# Integration von Feature Modellen in die evolutionäre Weiterentwicklung von Software-Produktlinien-Architekturen

***Ilka Philippow, Matthias Riebisch, Ilian Pashov***
*Technische Universität Ilmenau, Max-Plank-Ring 14, Postfach 100565, 98684 Ilmenau*
*{Ilka.Philippow|Matthias.Riebisch|Ilian.Pashov}@tu-ilmenau.de*

*Software-Produktlinien erfordern stabile Systemarchitekturen, die über mehrere Jahre an sich Veränderungen der Domäne angepasst werden können. Meist führen solche Anpassungen in einem zunehmenden Qualitätsverslust der Systemarchitektur der Produktlinie, bis diese ohne ein umfassendes Refactoring nicht mehr weiter verändert werden kann. Dies ist mit einem großen Risiko und hohen Kosten verbunden. In diesem Beitrag wird die Idee verfolgt, durch kleine Refactoring-Schritte eine kontinuierliche Anpassung der Produktlinien-Architektur an sich ändernde Anforderungen der Domäne vorzunehmen, wobei die Risiken verringert, ein evolutionärer Prozess erreicht und die Periode bis zur Notwendigkeit eines umfassenden Refactoring signifikant erhöht werden soll. Diese evolutionären Veränderung der Produktlinienarchitektur wird durch parallele Entwicklungen von konkreten Kundenapplikationen auf der Basis der betrachteten Produktlinie initiiert. Die Ziele der einzelnen Refactoring-Schritte werden dabei unter Nutzung von Featuremodellen bestimmt.*

## 1 Einführung

Eine Produktlinie [Kotler et al.1999] fasst eine Menge von verwandten Produkten einer Domäne. Die Produkte werden der Basis einer Produktlinie-Referenzarchitektur abgeleitet. Diese Referenzarchitektur enthält „core assets“ [Clements 1999], invariante Elemente, die in allen Produkten enthalten sind. Sie bietet die Basis für variable Elemente, die für die Ableitung eines spezifischen Produktlinienmitglieds (Kundenprodukt, Kundenanwendung) eingesetzt werden. Software-Produktlinien müssen wegen des Aufwands ihrer Entwicklung langfristig die Ableitung von Softwareprodukten ermöglichen. Diese Forderung hat für die Referenzarchitektur besondere Bedeutung. Die Entwicklung einer Produktlinie und die Entwicklung einer darauf basierenden Applikation können als zwei parallele Prozesse betrachtet werden, die in enger Beziehung zueinander stehen.

Auf Abb. 1 sind die Beziehungen zwischen der Produktlinien-Entwicklung und der Produkt-Entwicklung in Anlehnung an [ESAPS] dargestellt. Die oberen drei Gruppen von Aktivitäten dienen der Produktlinien-Entwicklung. Im Ergebnis der Domänenanalyse

entsteht das Produktlinienkonzept, welches als Featuremodell dargestellt werden kann und die Variabilität dokumentiert. Das Produktlinienkonzept ist Ausgangspunkt für den Entwurf der Produktlinien-Referenzarchitektur. Parallel zur Architektur muss ein Konzept für die Komposition bzw. Generation von Produkten entwickelt werden. Letztlich müssen die in der Architektur definierten Elemente implementiert und getestet werden. Für die Produkt-Entwicklung stehen zur Verfügung: das Feature-Modell für die Spezifikation einer Kundenanwendung, die Referenzarchitektur und die Komponenten für die Ableitung des Produktes. Die Produkt-Entwicklung im unteren Teil der Abbildung beginnt mit der Analyse der Kundenanforderungen unter Einbeziehung des Featuremodells. Genügt das darin festgehaltene Produktlinienkonzept den Kundenanforderungen, erfolgt die Produktableitung.

Die Produkt-Anforderungsanalyse kann neue Anforderungen aufwerfen, die nicht im Produktlinienkonzept vorgesehen sind. Diese sind daraufhin zu überprüfen, ob sie projektspezifisch nur für das konkrete Produkt relevant sind oder ob sie den verändernden Anforderungen der Produktliniendomäne zuzuordnen sind und in das Produktlinienkonzept aufgenommen werden müssen. Im Rahmen einer evolutionären Produktlinien-Entwicklung ist damit eine Modifikation des Produktlinienkonzeptes, d.h. sowohl des Featuremodells als auch der Referenzarchitektur und der Komponentenbasis verbunden.

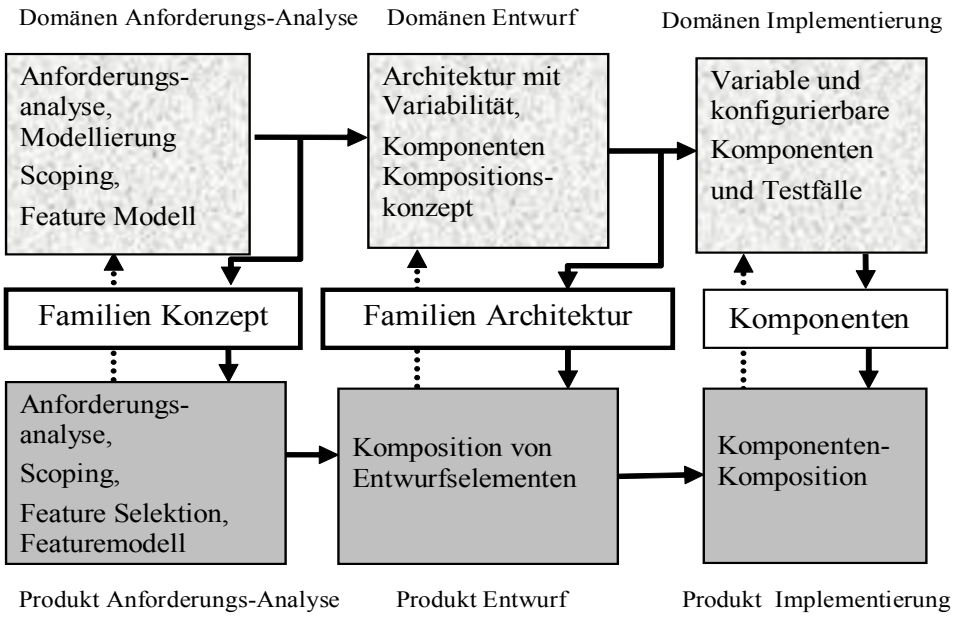


Abb. 1: Beziehungen zwischen Produktlinien- und Produktentwicklung

Mit der Zeit führen auf der einen Seite Veränderungen in der Produktdomäne wie z.B. neue funktionale Anforderungen, sich verändernde Laufzeitumgebungen und Plattformen, steigende Performance Ansprüche, rechtliche Bedingungen usw., und andererseits die ständigen Modifikationen der Referenzarchitektur dazu, dass diese nicht mehr in der Lage ist, den Ansprüchen der Produktdomäne zu folgen, ohne ein umfassendes und risikobehaftetes Reengineering und Refactoring. Diesem für Softwaresysteme bekannten Problem soll durch Ansätze wie z.B. Extreme Programming (XP) (Beck 1999) begegnet werden, in denen das Refactoring bereits in dem Entwicklungsprozess integriert ist. Diese bekannten Ansätze berücksichtigen jedoch nicht die Behandlung von Variabilität, wie sie in Produktlinien enthalten ist.

In diesem Beitrag wird ein Ansatz vorgestellt, bei dem durch Einführung von kleinen Refactoring-Schritten die Referenzarchitektur kontinuierlich an neue Anforderungen der Domäne angepasst wird. Das Ziel ist die Erhöhung des zeitlichen Abstandes zwischen zwei umfassenden, risikobehafteten Refactorings durch evolutionäre Weiterentwicklung (Abb. 2: Version 3 gegenüber 1 und 2). Der vorgestellte Ansatz wurde im Rahmen eines größeren industriellen Projektes zum Reengineering eines Bildverarbeitungssystems in Kooperation mit einem deutschen Großunternehmen entwickelt und evaluiert.

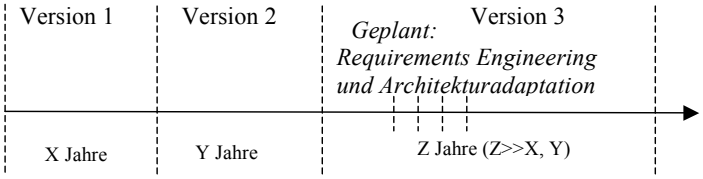


Abb. 2: Refactoring-Zeitrahmen

Ursachen für das Scheitern von Softwareprojekten liegen häufig in Fehlern beim Requirements Engineering [Ibanez et al. 1996], [Standish 1995]. Der Erfolg von Produktlinien wird durch das Management der Ergebnisse des Requirements Engineering, der Domäneninformationen und der produktlinienspezifischen Konzepte, maßgeblich bestimmt. Fehler im Requirements Engineering pflanzen sich in alle abgeleiteten Produktvarianten fort. Für das Management des Produktlinienkonzepts mit seinen vorgesehenen gemeinsamen und variablen Teilen werden Featuremodelle eingesetzt. Beim Refactoring werden Featuremodelle und ihre Korrespondenzen außerdem für das Auffinden von Redundanzen auf einem hohen Abstraktionsniveau verwendet. Dieser Ansatz basiert auf dem Fakt, das Features durch domänen-zugehörige Wörter beschrieben werden und somit die Anwendung herkömmlicher Information-Retrieval-Verfahren und -Werkzeuge möglich ist. Im Abschnitt 2 werden wesentliche Aspekte des Requirements Engineering und der Feature Modellierung für Produktlinien zusammengefasst. Für den hier verfolgten Ansatz zur ständigen Anpassung der Referenzarchitektur an den Domänenzustand spielt das Featuremodell eine wesentliche Rolle. Im Abschnitt 3 wird die Bedeutung von Featuremodellen im Reengineering-Prozess verdeutlicht. Im Abschnitt 4 wird dargestellt, wie

ausgehend von der Anforderungserfassung für ein Produkt das Featuremodell konsistent an die Domäne angepasst wird und einen Ausgangspunkt für eine Architekturadaptation darstellt. Das Featuremodell wird zur Unterstützung der Entscheidungsfindung im Refactoring hinzugezogen.

## 2 Requirements Engineering und Feature Modellierung für Produktlinien

Der Vorgang des Requirements Engineering ist in der Literatur mehrfach beschrieben worden, u.a. in [Hofmann 2000], [Sommerville et al. 1997] und [Loucopoulos 1995]. Man unterteilt dabei in drei Phasen. In der Auswahlphase müssen sich die Projektbeteiligten zunächst mit früheren und erwarteten Produkten, ihren Anforderungen und Problemen in der betrachteten Domäne vertraut machen. Das erfolgt durch Analyse früherer Produkte und Dokumente unter Einbeziehung von Interviews und Marktanalysen. Im Ergebnis dieser Phase wird die Entscheidung über die Entwicklung einer Produktlinie getroffen, die als Scoping bezeichnet wird. Im Gegensatz zu konventioneller Entwicklung wird hier sowohl über Produkte als auch über die Strategie der Produktlinie entschieden. In der nachfolgenden Phase wird die Domäneninformation strukturiert und modelliert. Das Produktlinienkonzept mit seinen gemeinsamen und variablen Teilen wird in einem Anforderungsmodell hinterlegt. In der letzten Phase erfolgt eine Validierung des Anforderungsmodells. Die Anforderungen werden in Kooperation der unterschiedlichen Projektbeteiligten diskutiert und abgestimmt. Die drei Phasen werden in der Regel in mehreren Iterationen durchlaufen.

Für Variabilitätsmodellierung in Produktlinien auf der Basis von Anforderungen haben sich Featuremodelle als geeignet erwiesen. Ursprünglich wurde die Featuremodellierung durch die FODA-Methode [Kang et al.1990] für die Strukturierung von Domäneneigenschaften aus Kundensicht eingeführt. Die Repräsentation von Features (Kernmerkmale), die in jedem Produkt enthalten sein müssen, werden mit einem Vollkreis markiert (siehe Abb. 3), optionale Features mit einem Lehrkreis und alternativ zur Wahl stehende Features (1 aus n) werden mit einem Bogen umspannt. In [Carnecki et al 2000] wurde das Featuremodell durch Featurebeziehungen (exclude Feature, require Feature, siehe Kennzeichnung [1] in Abb. 3) erweitert. In [Riebisch et al. 2002] ist eine Erweiterung alternativer Features mit der Angabe von Vielfachheiten (siehe Kennzeichnung [2] in Abb. 3) zur Auswahl vorgeschlagen. Featuremodelle bieten ein gutes Kommunikationsmittel für alle Projektbeteiligten, mit dessen Hilfe Kunden ihre spezifischen Produkte konkretisieren können. Dazu müssen Featuremodelle in maschinell auswertbarer Form vorliegen. Für ihre eindeutige Formulierung wurde eine Definition entwickelt, die diese Anforderungen erfüllt [Riebisch 2003]. Beziehungen zwischen Features werden mit Mitteln der Object Constraint Language OCL, einem Bestandteil der UML [OCL 2003] beschrieben. Darüber hinaus bieten Featuremodelle auch einen Ansatzpunkt für die Entwicklung einer

Referenzarchitektur und die automatisierte Ableitung von Kundenprodukten, wie in [Böllert 2002] gezeigt. Sie spielen hierfür eine zentrale Rolle bei der Koordination zwischen den beiden in Abb. 1 dargestellten Entwicklungsprozessen [Riebisch 2003].

Featuremodelle sind auf die hierarchische Strukturierung von Anforderungen einer Produktdomäne ausgerichtet (Abb. 3). Darüber hinaus können sie als Brückenglied zwischen Kundenanforderungen und Use Cases als Startpunkt für eine Softwareentwicklung betrachtet werden.

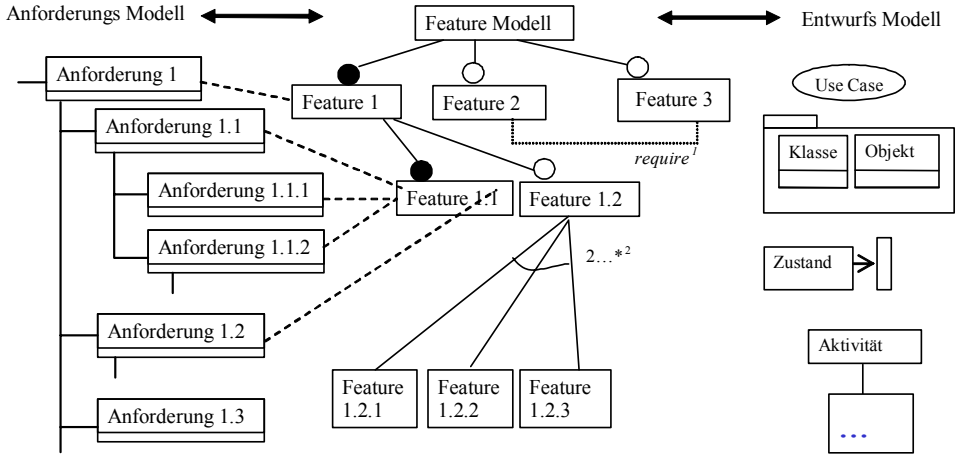


Abb. 3: Featuremodelle als Brückenglied zwischen Anforderungen und Entwurfsmodellen

Im Unterschied zur Entwicklung von Software für Einzelsysteme ist die evolutionäre Entwicklung und Wartung von Produktlinien darüber hinaus dadurch gekennzeichnet, dass

- frühere Produkt integriert und dazu ihre Architekturen erkannt, verstanden und beschrieben werden müssen (Reverse Engineering, Architecture Recovery),
- alte und neue Anforderungen miteinander verglichen werden müssen, neue Anforderungen erkannt und in die Produktlinie integriert werden müssen,
- die Integration neuer Anforderungen eine Modifikation des Architekturentwurf beinhaltet, wobei Anforderungen großen Umfangs zu umfassenden Reengineering- und Refactoring-Aktivitäten führen.
- neue Kern- und variable Komponenten implementiert werden müssen.

Für die Unterstützung einer evolutionären Weiterentwicklung ist die Verfolgbarkeit von Anforderungen über Features zu Entwurfs- und Implementierungselementen entsprechen der Abb. 3 in beiden Richtungen notwendig. Jedes Feature hat eine oder mehrere

zugeordnete Anforderungen. Diese Zuordnung wird über sog. Traceability Links [Sametinger et al. 2002] realisiert, die Verknüpfungen zwischen Features und Anforderungen sowie Komponenten herstellen. Zu jedem Feature gehört zumindest ein Use Case Diagramm und ein Klassendiagramm. Use Case Szenarien können durch weitere Aktivitäts- bzw. Zustands-Diagramme näher spezifiziert werden. Für den in diesem Beitrag verfolgten Ansatz, Featuremodelle zur Unterstützung von Refactoring zu verwenden, wird für jedes Feature eine korrespondierende Architekturkomponente erwartet und umgekehrt (Abb. 4). Die Traceability-Links sind hier durch „ImplementedBy" gekennzeichnet. Die Stellung eines Features in der Featurehierarchie repräsentiert den Einfluss eines Features auf die Referenzarchitektur.

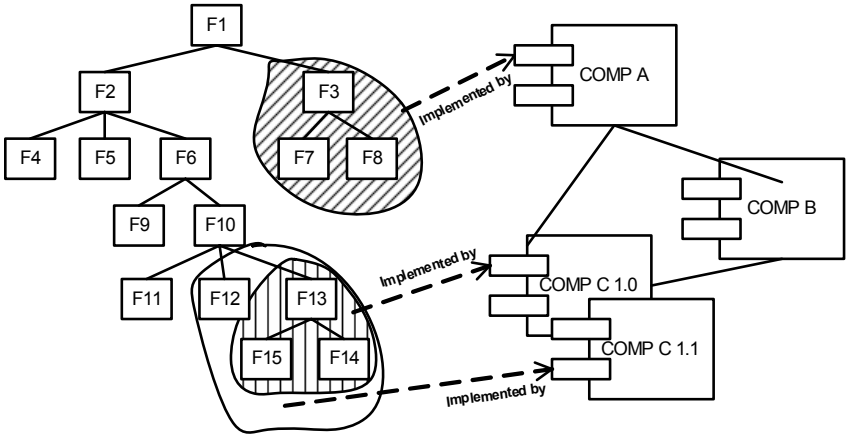


Abb. 4: Korrespondenzen zwischen Features, Komponenten und Komponentenversionen

# 3 Anwendung von Featuremodellen für das Recovery von Softwarearchitekturen

Verfahren zum Recovery (Wiedererkennen und Darstellen) von Softwarearchitekturen im Zusammenhang von Produktlinien sind in zwei Situationen relevant:

- Ausgangspunkt einer Produktlinienentwicklung sind ähnliche, bereits existierende Produkte, deren Leistungsumfang und Architektur zu analysieren sind.
- Architekturen bereits über längere Zeiträume existierender und Produktlinien sind zu überarbeiten, um dem Zustand der Veränderungen und Anforderungen in der Produktdomäne noch entsprechen zu können.

In den meisten Fällen stehen dabei die Entwickler vor dem Problem, das die Softwaredokumentation für das Verständnis der Legacy (Alt)-Architekturen unzureichend ist und

das kein Kommunikation mit den ehemaligen Systementwicklern mehr möglich ist. Existierende Ansätze zum Recovery gehen davon aus, die benötigten Informationen aus dem Quellcode zu extrahieren. In dem hier vorgestellten Ansatz wird vorgeschlagen, darüber hinaus das Wissen über die Problemdomäne zum Programmverständnis zu nutzen. Das ursprüngliche Domänenwissen liegt entweder bereits strukturiert in einem Featuremodell vor, welches mit Hilfe von Domänenexperten zu aktualisieren ist. Anderenfalls muss es aus dem Wissen der Domänenexperten und den Ergebnissen der Quellcodeanalyse entwickelt werden. Ziel ist es, das Recovery von Architekturen auf dem Niveau des Domänenwissens durch Featuremodelle zu unterstützen. Für das Recovery einer Softwarearchitektur sind zwei Aspekte zu analysieren:

- welche funktionalen Entwurfsziele liegen der Architektur zugrunde,
- welche Entwurfsentscheidungen wurden daraufhin getroffen.

Damit gibt es zwei Typen von Features, die einen funktionalen Problemraum bzw. einen entwurfsorientierten Lösungsraum beschreiben und in zwei korrespondierenden Featuremodellen dargestellt werden können. Die Featuremodellierung wird in den üblichen Recovery-Prozess integriert, der dann in vier Hauptphasen unterteilt werden kann (Abb. 5).

*Requirements Engineering und Domänenanalyse:*

Funktionale Anforderungen repräsentieren funktionale Entwurfsziele. Diese werden zusammen mit den Domänenexperten erarbeitet und in einem entwurfszielorientierten (funktionalen) Featuremodell dargestellt. Die Features werden durch Use Case Beschreibungen und Use Case Szenarien untersetzt. Die Szenarien werden im späteren Verlauf des Architektur-Recovery für die Hypothesenvalidierung benötigt.

*Analyse der Legacy-Architektur:*

Für die Architekturanalyse gibt es verschiedene Ansätze, die in den meisten Fällen auf dem Quellkode aufsetzen. In [Brooks et al. 1983] wird erstmalig eine Verbindung zum Domänenwissen hergestellt. Diese Idee wurde durch [Letowsky et al. 1986] weiterentwickelt. Der top-down Ansatz in [Rajlich et al. 1994] basiert auf der Aufstellung von Hypothesen, die durch den Code zu verifizieren sind. Diese Idee wird auch in dem hier vorgeschlagen Vorgehen aufgenommen. In [Rugaber 2000] werden verschiedene Möglichkeiten der Präsentation von Domänenwissen diskutiert. Der Ansatz hier orientiert auf die Nutzung der Brückenfunktion von Featuremodellen. Für die Darstellung der Analyseergebnisse unter Berücksichtigung der Domäneninformationen eignet sich das System Rigi [Storey et al. 1997], welches die Möglichkeit bietet, Softwarestrukturen durch einen Grapheditor in mehreren individuellen Fenstern darzustellen (hierarchical nested graphs). Es ist eine offene Plattform, die für die Feature orientierte Arbeitsweise erweiterbar ist. In [Riva et al. 2002] wird diese Idee aufgegriffen. Bisher jedoch gibt es jedoch noch keine praktikabel verwertbaren Werkzeuge. Für den hier vorgestellten Ansatz wurde eine Kombination aus verschiedenen Werkzeugen verwendet. Eine rela-

tionale Datenbasis wurde für die Featureverwaltung und Crossreferenz-Tabellen verwendet. Die Visualisierung und Bearbeitung erfolgte mit XML Editoren.

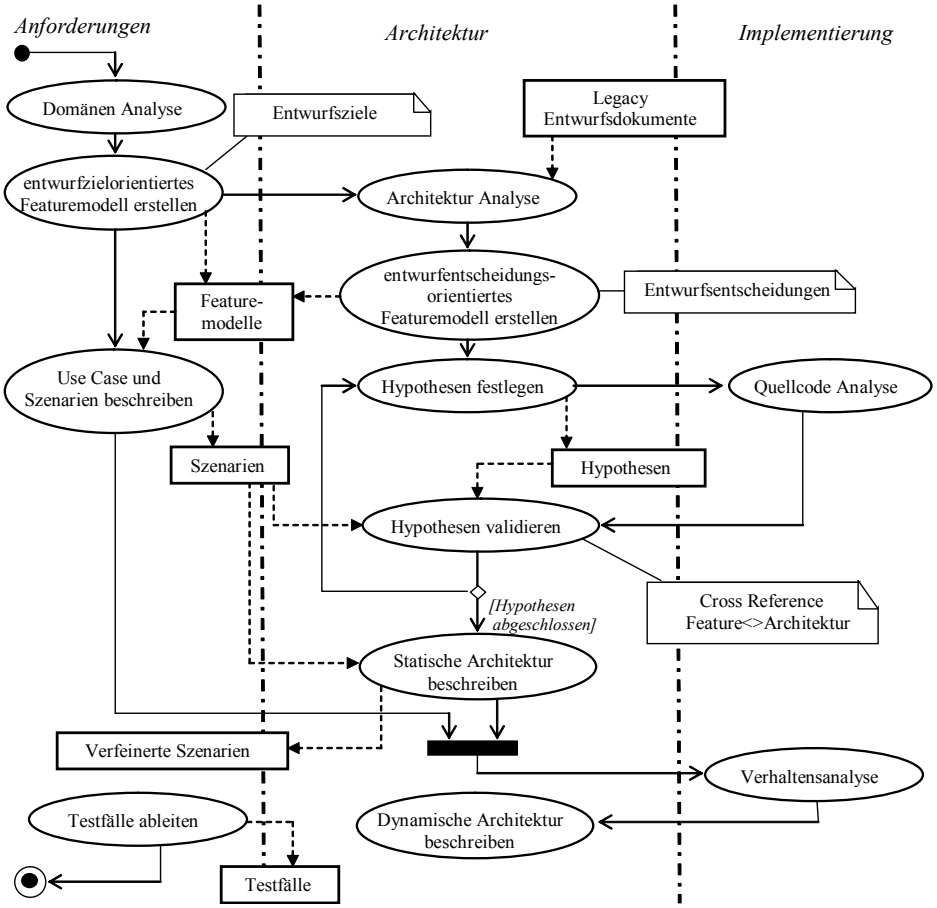


Abb. 5: Vorgehen für das Recovery von Softwarearchitekturen

Grundlagen für die Analyse sind der existierende Quellcode und weitere Entwurfsdokumente. Aus dem Vergleich mit den Anforderungen aus der davor liegenden Phase und der traditionell zu analysierenden Architektur erfolgt ein Nachvollziehen von Entwurfsentscheidungen. Die Ergebnisse werden in einem entwurfsentscheidungsorientierten Featuremodell strukturiert, weitgehend korrespondierend zu dem zielorientierten Featuremodell dargestellt und im späteren Verlauf für die Hypothesenaufstellung verwendet. Die Knoten des entscheidungsorientierten Featuremodells enthalten Lösungen (Strukturen, Pattern o.a.).

*Statische Architekturbeschreibung durch Hypothesenaufstellung und -validierung*

Die beiden Featuremodelle und die Use Case Beschreibungen dienen als Ausgangsbasis für die Aufstellung von Hypothesen betreffs der vorliegenden Architektur. Eine Hypothese beschreibt die Annahme der Existenz einer Beziehung zwischen einem Feature und einem Architekturelement (z.B. Klasse, Komponente, Interface, Methode u.a.). Die Hypothesen werden in Cross-Referenz-Tabellen „Feature zu Architekturelement“ und „Feature zu Quellcodeelement“ zusammengestellt. Dabei wird der Quellkode in konventioneller Weise analysiert. Die Hypothesen werden durch Architekturdiagramme untersetzt. Die Behandlung einer Hypothese kann zu weiteren Hypothesen führen. Eine Hypothese kann ungültig werden, wenn eine zunächst vermutete Beziehung zwischen Features, Architektur- und Quellcodeelementen fehlt. Features ohne ein korrespondierendes Architekturelement können optional, invalid oder obsolet sein. Wenn keiner dieser drei Fälle zutrifft ist offensichtlich eine fehlende Hypothese gefunden. Alle positiv validierten Hypothesen verbleiben in den Cross-Referenz-Tabellen und bilden die Grundlage für die statische Architekturbeschreibung.

Für die statische Architekturbeschreibung gibt es zwei bekannte und erprobte Ansätze, das *4 view* model [Hofmeister et al. 2000]: und das *4+1 view* model [Kruchten et al. 1995]. Obwohl das 4+1view model sehr populär in seiner Anwendung geworden ist soll darauf hingewiesen werden, das das 4 view model für die Arbeit mit „gemischten“ Systemen, basierend sowohl auf objektorientierten als auch nicht objektorientierten Teilsystemen, besser geeignet erscheint, wie es sich aus den Erfahrungen in dem Beispielprojekt gezeigt hat.

*Dynamische Architekturbeschreibung unter Verwendung der Use Case Szenarien*

In dieser Phase können die Use Case Szenarien für die Analyse des dynamischen Verhaltens als Ausgangspunkt eingesetzt werden. Aus den Verhaltensinformationen ist es möglich, Testfälle abzuleiten.

Der hier vorgestellte Vorgehensansatz wurde bisher nur bis zur Modellierung der statischen Architektur an einem größeren Projekt erprobt. Eine detaillierte Beschreibung dieses Vorgehens ist in [Pashov 2003] zu finden. Die dynamische Architekturbeschreibung ist Gegenstand weiterer Aktivitäten. Dazu ist es u.a. notwendig, das Feature Modell zu diesem Zeck weiterzuentwickeln und die Möglichkeiten einer Sicht auf die Architekturausführung zu schaffen.

# 4 Feature Modelle für evolutionäre Wartung und Small-Step-Refactoring von Produktlinien

Für die Wartung von Softwaresystemen gibt es zahlreiche Ansätze wie [Bass et al. 1998], [Bosch 2000], [Carriere et al. 1999]. Die bekannten Ansätze gehen jedoch kaum auf die Besonderheiten von Produktlinien ein. In [Johansson et al. 2002] wird das Problem des Verfalls von Produktlinienarchitekturen diskutiert und eine Metrik für die Erfas-

sung des Verfalls vorgeschlagen, jedoch keine Vorschläge, wie man dem Verfall begegnen kann.

Die Grundidee in diesem Beitrag besteht in der Einführung kleiner Refactoring Schritte für einen möglichst langen Erhalt der Produktlinienarchitektur in Kombination mit einer ständigen Anpassung an die sich ändernden Anforderungen einer Produktliniendomäne - eine evolutionären Wartung. Initiiert kann eine solche Wartung entweder durch geplante Termine, zu denen ein darauf ausgerichtetes Requirements Engineering zur Aktualisierung der Domänenanforderungen und eine entsprechende Adaptation der Referenzarchitektur der Produktlinie ausgeführt werden. Es erscheint jedoch als effektiv, parallel zur Entwicklung von Kundenprodukten die Ergebnisse der dabei ausgeführten Produktanforderungsanalyse (s. Abb.1) für eine Wartung der Produktlinie selbst zu nutzen.

Voraussetzungen für das in diesem Beitrag vorgeschlagene Vorgehen sind

- das Featuremodell der Produktfamilie, wobei die Position eines Features in der Hierarchie der Bedeutung eines Features für die Produktlinie entsprechen soll,
- eine definierte, verfolgbare Zuordnung zwischen Anforderungen, Features und Architekturelementen (vgl. Kap. 2) und
- das aus der Produktentwicklung resultierende Produktfeaturemodell.

Die evolutionäre Wartung beginnt unter Einbeziehung des Produktfeaturemodells mit einem Requirements Engineering, darauf fokussiert, die aktuellen Domänenanforderungen und Veränderungen zu erfassen und die Bedeutung eventueller Anforderungen für die Produktlinie zu evaluieren. Dafür ist die Kooperation von Domänenexperten (diese Rolle kann durch einen Produktmanager übernommen werden.), Produktlinienarchitekten und Komponentenentwickler notwendig. Featuremodelle sind für diese Projektbeteiligten nicht nur das gemeinsame Kommunikationsmittel. Das Featuremodell der Produktlinie wird auch im Prozess der Weiterentwicklung durch alle Beteiligten beeinflusst, aktualisiert und angepasst. Das Aktivitätsdiagramm in Abb. 6 zeigt in Anlehnung an [Riebisch 2003] die Kooperation der Projektbeteiligten.

Ausgangspunkt sind Anwendungsentwicklungen (Ableitung neuer Produktlinienmitglieder) auf der Basis der Produktlinie. Dabei wird in der Phase der Anforderungserfassung ein Featuremodell für das Zielprodukt erstellt. Aus diesem (oder auch mehreren) Produktprojekten werden die Anforderungen an ein paralleles oder zeitlich auch versetztes Projekt zur Produktlinienwartung und -weiterentwicklung übergeben.

Die Anforderungen, die durch die existierende Architektur nicht erfüllt werden, sind als neue, zunächst anhängige Feature zu erfassen. Diese anhängigen Feature sind dahingehend zu untersuchen, ob sie nur für das betrachtete Produktprojekt oder auch für die Produktfamilie für eine Anpassung an Domänenänderungen relevant sind und überhaupt in das Produktlinienkonzept integriert werden können (Kooperation: Produktmanager, Systemarchitekten). Die ausgewählten neuen Features werden in die Featurehierarchie eingeordnet, parallel dazu wird Referenzarchitektur angepasst. Diese Informationen werden den Entwicklern zu einer Aufwandsbestimmung übergeben und nach nochmali-

ger Prüfung letztlich implementiert. Dadurch erfolgt die Anpassung der Produktlinie an Änderungen der Produktdomäne. Während der nachfolgenden Implementierung kann es durch die Entwickler zu architekturbedingten Entscheidungen kommen, die durch Rückkopplung in einem Abgleich Modelle und Implementierung festzuhalten sind.

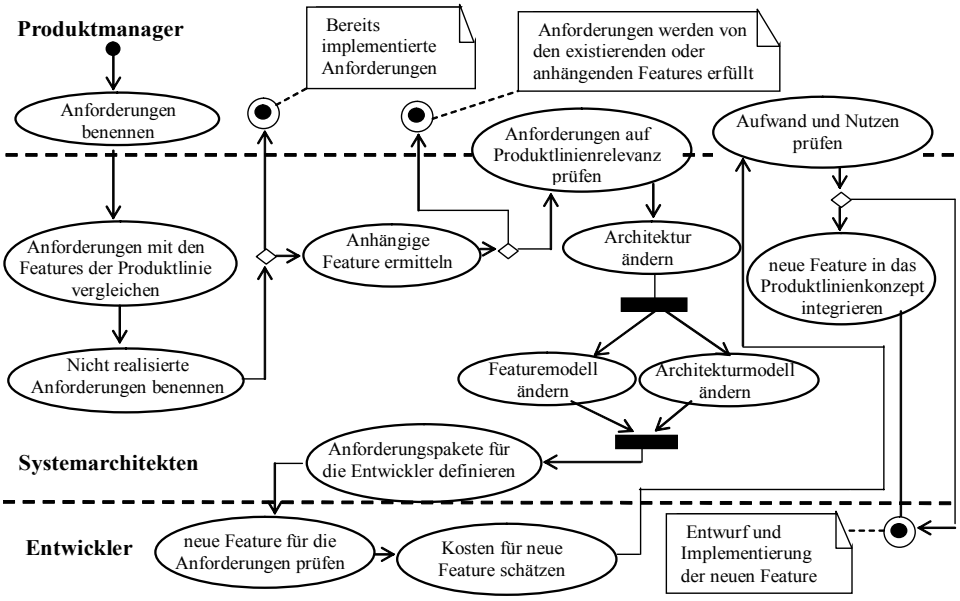


Abb. 6: Aktivitätsdiagramm des Requirements Engineering für die evolutionäre Wartung

Die Integration neuer Feature führt zu einer Veränderung der Featurehierarchie. Je höher ein Feature in der Baumstruktur angesiedelt ist desto höher sind Aufwand und Kosten für die Implementierung. Die Implementierung soll mit kleinen Refactoring-Aktivitäten verbunden werden, um die Wartbarkeit und Erweiterbarkeit der Produktlinienarchitektur über möglichst lange Zeit zu erhalten. Eine Voraussetzung dafür ist, wie in Kap. 2 erwähnt, eine vorliegende vollständige Beschreibung der Korrespondenzen, im Besonderen zwischen Features und Komponenten. Abb. 7 zeigt die in diesem Beitrag vorgeschlagenen Refactoring-Aktivitäten. Wenn Voraussetzungen (Featuremodell, Korrespondenzen) fehlen, ist zunächst ein Architektur-Recovery, wie beispielsweise in Abschnitt 3 beschrieben, durchzuführen.

Ausgehend von dem Featuremodell und den korrespondierenden Komponenten ist es möglich, die Architektur auf Disproportionen und Redundanzen zu untersuchen.

Die Suche nach Disproportionen orientiert auf das Aufdecken von monolithischen Komponenten, die durch ihre Komplexität die Wartbarkeit und Konfigurierbarkeit reduzieren können. Auf den ersten Blick sind solche Monolithen in einer Architektur recht einfach zu erkennen, wenn es sich um kleine Systeme handelt. Bei umfangreichen Systemen fehlen häufig geeignete Darstellungen, die solche Übersichten erlauben. In beiden Fällen fehlen den Entwicklern jedoch Richtlinien und Argumente, ab wann ein Aufbrechen der Monolithen durch Umstrukturierung erfolgen soll. Hier wird vorgeschlagen, die Featureanzahl pro Komponente als Metrik zu verwenden. Dazu sind zwei Schritte notwendig:

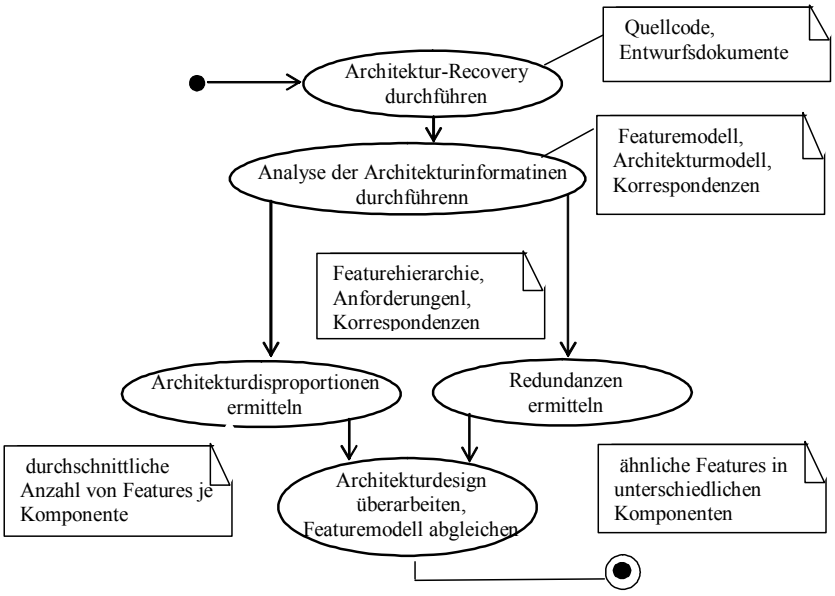


Abb. 7: Vorgehen bei Refactoring-Aktivitäten

- den Korrespondenzen folgend wird die Anzahl der Features pro Komponente ermittelt
- die mittlere Anzahl von Features für alle Komponenten wird berechnet.

Durch eine nachfolgende Umstrukturierung kann ein Ausbalancieren der Featureverteilung auf Komponenten erreicht werden. Dabei ist darauf zu achten, das durch Abgleich eine Übereinstimmung mit der Featurehierarchie des Modells und die korrekte Erfassung der Korrespondenzen erhalten bleibt. Ab welcher absoluten Featureanzahl eine Komponente als Monolith betrachtet wird hängt projektspezifisch von der immanenten Systemkomplexität ab. Für das durchgeführte Projekt Bildverarbeitung ergab eine Analyse für eine ausgewählte Anzahl von Komponenten die in Abb. 8 gezeigte Featureverteilung. Mit einem für das Projekt gültigen Toleranzbereich von 20% wurden die ersten beiden Features als Monolithen erkannt und zur Umstrukturierung vorgeschlagen.

| Komponente | Featurezahl | |
|---|---|---|
| Image Manager IM | *52* | |
| Flow Observer FO | *21* | |
| Visualisation Desk VD | 18 | |
| User Interface Interceptor UII | 18 | |
| Dispatcher DISP | 17 | |
| Optical Character Regognizer OCR | 16 | |
| External Information Management | 12 | |
| Image Injector II | 10 | |
| Image Souce Interface ISIF | 9 | |
| Image Consumer Interface ICIF | 9 | |
| Image database IDB | 5 | |
| Image reciever IR | 4 | |
| Test Desk TD | 3 | ca. 16 |

Featuredurchschnitt 16 und Toleranzbereich von +20%

Abb. 8: Auszug der Featureverteilung aus einem Projekt zur Bildverarbeitung

Beim Refactoring besteht darüber hinaus das Ziel, Redundanzen zu entfernen und die Modularisierung der Lösung zu homogenisieren. Featuremodelle und ihre Korrespondenzen werden dazu für das Auffinden von Redundanzen auf einem hohen Abstraktionsniveau verwendet. Herkömmliche Information-Retrieval-Verfahren und -Werkzeuge können dazu verwendet werden, die in den Featuremodellen verwendeten domänenzugehörigen Begriffe auszuwerten. Das Verfahren erfolgt in zwei Schritten:

- Durchführung einer Wortfrequenzanalyse zum Auffinden der häufigsten Worte unter Angabe einer Trefferquote (Häufigkeit in Prozent)
- Durchführung einer Wortkategorisierung (Baum von Kategorien und Subkategorien)

Den Auszug eines Teilergebnisses aus dem durchgeführten Projekt zeigt Abb. 9.

| **Wort** | **Häufigkeit** |
|---|---|
| Image | 20 (21,74%9 |
| Processing | 17 (18,48%) |
| statistics | 6 (6,522%) |
| device | 6 (6,522%) |
| control | 6 (6,522 %) |
| result | 5 (5,435%) |
| resource | 4 (4,348%) |

Kategorie: Image (20)
Unterkategorie: Convert (4)
Compress (1)
Tiff (1)
Source (2)
Format (2)
others

Kategorie: statistic (3)
Unterkategorie: report (3)
counter (2)
record (1)

Abb. 9: Ergebnisse einer Wortfrequenzanalyse und Wortkategorisierung

Die Ergebnisse dieser Schritte werden durch erfahrene System- und Domänenexperten für die Gruppenbildung zur Erfassung ähnlicher Feature verwendet, um Redundanzen aufzudecken. Zur Unterstützung wurde ein Werkzeug entwickelt, das die Navigation im

Kategorienbaum und die Verwaltung der Featuregruppierung ermöglicht. Folgende Schritte sind für die Erkennung ähnlicher Feature auszuführen:

- Häufige Featurenennungen bilden Gruppen, die Wortkategorien werden als Schlüsselwort festgelegt. Die Experten können zusätzliche Schlüsselworte benennen (Abb. 10 a.)
- Features und korrespondierende Komponenten werden, wenn vorhanden, Untergruppen zugeordnet (Abb.10b). Dabei werden die Schlüsselworte als Suchfilter eingesetzt, um ähnliche Features innerhalb einer Gruppe und die dazugehörende Untergruppe zu ermitteln.

Abb. 10 zeigt ein Beispiel aus dem Bildverarbeitungsprojekt. Hier wurden vier ähnliche Features ermittelt, die darauf hin zu überprüfen sind, ob eine Redundanz aus dem System entfernt werden kann. Für korrekte Resultate sind neben der Expertenmeinung auch die Anforderungsbeschreibungen hinzu zu ziehen. Die Entfernung von Features oder die Substitution mit aggregierenden Features ist im Feature Modell und in den Korrespondenzen zu erfassen.

| Feature-Gruppe | Schlüsselworte |
|---|---|
| Image | Image, Convert, Compress, Store, format, TIFF |
| Statistics | Statistic, Counter Report |

a) Festlegen von Gruppen und Schlüsselworten

| Komp. | Feature | Untergruppe | Gruppe |
|---|---|---|---|
| II | Convert raw image date into standard format (TIFF) | Image TIFF operation | Image |
| ISIF | Compress images (TIFF FAX4/JPEG) | Image TIFF operation | Image |
| IR | Compress recieved images (TIFF FAX4) | Image TIFF operation | Image |
| IDB | Store images in TIFF format | Image TIFF operation | Image |

b) Die Schlüsselworte weisen auf eine Gruppe ähnlicher, in unterschiedlichen Komponenten implementierter TIFF Operationen hin.

Abb. 10: Ergebnisse des Beispiels

# 5 Zusammenfassung

Featuremodelle mit ihrer Brückenfunktion zwischen Kundenanforderungen und Elementen einer Referenzarchitektur sind signifikant einsetzbar in der evolutionären Weiterentwicklung einer Produktlinie. In dem Beitrag wurde zunächst gezeigt, wie sie für ein Architektur-Recovery genutzt werden können In dem vorgestellten Ansatz wird die schritthaltende Weiterentwicklung der Referenzarchitektur an neue Anforderungen aus der Produktdomäne unter Einbeziehung kleiner Refactoring Schritte empfohlen, um die Wartbarkeit der Architektur über längere Zeiträume zu erhalten. Der Ansatz wurde im Rahmen eines Projektes zur Bilderkennung evaluiert, bei dem ein System mit mehr als 4 Mio LOC bearbeitet wurde. Obwohl es bisher keine durchgängige standardisierte Werkzeugunterstützung gibt, konnten folgende Vorteile des Vorgehens bereits festgestellt werden:

- verbesserte Kommunikation durch Einsatz von Featuremodellen,
- persistentes und konsistentes Wissen über die Domäne Produktlinie und die Architektur in einem zentralisierten, mit Korrespondenzen versehenen Featuremodell,
- längerer Erhalt der Wartbarkeit der Referenzarchitektur durch kontinuierliche Umstrukturierung zur Reduktion der Komplexität.

# Referenzen

[Baker 1995] Baker, S., On finding duplication and near-duplication in large software systems. In: Wills, l., Newcomb, P. and Chikofsky E.: 2nd Working Conference on Reverse Engineering, IEEE, 1995: S. 86--95.

[Bass et al. 1998] Bass, L., Clements, P., Kazman, R., Software Architecture in Practice, Addison-Wesley, 1998.

[Baxter et al. 1998] Baxter, I., Yahin, A., Moura, L., Sant'anna, M., Bier, L., Clone detection using abstract syntax trees. Proc. Proc. of ICSM98, pp. 368-377, Nov. 1998, Bethesda, ML, USA.

[ Beck 1999.] Beck, K.: Extreme Programming Explained, Addison-Wesley, 1999

[Boellert 2002] Boellert, K., Objektorientierte Entwicklung von Software-Produktlinien zur Serienfertigung von Software-Systemen. Dissertation. TU Ilmenau, 2002.

[Bosch 2000] Bosch, K., Design & Use of Software Architectures, Addison-Wesley, 2000.

[Brooks et al. 1983] Brooks, R.: Towards a Theory of the Comprehension of Computer Programs. Intl. J. Man-Machine Studies 18, 1983.

[Carriere et al. 1999] Carriere, S.J., Kazman, R., Woods, S.G., Assessing and Maintaining Architectural Quality. Proc. 3rd European Conference on Software Maintenance and Reenigineering, (CSMR99) pp. 22-30.

[Clement et al.1999] Clement, Paul; Northrop, Linda: Framework for software product line practice, version 2.7. SEI, 1999.

[Czarnecki et al. 2000] Czarnecki, K., Eisenecker, U.W.: Generative Programming. Addison Wesley, 2000.

[Ducasse et al. 1999] Ducasse, S., Rieger, M., Demeyer, S., A Language Independent Approach for Detecting Duplicated Code. Proc. International Conference on Software Maintenance (ICSM99). IEEE, September 1999: S.109-118.

[ESAPS] Engineering Software Architectures, Processes and Platforms for System Families. ITEA project 99005 of the Eureka Sigma!2023 Program.http://www.esi.es/esaps/

[Hofman 2000] Hofmann Hubert. Requirements Engineering, Deutscher Universitäts-Verlag, 2000.

[Hofmeister et al. 2000] Hofmeister, C., Nord, R., Soni, D.: Applied Software Architecture. Addison Wesley, 2000

[Ibanez1996] Ibanez, M.; Rempp, H., European User Survey Analysis, ESPITI. 1996

[Johansson et al. 2002] Johansson, E., Mart Hst M.:Tracking Software Degradation in Software Product Lines through Measurement of Design Rule Violation, Proc of 14th International Conference on Software Engineering and Knowledg Engineering, 2002, S. 249 – 254.

[Kang et al. 1990] Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021, SEI Institute, Carnegie Mellon University. 1990

[Kotler1999] Kotler, Philip; Bliemel, Friedhelm: Marketing-Management: Analyse, Planung, Umsetzung und Steuerung. Schäffer-Poeschel, 1999.

[Krinke 2001] Krinke, J., Identifying similar code with program dependence graphs. Proc. of the 8th Working Conference on Reverse Engineering (WCRE2001). S. 301-309.

[Kruchten 1995] Kruchten, P. B.: The 4+1 View Model of Architecture. IEEE Software, 12(6): 42-50.

[Letovsky et al. 1986] Letovsky, S., Soloway E.: Delocalized Plans and Program Comprehension. IEEE Software 3, May 1986

[Loucupoulus 1995] Loucopoulos Pericles. System Requirements Engineering. McGraw Hill, 1995.

[OCL 2003] Object Constraint Language. UML 2.0 Specification. Online verfügbar unter http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML

[Pashov et al.] Pashov, I., Riebisch, M., Using Feature Modeling for Program Comprehension and Software Architecture Recovery. Proc. 10th IEEE Symposium and Workshops on Engineering Computer Based Systems.

[Rajlich et al. 1994] Rajlich, V., J. Doran, Gudla R.: Layered Explanations of Software: A Methodology for Program Comprehension, 3d Workshop on Program Comprehension, WPC'93, Washington, D.C., pp. 46-52, 1994.

[Riebisch et al. 2002] Riebisch, M., Böllert, K., Streitferdt, D., Philippow, I.: Extending Feature Diagrams with UML Multiplicities. 6th Conference on Integrated Design & Process Technology, Pasadena, California, USA. June 23 – 30, 2002.

[Riebisch 2003] Riebisch, M.: Evolution und Komposition von Softwaresystemen. Habilitationsschrift. TU Ilmenau, Fakultät für Informatik und Automatisierung, 2003 (eingereicht).

[Riva et al. 2002] Riva, C., Rodriguez, J. V.: Combining Static and Dynamic Views for architecture reconstruction. Proc. of the Sixth European Conference on Software Maintenance and Reengineering. 2002.

[Rugaber et al. 2000] Rugaber, S.: The use of domain knowledge in program understanding. Annals of Software Engineering, 2002.

[Sametinger et al. 2002] Sametinger, J.; Riebisch, M.: Evolution Support by Homogeneously Documenting Patterns, Aspects and Traces. 6th European Conference on Software Maintenance and Reengineering, Budapest, Hungary, March 11-13, 2002. Computer Society Press, 2002, pp. 134-140

[Sommerville 1997] Sommerville, I. Sawyer, P. : Requirements Engineering. John Wiley and Sons Ltd.

[Standish 1995] CHAOS report. CHAOS chronicles. Standish Group, 1995. www.pm2go.com,

[Storey et al. 1997] Storey, M. D., Fracchia F.D., Müller H. A.: Rigi: A Visualization Environment for Reverse Engineering. Proc. of International Conference on Software Engineering, Boston, U.S.A., May 17-23, 1997.