aktuellen Problemlösungsprozesses berücksichtigt.

Im Folgenden wird eine Tabelle dargestellt, welche die Konzepte der Referenzmodelle, Frameworks und Fachkomponenten und *Produktlinien* als Formen der Wiederverwendung vergleichend gegenüberstellt. Dabei werden die herausgearbeiteten Kriterien genutzt. Im Anschluss an die Tabelle werden die Ausprägungen zu den einzelnen Formen erläutert.

| | **Komponenten-modell** | **Referenzmodelle** | **Framework** | **Produktlinien** |
|---|---|---|---|---|
| **Objekt der Wiederverwendung** | Komponenten | Problem-lösungswissen | Komponenten | Komponenten und Problem-lösungswissen |
| **Wirkungsbereiche der Wiederverwendung** | bei System-entwicklung | bei System-entwicklung oder bei Systembetrieb | bei System-entwicklung | Bei System-entwicklung und Systembetrieb |
| **Vollständigkeit** | Idealtypisch ja, aber nicht notwendig | Nein | Nein | zwingend |
| **Verfügbarkeit von Ergebnissen** | Umfassend | Hoch | Hoch | Gering |
| **Granularität von Ergebnissen** | Fachlich oder Problembezogen | Fachlich | Fachlich | Fachlich oder Problembezogen |
| **Behandlung von Ergebnissen** | Black Box | White Box | Black Box | Black Box |

**Tabelle 1: Vergleich von Konzepten der Wiederverwendung**

## 2.2 Framework

Eine Definition von Framework findet sich im Lexikon der Wirtschaftsinformatik[10]: „... Konglomerat von Ansätzen, die darauf zielen, die Produktivität in der Softwareentwicklung zu verbessern, indem die Wiederverwendung von Code ergänzt wird um eine effiziente Wiederverwendung von Entwurfs- und Domänenwissen.“ Diese Definition zeigt durchaus Überschneidungen mit dem Konzept der *Produktlinien*. Auch beim Framework wird Wiederverwendung auf Grundlage von Entwurfs- und Domänenwissen betrieben. Dabei wird davon ausgegangen, dass das entsprechende Wissen im Framework umgesetzt ist. Das heißt also, dass lediglich die Komponente als solche wiederverwendet wird. Diese ist allerdings auf der Grundlage von Wissen über die Domäne entwickelt worden.

Um eine Produkt zu erhalten, muss dieser gebotene Rahmen ausgefüllt werden. Dazu werden besondere Anforderungen an die Gestaltung von Frameworks gestellt. Dies betrifft beispielsweise die abstrakte einheitliche Applikationsschnittstelle, die verborgene Heterogenität zugrundeliegender Schichten und weitere.[11] Diese Forderungen implizieren eine angemessen abstrakte Implementierung des Framework.[12] Diese ist eine Reaktion auf die unterschiedlichen Anforderungen, die eine Domäne an ein Framework stellt. Bei der Nutzung des Frameworks innerhalb der Wiederverwendung wird der abstrakte Rahmen herangezogen und auf dieser Basis eine Anwendung entwickelt. Nachdem die Anwendung entwickelt wurde, verliert das Framework als solches an Bedeutung und das entstandene System tritt in den Vordergrund der Betrachtung. Somit ist der Umfang der Wiederverwendung auf die Systementwicklung zu begrenzen.

Wie schon beschrieben stellt das Framework einen Rahmen zur Verfügung, welcher durch eine individuelle Programmierleistung an die formulierten Anforderungen angepasst wird. Damit ist das entstehende Produkt nicht ausschließlich ein Produkt der Wiederverwendung, sondern eben auch ein Produkt der individuellen Programmierleistung. Daher ist dieses Konzept nicht vollständig im Sinne des Vergleiches.

Bezüglich der Art und Weise der Wiederverwendung lässt sich sagen, dass ein Framework, welches die geforderten Kriterien erfüllt, vorhanden ist oder entwickelt werden kann, indem ein vorhandenes Framework an die speziellen Anforderungen angepasst wird. Die Dienste, welche das Framework anbietet, sind elementarer Natur, beruhen allerdings auf den Ergebnissen der Untersuchung einer Domäne, daher kann von fachlichen Diensten gesprochen werden. Dabei wird das Framework unabhängig von seiner Implementierung über klare Schnittstellen, als Black Box verwendet.

## 2.3 Fachkomponenten

Die Konzepte der *Produktlinie* sowie der komponentenbasierten Anwendungsentwicklung ähneln sich grundsätzlich sehr. Ausgangspunkt bei beiden Konzepten ist die Annahme, dass sich der Aufwand zur Entwicklung von betrieblichen Informationssystemen reduzieren lässt, indem bereits vorhandene Ergebnisse genutzt werden. Weitere Analogien in den Annahmen sind bezüglich der Qualität sowie der Kosten zu sehen, diese sollen jedoch im Folgenden nicht näher betrachtet werden.

Die Objekte der Wiederverwendung sind hier vornehmlich Komponenten und eng mit diesen verbundene Ergebnisse wie beispielsweise die Dokumentation oder Tests. Dies ergibt sich aus der Annahme über die

---

[10] Vgl. Frank 1997, S. 167.

[11] Vgl. Kolland et al. 1993, S. 23 – 32.

[12] Vgl. Dazu drei Arten der Verwendung von Frameworks in: Frank 1997, S. 167.

Verfügbarkeit der Komponenten. So wird hier angenommen, dass Komponentenmärkte existieren, auf welchen Komponenten gekauft werden können. Somit wird davon ausgegangen, dass jede Art von Komponente verfügbar ist. Um nun weitere Ergebnisse als Komponente zu nutzen, müssten weitere Ergebnisse des Softwareentwicklungsprozesses angeboten werden. Dies ist nicht der Fall. Selbst wenn diese angeboten werden, sind sie als solche nicht wiederverwendbar, da während der Erzielung der Ergebnisse nicht explizit auf eine Wiederverwendbarkeit geachtet wurde.

Die so beschafften Komponenten werden bei der Systementwicklung genutzt. Im weiteren Verlauf werden durch derartige Wiederverwendung keine Einsparungspotentiale aktiviert.

Bei der Systementwicklung wird keine Vollständigkeit gefordert. So ist es vorteilhaft, eine benötigte Funktionalität eines Systems über eine Komponente bereitzustellen, aber es ist nicht für jede Funktionalität gefordert. Die Idee der Vollständigkeit findet jedoch auch bei der Komponentenwiederverwendung Beachtung. So wird die Vision formuliert, zukünftig Softwaresysteme lediglich aus Komponenten zusammenzusetzen.[13] Woher die gehandelten Komponenten stammen sollen, wird nicht betrachtet. Dennoch kann hier unterstellt werden, dass an einem Komponentenmarkt genau zwei professionelle Akteursgruppen aktiv sind. Einmal sind dies solche, die auf die Produktion von Komponenten spezialisiert sind und als Anbieter auftreten, und andererseits solche, die vornehmlich Anwendungen entwickeln und somit Komponenten nachfragen. Weitere Überlegungen hinsichtlich von Synergieeffekten, welche unter Umständen bei der Entwicklung und Verwendung solcher Komponenten auftreten, werden nicht angestellt.

Die Granularität eines Dienstes wird grundsätzlich nicht eingeschränkt, allerdings sind mit Fachkomponenten solche Komponenten gemeint, welche fachliche Funktionalität übernehmen oder diese direkt unterstützen. Die Komponenten werden als Black-Box-Komponenten verwendet.

Zentrale Unterschiede sind in den den Konzepten zugrundeliegenden Annahmen bezüglich der Verfügbarkeit von Komponenten und dem Umfang der Verwendung von Komponenten zu sehen. Das Konzept der komponentenbasierten Anwendungsentwicklung geht davon aus, dass beliebig viele Komponenten zur Verfügung stehen, welche in der Lage sind, die im zu entwickelnden System benötigten Funktionalitäten anzubieten. Somit steht das Konzept der komponentenbasierten Anwendungsentwicklung vor dem Problem des Auffindens von Komponenten sowie der Integration von Komponenten in ein bestehendes Informationssystem. Auf diesen Gebieten finden sich daher Schwerpunkte der Forschung komponentenbasierter Anwendungsentwicklung. Bei *Produktlinien* werden verfügbare, gefundene und selbst erstellte Komponenten in der *Asset Base* gepoolt. Dabei wird davon ausgegangen, dass eine im Pool befindliche Komponente Funktionalität bereitstellt, welche innerhalb verschiedener Produkte benötigt wird. Weiterhin wird implizit davon ausgegangen, dass sie diese Funktionalität besser anbietet, als eine vergleichbare am Markt verfügbare Komponente. Daher finden sich *Produktlinien* meistens auf Softwaremärkten, die relativ klein sind und somit sehr spezielle Komponenten benötigen.[14] Da, wie herausgestellt, die Komponenten innerhalb des Konzeptes der *Produktlinien* gepoolt werden, bietet es sich an, die Eigenschaften dieser Komponenten bezüglich der Wiederverwendung in der speziellen Domäne zu optimieren. So kann die Integrationsfähigkeit einer

---

[13] Ackermann et al. 2002: "Diesem Ziel liegt als *Leitbild*, im Sinne eines idealen zukünftigen Zustands, die Idee einer kompositorischen, plug-and-play-artigen Wiederverwendung von Black-Box-Komponenten zu Grunde, deren Realisierung einem Verwender vorborgen bleibt und die auf einem Softwaremarkt gehandelt werden."

[14] Vgl. 6 Management.

Komponente hinsichtlich der Nutzung innerhalb der *Produktlinie* verbessert werden, indem eine notwendige Parametrisierung auf die Parameter beschränkt wird, welche innerhalb der *Produktlinie* variieren. Das Auffinden der Komponenten, um sie der *Asset Base* hinzufügen zu können, spielt hierbei eine untergeordnete Rolle. Häufig wird davon ausgegangen, dass derartige Komponenten bereits im Unternehmen vorhanden sind.[15] Daher wird vorrangig versucht, vorhandene Systeme in derartige Komponenten zu zerlegen, um sie im Rahmen einer *Produktlinie* zu nutzen. Bezüglich der Reichweite der Wiederverwendung kann vermerkt werden, dass im Konzept der *Produktlinien* eine mehrfache Nutzung einer Komponente angestrebt wird und sogar notwendig ist. Dies ist innerhalb der komponentenbasierten Anwendungsentwicklung nicht zwingend erforderlich und wird daher auch nicht gefordert. Hinsichtlich des Umfangs der wiederverwendeten Artefakte sind nur geringe Unterschiede feststellbar, so wird auch innerhalb der komponentenbasierten Anwendungsentwicklung gefordert, nicht nur den ausführbaren Code sondern weitere Ergebnisse der Komponentenentwicklung zu nutzen[16]. Diesbezügliche Unterschiede ergeben sich lediglich aus der Anpassung der Komponenten für eine Nutzung innerhalb der *Produktlinie* sowie der in der Domain Analysis, welche Ergebnisse zur Verfügung stellt, welche in dieser Form bei der komponentenbasierten Anwendungsentwicklung nicht zur Verfügung stehen.

Zusammenfassend lässt sich also festhalten, dass die beiden Konzepte sehr ähnlich sind. Man könnte sogar soweit gehen, dass die *Produktlinien* eine Unterform der komponentenbasierte Anwendungsentwicklung sind, wobei Unterschiede in den Annahmen über die Verfügbarkeit von Komponenten bestehen. Die schlechtere Verfügbarkeit von Komponenten wird innerhalb der *Produktlinien* zu heilen versucht, indem eine „Komponentenelite“ erstellt und genutzt wird. Bestrebungen innerhalb der komponentenbasierten Anwendungsentwicklung gehen heute meist in die Richtung, eine Spezifikation von Komponenten zu erzeugen. Die Nutzung einer solchen Spezifikationen innerhalb der *Produktlinien* kann ebenfalls durchaus sinnvoll sein, um vorhandene Komponenten besser aufzufinden. Wenn eine derartige Markttransparenz erzeugt wird, wie sie mit Spezifikationen angestrebt wird, ist das Konzept der *Produktlinien* in Teilen obsolet, da das Auffinden, auch von exotischen Komponenten weniger Probleme bereitet.

## 2.4 Referenzmodelle

Referenzmodelle modellieren und formalisieren Abläufe sowie Strukturen innerhalb des betrieblichen Wertschöpfungsprozesses.[17] Es werden dabei Branchen- sowie Software-Referenzmodelle unterschieden.

Bei der Modellierung von Branchen-Referenzmodellen werden allgemeingültige Prozesse und Strukturen einer Branche einbezogen. Dieses Vorgehen ähnelt dem der Domain Analysis, wobei diese lediglich die Strukturen und die Abläufe eines Unternehmens berücksichtigt. Daher sind die Ergebnisse der Domain Analysis weniger spezifisch als die eines Referenzmodells. Es lässt sich festhalten, dass Unterschiede hinsichtlich Detaillierungsgrad des Modellierungsgegenstandes, Modellzweck und repräsentierten Original zu sehen sind[18].

Besonders deutlich wird der oben genannte Unterschied bei der Betrachtung der Artefakte, welche wiederverwendet werden. Bei Referenzmodellen wird Problemlösungswissen wiederverwendet. Dieses entstammt aus einer umfassenden Analyse der Domäne, wobei hier der Domänenbegriff weiter gefasst ist

---

[15] Vgl. im Unterschied dazu Annahmen in: Ackermann et al. 2002, S. 1.
[16] Vgl. Definition von Artefakt in: Ackermann et al. 2002, S. 1.
[17] Vgl. Marent 1995, S. 304f.
[18] Vgl. Marent 1995, S. 303 - 313.

als bei den *Produktlinien*. So ist eine Domäne hier das Umfeld einer Branche, wobei alle variierenden Parameter der einzelnen Akteure innerhalb der Branche außen vor gelassen werden. So ist es möglich, ein für nahezu alle Unternehmen der Branche anwendbares Modell zu liefern. Dabei löst sich dieses Modell vollständig von einer tatsächlichen Implementierung. Vielmehr ist es auf die fachlichen Aspekte einer Domäne konzentriert.

Ein solches Modell findet häufig bei der Einführung von Standardsoftware Anwendung. Dort stellt es einen Aktionsrahmen zur Verfügung, in dem die Besonderheiten des Unternehmens berücksichtigt werden. Somit ist es möglich die Standardsoftware an den durch das Referenzmodell gebotenen Rahmen mittels Parametrisierung anzupassen. Hier sind durchaus Vorteile erkennbar. So kann einmal gewonnenes Wissen über die Spezifikationen innerhalb einer Branche in verschiedenen Unternehmen Anwendung finden. Diese Ergebnisse können ebenso bei der Entwicklung von Individualsoftware Verwendung finden. Nicht nur bei der Entwicklung von Informationssystemen kann ein Referenzmodell genutzt werden. Ebenso ist es denkbar, die Prozesse eines Unternehmens durch derartige Modelle zu optimieren beziehungsweise dem Branchenstandard anzupassen.

Ein Referenzmodell abstrahiert von der Masse der Unternehmen einer Branche, um einen modellierbaren, kleinsten gemeinsamen Nenner zu suchen, daher ist die Vollständigkeit nicht gefordert und nicht gegeben.

Software-Referenzmodelle beschreiben Strukturen, Funktonen und Abläufe, wie sie durch Standardsoftware unterstützt werden. Ziel ist es allgemein formulierte Geschäftsmodelle auf die Charakteristik eines Unternehmens abzustimmen. Das Finden von Parallelen zwischen diesem Konzept und dem der *Produktlinien* scheint konstruiert. Daher wird hier auf eine Erläuterung verzichtet.

# 3 Das Konzept der Produktlinien

## 3.1 Einleitung

Ausgehend von den in der Einleitung angesprochenen Zielen und Problemen der Wiederverwendung, soll im Folgenden das Konzept der *Produktlinien* dargestellt werden. Dazu wird das Konzept in einem ersten Schritt anhand seines Ablaufes charakterisiert. Im zweiten Schritt wird ein mögliches Vorgehen aufgezeigt, um Ergebnisse zu produzieren, welche wiederverwendet werden können. Im dritten Schritt wird die Entwicklung von Softwareprodukten dargestellt. Die abschließende Bewertung des Produktlinienkonzeptes erfolgt auf Basis der Ziele und Probleme der Wiederverwendung. Insgesamt orientiert sich die Darstellung an der Eignung des *Produktlinien* Konzeptes zur Entwicklung betriebswirtschaftlicher Anwendungen.

Um die Komplexität des Themas aufzuzeigen, findet sich in Anhang A eine Literaturliste. Die Autoren[19] der Literatursammlung erheben den Anspruch, eine Bibliographie erstellt zu haben. Dem kann beigepflichtet werden. Da die Literaturliste erst in der Mitte der Literatursammlung im Rahmen dieser Arbeit gefunden wurde, wurden vorher zirka 50 relevante Quellen recherchiert. Von diesen fanden sich nahezu alle in der Literaturliste wieder. Daher wird an dieser Stelle eine annähernde Vollständigkeit der Bibliographie angenommen. Dies mag daran liegen, dass die Autoren jedem die Möglichkeit geben, nicht aufgeführte Arbeiten hinzuzufügen, was den nicht immer stringenten Stil der Referenzen begründet.

[19] Die Bibliographie ist eine Auflistung des Institut Experimentelles Software Engineering der Fraunhofer Gesellschaft.

In Anhang B wurden alle Institute aufgelistet, welche sich mit dem Thema *Produktlinien* beschäftigen und deren Arbeiten hier eingeflossen sind. Wenn publizierte Vorgehensmodelle vorliegen, werden diese ebenfalls in der Auflistung berücksichtigt.

Das folgende Kapitel beschäftigt sich mit dem Konzept der *Produktlinien*. Dazu wird am Anfang eine Definition einer *Produktlinie* gegeben. Im darauffolgenden Kapitel finden die *Assets*, welche die Basis der *Produktlinie* bilden, Betrachtung. In etwa dem gleichen Umfang wird anschließend die Domain Analysis erörtert.

Im Folgenden, wie auch schon bisher wird der Begriff der *Produktlinie* synonym zu dem englischsprachigen Begriff der Product Line verwendet. Dabei ist stets eine Software *Produktlinie* oder Software Product Line gemeint.

## 3.2 Produktlinien – eine Definition

Eine *Produktlinie* beschreibt eine Menge von Produkten; diese werden der *Produktlinie* anhand ihrer Eigenschaften zugeordnet. Das Software Engineering Institute an der Carnegie Mellon University definiert eine *Produktlinie*[20] wie folgt: „A Product Line is defined to be a group of products sharing a common, managed set of features that satisfy needs of a selected market or mission."[21] Es wird deutlich, dass Produkte einer *Produktlinie* gemeinsame Eigenschaften aufweisen. Dies allein ist jedoch nicht ausreichend um eine Zuordnung vorzunehmen. Vielmehr ist es notwendig, dass die Produkte gewisse Anforderungen erfüllen. Diese liegen, in dieser recht allgemein gefassten Definition darin, dass die Produkte die Bedürfnisse eines bestimmten Marktes oder einer bestimmten Aufgabe befriedigen. Das heißt also, dass an Produkte einer *Produktlinie* die Anforderung gestellt wird, Gemeinsamkeiten auf einer fachlichen Ebene aufzuweisen. Rein technische Gemeinsamkeiten, wie beispielsweise die Implementierungssprache, werden in dieser Definition nicht berücksichtigt. Es ist somit offensichtlich, dass im Rahmen des *Produktlinien* Konzeptes keine absolute Wiederverwendbarkeit[22] angestrebt wird. Vielmehr ist es das Ziel, Ergebnisse innerhalb eines begrenzten fachlichen Horizontes wiederzuverwenden. Dieser Rahmen, in dem die Produkte einer *Produktlinie* Wiederverwendbarkeit aufweisen,[23] sind soll im Folgenden, in Übereinstimmung mit der Literatur, als Domäne bezeichnet werden. In Anlehnung an den Arbeitsbericht der Arbeitsgruppe für Wiederverwendung und *Produktlinien* des WISR8[24] wird unter einer Domäne: „... a specialized body of knowledge – an ara of expertise …"[25] verstanden.[26] Im Folgenden ist noch die Frage zu stellen, was in der Definition der *Produktlinie* mit Product beschrieben wird. Product meint in diesem Zusammenhang das Ergebnis der unternehmerischen Wertschöpfung, welches direkt oder indirekt zum Unternehmenserfolg beiträgt. In unserem Fall handelt es sich also um Software. Ob diese, wie bei Softwareentwicklung für dritte üblich, direkt oder indirekt, indem Prozesse im Unternehmen unterstützt werden, dem Unternehmenserfolg dient, ist belanglos.

---

[20] Vgl. Bass et al. 2000, S. 1.

[21] Eine Produktlinie ist definiert als eine Gruppe von Produkten die eine gemeinsame und strukturierte Menge von Eigenschaften teilen, welche die Anforderungen eines bestimmten Marktes oder einer bestimmten Aufgabe befriedigen.

[22] Absolute Wiederverwendung meint die globale Nutzung von Ergebnissen auch über die Grenzen eines Unternehmens hinaus. Beispielsweise der Handel mit Komponenten auf entsprechenden Märkten.

[23] Die Formulierung wurde mit Bedacht gewählt da, wie sich später zeigen wird, keine Produkte wiederverwendbar sind.

[24] Vgl. Clements 1997, S. 3.

[25] Ein spezialisierter Wissensraum – ein von Expertenwissen dominiertes Gebiet.

[26] Zur näheren Erläuterung der Domain vergleiche Abschnitt 4.2.1 in diesem Dokument.

Es bleibt also festzuhalten, dass unter einer *Produktlinie* Erzeugnisse subsumiert werden, welche die Anforderungen einer bestimmten Domäne erfüllen. Damit ist das Konzept der *Produktlinien* markt- oder kundengerichtet, da es die auf dieser Ebene formulierten Anforderungen zu befriedigen versucht[27]. Es soll nun anhand dieser Definition einer *Produktlinie* die Abgrenzung zum verwandten Konzept der Produktfamilie erarbeitet werden. Umgangssprachlich tritt der Begriff Produktfamilie häufig in Verbindung mit ähnlichen Produkten eines Unternehmens auf. Dabei unterscheiden sich die einzelnen Artikel meist nur in wenigen Merkmalen und dem Preis. Dies gibt dem Unternehmen den Spielraum, den Markt zu segmentieren und so verschiedene Kundengruppen mit ähnlich gearteten Produkten anzusprechen.[28] Hier deutet sich der Unterschied bereits an, wohingegen eine *Produktlinie* gerade einen Markt adressiert, verfolgt die Produktfamilie eher das Ziel, einen Markt zu segmentieren. Die Eigenschaft der Marktsegmentierung soll jedoch nicht als vordergründiges Differenzierungsmerkmal genutzt werden. Vielmehr soll die Frage beantwortet werden, was die Produkte einer Produktfamilie gemein haben. Weiss und Lai[29] merken dazu an: „… a [product] family is a set of items that have common aspects and predicted variabilities …"[30]. Wie zu sehen ist, wird bei der Definition einer Produktfamilie weniger auf den Anwendungsrahmen als vielmehr auf die Konstruktionseigenschaften, die Zusammensetzung Wert gelegt. Dementsprechend ist bei Clements[31] zu lesen „... a product family is a technology- or implementation-dependent concept..."[32]. Im Gegensatz zum kunden- oder marktgerichtetem Konzept der *Produktlinien* ist das Konzept der Produktfamilie also eher technologie- und implementierungsabhängig. Insgesamt ist eine Abgrenzung der Konzepte nur in Bezug auf konkrete Ausprägungen möglich. Clements[33] fordert sogar, die vorhandenen Grenzen zwischen den Konzepten zu durchbrechen. So sieht er eine Produktfamilie als „... a set of related systems that are built from a common set of core assets ..."[34]. Er fordert eben dies auch für das Konzept der *Produktlinien*, darauf soll jedoch im folgenden Kapitel vertiefend eingegangen werden. In einer späteren Schrift von Clements[35] ist dementsprechend zu lesen, dass *Produktlinien* und Produktfamilien ein und dasselbe Konzept adressieren[36]. Dabei unterstellt er folgende terminologischen Gleichheiten.

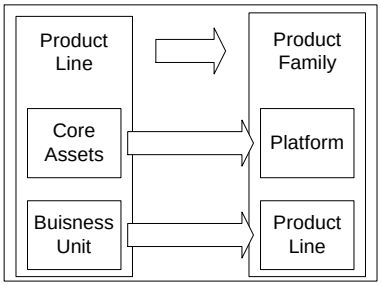


**Abbildung 1: Terminologische Analogien von Produktfamilien und Produktlinien[37]**

[27] Clements 2000, S. 3.
[28] Vgl. Meffert 2000.
[29] Weis; Lai 1999, S. 2.
[30] Eine Produktfamilie ist eine Menge von Gegenständen, welche gemeinsame Aspekte und vorhersehbare Abweichungen aufweisen.
[31] Clements 2000, S. 3.
[32] Die Produktlinie ist ein technologie- oder implementierungsabhängiges Konzept.
[33] Clements 1997, S. 3.
[34] Eine Menge verwandter Systeme welche aus einer einheitlichen Menge von Kernbestandteilen aufgebaut sind.
[35] Vgl. Clements; Northrop 2001, S. 14f.
[36] Ähnliche Analogien sieht auch Maccari 2000.
[37] In abgewandelter Form übernommen aus: Clements; Northrop 2001, S. 15.

An dieser Stelle soll für den Rest des Dokuments die Terminologie der *Produktlinien* angewandt werden. Für eine vertiefende Betrachtung der Unterschiede zwischen den Konzepten sei auf die in diesem Abschnitt angesprochene Literatur verwiesen.

Ausführungen zur Abgrenzung der *Produktlinien* gegenüber anderen Konzepten können Clements et al.[38] bzw. den Ausführungen im 2 Abschnitt des Dokumentes entnommen werden. Es sollen abschließend alle so herausgearbeiteten Unterschiede zu verwandten Konzepten dargelegt werden:

- Geplante Wiederverwendung.
- Nutzung vielfältiger Ergebnisse des Entwicklungszyklus.
- Einbeziehung einer fachlichen Analyse der Domäne
- Berücksichtigung der Wiederverwendung bereits bei der Entwicklung von Komponenten
- Produkte werden als Kompositionen betrachtet, wobei die einzelnen Bestandteile einer solchen Komposition der Wartung, Pflege und Weiterentwicklung unterworfen sind. Das Produkt als solches kann nur durch Neukomposition oder Änderungen an den Bestandteilen verändert werden.

Im Folgenden sollen diese Aussagen auf das Konzept der *Produktlinien* angewandt werden, dabei soll herausgestellt werden, in welchem Maße die getroffenen Aussagen und die gestellten Anforderungen erfüllt sind.

## 3.3 Entwurf und Betrieb einer Produktlinie

Der Entwurf einer *Produktlinie* beinhaltet alle Phasen, welche notwendig sind, um das Konzept in die betriebliche Praxis umzusetzen. Der Betrieb der *Produktlinie* zeigt, inwieweit das Konzept der *Produktlinien* die betriebliche Entwicklung von Software unterstützt und welche Aktivitäten notwendig sind, um diese Unterstützung bestmöglich zu gewährleisten. Im Folgenden soll prototypisch das Vorgehen zur Realisierung einer *Produktlinie* aufgezeigt werden. Dabei werden etwaige organisatorische Randbedingungen nicht berücksichtigt.

Im ersten Schritt sollte im Unternehmen evaluiert werden, inwieweit der Wechsel zu einer Softwareentwicklung mittels *Produktlinien* Vorteile verspricht. Diese sollten mit den damit verbundenen Nachteilen verglichen werden, um so den Nutzen einer Einführung zu beurteilen. Wenn diese Entscheidung positiv ausfällt, gilt es, die organisatorischen Randbedingungen zu schaffen. Dazu sollten verschiedene Modelle des organisatorischen Wandels berücksichtigt werden und das am besten geeignetste sollte zur Anwendung kommen.[39] Im nächsten Schritt wird der Umfang der einzuführenden *Produktlinie* bestimmt. Diese und die folgenden Aktivitäten sind mit denen der initialen Einführung einer *Produktlinie* verwandt, welche in dem Abschnitt 4.2.2 in diesem Dokument behandelt werden. An dieser Stelle soll der erste Schritt weitere Betrachtung finden. Im Wesentlichen lassen sich drei potentielle Vorteile[40] und damit Ziele bei der Einführung von *Produktlinien* feststellen.[41]

- Qualität und Kosten

---

[38] Vgl. Clements; Northrop 2001, S. 11ff.
[39] Vgl. Staehle 1999, S. 579ff.
[40] Vgl. Sagarduy et al. 2000.
[41] Eine Operationalisierung der hier aufgezeigten Vorteile findet sich in: Bandinelli; Sagarduy 1996.

Sie werden hier zusammen betrachtet, da ein klarer Trade off besteht. Die Einführungskosten einer *Produktlinie* sind, durch eine Masse durchzuführender Aktivitäten sowie die Entwicklung neuer *Assets* hoch. Daher wird darauf abgezielt, die *Asset Base* aus vorhandenen Softwaresystemen zu extrahieren.[42] Da so alle Bausteine der vorhandenen Produkte innerhalb der *Asset Base* vorliegen, sind Verbesserungen an einem *Asset* allen Anwendungen zuträglich. Dies steigert tendenziell die Qualität der Softwarebausteine[43] und damit die der Produkte, welche aus ihnen zusammengesetzt sind. Durch ein solches Vorgehen werden die Kosten für korrektive Wartung[44] gesenkt.

- Time-to-Market

Häufig sind es nicht die Kosten, welche eine vordergründige Rolle spielen, sondern vielmehr die benötigte Zeit zur Umsetzung von Kundenanforderungen oder zur Reaktion auf Marktveränderungen. Im Rahmen einer *Produktlinie* ist es möglich, die Grundfunktionalität neuer Produkte aus der vorhandenen *Asset Base* zu nutzen und nur die neu benötigte Funktionalität innerhalb der *Asset Base* bereitzustellen.[45] Diese Prozesse laufen zudem noch parallel ab.[46]

Wenn die ermittelten Vorteile innerhalb einer Organisation den Nachteilen überlegen sind, wird eine *Produktlinie* eingeführt. Dazu ist es notwendig, die Infrastruktur zur geplanten Wiederverwendung zur Verfügung zu stellen.[47] Des Weiteren müssen die Rahmenbedingungen dieser Wiederverwendung ermittelt werden.[48]

Nach der Umsetzung der *Produktlinie* muss diese auch betrieben und gewinnbringend eingesetzt werden, um die beschriebenen Vorteile zu realisieren. Daher soll der Betrieb oder die Arbeit mit einer *Produktlinie* erläutert werden. In der Literatur besteht über den Betrieb einer *Produktlinie* weitestgehend Übereinstimmung, sodass die Ausführungen in diesem Dokument meist auf Clements et al.[49] beruhen, da das die wohl umfassendste Darstellung des Ablaufes in der Literatur darstellt.

---

[42] Diesbezügliche Probleme werden in Bandinelli; Rementeria 1996. behandelt.
[43] Vgl. Bosch 1999.
[44] Vgl. Bosch 2000, S. 191.
[45] Vgl. Bosch 2000, S. 191.
[46] Vgl. Clements; Northrop 2001, S. 29ff.
[47] Vgl. 4.1 Assets als Grundlage aller Produktlinien in diesem Dokument.
[48] Vgl. 4.2 Domain Analysis in diesem Dokument.
[49] Vgl. Clements; Northrop 2001.

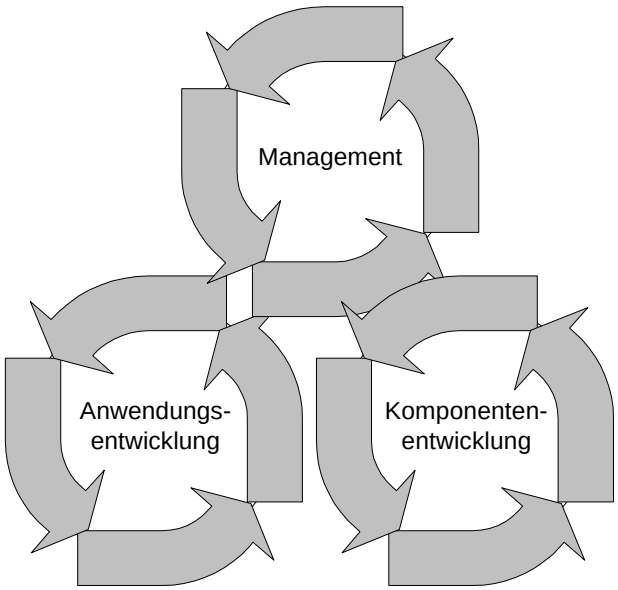


**Abbildung 2: Betrieb einer Produktlinie**

Daraus ergeben sich nun die drei grundlegenden Aktivitäten. In den folgenden Abschnitten wird erst die Komponentenentwicklung, dann die Anwendungsentwicklung und anschließend das Management erläutert. Am Ende findet sich ein Abschnitt zu *Produktlinien* und deren Einsatz in der Praxis sowie eine abschließende Bewertung des Konzeptes.

# 4 Komponentenentwicklung

## 4.1 Assets als Grundlage aller Produktlinien

### 4.1.1 Definition eines Assets – der Komponentenbegriff

*Core Assets* bilden die Grundlage eines Softwaresystem im Rahmen einer *Produktlinie*.[50] Dabei wird ein *Core Asset* wie folgt definiert[51]: „... are those assets that form the basis for the software product line. Core assets often include, but are not limited to, the architecture, reusable software components, domain models, requirement statements, documentation and specification, performance models schedules, budgets, test plans, test cases, work plans, and process descriptions. The architecture is key among the collection of core assets. …"[52]. *Core Assets* stellen die Basis einer Software *Produktlinie*. Im Folgenden ist zu klären, was *Assets* im Allgemeinen sind. Withey[53] charakterisiert diese wie folgt: „A software asset is a description of a solution or knowledge that application engineers use to built or modify products in a product line. To reduce work, the description must be able to explain, or implement trough manipulations,

[50] Vgl. Clements; Northrop 2001, S. 32f.

[51] Vgl. Clements; Northrop 2001, S. 14.

[52] [Core assets] sind Assets, welche die Basis einer Produktlinie bilden. Häufige aber nicht ausschließlich sind Assets Architektur, wiederverwendbare Softwarekomponenten, Modelle der Anwendungsdomäne, Anforderngsdokumente, Dokumentation und Spezifikation, Anforderungsmodelle hinsichtlich Performance, Budgets, Testpläne, Testfälle, Arbeitspläne und –beschreibungen. Die Architektur bildet den Schlüssel der Sammlung an Core Assets.

[53] Siehe Withey 1996, S. 13.

changes necessary for different products. The description may be executable."[54] Der bisherige Gebrauch des Wortes Ergebnis für ein Software *Asset* ist also durchaus gerechtfertigt. Gemäß der Definition nach Withey ist sogar jedes beliebige Ergebnis des Softwareentwicklungsprozesses als *Asset* zu definieren. Dieser Definitionen soll sich hier nicht ohne Einschränkung angeschlossen werden. Davon ausgehend, dass *Assets* dem Entwickler während aller Phasen der Softwareentwicklung zur Verfügung stehen, steht dieser vor dem Problem der Auswahl eines *Assets*, welches seine Arbeit unterstützt. Zur Gewährleistung des Erfolges dieses Einsatzes ist es notwendig, dass das verwendete Ergebnis früherer Bestrebungen eine gewisse Qualität aufweist. Ohne Zweifel ist die Bestimmung der Qualität einer Softwarekomponente oder anderer Ergebnisse nicht trivial[55], dennoch muss diese gewährleistet sein. Dies hat im Wesentlichen zwei Gründe. Zum einen ist dies mit der häufigen Verwendung zu begründen. Das *Asset* wird innerhalb vieler Produkte wiederverwendet, daher muss es eine gewisse Flexibilität innerhalb eines definierten Anwendungsbereiches aufweisen. Wenn diese Flexibilität nicht gegeben ist, erhöht dies den Aufwand der Wartung. Im schlimmsten Fall muss die Komponente aus bereits bestehenden Produkten herausgelöst und ersetzt werden. Weiterhin hat die mangelhafte Qualität innerhalb einer Komponente weitreichende Wirkung, da sie in mehreren Produkten negativ und unvorhersehbar wirken kann. Daher ergibt sich die Forderung nach Flexibilität. Wenn diese bei Komponenten einer *Produktlinie* fehlt, steigt der Aufwand der Entwicklung überproportional stark an. Ein weiterer Grund, welcher die Qualitätssicherung notwendig macht, liegt in der Organisation der *Assets*. Diese werden innerhalb einer *Asset Base* gehalten. Bei Bedarf wird aus dieser das benötigte *Asset* herausgesucht und verwendet. Dabei muss davon ausgegangen werden, dass nicht sichergestellt werden kann dass überschneidungsfreie Ergebnisse angeboten werden. Insbesondere ist das der Fall, wenn alle beliebigen *Assets* der *Asset Base* zugeführt werden. Somit stehen zur Lösung eines Problems mehrere *Assets* bereit. Dies erfordert, zur Lösung eines Problems, eine Suche, welche wiederum die Entwicklungszeit und –kosten erhöht. Daher scheint es sinnvoll, nicht alle Ergebnisse des Softwareentwicklungsprozesses in die *Asset Base* aufzunehmen. Im Folgenden soll daher unter einem *Asset*, in Anlehnung an Withey eine Beschreibung einer Lösung oder von Wissen verstanden werden. Diese wird durch die Anwendungsentwickler genutzt, um Produkte einer *Produktlinie* zu bauen oder zu verändern. Dabei muss die Beschreibung oder die Implementierung in der Lage sein, die Unterschiede zwischen einzelnen Produkten zu berücksichtigen und etwaigen weiteren bekannten Lösungen überlegen sein. Es wird an dieser Stelle vermutet, dass die hier verwendete Definition Witheys Bild eines *Assets* entspricht, er in seiner Arbeit jedoch nicht vordergründiges Gewicht in die Definition eines *Assets* legt[56].

### 4.1.2 Die Bereitstellung der Assets

Die Ursprünge von *Assets*, welche die *Asset Base* der *Produktlinie* bilden, unterscheiden sich. Grundsätzlich werden in der Literatur drei verschiedene Möglichkeiten genannt:

- Verwendung von vorhandenen Softwareartefakten innerhalb der *Produktlinie*[57].
- Entwicklung neuer *Assets* und

---

[54] Ein Software Asset ist die Beschreibung einer Lösung oder von Wissen, welche die Anwendungsentwickler nutzen, um Produkte einer Produktlinie zu bauen oder zu verändern. Um die Arbeit zu reduzieren, muss die Beschreibung oder die Implementierung in der Lage sein, die Unterschiede zwischen einzelnen Produkten zu berücksichtigen. Diese Beschreibung kann ausführbar sein.

[55] Siehe Balzert 1998, S. 253 ff.

[56] Siehe Withey 1996, S. 21ff. Dort wird die von ihm auf S. 13 gegebene Definition konkretisiert.

[57] Vgl. Eisenbarth et al. 2001. Sowie Bergey et al. 2000, S. 3 und S. 10ff.

- Erwerb von *Assets*[58].

Als Ergebnisse der Beschaffung von *Core Assets* sind zu nennen:

- Produktliste[59]:

Dies ist eine Charakterisierung der Produkte einer *Produktlinie*. Dabei werden Gemeinsamkeiten und Unterschiede der einzelnen Produkte herausgestellt und bewertet. Eine solche Liste soll den Raum der möglichen Anwendung auf Basis der *Core Assets*[60] innerhalb der *Asset Base* beschränken. Somit wird der Umfang der *Produktlinie* festgelegt. Eine solche Aufstellung beinhaltet einen starken zeitlichen Bezug, es ist daher unzweifelhaft, dass sich die Produktliste weiterentwickeln, um den Veränderungen des Marktes, der Organisation oder sonstiger Umwelt gerecht zu werden. Diese Produktliste stellt den Startpunkt der Entwicklung einer *Produktlinie* dar und wird im weiteren zeitlichen Verlauf genutzt um die *Produktlinie* aktuell zu halten.

- *Core Asset*

Natürlich der zentrale Bestandteil der Beschaffung. Sie bilden letztlich die Basis, aus der die verschiedenen Mitglieder einer *Produktlinie* zusammengesetzt sind. Im allgemeinen Fall ist eine Architektur vorgegeben, welche die Mitglieder einer *Produktlinie* teilen.[61] Dennoch stellt sich die Architektur als ein Softwarebaustein dar, der im Rahmen der *Produktlinien* wiederverwendet wird. Zu einem Softwarebaustein gehören alle damit verbundenen Dokumente und sonstige im Rahmen der Entwicklung erzielten Ergebnisse. Des Weiteren können frühere Dokumente der Anwendungsentwicklung sowie Commercial of-the-shelf (COTS) Komponenten als *Core Assets* gelten.

- Produktionsplan

Dieser beschreibt die Produktion, also wie die Produkte der *Produktlinie* aus den vorhandenen *Core Assets* zusammengestellt werden.

Es stehen wie oben zu sehen drei Wege zur Verfügung, um die genanten Ziele zu erreichen. Der Produktionsplan sowie die Produktliste sind Ergebnisse, welche in jedem Fall in-house erstellt werden. Wohingegen bei den *Assets* die Möglichkeit besteht, von extern zuzukaufen[62]. Die Produktliste basiert dabei auf von der Marketingabteilung bereitgestellten Informationen über Marktnachfrage und Absatzchancen. Dabei sollte der Zeitrahmen hier nicht zu eng gewählt werden. Eine taktische Ausrichtung (> 3 Jahre) ist unbedingt erforderlich um nicht den Entwicklungen am Markt hinterherzuhinken. Für diesen Zeitraum ist ein Absatzprogramm grob zu formulieren. Des Weiteren müssen Merkmale gesammelt werden, welche den Absatz der Produkte im Zeitverlauf entscheidend

---

[58] Vgl. Clements

[59] Vgl. Clements

[60] Zum Verständnis von Core Assets vgl. 4.1.1 Definition eines Assets.

[61] Hier ist ein Widerspruch zu sehen. Dieser liegt darin, das sie Architektur eben nicht den Komponentenstatus innerhalb der Asset Base hat. Vielmehr stellt die Architektur eine all umfassende Komponente dar, welche innerhalb jeder Anwendung verwendet wird und somit auch den Komponenten zu Grunde liegt. Zur weiteren Vertiefung vgl. 7 Bewertung.

[62] Hier ist ein Problem zu sehen, welches das Auffinden von Komponenten betrifft. Die bei der Entwicklung von Produkten verwendeten Komponenten, befinden sich in der Asset Base. Das Auffinden solcher Komponenten stellt sich als wenig schwierig dar, da die Anzahl aller verfügbaren Komponenten dadurch eingegrenzt wird, das die verwendbaren Komponenten aus der Asset Base stammen müssen. Jedoch findet das Problem des Auffindens von Komponenten zur Schaffung der Asset Base keine Betrachtung somit sind hier ähnliche Probleme wie bei sonstigen Formen der Wiederverwendung zu sehen. Zur weiteren Vertiefung vgl. 7 Bewertung.

beeinflussen. Dieser erste Schritt[63] der Bereitstellung der *Assets*, mündet in der Einigung über den Umfang[64], welchen die zu erstellende *Produktlinie*[65] besitzen soll.[66] Im zweiten Schritt zur Generierung einer *Asset Base* werden vorhandene Strukturen aufgedeckt. Ausgangspunkt ist der Grundgedanke der *Produktlinien*. Dieser sagt, dass alle so verwandten Produkte dasselbe Marktsegment bedienen. Daher werden in der Funktionalität der Produkte Überlappungen vermutet, diese sollen aufgedeckt werden. Ein mögliches Vorgehen ist die Formulierungen eines theoretischen Systems, welches den Umfang der in Schritt 1 definierten Produkte hat.[67] Mittels dieser Beschreibung können Gemeinsamkeiten der Produkte sichtbar gemacht werden. Aus dieser Menge von fachlichen Gemeinsamkeiten müssen in der sich anschließenden Analyse Muster gefunden werden. Dazu werden Kriterien formuliert, welche die Stärke bzw. Größe einer Überlappung wiederspiegeln sollen.[68] Die so ausgewählten Überlappungen bilden dann die gesuchten Gemeinsamkeiten in der Funktionalität der Produkte. Auf Grundlage dieser werden im dritten Schritt die *Assets*[69] ausgewählt. Dabei werden die formulierten Funktionalitäten als Anforderungen an die Komponenten verstanden. Nebenbedingungen sind dabei die Kostenersparnisse, welche durch eine bestimmte Komponente erreicht werden, die benötigte Flexibilität der Komponente, welche den Einsatz in verschiedenen Produkten erlaubt sowie die Fähigkeiten und Erfahrungen der Entwickler, welche den Integrationsaufwand begrenzen. Anhand dieser Größen können Activity Costs[70] ermittelt werden, welche minimiert werden sollten. Dieses Vorgehen ist nicht das einzig mögliche[71], Ziel ist es, am Ende dieses Prozesses aus den möglichen Komponenten die auswählen zu können, die der *Produktlinie*, unter welchen Bedingungen auch immer, am dienlichsten sind. Hier wurde davon ausgegangen, dass vorhandene Softwarebausteine genutzt werden. Wenn solche nicht vorliegen, nicht gefunden werden oder nicht die erforderliche Qualität aufweisen, ist es notwendig, Eigenentwicklungen zu initiieren. Grundsätzlich orientieren sich derartige Eigenentwicklungen an den Kriterien, die jedem Softwareentwicklungsprozess zugrunde liegen. Als Ausgangspunkt und gleichzeitig als Anforderungen an eine Komponente können die Ergebnisse der vorangegangenen Schritte genutzt werden. Auch herbei gilt es, die Kosten so gering als möglich zu halten, da die Kosten der Softwareentwicklung die Activity Costs erhöhen. Hier wird ebenfalls deutlich, welches Kriterium zu einer Eigenentwicklung führen kann. Wenn beispielsweise die Integrationskosten einer vorhandenen Softwarekomponente als zu hoch vermutet werden, lässt sich anstatt dieser ebenso eine Eigenentwicklung[72] einsetzen. Dazu müssen die voraussichtlichen Entwicklungskosten[73] sowie die Kosten der Integration ermittelt werden. Wenn diese geringer als die

---

[63] Es wird versucht das Vorgehen unabhänig vom Beschaffungstyp der Assets zu beschreiben, dabei wird sich jedoch an einer Neubeschaffung orientiert (Eigenentwicklung oder Einkauf). Für den Fall der Verwendung von existierenden Assets ist ein ähnliches Vorgehen beschrieben in: Bass et al. 1998, S. 13ff. Und in Bayer et al. 1999, S. 446 – 463.

[64] Ein systematischer Ansatz zur Ableitung des Umfanges findet sich in: DeBaud et al. 1998, S. 128 – 131.

[65] Die Asset Base stellt die Basis einer Produktlinie, daher ist die Zusammenstellung dieser ein Schritt auf dem Weg zur Produktlinie. Vgl. Bergey et al. 2000, S. 3.

[66] In Bergey et al. 2000. S. 5ff. werden noch weitere als die genannten Informationen als Ergebnis des ersten Schrittes gefordert. An dieser Stelle soll jedoch weiterhin an den Ausführungen in Withey 1996 orientiert werden, da diese Allgemeingültiger sind. Dies gilt speziell im Hinblick auf die möglichen Wege Assets zu beschaffen.

[67] Dieses Vorgehen wird in Withey 1996, S. 21 als das zu wählende angenommen. Ohne weitere Aufwand zu investieren, wird davon ausgegangen, dass durchaus weitere Verfahren geeignet sind.

[68] Vgl. Withey 1996, S. 25ff.

[69] Ausgangspunkt bilden Kandidaten die vorher ausgewählt wurden. Ein mögliches Vorgehen zu dieser Vorauswahl findet sich in Withey 1996, S. 33.

[70] Vgl. Withey 1996, S. 28ff.

[71] Vgl. Bergey et al. 2000, S. 6f.

[72] Unter Berücksichtigung der Fähigkeiten der Entwickler bzw. diese unterstellt.

[73] Ungeachtet entstehender Probleme. Vgl. Kroha 1997, S. 277.

Kosten der entsprechenden Komponente sind, wird die Eigenentwicklung gewählt.[74] Letztlich liegen am Ende des vierten Schrittes die *Assets* vor, welche die *Asset Base* und damit die Basis der *Produktlinie* formen. Auch wenn die bisherigen Ausführungen meist auf Softwarekomponenten abstellten, sind diese nicht alles, was schließlich in der *Asset Base* bereitgestellt wird. Dort sind ebenfalls alle zur Komponente vorhandenen Dokumentationen, Spezifiktionen, Testfälle und Anwendungsfälle abgelegt und können damit innerhalb der Anwendungsentwicklung genutzt werden. Der Inhalt der *Asset Base* wird schließlich noch durch die Ergebnisse der Domain Analysis vervollständigt. Diese sollen im nächsten Abschnitt erläutert werden.

## 4.2 Domain Analysis

### 4.2.1 Begriffsbildung

Die Domain Analyse ist der erste Schritt zur Einführung einer *Produktlinie* in einem Unternehmen. Um ein erstes Bild zu erhalten, soll der Begriff in seine Bestandteile zerlegt werden. Es wird dabei weiterhin auf dem englischsprachigen Begriff gearbeitet. Dies hat den Grund, dass es viele verwandte Konzepte gibt, welche sich lediglich in inhaltlichen Details[75] oder dem Kontext[76] unterscheiden. Es soll zuerst der Begriff als solcher analysiert werden, im weiteren Verlauf wird er als Eigenname für das beschriebene Vorgehen genutzt.

Domain übersetzt das Wörterbuch[77] im mathematisch-naturwissenschaftlichen Kontext als Funktionsbereich. Also genau der Bereich, in dem eine getroffene Aussage Gültigkeit besitzt. In der Mathematik ist der Funktionsbereich einer Funktion f, genau der Bereich in dem das Ergebnis von f in Abhängigkeit einer Variablen x zulässig ist. Der Funktionsbereich beschreibt somit die Menge aller zulässigen Ergebnisse einer Funktion f.[78] Bezogen auf den hier verwendeten Kontext ist der Funktionsbereich der Bereich, in dem eine Aussage wahr ist oder gilt. Dabei sind der Ursprung der Aussage sowie deren Bezug vollkommen irrelevant. Mittels eines Aussagensystems, also einer Menge von Aussagen, ist man somit in der Lage, einen Bereich abzugrenzen, in dem alle getroffenen Aussagen Gültigkeit besitzen. Dies soll im Folgenden als Domain verstanden werden. Eine etwas problembezogenere Definition von Domain liefert des Fraunhofer Institut[79]: „An area of activity or knowledge. A number of different classification schemes have been proposed for domains; some of the classes of domains that have been identifyed include: problem, application, horizontal, vertical, technology, computer sience, execution, etc.”[80] Unter einem Wissensbereich ist in etwa das zu verstehen, was weiter oben der Funktionsbereich im Kontext war. Der Begriff des Betätigungsfeldes erweitert den Domainbegriff. Somit ist eine Menge von Tätigkeiten, welche ein Ziel innerhalb eines Zielsystems zu erreichen versuchen, ebenfalls als Domain zu verstehen.

Analysis soll als Analyse beziehungsweise Gliederung verstanden werden. Auch an dieser Stelle lässt sich wieder ein Bezug zur Mathematik finden. Die Analyse einer Funktion in der Mathematik beschäftigt

---

[74] Bei gleicher Qualität und Flexibilität. Vgl. Withey 1996, S. 29.
[75] Vgl. Balzert 1998, S. 674.
[76] Vgl. Dittrich 1999, S. 875ff oder Schiffer; Templ 1999, S. 999ff.
[77] Siehe Pons 1998, S. 331, domain.
[78] Vgl. Walter 2001, S. 5.
[79] Vgl. IESE 2002.
[80] Ein Betätigungsfeld oder Wissensbereich. Es wurden eine Menge von Klassifikationsschemata für Domain vorgeschlagen. Einige der Klassen von Domain, welche identifiziert wurden beinhalten: Probleme, Anwendungen, horizontal, vertikal, Technologie, Computerwissenschaften, Ausführung, usw.

sich mit dem Finden von Kennzahlen, welche die Eigenschaften der Funktion wiederspiegeln. Eine der Kennzahlen, welche die mathematische Analyse liefert, ist die des Funktionsbereiches der Funktion. Ähnlich soll dies hier gesehen werden. Im ersten Schritt wird grob eine Domain abgesteckt, welche in der Folge analysiert wird, dabei werden Kennzahlen und Aussagensysteme zur Domain ermittelt und formuliert. Dadurch wird der Inhalt der Domain transparenter und die Abgrenzung der betrachteten Domain zu anderen wird klarer. Auch hier soll zur weiteren Erläuterung eine Definition herangezogen werden, an dieser Stelle soll die Definition nach Succi et al.[81] genutzt werden: „Domain anaysis is [..] the process of creating models to reason an predict aspects about …[a Domain]“[82]. Es handelt sich daher also um einen Prozess, der Informationen über die Domain zusammenträgt und ordnet. Auf Grundlage der so zusammengetragenen Informationen wird ein Modell der Domain gefertigt.

### 4.2.2 Das Vorgehen und die Ergebnisse

Das Verhältnis von Domäne und *Produktlinie* wird in der Literatur kontrovers diskutiert. Inhaltlich geht es um die Schwierigkeit der Abgrenzung. Die Frage lautet, inwieweit eine *Produktlinie* genau eine Domäne abdeckt, innerhalb einer Domäne einen Bereich abdeckt oder eine Menge von Domänen beinhaltet.[83] Diese Unterscheidung soll hier nicht berücksichtigt werden. Unvermeidlich ist es innerhalb der Durchführung einer Domain Analysis jedoch, die zu untersuchende Domäne genau abzugrenzen. Hierzu wird auf die Literatur verwiesen.

Als mögliche Ausgangspunkte der Domain Analysis werden in der Literatur folgende genannt[84]:

- Domain Analysis als erster Schritt der Entwicklung einer *Produktlinie*.

Dabei beginnt die Entwicklung eines Informationssystems vollkommen neu, denkbar sind solche Ansätze bei Gründungen von Unternehmen, welche ihren strategischen Rahmen formulieren. Im anschließenden Schritt wird eine Domain Analysis durchgeführt, welche den Betätigungsbereich absteckt. Dabei wird ein Top Down Vorgehen gewählt, ausgehend von der strategischen Ausrichtung des Unternehmens werden Rahmen für zukünftige Softwareprodukte formuliert. Dieser „grüne Wiese“ Ansatz sollte in der Realität weniger oft auftreten, da das Vorgehen zur Softwareentwicklung meist an den aktuellen Bedürfnissen orientiert ist. Diese sind meist das schnelle Erstellen eines vertreibbaren Produktes, dafür ist das Konzept der *Produktlinien* initial jedoch schlecht geeignet.

- Domain Analysis als begleitender Prozess bei der Entwicklung von *Produktlinien*.

Bei diesem Ansatz, wird das Domainmodel parallel zur *Produktlinie* entwickelt. Dabei handelt es sich um eine Bottom-up Analyse, da sie nicht auf einem abstrakten Level, also losgelöst von etwaigen Anwendung die Domain beschreibt, sondern anhand der erzielten Fortschritte beispielsweise bei der *Asset* Entwicklung ein Domainmodel erstellt wird. Hierbei kommt es zu ständigen Brührungen zwischen der Domain Analysis und der Assetentwicklung. Beide Prozesse laufen parallel und sind konkurrierend, komplementär oder neutral zueinander. Bei einem derartigen Vorgehen gestaltet sich die organisatorische Implementierung mitunter schwierig. Ziel muss es sein, das homogene Wirken interdependenter Prozesse auf die Oberziele der Organisation auszurichten.

---

[81] Vgl. Succi et al. 2000.

[82] Der Prozess des Sichtbar machen, ordnen und modellieren der Informationen zu einer Domain auf einer Abstraktionsebene, welche oberhalb des ausführbaren Codes liegt.

[83] Vgl. Schmid 2000.

[84] Vgl. Bass et al. 1998, S. 14.

- Domain Analysis auf Grundlage eines existierenden Systems.

Hierbei wird davon ausgegangen, dass ein vorhandenes System auf der Grundlage von *Produktlinien* umgesetzt werden soll. Das vorhandene System, welches die zukünftigen Aufgaben des zu entwickelnden Systems bisher übernommen hat, liegt vor und kann zur Weiterentwicklung genutzt werden. In diesem Szenario liegt die Hauptaufgabe der Domain Analysis in der Sichtbarmachung von Wiederverwendungspotentialen innerhalb des alten Systems. Die dort ausgelösten Teile sollen im Zuge der Assetbereitstellung überarbeitet und der *Asset Base* hinzugefügt werden. Somit werden sie für zukünftige Projekte nutzbar gemacht. Eine Annahme dieser Form der Analyse ist, dass ein Domainmodell, wenn nicht explizit, dann doch wenigstens implizit im Altsystem vorliegt. Dieses stammt aus dem Wissen, welches das Altsystem implementiert. Zur Formulierung eines Domainmodells ist somit das Gegenstromverfahren denkbar, welches parallel zur Entwicklung der *Produktlinie* abläuft. Dabei werden die Erfordernisse des Marktes untersucht und dementsprechende Strategien entwickelt. Außerdem werden Potentiale aus dem Altsystem isoliert und als *Assets* zur Verfügung gestellt.

Ziel in jedem dieser Szenarien ist es, ein Domainmodel zu erstellen. Unabhängig von den äußeren Gegebenheiten, die zur Domain Analysis vorliegen, werden verschiedene Arten der Domain Analysis unterschieden. Im Allgemeinen gibt es eine Unterscheidung in Ad-Hoc, opportunistisch, systematische und *Produktlinien* Formen[85]. Die Ad-Hoc Analyse ist meist eine Wiedergabe persönlicher Meinungen eines Experten der Domain. Bei der opportunistischen Analyse wird die Domain meist sehr eng beschrieben. Oftmals wird lediglich das Umfeld einer Applikation berücksichtigt und diese nur im Hinblick auf einen Kunden. Die systematische Analyse ist tendenziell darauf ausgerichtet, Domains zu analysieren, welche mehrere Systeme und mehrere Kunden beinhalten und somit durchaus ein Umfeld einer *Produktlinie* bilden können. Dieser Ansatz setzt an den Einflüssen der Lebenszyklen der einzelnen Artefakte an und untersucht deren Einfluss auf aus diesen Artefakten zusammengesetzte Produkte. Die Domain Analysis, welche für *Produktlinien* verwendet wird, ist ebenfalls auf mehrere Applikationen und Kunden ausgerichtet. Einen tragenden Bestandteil einer solchen Analyse bilden Marktanalysen sowie Geschäftspläne. Somit ist diese Art der Domain Analysis am stärksten am Markt orientiert. Im Folgenden sollen drei konkrete Verfahren kurz dargestellt werden. Insgesamt sind fünf Verfahren mit kurzen Abrissen zum Inhalt bei Clements et al.[86] zu finden. Weitere Erläuterungen und tiefergehende Betrachtungen zu den jeweiligen Verfahren finden sich in den dort ausgewiesenen Literaturquellen:

- Scope, commonality and variability[87]:

Stellt ein systematisches Instrument zur Beschreibung von Domains dar. Es stellt Unterschiede und Gemeinsamkeiten gegenüber. Es stammt aus dem Hause Lucent Technologies und ist Bestandteil des FAST Ansatzes[88]. Der FAST Prozess betrachtet die beiden Lebenszyklen des Domain Engineering sowie des Application Engineering. Dabei nutzt es die Ergebnisse der Analyse der Gemeinsamkeiten um Domain Mitglieder zu spezifizieren und neue Mitglieder aus vorhandenen Spezifikationen zu generieren.

- Domain Analysis and Design Process[89]:

---

[85] Vgl. Maymir-Ducharme 1997.
[86] Vgl. Clements; Northrop 2001, S. 145ff.
[87] (Umfang, Gemeinsamkeiten und Unterschiede) Vgl. Cuka; Weiss 1997.
[88] Vgl. Weiss 1995.
[89] Vgl. DoD 1992.

Dieser Ansatz basiert auf der objektorientierten Analyse und dem objektorientierten Entwurf. Auch bei dieser Art der Domain Analysis werden Gemeinsamkeiten gesucht und es werden Anforderungen an Objekte formuliert, welche genutzt werden könnten.

- Feature-oriented domain analysis[90]:

Im Zentrum dieser Methode werden die Funktionen von Software, die ein Nutzer von Software aus dieser Domain fordert, berücksichtigt. Anhand dieser werden Produkte und Prozesse definiert. Als Ergebnis liegt neben dem Modell der Domain eine Beschreibung der Beziehungen zu anderen Domains vor. Des Weiteren werden die gemeinsamen und die unterschiedlichen Funktionen der Produkte einer so analysierten Domain dargestellt.

Inhaltlich ist das angestrebte Ergebnis der Domain Analysis das sogenannte Domainmodel. Dieses ist ein Aussagensystem über die Domäne, deren Struktur darin erläutert werden soll. Dieses Aussagensystem soll den Umfang der *Produktlinie* abstecken. Diese Begrenzung geht in den sich anschließenden Prozess der Entwicklung der *Assets* ein. Des Weiteren werden Unterschiede und Gemeinsamkeiten der verschiedenen Mitglieder einer *Produktlinie* dokumentiert, diese ermöglichen es, in der folgenden Phase eine Anforderung bezügliche der Flexibilität der *Assets* zu formulieren. In welcher Form die genannten Ergebnistypen vorliegen, ist kaum spezifiziert und höchstens in konkreten Ausprägungen von *Produktlinien* definiert. Ein recht gutes Beispiel für eine *Produktlinie* im Allgemeinen sowie für Ergebnistypen der Domain Analysis im Speziellen liefert Clements et al.[91] Dort sind folgende drei Ergebnistypen spezifiziert:

- Domain Definition

Dieses Dokument stellt die *Produktlinie* überblicksartig dar. Es beschreibt die Funktionalität von verwandten Systemen und formuliert die Anforderungen und den Umfang der *Assets*, welche zum Einsatz kommen sollen und daher innerhalb der *Asset Base* zur Verfügung gestellt werden müssen. Der Umfang wurde anhand der Aufgaben, Ziele und der System Charakteristik abgeleitet.

- Anforderungsspezifikation

Hier werden die Anforderungen an die einzelnen Produkte der *Produktlinie* dokumentiert, dabei sind nicht für jedes Produkt Anforderungen spezifiziert, sondern es finden sich vielmehr allgemeine Anforderungen, welche an jedes Mitglied der *Produktlinie* gleichermaßen gestellt werden. Beispielsweise werden Fähigkeiten der Produkte dargestellt, welche in den Funktionsumfang integriert sein müssen. Somit wird jede Anforderung einzeln aufgeführt, beschrieben und in eine Struktur von Anforderungen eingeordnet. Welche Abweichungen zwischen den Produkten bezüglich gewisser Anforderungen bestehen, wird mittels verschiedener Varianten dargestellt, die jedoch in ihren Einzelheiten innerhalb der Domain Definition spezifiziert sind.

- Domain Spezifikation

In Abgrenzung zur Domain Definition, welche den Umfang der *Produktlinie* dokumentiert, werden in diesem Dokument die internen Strukturen der Domain näher erläutert. Dies wird mittels geeigneter Notationen unterstützt. So werden beispielsweise Anwendungsfälle mittels Use-Case Diagrammen der UML spezifiziert. An dieser Stelle finden wiederum die Unterschiede zwischen den einzelnen Produkten Berücksichtigung und sind durch Verweise in die Domain Definition dokumentiert.

---

[90] Funktionsorientierte ~

[91] Vgl. Clements et al. 2001.

## 4.3 Produktlinien und Architektur

Die Architektur ist ein wesentlicher Bestandteil einer *Produktlinie*. Dies begründet sich in der Tatsache, dass alle Produkte die wesentlichen Züge der Architektur teilen müssen. Da in einer Domain Analysis bzw. bei der Bereitstellung der *Assets* nur innerhalb eines begrenzten zeitlichen Rasters geplant werden kann müssen auch zukünftige, noch nicht geplante Produkte diese Architektur nutzen. Ein Wechsel der Architektur innerhalb einer *Produktlinie* ist sehr schwierig. Da alle Produkte eine gemeinsame Architektur haben sollen, müsste die Architektur vorhandener Produkte ebenfalls umgestellt bzw. angepasst werden. Ein weiterer Grund, welcher die Bedeutung der Architektur unterstreicht ist die Bereitstellung des Rahmens, welcher es Komponenten eines Produktes ermöglicht, miteinander zu kommunizieren.

Außer der großen Bedeutung der Architektur, stellt sich ein weiteres Problem. Dieses liegt in der Tatsache begründet, dass viele Faktoren die Gestaltung der Architektur beeinflussen. Ähnlich einer Komponente wird an sie die Anforderung gestellt abgeschlossen zu sein. Im Unterschied zu einer Komponente jedoch muss sie vielseitige Dienste übernehmen und koordinieren, die teilweise zur Zeit der Entwicklung noch nicht bekannt sind, daher muss die Architektur leicht erweiterbar sein. Weitere Einflüsse liegen in den Anforderungen an die Produkte einer *Produktlinie*, welche die Architektur alle berücksichtigen muss. Eine Komponente hingegen braucht nur die auf Ihre Funktionalität bezogenen Anforderungen zu erfüllen. Weitere Anforderungen sind Wartbarkeit, Benutzbarkeit und Antwortzeitverhalten.[92]

Der 8. Workshop in Software Reuse (WISR8)[93] unterschied bezüglich der Anforderungen an die Architektur bei etwaig notwendigen Anpassungen folgende 3 Fälle. Die Änderung betrifft im ersten Fall eine Komponente. Im zweiten Fall sind mehrere Komponenten betroffen und letztlich im dritten die gesamte Architektur. Die Forderung der Arbeitsgruppe ist, dass alle Änderungen an einem Produkt der *Produktlinie* stets in die zwei ersten Klassen fallen sollen. Wenn dies nicht gewährleistet werden kann, soll eine Architektur entworfen werden, welche die Kosten für Veränderungen minimal hält. Die Arbeitsgruppe warnt in diesem Zusammenhang vor einem „Abdriften“ der Architektur, durch Veränderung der tragenden Komponenten innerhalb einzelner Anwendungen.

Um die Architektur gewinnbringend in eine *Produktlinie* einzubringen, sind der *Produktlinie* vorausgehende (Up-Front -) und erhaltende (sustaining investemnts) Untersuchungen der Architektur notwendig.[94] In die erste Gruppe von Untersuchungen ist die Klärung von grundlegenden Fragen vorzunehmen. Insbesondere, wie soll die Architektur genutzt werden oder inwieweit wird die Architektur spezifiziert. In die zweite Gruppe fallen in erster Linie Wartungsarbeiten speziell für die *Assets* sowie die Produkte. Diese sollen sicherstellen, dass die Architektur nach den vorgegebenen Prinzipien genutzt wird.

# 5 Anwendungsentwicklung

## 5.1 Nutzung der Komponenten zur Entwicklung

Softwareprodukte werden praktisch nach allen Vorgehensmodellen auf der Basis von Anforderungen entwickelt. Ebenso ist dies auch hier der Fall. Beim Konzept der *Produktlinie* gibt es zwei Quellen,

[92] Vgl. Cohen 2000.
[93] Vgl. Clements 2000, S. 7.
[94] Vgl. Cohen 2000.

welche Anforderungen definieren. Zum einen ist das der Kunde bzw. der Markt, von dort stammen die Anforderungen, welche direkt an das Produkt gestellt werden. Dies sind zumeist funktionelle Aspekte der fachlichen sowie der technischen Ebene. Eine zweite Quelle von Anforderungen ergibt sich aus der *Produktlinie* selber. Dies sind zum einen die Anforderungen, welche innerhalb der Domain Analysis gefunden wurden und zum anderen Anforderungen aus dem Konzept selber. Eine dieser letztgenannten Anforderungen ist beispielsweise, dass das Produkt aus vorhandenen bzw. neu zu entwickelnden Komponenten bestehen muss. Anforderungen aus der Domain Analysis sind beispielsweise der Umfang der *Produktlinie*. Das neu zu entwickelnde Produkt muss diesem genügen. Wenn dies nicht der Fall ist, wird das Produkt entweder einer etwaig vorhandenen anderen *Produktlinie* zugeordnet oder der Umfang der vorliegenden *Produktlinie* muss verändert werden. Ein weiterer Input ist der Produktionsplan; dieser im Rahmen der Domian Analysis definierte Bauplan gibt die Zusammenstellung des Produktes aus den vorhandenen *Assets* vor.

Innerhalb der Anwendungsentwicklung ist der Punkt der tatsächlichen Wiederverwendung von Ergebnissen erreicht. Ein Prozess, der sich ebenfalls so benennen lassen würde, findet im Rahmen der Assetentwicklung statt. Dort muss überlegt werden, welcher *Asset* in die *Asset Base* aufgenommen wird. Dies wird an dieser Stelle nicht betrachtet.

Die Auswahl eines *Assets* stellt sich als ein gerichteter Suchprozess dar. Dabei sind drei Szenarien denkbar:

- Keine Implementierung vorhanden[95]:

Der ungünstigste Fall, der bei der produktlinienorientierten Softwareentwicklung eintreten kann. In diesem Fall ist der Softwareentwickler gezwungen, die benötigte Funktionalität selbst bereitzustellen. Dabei sind die Umstände, die dazu geführt haben, dass keine Implementierung vorhanden ist, zu berücksichtigen. Bei jungen *Produktlinien* ist es möglich, dass das benötigte Ergebnis noch nicht umgesetzt wurde. Ebenfalls denkbar wäre, dass der Softwareentwickler ein Ergebnis sucht, welches bisher noch nicht erreicht wurde, beispielsweise ein neues Feature. In beiden Fällen wird das Problem in die Zuständigkeit der Assetentwicklung[96] gegeben. Eine dritte Möglichkeit ist, dass Implementierungen vorhanden sind, jedoch nicht genutzt werden können, da sie jeweils an ein bestimmtes Produkt der *Produktlinie* angepasst sind. Dies ist ein Indikator dafür, dass im Rahmen der Assetentwicklung keine ausreichend flexible Lösung gefunden wurde[97] und daher dieses Ergebnis für jedes Produkt einer *Produktlinie* individuell entwickelt wird. Durch ein solches Vorgehen werden die Mechanismen der *Produktlinie* außer Kraft gesetzt, somit ist es nur in Ausnahmefällen anzuwenden[98].

- Eine Implementierung vorhanden[99]:

Der günstigste Fall, dabei ist genau eine Komponente in der *Asset Base* vorhanden, welche die angebotene Funktionalität in allen Produkten der *Produktlinie* bereitstellt. Solche Lösungen realisieren den Kernbestandteil der Wiederverwendung[100] innerhalb einer *Produktlinie*. Derartige Assets zeichnen

---

[95] Vgl. Bosch 2000, S. 268.
[96] Vgl. Varinate E in Halsmans et al. 2001, S. 6/4.
[97] Vgl. Varinate C in Halsmans et al. 2001, S. 6/4.
[98] Vgl. Bosch 2000, S. 269.
[99] Vgl. Bosch 2000, S. 269.
[100] Vgl. Varinate A in Halsmans et al. 2001, S. 4.

sich meist durch eine hohe Flexibilität aus. Diese kann aus der Art der zu behandelten Aufgabe oder aus der qualitativ anspruchsvollen Implementierung stammen.

- Mehr als ein Implementierung vorhanden[101]:

In den meisten Fällen existieren Unterschiede zwischen den Mitgliedern einer *Produktlinie*. Häufig sind die sich ergebenden Unterschiede in den Anforderungen so stark, dass diese nicht für alle Produkte einer *Produktlinie* durch nur eine gemeinsam genutzte Komponente befriedigt werden können. Somit existieren in der *Asset Base* Ergebnisse, welche inhaltlich gleich sind jedoch in Ihren Eigenschaften auf bestimmte Produkte angepasst sind. Hier sind alle, in Halsmans[102] et al., genannten Lösungsszenarien denkbar.

In allen oben dargestellten Szenarien kann eine weitere Handlungsalternative Berücksichtigung finden. Dabei wird die Architektur als die Schlüsselkomponente der *Produktlinie* verändert[103]. Da dieses Vorgehen nicht nur das aktuelle Problem fokussiert, sondern auch Auswirkungen auf alle anderen Mitglieder der *Produktlinie* zeigt, ist ein solches Vorgehen nur dann anzuwenden, wenn keines der anderen Szenarien zu einer befriedigenden Lösung führt oder durch die Architekturanpassung Potentiale zur weitreichenden Verbesserung der gesamten *Produktlinie* gesehen werden.

Im Ergebnis eines Durchlaufes der Anwendungsentwicklung liegt ein Produkt vor. Des Weiteren sind unter den oben genannten Umständen diverse Veränderungen an den Ergebnissen der Domain Analysis aufgetreten. Diese werden anschließend in die *Produktlinie* eingepflegt.

## 5.2 Die Evolution einer Produktlinie

Die Evolution einer *Produktlinie* ist von der Fortentwicklung der einzelnen *Assets* sowie der Architektur gekennzeichnet. Folgende Grafik verdeutlicht das Problem.

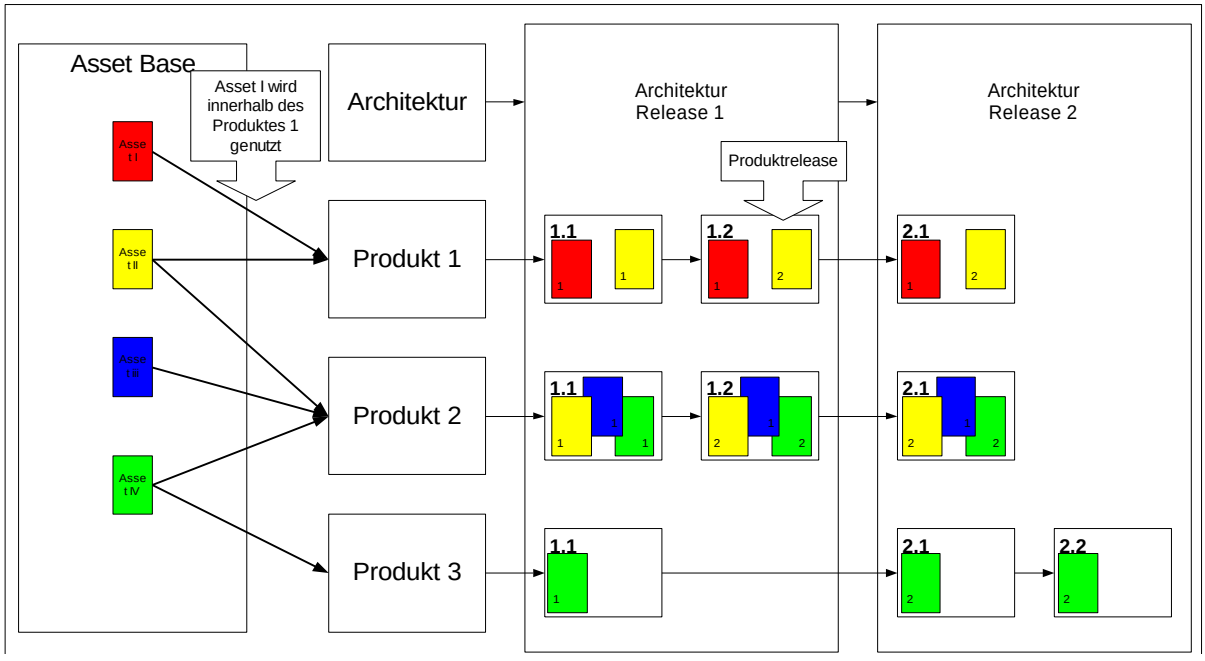


**Abbildung 3: Evolution einer Produktlinie[104]**

[101] Bosch 2000, S. 269.
[102] Vgl. Varianten A bis F in Halsmans et al. 2001, S. 6/4f.
[103] Vgl. Varinate F in Halsmans et al. 2001, S. 6/4.
[104] In abgeänderter Form übernommen aus Bass et al. 1997, S. 15.

In Abbildung 2 wird auf die Betrachtung weiterer *Assets* außer der Architektur verzichtet, da sonst die Übersichtlichkeit leiden würde. Dennoch müssen in der Realität nicht nur die verschiedenen Releases der Architektur berücksichtigt werden, sondern auch die Entwicklungen von anderen *Assets* sind für die Ausgestaltung einzelner Produkte interessant. Hier wird die Problematik, welche die Versionierung einer *Produktlinie* erschwert, deutlich. Um die Skaleneffekte, welche durch eine *Produktlinie* erzeugt werden voll auszunutzen, ist es unbedingt erforderlich, dass ein Produkt zum Zeitpunkt x, in dem es erweitert werden soll, auf den jeweils aktuellen Versionen der verwendeten *Core Assets* arbeitet. Erforderlich wird dies, da jeweils nur die aktuellen Releases einer Wartung, Pflege und Weiterentwicklung unterworfen sind. Ziel einer *Produktlinie* ist es jedoch, die Wartungsarbeiten an einem *Core Asset* allen Produkten zugute kommen zu lassen.[105] Diese Forderung erschwert die Wartung von *Produktlinien* aus zwei Gründen[106]: (1) eine Änderung kann nicht als die Anpassung einer Komponente an ein Problem verstanden werden, sondern muss immer den Anforderungen der *Produktlinie* entsprechen. (2) Bevor eine Änderung eins Produktes vorgenommen werden kann, muss es auf die aktuellen Versionen der verwendeten *Assets* angepasst werden. Bisher wurde lediglich der Fall betrachtet, dass eine neue Funktion der *Produktlinie* hinzugefügt wurde oder die *Produktlinie* einer allgemeinen Wartung unterzogen wird. Diese Arten der Evolution einer *Produktlinie* sollen nun konkretisiert werden. In Anlehnung an Bosch[107] werden hier sechs Arten unterschieden:

- Umsetzen von Qualitätsanforderungen:[108]

Diese Änderungen betreffen nur ein *Asset* somit werden die notwendigen Änderungen vorgenommen und das veränderte *Asset* wird der *Asset Base* hinzugefügt. Im weiteren Verlauf wird das veränderte *Asset* in die vorhandenen, das *Asset* benutzenden, Produkte integriert.

- Neue Plattform:

Hier muss unterschieden werden, ob das Produkt lediglich auf eine neue Plattform „gehoben" wird, oder ob das Produkt zwei Plattformen unterstützen soll. Der erste Fall ist in der Handhabe ähnlich dem der Umsetzung von Qualitätsanforderungen. Der zweite Fall kann unter Umständen in einem neuen Produkt oder einer neuen *Produktlinie* gipfeln.

- Erweiterung der Funktionalität:[109]

Unter gewissen Umständen ist es sinnvoll, gewisse Funktionalität nicht von Anafang an in vollem Umfang zur Verfügung zu stellen, sondern nach und nach aufzubauen.[110] Die dazu notwendigen Änderungen sind ähnlich denen der Umsetzung neuer Qualitätsanforderungen.

- Neue Funktionalität hinzufügen:

---

[105] Vgl. Bass et al. 1997, S. 17.

[106] Vgl. Clements 1997, S. 9.

[107] Vgl. Bosch 2000, S. 284ff.

[108] Vgl. Abbildung 3: Evolution einer Produktlinie: Der Wechsel vom ersten zum zweiten Release der Produkte 1 und 2 innerhalb des ersten Architekturrelease kann eine solche Änderung beinhalten, dabei wurde lediglich eine Komponente verändert und in Produkt 1 und 2 eingepflegt.

[109] Vgl. Abbildung 3: Evolution einer Produktlinie: Der Sprung von einem Architekturrelease zum anderen stellt eine derartige Änderung dar.

[110] Dies ist beispielsweise der Fall wenn die Time-to-Market (wg. Konkurrenzdruck o.ä.) verkürzt werden soll, dann kann unter Umständen im ersten Schritt eine Minimalversion veröffentlicht werden, welche nach und nach weiter entwickelt wird.

Dies ist die häufigste Form des Eingriffes in eine *Produktlinie*. Auch hier ist meist nur ein *Asset* betroffen, daher ist die Änderung ähnlich der bei der Umsetzung neuer Qualitätsanforderungen.

- Ein neues Produkt aufnehmen:[111]

Dabei können diverse Probleme auftreten. Oftmals lassen sich diese durch Hinzufügen einer Komponente oder durch das Ergänzen von Funktionalität abdecken. In schweren Fällen müssen vorhandenen *Assets* generalisiert werden. Das heißt, sie bieten im Moment Funktionalität an, die zu stark auf die Bedürfnisse vorhandener Produkte ausgelegt ist. Eine derartige Generalisierung kann weitreichende Auswirkungen haben, da die veränderte Komponente ebenfalls in vorhandene Produkte integriert werden muss.

- Eine neue *Produktlinie* erzeugen:

Einige Änderungen machen es notwendig neue *Produktlinien* einzuführen. So beispielsweise die Unterstützung einer weiteren Plattform. Derartige Erweiterungen stellen die stärkste Art der Veränderung einer *Produktlinie* dar. In solche Situationen sind neuerlich Entscheidungen zu treffen, welche in den Abschnitten 4, 4.2 und 4.3 näher erläutert werden.

# 6 Management

Die hier subsumierten Aktivitäten haben die Aufgabe, die benötigten Ressourcen zur Verfügung zu stellen und die Aktivitäten zu koordinieren und zu überwachen. Dazu werden zwei Formen des Managements unterschieden.

Das organisatorisch Management oder Projektmanagement hat die Aufgabe, die Ablauf- und die Aufbaustrukturen in einem Unternehmen derart zu konfigurieren, dass diese das Konzept der *Produktlinien* tragen. Nach Clements et al.[112] Sind folgende Aufgaben innerhalb des Projektmanagements zu bewältigen:

- Überwachung, dabei wird sichergestellt, das die organisatorischen Rahmenbedingungen den Anforderungen einer *Produktlinie* gerecht werden.
- Einbeziehung der Nutzer, diese Aktivität versucht eine Schnittstelle zwischen den Nutzern und den Entwicklern einer *Produktlinie* zu schaffen.
- Entwurf einer Beschaffungsstrategie, dabei werden strategische Richtlinien formuliert, welche die Beschaffung von *Assets* betreffen. Diese werden dann als Vorgaben im Prozess der Komponentenentwicklung genutzt.
- Finanzierung, dabei werden die Aktivitäten im Rahmen der *Produktlinie* hinsichtlich der Finanzierbarkeit überprüft sowie geeignet gesteuert.
- Weiterentwicklung der Organisationsstruktur, Ziel dieser Aktivität ist es die Organisationsstruktur schrittweise stärker auf das Konzept der Wiederverwendung auszurichten.
- Marktanalysen zur systematischen Untersuchung der externen Faktoren, welche den Erfolg eines Produktes/ einer *Produktlinie* beeinflussen.

---

[111] Vgl. Abbildung 3: Evolution einer Produktlinie: So geschehen mit Produkt zwei, in diesem Zuge mussten Komponenten angepasst werden, somit wechselten die Produkt 1 und 3 in ein neues Release.

[112] Vgl. Clemens; Northrop 2001.

- Steuerung, zentrales Feld ist hierbei die Ablauforganisation, dazu werden Schnittstellen zwischen den einzelnen organisatorischen Einheiten geschaffen.
- Organisation der einzelnen Projekte, dabei steht das Portfolio an aktuellen und zukünftigen Projekten im Mittelpunkt, die einzelnen Projekte werden zeitlich in eine Reihenfolge gebracht, welche unter zu definierenden Bedingungen optimal ist.
- Risikomanagement
- Schaffung von festen Strukturen innerhalb des Unternehmens.
- Überwachung der Technologie. Dazu werden im Unternehmen genutzte Technologien ständig in Frage gestellt, um auf dem neusten Stand zu bleiben.
- Personaltraining, um Personalbedarf bei der Entwicklung schnell und effizient zu decken.

Diese Aktivitäten sind solche, die den Rahmen für eine produktlinienorientierte Softwareentwicklung bereitstellen. Es sind Tätigkeiten die zur Schaffung von Randbedingungen notwendig sind, welche den Erfolg der *Produktlinie* unterstützen.

Das technische Management subsumiert Aktivitäten, welche die Entwicklung von Produkten und von Komponenten maßgebend beeinflussen. Dabei gilt es beide Prozesse so gut wie möglich zu unterstützen um deren Erfolg zu gewährleisten. Die wichtigsten Aktivitäten im Rahmen des technischen Managements sind:

- Konfigurationsmanagement
- Datensammlung, Leistungsmessung und –verfolgung
- Festlegung von Abläufen innerhalb der Komponenten- und Anwendungsentwicklung.
- Technische Planung
- Technische Risikoanalyse
- Werkzeugunterstützung

Diese Aktivitäten besitzen alle eine direkte Schnittstelle zur Entwicklung. Dies verdeutlicht folgende Grafik.

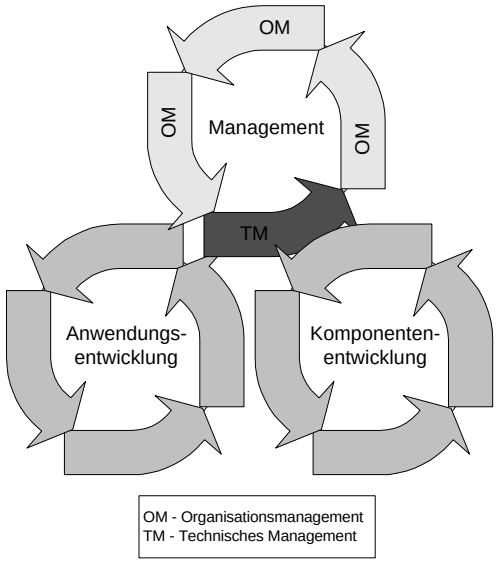


**Abbildung 4: Management einer Produktlinie**

# 7 Bewertung

## 7.1 Bewertung anhand der Ziele der Wiederverwendung

Eine Bewertung soll anhand der unter 1 definierten Kriterien erfolgen.

- Senkung des mengenmäßigen Einsatzes zur Erzielung des im Rahmen des Projektes definierten Erfolges.

Diese Forderung ist eine Implikation aus der Forderung einer höheren Produktivität. Im Konzept der *Produktlinien* geht es im Kern um die Wiederverwendung von Ergebnissen des Softwareentwicklungsprozesses. Dabei geht es über die alleinige Nutzung des einmal erzeugten Codes hinaus. Dies hat es mit einigen anderen wiederverwendungsorientierten Arten der Softwareentwicklung gemein. Allerdings wird im betrachteten Konzept noch ein weiterer wesentlicher Punkt berücksichtigt. Entgegen der Entwicklung von Komponenten, die soweit als sinnvoll von der Domäne der Entwicklung abstrahieren, versucht das Konzept der *Produktlinie* die Domäne so stark als möglich mit einzubinden. Die Forderung der stärkeren Abstraktion resultiert aus der Annahme der komponentenbasierten Anwendungsentwicklung, dass derartige Software eher wiederverwendet werden kann, da etwaige Integrationsprobleme zwischen zwei Domänen minimiert werden. Davon gehen die *Produktlinien* weg. Im Zentrum der Betrachtung steht die Anwendungsdomäne. Umzusetzende Restriktionen bezüglich einer Wiederverwendung müssen nur in ihrem Kontext betrachtet werden.

Ein weiterer Vorteil liegt in der Fülle von Komponenten. So werden Produkte ausschließlich aus Komponenten zusammengesetzt. Unter der Annahme, dass die *Asset Base* diesbezügliche Qualitäten einräumt, kann von einer kürzeren Entwicklungszeit ausgegangen werden. Das Argument, dass Entwicklungszeit von der Anwendungsentwicklung in die Komponentenentwicklung verlagert wurde, ist durchaus gerechtfertigt. Dennoch ist es der Organisation möglich, schneller auf Marktveränderungen zu reagieren. Dies resultiert aus der Annahme, dass neue Produkte nur in wenigen Aspekten von alten verschieden sind, somit sind lediglich die neuen Aspekte in Form von Komponenten der *Asset Base* hinzuzufügen. Die Anwendungsentwicklung besteht dann nur noch aus der Kombination der benötigten Komponenten.

Daher ist die *Produktlinie* in der Lage, Ziele der Softwareentwicklung im Sinne der Anforderung produktiver zu erreichen.

- Sicherung der Qualität der verwendeten Ergebnisse.

Im Rahmen der Domain Analysis werden die Anforderungen an die Produkte klar definiert. Da in diesem Punkt zusätzliche Kräfte aufgebracht werden, um die fachlichen Aspekte einer Domäne zu beleuchten, kann davon ausgegangen werden, dass hier Ergebnisse erzielt werden, welche qualitativ vergleichsweise hochwertig sind. Allerdings wird keine explizite Qualitätssicherung gefordert, daher muss davon ausgegangen werden, dass die ereichten Ergebnisse eine, aufgrund des Aufwandes, moderate aber keine herausragende Qualität aufweisen. An dieser Stelle ist ein Defizit des Konzeptes zu sehen.

- Aufwand zur Generierung des Architekturentwurfes sowie zur Generierung des Fachkonzeptes.

Der Aufwand, welcher in die Architektur investiert wird ist als grundsätzlich hoch einzustufen. Ursprung dieser Aussage ist, dass alle Produkte einer *Produktlinie* eine gemeinsame Architektur teilen. Dies stellt daher auch besondere Ansprüche an die Architektur, welche von denen herkömmlicher Anwendungsentwicklung abweichen und darüber hinausgehen. Ausgangspunkt des formulierten

Anspruchkriteriums war, dass die Ausbaufähigkeit des Systems gegeben sein sollte. Davon kann beim Konzept der *Produktlinie* ausgegangen werden, da ebendies ein Ziel der *Produktlinien* ist. Es soll eine Architektur geschaffen werden, welche für alle Anwendungen einer Domäne einheitlich ist – dies impliziert die Notwendigkeit der Ausbaufähigkeit. Allerdings kann diese nicht losgelöst betrachtet werden. Ausgangspunkt dieser gemeinsamen Architektur ist die Domain Analysis und das Ausmaß der dort vorhergesehenen Entwicklungen. Dies dürfte, besonders in schnelllebigen Märkten, eingeschränkt sein. Die Qualität des Fachkonzeptes, kann aufgrund der strategischen Ausrichtung sowie der Bedeutung innerhalb des Konzeptes als hoch angenommen werden

- Verkürzung der Entwicklungszeiten

Potentiale dazu sind durchaus vorhanden. Allerdings sind diese nur zu realisieren, wenn die Voraussetzungen stimmen. Dies betrifft zum einen die Qualität und den Umfang der *Asset Base*. Wenn diese den Anforderungen genügen und auf vorhergesehene Änderungen des Marktes reagiert werden muss, kann die Entwicklungszeit signifikant verkürzt werden. Grundlage der beiden notwendigen Voraussetzungen liegen jeweils in der Qualität der Ergebnisse, welche während der Domain Analysis gewonnen wurden.

- eine Reduzierung der Entwicklungskosten

Unter denen im vorangegangen Punkt formulierten Einschränkungen kann davon ausgegangen werden. Im Folgenden sollen einige Probleme der *Produktlinien* kurz umrissen werden.

Das Konzept der *Produktlinie* erlaubt keine Anpassungen an die Ansprüche eines einzelnen Nutzers. Zu begründen ist diese Aussage im Rahmen des Versionsmanagements. Laut den Voraussetzungen einer *Produktlinie* besteht ein Produkt nur aus Assts der *Asset Base*. Das heißt eine Änderung in einem Produkt hätte immer eine Änderung in einem *Asset* zur Folge. Dies wiederum würde bedeuten, dass alle anderen Produkte, welche dieses *Asset* nutzen, zum nächst möglichen Termin auf das neue Release des *Assets* umgestellt werden müssten. Da die Änderungen in diesem Szenario jedoch lediglich auf die Bedürfnisse eines Kunden bezüglich eines bestimmten Produktes angepasst sind, ist es möglich, dass diese Änderungen an anderer Stelle schadhaft wirken. Die Alternative für solche Fälle, zwei *Assets* zur Verfügung zu stellen, dürfte recht schnell die Prämissen der *Produktlinien* sprengen. Da in diesem Fall die Wartung zweier *Assets* nötig wäre, dies würde die notwendige Arbeit verdoppeln. Im Resultat kann auch bei dieser Einschränkung gesagt werden, dass *Produktlinien* auf Märkten in denen so etwas vorkommt, kaum geeignet sind. Dem Autor sind auch keine Anwendungsbereiche bekannt, in denen eine *Produktlinie* vor derartige Anforderungen gestellt wurde.[113] Daraus kann man schließen, dass Produkte auf Basis von *Produktlinien* für nur einen Nutzer entwickelt werden können. Dies ist beispielsweise der Fall, wenn die Software im Unternehmen selbst genutzt wird. Oder sie werden lediglich für einen Markt entwickelt, wobei hier etwaige Großkunden mit speziellen Anforderungen nicht berücksichtigt werden können. Dies stellt eine starke Einschränkung des Einsatzbereiches dar.

Ebenfalls problematisch ist die Annahme einer Architektur, welche durch alle Produkte einer *Produktlinie* geteilt wird. Darin ist ein Wiederspruch zu einer Aussage zu sehen, welche in Abschnitt 2.2 Framework getroffenen wird. Die Architektur ist ein Bestandteil der *Asset Base*. Komponenten, welche in der *Asset Base* liegen, *können* zur Erstellung von Produkten genutzt werden, die Architektur hingegen *muss* genutzt werden. Daher ist das Konzept der *Produktlinien* dem Ansatz der regkonfigurierbaren Architektur doch

.

näher, als das von den Vertretern eingestanden wird. Die mit dem Konzept der regkonfigurierbaren Architektur verbundenen Nachteile wie Anfälligkeit gegenüber neuen Technologien oder gegenüber sich wandelnden Anforderungen können daher in ähnlicher Form auch auf *Produktlinien* Anwendung finden.

Ein weiteres Problem betrifft das Auffinden von Komponenten. Die bei der Entwicklung von Produkten verwendeten Komponenten, befinden sich in der *Asset Base*. Das Auffinden solcher Komponenten stellt sich als wenig schwierig dar, da die Anzahl aller verfügbaren Komponenten dadurch eingegrenzt wird, dass die verwendbaren Komponenten aus der *Asset Base* stammen müssen. Jedoch findet das Problem des Auffindens von Komponenten zur Schaffung der *Asset Base* keine Betrachtung. Vermutlich ist das hier auftretende Problem noch schwieriger zu lösen als bei der Beschaffung von Komponenten bei nur einem Produkt. Bei einem Produkt ist es vergleichsweise einfach, eine Komponente zu finden, welche lediglich durch Parametrisierung in der Lage ist, die an sie gestellten Anforderungen zu erfüllen. Bei *Produktlinien* ist das Problem ungleich schwerer zu lösen. Dies begründet sich in der Tatsache, dass Komponenten nicht nur mittels Parametrisierung an die Bedürfnisse eines Programms angepasst werden müssen. Vielmehr muss mit einer Komponente auf die Bedürfnisse vieler Programme reagiert werden können.

## 7.2 Bewertung des Konzeptes

Dieser Abschnitt soll sich an einigen der unter 3.2 Produktlinien – eine Definition herausgearbeiteten Differenzierungsmerkmalen orientieren. Dazu soll die Güte der Umsetzung der einzelnen Differenzierungsmerkmale beurteilt werden.

- Geplante Wiederverwendung.

Dieses Merkmal zielt darauf ab, dass Softwarekomponenten nicht auf ein spezielles System zugeschnitten sind, sondern in einer Anzahl von Systemen eingesetzt werden können. Auf diese Prämisse wird der Softwareentwicklungsprozess ausgerichtet. Derartige Bestrebungen sind im Konzept der *Produktlinien* zu erkennen. So werden Komponenten innerhalb einer *Asset Base* gehalten, welche im Rahmen verschiedenster Produkte eingesetzt werden. Diese *Assets* werden dabei in einem speziellen Prozess, der Komponentenentwicklung, auf die Bedürfnisse einer Menge von Produkten, hin entwickelt. Daher ist das Merkmal der geplanten Wiederverwendung in diesem ersten Schritt erfüllt. Fraglich bleibt, ob das Konzept der geplanten Wiederverwendung ohne Einschränkungen gültig ist. So zielen beispielsweise die komponentenbasierten Anwendungssysteme eher auf eine globale Wiederverwendung ab. Dabei gilt es Komponenten in verschiedenen Kontexten wiederzuverwenden. Um dies zu gestatten, wird von einer Beschaffung von Komponenten auf Märkten ausgegangen. Diesem Konzept genügt das der *Produktlinien* nicht, die im Rahmen der Aktivitäten erstellten Komponenten sind speziell auf die Bedürfnisse der Domäne abgestimmt. Dass ähnliche Domänen existieren, wird vernachlässigt. Somit ist die Wiederverwendung innerhalb der Domäne geplant, sonstige Wiederverwendung wird nicht berücksichtigt.

- Nutzung vielfältiger Ergebnisse des Entwicklungszyklus.

Diese Forderung scheint logisch, so werden nicht nur ausführbare Codestücke genutzt, sondern auch Erkenntnisse, welche in Form von Dokumenten vorliegen, in die Entwicklung mit einbezogen. Innerhalb des Konzeptes der *Produktlinien* ist in den meisten Fällen von Code die Rede. Auch wenn der Begriff *Asset* regelmäßig alle Arten von Ergebnissen des Softwareentwicklungsprozesses meint, findet innerhalb der Ausführungen verschiedener Autoren meist der Code spezielle Betrachtung. Ein spezielles

Wissensmanagement, welches obiges leisten würde ist nur in Ansätzen vorhanden und noch ausbaubedürftig.[114]

- Berücksichtigung der Wiederverwendung bereits bei der Entwicklung von Komponenten

Dieses Kriterium unterscheidet sich nur leicht von dem der geplanten Wiederverwendung. Hier wird speziell auf Anforderungen abgezielt, die sich nicht aus der Wiederverwendung im Allgemeinen[115] ergeben, sondern auf die Berücksichtigung der Anforderungen aller Produkte der Wiederverwendung.[116] Dies ist bei der Entwicklung von *Produktlinien* explizit gefordert. So werden die Komponenten aufgrund der Anforderungen der Domäne gestaltet. Daher erfüllen sie jeweils eine dort definierte Aufgabe. In dieser Prämisse unterscheidet sich das Konzept der *Produktlinien* maßgeblich von anderen. So werden in den meisten anderen Szenarien der Wiederverwendung Komponenten meist im Rahmen der Anwendungsentwicklung erzeugt. Eine weitere häufig gefundene Variante ist die Entwicklung von Komponenten, welche direkt auf eine Wiederverwendung ausgerichtet sind. Derartige Komponenten werden aber häufig losgelöst von etwaigen Domänen oder Anwendungen entwickelt und auf Märkten angeboten. In diesem Zusammenhang spricht man auch von COTS.

- Produkte werden als Kompositionen betrachtet, wobei die einzelnen Bestandteile einer solchen Komposition der Wartung, Pflege und Weiterentwicklung unterworfen sind. Das Produkt als solches kann nur durch Neukomposition oder Änderungen an den Bestandteilen verändert werden.

Diese Prämisse der *Produktlinien* stellt in den Augen des Autors den größten Vorteil der produktlinienorientierten Softwareentwicklung dar. Da dieses Konzept in den *Produktlinien* fest verankert ist, beinhaltet es ein großes Einsparungspotential. Nicht nur, dass Entwicklungsleistungen in Form von Komponenten wiederverwendet werden, sondern auch Aktivitäten der Wartung wirken sich in vielfältiger Weise auf die verschiedenen Produkte aus. Da die Wartung eines Informationssystems meist den größten Anteil an den anfallenden Kosten trägt, werden an dieser Stelle hohe Einsparungspotentiale eröffnet. Als Problematisch sind die Annahmen bezüglich der Architektur innerhalb einer *Produktlinie* zu werten. Es wird ausgesagt, dass die Architektur eine Komponente innerhalb der *Asset Base* ist. Diese Aussage verwirrt etwas. So ist die *Asset Base* als ein Pool zu betrachten der dem Anwendungsentwickler Werkzeuge in Form von Komponenten bereitstellt. So werden die Komponenten extrahiert, welche einen Beitrag zur Lösung des behandelten Problems leisten. Dies ist bei der Architektur nicht gegeben. Diese stellt sich vielmehr als allumfassende Komponente dar, welche der Anwendungsentwickler nutzen muss. Im optimalen Fall ist die vorhandene Architektur zur Lösung des Problems geeignet. Wenn dies jedoch nicht der Fall ist, muss eine neue Architektur der *Asset Base* hinzugefügt werden. Diese soll nun die Bedürfnisse der Anwendung eher erfüllen. Fraglich bei einem derartigen Vorgehen bleibt inwieweit im Rahmen der neuen Architektur eine Wiederverwendung der vorhandenen Komponenten erfolgen kann.

[114] Vgl. Schlick; Hein.
[115] Abgeschlossenheit, Vermarktbar ...
[116] Meint inhaltlich die Ergebnisse der Domain Analysis.

# Anhang A[117]

[1] E. Addy. Report from the First Annual Workshop on Software Architectures in Product Line Acquisitions. ACM SIGSOFT Software Engineering Notes, 23(3):32–39, May 1998.

[2] Edward A. Addy. Verification and Validation in Software Product Line Engineering. PhD thesis, College of Engineering and Mineral Resources at West Virgina University, Morgantown, West Virginia, 1999.

[3] B. Adelson and E. Soloway. The Role of Domain Experience in Software Design. IEEE Transactions on Software Engineering, 11(11):1351–1360, November 1985.

[4] O.-A. Agyapong and P. Bobbie. The Design of an Expert System for Domain Knowledge Engineering and Decision Making: A Case Study in the Criminal Justice System. International Journal of Software Engineering & Knowledge Engineering, 8(1):21–33, March 1998.

[5] J. Alexander. Knowledge Level Engineering: Ontological Analysis. In Proceedings of the fifth National Conference on Artificial Intelligence, pages 963–968. American Association for Artificial Intelligence, 1986.

[6] B. Allen and P. Holtzman. Simplifying the Construction of Domain-Specific Automatic Programming Systems: The NASA Automated Software Development Workstation Project. In Proceedings of the Space Operations Automation and Robotics Workshop, pages 407–410, Houston, TX, August 1987. NASA Johnson Space Center.

[7] Pierre America, Henk Obbink, Rob van Ommering, and Frank van der Linden. CoPAM: A Component-Oriented Platform Architecting Method Family for Product Family Engineering. In Proceedings of the First Software Product Line Conference [201], pages 167–180.

[8] Pierre America and Jan van Wijgerden. Requirements Modeling for Families of Complex Systems. In Third International Workshop on Software Architectures for Product Families [359], pages 199–209.

[9] M. Anastasopoulos and C. Gacek. Implementing Product Line Variabilities. Technical Report IESE Report No. 089.00/E, Version 1.0, Fraunhofer Institute for Experimental Software Engineering (IESE), November 2000.

[10] M. Anastatsopoulos, J. Bayer, O. Flege, and C. Gacek. A Process for Product Line Architecture Creation and Evaluation. PuLSE-DSSA Version 2.0. Technical Report IESE Report No. 038.00/E, Fraunhofer Institute for Experimental Software Engineering (IESE), June 2000.

[11] J. Angele, D. Fensel, D. Landes, and R. Studer. Developing Knowledge-Based Systems with MIKE. Automated Software Engineering, 5(4):389–418, October 1998.

[12] G. Arango. Domain Engineering for Software Reuse. PhD thesis, University of California at Irvine, 1988.

[13] G. Arango. Evaluation of a Reuse-Based Software Construction Technology. In Proceedings of the Second IEE/BCS Conference: Software Engineering 88, pages 85–92, London, UK, July 1988. IEE.

---

[117] Zusammengestellt von: Institut Experimentelles Software Engineering des Fraunhofer Institutes. http://www.iese.fraunhofer.de/PuLSE/Bibliography/SplBib_a4.pdf. Abruf: 01.06.2002

[14] G. Arango. Notes on the Application of the COBWEB Clustering Function to the Identification of Patterns ofReuse. In Fifth International Workshop on Software Specification and Design, 1988.

[15] G. Arango. Domain Analysis – From Art Form To Engineering Discipline. In Fifth International Workshop on Software Specification and Design, pages 152–159, September 1989.

[16] G. Arango. Domain Analysis Methods. In W. Shaefer, R. Prieto-Diaz, and M. Matsumoto, editors, Software Reusability. Ellis Horwood, 1993.

[17] G. Arango. Domain Analysis. In J. Marciniak, editor, Encyclopedia of Software Engineering, volume 1, pages 424–434. John Wiley & Sons, 1994.

[18] G. Arango and R. Prieto-Diaz. Domain Analysis and Software Systems Modeling, chapter Domain Analysis Concepts and Research Directions, pages 9–33. IEEE Computer Society Press, 1991. Book out of print. [19] G. Arango, E. Schoen, and R. Pettengill. A Process for Consolidating and Reusing Design Knowledge. In Proceedings of the Fifteenth International Conference on Software Engineering (ICSE'93), pages 231–242, 1993.

[20] M. Ardis and D. Weiss. Defining Families: The Commonality Analysis. In Proceedings of the Nineteenth International Conference on Software Engineering (ICSE'97), pages 649–650, May 1997.

[21] Mark Ardis, Nigel Daley, Daniel Hoffman, Harvey Siy, and David Weiss. Software Product Lines: A Case Study. Software – Practice and Experience, 30(7):825–847, June 2000.

[22] Mark Ardis, Peter Dudak, Liz Dor, Wen jenq Leu, Lloyd Nakatani, Bob Olsen, and Paul Pontrelli. Domain Engineered Configuration Control. In Proceedings of the First Software Product Line Conference [201], pages 479–493.

[23] J. Armitage. Process Guide for the Domain-Specific Software Architectures (DSSA) Process Life Cycle. Technical Report CMU/SEI-93-SR-021, Software Engineering Institute, Carnegie Mellon University, 1993.

[24] M. Asdjodi. Knowledge-Based Component Composition: An Approach to Software Reusability. PhD thesis, University of Alabama, Huntsville, 1988.

[25] Colin Atkinson, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland Laqua, Dirk Muthig, Barbara Paech, J¨ urgen W¨ ust, and J¨ org Zettel. Component-based Product Line Engineering with UML. Component Software Series. Addison-Wesley, 2001.

[26] Colin Atkinson, Joachim Bayer, and Dirk Muthig. Component-Based Product Line Development: The KobrA Approach. In Proceedings of the First Software Product Line Conference [201], pages 289–309.

[27] Colin Atkinson and Dirk Muthig. Enhancing Component Reusability through Product Line Technology. In Proceedings of the Seventh International Conference on Software Reuse [233], pages 93–108.

[28] Marko Auerswald, Martin Herrmann, Stefan Kowalewski, and Vincent Schulte-Coerne. Reliability-Oriented Product Line Engineering of Embedded Systems. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 79–96.

[29] Felix Bachmann and Len Bass. Managing Variability in Software Architecture. In Proceedings of the ACM SIGSOFT Symposium on Software Reusability (SSR'01), pages 126–132, May 2001.

[30] S. Bailin. KAPTUR, Elvis, Hendrix, and Other Acronyms: Domain Engineering at CTA. In Proceedings of the Sixth Workshop on Institutionalizing Software Reuse, 1993.

[31] S. Bailin. Applying Multi-Media to the Reuse of Design Knowledge. In Proceedings of the Eighth Workshop on Institutionalizing Software Reuse, 1997.

[32] S. Bailin and M. Simos. Learning and Inquiring Based Reuse Adoption (LIBRA): A Field Guide to Reuse Adoption through Organizational Learning. Technical Report STARS-PA33-AG01/001/02, Software Technology for Adaptable, Reliable Systems (STARS), February 1996.

[33] R. Balzer. Design Refinement in DSSAs. In Proceedings of the JSGCC Software Initiative Strategy Workshop, Vail, Colorado, December 1992.

[34] Robert Balzer. An Architectural Infrastructure for Product Families. In Proceedings of the Second International Workshop on Development and Evolution of Software Architectures for Product Families, pages 158–160, February 1998.

[35] S. Bandinelli. Reference Architectures in a Product Line Process Context. Technical Report ESI-1996-REUSE02, European Software Institute, 1996.

[36] Sergio Bandinelli. Light-Weight Product-Family Engineering. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 327–331.

[37] Sergio Bandinelli and Goiuri Sagardui Mendieta. Domain Potential Analysis: Calling the Attention on Business Issues of Product Lines. In Third International Workshop on Software Architectures for Product Families [359], pages 75–81.

[38] V. Basili, S. Condon, K. El Emam, R. Hendrick, and W. Melo. Characterizing and Modeling the Cost of Rework in a Library of Reusable Software Components. In Proceedings of the Nineteenth International Conference on Software Engineering (ICSE'97), pages 282–291, 1997. [39] L. Bass, G. Chastek, P. Clements, L. Northrop, D. Smith, and J. Withey. Second Product Line Practice Workshop Report. Technical Report CMU/SEI-98-TR-015, Software Engineering Institute, Carnegie Mellon University, April 1998.

[40] L. Bass, P. Clements, S. Cohen, L. Northrop, and J. Withey. Product Line Practice Workshop Report. Technical Report CMU/SEI-97-TR-003, Software Engineering Institute, Carnegie Mellon University, June 1997.

[41] Len Bass, Grady Campbell, Paul Clements, Linda Northrop, and Dennis Smith. Third Product Line Practice Workshop Report. Technical Report CMU/SEI-99-TR-003, Software Engineering Institute, Carnegie Mellon University, March 1999.

[42] Len Bass, Paul Clements, Patrick Donohoe, John McGregor, and Linda Northrop. Fourth Product Line Practice Workshop Report. Technical Report CMU/SEI-2000-TR-002, Software Engineering Institute, Carnegie Mellon University, February 2000.

[43] Len Bass, Mark Klein, and Felix Bachmann. Quality Attribute Design Primitives and the Attribute Driven Design Method. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 163–176.

[44] P. Bassett. Framing Software Reuse. Lessons From the Real World. Yourdon Press, 1997.

[45] D. Batory. Graphical Specification of Avionics Architectures in ADAGE. Technical Report ADAGE-UT-92-07, Department of Computer Science, University of Texas, Austin, Texas, October 1992.

[46] D. Batory. A Domain Model of Radio Navigation, Guidance, and Flight Director Software. Technical Report ADAGE-UT-93-01, Department of Computer Science, University of Texas, Austin, TX, January 1993.

[47] D. Batory. A Process and Retrospection on Creating a Domain Model for Avionics Software. Technical Report ADAGE-UT-93-04, Department of Computer Science, University of Texas, Austin, TX, May 1993.

[48] D. Batory. le: A Type Expressions Language. Technical Report ADAGE-UT-93-02, University of Texas, Austin, Texas, May 1993.

[49] D. Batory. A Software Generator for Colored Type Expressions. Technical Report ADAGE-UT-94-02, Department of Computer Science, University of Texas, Austin, TX, 1994.

[50] D. Batory. Extensible Realm Interfaces. Technical Report ADAGE-UT-94-01, Department of Computer Science, University of Texas, Austin, TX, 1994.

[51] D. Batory. On the Relationship of ADAGE and Design Patterns. Technical Report ADAGE-UT-95-01, Department of Computer Science, University of Texas, Austin, TX, 1995.

[52] D. Batory. Software System Generators, Architectures, and Reuse. Tutorial, April 1997.

[53] D. Batory and L. Coglianese. Techniques for Software System Synthesis in ADAGE. Technical Report ADAGE-UT- 93-05, Department of Computer Science, University of Texas, Austin, TX, 1993.

[54] D. Batory, L. Coglianese, M. Goodwin, and S. Shafer. Creating Reference Architectures: An Example from Avionics. In Proceedings of the ACM SIGSOFT Symposium on Software Reusability (SSR'95), pages 27–37, 1995.

[55] D. Batory, L. Coglianese, M. Goodwin, and R. Smith. A Domain Model for Avionics Software. Technical Report ADAGE-UT-92-01, Department of Computer Science, University of Texas, Austin, Texas, February 1992.

[56] D. Batory, L. Coglianese, S. Shafer, and W. Tracz. The ADAGE Avionics Reference Architecture. Technical Report ADAGE-UT-94-03, Department of Computer Science, University of Texas, Austin, TX, 1994.

[57] D. Batory and B. Geraci. Composition Validation and Subjectivity in GenVoca Generators. IEEE Transactions on Software Engineering, 23(2):67–82, February 1997.

[58] D. Batory, C. Johnson, B. MacDonald, and D. von Heeder. Achieving Extensibility Through Product-Lines and Domain-Specific Languages: A Case Study. In W. B. Frakes, editor, Proceedings of the Sixth International Conference on Software Reuse, pages 117–136, June 2000.

[59] D. Batory, D. McAllister, L. Coglianese, and W. Tracz. Domain Modeling in Engineering Computer-Based Systems. Technical Report ADAGE-UT-94-04, University of Texas, Austin, TX, 1994. [60] D. Batory, V. Singhal, and M. Sirkin. Implementing a Domain Model for Data Structures. International Journal of Software Engineering & Knowledge Engineering, 2(3):375–402, 1992.

[61] D. Batory, V. Singhal, J. Thomas, S. Dasari, B. Geraci, and M. Sirkin. The GenVoca Model of Software Systems Generators. IEEE Software, pages 89–94, September 1994.

[62] D. Batory, W. Tracz, S. Shafer, and L. Coglianese. Representation Issues in Creating an Avionics Domain-Specific Software Architecture Domain Model. Technical Report ADAGE-LOR-94-01, Loral Federal Systems, 1994.

[63] Don Batory, Rich Cardone, and Yannis Smaragdakis. Object-Oriented Frameworks and Product Lines. In Proceedings of the First Software Product Line Conference [201], pages 227–247.

[64] Joe Baumann. The Perfect Architecture is Non-Optimal. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 235–244.

[65] I. D. Baxter. Transformation Systems: Domain-Oriented Component and Implementation Knowledge. In Proceedings of the Ninth Workshop on Institutionalizing Software Reuse, Austin, TX, USA, January 1999.

[66] J. Bayer, O. Flege, and C. Gacek. Creating Product Line Architectures. In Third International Workshop on Software Architectures for Product Families [359], pages 210–216.

[67] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud. PuLSE: A Methodology to Develop Software Product Lines. In Proceedings of the Fifth ACM SIGSOFT Symposium on Software Reusability (SSR'99), pages 122–131, Los Angeles, CA, USA, May 1999. ACM.

[68] J. Bayer, J.-F Girard, M. Wuerthner, J.-M. DeBaud, and M. Apel. Transitioning Legacy Assets to a Product Line Architecture. In Proceedings of the Seventh European Software Engineering Conference (ESEC'99), LNCS 1687, pages 446–463, Toulouse, France, September 1999. Springer.

[69] J. Bayer, D. Muthig, and T. Widen. Customizable Domain Analysis. In Proceedings of the First International Symposium on Generative and Component-Based Software Engineering (GCSE '99), Erfurt, Germany, September 1999.

[70] Joachim Bayer. Introducing Separation of Concerns to Product Line Engineering. In Proceedings of the GCSE'00 Young Researchers Workshop, 2000.

[71] Joachim Bayer. Towards Engineering Product Lines Using Concerns. In Workshop on Multi-Dimensional Separation of Concerns in Software Engineering (ICSE 2000), June 2000.

[72] Joachim Bayer, Cristina Gacek, Dirk Muthig, and Tanya Widen. PuLSE-I: Deriving Instances from a Product Line Infrastructure. In Seventh IEEE international Conference and Workshop on the Engineering of Computer-Based System, pages 237–245, 2000.

[73] Joachim Bayer, Dirk Muthig, and Brigitte Goepfert. The Library Systems Product Line: A KobrA Case Study. Technical Report IESE-Report No. 024.01/E, Fraunhofer Institute for Experimental Software Engineering (IESE), November 2001.

[74] Joachim Bayer and Tanya Widen. Introducing Traceability to Product Lines. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 399–406.

[75] M. Becker and J. Diaz-Herrera. Creating Domain Specific Libraries: A Methodology and Design Guidelines. In Proceedings of the Third International Conference on Software Reuse, pages 158–168, 1994.

[76] Martin Becker, Lars Geyer, Andreas Gilbert, and Karsten Becker. Comprehensive Variability Modelling to Facilitate Efficient Variability Treatment. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 279–285.

[77] J. Bergey, P. Clements, S. Cohen, P. Donohoe, L. Jones, B. Krut, L. Northrop, S. Tilley, D. Smith, and J. Withey. DoD Product Line Practice Workshop Report. Technical Report CMU/SEI-98-TR-007, Software Engineering Institute, Carnegie Mellon University, May 1998.

[78] John Bergey, Sholom Cohen, Matthew Fischer, Grady Campbell, Lawrence Jones, Robert Krut, Linda Northrop, William O'Brien, Dennis Smith, and Albert Soule. Fourth DoD Product Line Practice Workshop Report. Technical Report CMU/SEI-2001-TR-017, Software Engineering Institute, Carnegie Mellon University, October

2001. [79] John Bergey, Liam O'Brien, and Dennis Smith. Mining Existing Assets for Software Product Lines. Technical Note CMU/SEI-2000-TN-008, Software Engineering Institute, Carnegie Mellon University, May 2000.

[80] John Bergey, Liam O'Brien, and Dennis Smith. Options Analysis for Reengineering (OAR): A Method for Mining Legacy Assets. Technical Note CMU/SEI-2001-TN-013, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, June 2001.

[81] John K. Bergey and Wolfhart B. Goethert. Developing a Product Line Acquisition Strategy for a DoD Organization: A Case Study. Technical Note CMU/SEI-2001-TN-021, Software Engineering Institute, Carnegie Mellon University, April 2001.

[82] A. Berns. Product Line Metrics Framework Document. Technical Report STARS-VC-K017R1/001/01, Comprehensive Approach to Reusable Defense Software (CARDS), October 1996.

[83] S. Bhansali. Reusing Software Design: A Generic Architecture-based Approach. In Proceedings of the Sixth Workshop on Institutionalizing Software Reuse, 1993.

[84] S. Bhansali. A Knowledge-Assisted Approach to Parameterized Reuse. International Journal of Software Engineering & Knowledge Engineering, 6(4):641–671, December 1996.

[85] T. Biggerstaff. The Nature of Semi-Formal Information in Domain Models. Technical Report STP-289-88, Microelectronics and Computer Technology Corporation, Austin, TX, September 1988.

[86] T. J. Biggerstaff. Generation Flexibility versus Performance. In Proceedings of the Ninth Workshop on Institutionalizing Software Reuse, Austin, TX, USA, January 1999.

[87] P. Binns, M. Englehart, M. Jackson, and S. Vestal. Domain-Specific Software Architectures for Guidance, Navigation and Control. International Journal of Software Engineering & Knowledge Engineering, 6(2):201– 227, 1995.

[88] Marie-Jos´e Blin, Francoise Fabret, Olga Kapitskaia, and Francois Llirbat. ProjectLeader: A Constraint-Based Process Support for the Distributed Design of Component-Based Products. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 191–206.

[89] G¨ unter B¨ockle. Model-Based Requirements Engineering for Product Lines. In Proceedings of the First Software Product Line Conference [201], pages 193–203.

[90] G¨ unter B¨ockle, Jesus Bermejo Mu˜ noz, Peter Knauber, Charles W. Krueger, Julio Cesar Sampaio do Prado Leite, Frank van der Linden, Linda Northrop, Michael Stark, and David M. Weiss. Adopting and Institutionalizing a Product Line Culture. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302].

[91] B. Boehm and W. Scherlis. Megaprogramming. In Proceedings of the DARPA Software Technology Conference, pages 63–82. Meridien Corp., Arlington, VA, 1992.

[92] Kai B¨ollert and Detlef Streitferdt. Requirements for Modeling Software Product Lines. In Proceedings of the GCSE 2000 Young Researchers Workshop, 2000.

[93] S. Bologna, T. Sivertsen, and H. Valisuo. Rigorous Engineering Practice and Formal Reasoning of Deep Domain Knowledge – The Basis of Dependable Knowledge Based Systems for Plant Control. International Journal of Software Engineering & Knowledge Engineering, 3(1):53–98, 1993.

[94] J. Bosch. Product-Line Architectures in Industry: A Case Study. In Proceedings of the First Nordic Software Architecture Workshop, 1998.

[95] J. Bosch. Evolution and Composition of Reusable Assets in Product-Line Architectures: A Case Study. In Proceedings of the First Working IFIP Conference on Software Architecture, 1999.

[96] J. Bosch. Product-Line Architectures in Industry: A Case Study. In Proceedings of the 21 st International Conference on Software Engineering (ICSE'99), pages 544–554, Los Angeles, CA, USA, May 1999.

[97] Jan Bosch. Design and Use of Software Architectures. Addison-Wesley, 2000.

[98] Jan Bosch. Organizing for Software Product Lines. In Third International Workshop on Software Architectures for Product Families [359], pages 117–134. [99] Jan Bosch. Adopting Software Product Lines: Approaches, Artefacts and Organization. In Proceedings of the International Workshop on Product Line Engineering - The Early Steps: Planning, Modeling, and Managing (PLEES'01) [496]. IESE-Report No. 050.01/E.

[100] Jan Bosch, editor. Generative and Component-Based Software Engineering, LNCS 2186. Springer, 2001. Third International Conference, GCSE 2001 Erfurt, Germany, September 2001.

[101] Jan Bosch. Software Product Lines: Organizational Alternatives. In Proceedings of the 23rd International Conference on Software Engineering, pages 91–100. IEEE Computer Society Press, November 2001.

[102] Jan Bosch and PerOlof Bengtsson. Component Evolution in Product-Line Architectures. In Proceedings of 1999 International Workshop on Component-Based Software Engineering, 1999.

[103] Jan Bosch, Gert Florijn, Danny Greefhorst, Juha Kuusela, Henk Obbink, and Klaus Pohl. Variability Issues in Software Product Lines. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 11–19.

[104] Jan Bosch and Mattias H¨ogstr¨ om. Product Instantiation in Software Product Lines: A Case Study. In Second International Symposium on Generative and Component-Based Software Engineering, October 2000.

[105] Jan Bosch and Alexander Ran. Evolution of Software Product Families. In Third International Workshop on Software Architectures for Product Families [359], pages 168–183.

[106] C. Braun, R. Coutant, and J. Armitage. Domain Specific Software Architectures: A Process for Architecture-Based Software Engineering.

[107] C. Braun, W. Hatch, T. Ruegsegger, B. Balzer, M. Feather, N. Goldman, and D. Wile. Domain-Specific Software Architectures – Command and Control. In Proceedings of 1992 IEEE Symposium on Computer-Aided Control System Design, Napa, CA, March 1992. This report was also published in [MettalaGraham92a].

[108] D. Bristow, B. Bulat, and R. Burton. Product-Line Process Development. Technical report, Software Technology for Adaptable, Reliable Systems (STARS), February 1995.

[109] L. Brownsword and P. Clements. A Case Study in Successful Product Line Development. Technical Report CMU/SEI-96-TR-016, Software Engineering Institute, Carnegie Mellon University, 1996.

[110] L. Brownsword, P. Clements, and U. Olsson. Successful Product Line Engineering: A Case Study. Technical Report, Software Engineering Institute, Carnegie Mellon University, 1996.

[111] G. Bruns and C. Potts. Domain Modeling Approaches to Software Development. Technical Report STP-186-88, Microelectronics and Computer Technology Corporation, Austin, TX, June 1988.

[112] B. Bulat. SWSC Domain Engineering Experience. Technical Report STARS A014-005, Software Technology for Adaptable, Reliable Systems (STARS), 1995.

[113] R. Burdick. Domain Analysis and Information Engineering: Promoting a Combined Attack on Stovepipe Systems. In Proceedings of the Sixth Workshop on Institutionalizing Software Reuse, 1993.

[114] R. Burdick. Organizational Domain Modeling Support: A KAPTUR and RLF Integration Strategy. 1993.

[115] R. Capilla. Application of Domain Analysis to Knowledge Reuse. In Proceedings of the Eighth Workshop on Institutionalizing Software Reuse, 1997.

[116] Rafael Capilla and Juan C. Due˜ nas. Modelling Variability with Features in Distributed Architectures. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 299– 311.

[117] R. Cardone. On the Relationship of Aspect-Oriented Programming and GenVoca. In Proceedings of the Ninth Workshop on Institutionalizing Software Reuse, Austin, TX, USA, January 1999.

[118] Rodrigo Ceron, Juan C. Duenas, and Juan A. de la Puente. A First Assessment of Development Process with Respect to Product Lines and Component Based Development. In Third International Workshop on Software Architectures for Product Families [359], pages 158–167.

[119] G. Chastek, P. Donohoe, K. C. Kang, and S. Thiel. Product Line Analysis: A Practical Introduction. Technical Report CMU/SEI-2001-TR-001, Software Engineering Institute, Carnegie Mellon University, June 2001. [120] Y. C. Cheong and S. Jarzabek. Frame-based Method for Customizing Generic Software Architectures. In Proceedings of the Fifth ACM SIGSOFT Symposium on Software Reusability (SSR'99), pages 103–112, Los Angeles, CA, USA, May 1999. ACM.

[121] S. Cherki, W. El Kaim, P. Josset, and F. Paris. Domain Analysis and Product-Line Scoping: A Thomson-CSF Product Line Case. In Proceedings of the International Workshop on Software Product Lines: Economics, Architectures, and Implications [334]. IESE-Report 070.00/E.

[122] S.-B. Cho and T. Ray. An Evolutionary Approach to Program Transformation and Synthesis. International Journal of Software Engineering & Knowledge Engineering, 5(2):179–192, 1995.

[123] P. Clements. From Domain Models to Architectures. In Workshop on Software Architecture, Los Angeles, CA, 1994.

[124] P. Clements. Report of the Reuse and Product Lines Working Group of WISR8. Technical Report CMU/SEI-97- SR-010, Software Engineering Institute, Carnegie Mellon University, August 1997.

[125] P. Clements. Successful Product Line Engineering Requires more than Reuse. In Proceedings of the Eighth Workshop on Institutionalizing Software Reuse, 1997.

[126] P. Clements and N. Weiderman. Notes on the Second International Workshop on Development and Evolution of Software Architectures for Product Families. ACM SIGSOFT Software Engineering Notes, 23(3):39–43, May 1998.

[127] P. C. Clements. Essential Product Line Practices. In Proceedings of the Ninth Workshop on Institutionalizing Software Reuse, Austin, TX, USA, January 1999.

[128] Paul Clements, Sholom Cohen, Patrick Donohoe, and Linda Northrop. Control Channel Toolkit: A Software Product Line Case Study. Technical Report CMU/SEI-2001-TR-030, Software Engineering Institute, Carnegie Mellon University, September 2001.

[129] Paul Clements, Patrick Donohoe, Kyo Kang, John McGregor, and Linda Northrop. Fifth Product Line Practice Workshop Report. Technical Report CMU/SEI-TR-027, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, September 2001.

[130] Paul C. Clements. On the Importance of Product Line Scope. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 69–77.

[131] Paul C. Clements and Linda Northrop. Software Product Lines: Practices and Patterns. SEI Series in Software Engineering. Addison-Wesley, August 2001.

[132] W. Codenie, K. DeHondt, P. Steyaert, and A. Vercammen. From Custom Applications to Domain-Specific Frameworks. Communications of the ACM, 40(10):71–77, October 1997.

[133] L. Coglianese. Architecture Component Relationships for the DSSA-ADAGE Project. Technical Report ADAGE-IBM- 93-11, IBM Federal Systems Company, 1993.

[134] L. Coglianese and R. Smith. Core Avionics Domain Analysis. Technical Report ADAGE-IBM-92-06, IBM Federal Systems Company, 1992.

[135] S. Cohen. Process and Products for Software Reuse and Domain Analysis. In Proceedings of the Fourth Workshop on Institutionalizing Software Reuse, 1991.

[136] S. Cohen. A Model Base for Software Engineering. In Proceedings of the Sixth Workshop on Institutionalizing Software Reuse, 1993.

[137] S. Cohen. Object Technology, Architectures, and Domain Analysis – What's the Connection? In Proceedings of the Eighth Workshop on Institutionalizing Software Reuse, 1997.

[138] S. Cohen. From Use Cases to Domains and Architecture. In Proceedings of the Ninth Workshop on InstitutionalizingSoftware Reuse, Austin, TX, USA, January 1999.

[139] S. Cohen. Guidelines for Developing a Product Line Concept of Operations. Technical Report CMU/SEI-99-TR-008, Software Engineering Institute, Carnegie Mellon University, August 1999.

[140] S. Cohen, S. Friedman, L. Martin, T. Royer, N. Solderitsch, and R. Webster. Concept of Operations for the ESC Product Line Approach. Technical Report CMU/SEI-96-TR-018, Software Engineering Institute, Carnegie Mellon University, September 1996.

[141] S. Cohen, S. Friedman, L. Martin, N. Solderitsch, and R. Webster. Product Line Identification for ESC-Hanscom. Technical Report CMU/SEI-95-SR-024, Software Engineering Institute, Carnegie Mellon University, 1996.

[142] S. Cohen and L. M. Northrop. Object-Oriented Technology and Domain Analysis. In Proceedings of the Fifth International Conference on Software Reuse, pages 86–93, Vancouver, BC, Canada, June 1998.

[143] S. Cohen, J. Stanley, S. Peterson, and R. Krut. Application of Feature-Oriented Domain Analysis to the Army Movement Control Domain. Technical Report CMU/SEI-91-TR-28, Software Engineering Institute, Carnegie Mellon University, June 1992.

[144] Sholom Cohen. From Product-Line Architectures to Products. In Proceedings of Workshop on Object Technology for Product-Line Architectures, Lisbon, Portugal, 1999.

[145] Sholom Cohen. Case Study: Building and Communicating a Business Case for a DoD Product Line. Technical Note CMU/SEI-2001-TN-020, Software Engineering Institute, Carnegie Mellon University, April 2001.

[146] Sholom Cohen. Predicting when Product Line Investment Pays. In Proceedings of the Second International Workshop on Software Product Lines: Economics, Architectures, and Implications [336], pages 15–18. IESE-Report.No. 051.01/E.

[147] Sholom Cohen, Brian Gallagher, Matthew Fischer, Lawrence Jones, Robert Krut, Linda Northrop, William O'Brien, Dennis Smith, and Albert Soule. Third DoD Product Line Practice Workshop Report. Technical Report CMU/SEI-2000-TR-024, Software Engineering Institute, Carnegie Mellon University, 2000.

[148] P. Collins. Toward a Reusable Domain Analysis. In Proceedings of the Fourth Workshop on Institutionalizing Software Reuse, 1991.

[149] Denis Conan, Michael Coriat, and Nicolas Farcet. A Software Component Development Meta-Model for Product Lines. In Fifth International Workshop on Component-Oriented Programming (WCOP 2000), June 2000.

[150] S. Condon, C. Seaman, V. Basili, S. Kraft, J. Kontio, and Y. Kim. Evolving the Reuse Process at the Flight Dynamics Division (FDD) Goddard Space Flight Center. In Proceedings of the 21 st Annual Software Engineering Workshop, December 1996.

[151] J. Coplien. Multi-Paradigm Design in C++. Addison-Wesley, 1998.

[152] J. Coplien. Multi-Paradigm Design. PhD thesis, Vrije Universiteit Brussel, July 2000.

[153] J. Coplien, D. Hoffmann, and D. Weiss. Commonality and Variability in Software Engineering. IEEE Software, pages 37–45, November/December 1998.

[154] Michel Coriat, Jean Jourdan, and Fabien Boisbourdin. The SPLIT Method: Building Product Lines for Software Intensive Systems. In Proceedings of the First Software Product Line Conference [201], pages 147–166.

[155] Michel Coriat and Fr´ed´eric Waeber. Product Line Process Framework: The Wheels Process. In Proceedings of the International Workshop on Software Product Lines: Economics, Architectures, and Implications [334]. IESE-Report 070.00/E.

[156] P. Cornwell. Scoping the Task and Application Domain for Knowledge Acquisition. In Proceedings of the Ninth Workshop on Institutionalizing Software Reuse, Austin, TX, USA, January 1999.

[157] P. Collins Cornwell. HP Domain Analysis: Producing Useful Models for Reusable Software. Hewlett-Packard Journal, 47(4), August 1996.

[158] Software Productivity Consortium Services Corporation. Consortium's Synthesis Reuse Approach in Navy/STARS, Rockwell Programs.

[159] Software Productivity Consortium Services Corporation. Reuse-Driven Software Processes Guidebook.

[160] R. Creps. Domain Specific Environment Repository Composite Paradigm Report. Technical Report STARS-SC- 03068/001/00, Software Technology for Adaptable, Reliable Systems (STARS), May 1991.

[161] R. Creps, R. Prieto-Diaz, M. Davis, M. Simos, P. Collins, and G. Wickman. Using a Conceptual Framework for Reuse Processes as a Basis for Reuse Adoption and Planning. Technical report, Software Technology for Adaptable, Reliable Systems (STARS), December 1993. [162] R. Crispen, B. Freemon, K. King, and W. Tucker. DARTS: A Domain Architecture for Reuse in Training Systems. In Proceedings of the I/ITSEC, November 1993.

[163] D. Cuka and D. Weiss. Specifying Executable Commands: An Example of FAST Domain Engineering. Submitted to IEEE Transactions on Software Engineering, 1997.

[164] M. Cusumano. Japan's Software Factories. Oxford University Press, Oxford, UK, February 1991.

[165] K. Czarnecki. Leveraging Reuse Through Domain-Specific Architectures. In Proceedings of the Eighth Workshop on Institutionalizing Software Reuse, 1997.

[166] James C. Dager. Cummins' Experience in Developing a Software Product Line Architecture for Realtime Embedded Diesel Engine Controls. In Proceedings of the First Software Product Line Conference [201], pages 23–45.

[167] Eric M. Dashofy and Andr´e van der Hoek. Representing Product Family Architectures in an Extensible Architecture Description Language. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 313–326.

[168] M. Davis. STARS Framework for Reuse Processes. In Proceedings of the Fourth Workshop on Institutionalizing Software Reuse, 1991.

[169] M. Davis. STARS Reuse Maturity Model: Guidelines for Reuse Strategy Formulation. Technical report, Software Technology for Adaptable, Reliable Systems (STARS), October 1992.

[170] M. Davis. Reuse Strategy Model: Planning Aid for Reuse-based Projects. Technical Report CDRL 5159, Software Technology for Adaptable, Reliable Systems (STARS), July 1993.

[171] M. Davis. Representing domain models graphically. In Proceedings of the Seventh Workshop on Institutionalizing Software Reuse, 1995.

[172] M. Davis and H. Hawley. Dialogue-Specified Reuse of Domain Engineering Workproducts. In Proceedings of 1994 Washington Ada Symposium (WAdaS '94), 1994.

[173] M. Davis and H. Hawley. Reuse of Software Process and Product Through Knowledge-based Adaption. In Proceedings of the Third Conference on Software Reusability, November 1994.

[174] Margaret J. Davis. Reengineering and the Product Line Approach to Software Development. In 4th Reengineering Forum, September 1994.

[175] A. de Cima, C. Werner, and A. Cerqueira. The Design of Object-Oriented Software with Domain Architecture Reuse. In Proceedings of the Third International Conference on Software Reuse, pages 178–187, 1994.

[176] Fons de Lange and Tom Jansen. The Philips-OpenTV Product Family Architecture for Interactive Set-Top Boxes. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 177–190.

[177] J.-M. DeBaud. The Construction of Software Systems using Domain-Specific Reuse Infrastructures. PhD thesis, Georgia Institute of Technology, December 1996.

[178] J.-M. DeBaud. Lessons Learned From a Domain-Based Re-Engineering Effort. In Proceedings of the Working Conference on Reverse Engineering (WCRE '96), pages 217–226, November 1996.

[179] J.-M. DeBaud. Viewing a DSSA in Context: Problems versus Solutions. In Proceedings of the 2nd International Software Architecture Workshop ISAW-2, pages 19–23, October 1996.

[180] J.-M. DeBaud. Towards a Customizable Domain Analysis Framework: Initial Lessons from the Field. In Proceedings of the European Reuse Workshop 1997 (ERW'97), pages 112–115, November 1997.

[181] J.-M. DeBaud, O. Flege, and P. Knauber. PuLSE-DSSA - A Method for the Development of Software Reference Architectures. In Proceedings of the Third International Software Architecture Workshop, November 1998.

[182] J.-M. DeBaud and R. LeBlanc. Problem-Oriented Domain Analysis.

[183] J.-M. DeBaud and R. LeBlanc. Using Domain-Specific Software Architectures to Articulate Component Repositories. In Proceedings of the First International Workshop on Architectures for Software Systems, pages 67–71,

April 1995. [184] J.-M. DeBaud, B. Moopen, and S. Rugaber. Domain Analysis and Reverse Engineering. In Proceedings of the International Conference on Software Maintenance (ICSM'94), pages 326–335, 1994.

[185] J.-M. DeBaud and S. Rugaber. A Software Re-Engineering Method using Domain Models. In Proceedings of the International Conference on Software Maintenance (ICSM'95), pages 204–213, 1995.

[186] J.-M. DeBaud and K. Schmid. Identifying and Evolving the Scope of Software Product Lines. In Proceedings of the European Reuse Workshop 1998 (ERW'98), volume II, pages 69–72, 1998.

[187] J.-M. DeBaud and K. Schmid. PuLSE-Eco: a Context-Based Approach to Scope a Software Product Line. In Second Workshop on Software Architectures in Product Line Acquisition, June 8-10, Salem, MA, 1998.

[188] J.-M. DeBaud and K. Schmid. A Systematic Approach to Derive the Scope of Software Product Lines. In Proceedings of the 21 st International Conference on Software Engineering (ICSE'99), pages 34–43, Los Angeles, CA, USA, May 1999.

[189] J.-M. DeBaud and Klaus Schmid. A Practical Comparison of Major Domain Analysis Approaches - Towards a Customizable Domain Analysis Framework. In Proceedings of the Tenth Conference on Software Engineering and Knowledge Engineering (SEKE'98), pages 128–131, June 1998.

[190] P. Devanbu. Research Issues with Application Generators. In Proceedings of the Sixth Workshop on Institutionalizing Software Reuse, 1993.

[191] J. Diaz-Herrera, S. Coehn, and J. Withey. Institutionalizing Systematic Reuse: A Model-Based Approach. In Proceedings of the Seventh Workshop on Institutionalizing Software Reuse, 1995.

[192] Jorge L. Diaz-Herrera and Juan Carlos Guzm´an. Product Lines in the Context of Embedded Systems. In Proceedings of the Second International Workshop on Software Product Lines: Economics, Architectures, and Implications [336], pages 19–25. IESE-Report. No. 051.01/E.

[193] Jorge L. Diaz-Herrera and Vijay K. Madisetti. Embedded Systems Product Lines: A Technical Analysis. In Proceedings of the International Workshop on Software Product Lines: Economics, Architectures, and Implications [334]. IESE-Report 070.00/E.

[194] D. Dikel, D. Kane, S. Ornburn, W. Loftus, and J. Wilson. Applying Software Product-Line Architecture. IEEE Computer, pages 49–55, August 1997.

[195] Ebru Dincel, Nenad Medvidovic, and Andr´e van der Hoek. Measuring Product Line Architectures. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 333–338.

[196] J. do Prado Leite, M. Sant'Anna, and F. de Freitas. Draco-PUC: A Technology Assembly for Domain Oriented Software Development. In Proceedings of the Third International Conference on Software Reuse, pages 94–100, 1994.

[197] Liliana Dobricia and Eila Niemel¨a. A Strategy for Analyzing Product Line Software Architectures. Technical Report VTT-PUBS-427, VTT Electronics, December 2000.

[198] DoD Software Reuse Initiative, DISA/CIM, Arlington, VA. Domain Analysis Guidelines, draft edition, May 1992.

[199] Bryan S. Doerr and David Sharp. Freeing Product Line Architectures from Execution Dependencies. In Proceedings of the First Software Product Line Conference [201], pages 313–329.

[200] Tom Dolan, Ruud Weterings, and J. C. Wortmann. Stakeholder-Centric Assessment of Product Family Architecture – Practical Guidelines for Information System Interoperability and Extensibility. In Third International Workshop on Software Architectures for Product Families [359], pages 225–243.

[201] Patrick Donohoe, editor. Software Product Lines: Experience and Research Directions. Proceedings of the First Software Product Line Conference. Kluwer Academic Publishers, August 2000.

[202] S. Dowle. Domain Modeling for Requirements Specification. In Proceedings of the Third International Conference on Command, Control, Communications and Management Information Systems, pages 1–7, 1989.

[203] M. Dunn and J. Knight. Creating and Using An Industrial Domain Model. In Proceedings of the Sixth Workshop on Institutionalizing Software Reuse, 1993.

[204] S. Edwards and B. Weide. WISR8 8th Annual Workshop on Software Reuse Summary and Working Group Reports. ACM SIGSOFT Software Engineering Notes, 22(5):17–32, September 1997. [205] Alexander Egyed, Nikunj Mehta, and Nenad Medvidovic. Software Connectors and Refinement in Family Architectures. In Third International Workshop on Software Architectures for Product Families [359], pages 95–105.

[206] D. Eichmann. Representing Knowledge in Domain Engineering. In Proceedings of the Eighth Workshop on Institutionalizing Software Reuse, 1997.

[207] Ulrich W. Eisenecker and Krzysztof Czarnecki. Generative Programming: Methods, Tools, and Applications. Addison-Wesley, 2000.

[208] W. Eixelsberger. Recovery of a Reference Architecture: A Case Study. In Proceedings of the Third International Software Architecture Workshop, November 1998.

[209] M. Englehart and M. Jackson. ControlH: A Specification Language and Code Generator for Real-Time GN&C Applications. Technical report, Honeywell Technology Center, 1995.

[210] L. Erman and F. Hayes-Roth. DOCSET: DSSA Open Common System-Engineering Toolkit – A Vision. Technical report, Teknowledge Federal Systems, Palo Alto, CA, April 1993.

[211] L. Erman and F. Hayes-Roth. Teknowledge's ProtoTech Project: DSSA Open Common System-Engineering Toolkit. Technical report, Teknowledge Federal Systems, Palo Alto, CA, April 1994.

[212] J. Estep and S. Hissam. Technical Concept Document Central Archive for Reusable Defense Software (CARDS). Technical Report STARS-VC-B009/001/00, Software Technology for Adaptable, Reliable Systems (STARS), February 1992.

[213] Stuart R. Faulk, Robert R. Harmon, and David M. Raffo. Value-Based Software Engineering (VBSE): A Value-Driven Approach to Product-Line Engineering. In Proceedings of the First Software Product Line Conference [201], pages 205–223.

[214] D. Fensel. Reuse of Problem-Solving in Knowledge Engineering. In Proceedings of the Sixth Workshop on Institutionalizing Software Reuse, 1993.

[215] Stefan Ferber, Peter Heidl, and Peter Lutz. Reviewing Product Line Architectures: Experience Report of ATAM in an Automotive Context. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 351–368.

[216] J. Fernandez. A Taxonomy of Coordination Mechanisms Used in Real-Time Software Based on Domain Analysis. Technical Report CMU/SEI-93-TR-34, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, December 1993.

[217] S. Fickas and P. Nagarajan. Domain Analysis and Software Systems Modeling, chapter Critiquing Software Specifications, pages 224–234. IEEE Computer Society Press, 1991. Book out of print.

[218] G. Fischer. Seeding, Evolutionary Growth and Reseeding: Constructing, Capturing and Evolving Knowledge in Domain-Oriented Design Environments. Automated Software Engineering, 5(4):447–464, October 1998.

[219] G. Fischer, A. Girgensohn, K. Nakakoji, and D. Redmiles. Supporting Software Designers with Integrated Domain-Oriented Design Environments. IEEE Transactions on Software Engineering, 18(6):511–522, June 1992.

[220] Oliver Flege. System Family Architecture Description Using the UML. Technical Report IESE-Report No. 092.00/E, Fraunhofer Institute for Experimental Software Engineering (IESE), December 2000.

[221] B. Frakes. Automating Domain Engineering. In Proceedings of the Eighth Workshop on Institutionalizing Software Reuse, 1997.

[222] B. Frakes. Domain Engineering Education. In Proceedings of the Ninth Workshop on Institutionalizing Software Reuse, Austin, TX, USA, January 1999.

[223] W. Frakes. A Graduate Course on Software Reuse, Domain Analysis, and Re-Engineering. In Proceedings of the Sixth Workshop on Institutionalizing Software Reuse, 1993.

[224] R. France and T. Horton. Applying Domain Analysis and Modeling: An Industrial Experience. In Mansur Samadzadeh, editor, Proceedings of the ACM SIGSOFT Symposium on Software Reusability (SSR'95), pages 206–214, April 1995. [225] S. Fraser. Reuse by Design - A Team Approach. In Proceedings of the Fifth Workshop on Institutionalizing Software Reuse, 1992.

[226] S. Fraser, D. Leishman, and R. McLellan. Patterns, Teams and Domain Engineering. In Proceedings of the ACM SIGSOFT Symposium on Software Reusability (SSR'95), pages 222–224, 1995.

[227] S. Fraser, S. Peterson, D. Schmidt, M. Simos, W. Tracz, and N. Zalman. Panel: Domain Analysis: From Tar Pit Extraction to Object Mania? In Proceedings of the Fourth International Conference on Software Reuse, pages 227–235, April 1996.

[228] P. Freeman. A Conceptual Analysis of the Draco Approach to Constructing Software Systems. IEEE Transactions on Software Engineering, 1987.

[229] B. Freemon and R. Crispen. Testing a Technology for Reuse. In Proceedings of the Fourteenth I/ITSEC,November 1992.

[230] C. Fronzcek, G. Jackelen, and S. Riesbeck. Engineer's Handbook Central Archive for Reusable Defense Software (CARDS). Technical report, Software Technology for Adaptable, Reliable Systems (STARS), 1994.

[231] C. Fuhrman, N. Solderitsch, S. Yacoub, , and H. Ammar. An Integrated Tool Environment for DoD Product Line Engineering. In Proceedings of the First Symposium on Reusable Architectures and Components for Developing Distributed Information Systems (RACDIS'99), 1999.

[232] C. Gacek. Exploiting Domain Architectures in Software Reuse. In Proceedings of the ACM SIGSOFT Symposium on Software Reusability (SSR'95), pages 229–232, 1995.

[233] Christina Gacek, editor. Software Reuse: Methods, Techniques, and Tools. 7th International Conference, ICSR-7. Austin, TX, USA, April 2002, Proceedings, LNCS 2319. Springer, April 2002.

[234] Christina Gacek, Peter Knauber, Klaus Schmid, and Paul Clements. Successful Software Product Line Development in a Small Organization: A Case Study. Technical Report IESE-Report No. 013.01/E, Fraunhofer Institute for Experimental Software Engineering (IESE), March 2001.

[235] Cristina Gacek, Jean Jourdan, and Michel Coriat, editors. Proceedings of Product Line Architecture Workshop. The First Software Product Line Conference (SPLC1), Denver, CO, USA. Fraunhofer Institute for Experimental Software Engineering (IESE), August 2000. IESE-Report 053.00/E.

[236] H. Gall, R. Kloesch, and R. Mittermeir. Using Domain Knowledge to Improve Reverse Engineering. In International Journal of Software Engineering & Knowledge Engineering, volume 6, pages 477–505, 1995.

[237] G. Gambhir. Use of Domain Analysis to Implement the Developer Off-The-Shelf Systems (DOTSS) System Acquisition Approach. ACM SIGSOFT Software Engineering Notes, 22(2):48–53, March 1997.

[238] M.-A. Gandrieau, H. Schindler, I. Nordgard, and W. Eixelsberger. ESSI DARE: Improving Reuse of Software Systems with Domain Analysis and Object-Oriented Modeling. In Proceedings of the Software Process Improvement Conference, pages 174–191, December 1995.

[239] Gerald C. Gannod and Robyn R. Lutz. An Approach to Architectural Analysis of Product Lines. In Proceedings of the 22 nd International Conference on Software Engineering (ICSE'00), pages 548–557, 2000.

[240] A. Gargaro and S. Peterson. Transitioning a modelbased software engineering architectural style to ada95. Technical Report CMU/SEI-96-TR-017, Software Engineering Institute, Carnegie Mellon University, August 1996.

[241] Birgit Geppert and Frank Roessler. Combining Product Line Engineering with Options Thinking. In Proceedings of the International Workshop on Product Line Engineering - The Early Steps: Planning, Modeling, and Managing (PLEES'01) [496]. IESE-Report No. 050.01/E.

[242] K. Gilroy, E. Comer, K. Grau, and P. Merlet. Impact of Domain Analysis on Reuse Methods. Technical Report C04-087LD-0001-00, U.S. Army Communications-Electronics Command, Ft. Monmouth, NJ, November 1989.

[243] J. Goguen. Reusing and Interconnecting Software Components. IEEE Computer, pages 16–27, February 1986.

[244] J. Goguen. Requirements for Supporting the Formal Verification of Parameterized Programs. Technical Report ADAGE-OXF-92-01, Programming Research Group, Oxford University Computing Lab, December 1992. [245] J. Goguen. Comments on Two ADAGE Process Models and a DSSA Architecture Description Language Proposal. Technical Report ADAGE-OXF-93-02, Programming Research Group, Oxford University Computing Lab, October 1993.

[246] J. Goguen. Parameterization and Formal Verification in ADAGE. Technical Report ADAGE-OXF-93-01, Programming Research Group, Oxford University Computing Lab, April 1993.

[247] J. Goguen and W. Tracz. An Implementation-Oriented Semantics for Module Composition. Technical Report ADAGE-OXF-95-01, Programming Research Group, Oxford University Computing Lab, 1995.

[248] H. Gomaa. A Domain Requirements Analysis and Specification Method. Technical report, George Mason University, Fairfax, VA, February 1990.

[249] H. Gomaa. An Object-Oriented Domain Analysis and Modeling Method for Software Reuse. In Proceedings of the Hawaii International Conference on System Sciences, pages 46–56, January 1992.

[250] H. Gomaa. Methods and Tools for Domain Specific Software Architectures. In Proceedings of the Sixth Workshop on Institutionalizing Software Reuse, 1993.

[251] H. Gomaa. Design Methods for Domain-Specific Software Architectures. In Proceedings of the First Interna-tional Workshop on Architectures for Software Systems, pages 101–108, April 1995.

[252] H. Gomaa. Domain Modeling Methods and Environments. In Proceedings of the ACM SIGSOFT Symposium on Software Reusability (SSR'95), pages 256–258, 1995.

[253] H. Gomaa. Reusable Software Requirements and Architectures for Families of Systems. Journal of Systems and Software, 10:189–202, 1995.

[254] H. Gomaa. Object Oriented Analysis and Modeling for Families of Systems with UML. In W. B. Frakes, editor, Proceedings of the Sixth International Conference on Software Reuse, pages 89–99, June 2000.

[255] H. Gomaa and G. A. Farrukh. Composition of Software Architectures from Reusable Architecture Patterns. In Proceedings of the Third International Software Architecture Workshop, November 1998.

[256] H. Gomaa, L. Kerschberg, V. Sugumaran, C. Bosch, and I. Tavakoli. A Prototype Domain Modeling Environment for Reusable Software Architectures. In Proceedings of the Third International Conference on Software Reuse, pages 74–83, 1994.

[257] H. Gomaa, L. Kerschberg, V. Sugumaran, C. Bosch, I. Tavakoli, and L. O'Hara. A Knowledge-Based Software Engineering Environment for Reusable Software Requirements and Architectures. Automated Software Engineering, 3(3/4):285–307, August 1996.

[258] Hassan Gomaa. Modeling Software Product Lines with UML. In Proceedings of the Second International Workshop on Software Product Lines: Economics, Architectures, and Implications [336], pages 27–31. IESE-Report.No. 051.01/E.

[259] Hassan Gomaa and Mark Gianturco. Domain Modeling for World Wide Web Based Software Product Lines with UML. In Proceedings of the Seventh International Conference on Software Reuse [233], pages 78–92.

[260] S. Gossain, D. Batory, H. Gomaa, M. Lubars, C. Pidgeon, and E. Seidewitz. PANEL: Objects and Domain Engineering. In ACM Sigplan Notices, volume 30, pages 333–336. ACM Press, October 1995.

[261] M. Griss. Bus-Based Kits for Reusable Software. In Proceedings of the Fourth Workshop on Institutionalizing Software Reuse, 1991.

[262] M. Griss. Towards Tools and Languages for Hybrid Domain-Specific Kits. In Proceedings of the Sixth Workshop on Institutionalizing Software Reuse, 1993.

[263] M. Griss, J. Favaro, and M. d'Alessandro. Integrating Feature Modeling with the RSEB. In Proceedings of the Fifth International Conference on Software Reuse, pages 76–85, Vancouver, BC, Canada, June 1998.

[264] M. Griss and K. Wentzel. Hybrid Domain-Specific Kits. Journal of Systems and Software, 30:213–230, 1995.

[265] M. L. Griss. Implementing Product-Line Features with Component Reuse. In W. B. Frakes, editor, Proceedings of the Sixth International Conference on Software Reuse, pages 137–152, June 2000.

[266] Martin L. Griss. Implementing Product-Line Features by Composing Component Aspects. In Proceedings of the First Software Product Line Conference [201], pages 271–288. [267] Martin L. Griss. Product-Line Architectures. In George T. Heineman and William T. Councill, editors, Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley, 2001.

[268] D. Gross and L. Stuckey. Overview of the Air Vehicle Training Systems Demonstration Project. Simulation, December 1994.

[269] D. Gross and L. Stuckey. The Heritage of the Air Vehicle Training Systems Domain. In Proceedings of the Sixteenth Interservice/Industry Training Systems Conference, November 1994.

[270] E. Guerreri. Enhancing the Use of Domain Analysis. In Proceedings of the Seventh Workshop on Institutionalizing Software Reuse, 1995.

[271] E. Guerrieri. Case Study: Digitals' Application Generators. IEEE Software, September 1994.

[272] N. Gupta, L. Jagadeesan, E. Koutsofios, and D. Weiss. Auditdraw: Generating Audits the FAST Way. In Proceedings of the 3rd IEEE International Symposium on Requirements Engineering, January 1997.

[273] Svein Hallsteinsen and Eric Swane. Handling the Diversity of Networked Devices By Means of a Product Family Approach. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 245–263.

[274] G¨ unther Halmans and Klaus Pohl. Considering Product Family Assets when Defining Customer Requirements. In Proceedings of the International Workshop on Product Line Engineering - The Early Steps: Planning, Modeling, and Managing (PLEES'01) [496]. IESE-Report No. 050.01/E.

[275] J. Hamilton. SEE Integration to Support Megaprogramming. In Sixth Conference on Software Engineering Environments, July 1993.

[276] Maarit. Harsu. A Survey of Product-Line Architectures. Technical report, Software Systems Labratory, Tampere University of Technology, October 2001.

[277] B. Hayes-Roth, K. P eger, P. Lalanda, P. Morignot, and M. Balabanovic. A Domain-Specific Software Architecture for Adaptive Intelligent Systems. IEEE Transactions on Software Engineering, 21(4):288–301, April 1995.

[278] F. Hayes-Roth. Architecture-Based Acquisition and Development of Software: Guidelines and Recommendations from the ARPA Domain-Specific Software Architecture (DSSA) Program. Technical report, Teknowledge Federal Systems, Palo Alto, CA, October 1994.

[279] F. Hayes-Roth and W. Tracz. DSSA Tool Requirements for Key Process Functions. Technical report, Teknowledge Federal Systems, Palo Alto, CA, February 1994. Version 1.1.

[280] F. Hayes-Roth and W. Tracz. DSSA Tool Requirements for Key Process Functions. Technical report, Teknowledge Federal Systems, Palo Alto, CA, October 1994. Version 2.0.

[281] Andreas Hein, John MacGregor, and Steffen Thiel. Configuring Software Product Line Features. In Workshop Feature Interaction in Composed Systems, Budapest, Hungary, June 2001.

[282] Andreas Hein, Michael Schlick, and Renato Vinga-Martins. Applying feature models in industrial settings. In Proceedings of the First Software Product Line Conference [201], pages 47–70.

[283] G. Heller and R. Hendrick. Experiences in Building a Strategic Reuse Asset Library for Practical Business Solutions. In 6th Annual Technology and Business Solutions Conference, July 1996.

[284] S. Henninger. Developing Domain Knowledge Through the Reuse of Project Experiences. In Mansur Samadzadeh, editor, Proceedings of the ACM SIGSOFT Symposium on Software Reusability (SSR'95), pages 186–195, April 1995.

[285] S. Henninger. Accelerating the Successful Reuse of Problem Solving Knowledge Through the Domain Lifecycle. In Proceedings of the Fourth International Conference on Software Reuse, pages 124–133, April 1996.

[286] S. Henninger, K. Lappala, and A. Raghavendran. An Organizational Learning Approach to Domain Analysis. In Proceedings of the Seventeenth International Conference on Software Engineering (ICSE'95), pages 95–103. ACM Press, 1995.

[287] Scott Henninger. Supporting the Domain Lifecycle. In Proceedings of the International Workshop on Computer-Aided Software Engieering, Toronta, Canada, July 1995. [288] J. Hess, W. Novak, P. Carroll, S. Cohen, R. Holibaugh, K. Kang, and S. Peterson. Domain Analysis and Software Systems Modeling, chapter A Domain Analysis Bibliography, pages 258–287. IEEE Computer Society Press, 1991. Book out of print.

[289] J. Higgins, W. Tracz, and E. Newton. DOMAIN (DOmain Model All INtegrated) User Guide. Technical Report ADAGE-LOR-94-06A, Loral Federal Systems, September 1994.

[290] R. Holibaugh. Domain Analysis Products. STARS Newsletter, 3(2), September 1992.

[291] R. Holibaugh. Joint Integrated Avionics Working Group (JIAWG) Object-Oriented Domain Analysis Method. Technical Report CMU/SEI-92-SR-3, Software Engineering Institute, Carnegie Mellon University, 1993.

[292] Clifford R. Hollander and John Ohlinger. CCT: A Component-Based Product Line Architecture for Satellite-Based Command and Control Systems. In Proceedings of Workshop on Object Technology for Product-Line Architectures, Lisbon, Portugal, 1999.

[293] C. Hollenbach and W. Frakes. Software Process Reuse in an Industrial Setting. In Proceedings of the Fourth International Conference on Software Reuse, pages 22–30, April 1996.

[294] R. Holmes, S. Rotter, and S. Parker. C3 Domain Analysis: Lessons Learned. Technical Report NRaD TD 2635, Naval Command, Control and Ocean Surveillance Center, RDT&E Division, San Diego, CA, September 1993.

[295] Honeywell, Inc. DoME Guide, 1999. Version 5.2 of the DoME Guide.

[296] J. Hook and L. Walton. The Design of Message Specification Language, June 1997.

[297] H. James Hoover, Tony Olekshy, Garry Froehlich, and Paul Sorenson. Developing Engineered Product Support Applications. In Proceedings of the First Software Product Line Conference [201], pages 451–476.

[298] W. Humphrey and P. Feiler. Software Process Development and Enactment: Concepts and Definitions. Technical Report CMU/SEI-92-TR-004, Software Engineering Institute, Carnegie Mellon University, 1992.

[299] INTERNATIONAL BUSINESS MACHINES CORPORATION (IBM). Application Blueprint Definition for C3. Technical Report CDRL 01490-001A, INTERNATIONAL BUSINESS MACHINES CORPORATION (IBM), 1990.

[300] Tuomas Ihme. A ROOM Framework for the Spectrometer Controller Product Line. In Proceedings of Workshop on Object Technology for Product-Line Architectures, Lisbon, Portugal, 1999.

[301] C. Insalaco and W. Tracz. GLUE (Graphical Layout User Environment) User Guide. Technical Report ADAGE-LOR-94-04A, Loral Federal Systems, August 1995.

[302] European Software Institute, editor. Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4), Bilbao, Spain, October 2001.

[303] N. Iscoe. Domain Modeling - Overview and Ongoing Research at EDS. In Proceedings of the Fifteenth International Conference on Software Engineering (ICSE'93), pages 198–200, 1993.

[304] N. Iscoe, G. Williams, and G. Arango. Domain Modeling for Software Engineering. In Proceedings of the Thirteenth International Conference on Software Engineering (ICSE'91), pages 340–343, 1991.

[305] Kiyoshi Itoh, Toyohiko Hirota, Satoshi Kumagai, and Hiroyuki Yoshida, editors. Domain Oriented Systems Development: Principles and Approaches. Gordon and Breach Science Publishers, 1998.

[306] Khaled Jaber, Nader Nada, and David Rine. Product Line Stakeholder Viewpoint Approach and Validation Model. In Proceedings of the ACM Symposium on Applied Computing, pages 871–875, Como, Italy, 2000.

[307] Ivar Jacobson, Martin Griss, and Patrik Jonsson. Software Reuse. Architecture, Process and Organization for Business Success. Addison-Wesley, 1997.

[308] C. Blake Jaktman. Detecting architectural erosion in an evolving product-line architecture. In Proceedings of the First Nordic Software Architecture Workshop, 1998.

[309] S. Jarzabeck. Modeling Multiple Domains in Software Reuse. ACM SIGSOFT Software Engineering Notes, 22(3):65–74, May 1997.

[310] S. Jarzabek. Domain Model-Driven Software Reengineering and Maintenance. Journal of Systems and Software, 23:37–51, 1993. Construction of the domain model is the interesting aspect here.

[311] Mehdi Jazayeri, Alexander Ran, and Frank van der Linden. Software Architecture for Product Families: Principles and Practices. Addison-Wesley, May 2000.

[312] Hans Peter Jepsen and Flemming Nielsen. A Two-Part Architectural Model as Basis for Frequency Converter Product Families. In Third International Workshop on Software Architectures for Product Families [359], pages 31–54.

[313] Isabel John. Building Domain Models from Legacy Documentation Assets. In GCSE Young Researchers Workshop, 2001.

[314] Isabel John. Integrating Legacy Documentation Assets into a Product Line. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 111–122.

[315] K. Kang. Features Analyses: An Approach to Domain Analysis. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, July 1989.

[316] K. Kang. Results from Domain Analysis Workshop Group. Technical report, Reuse in Practice Workshop, Pittsburgh, PA, 1989.

[317] 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, November 1990.

[318] Kyo C. Kang. Feature-Oriented Development of Applications for a Domain. In Proceedings of the Fourth International Conference on Software Reuse, pages 354–355, 1998.

[319] Kyo C. Kang, Kwanwoo Lee, Jaejoon Lee, and Sajoong Kim. Feature Oriented Product Line Software Engineering: Principles and Guidelines. In Domain Oriented Systems Development – Practices and Perspectives. Gordon Breach Science Publishers, 2002.

[320] A. Karhinen, A. Ran, and T. Tallgren. Configuring Designs for Reuse. ACM SIGSOFT Software Engineering Notes, 22(3):199–208, May 1997.

[321] E.-A. Karlsson, editor. Software Reuse: A Holistic Approach. John Wiley & Sons, 1995.

[322] M. Kasunic. Synthesis: A Reuse-Based Software Development Methodology, Process Guide, Version 1.0. Technical report, Software Productivity Consortium Services Corporation, October 1992.

[323] Iliyan Kaytazov, Giancarlo Succi, and Witold Pedrycz. The Implementation of Holmes: A Tool to Support Domain Analysis and Engineering. In Proceedings of the Second International Workshop on Software Product Lines: Economics, Architectures, and Implications [336], pages 93–97. IESE-Report. No. 051.01/E.

[324] B. Keepence and M. Mannion. Using Patterns to Model Variability in Product Families. IEEE Software, pages 102–108, July/August 1999.

[325] Pertti Kellomaki and Tommi Mikkonen. Separating Product Variance and Domain Concepts in the Specification of Software Product Lines. In Proceedings of the International Workshop on Aspects and Dimensional Computing at ECOOP 2000, 2000.

[326] Tomoji Kishi and Natsuko Noda. Aspect-Oriented Analysis for Product Line Architecture. In Proceedings of the First Software Product Line Conference [201], pages 135–145.

[327] W. Kleppinger, D. Tamanaha, and L. Osterweil. A Framework for Understanding the Uses of Process Modeling Formalisms. In Proceedings of the 3rd Irvine Software Symposium (ISS'93), April 1993.

[328] C. Klingler and D. Creps. The Reuse Oriented Software Evolution (ROSE) Process Model, Version 0.5-Draft. Technical report, Software Technology for Adaptable, Reliable Systems (STARS), 1993.

[329] C. Klingler and J. Solderitsch. A Process for Domain Architecture Definition and Asset Implementation. Technical report, Software Technology for Adaptable, Reliable Systems (STARS), 1996.

[330] C. Klingler and J. Solderitsch. Domain Architecture-Based Generation for Ada Reuse (DAGAR) Guidebook. Technical Report STARS-PA19-S007/001/00, Software Technology for Adaptable, Reliable Systems (STARS), June 1996.

[331] Peter Knauber, Jesus Bermejo, G¨ unther B¨ ockle, Julio Cesar Sampaio do Prado, Frank van der Linden, Linda Northrop, Michael Stark, and David M. Weiss. Quantifying Product Line Benefits. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 153–161.

[332] Peter Knauber, Dirk Muthig, Klaus Schmid, and Tanya Widen. Applying Product Line Concepts in Small and Medium-Sized Companies. IEEE Software, 17(5):88–95, September/October 2000.

[333] Peter Knauber and Klaus Pohl, editors. Proceedings of 1. Deutscher Software Produktlinien Workshop (DSPL-1), Kaiserslautern, Germany. Fraunhofer Institute for Experimental Software Engineering (IESE), November 2000. IESE-Report 076.00/E.

[334] Peter Knauber and Giancarlo Succi, editors. Proceedings of Software Product Lines: Economics, Architectures, and Implications. Workshop #15 at 22nd International Conference on Software Engineering (ICSE), Limerick, Ireland, June 2000. Fraunhofer Institute for Experimental Software Engineering (IESE). IESE-Report 070.00/E.

[335] Peter Knauber and Giancarlo Succi. Perspectives on Software Product Lines: Report on Second International Workshop on Software Product Lines: Economics, Architectures, and Implications. In Proceedings of the Second International Workshop on Software Product Lines: Economics, Architectures, and Implications [336], pages 7–13. IESE-Report. No. 051.01/E.

[336] Peter Knauber and Giancarlo Succi, editors. Proceedings of the Second International Workshop on Software Product Lines: Economics, Architectures, and Implications, Toronto, Canada, May 2001. Fraunhofer Institute for Experimental Software Engineering (IESE). IESE-Report. No. 051.01/E.

[337] P. Kogut and P. McLoone. Architecture Acquisition Guidelines. Technical Report STARS-VC-K017R1/001/00, Software Technology for Adaptable, Reliable Systems (STARS), February 1996.

[338] P. Kogut and R. Nilson. Domain Engineering Methods and Tools Handbook Volume I - Methods, Comprehensive Approach to Reusable Defense Software (CARDS). Technical report, Software Technology for Adaptable, Reliable Systems (STARS), 1994.

[339] P. Kogut and R. Nilson. Domain Engineering Methods and Tools Handbook Volume II - Tools, Comprehensive Approach to Reusable Defense Software (CARDS). Technical report, Software Technology for Adaptable, Reliable Systems (STARS), 1994.

[340] Charles W. Krueger. Easing the Transition to Software Mass Customization. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 265–277.

[341] R. Krut. Integrating 001 Tool Support into the Feature-Oriented Domain Analysis Methodology. Technical Report CMU/SEI-93-TR-11, Software Engineering Institute, Carnegie Mellon University, May 1993.

[342] R. Krut and N. Zalman. Domain Analysis Workshop Report for the Automated Prompt and Response System Domain. Technical Report CMU/SEI-96-SR-001, Software Engineering Institute, Carnegie Mellon University, 1996.

[343] Juha Kuusela and Juha Savolainen. Requirements Engineering for Product Families. In Proceedings of the 22nd International Conference on Software Engineering (ICSE), pages 61–69, Limerick, Ireland, June 2000. ACM.

[344] Philippe Lalanda. Style-Specific Techniques to Design Product-Line Architectures. In Proceedings of the First Working IFIP Conference on Software Architecture (WICSA1), San Antonio, TX, 1999.

[345] W. Lam. Achieving Requirements Reuse: A Domain-Specific Approach from Avionics. Journal of Systems and Software, 38:197–209, 1997.

[346] W. Lam. Creating Reusable Architectures: Initial Experience Report. ACM SIGSOFT Software Engineering Notes, 22(4):39–43, July 1997.

[347] W. Lam. Viewpoint-Centered Reuse: Bridging the Gap between Reusability and the Needs of the Reuser. ACM SIGSOFT Software Engineering Notes, 23(1):100–103, January 1998.

[348] W. Lam and J. McDermid. A Summary of Domain Analysis Experience By Way of Heuristics. ACM SIGSOFT Software Engineering Notes, 22(3):54–64, May 1997.

[349] W. Lam and B. Whittle. A Taxonomy of Domain-Specific Reuse Problems and their Resolutions. ACM SIGSOFT Software Engineering Notes, 21(5):72–77, September 1996.

[350] D. Lea. Design Patterns for Avionics Control Systems. Technical Report ADAGE-OSW-94-01, SUNY Oswego and NY CASE Center, November 1994.

[351] R. Leach. Software Reuse. Methods, Models and Costs. McGraw-Hill, 1997. [352] Kwanwoo Lee, Kyo C. Kang, Eunman Koh, Wonsuk Chae, Bokyoung Kim, and Byoung Wook Choi. Domain-Oriented Engineering of Elevator Control Software: A Product Line Practice. In Proceedings of the First Software Product Line Conference [201], pages 3–22.

[353] Kwanwoo Lee, Kyo C. Kang, and Jaejoon Lee. Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In Proceedings of the Seventh International Conference on Software Reuse [233], pages 62–77.

[354] J. C. S. P. Leite. Are Domains Really Cost Effective? In Proceedings of the Ninth Workshop on Institutionalizing Software Reuse, Austin, TX, USA, January 1999.

[355] J. Lettes, C. Klingler, and K. Roth. Federal Technology Leadership Award – A Technology Transition (TT) Success Story. Loral Systems and Software Resource Center Newsletter, January 1996.

[356] B. Lewis and D. McConnell. Reengineering Real-Time Embedded Software onto a Parallel Processing Platform. In Proceedings of the Third Working Conference on Reverse Engineering, pages 11–19. IEEE Computer Society Press, 1996.

[357] Oliver Lewis, Mike Mannion, and William Buchanan. Performance Issues of Variability Design for Embedded System Product Lines. In Proceedings of the International Workshop on Software Product Lines: Economics, Architectures, and Implications [334]. IESE-Report 070.00/E.

[358] R. Lied, L. Pautler, and P. Helmers. Introducing Software Reuse Technology. Bell Labs Technical Journal, pages 188–199, Winter 1997.

[359] Frank van der Linden, editor. Software Architectures for Product Families. Proceedings of the Third International Workshop on Software Architectures for Product Families, LNCS 1951, Las Palmas de Gran Canaria, Spain, March 2000. Springer.

[360] Frank van der Linden. Platform Engineering for the Medical Domain. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 207–221.

[361] Frank van der Linden and Henk Obbink. ESAPS – Engineering Software Architectures, Processes and Platforms for System Families. In Third International Workshop on Software Architectures for Product Families [359], pages 116–125.

[362] M. Lipshutz. Domain Architecture Tutorial: DAGAR (Domain Architecture-based Generation for Ada Reuse), June 1996.

[363] M. Lipshutz, R. Creps, and M. Simos. Organizational Domain Modeling (ODM) Tutorial, January 1997.

[364] C. Lissoni and G. Stellucci. PROFANES: An Experiment with Combined Use of Domain Analysis and Object-Oriented Framework. In Proceedings of the European Reuse Workshop 1997 (ERW'97), pages 97–104, November 1997.

[365] B. Liver and D. Allemang. A Functional Representation for Software Reuse and Design. International Journal of Software Engineering & Knowledge Engineering, 5(2):227–269, 1995.

[366] Lockheed Martin Federal Systems and U.S. Air Force/Space and Warning Systems Center (SWSC). Domain Engineering Guidebook, September 1996.

[367] M. Lowry, A. Philpot, T. Pressburger, and I. Underwood. A Formal Approach to Domain-Oriented Software Design Environments. In Proceedings of the Ninth Knowledge-Based Software Engineering Conference, 1994.

[368] M. Lowry, A. Philpot, T. Pressburger, and I. Underwood. AMPHION: Automatic Programming for Scientific Subroutine Libraries. In Proceedings of the International Symposium on Methodologies for Intelligent Systems, pages 326–355, October 1994.

[369] M. Lowry and J. van Baalen. METAAMPHION: Synthesis of Efficient Domain-Specific Program Synthesis Systems. In Proceedings of the 10th Knowledge-Based Software Engineering Conference, pages 2–10, November 1995.

[370] M. Lowry and J. van Baalen. META-AMPHION: Synthesis of Efficient Domain Specific Program Synthesis Systems. Automated Software Engineering, 4(2):199–241, April 1997.

[371] M. Lubars. A Domain Modeling Representation. Technical Report STP-366-88, Microelectronics and Computer Corporation, Austin, TX, November 1988. [372] M. Lubars. Domain Analysis and Software Systems Modeling, chapter Domain Analysis and Domain Engineering in IDeA. IEEE Computer Society Press, 1991. Book out of print.

[373] C.-H. Lung, J. Cochran, G. Mackulak, and J. Urban. Computer Simulation Software Reuse by Generic/Specific Domain Modeling Approach. International Journal of Software Engineering & Knowledge Engineering, 4(1):81– 102, 1994.

[374] C.-H. Lung and J. Urban. An Expanded View of Domain Modeling for Software Analogy, 1995.

[375] Robyn R. Lutz. Toward Safe Reuse of Product Family Specifications. In Proceedings of the Fifth ACM SIGSOFT Symposium on Software Reusability (SSR'99), pages 17–26, Los Angeles, CA, USA, May 1999. ACM.

[376] Robyn R. Lutz. Extending the Product Family Approach to Support Safe Reuse. The Journal of Systems and Software, 53(3), September 2000.

[377] R. Macala, L. Stuckey, and D. Gross. Managing Domain-Specific, Product-Line Development. IEEE Software, pages 57–67, May 1996.

[378] Alessandro Maccari and Claudio Riva. Architectural Evolution of Legacy Product Families. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 63–68.

[379] Alessandro Maccari and Antti-Pekka Tuovinen. System Family Architectures: Current Challenges at Nokia. In Third International Workshop on Software Architectures for Product Families [359], pages 106–115.

[380] P. Maccario. Integration of Domain Analysis in a Development Methodology. In Proceedings of the European Reuse Workshop 1997 (ERW'97), pages 109–111, November 1997.

[381] P. Maccario. The Domain Analysis Integrated in an Object Oriented Development Methodology. In Proceedings of the Eighth Workshop on Institutionalizing Software Reuse, 1997.

[382] John MacGregor. A Product Line Process for the Production of Platform Software at Bosch. In Proceedings of the International Workshop on Software Product Lines: Economics, Architectures, and Implications [334]. IESE-Report 070.00/E.

[383] John MacGregor. A Proposal for a Product Line Product Derivation Process. In Proceedings of the Second International Workshop on Software Product Lines: Economics, Architectures, and Implications [336], pages 33–38. IESE-Report. No. 051.01/E.

[384] N. Madhav. Verification of Composition Operations in LILEANNA for DSSA-ADAGE. Technical Report ADAGE-BU-94-01D, Binghamptom University, October 1994.

[385] M. Mannion, B. Keepence, and D. Harper. Using Viewpoints to Define Domain Requirements. IEEE Software, pages 95–102, January 1998.

[386] M. Mannion, B. Keepence, H. Kaindl, and J. Wheadon. Reusing Single System Requirements for Application Family Requirements. In Proceedings of the 21 st International Conference on Software Engineering (ICSE'99), pages 453–462, May 1999.

[387] M. Mannion, O. Lewis, H. Kaindl, G. Montroni, and J. Wheadon. Representing Requirements on Generic Software in an Application Family Model. In W. B. Frakes, editor, Proceedings of the Sixth International Conference on Software Reuse, pages 153–169, June 2000.

[388] F. Maymir-Ducharme. Domain Engineering - Varying Rationales and Approaches. In Proceedings of the Seventh Workshop on Institutionalizing Software Reuse, 1995.

[389] F. Maymir-Ducharme. The Product Line Business Model. In Proceedings of the Eighth Workshop on Institutionalizing Software Reuse, 1997.

[390] F. Maymir-Ducharme, P. Clements, K. Wallnau, and R. Krut. The Unified Information Security (INFOSEC) Architecture (UIA) Gad y Project. Technical Report CMU/SEI-95-TR-015, Software Engineering Institute, Carnegie Mellon University, October 1995.

[391] G. Mayobre. Maximizing Reuse with an Evolution Oriented Domain Engineering. In Proceedings of the Sixth Workshop on Institutionalizing Software Reuse, 1993.

[392] D. McAllester. An ADAGE System Vision. Technical Report ADAGE-MIT-92-01, MIT, June 1992.

[393] D. McAllester. Variational Attribute Grammars for Computer Aided Design (Release 3.0). Technical Report ADAGE-MIT-94-02, MIT, 1994.

[394] R. McCabe, G. Campbell, and S. Wartik. Domain Engineering Guidebook. Technical Report SPC-92041-CMC, Software Productivity Consortium Services Corporation, Herndon, Virginia, December 1992.

[395] Dave McComas, Stephen Leake, Michael Stark, Maurizio Morisio, Guilherne Travassos, and Michael White. Addressing Variability in a Guidance, Navigation, and Control Flight Software Product Line. In Proceedings of the First Software Product Line Conference [201], page 552.

[396] D. McConnell, B. Lewis, and L. Gray. Re-Engineering a Single Threaded Embedded Missile Application Onto a Parallel Processing Platform Using MetaH. In Proceedings of the 4th International Workshop on Parallel and Distributed Real-Time Systems, April 1996.

[397] John D. McGregor. Validating Domain Models. Journal of Object-Oriented Programming, pages 12–17, July/August 1999.

[398] John D. McGregor. Building Reusable Test Assets for a Product Line. Tutorial. In First Software Product Line Conference, August 2000.

[399] John D. McGregor. Structuring Test Assets in a Product Line Effort. In Proceedings of the Second International Workshop on Software Product Lines: Economics, Architectures, and Implications [336], pages 89–92. IESE-Report. No. 051.01/E.

[400] John D. McGregor. Testing a Software Product Line. Technical Report CMU/SEI-2001-TR-022, Software Engineering Institute, Carnegie Mellon University, December 2001.

[401] M. D. McIlroy. Mass-Produced Software Components. In J. M. Buxton, P. Naur, and B. Randell, editors, Software Engineering Concepts and Techniques, 1968 NATO Conference on Software Engineering, pages 88–98, 1976.

[402] J. Meekel, T. Horton, R. France, C. Mellone, and S. Dalvi. From Domain Models to Architecture Frameworks. ACM SIGSOFT Software Engineering Notes, 22(3):75–80, May 1997.

[403] Julio Mellado and Juan C. Due˜ nas. Automated Validation Environment for a Product Line of Railway Traffic Control Systems. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 389–397.

[404] Julio Mellado, Manuel Sierra, Ana Romera, and Juan C. Duenas. Railway-Control Product Families: The Alcatel TAS Platform Experience. In Third International Workshop on Software Architectures for Product Families [359], pages 55–63.

[405] E. Mettala and M. Graham. The Domain-Specific Software Architecture Program. Technical Report CMU/SEI-92-SR-009, Software Engineering Institute, Carnegie Mellon University, 1992.

[406] R. Might. Domain Models: What are They? How are They Used?

[407] A. Mili, S. Yacoub, E. Addy, and H. Mili. Toward an Engineering Discipline of Software Reuse. IEEE Software, pages 22–31, September/October 1999.

[408] Ali Mili and Sherif M. Yacoub. A Comparative Analysis of Domain Engineering Methods: A Controlled Case Study. In Proceedings of the International Workshop on Software Product Lines: Economics, Architectures, and Implications [334]. IESE-Report 070.00/E.

[409] Hafedh Mili, Ali Mili, Sherif Yacoub, and Edward Addy. Reuse-Based Software Engineering: Techniques, Organization, and Controls. John Wiley & Sons, 2001.

[410] S. Moody. Exploring Frameworks and Representations for Domain Specific Automatic Code Generation. In Proceedings of the Fourth Workshop on Institutionalizing Software Reuse, 1991.

[411] J. Moore and S. Bailin. Domain Analysis and Software Systems Modeling, chapter Domain Analysis: Framework for Reuse, pages 179–203. IEEE Computer Society Press, 1991. Book out of print.

[412] Yoshitomi Morisawa. A Computing Model of Product Lines for Distributed Processing Systems, its Product Sets, and its Applications. In Proceedings of the First Software Product Line Conference [201], pages 371–394. [413] Maurizio Morisio, Guilherme H. Travassos, and Michael E. Stark. Extending UML to Support Domain Analysis.In Proceedings of the Fifteenth IEEE International Conference on Automated Software Engineering (ASE'00), 2000.

[414] E. Mugisa. A Reuse Triplet for Systematic Software Reuse. ACM SIGSOFT Software Engineering Notes, 22(4):65–69, July 1997.

[415] Dirk Muthig. A Technique for Variability and Decision Modeling Facilitating the Incremental Introduction of Product Line Engineering. In Proceedings of the GCSE 2000 Young Researchers Workshop, 2000.

[416] Dirk Muthig. An Incremental Transition Strategy is Key to a Successful Introduction of Product Line Engineering. In Proceedings of the Second International Workshop on Software Product Lines: Economics,Architectures, and Implications [336], pages 39–42. IESE-Report. No. 051.01/E.

[417] Dirk Muthig and Joachim Bayer. Helping Small and Medium-Sized Enterprises in Moving Towards Software Product Lines. In Proceedings of the International Workshop on Software Product Lines: Economics, Architectures, and Implications [334]. IESE-Report 070.00/E.

[418] K. Nakakoji. Software Reuse in Integrated, Domain-Oriented Knowledge-based Design Environments. In Proceedings of the Fourth Workshop on Institutionalizing Software Reuse, 1991.

[419] M. Natori, A. Kagaya, and S. Honiden. Reuse of Design Processes Based on Domain Analysis. In Proceedings of the Fourth International Conference on Software Reuse, pages 31–40, April 1996.

[420] J. Neighbors. The Draco Approach to Constructing Software from Reusable Components. IEEE Transactions on Software Engineering, 10(5):564–573, September 1984.

[421] J. Neighbors. Report on the Domain Analysis Working Group Session. In Proceedings of the Workshop on Software Reuse, Boulder, CO, October 1987. Rocky Mountain Institute of Software Engineering.

[422] J. Neighbors. Domain Analysis and Software Systems Modeling, chapter DRACO: A Method for Engineering Reusable Software Systems, pages 34–52. IEEE Computer Society Press, 1991. Book out of print.

[423] J. Neighbors. The Commercial Application of Domain Analysis. In Proceedings of the Fifth Workshop on Institutionalizing Software Reuse, 1992.

[424] J. Neighbors. The Evolution from Software Components to Domain Analysis. International Journal of Software Engineering & Knowledge Engineering, 2(3):325–354, September 1992.

[425] Eila Niemela. A Component Framework of a Distributed Control Systems Family. PhD thesis, Faculty of Science, University of Oulu, Oulu, Finland, 1999.

[426] Eila Niemel¨a and Tuomas Ihme. Product Line Software Engineering of Embedded Systems. In Proceedings of the ACM SIGSOFT Symposium on Software Reusability (SSR'01), pages 118–125, May 2001.

[427] R. Nilson, P. Kogut, and G. Jackelen. Component Provider's and Tool Developer's Handbook Central Archive for Reusable Defense Software (CARDS). Technical Report STARS-VC-B017/001/00, Software Technology for Adaptable, Reliable Systems (STARS), March 1994.

[428] J. Ning, K. Miriyala, and W. Kozaczynski. An Architecture-Driven, Business-Specific, and Component-Based Approach to Software Engineering. In Proceedings of the Third International Conference on Software Reuse, pages 84–93, 1994.

[429] Robert L. Nord. Meeting the Product Line Goals for an Embedded Real-Time System. In Third International Workshop on Software Architectures for Product Families [359], pages 19–29.

[430] Liam O'Brien. Architecture Reconstruction to Support a Product Line Effort: A Case Study. Technical Note CMU/SEI-2001-TN-015, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, July 2001.

[431] J. O'Connor, C. Mansour, J. Turner-Harris, and G. Campbell. Exploring Systematic Reuse for Command and Control Systems. Technical Report SPC-92020-CMC, Software Productivity Consortium Services Corporation, April 1994.

[432] J. Palmer and Y. Liang. Approaches to Domain Model Construction. In Proceedings of the Thirteenth International Conference on Software Engineering (ICSE'91), April 1991.

[433] J. Palmer and Y. Liang. Domain Model Construction and Learning from Examples. Technical report, Center for Software Systems Engineering, George Mason University, September 1991.

[434] D. Parnas. On the Design and Development of Program Families. IEEE Transactions on Software Engineering, SE-2(1):1–9, March 1976.

[435] Alessandro Pasetti and Wolfgang Pree. Two Novel Concepts for Systematic Product Line Development. In Proceedings of the First Software Product Line Conference [201], pages 249–270.

[436] T. Payton. STARS Conceptual Framework for Reuse Processes (CFRP), Volume I: Definition. Technical Report STARS-VC-A018/001/00, Software Technology for Adaptable, Reliable Systems (STARS), October 1993.

[437] T. Payton. STARS Conceptual Framework for Reuse Processes (CFRP) Volume II: Application. Technical Report STARS-VC-A018/002/00, Software Technology for Adaptable, Reliable Systems (STARS), September 1993.

[438] D. Perry. Generic Architecture Descriptions for Product Lines. In F. van der Linden, editor, Development and Evolution of Software Architectures for Product Lines. Proceedings of the Second International ESPRIT ARES Workshop, LNCS 1429, pages 51–56. Springer, 1998.

[439] D. E. Perry. Some Holes in the Emperor's Reused Clothes. In Proceedings of the Ninth Workshop on Institutionalizing Software Reuse, Austin, TX, USA, January 1999.

[440] Dewayne E. Perry. A Product Line Architecture for a Network Product. In Third International Workshop on Software Architectures for Product Families [359], pages 41–54.

[441] H. Perunka, E. Niemilae, and J. Kalaoja. Feature-Oriented Approach to Design Reusable Software Architectures and Components in Embedded Systems. In Proceedings of the European Reuse Workshop 1997 (ERW'97), pages 71–76, November 1997.

[442] S. Peterson and J. Stanley. Mapping a Domain Model and Architecture to a Generic Design. Technical Report CMU/SEI-94-TR-8, Software Engineering Institute, Carnegie Mellon University, May 1994.

[443] J. Petro, R. Whitehead, R. Snider, L. Meheran, and N. Keller. Command Center Product Line Architecture Report (CCPLAR). Technical Report STARS-VC-K017R1/001/00, Comprehensive Approach to Reusable Defense Software (CARDS), September 1996.

[444] Ilka Philippow and Matthias Riebisch. Systematic Definition of Reusable Architectures. In Proceedings of the 8th IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2001), pages 128–135, April 2001.

[445] C. Pidgeon. Organizing and Enabling Domain Engineering to Facilitate Software Maintenance. In Proceedings of the Eighth Workshop on Institutionalizing Software Reuse, 1997.

[446] Klaus Pohl, Mathias Brandenburg, and Alexander G¨ utlich. Scenario-Based Change Integration in Product Family Development. In Proceedings of the Second International Workshop on Software Product Lines: Economics, Architectures, and Implications [336], pages 43–47. IESE-Report. No. 051.01/E.

[447] Klaus Pohl and Andreas Reuys. Considering Variabilities During Component Selection in Product Family Development. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 21–36.

[448] Herman Postema. Platform Based Product Development. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 377–388.

[449] J. Poulin. Populating Software Repositories: Incentives and Domain-Specific Software. Journal of Systems and Software, 30:187–199, 1995.

[450] Paolo Predonzani, Giancarlo Succi, and Tullio Vernazza. Strategic Software Production With Domain-Oriented Reuse. Artech House, 2000.

[451] W. Pree. Lean Product-Line Architectures for Client-Server Systems - Concepts and Experience. In L. Barroca, J. Hall, and P. Hall, editors, Software Architectures, pages 145–157. Springer, 1999.

[452] T. Pressburger and M. Lowry. Automatic Domain-Oriented Software Design Using Formal Methods. In Proceedings of the Software Systems in Engineering Energy Sources Technology Conference and Exhibition, pages 33–42, January 1995.

[453] R. Prieto-Diaz. Domain Analysis For Reusability. In Proceedings of the Eleventh Annual International Computer Software and Application Conference, pages 63–69, 1987. The focus is on the DA process, NOT on reuse. [454] R. Prieto-Diaz. Domain Analysis: An Introduction. ACM SIGSOFT Software Engineering Notes, 15(2):47, April 1990.

[455] R. Prieto-Diaz. STARS Reuse Library Process Model: Domain Analysis, March 1991.

[456] R. Prieto-Diaz. Software Reuse: Issues and Experiences. American Programmer, 6(8):10–18, August 1993.

[457] R. Prieto-Diaz. Some Experiences in Domain Analysis. In Proceedings of the Sixth Workshop on Institutionalizing Software Reuse, 1993.

[458] R. Prieto-Diaz. Status Report: Software Reusability. IEEE Software, 10(3):61–66, 1993.

[459] R. Prieto-Diaz and J. Neighbors. Module Interconnection Languages. Journal of Systems and Software, 6(4):307–334, November 1986.

[460] B. J. Pronk. Medical Product Line Architectures – 12 years of experience. In Proceedings of the First Working IFIP Conference on Software Architecture, pages 357–367, San Antonio, TX, USA, February 1999. Kluwer Academic Publishers.

[461] B. J. Pronk. An Interface Based Platform Approach. In Proceedings of the First Software Product Line Conference [201], pages 331–351.

[462] Anu Purhonen. Quality Attribute Taxonomies for DSP Software Architecture Design. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 223–233.

[463] M. Ramachandran. Domain-Specific Software Architecture Based on a Building Block Method. In Proceedings of the Seventh Workshop on Institutionalizing Software Reuse, 1995.

[464] M. Ramachandran and W. Fleischer. Design for Large Scale Software Reuse: An Industrial Case Study. In Proceedings of the Fourth International Conference on Software Reuse, pages 104–111, April 1996.

[465] Ramkumar Ramaswamy and Uttara Nerurkar. Creating Malleable Architectures for Application Software Product Families. In Proceedings of Workshop on Object Technology for Product-Line Architectures, Lisbon, Portugal, 1999.

[466] Balasubramaniam Ramesh, Amrit Tiwana, and Kannan Mohan. Supporting Information Product and Service Families with Traceability. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 339–350.

[467] R. Randall, R. Ekman, S. Kent, and G. Turner. Integrating a SEE for Megaprogramming: Lessons Learned. In Proceedings of the Annual Software Technology Conference (STC'95), April 1995.

[468] M. Riebisch, K. B¨ollert, D. Streitferdt, and B. Franczyk. Extending the UML to Model System Families. In M. M. Tanik and A. Ertas, editors, Fifth World Conference on Integrated Design and Process Technology, 2000.

[469] M. Riebisch, Detlef Streiferdt, and Ilka Philippow. Feature Scoping for Product Lines. In Proceedings of the International Workshop on Product Line Engineering - The Early Steps: Planning, Modeling, and Managing (PLEES'01) [496]. IESE-Report No. 050.01/E.

[470] Matthias Riebisch and Ilka Philippow. Evolution of Product Lines Using Traceability. In Proceedings of the Workshop on Engineering Complex Object-Oriented Systems for Evolution at OOPSLA 2001, 2001.

[471] David Rine, David Fortini, Nader Nada, and Mahmoud Elish. Test Framework Product Line for Low Earth Orbit Satellite Constellations. In Proceedings of the Second International Workshop on Software Product Lines: Economics, Architectures, and Implications [336], pages 49–67. IESE-Report. No. 051.01/E.

[472] S. Robak, B. Franczyk, and K. Politowicz. Extending UML for Modeling Variabilities for System Families. International Journal of Applied Mathematics and Computer Science, 2001.

[473] J. Robbins, D. Hilbert, and D. Redmiles. Extending Design Environments to Software Architecture Design. Automated Software Engineering, 5(3):261–290, July 1998.

[474] D. Robertson. Domain Specific Problem Description. In Proceedings of the Eighth Conference on Software Engineering and Knowledge Engineering (SEKE'96), pages 206–213, 1996.

[475] W. Robinson. Integrating Multiple Specifications Using Domain Goals. Technical Report CIS-TR-89-03, University of Oregon, Eugene, Oregon, February 1989. [476] W. Rolling. A Preliminary Annotated Bibliography on Domain Engineering. ACM SIGSOFT Software Engi-neering Notes, 19(3):82–84, July 1994.

[477] W. Rossak, V. Kirova, L. Jololian, H. Lawson, and T. Zemel. A Generic Model for Software Architectures. IEEE Software, pages 84–92, August 1997.

[478] S. Rugaber. Domain Analysis and Reverse Engineering. Technical report, Georgia Institute of Technology, College of Computing, January 1994.

[479] Serge Salicki and Nicolas Farcet. Expression and Usage of the Variability in the Software Product Lines. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 287–297.

[480] Andrea Savigni. Evaluating the Kaleidoscope Product-Line Architecture for Monitoring and Control Systems. In Proceedings of the Second International Workshop on Software Product

Lines: Economics, Architectures, and Implications [336], pages 69–73. IESE-Report. No. 051.01/E.

[481] Juha Savolainen and Juha Kuusela. Violatility Analysis Framework for Product Lines. In Proceedings of the ACM SIGSOFT Symposium on Software Reusability (SSR'01), pages 133–141, May 2001.

[482] Stephen R. Schach and Amir Tomer. Development/Maintenance/Reuse: Software Evolution in Product Lines. In Proceedings of the First Software Product Line Conference [201], pages 437–450.

[483] Michael Schlick and Andreas Hein. Knowledge Engineering in Software Product Lines. In European Conference on Artifical Intelligence (ECAI 2000), 2000.

[484] K. Schmid and C. Gacek. Implementation Issues in Product Line Scoping. In W. B. Frakes, editor, Proceedings of the Sixth International Conference on Software Reuse, pages 170–189, June 2000.

[485] K. Schmid and M. Schank. PuLSE-BEAT - A Decision Support Tool for Scoping Product Lines. In Third International Workshop on Software Architectures for Product Families [359], pages 64–74.

[486] K. Schmid and T. Widen. Customizing the PuLSE Product Line Approach to the Demands of an Organization. In Reidar Conradi, editor, Software Process Technology, 7th European Workshop, EWSPT'2000, LNCS 1780, pages 221–238, Kaprun, Austria, February 2000. Springer.

[487] Klaus Schmid. Multi-Staged Scoping for Software Product Lines. In Proceedings of the International Workshop on Software Product Lines: Economics, Architectures, and Implications [334]. IESE-Report 070.00/E.

[488] Klaus Schmid. Product Line Mapping Report. Technical Report IESE-Report No. 028.00/E, Fraunhofer Institute for Experimental Software Engineering (IESE), October 2000.

[489] Klaus Schmid. Scoping Software Product Lines - An Analysis of an Emerging Technology. In Proceedings of the First Software Product Line Conference [201], pages 513–532.

[490] Klaus Schmid. A Framework for Product Line Quality Model Development. The PuLSE-Eco Meta Quality Model. Technical Report IESE-Report No. 047.00/E, Fraunhofer Institute for Experimental Software Engineering (IESE), June 2001.

[491] Klaus Schmid. An Assessment Approach to Analyzing Benefits and Risks of Product Line. In Proceedings of the 25th Annual International Computer Software and Applications Conference (COMPSAC 2001), pages 525–530, 2001.

[492] Klaus Schmid. An Initial Model of Product Line Economics. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 37–47.

[493] Klaus Schmid. People Issues in Developing Software Product Lines. In Proceedings of the Second International Workshop on Software Product Lines: Economics, Architectures, and Implications [336], pages 75–80. IESE-Report. No. 051.01/E.

[494] Klaus Schmid. Integrating Reference Architecture Definition and Reuse Investment Planning. In Proceedings of the Seventh International Conference on Software Reuse [233], pages 137–152.

[495] Klaus Schmid, Ulrike Becker-Kornstaedt, Peter Knauber, and Florian Bernauer. Introducing a software modeling concept in a medium-sized company. In Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000), pages 558–567, Limerick, Ireland, 2000.

[496] Klaus Schmid and Birgit Geppert, editors. Proceedings of the PLEES'01. International Workshop on Product Line Engineering - The Early Steps: Planning, Modeling, and Managing. Fraunhofer Institute for Experimental Software Engineering (IESE), September 2001. IESE-Report No. 050.01/E.

[497] Klaus Schmid and Isabel John. Product Line Development as a Rational, Strategic Decision. In Proceedings of the International Workshop on Product Line Engineering - The Early Steps: Planning, Modeling, and Managing (PLEES'01) [496]. IESE-Report No. 050.01/E.

[498] Detlef Schmitt. A Framework Development Process for Product-Line Architectures. In Net.Object Days 2001, Erfurt, Germany, September 2001.

[499] K. Schnell, N. Zalman, and A. Bhatt. Transitioning domain analysis: An industry experience. Technical Report CMU/SEI-96-TR-009, Software Engineering Institute, Carnegie Mellon University, June 1996.

[500] M. Schuetze, J. P. Riegel, and G. Zimmermann. A Pattern-Based Application Generator for Building Simulation. ACM SIGSOFT Software Engineering Notes, 22(6):468–482, November 1997.

[501] R. Selby. Amadeus Support for Measurement of ADAGE Processes. Technical Report ADAGE-UCI-92-04, University of California, Irvine, CA, 1992.

[502] David C. Sharp. Component Based Product Line Development of Avionics Software. In Proceedings of the First Software Product Line Conference [201], pages 353–369.

[503] Y. Shimizu, N. Fujimaki, and M. Hirayama. A Systematic Approach to Domain-Oriented Software Development. In Proceedings of the Twentieth International Conference on Software Engineering (ICSE'98), pages 499–502, April 1998.

[504] S. Shlaer and S. Mellor. An Object-Oriented Approach to Domain Analysis. ACM SIGSOFT Software Engineering Notes, 14(5):66–77, July 1989.

[505] M. Simos. The Domain-Oriented Software Life Cycle: Towards an Extended Process Model for Reusability. In Proceedings of the Workshop on Software Reuse, Boulder, CO, October 1987.

[506] M. Simos. Domain Analysis and Software Systems Modeling, chapter The Growing of an Organon: A Hybrid Knowledge-Based Technology and Methodology for Software Reuse, pages 204–221. IEEE Computer Society Press, 1991. Book out of print.

[507] M. Simos. Towards An Industry-Wide Consensus Reuse Process Model. In Proceedings of the Fifth Workshop on Institutionalizing Software Reuse, 1992.

[508] M. Simos. Where the Rubber Meets the Road: Applying Organization Domain Modeling (ODM) on the STARS Army/Unisys Demonstration Project. Technical report, Software Technology for Adaptable, Reliable Systems (STARS), 1994.

[509] M. Simos. Organization Domain Modeling (ODM): Formalizing the Core Domain Modeling Life Cycle. In Mansur Samadzadeh, editor, Proceedings of the ACM SIGSOFT Symposium on Software Reusability (SSR'95), pages 196–205, April 1995.

[510] M. Simos. Lateral Domains: Beyond Product-Line Thinking. In Proceedings of the Eighth Workshop on Institutionalizing Software Reuse, 1997.

[511] M. Simos, D. Allemang, C. Hammons, L. Mantock, C. Klingler, L. Levine, and D. Creps. Canvas Knowledge Acquisition Guidebook, Version 1.0. Software Technology for Adaptable, Reliable Systems (STARS), August 1996.

[512] M. Simos, D. Allemang, C. Hammons, L. Mantock, C. Klingler, L. Levine, and D. Creps. Canvas Knowledge Acquisition Guidebook, Version 2.0. Software Technology for Adaptable, Reliable Systems (STARS), December 1996.

[513] M. Simos and J. Anthony. Weaving the Model Web: A Multi-Modeling Approach to Concepts and Features in Domain Engineering. In Proceedings of the Fifth International Conference on Software Reuse, pages 94–102, Vancouver, BC, Canada, June 1998.

[514] M. A. Simos. Domain Envisioning: A Lightweight, Incremental Approach to Getting a Company Started with with Systematic Reuse. In Proceedings of the Ninth Workshop on Institutionalizing Software Reuse, Austin, TX, USA, January 1999. [515] M. Sitaraman. Conventional Domain Analysis Limits Reusability. In Proceedings of the Eighth Workshop on Institutionalizing Software Reuse, 1997.

[516] D. Smith and E. Parra. Transformational Approach to Transportation Scheduling. In Proceedings of the 8th Knowledge-Based Software Engineering Conference, pages 60–68. IEEE Computer Society Press, September 1993.

[517] Software Engineering Laboratory. The Generalized Support Software (GSS) - A Description of Its Current Software Development Process, February 1996.

[518] Software Productivity Consortium Services Corporation. Reuse Adoption Guidebook, Version 02.00.05, November 1993.

[519] Software Productivity Consortium Services Corporation. Reuse-Driven Software Processes Guidebook, Version 02.00.03, November 1993.

[520] Software Technology for Adaptable, Reliable Systems (STARS). Army STARS Demonstration Project Experience Report, February 1995.

[521] Software Technology for Adaptable, Reliable Systems (STARS). Navy/STARS Demonstration Project Experience Report, March 1995.

[522] Software Technology for Adaptable, Reliable Systems (STARS). Space Command and Control Architectural Infrastructure (SCAI) Air Force / STARS Demonstration Project - Experience Report, March 1995.

[523] Software Technology for Adaptable, Reliable Systems (STARS). Army STARS Demonstration Project Experience Report, April 1996.

[524] Software Technology for Adaptable, Reliable Systems (STARS). Enhanced Domain Generation Environment (EDGE) Users Manual, Version 2.1, April 1996.

[525] Software Technology for Adaptable, Reliable Systems (STARS). Organization Domain Modeling (ODM) Guide-book, Version 2.0, June 1996.

[526] J. Solderitsch. An Organon: Intelligent Reuse of Software Assets and Domain Knowledge. In Proceedings of the Fourth Workshop on Institutionalizing Software Reuse, 1991.

[527] J. Solderitsch. An Architecture-Centric Approach to Domain Engineering. In Proceedings of the first International Workshop on Architectures for Software Systems, pages 246–248, April 1995.

[528] J. Solderitsch. Bridging the Gap Between Domain Modeling and Domain Architecture Definition. Technical Report STARS-PA19-S004R1/001/00, Software Technology for Adaptable, Reliable Systems (STARS), April 1996.

[529] J. Solderitsch, K. Wallnau, and J. Thalhamer. Constructing Domain-Specific Ada Reuse Libraries. In Proceedings of the Seventh Annual National Conference on Ada Technology, Ft. Monmouth, NJ, March 1989. U.S. Army Communications-Electronics Command.

[530] J. Solderitsch, G. Wickman, D. Kweder, and M. Horton. An Architecture and Generator for an Army IEW Domain. In Proceedings of the Software Technology Conference, 1995.

[531] Y. Srinivas. What is a Domain? Technical Report ASE-RTP-102, University of California, Irvine, Department of Information and Computer Science, October 1988.

[532] Y. Srinivas. Domain Analysis and Software Systems Modeling, chapter Algebraic Specification for Domains, pages 90–124. IEEE Computer Society Press, 1991. Book out of print.

[533] Mike Stark, Dave McComas, Guilherme H. Travassos, and Maurizio Morisio. Developing a Product Line Approach for Flight Software. In Proceedings of the 25th Annual Software Engineering Workshop, pages 1–13, 2000.

[534] Alan Stephenson, Darren Buttle, and John McDermid. Extending Commonality Analysis for Embedded Control System Families. In Third International Workshop on Software Architectures for Product Families [359], pages 204–211.

[535] Zo¨e Stephenson and John McDermid. Tracing Features with Decision Models. In Proceedings of the Second International Workshop on Software Product Lines: Economics, Architectures, and Implications [336], pages 81–84. IESE-Report. No. 051.01/E. [536] Perdita Stevens. UML for Describing Product-Line Architectures? In ECOOP Workshop on Object Technology for Product Line Architectures, June 1999.

[537] M. Stickel, R. Waldinger, M. Lowry, T. Pressburger, and I. Underwood. Deductive Composition of Astronomical Software from Subroutine Libraries. In Proceedings of the Conference on Automated Deduction, June 1994.

[538] Christoph Stoermer and Markus Roeddiger. Introducing Product Lines in Small Embedded Systems. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 97– 109.

[539] D. A. Stuart. Reuse and Analysis. In Proceedings of the Ninth Workshop on Institutionalizing Software Reuse, Austin, TX, USA, January 1999.

[540] Douglas Stuart, Wonhee Sull, and T. W. Cook. Dependency Navigation in Product Lines Using XML. In Third International Workshop on Software Architectures for Product Families [359], pages 82–94.

[541] Douglas Stuart, Wonhee Sull, Steve Pruitt, Deborah Cobb, Fred Waskiewicz, and T. W. Cook. The SSEP Toolset for Product Line Development: An XML Based Architecture Centric Approach. In Proceedings of the First Software Product Line Conference [201], pages 413–435.

[542] Giancarlo Succi, Jason Yip, and Eric Liu. Analysis of the Essential Requirements for a Domain Analysis Tool. In Proceedings of the International Workshop on Software Product Lines: Economics, Architectures, and Implications [334]. IESE-Report 070.00/E.

[543] A. Sutcliffe and N. Maiden. Domain Modeling for Reuse. In Proceedings of the Third International Conference on Software Reuse, pages 169–177, 1994.

[544] A. Sutcliffe and N. Maiden. The Domain Theory for Requirements Engineering. IEEE Transactions on Software Engineering, 24(3):174–196, March 1998.

[545] Alistair Sutcliffe. Domain Analysis for Software Reuse. The Journal of Systems and Software, 50(3):175–199, 2000.

[546] Mikael Svahnberg and PerOlof Bengtsson. Software product lines from customer to code. Technical Report HK/R-RES-00/1-SE, Department of Software Engineering and Computer Science, University of Karlskrona/Ronneby, Ronneby, Sweden, January 2000.

[547] Mikael Svahnberg and Jan Bosch. A Case Study on Product Line Architecture Evolution. In Proceedings of the Second Nordic Workshop on Software Architecture (NOSA'99), August 1999.

[548] Mikael Svahnberg and Jan Bosch. Characterizing Evolution in Product Line Architectures. In N. Debnath and R. Lee, editors, Proceedings of the 3rd annual IASTED International Conference on Software Engineering and Applications, pages 92–97, Anaheim, CA, October 1999.

[549] Mikael Svahnberg and Jan Bosch. Evolution in Software Product Lines: Two Cases. Journal of Software Maintenance: Research and Practice, 11(6):391–422, 1999.

[550] Mikael Svahnberg and Jan Bosch. Issues Concerning Variability in Software Product Lines. In Third International Workshop on Software Architectures for Product Families [359], pages 146–157.

[551] Mikael Svahnberg and Michael Mattson. Conditions and Restrictions for Product Line Generation Migration. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 141–152.

[552] F. Svoboda. Reuse-based Reengineering: Notes from the Underground. In Proceedings of the Fourth Systems Reengineering Technology Workshop, February 1994.

[553] F. Svoboda, F. Maymir-Ducharme, and J. Poulin. SRI Workshop Summary: Domain Analysis in the DoD. ACM SIGSOFT Software Engineering Notes, 21(1):55–67, January 1996.

[554] H. Tan and T. Ling. Components Reuse for Data-Intensive Business Programs Through an Object-Oriented Architecture. Journal of Systems and Software, pages 3–20, 1996.

[555] R. Taylor. Process Formalization and Domain Engineering. Technical Report ADAGE-UCI-95-01, University of California, Irvine, CA, June 1995.

[556] R. Taylor, W. Tracz, and L. Coglianese. Software Development Using Domain-Specific Software Architectures. Technical Report ADAGE-UCI-94-01C, University of California, Irvine, CA,

1994. [557] A. Terry, F. Hayes-Roth, L. Erman, N. Coleman, M. Devito, G. Papanogopoulos, and B. Hayes-Roth. Overview of Teknowledge's Domain-Specific Software Architecture Program. ACM SIGSOFT Software Engineering Notes, 19(4):68–76, October 1994.

[558] A. Terry, R. London, G. Papanagopoulos, and M. Devito. The ARDEC/Teknowledge Architecture Description Language. Technical report, Teknowledge Federal Systems, 1995.

[559] S. Thibault and C. Consel. A Framework for Application Generator Design. ACM SIGSOFT Software Engineering Notes, 22(3):131–135, May 1997.

[560] S. A. Thibault, R. Marlet, and C. Consel. Domain-Specific Languages: From Design to Implementation Application to Video Device Drivers Generation. IEEE Transactions on Software Engineering, 25(3):363–377, May 1999.

[561] Steffen Thiel. On the Definition of a Framework for an Architecting Process Supporting Product Family Development. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 123–139.

[562] Steffen Thiel and Fabio Peruzzi. Starting a Product Line Approach for an Envisioned Market: Research and Experience in an Industrial Environment. In Proceedings of the First Software Product Line Conference [201], pages 495–512.

[563] Jeffrey M. Thompson and Mats P. E. Heimdahl. Ideas on How Product-Line Engineering can be Extended. In Proceedings of the Second International Workshop on Software Product Lines: Economics, Architectures, and Implications [336], pages 85–88. IESE-Report. No. 051.01/E.

[564] Peter Toft, Derek Coleman, and Joni Ohta. A Cooperative Model for Cross-Divisional Product Development for a Software Product Line. In Proceedings of the First Software Product Line Conference [201], pages 111–132.

[565] A. Toptsis. Heuristic Organization and Domain Analysis of Software Repositories. International Journal of Software Engineering & Knowledge Engineering, 5(2):193–210, 1995.

[566] W. Tracz. Domain-Specific Software Architecture (DSSA) Frequently Asked Questions (FAQ). ACM SIGSOFT Software Engineering Notes, 19(2):52–56, April 1994.

[567] W. Tracz. An Environment to Support Domain-Specific Software Architectures. Technical Report ADAGE-LOR-94-02, Loral Federal Systems, 1995.

[568] W. Tracz. DSSA Pedagogical Example. ACM SIGSOFT Software Engineering Notes, 20(3):49–62, July 1995.

[569] W. Tracz. Highlights from the DoD Product Line Practice Workshop Product Lines: Bridging the Gap –Commercial Success to DoD Practice. ACM SIGSOFT Software Engineering Notes, 23(3):29–31, May 1998.

[570] W. Tracz, P. Angeline, S. Shafer, and L. Coglianese. Experience Using an Avionics Domain-Specific Software Architecture. Technical Report ADAGE-LOR-94-14, Loral Federal Systems, 1994.

[571] W. Tracz and L. Coglianese. An Outline for a Domain Specific Software Architecture Engineering Process. In Proceedings of the Fourth Workshop on Institutionalizing Software Reuse, 1991.

[572] W. Tracz and L. Coglianese. A CASE for Domain-Specific Software Architectures. In Proceedings of the Fifth Workshop on Institutionalizing Software Reuse, 1992.

[573] W. Tracz and L. Coglianese. Domain-Specific Software Architecture Engineering Process Guidelines. Technical Report ADAGE-IBM-92-02, Loral Federal Systems, 1992.

[574] W. Tracz and L. Coglianese. DSSA-ADAGE Operational Scenarios and System Vision. Technical Report ADAGE-IBM-92-01B, IBM Federal Systems Company, April 1992.

[575] W. Tracz and L. Coglianese. An Adaptable Software Architecture for Integrated Avionics. Technical Report ADAGE-IBM-93-03, Loral Federal Systems, 1993.

[576] W. Tracz and L. Coglianese. DOMAIN (DOmain Model All INtegrated) – A DSSA Domain Analysis Tool. Technical Report ADAGE-LOR-94-11, Loral Federal Systems, February 1995.

[577] W. Tracz, L. Coglianese, and P. Young. A Domain-Specific Software Architecture Engineering Process Outline. ACM SIGSOFT Software Engineering Notes, 18(2):40–49, April 1993. [578] W. Tracz, S. Shafer, and L. Coglianese. Design Records: A Way to Organize Domain Knowledge. In Proceedings of the Sixth Workshop on Institutionalizing Software Reuse, 1993.

[579] W. Tracz, S. Shafer, and L. Coglianese. DSSA-ADAGE Design Records. Technical Report ADAGE-IBM-93-05A, Loral Federal Systems, 1994.

[580] W. Tracz, S. Shafer, and A. Villarica. MEGEN (Module Expression GENerator). Technical Report ADAGE-LOR-94-09, Loral Federal Systems, 1994.

[581] L. Underhill. The Sociology of Megaprogramming: Experiences in Generating an Organizational Learning Enterprise. In Proceedings of the 7th Annual Software Technology Conference (STC'95), April 1995.

[582] P. van den Hamer, F. van der Linden, A. Saunders, and H. te Sligte. An Integral Hierarchy and Diversity Model for Describing Product Family Architectures. In F. van der Linden, editor, Development and Evolution of Software Architectures for Product Lines. Proceedings of the Second International ESPRIT ARES Workshop, LNCS 1429, pages 66–75. Springer, February 1998.

[583] 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), 2001.

[584] Rob van Ommering. Beyond Product Families: Building a Product Population? In Third International Workshop on Software Architectures for Product Families [359], pages 174–185.

[585] Rob van Ommering. Roadmapping a Product Population Architecture. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 49–61.

[586] Rob van Ommering, Frank van der Linden, Jeff Kramer, and Jeff Magee. The Koala Component Model for Consumer Electronics Software. IEEE Computer, pages 78–85, March 2000. Adds new diversity features and partial-evaluation techniques to a standard hierarchical component model, which makes it suitable for developing software product lines.

[587] Tuomo Vehkomaeki and Kari Kaensaelae. A Comparison of Software Product Family Process Frameworks. In Third International Workshop on Software Architectures for Product Families [359], pages 135–145.

[588] Alessandro Dionisi Vici, Nicola Argentieri, Azza Mansour, Massimo d'Alessandro, and John Favaro. FODAcom: An Experience with Domain Analysis in the Italian Telecom Industry. In Proceedings of the Fourth International Conference on Software Reuse, pages 166–175, 1998.

[589] Renato Vinga-Martins. Requirements Traceability for Product Lines. In Proceedings of Workshop on Object Technology for Product-Line Architectures, Lisbon, Portugal, 1999.

[590] W. Vitaletti and E. Guerrieri. Domain Analysis within the ISEC Rapid Center. In Proceedings of the Eighth Annual National Conference on Ada Technology, pages 460–470, Ft. Monmouth, NJ, March 1990. U.S. Army Communications-Electronics Command.

[591] A. von Mayrhauser, R. Mraz, and J. Walls. Domain Based Regression Testing. In Proceedings of the International Conference on Software Maintenance (ICSM'94), pages 26–35, 1994.

[592] A. von Mayrhauser, M. Shumway, P. Ocken, and R. Mraz. On Domain Models for System Testing. In Proceedings of the Fourth International Conference on Software Reuse, 114-123 1996.

[593] W. Waite and A. Sloane. Software Synthesis via Domain-Specific Software Architectures. Technical Report CU-CS-611-92, University of Colorado, September 1992.

[594] L. Walton and J. Hook. Message Specification Language (MSL): A Domain Specific Design Language for Message Translation and Validation. Technical report, Department of Computer Science and Engineering, Oregon Graduate Institute of Science and Technology, Portland, OR, August 1994.

[595] L. Walton and J. Hook. Creating and Verifying Domain Specific Design Languages. Technical report, Department of Computer Science and Engineering, Oregon Graduate Institute of Science and Technology, Portland, OR, August 1996.

[596] L. Walton and J. Hook. On Understanding a Commonality Analysis. In Proceedings of the OOPSLA'96 Workshop on Domain Analysis: Processes and Results, October 1996.

[597] Thomas Wappler. Remember the Basics: Key Success Factors for Launching and Institutionalizing a Software Product Line. In Proceedings of the First Software Product Line Conference [201], pages 73–84. [598] S. Wartik. The Role of Process Families in Reuse Adoption. In Proceedings of the Sixth Workshop on Institutionalizing Software Reuse, 1993.

[599] S. Wartik and R. Prieto-Diaz. Criteria for Comparing Domain Analysis Approaches. In Proceedings of the Fourth Workshop on Institutionalizing Software Reuse, 1991.

[600] S. Wartik and R. Prieto-Diaz. Criteria for Comparing Reuse-Oriented Domain Analysis Approaches. International Journal of Software Engineering & Knowledge Engineering, 2(3):403–431, 1992.

[601] Diana L. Webber and Hassan Gomaa. Modeling Variability with the Variation Point Model. In Proceedings of the Seventh International Conference on Software Reuse [233], pages 109–122.

[602] Josef Weing¨artner. Product Family Engineering and Testing in the Medical Domain – Validation Aspects. In Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4) [302], pages 369–376.

[603] D. Weiss. Software Synthesis: The FAST Process. In Proceedings of the International Conference on Computing in High Energy Physics (CHEP), September 1995.

[604] D. Weiss. Defining Families: The Commonality Analysis. submitted to TSE, 1997.

[605] David M. Weiss. Commonality Analysis: A Systematic Process for Defining Families. In Second International Workshop on Development and Evolution of Software Architectures for Product Families, February 1998.

[606] David M. Weiss and Chi Tau Robert Lai. Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley, 1999.

[607] S. White and C. Lemus. Architecture Reuse Through a Domain Specific Language Generator. In Proceedings of the Eighth Workshop on Institutionalizing Software Reuse, 1997.

[608] M. Wickman and J. Solderitsch. A Systematic Software Reuse Program based on an Architecture-Centric Domain Analysis. Technical report, Software Technology for Adaptable, Reliable Systems (STARS), 1994.

[609] T. Widen and J. Hook. Software Design Automation: Language Design in the Context of Domain Engineering. In Proceedings of the Tenth Conference on Software Engineering and Knowledge Engineering (SEKE'98), pages 308–317, June 1998.

[610] Jan Gerben Wijnstra. Component Frameworks for a Medical Imaging Product Family. In Third International Workshop on Software Architectures for Product Families [359], pages 4–18.

[611] Jan Gerben Wijnstra. Supporting Diversity with Component Frameworks as Architectural Elements. In Proceedings of the 22 nd International Conference on Software Engineering (ICSE'00), pages 50–59, Limerick, Ireland, 2000.

[612] Jan Gerben Wijnstra. Components, Interfaces and Information Models within a Platform Architecture. In Jan Bosch, editor, Generative and Component-Based Software Engineering, LNCS 2186, pages 25–35, Erfurt, Germany, September 2001. Springer.

[613] J. Withey. Investment Analysis of Software Assets for Product Lines. Technical Report CMU/SEI-96-TR-010, Software Engineering Institute, Carnegie Mellon University, November 1996.

[614] William G. Wood. Government Product Lines. In Proceedings of the First Software Product Line Conference [201], pages 183–192.

[615] Sherif Yacoub, Ali Mili, Chakri Kaveri, and Mark Dehlin. A Hierarchy of COTS Certification Criteria. In Proceedings of the First Software Product Line Conference [201], pages 397–412.

# Anhang B

| **Institut** | Software Engineering Institute (SEI). Ein Institut der Carnegie Mellon University in Pitsburg. | Institut Experimentelles Software Engineering (IESE). Ein Institut der Fraunhofer Gesellschaft. | Gesellschaft für Informatik – Fachgruppe 2.1.6 | European Software Institute (ESI) | Department of Software Engineering and Computer Sience der Universität Karls-krone/Ronneby in Schweden. |
|---|---|---|---|---|---|
| **Forschungs-leiter** | Linda M. Northrop | Klaus Schmid | Horst Lichter | Frank van d. Linden Henk Obbink | Jan Bosch |
| **Forschungs-inhalte** | Domain Engineering<br>Software Reengineering<br>Architekturen | Umfang einer Produktlinie<br>Produktlinien und Domain Analysis<br>Betriebliche Fragen und Produktlinien | Requirements Engineering für die Entwicklung von Produktlinien | Anforderungen an Case Tools zur Unterstützung der System Familien<br>System Familien Entwicklungsprozess, -methoden und –werkzeuge<br>Domainspezifische Plattformen | Entwurf und Verwendung von Software Architekturen<br>Wartung von Software Architekturen |
| **Besonder-heiten** | | | | Benutzen Terminologie der Produkt Familien (System Familien). | Wirkte am ESAPS Projekt der ESI mit. |
| **Vorgehens-modell** | „Architecture Tradeoff Analysis Method“ - kein komplettes Vorgehensmodell. | „PuLSE“ | Befindet sich zur Zeit in Arbeit. | „ESAPS“ Ist ein Projekt, welches 6 Schritte enthält. | |

**Tabelle 2: Forschungseinrichtungen auf dem Gebiet der Produktlinien**

# Literaturverzeichnis


Ackermann, J.; Brinkop, F.; Conrad, S.; Fettke, P.; Frick, A.; Glistau, E.; Jaekel, H.; Kotlar, O.; Loos, P.; Mrech, H.; Ortner, E.; Raape, U.; Overhage, S.; Sahm, S.; Schmitendorf, A.; Teschke, T.; Turowski, K.: Vereinheitlichte Spezifikation von Fachkomponenten. Memorandum des Arbeitskreises 5.10.3 Komponentenorientierte betriebliche Anwendungsentwicklung, 2002.

Balzert, H.: Lehrbuch der Software-Technik – Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung. Heidelberg, Berlin 1998.

Balzert, H.: Lehrbuch der Software-Technik – Software-Entwicklung. Heidelberg, Berlin 2000.

Bandinelli, S.: Reference Architecture in a Product Line Process Context. Publication of the European Software Institute, 1996.

Bandinelli, S.; Rementeria, S.: Software reuse introduction requires a process perspective. Publication of the Software Institute 1996.

Bandinelli, S.; Sagarduy, G.: A Unifying Framework for Reuse Economic Models. Publication of the Software Institute 1996.

Bass, L.; Chastek, G.; Clements, P.; Northrop, L.; Smith, D.; Withey, J.: Second Product Line Paractice Workshop Report. Technical Report of the Software Engeneering Institute, Carnegie Mellon University. Pitsburg 1998.

Bass, L.; Clements, P.; Donohoe, P.; McGregor, J.; Northrop, L.: Forth Product Line Paractice Workshop Report. Technical Report of the Software Engeneering Institute, Carnegie Mellon University. Pitsburg 2000.

Bass, L.; Clements, P.; Sholom, C.; Northrop, L.; Withey, J.: Product Line Practice Workshop Report. Technical Report of the Software Engeneering Institute, Carnegie Mellon University. Pitsburg 1997.

Bayer, J.; Girard, J.-F.; Würthner, M.; DeBaud, J.-M.; Apel, M.: Transitioning Leagacy Assets to a Product Line Architecture. In: Proceedings of the Seventh European Software Engineering Conference (ESEC'99). Toulouse 1999. S. 446 – 463.

Bergey, J.; O'Brien, L.; Smith, D.: Mining Existing Assets for Software Product Lines. Technical Note of the Software Engeneering Institute, Carnegie Mellon University. Pitsburg 2000.

Biethahn, J.; Mucksch, H.; Ruf, W.: Ganzheitliches Informationsmanagement. München, Wien 2000.

Bosch, J.: Design and Use of Software Architectures – Adapting and evolving a product-line approach. Harlow et al. 2000.

Bosch, J.: Product Line Architectures in Industry: A Case Study. Proceedings of the ICSE. Los Angeles 1999.

Clements, P.: Essential Product Line Practices. Paper of the Software Engeneering Institute, Carnegie Mellon University. Pitsburg.

Clements, P.: Report of the Reuse and Product Line Working Group of WISR8. Special Report of the Software Engeneering Institute, Carnegie Mellon University. Pitsburg 1997.

Clements, P.; Cohen, S.; Donohoe, P.; Northrop, L.: Control Channel Toolkit: A Software Product Line Case Study. Technical Report of the Software Engineering Institute, Carnegie Mellon University. Pitsburg 2001.

Clements, P.; Northrop, L.: Software Product Lines. Boston et al. 2001.

Cuka, C.; Weiss, D.: Specifying executable Commands: An example of FAST Domain Engineering. In: IEEE Transaction on Software Engineering, 1997.

DeBaud, J.-M.; Schmid, K.: A Systematic Approach to Derive the Scope of Software Product Lines. In: Proceedings of the Tenth Conference of Software Engineering and Knowledge Engineering 1998 (SEKE'98). S. 128 – 131.

Dittrich, K.: Datenbanksysteme. In: Rechenberg, P.; Pomberger, G. (Hrsg.): Informatik-Handbuch. München, Wien 1999. S. 875ff.

DoD Software Reuse Initiative: Domain Analysis Guidelines. Arlington 1992.

Eisenbarth, T.; Simon, D.: Guiding Feature Asset Mining for Product Line Development. In: Proceedings of the International Workshop on Porduct Line Engeneering: The Early Steps: Planning, Modeling, and Managing (PLEES'01). Erfurt 2001.

Frank, U.: Framework. In: Mertens, P.: Lexikon der Wirtschaftsinformatik. Berlin et al. 1997. S. 167.

Frank, U.; Jung, J.: Prototypische Vorgehensweise für den Entwurf anwendungsnaher Komponenten. In: Proceedings der Verbundtagung VertIS 2001, Bamberg.

Goos, G.; Zimmermann, W.: Programmiersprachen. In Rechenberg, P.; Pomberger, G. (Hrsg.): Informatik-Handbuch. München, Wien 1999.

Halsmans, G.; Pohl, K.: Considering Product Family Assets when Defining Customer Requirements. In: Proceedings of the International Workshop on Porduct Line Engeneering: The Early Steps: Planning, Modeling, and Managing (PLEES'01). Erfurt 2001.

Institut Experimentelles Software Engineering (IESE) des Fraunhofer Institut: Bibliography Definitions. http://www.iese.fhg.de/Pulse/Bibliography/definitions/index.html. Abruf am 2002-06-05.

Institut Experimentelles Software Engineering des Fraunhofer Institutes: Software Product Line Bibliography. http://www.iese.fraunhofer.de/PuLSE/Bibliography/SplBib_a4.pdf. Abruf: 01.06.2002

Kolland, M.; Mehner, T.; Kuhn, K.-J.: Software Plattformen – Mehr als ein Schlagwort. In: Wirtschaftsinformatik 35 (1993) 1, S. 23 – 32.

Kroha, P.: Softwaretechnologie. München et al. 1997.

Maccari, A.: Architectural Evolution of Product Families. In: Proceedings of Product Line Architecture Workshop, 2000.

Marent, C.: Branchenspezifische Referenzmodelle für betriebswirtschaftliche IV-Anwendungsbereiche. In: Wirtschaftsinformatik 37 (1995) 3, S. 303 – 313.

Maymir-Ducharme, F. A.: The Product Line Buisness Model. In: Proceedings of the Eighth Workshop on Institutionlizing Software Reuse, 1997.

McGregor, J. D.: Testing a Software Product Line. Technical Report of the Software Engeneering Institute, Carnegie Mellon University. Pitsburg 2001.

Meffert, H.: Marketing. 9. Aufl., Wiesbaden 2000.

Pons: Wörterbuch für Schule und Studium. Stuttgart et al. 1998.

Rezagholi, M.: Management der Wiederverwendung in der Softwareentwicklung. In: Wirtschaftsinformatik 37 (1997) 3, S. 221 – 230.

Rost, J.: Wiederverwendbare Software. In: Wirtschaftsinformatik 39 (1997) 4, S. 357 – 365.

Sagarduy, G.; Bandinelli, S.; Lerchundi, R.: Product-line Analysis: Do we go ahead? In: Proceedings of the international Workshop on Software Product Lines: Economics, Architecture and Implications 2000.

Scheer, A.-W.: Wirtschaftsinformatik. Berlin et al. 1998.

Schiffer, S.; Templ, J.: Internetdienste. In: Rechenberg, P.; Pomberger, G. (Hrsg.): Informatik-Handbuch. München, Wien 1999. S. 999ff.

Schlick, M.; Hein, A.: Knowledge Engineering in Software Product Lines.

Schmid, K.: Multi-Staged Scoping for Software Product Lines. In: Proceedings of Software Product Lines: Economic, Architectures, and Implications. Limerick 2000.

Staehle, W. H.: Management. München et al. 1999.

Succi, G.; Yip, J.; Liu, E.: Analysis of the Essential Requirements for a Domain Analysis Tool. In: Proceedings of the international Workshop on Software Product Lines: Economics, Architecture and Implications 2000.

Ulrich, F.: Framework. In: Mertens, P.(Hrsg.): Lexikon der Wirtschaftsinformatik. Berlin et al. 1997, S. 167-168.

Walter, W. : Analysis 1. Berlin 2001.

Weiss, D.: Software Synthesis: The FAST Process. In: Proceedings of the International Conference on Computing in High Energy Phisics (CHEP), 1995.

Weis, D. M. ; Lai, R.: Software Product-Line Engneering. Reading et al. 1999.

Withey, J.: Investment Analysis of Software Assets for Product Lines. Technical Report of the Software Engeneering Institute, Carnegie Mellon University. Pitsburg 1996.

Wöhe, G.: Einführung in die Allgemeine Betriebswirtschaftslehre. München 2000.

# Glossar

| | |
|---|---|
| Artifact | Alle Produkte, die während des Softwareentwicklungszyklus generiert werden. Ist ein Objekt der Wiederverwendung. |
| Asset | Ein Ergebnis der Komponentenentwicklung oder der Domain Analysis. |
| Asset Base | Eine Sammlung von Artifacten und eine entsprechende Zugriffsstruktur. Sie wird während der Domain Implementierung erzeugt und bei der Anwendungsentwicklung genutzt. Sie kann auch Produkte der Domain Analysis beinhalten. |
| Domain | Stellt ein Wissensgebiet oder ein Betätigungsfeld dar. Ist eine realweltliche und damit unscharfe Menge. |
| Domainmodel | Ist die Beschreibung einer Domain mit Hilfe von scharfen Mengen. Da durch eine derartige Beschreibung Informationen verloren gehen, spricht man von einem Modell. |
| Ergebnis (bzgl. Produktlinien) | Beschreibung einer Lösung oder von Wissen, als Output des Softwareentwicklungsprozesses. |
| Objekt der Wiederverwendung | Ist jedes Produkt eines Prozesses, welches außerhalb des Prozesses wiederholt genutzt wird, um eine Wiederholung der Aktivitäten zur Erreichung dieses Produktes zu vermeiden. Produkte, welche (lediglich) auf eine solche Nutzung hin entwickelt wurden, sind eingeschlossen. |
| Opportunistische Wiederverwendung | Das Vorgehensmodell spezifiziert Methoden, welche das Identifizieren wieder zu verwendender Komponenten ermöglichen. Außerdem wird ein Vorgehen zum Auffinden der Komponenten vorgeschlagen. Es unterbleibt jedoch eine Aussage, wie und unter welchen Umständen Vorteile aus einem derartigen Vorgehen gezogen werden können. |
| Produkt der Wiederverwendung | Ist ein Ergebnis eines Prozesses, welches unter der Einbeziehung von Ergebnissen eines anderen Prozesses entsteht. Die einbezogenen Ergebnisse müssen dabei Objekte der Wiederverwendung sein. |