IAS-Forschungsberichte

Forschungsbericht
Institut für Automatisierungs- und Softwaretechnik

Hrsg.: Prof. Dr.-Ing. Dr. h. c. P. Göhner

Pascal Jost

Evolutionäres Domain-Engineering zur Entwicklung von Automatisierungssystemen

Band 2/2007

Universität Stuttgart

Evolutionäres Domain-Engineering zur Entwicklung von Automatisierungssystemen

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart zur Erlangung der Würde eines
Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

Vorgelegt von
Pascal Jost
aus Pforzheim

Hauptberichter:	Prof. Dr.-Ing. Dr. h. c. Peter Göhner
Mitberichter:	Prof. Dr.-Ing. Dr. h. c. mult. Paul J. Kühn
Tag der Einreichung:	05.12.2006
Tag der mündlichen Prüfung:	05.07.2007

Institut für Automatisierungs- und Softwaretechnik
der Universität Stuttgart

2007

IAS-Forschungsberichte

Band 2/2007

Pascal Jost

Evolutionäres Domain-Engineering zur Entwicklung von Automatisierungssystemen

D 93 (Diss. Universität Stuttgart)

Shaker Verlag
Aachen 2007

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Zugl.: Stuttgart, Univ., Diss., 2007

Printed in Germany.

ISBN 978-3-8322-6432-1
ISSN 1610-4781

Shaker Verlag GmbH • Postfach 101818 • 52018 Aachen
Telefon: 02407 / 95 96 - 0 • Telefax: 02407 / 95 96 - 9
Internet: www.shaker.de • E-Mail: info@shaker.de

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Automatisierungs- und Softwaretechnik (IAS) der Universität Stuttgart.

Danksagung

Zu allererst bedanke ich mich bei meinem Doktorvater und Leiter des IAS, Prof. Dr-Ing. Dr. h. c. Peter Göhner sehr herzlich für die Unterstützung, die vielen wertvollen Anregungen, die zahlreichen kritischen Diskussionen, die Geduld sowie die Übernahme des Hauptberichts.

Mein herzlicher Dank gilt Prof. Dr-Ing. Dr. h. c. mult. Paul J. Kühn, Leiter des Institutes für Kommunikationsnetze und Rechnersysteme, für das Interesse und die Übernahme des Mitberichts.

Bei meinen ehemaligen Kolleginnen und Kollegen sowie den derzeitigen Mitarbeitern am IAS bedanke ich mich ganz herzlich für die tolle Zusammenarbeit, die kollegiale Unterstützung, die vielen konstruktiven Diskussionsbeiträge und die Motivation. Ganz besonders bedanke ich mich bei allen, die das Manuskript zur vorliegenden Arbeit gelesen haben, für ihren Einsatz und die hilfreichen Anmerkungen.

Nicht unerwähnt lassen möchte ich die vielen Studenten, die durch Studien- und Diplomarbeiten einen wesentlichen Beitrag zum Gelingen dieser Arbeit beitrugen.

Ganz besonders bedanke ich mich bei meiner Familie für den Rückhalt, die moralische Unterstützung sowie die unendliche Geduld. Das gilt insbesondere für meine Frau, die mich während der Entstehung dieser Arbeit oft entbehren musste.

Stuttgart, im Juli 2007

Pascal Jost

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

CAN	**C**ontroller **A**rea **N**etwork
DoD	**D**epartment **o**f **D**efense (Verteidigungsministerium der USA)
DSSA	**D**omänen**s**pezifische **S**oftware**a**rchitektur (**D**omain-**S**pecific **S**oftware **A**rchitecture)
FAST	**F**amily-Oriented **A**bstraction, **S**pecification and **T**ranslation
FODA	**F**eature-**O**riented **D**omain **A**nalysis
JODA	**J**oint Integrated Avionics Working Group **O**bject-oriented **D**omain **A**nalysis
KMU	**K**leine und **m**ittelständische **U**nternehmen
ODM	**O**rganization **D**omain **M**odeling
PuLSE	**P**rod**u**ct-**L**ine **S**oftware **E**ngineering
ROOM	**R**eal-Time **O**bject-**o**riented **M**odeling
SWK	**S**oft**w**are**k**omponente
UML	**U**nified **M**odeling **L**anguage
VP	**V**ariations**p**unkt (**V**ariation-**P**oint)

Begriffsverzeichnis

Anwendbarkeit (Usability): Die Anwendbarkeit (Usability) ist ein Qualitätsmerkmal einer *mehrfach verwendbaren Softwareeinheit*. Sie beschreibt, wie einfach mehrfach verwendbare Softwareeinheiten angewendet werden können. Dabei sind insbesondere die Instanziierung und die Integration in eine neue Umgebung relevant [MMYA02].

Application-Engineering: Application-Engineering ist der Prozess der Analyse, Spezifikation und Implementierung von Applikationen. Im Kontext des *Domain-Engineering* wird das Application-Engineering mit Hilfe der mehrfach verwendbaren Ergebnisse des Domain-Engineering durchgeführt [SEI03].

Mehrfach verwendbare Arbeitsergebnisse (Assets): Der Begriff mehrfach verwendbares Arbeitsergebnis ist weiter gefasst als *mehrfach verwendbare Softwareeinheiten*. Es ist die mehrfach verwendbare Form jedes Arbeitsergebnisses, das während der Softwareentwicklung anfällt. Dies kann neben der Softwarearchitektur und der Implementierung beispielsweise ein domänenspezifisches System-Modell, eine domänenspezifische Benutzungsanleitung oder Ähnliches sein. Ein mehrfach verwendbares Arbeitsergebnis besitzt geeignete Konfigurationsmöglichkeiten und ggf. Schnittstellen zur Integration [SCK+96, CzEi00].

Domänenanalyse: Die Domänenanalyse (Domain Analysis) ist eine Hauptaktivität des *Domain-Engineering*. Die Aufgabe der Domänenanalyse ist Definition einer Menge von mehrfach verwendbaren Anforderungen für die Softwareprodukte der Domäne [CzEi00].

Domain-Engineering: Domain-Engineering ist der Prozess der Analyse, Spezifikation und Implementierung von mehrfach verwendbaren Softwareeinheiten in einer Domäne. Die mehrfach verwendbaren Softwareeinheiten werden bei der Entwicklung mehrerer Softwareprodukte eingesetzt. Die drei Hauptaktivitäten des Domain-Engineering sind: *Domänenanalyse* (Domain Analysis), *Domänenentwurf* (Domain Design) und *Domänenimplementierung* (Domain Implementation) [SEI03].

Domänenentwurf: Der Domänenentwurf (Domain Design) ist eine Hauptaktivität des *Domain-Engineering*. Die Aufgabe des Domänenentwurfs ist die Erstellung einer gemeinsamen Architektur für die Softwareprodukte der Domäne [CzEi00].

Domänenimplementierung: Die Domänenimplementierung (Domain Implementation) ist eine Hauptaktivität des *Domain-Engineering*. Die Aufgabe der Domänenimplementierung ist die Umsetzung von mehrfach verwendbaren Softwareeinheiten [CzEi00].

Modulare Einheit: Modulare Einheit nach der Definition von [KLM+97] wird in dieser Arbeit wegen der Verwechslungsgefahr mit dem Begriff Softwarekomponente als Synonym für den Begriff Systemkomponente verwendet (vgl. [Göhn03]). Demnach ist eine modulare Einheit ein abgegrenzter Teil eines Softwaresystems. Sie dient als Baustein für die physikalische Struktur einer Anwendung. Beispiele für modulare Einheiten sind Funktionen, Prozeduren, abstrakte Datenobjekte, abstrakte Datentypen, Klassen [Balz96].

Hierarchische Einheit: Eine hierarchische Einheit ist eine *modulare Einheit*, die eine als potenziell mehrfach verwendbar erkannte Lösung enthält. Zu einer hierarchischen Einheit existieren ähnliche modulare Einheiten in anderen bestehenden Applikationen.

Interessensgruppen (Stakeholder): Interessensgruppen können firmeninterne Gruppen, wie z.B. Geschäftsleitung, Marketingabteilung, Hardware- sowie Softwareentwickler oder externe Gruppen, wie z.B. Kunden, Standardisierungsgremien und Gesetzgeber sein, die ein Interesse an einer Domäne haben [CzEi00].

Konfigurationsmöglichkeit: Konfigurationsmöglichkeiten dienen der geplanten Anpassung *mehrfach verwendbarer Softwareeinheiten* für den Einsatz in spezifischen Applikationen. Sie erhöhen die *Anwendbarkeit* (Usability) der mehrfach verwendbaren Softwareeinheiten. *Domänenspezifische Softwarearchitekturen* können beispielsweise über die Auswahl einer *Softwarekomponente* aus einer zuvor festgelegten Menge alternativer Softwarekomponenten angepasst werden. Mehrfach verwendbare Softwarekomponenten bieten Parameter zur Anpassung.

Mehrfachverwendbarkeit: Die Mehrfachverwendbarkeit nach der Definition von Mili et al. [MMYA02] wird durch die beiden Qualitätsmerkmale *Nützlichkeit* (Usefulness) und *Anwendbarkeit* (Usability) festgelegt.

Merkmal (Feature): (1) Nach der Methode FODA [KCH+90] ist ein Merkmal eine generelle Eigenschaft von Applikationen innerhalb einer Domäne, das für Kunden bzw. Benutzer sichtbar ist. (2) Die Methode ODM [SCK+96] fasst den Begriff weiter. Für sie ist nicht die Sichtbarkeit für Kunden bzw. Benutzer wichtig, sondern die Relevanz für einen oder mehrere *Interessensgruppen* (Stakeholder).

Nützlichkeit (Usefulness): Die Nützlichkeit (Usefulness) ist ein Qualitätsmerkmal einer *mehrfach verwendbaren Softwareeinheit*. Sie beschreibt, wie nützlich mehrfach verwendbare Softwareeinheiten bei ihrer Anwendung sind und zeigt damit die Nachfrage nach einer mehrfach verwendbaren Softwareeinheit bei der Entwicklung neuer Applikationen an [MMYA02].

Softwarearchitektur: Eine Softwarearchitektur beschreibt die Struktur des Softwaresystems durch *modulare Einheiten* und deren Beziehungen untereinander [Balz96].

Domänenspezifische Softwarearchitektur: (auch mehrfach verwendbare Softwarearchitektur) Eine domänenspezifische Softwarearchitektur beinhaltet den Bauplan für die Applikationen einer Domäne. Sie besteht neben *Softwarekomponenten* zusätzlich aus deren Verbindungen sowie vordefinierten Parametern. Um eine Applikation zu erstellen, hat der Anwender die Möglichkeit, Teile der domänenspezifischen Softwarearchitektur selbst an vordefinierten Stellen zu adaptieren oder Softwarekomponenten auszutauschen.

Mehrfach verwendbare Softwareeinheit: Oberbegriff für mehrfach verwendbare Softwarearchitekturen (domänenspezifischen Softwarearchitekturen), Softwarekomponenten und Implementierungen.

Mehrfach verwendbare Softwarekomponente: Bei mehrfach verwendbaren Softwarekomponenten handelt es sich um generische Software-Bausteine mit definiertem Verhalten, Schnittstellen und Parametern. Sie werden bei der Anwendung durch die Wahl der Parameter an ihre Anforderungen angepasst und durch die Verbindung mit anderen Softwarekomponenten zu einer Applikation integriert [Göhn98, Dujm01].

Variationspunkt: Ein Variationspunkt dient der Realisierung optionaler und alternativer Eigenschaften in *mehrfach verwendbaren Softwareeinheiten*. Dabei wird genau festgelegt welche Optionen bzw. Alternativen bestehen. Er ermöglicht die Bildung von Varianten, die bei der Instanziierung durch die Auswahl der definierten Optionen bzw. Alternativen erzeugt werden.

Zusammenfassung

Die Mehrfachverwendung von Software gewinnt in der Prozessautomatisierung zunehmend an Bedeutung. Besonders kleine und mittelständische Unternehmen (KMU) sehen die Mehrfachverwendung als Möglichkeit, dem wachsenden Zeit- und Kostendruck zu begegnen. Der Mangel an existierender mehrfach verwendbarer Software, die sich für den Einsatz in Prozessautomatisierungssystemen eignet, zwingt die Unternehmen dazu, solche Software selbst zu entwickeln. Die heute verfügbaren Domain-Engineering-Methoden zur Unterstützung der Entwicklung von Software für die Mehrfachverwendung sind für allgemeine Softwaresysteme ausgelegt und für den Einsatz in Großunternehmen optimiert. Kleine und mittelständische Unternehmen, die Automatisierungssysteme erstellen, benötigen Domain-Engineering-Methoden, die auf ihre finanziellen und personellen Möglichkeiten angepasst sind und die spezifischen Merkmale von Automatisierungssystemen berücksichtigen.

In der vorliegenden Arbeit wird ein Konzept zur iterativen Entwicklung mehrfach verwendbarer Softwarekomponenten und einer domänenspezifischen Softwarearchitektur für Automatisierungssysteme vorgestellt, das auf dem Evolutionsprinzip basiert. Das Konzept erlaubt die zeitliche Verteilung des Entwicklungsaufwandes durch das iterative Vorgehen. In den einzelnen Iterationsschritten werden Softwarekomponenten separat entwickelt. Dazu unterstützt das Konzept die frühe Zerlegung der Domäne in Sub-Domänen, aus denen die Softwarekomponenten entstehen, sowie die Integration der Softwarekomponenten zu einer domänenspezifischen Softwarearchitektur. Die Entwicklung erfolgt nach dem Vorbild von Domain-Engineering-Methoden. Bei der Entwicklung von Automatisierungssystemen sind insbesondere die Vorgänge im zu automatisierenden technischen Prozess sowie die Einrichtungen, die zur Automatisierung notwendig sind, zu berücksichtigen. Um die relevanten Unterschiede und Gemeinsamkeiten von Automatisierungssystemen beim Domain-Engineering berücksichtigen zu können, wird ein Fragenkatalog eingesetzt. Er unterstützt bei der Analyse der Automatisierungsaufgaben und der zur Automatisierung eingesetzten Einrichtungen. Das Konzept wird in einer Methodik umgesetzt, welche die Anwender mit definierten Aktivitäten und Produkten bei der evolutionären Entwicklung mehrfach verwendbarer Software unterstützt.

Die Methodik führt zu einer zeitlichen Verteilung des Entwicklungsaufwandes sowie zu einer frühen Nutzung von Teilergebnissen. Damit erfüllt sie die Grundvoraussetzung für den Einsatz in KMU. Insbesondere bei der Analyse werden Automatisierungsaufgaben und Einrichtungen zur Automatisierung berücksichtigt. Dadurch wird die Berücksichtigung der für Automatisierungssysteme relevanten Informationen bei der Entwicklung der mehrfach verwendbaren Software unterstützt.

Abstract

Software reuse in modern industrial automation systems is gaining in importance. Small and medium-sized companies (SMC) in particular, see reuse as a possibility to lower costs and shorten time-to-market. The market lacks existing reusable software which is applicable for industrial automation systems. Therefore, companies are forced to develop reusable software on their own. The currently available Domain Engineering Methods to support the development of software for reuse are tailored for general software systems and optimized for the usage in large companies. SMCs developing software for industrial automation systems will need Domain Engineering Methods that fit their financial and personnel requirements and consider the specific features of industrial automation systems.

In this thesis a concept for the iterative development of software components and a domain-specific software architecture will be introduced. It is based on the principle of evolution. The concept enables the temporal distribution of the development via iterative development. Within the single steps of iteration the software components will be developed separately. Therefore, the concept supports the early decomposition of the domain into sub-domains, from which software components will result. Further, the integration of these software components to a domain-specific software architecture is supported. The concept for the development of software components is modeled on domain engineering methods that consider common and variable features of legacy software. Software for industrial automation systems is mainly influenced by automation tasks and facilities necessary for automation. This leads to special properties, such as real-time constraints and access on computer hardware and peripheral hardware. To include the relevant commonalities and variabilites of software for industrial automation these special properties need to be considered during domain engineering. Therefore, domain engineering is extended by a catalogue of questions which support the analysis of the automation task and the facilities used for automation. The concept is used in a methodology that supports the customer with defined activities and products during the evolutionary development of reusable software.

The conceived methodology leads to a temporal distribution of the development effort and to an early usage of partial results. Therefore, the basic pre-condition for the usage of the methodology for SMC is fulfilled. In addition, the methodology considers automation tasks and facilities used for automation especially during analysis. This enables the consideration of information which is relevant for software for industrial automation during the development of reusable software for industrial automation.

1 Einleitung und Motivation

Blicken wir zunächst auf die großartigen Leistungen der Evolution: Wir verdanken ihr eine gewaltige Fülle unterschiedlicher Baupläne von unbeschreiblicher Schönheit, Komplexität und Eleganz, ganz zu schweigen von deren Funktionstüchtigkeit.

Ray Kurzweil

1.1 Bedeutung mehrfach verwendbarer Softwareeinheiten für Automatisierungssysteme

Die Software übernimmt einen immer größer werdenden Teil der Funktionalität von Prozessautomatisierungssystemen und löst damit die Hardware als differenzierendes Wettbewerbsmerkmal ab. Sie bietet gegenüber der Hardware mehr Flexibilität bei Änderungen. Dieser Vorteil wird genutzt, um durch Anpassungen spezielle Kundenwünsche zu erfüllen. Dabei entsteht eine Menge angepasster Automatisierungssysteme, die Unterschiede und Gemeinsamkeiten zueinander aufweisen. Mit dem Anteil der Software an der Funktionalität von Prozessautomatisierungssystemen nimmt auch der Anteil der Automatisierungssysteme am Entwicklungsaufwand und den Entwicklungskosten zu.

Bei Softwareprojekten zeichnet sich seit Anfang der neunziger Jahre ein Trend ab, der weg von der kompletten Neuentwicklung von Software und hin zum Einsatz mehrfach verwendbarer Softwareeinheiten führt. Mehrfach verwendbare Softwareeinheiten werden für den Einsatz in zukünftigen Applikationen entwickelt und sind für die Anwendung bei der Entwicklung neuer Applikationen anpassbar und integrierbar. Die Mehrfachverwendung bringt vielfältige Vorteile mit sich. So verringert sich beispielsweise der Entwicklungsaufwand für Applikationen. Es können kürzere Entwicklungszeiten und niedrigere Entwicklungskosten erreicht werden [MMYA02, Szyp98].

Für Anwender mehrfach verwendbarer Softwareeinheiten besteht die Möglichkeit, diese selbst zu entwickeln oder sie von Drittanbietern zu beziehen. Beides stellt eine Investition dar, die sich durch den mehrfachen Einsatz der Ergebnisse amortisiert [ADH+00]. Um eine schnelle Amortisierung zu erreichen, müssen die mehrfach verwendbaren Softwareeinheiten in vielen Applikationen einsetzbar sein. Das ist dann der Fall, wenn die Applikationen Gemeinsamkeiten aufweisen, die von den mehrfach verwendbaren Softwareeinheiten abgedeckt werden.

Besteht eine Menge durch Anpassung entstandener Applikationen für Automatisierungssysteme, erscheint die Mehrfachverwendung lohnend, da die einzelnen Applikationen Gemeinsamkeiten zueinander aufweisen. Die Hersteller solcher Applikationen streben deshalb den Einsatz

mehrfach verwendbarer Softwareeinheiten für ihre Produkte an. Daraus resultiert ein Bedarf an geeigneten mehrfach verwendbaren Softwareeinheiten. Speziell bei Automatisierungssystemen müssen diese auch Anforderungen erfüllen, die aus den Automatisierungsaufgaben und den eingesetzten Einrichtungen zur Automatisierung, wie beispielsweise Rechnerhardware oder Prozessperipherie, resultieren [LaGö99a].

Im Gegensatz zum Internet- oder Windows-Umfeld finden sich nur selten Anbieter, die mehrfach verwendbare Softwareeinheiten für die spezifischen Anforderungen von Automatisierungssystemen liefern. Interessenten bleibt daher nur der Weg, die mehrfach verwendbaren Softwareeinheiten selbst zu entwickeln.

1.2 Problemstellung bei der Entwicklung mehrfach verwendbarer Softwareeinheiten

Bei der Entwicklung mehrfach verwendbarer Softwareeinheiten bedarf es eines anderen Vorgehens als bei der Entwicklung einzelner Applikationen, da unter anderem auch projektübergreifende Gesichtspunkte zu berücksichtigen sind. In der Praxis wird Wiederverwendung häufig ad hoc durchgeführt. Dabei werden Teile bestehender Applikationen in die aktuelle Applikation kopiert und an die neuen Anforderungen angepasst. Dieses Vorgehen wird Ad-hoc-Wiederverwendung genannt und bedingt keinen nennenswerten zusätzlichen Entwicklungsaufwand. Die Ergebnisse des Vorgehens sind nicht notwendigerweise in zukünftigen Applikationen nützlich (Usefulness) und nicht notwendigerweise einfach anwendbar (Usability). Das Vorgehen ist nicht strukturiert, aber einfach anzuwenden.

Domain-Engineering ermöglicht die gezielte Entwicklung von Softwareeinheiten für die Mehrfachverwendung [CzEi00, MMYA02]. Die Ergebnisse sind für die Entwicklung zukünftiger Applikationen nützlich sowie einfach anwendbar. Die Entwicklung der mehrfach verwendbaren Softwareeinheiten wird durch ein strukturiertes Vorgehen unterstützt. Es nutzt bestehende Applikationen innerhalb eines Anwendungsbereiches (Domäne) zur Analyse von Unterschieden und Gemeinsamkeiten. Die Ergebnisse dieser Analyse dienen als Ausgangspunkt für den Entwurf und die Implementierung konfigurierbarer Softwarekomponenten und Softwarearchitekturen. Domain-Engineering ist mit Entwicklungsaufwand verbunden, der in Vorleistung zu erbringen ist. Die Vorteile der Mehrfachverwendung kommen deshalb erst nach mehreren Einsätzen der mehrfach verwendbaren Ergebnisse zum Tragen.

Aus der Literatur bekannte Domain-Engineering-Methoden entstanden im Rahmen von Militärprojekten und in Großunternehmen. Sie sind an die Bedürfnisse und Möglichkeiten ihrer Anwender angepasst. Automatisierungssysteme dagegen werden häufig in kleinen und mittelständischen Unternehmen (KMU) entwickelt. Diesen fehlen oft die finanziellen Mittel, um die Investition für die Entwicklung mehrfach verwendbarer Softwareeinheiten im Voraus zu erbringen [KMSW00].

Auch für Domänen von Automatisierungssystemen wurden mit Hilfe von Domain-Engineering bereits mehrfach verwendbare Softwareeinheiten entwickelt [BrCl96]. Eine explizite Unterstützung durch das Domain-Engineering für die spezifischen Eigenschaften von Automatisierungssystemen gibt es jedoch nicht.

1.3 Zielsetzung der Arbeit

Um auch KMU den Einsatz von Domain-Engineering zu ermöglichen, ist der in Vorleistung zu erbringende Entwicklungsaufwand zeitlich zu verteilen bzw. zu reduzieren. Das zeitliche Verteilen des Entwicklungsaufwandes ist nur sinnvoll im Zusammenspiel mit der frühen Nutzung der entstehenden Teilergebnisse. Damit lässt sich eine zeitliche Verteilung der Investition für das Domain-Engineering sowie eine frühe Amortisierung erreichen. Neben den wirtschaftlichen Voraussetzungen muss das Domain-Engineering spezifische Eigenschaften von Automatisierungssystemen, wie beispielsweise Echtzeiteigenschaften oder Schnittstellen zu Rechnerhardware, Prozessperipherie, usw., berücksichtigen.

Das Ziel der vorliegenden Arbeit ist die Konzeption einer Domain-Engineering-Methode zur Entwicklung mehrfach verwendbarer Softwareeinheiten in Form einer domänenspezifischen Softwarearchitektur und mehrfach verwendbarer Softwarekomponenten. Diese Domain-Engineering-Methode muss spezifische Eigenschaften von Automatisierungssystemen und die Einschränkungen bei der Anwendung in KMU berücksichtigen.

1.4 Gliederung

Im folgenden Kapitel werden die Grundlagen des Domain-Engineering erläutert. Die Ziele, die Vorgehensweise und die möglichen Auswirkungen bei der Anwendung von Domain-Engineering werden betrachtet und die Eigenschaften bestehender Domain-Engineering-Methoden analysiert.

Kapitel 3 untersucht die Anwendbarkeit von Domain-Engineering-Methoden. Dabei werden Randbedingungen in Großunternehmen und in KMU sowie Eigenschaften von Automatisierungssystemen berücksichtigt. Aus den Ergebnissen der Untersuchung werden spezielle Anforderungen an Domain-Engineering-Methoden abgeleitet, die sich für den Einsatz in KMU und zur Entwicklung von Automatisierungssystemen eignen.

In Kapitel 4 wird der im Rahmen dieser Arbeit entwickelte Lösungsansatz des evolutionären Domain-Engineering eingeführt. Das aus der Natur bekannte Evolutionsprinzip wird auf das Domain-Engineering übertragen und das resultierende evolutionäre Domain-Engineering skizziert. Für das evolutionäre Domain-Engineering ergeben sich Rahmenbedingungen, die sich aus dem Evolutionsprinzip ableiten.

In Kapitel 5 werden die in Kapitel 2 dargestellten Merkmale des Domain-Engineering, die in Kapitel 3 abgeleiteten Anforderungen sowie die in Kapitel 4 beschriebenen Rahmenbedingungen aufgegriffen und ein Konzept zur iterativen Entwicklung mehrfach verwendbarer Softwareeinheiten erstellt. Das Konzept setzt das evolutionäre Vorgehen durch die iterative Entwicklung mehrfach verwendbarer Teilergebnisse und die ständige Verbesserung der Mehrfachverwendbarkeit dieser Teilergebnisse um.

In Kapitel 6 wird das in Kapitel 5 entwickelte Konzept in einer Methodik zur Unterstützung der Anwender bei der strukturierten Entwicklung mehrfach verwendbarer Softwarekomponenten und einer domänenspezifischen Softwarearchitektur umgesetzt.

In Kapitel 7 wird die Methodik, die in Kapitel 6 abstrakt beschrieben ist, verfeinert und detailliert erläutert. Aktivitäten werden in Teilaktivitäten zerlegt, Regeln sowie Anleitungen für die Durchführung aufgestellt. Zur Dokumentation der Arbeitsergebnisse werden Notationen eingeführt.

Kapitel 8 enthält ein Anwendungsbeispiel des evolutionären Domain-Engineering. Anhand der Domäne der Sortieranlagen werden das evolutionäre Domain-Engineering und dessen Auswirkungen auf die Domäne über mehrere Generationen hinweg aufgezeigt. Es wird insbesondere auf die zeitliche Verteilung des Entwicklungsaufwandes sowie die iterative Entwicklung der mehrfach verwendbaren Softwarekomponenten und der domänenspezifischen Softwarearchitektur eingegangen.

Abschließend wird in Kapitel 9 eine Bewertung der Ergebnisse unter Berücksichtigung der aufgestellten Anforderungen durchgeführt. Es folgen eine Zusammenfassung und ein Ausblick auf Möglichkeiten zur Weiterführung der vorliegenden Arbeit.

2 Grundlagen von Domain-Engineering

In diesem Kapitel wird zunächst die Wiederverwendung von Software ohne den Einsatz von Domain-Engineering beschrieben, um dann die Grundlagen der Mehrfachverwendung von Software mit Domain-Engineering zu erarbeiten. Dazu erfolgt eine allgemeine Betrachtung der Ziele und Merkmale des Domain-Engineering. Anschließend werden Auswirkungen beim Einsatz von Domain-Engineering auf eine anwendende Organisation beschrieben. Das Kapitel schließt mit der Vorstellung von verschiedenen aus der Literatur bekannten Domain-Engineering-Methoden und der Betrachtung ihrer spezifischen Eigenschaften.

2.1 Ad-hoc-Wiederverwendung

Wiederverwendung ohne Domain-Engineering wird ohne strategische Planung und damit ad hoc durchgeführt. Die Ad-hoc-Wiederverwendung wird auch „Clone and Own“ oder „Copy, Paste and Modify“ genannt. Bei der Ad-hoc-Wiederverwendung erkennt ein erfahrener Softwareentwickler, dass ein aktuelles Problem bereits in einem abgeschlossenen Projekt gelöst wurde. Die bestehende Lösung wird aus diesem Projekt in das aktuelle Projekt kopiert. Welche Formen diese Lösungen haben, wird bei der Ad-hoc-Wiederverwendung nicht berücksichtigt. Es kann sich sowohl um Fragmente des Source-Codes, um eine oder mehrere modulare Einheiten[1] der Softwarearchitektur als auch um vollständige Applikationen handeln. Die Lösungen besitzen keine Konfigurationsmöglichkeiten. Ihre Softwarearchitektur und ihr Source-Code sind gegebenenfalls bezüglich des Funktionsumfangs, des Verhaltens und der Schnittstellen an die Anforderungen des aktuellen Projekts anzupassen. Die Effizienz der Ad-hoc-Wiederverwendung hängt maßgeblich vom Wissen und der Erfahrung der Softwareentwickler ab. Dieses Wissen ist nicht dokumentiert. Dadurch haben Softwareentwickler mit weniger Erfahrung Schwierigkeiten, von der Ad-hoc-Wiederverwendung zu profitieren.

Betrachtet man die einzelnen ad hoc wieder verwendeten Lösungen über den Lebenszyklus der damit entwickelten Produkte, ergibt sich eine Menge von Applikationen, die gleiche oder ähnliche Teile aufweisen. Solche Teile kommen in angepasster Form in allen oder in einigen betrachteten Applikationen vor. Abbildung 2.1 zeigt ein mögliches Szenario der Ad-hoc-Wiederverwendung.

[1] Aufgrund der Verwechslungsgefahr mit dem Begriff „Softwarekomponente“ wird in dieser Arbeit anstelle von „Systemkomponente“ der Begriff „modulare Einheit“ nach [KLM+97] verwendet.

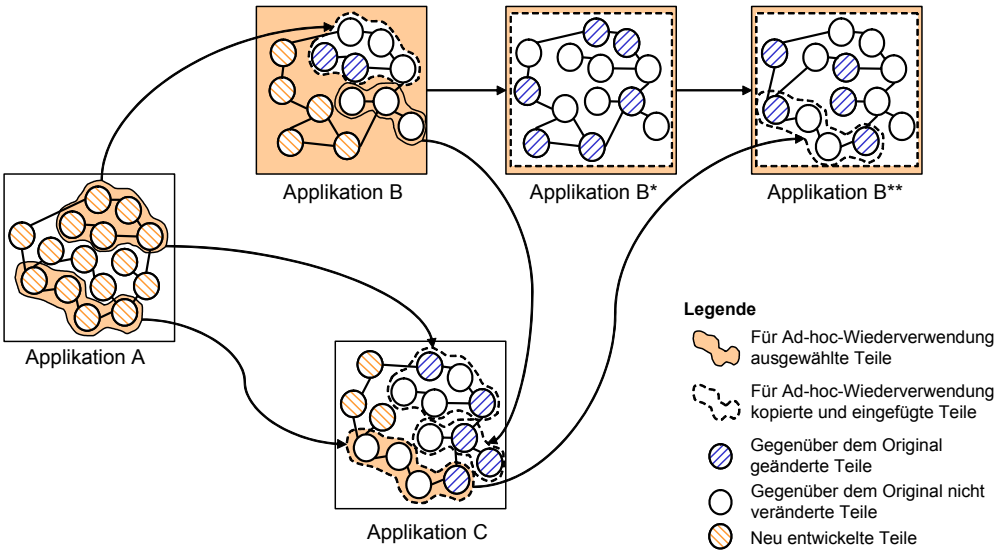


Abbildung 2.1: Entstehungsgeschichte von Applikationen durch die Entwicklung mit Ad-hoc-Wiederverwendung

Bei der Entwicklung neuer Applikationen wird auf den Fundus an Lösungen in bestehenden Applikationen zurückgegriffen. So entsteht zum einen eine Menge von Applikationen, die Unterschiede und Gemeinsamkeiten aufweisen und zum anderen Wissen und Erfahrung der Softwareentwickler über die Zusammenhänge ad hoc wieder verwendeter Lösungen.

2.2 Ziele des Domain-Engineering

Im Gegensatz zur Ad-hoc-Wiederverwendung bietet Domain-Engineering Unterstützung bei der systematischen Entwicklung mehrfach verwendbarer Softwareeinheiten. Man spricht auch von der Entwicklung für die Mehrfachverwendung. Die entstehenden mehrfach verwendbaren Softwareeinheiten werden bei der Entwicklung zukünftiger Applikationen durch Konfiguration und Integration eingesetzt. Aufwändiges Neuentwickeln wird dabei so weit wie möglich vermieden.

Auf den Internet-Seiten des Software Engineering Institut der Carnegie Mellon University wird der Begriff *Domain-Engineering* wie folgt definiert: Domain-Engineering ist der Prozess der Analyse, Spezifikation und Implementierung von mehrfach verwendbaren Softwareeinheiten in einer Domäne. Die mehrfach verwendbaren Softwareeinheiten werden bei der Entwicklung von Softwareprodukten vielfach eingesetzt. Die drei Hauptaktivitäten des Domain-Engineering sind: *Domänenanalyse* (Domain Analysis), *Domänenentwurf* (Domain Design) und *Domänen-implementierung* (Domain Implementation) [SEI04]. Aus dieser Definition lassen sich die zwei Ziele des Domain-Engineering ableiten. Zum einen die Entwicklung von mehrfach verwendbaren Softwareeinheiten und zum anderen das strukturierte Vorgehen bei der Entwicklung der mehrfach verwendbaren Softwareeinheiten in Form eines definierten Entwicklungsprozesses.

2.2.1 Entwicklung mehrfach verwendbarer Softwareeinheiten

In diesem Abschnitt wird der Begriff Mehrfachverwendbarkeit diskutiert, um darzustellen, welche Eigenschaften eine mehrfach verwendbare Softwareeinheit haben muss, die im Rahmen des Domain-Engineering entwickelt wird. Anschließend werden verschiedene Formen mehrfach verwendbarer Softwareeinheiten vorgestellt und ihre Einsatzgebiete diskutiert.

Mehrfachverwendbarkeit von Softwareeinheiten

In der Literatur sind viele Definitionen des Begriffes Mehrfachverwendbarkeit zu finden. Es werden hier drei häufig zitierte Definitionen vorgestellt.

Die Mehrfachverwendbarkeit nach der Definition von Mili et al. [MMYA02] wird durch die beiden Merkmale *Nützlichkeit* (Usefulness) und *Anwendbarkeit* (Usability) festgelegt. Die Nützlichkeit beschreibt die Häufigkeit der Anwendung, also die Nachfrage nach einer mehrfach verwendbaren Softwareeinheit. Die Anwendbarkeit ist die Einfachheit, mit der mehrfach verwendbare Softwareeinheiten instanziiert und integriert werden können. Die Instanziierung beinhaltet die Konfiguration und die Parametrisierung zur Anpassung der mehrfach verwendbaren Softwareeinheiten an die Anforderungen einer Applikation. Die Integration ist das Einfügen der Instanzen in die Struktur einer Applikation. Die Nützlichkeit und die Anwendbarkeit sind nicht unabhängig voneinander. Eine kleine Softwareeinheit wird häufig eingesetzt und erhält deshalb gute Werte für die Nützlichkeit. Um eine Applikation durch die Anwendung vieler kleiner Softwareeinheiten zu erstellen, muss mit einem hohen Aufwand für die Integration gerechnet werden. Daraus ergibt sich ein schlechter Wert für die Anwendbarkeit. Große mehrfach verwendbare Softwareeinheiten haben in der Regel schlechtere Werte für die Nützlichkeit und bessere für die Anwendbarkeit, da sie nicht so häufig eingesetzt werden können, aber weniger Aufwand bei der Erstellung einer Applikation bedeuten.

Andere Definitionen, wie beispielsweise [Somm95] oder [Karl95], sehen die Mehrfachverwendbarkeit als Qualitätsmerkmal einer Software. Dabei wird die Mehrfachverwendbarkeit (Reusability) je nach Autor aus unterschiedlichen Qualitätsmerkmalen zusammengesetzt. Eine Gegenüberstellung der Zusammensetzung der Mehrfachverwendbarkeit nach [Somm95] und [Karl95] ist in Tabelle 2.1 dargestellt. Diese Qualitätsmerkmale werden durch verschiedene Eigenschaften einer Softwarearchitektur beeinflusst. Dabei sind einige allgemein bekannte Regeln für eine gute Softwarearchitektur auf Softwarearchitekturen für die Mehrfachverwendbarkeit übertragbar. Darunter sind die Kopplung und die Bindung der modularen Einheiten einer Softwarearchitektur. Eine modulare Einheit, die für die Mehrfachverwendung geeignet ist, sollte eine nicht triviale Zielsetzung realisieren. Außerdem sollten die mehrfach verwendbaren modularen Einheiten eine geringe Kopplung und eine hohe Bindung besitzen. Sie müssen so unabhängig wie möglich sein. Jede Abhängigkeit bedeutet eine Einschränkung ihrer Mehrfachverwendbarkeit. Außer diesen gibt es noch weitere spezielle Regeln, die ausschließlich in Verbindung mit der Entwicklung für die Mehrfachverwendung anzuwenden sind. Darunter

fällt die Betrachtung der Variabilitäten unter dem Gesichtspunkt der Kopplung und Bindung. Variabilitäten sollten in modularen Einheiten gekapselt werden. Daraus folgt für Variabilitäten ebenfalls die Forderung nach einer geringen Kopplung sowie einer hohen Bindung.

Tabelle 2.1: Zusammensetzung der Mehrfachverwendbarkeit aus Qualitätsmerkmalen

Mehrfachverwendbarkeit nach [Somm95]	**Mehrfachverwendbarkeit nach [Karl95]**
Anpassbarkeit (Adaptability)	Anpassbarkeit (Adaptability)
Verständlichkeit (Understandability)	Verständlichkeit (Understandability)
Wartbarkeit (Maintainability)	Portierbarkeit (Portability)
Vollständigkeit (Completeness)	Vertrauenswürdigkeit (Confidence)

Die Definitionen nach [Somm95] und [Karl95] sind im Bereich der von [MMYA02] beschriebenen Anwendbarkeit (Usability) einzuordnen. Die Nützlichkeit (Usefulness) kann über die Qualitätsmerkmale einer mehrfach verwendbaren Softwareeinheit nicht beurteilt werden. Dazu ist immer die Betrachtung bestehender Anwendungen der entsprechenden Softwareeinheit notwendig. Da [Somm95] und [Karl95] die Anwendbarkeit (Usefulness) nicht berücksichtigen, wird im Folgenden die Definition der Mehrfachverwendbarkeit nach [MMYA02] verwendet.

Formen mehrfach verwendbarer Softwareeinheiten

Zur Realisierung mehrfach verwendbarer Softwareeinheiten werden in der Literatur verschiedene Möglichkeiten beschrieben. Diese bieten zur Unterstützung der Anwendbarkeit (Usability) Mechanismen zur Spezialisierung und Anpassung. Es handelt sich dabei um generische Bausteine, die mit ihren Merkmalen eine definierte Menge von Varianten abdecken. Von diesen generischen Bausteinen können Instanzen gebildet werden, die jeweils genau eine dieser möglichen Varianten darstellen. Bei der Umsetzung der generischen Bausteine wird die Form der mehrfach verwendbaren Softwareeinheiten unterschieden. Einzelne Formen sind auf die Mehrfachverwendung unterschiedlicher Arbeitsergebnisse aus dem Softwareentwicklungsprozess ausgelegt. In [FeVe99] werden die folgenden Formen mehrfach verwendbarer Softwareeinheiten beschrieben:

- mehrfach verwendbare Anforderungen
- mehrfach verwendbare Softwarearchitekturen
- mehrfach verwendbare Softwarekomponenten
- mehrfach verwendbare Arbeitsergebnisse (Assets)

Wurden mehrfach verwendbare Softwarearchitekturen für eine spezielle Domäne entwickelt, spricht man von domänenspezifischen Softwarearchitekturen. Die mehrfach verwendbaren Anforderungen, Softwarearchitekturen und Softwarekomponenten sind generische Bausteine zur Bildung von Instanzen. Sie besitzen Konfigurationsmöglichkeiten für die Anpassung an wechselnde Anforderungen und Schnittstellen zur Integration von Instanzen. Der Begriff mehrfach verwendbares Arbeitsergebnis ist weiter gefasst. Es ist die mehrfach verwendbare Form jedes Arbeitsergebnisses, das während der Softwareentwicklung anfällt. Dies kann z.B. ein domänenspezifisches System-Modell, eine domänenspezifische Dokumentation oder Ähnliches sein. Ein mehrfach verwendbares Arbeitsergebnis besitzt ebenfalls geeignete Konfigurationsmöglichkeiten und ggf. Schnittstellen zur Integration [SCK+96, CzEi00].

Die am häufigsten verwendeten Formen sind die domänenspezifische Softwarearchitektur und dazu passende mehrfach verwendbare Softwarekomponenten sowie mehrfach verwendbare Softwarekomponenten, die unabhängig von einer domänenspezifischen Softwarearchitektur eingesetzt werden [FeVe99].

Einsatzgebiete mehrfach verwendbarer Softwareeinheiten

Die mehrfach verwendbaren Softwareeinheiten werden für unterschiedliche Anwendungsbereiche, so genannte Domänen, entwickelt. Dabei unterscheidet man zwischen horizontalen und vertikalen Domänen [SCK+96, CzEi00, MMYA02]. In vertikalen Domänen werden mehrfach verwendbare Softwareeinheiten für Produktlinien zusammengefasst. Eine Produktlinie enthält gleichartige Produkte mit ähnlichen Merkmalen. In horizontalen Domänen werden mehrfach verwendbare Softwareeinheiten über die Grenzen von Produktlinien hinweg eingesetzt. Dieser Fall tritt dann auf, wenn in verschiedenen Produktlinien ähnliche Technologien, Algorithmen oder Methoden Anwendung finden [Diaz01]. Beispiele für die Zusammenhänge zwischen horizontalen und vertikalen Domänen zeigt Abbildung 2.2.

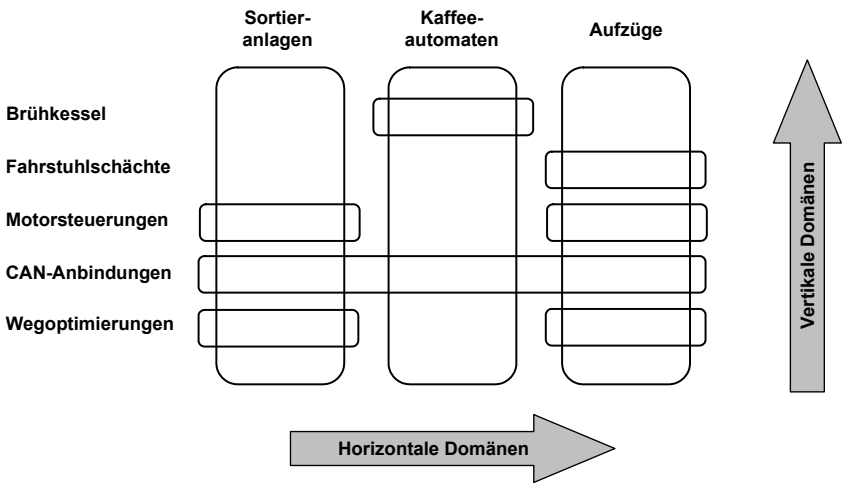


Abbildung 2.2: Beispiele für die Zusammenhänge zwischen horizontalen und vertikalen Domänen

Die Form der mehrfach verwendbaren Softwareeinheit beeinflusst die Flexibilität bzw. die Anpassbarkeit sowie den Umfang der gespeicherten Information. Mehrfach verwendbare Softwarekomponenten sind sehr flexibel und einfach anpassbar. Sie eignen sich gut für den Einsatz in horizontalen Domänen. Mehrfach verwendbare Softwarearchitekturen sind weniger flexibel, enthalten aber Informationen über die Struktur einer zu erstellenden Applikation. Damit eignen sie sich gut für den Einsatz in vertikalen Domänen [CzEi00].

2.2.2 Strukturiertes Vorgehen bei der Entwicklung mehrfach verwendbarer Softwareeinheiten

Zur Darstellung des strukturierten Vorgehens bei der Entwicklung mehrfach verwendbarer Softwareeinheiten wird das Prinzip des Domain-Engineering erläutert und anschließend der Entwicklungsprozess beschrieben, der dieses Prinzip unterstützt. Die Ausführungen in diesem Abschnitt basieren auf [Krue92], [SoSo99], [CzEi00] und [MMYA02], die jeweils eine allgemeine Darstellung von Domain-Engineering enthalten.

Prinzip des Domain-Engineering

Mehrfach verwendbare Softwareeinheiten sollen in zukünftigen Applikationen eingesetzt werden. Das heißt, dass diese Softwareeinheiten auf eine Art und Weise entworfen und implementiert werden müssen, die es erlaubt, sie an bisher noch unbekannte Anforderungen zukünftiger Applikationen anzupassen. Dieser Entwicklung werden beim Domain-Engineering zwei Annahmen zugrunde gelegt. Die eine Annahme besagt, dass sich Merkmale (Features) von Software dann zur Mehrfachverwendung eignen, wenn sie in bestehenden Applikationen bereits häufig eingesetzt wurden. Daraus wird geschlossen, dass sie auch für die Entwicklung zukünftiger Applikationen nützlich sein werden. Die zweite Annahme geht davon aus, dass sich Merkmale, die bei der Entwicklung zukünftiger Applikationen häufig benötigt werden, für die Mehrfachverwendung eignen. Mit Hilfe dieser beiden Annahmen werden aus bestehenden Applikationen und dem zukünftigen Bedarf mehrfach verwendbare Softwareeinheiten entwickelt. Dabei werden die bestehenden Applikationen bzw. Teile davon als Instanzen betrachtet, zu denen ein Typ (z.B. ein generischer Baustein) gefunden werden muss.

Domain-Engineering unterstützt Entwickler bei der Erstellung von mehrfach verwendbaren Softwareeinheiten durch methodisches, strukturiertes Vorgehen. Dieses Vorgehen wird in einem Entwicklungsprozess beschrieben, der Aktivitäten sowie Produkte als Ergebnisse dieser Aktivitäten enthält.

Entwicklungsprozess bei Domain-Engineering

Eine wesentliche Eigenschaft des Domain-Engineering ist die Zweiteilung des Entwicklungsprozesses. Im ersten Teil, dem eigentlichen Domain-Engineering, werden mehrfach verwendbare Softwareeinheiten entwickelt und in einer Domänenbibliothek (Domain

Repository) hinterlegt (development for reuse). Im zweiten Teil, dem Application-Engineering, entstehen mit Hilfe der mehrfach verwendbaren Softwareeinheiten neue Applikationen (development with reuse). Beide Teile dieses zweistufigen Entwicklungsprozesses sind jeweils in drei Aktivitäten untergliedert. In Abbildung 2.3 ist die Zweiteilung des Entwicklungsprozesses dargestellt, wobei in Hinblick auf die Thematik der vorliegenden Arbeit nur für das Domain-Engineering auch die einzelnen Aktivitäten dargestellt sind. Die folgenden Abschnitte bieten einen Überblick über die Aktivitäten des Domain-Engineering. Dies sind im Einzelnen Domänenanalyse, Domänenentwurf und Domänenimplementierung.

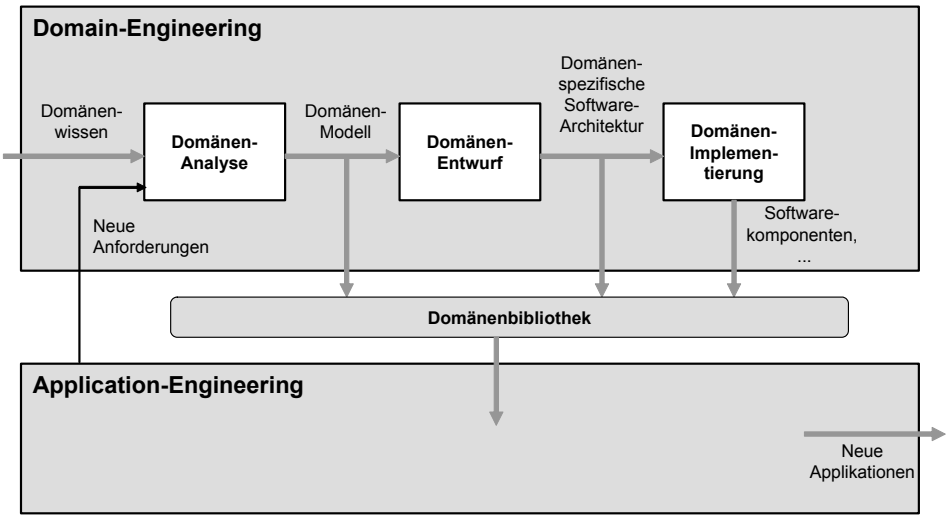


Abbildung 2.3: Zweigeteilter Entwicklungsprozess bei Domain-Engineering

Domänenanalyse

Das Ziel der Aktivität Domänenanalyse ist die Auswahl und Definition einer geeigneten Domäne, für welche die Entwicklung mehrfach verwendbarer Einheiten ökonomisch lohnenswert erscheint, sowie die explizite Darstellung der Variabilitäten, die zwischen den bestehenden Applikationen der ausgewählten Domäne existieren. Die Aktivität ist dazu in zwei Aufgabenbereiche getrennt und beinhaltet Teilaktivitäten zur Auswahl und Definition der Domäne (Domain Scoping) sowie zur Untersuchung der Unterschiede und Gemeinsamkeiten bestehender Applikationen (Domain Modeling).

Zunächst wird eine Domäne ausgewählt und abgegrenzt. Die Auswahl kann sich z.B. auf Geschäftsanalysen oder Risikoanalysen stützen. Es wird z.B. mit Hilfe von Regeln beschrieben, welche Applikationen zur Domäne gehören und welche nicht. Anschließend werden *Interessensgruppen* (Stakeholders) und deren Ziele in Verbindung mit der Domäne identifiziert. Interessensgruppen können firmeninterne Gruppen, wie z.B. Geschäftsleitung, Marketingabteilung, Hardware- sowie Softwareentwickler oder externe Gruppen, wie z.B. Kunden, Standardisierungsgremien und Gesetzgeber sein. Die Ergebnisse der Auswahl und Definition der Domäne sind eine definierte Domäne und identifizierte Interessensgruppen mit unterschiedlichen Zielen hinsichtlich der Domäne.

Vor der Erstellung eines Modells der Domäne (Domain Model) werden Informationen über die Domäne gesammelt. Als Informationsquellen dienen bestehende Applikationen innerhalb der Domäne, Domänen-Experten, Handbücher und Dokumentation, bereits bekannte Anforderungen an zukünftige Applikationen, aktuelle und potenzielle Kunden, Standards, Markt-Studien, etc., die zusammengefasst auch als Domänenwissen (Domain Knowledge) bezeichnet werden. Im Domänenmodell (Domain Model) werden die gemeinsamen und unterschiedlichen Merkmale der Applikationen innerhalb der Domäne explizit beschrieben.

Domänenentwurf

Das Domänenmodell bildet den Ausgangspunkt für den Entwurf auf Domänen-Ebene. Das Ziel des Domänenentwurfs ist der Entwurf mehrfach verwendbarer Softwareeinheiten für die betrachtete Domäne. Die Mehrfachverwendbarkeit dieser Softwareeinheiten wird durch die Abstraktion von Unterschieden und Gemeinsamkeiten bestehender Applikationen zur Bildung generischer Eigenschaften erreicht. Abhängig von der Form der mehrfach verwendbaren Softwareeinheiten, die entstehen sollen, ergeben sich unterschiedliche Tätigkeiten beim Entwurf. Bei mehrfach verwendbaren Softwarekomponenten müssen beispielsweise generische, domänenspezifische Softwarearchitekturen (Domain-specific Software Architecture) entworfen werden, welche die Struktur der Softwarekomponenten beschreiben.

Domänenimplementierung

Bei der Domänenimplementierung werden die Ergebnisse des Domänenentwurfs umgesetzt. Die mehrfach verwendbaren Softwareeinheiten werden realisiert und es entstehen mehrfach verwendbare Softwarekomponenten oder mehrfach verwendbare Arbeitsergebnisse. Außerdem können Werkzeuge zur Anwendung der mehrfach verwendbaren Softwareeinheiten geschaffen werden, welche die Instanziierung und Integration unterstützen.

2.2.3 Darstellung von Arbeitsergebnissen des Domain-Engineering

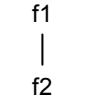
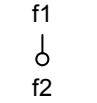
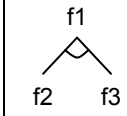
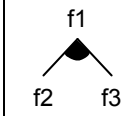
Die Unterschiede und Gemeinsamkeiten innerhalb der Domäne sind in den Arbeitsergebnissen der einzelnen Aktivitäten des Domain-Engineering darzustellen. Im Folgenden werden Möglichkeiten zur Beschreibung des Domänenmodells, der domänenspezifischen Software-architektur und dazu passender mehrfach verwendbarer Softwarekomponenten sowie der Implementierung vorgestellt.

Domänenmodell

Die Ergebnisse der Domänenanalyse sind Unterschiede und Gemeinsamkeiten der bestehenden Applikationen sowie deren Beziehungen zur Domäne. Die Beziehungen zur Domäne zeigen, ob ein Merkmal in allen untersuchten Applikationen der Domäne vorkommt und damit eine Gemeinsamkeit darstellt oder ob es nur in einzelnen Applikationen vorkommt und einen Unterschied beschreibt. Unterschiede werden in optional auftretende Merkmale und alternativ

zueinander vorkommende Merkmale unterteilt. Zur Darstellung der Ergebnisse hat sich das Merkmaldiagramm (*Feature Diagram)* nach [KCH+90] und [CzEi00] durchgesetzt. Tabelle 2.2 gibt einen Überblick über die verschiedenen Möglichkeiten der Beziehungen von Merkmalen sowie deren grafische Darstellung.

Tabelle 2.2: Beziehungen von Merkmalen (Feature) innerhalb einer Domäne und deren Darstellung im Merkmaldiagramm

Beziehung von Merkmalen innerhalb einer Domäne	**Bezeichnung der Beziehung**	**Grafische Darstellung der Beziehung**
Ein Merkmal ist in allen betrachteten Applikationen vorhanden.	vorgeschriebenes Merkmal (mandatory feature)	 f2 ist ein vorgeschriebenes Merkmal von f1
Ein Merkmal ist in einigen der betrachteten Applikationen vorhanden.	optionales Merkmal (optional feature)	 f2 ist ein optionales Merkmal von f1
Genau ein Merkmal aus einer Menge von Merkmalen mit unterschiedlichen Ausprägungen ist in allen betrachteten Applikationen vorhanden.	alternative Merkmale (alternative features)	 f2 und f3 sind alternative Merkmale von f1 (1 aus vielen)
Ein oder mehrere Merkmale aus einer Menge von Merkmalen mit unterschiedlichen Ausprägungen sind in allen betrachteten Applikationen vorhanden.	Oder-Merkmale (or features)	 f2 und f3 sind Oder-Merkmale von f1 (mehrere aus vielen)

Neben den Beziehungen der Merkmale zur Domäne können Abhängigkeiten zwischen den Merkmalen auftreten. Die Möglichkeit ein alternatives Merkmal auszuwählen kann beispielsweise vom Vorhandensein eines anderen, optionalen Merkmals abhängen. Das Merkmaldiagramm bietet Möglichkeiten, die Abhängigkeiten zwischen Merkmalen grafisch darzustellen. Abbildung 2.4 zeigt ein Beispiel für ein Merkmaldiagramm eines Automobils nach [KCH+90]. Es beinhaltet die Möglichkeit, entweder ein manuelles oder ein automatisches Getriebe zu wählen. Die Darstellung der Beziehungen der beiden Getriebeformen zur Domäne lässt es nicht zu, beide Alternativen gleichzeitig zu wählen. Außerdem wird eine Abhängigkeit zwischen Merkmalen dargestellt, die eine Auswahl der optionalen Klimaanlage auf Automobile mit einer Leistung > 50kW beschränkt.

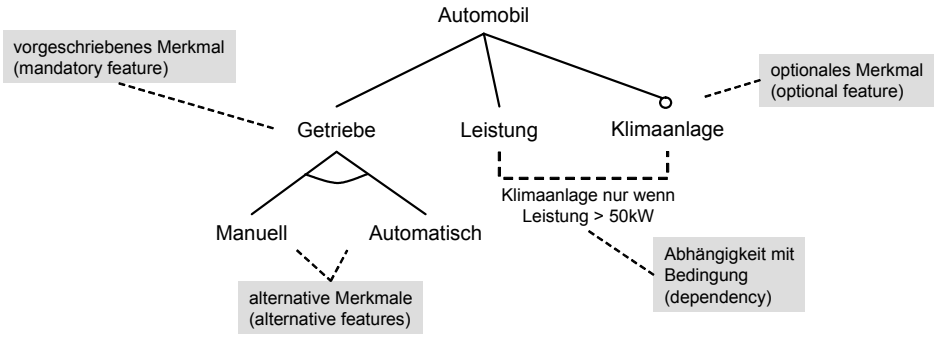


Abbildung 2.4: Beispiel eines Merkmaldiagramms zur Darstellung der Unterschiede und Gemeinsamkeiten von Automobilen

Domänenspezifische Softwarearchitektur und mehrfach verwendbare Softwarekomponenten

Eine Softwarearchitektur ist ein Entwurfs-Modell für eine einzelne Applikation. Sie beschreibt laut Shaw und Garlan [ShGa96], wie eine Applikation in modulare Einheiten zerlegt ist und wie diese miteinander kommunizieren und interagieren. Eine domänenspezifische Softwarearchitektur ist eine generische Softwarearchitektur für eine Domäne. Sie beschreibt, wie Applikationen innerhalb einer Domäne zerlegt sind. Aus dieser domänenspezifischen Softwarearchitektur lassen sich durch Spezialisierung und Anpassung Instanzen von Softwarearchitekturen für einzelne Applikationen der Domäne ableiten [Diaz01]. Betrachtet man Softwarekomponenten als Bausteine für die Entwicklung von Applikationen, entspricht die domänenspezifische Softwarearchitektur dem Bauplan für das Zusammenfügen solcher Bausteine. Es sind Konfigurationsmöglichkeiten notwendig, um eine domänenspezifische Softwarearchitektur bei der Instanziierung anpassen zu können. Dazu werden bei domänenspezifischen Softwarearchitekturen so genannte *Variationspunkte* (Variation Points) eingesetzt. Variationspunkte sind definierte Stellen der domänenspezifischen Softwarearchitektur, an denen alternative oder optionale Softwarekomponenten eingesetzt werden können. Bei alternativen Variationspunkten wird bei der Instanziierung eine passende Softwarekomponente ausgewählt. Es ist zu gewährleisten, dass die als Alternativen zur Verfügung stehenden Softwarekomponenten durch ihren Funktionsumfang und ihre Schnittstellen austauschbar sind. Demgegenüber bieten optionale Variationspunkte die Möglichkeit zu bestimmen, ob eine Softwarekomponente bei der Instanziierung integriert wird oder nicht. Czarnecki und Eisenecker beschreiben in [CzEi00] verschiedene Möglichkeiten zur Darstellung von Variationspunkten in Softwarearchitekturen. Dabei wird berücksichtigt, ob die betroffenen modularen Einheiten optional oder alternativ vorkommen. Es wird auch nach dem Zeitpunkt der Konfiguration, der so genannten *Binding-Time*, unterschieden. Diese bestimmt, ob die Konfiguration vor der Übersetzung der Software (Compile-Time) oder zur Laufzeit (Run-Time) vorgenommen wird. Zur Darstellung von Variationspunkten, die vor der Übersetzung konfiguriert werden, eignen sich für objektorientierte Softwarearchitekturen beispielsweise *Template-Klassen*. Für

Variationspunkte, die während der Laufzeit konfiguriert werden, sind u.a. bedingte Vererbungsmechanismen einsetzbar.

Eine mehrfach verwendbare Softwarekomponente muss nach Göhner [Göhn98] eine in sich abgeschlossene funktionale Einheit mit vollständig spezifizierten Schnittstellen bilden. Sie muss gut dokumentiert, qualitativ hochwertig und unverändert mehrfach verwendbar sein. Zusätzlich soll eine Softwarekomponente unabhängig von anderen Softwarekomponenten an eine gegebene Aufgabenstellung anpassbar und bezüglich bestimmter Eigenschaften konfigurierbar sein. Für den Entwickler einer mehrfach verwendbaren Softwarekomponente ist darüber hinaus ihre innere Softwarearchitektur wichtig. Eine mehrfach verwendbare Softwarekomponente kann modulare Einheiten enthalten, deren Struktur durch eine innere Softwarearchitektur beschrieben wird. Es werden Parameter eingesetzt, um eine mehrfach verwendbare Softwarekomponente bei der Instanziierung anpassen zu können. Mit Parametern lassen sich bei der Konfiguration die Auswahl von Optionen oder Alternativen sowie die individuelle Eingabe von Werten realisieren. Sie beeinflussen die Eigenschaften der Softwarekomponente auf eine während der Entwicklung vordefinierte Weise. Die Parameter können sowohl die interne Softwarearchitektur als auch die Implementierung der Softwarekomponenten beeinflussen. Im Fall der Softwarearchitektur werden, wie bei der domänenspezifischen Softwarearchitektur, mit Hilfe von Variationspunkten geeignete Konfigurationsmöglichkeiten definiert und die vorgestellten Mechanismen zur Umsetzung solcher Variationspunkte eingesetzt. Ist die Implementierung betroffen, sind die Mechanismen für die Konfiguration abhängig von der Programmiersprache bzw. dem eingesetzten Compiler.

Eine einheitliche Notation zur Darstellung von domänenspezifischen Softwarearchitekturen und mehrfach verwendbaren Softwarekomponenten gibt es nicht. Die in der Softwareentwicklung häufig eingesetzte Unified Modeling Langage (UML) [OMG03a] bietet keine explizite Möglichkeit zur Darstellung von Variationspunkten. Es gibt jedoch Untersuchungen von Clauß [Clau01], Gomaa [Goma01], Ziadi et al. [ZHJ03] und Anderen, die Erweiterungsmöglichkeiten der UML, wie beispielsweise *Stereotypes* oder *Tagged Values*, zur Darstellung von Variationspunkten zu nutzen.

Implementierung für die Mehrfachverwendung

Die Möglichkeiten zur Umsetzung von Variationspunkten in der Implementierung hängen sowohl von der eingesetzten Programmiersprache als auch von den verwendeten Werkzeugen wie Präprozessor, Compiler oder Linker ab.

Variationspunkte, die zur Laufzeit konfiguriert werden, können beispielsweise durch bedingte Verzweigungen (if-then-else, switch-case, ...) realisiert werden. Diese Sprachkonstrukte werden von allen Programmiersprachen unterstützt. Mechanismen, wie beispielsweise die Vererbung, stehen dagegen nur in objektorientierten Programmiersprachen zur Verfügung.

Zur Umsetzung von Variationspunkten, die bereits vor der Übersetzung konfiguriert werden, bietet die objektorientierte Programmiersprache C++ [Stro00] beispielsweise die Möglichkeit zur Implementierung von Template-Klassen. Die Sprachkonstrukte werden jedoch nicht von allen Compilern einheitlich interpretiert. Die objektorientierte Programmiersprache Java unterstützt Template-Klassen nicht [CzEi00]. Bei den Programmiersprachen C/C++ kann durch Präprozessor-Anweisungen eine bedingte Übersetzung des Source-Codes erreicht werden [KeRi90]. Diese ermöglichen das Ausblenden und Austauschen beliebiger Teile des Source-Codes vor der Übersetzung. Gesteuert wird die bedingte Übersetzung durch Parameter.

2.3 Auswirkungen bei der Anwendung von Domain-Engineering

Wird Domain-Engineering mit den in Abschnitt 2.2.2 beschriebenen Eigenschaften von einem Unternehmen eingesetzt, hat dies verschiedene Auswirkungen auf das Unternehmen und die von ihm hergestellten Produkte. Davon sind der Entwicklungsaufwand und seine Verteilung über die Zeit sowie auf verschiedene Projekte betroffen. Außerdem führt die Einführung des zweigeteilten Entwicklungsprozesses zu einer Veränderung in der Organisationsstruktur des Unternehmens und hat Folgen für die Softwarearchitektur und die Implementierung der Produkte. Diese drei Punkte werden bezüglich der Auswirkungen bei der Anwendung von Domain-Engineering getrennt untersucht. Die Art des anwendenden Unternehmens wird dabei zunächst nicht berücksichtigt.

2.3.1 Entwicklungsaufwand

Der kausale Zusammenhang zwischen der Entwicklung von mehrfach verwendbaren Softwareeinheiten und deren Anwendung bei der Entwicklung von Applikationen lässt Rückschlüsse auf den zeitlichen Ablauf von Domain-Engineering und Application-Engineering zu. Für die Entwicklung mit Hilfe der Mehrfachverwendung sind mehrfach verwendbare Softwareeinheiten notwendig. Diese müssen vor der Anwendung in Vorleistung entwickelt werden. In Abbildung 2.5 ist dieser Zusammenhang in Form eines Balkenplanes dargestellt.

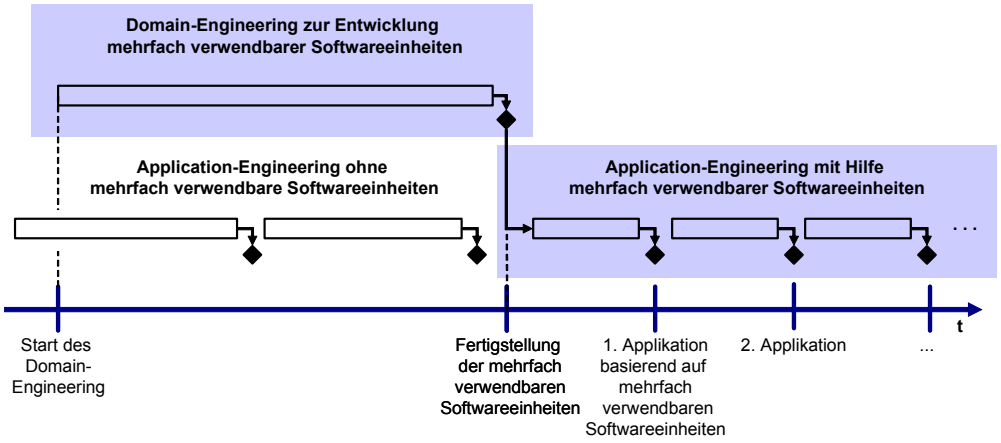


Abbildung 2.5: Zeitliche Zusammenhänge zwischen Domain-Engineering und Application-Engineering

Der in Vorleistung erbrachte Entwicklungsaufwand stellt eine Investition für die anwendende Organisation dar. Dieser Investition stehen Einsparungen bei der Entwicklung jeder Applikation mit Hilfe der mehrfach verwendbaren Softwareeinheiten gegenüber. Durch häufige Anwendung der mehrfach verwendbaren Softwareeinheiten amortisiert sich die getätigte Investition. Ein Return-of-Investment tritt ein, sobald die Summe der Einsparungen den Betrag der Investition erreicht hat. Dieser Zusammenhang wurde in einer Studie der Firma Lucent Technologies an 25 Domänen untersucht [ADH+00]. Dabei wurden für die Entwicklung von mehrfach verwendbaren Softwareeinheiten Investitionskosten ermittelt, die in der Regel unter den Entwicklungskosten von drei ohne Mehrfachverwendung entwickelten Applikationen lagen. Die Entwicklungskosten einer Applikation, die mit Hilfe der Ergebnisse des Domain-Engineering entwickelt wurde, belaufen sich laut der Studie auf etwa ein Viertel der Entwicklungskosten von Applikationen ohne Domain-Engineering. Der Return-of-Investment war nach der Entwicklung von ca. 4 neuen Applikationen unter Zuhilfenahme der entwickelten mehrfach verwendbaren Softwareeinheiten erreicht. Abbildung 2.6 zeigt eine grafische Darstellung dieser Zusammenhänge. In dem gezeigten Diagramm wird der kumulierte Entwicklungsaufwand über die Anzahl der entwickelten Applikationen innerhalb einer Domäne aufgetragen. Um vergleichbare Ergebnisse zu bekommen, wurde der Entwicklungsaufwand auf den durchschnittlichen Entwicklungsaufwand für eine Applikation normiert. Außerdem wird davon ausgegangen, dass bei der Entwicklung ohne Domain-Engineering jede Applikation komplett neu entwickelt wird [ADH+00].

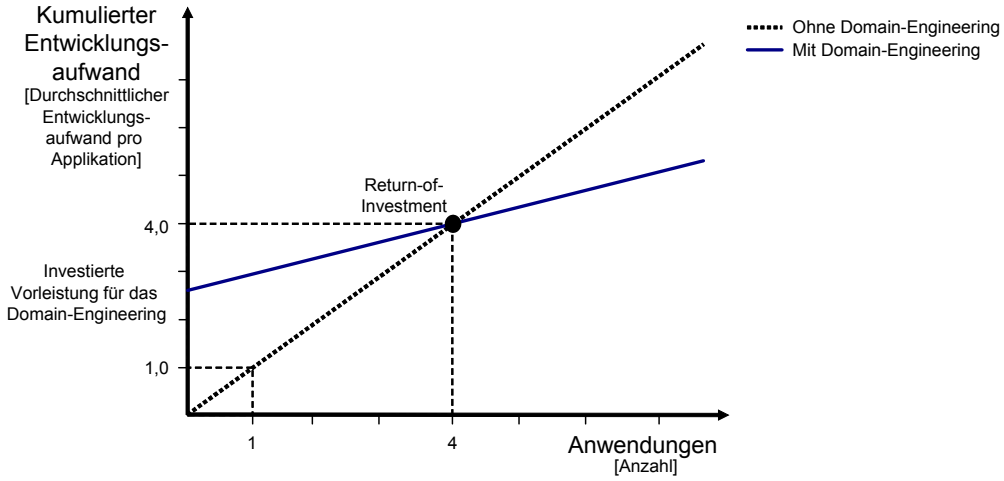


Abbildung 2.6: Abschätzung des Entwicklungsaufwandes mit und ohne Domain-Engineering

Die investierte Vorleistung geht in Form eines Startwertes in die Darstellung ein. Durch den auf 25% reduzierten Entwicklungsaufwand für Applikationen unter Anwendung der Ergebnisse des Domain-Engineering ergibt sich im Vergleich zum Entwicklungsaufwand ohne Domain-Engineering eine geringere Steigung der beiden dargestellten Geraden. Der Schnittpunkt der beiden Geraden bezeichnet den Return-of-Investment. Ab diesem Punkt hat sich die investierte Vorleistung amortisiert.

2.3.2 Organisationsstruktur

Zur Anwendung des zweigeteilten Entwicklungsprozesses, wie er in Abschnitt 2.2.2 vorgestellt wurde, ist nach [MMYA02] die Bildung von zwei oder mehr Teams sinnvoll. Ein Team hat die Aufgabe, mehrfach verwendbare Softwareeinheiten mit Hilfe einer Domain-Engineering-Methode zu erstellen. Nach der Erstellung wird dieses Team mit der Wartung, Pflege und Weiterentwicklung der mehrfach verwendbaren Softwareeinheiten betraut[2]. Mindestens ein weiteres Team erstellt Applikationen auf Basis der mehrfach verwendbaren Softwareeinheiten. Die Entwicklungsteams müssen eng zusammenarbeiten. Dazu sind eine intensive Kommunikation sowie ein abgestimmtes Management notwendig. Die Bildung unterschiedlicher Teams zur Bearbeitung der Entwicklungsprozesse wirkt sich auf die Organisationsstruktur eines Unternehmens aus. Zur Sicherung des Umsatzes wird in der Regel während der Entwicklung der mehrfach verwendbaren Softwareeinheiten die herkömmliche Entwicklung ohne Mehrfachverwendung solange weiter betrieben, bis die mehrfach verwendbaren Softwareeinheiten zur Anwendung bereit stehen. Es entsteht daher durch die Einführung des Domain-Engineering der Bedarf für ein neues Team. Das ist mit entsprechendem Personalbedarf gekoppelt.

[2] Der Aufwand für Wartung, Pflege und Weiterentwicklung sind in der Abschätzung des Entwicklungsaufwandes mit und ohne Domain-Engineering nach [ADH+00] nicht berücksichtigt.

2.3.3 Softwarearchitektur und Implementierung

Wie in Abschnitt 2.3.1 erläutert, werden mehrfach verwendbare Softwareeinheiten in Vorleistung entwickelt und bei der Entwicklung neuer Applikationen eingesetzt. Dabei kommen für die Realisierung der neuen Applikationen je nach Form der eingesetzten Softwareeinheiten eine neue Softwarearchitektur und eine neue Implementierung zum Einsatz. Diese lösen bestehende Softwarearchitekturen und Implementierungen ab. Der Wechsel ist abrupt. Ein fließender Übergang ist nicht vorgesehen.

2.4 Domain-Engineering-Methoden

2.4.1 Historie und Überblick über die Entwicklung der Methoden

Die ursprüngliche Idee für das Domain-Engineering kann bis in die siebziger Jahre zurückverfolgt werden. Parnas schlug 1976 vor, die gemeinsamen Eigenschaften einer Familie von Programmen zu untersuchen, um daraus spezielle Eigenschaften individueller Mitglieder der Familie abzuleiten [Parn76]. Im Jahre 1980 lieferte Neighbours mit der Methode Draco [Neigh80] die erste strukturierte Unterstützung bei dieser Aufgabe. Er verwendet dabei erstmals den Begriff „Domain Analysis“ für die Untersuchung der Unterschiede und Gemeinsamkeiten. Später wurde daraus die Bezeichnung „Domain Engineering“ für die Analyse, den Entwurf und die Implementierung zur Entwicklung mehrfach verwendbarer Softwareeinheiten abgeleitet. Auf Basis der Arbeit von Neighbours entstanden bis heute eine Reihe verschiedener Domain-Engineering-Methoden. Sie besitzen in der Regel die in Abschnitt 2.2.2 beschriebenen Merkmale [Arra94, CzEi00, MMYA02]. Darüber hinaus wurden sie für besondere Einsatzgebiete oder zur Erstellung bestimmter Formen mehrfach verwendbarer Softwareeinheiten angepasst. Einige dieser Methoden werden in den folgenden Abschnitten vorgestellt.

2.4.2 FODA / MBSE

FODA (Feature-oriented Domain Analysis) wurde 1990 am Software Engineering Institute (SEI) der Carnegie Mellon University entwickelt [KCH+90]. Das SEI wurde dabei finanziell durch das Verteidigungsministerium der USA (DoD) unterstützt. Die Methode fand in der Army Movement Control Domain [CSPK92] sowie bei weiteren Militärprojekten [CARDS94, BrCl96] Anwendung. Im zivilen Bereich wurde FODA unter anderem von der Telecom Italia eingesetzt [VAM+98]. 1997 wurde FODA überarbeitet und in die Methode MBSE (Model Based Software Engineering) integriert [MBSE97]. MBSE unterstützt sowohl das Domain-Engineering als auch das Application-Engineering, wobei FODA als wesentlicher Bestandteil des Domain-Engineering übernommen wurde.

Die Methode FODA / MBSE dient der Entwicklung einer domänenspezifischen Softwarearchitektur sowie dazu passender Softwarekomponenten. Die Ergebnisse werden neu entwickelt. Von bestehenden Applikationen werden Merkmale als Anforderungen an die mehrfach verwendbaren Ergebnisse abgeleitet. Merkmale sind bei FODA die Eigenschaften von Applikationen innerhalb einer Domäne, die für Kunden bzw. Benutzer sichtbar sind.

FODA / MBSE folgt im Wesentlichen dem in Abschnitt 2.2.2 dargestellten zweigeteilten Entwicklungsprozess. Darüber hinaus führt FODA die Merkmalmodellierung (Feature Modeling[3]) für die Analyse der Domäne ein. Bei dieser Art der Modellierung werden Unterschiede und Gemeinsamkeiten bestehender Applikationen innerhalb der Domäne identifiziert und in einem Merkmalmodell (Feature Model) beschrieben. Der zentrale Teil eines Merkmalmodells ist das Merkmaldiagramm (Feature Diagram), wie es in Abschnitt 2.2.3 vorgestellt wurde. Neben dem Merkmaldiagramm besteht das Merkmalmodell noch aus einem Informationsmodell (Information Model), einem Ausführungsmodell (Operational Model) und einem Domänenwörterbuch (Domain Dictionary). Diese ergänzen die grafische Darstellung des Merkmaldiagramms mit textuellen Informationen über Beziehungen zwischen den Applikationen sowie den Abhängigkeiten zwischen den Merkmalen [KCH+90].

2.4.3 ODM

Die aktuelle Version 2.0 der Methode ODM (Organization Domain Modeling) entstand 1996 bei der Firma Organon Motives. Unterstützt wurde die Entwicklung durch das DoD. ODM wurde sehr ausführlich in einem Guidebook [SCK+96] beschrieben, das detaillierte Anleitungen zur Anwendung und Anpassung sowie Beispiele enthält. Die Methode wurde in verschiedenen militärischen und zivilen Projekten angepasst und angewendet. Darunter befinden sich beispielsweise das STARS-Projekt des DoD [Simo94, SCK+96], bei dem Firmen wie Lockheed Martin und Logicon beteiligt waren, sowie Projekte bei Hewlett Packard [Corn96] und Rolls-Royce [KLW+96].

Das Ergebnis der Methode ODM sind mehrfach verwendbare Arbeitsergebnisse (Assets). Im Gegensatz zu FODA verwendet ODM eine generellere Definition des Begriffs Merkmal. Nach ODM muss ein Merkmal für einen oder mehrere *Interessensgruppen* (Stakeholder) relevant sein. Die Auswirkungen dieser Definition bei der Analyse von Unterschieden und Gemeinsamkeiten werden in folgendem Beispiel verdeutlicht: Ein Hersteller von Waschmaschinen baut unterschiedliche Wasserpumpen in seine Waschmaschinen vom Typ A und Typ B ein. Da der Typ der Wasserpumpe für den Kunden bzw. Benutzer nicht sichtbar ist, stellt dies nach der Definition von FODA kein Merkmal dar und wird deshalb bei der Analyse der Domäne nicht berücksichtigt. Nach der Definition von ODM stellt der Typ der Wasserpumpe dagegen ein Merkmal dar, da er für mehrere Interessensgruppen, wie z.B. die Abteilung für den Einkauf oder

[3] In Übersetzungen werden Features als Merkmale und Feature Modeling folglich als Merkmalmodellierung, das Feature Diagram als Merkmaldiagramm, usw. bezeichnet.

die Hardwareentwicklung, relevant ist. So können mehr Unterschiede und Gemeinsamkeiten erkannt werden und in die Entwicklung der mehrfach verwendbaren Arbeitsergebnisse einfließen.

Der Entwicklungsprozess ist in ODM sehr ausführlich beschrieben. Die drei Phasen des Domain-Engineering (siehe Abschnitt 2.2.2) werden in Sub-Phasen und diese wiederum in Tasks zerlegt. ODM misst den organisatorischen Aspekten des Domain-Engineering große Bedeutung zu. So werden beispielsweise alle am Projekt beteiligten Personen und Institutionen, so genannte Stakeholders, bezüglich ihrer Rolle im Projekt und ihrer Interessen an den Ergebnissen des Domain-Engineering analysiert. Mit Hilfe einer Zuordnung dieser Stakeholder zu den Arbeitsergebnissen der Methode kann vom Merkmal bis zur mehrfach verwendbaren Softwareeinheit alles auf die Interessen einzelner Stakeholder zurückverfolgt werden.

Die Methode ODM verwendet eine sehr allgemeine Definition des Begriffes Modellierung und kann für die Verwendung mit jeder spezifischen Modellierungstechnik oder Notation angepasst werden. ODM kann sowohl mit Methoden zu objektorientierter Analyse und Design als auch mit Methoden zur strukturierten Softwareentwicklung kombiniert werden. Die Anpassung an seine Bedürfnisse muss der Anwender selbst vornehmen. Eine Anleitung dazu ist in [SCK+96] enthalten.

Eine Besonderheit der Methode ODM ist die Möglichkeit zur Einbeziehung zukünftiger Entwicklungen einer Domäne in Analyse und Entwicklung. Dazu werden neben Aktivitäten, die es erlauben bestehende Merkmale für zukünftige Applikationen anwendbar zu machen, auch Aktivitäten beschrieben, die zukünftige Entwicklungen des Marktes und fortschreitende Entwicklungen neuer Technologien berücksichtigen. Diese, so genannte innovative Modellierung [SCK+96], basiert beispielsweise auf Umfragen bei Kunden über zukünftige Anforderungen oder auf Marktstudien.

2.4.4 DSSA

Die Methode DSSA (Domain-Specific Software Architecture[4]) [TrCo92, Trac95] wurde bei IBM Federal Systems Company entwickelt und vom Office of Naval Research[5] finanziell unterstützt. Fachliche Unterstützung fand das Projekt beim SEI der Carnegie Mellon University. Die erste Veröffentlichung zu DSSA stammt von Tracz und Coglianese aus dem Jahre 1992 [TrCo92]. Die Methode wurde im Rahmen des ADAGE-Projekts[6] unter anderem von Loreal Federal Systems eingesetzt [Trac94].

---

[4] DSSA ist nach der Form der unterstützten Softwareeinheiten benannt. Die Methode beinhaltet einen Entwicklungsprozess Namens „DSSA Process“.

[5] Das Office of Naval Research ist ein gemeinsames Forschungszentrum der US Navy und der US Marines

[6] ADAGE-Projekt (Avionics Domain Application Generation Environment) zur Entwicklung von Software für Hubschrauber

Die Methode DSSA konzentriert sich auf die Entwicklung von domänenspezifischen Softwarearchitekturen. Eine domänenspezifische Softwarearchitektur stellt eine Referenz-Architektur für eine Familie von Applikationen dar. Durch die Fokussierung auf Softwarearchitekturen entstand das Bedürfnis, die Softwarekomponenten einer Softwarearchitektur und deren Beziehungen untereinander formal zu beschreiben. Das führte zur Entwicklung von Architekturbeschreibungssprachen. Als Beispiel ist LILEANA [Trac94] zu nennen. Diese wurde speziell für Softwarearchitekturen mit Merkmalen zur Unterstützung der Programmiersprache Ada entwickelt. Die Firmen Teknowledge und Honeywell, beide Projektpartner im ADAGE-Projekt, entwickelten weitere Architekturbeschreibungssprachen zur Anwendung mit DSSA.

Der Entwicklungsprozess von DSSA folgt im Wesentlichen dem in Abschnitt 2.2.2 vorgestellten zweigeteilten Entwicklungsprozess bestehend aus Domain-Engineering und Application-Engineering.

2.4.5 FAST

Die Methode FAST (Family-oriented Abstraction, Specification and Translation) wurde von Weiss und Lai bei Lucent Technologies entwickelt und 1999 in Form eines Buches veröffentlicht [WeLa99]. Sie fand Anwendung in mehr als 25 Domänen bei Lucent Technologies [ADH+00]. Dabei konnte für Mitglieder der Domäne eine Verringerung der Entwicklungszeit und der Entwicklungskosten auf 60% - 70% erzielt werden.

Bei FAST entstehen eine domänenspezifische Softwarearchitektur, mehrfach verwendbare Softwarekomponenten und eine domänen-spezifische Architekturbeschreibungssprache, die die Dokumentation von Variabilitäten unterstützt.

FAST unterstützt die Zweiteilung des Entwicklungsprozesses gemäß Abschnitt 2.2.2. Das Domain-Engineering wird jedoch nur in die beiden Phasen Domänenanalyse und Domänenimplementierung aufgeteilt. Die für den Domänenentwurf notwendigen Tätigkeiten werden von diesen Phasen abgedeckt. Im Rahmen der Domänenanalyse wird eine so genannte *Commonality-Analysis* zur Untersuchung der Gemeinsamkeiten einer Domäne durchgeführt. Dabei handelt es sich um eine moderierte Gruppendiskussion von Domänen-Experten. Ihr Ziel ist es, ein gemeinsames Verständnis der Domänen-Experten über die Gemeinsamkeiten und Unterschiede der Domäne zu bekommen. In der Domänenimplementierung werden die Ergebnisse der Domänenanalyse umgesetzt.

2.4.6 PuLSE

PuLSE (Product-Line Software Engineering) [DFK98, BFK+99, ABFG00] wurde am Institut für Experimentelles Software Engineering (IESE) der Fraunhofer Gesellschaft entwickelt und durch die Europäische Union und das Bundesministerium für Bildung und Forschung (BMBF)

über das Projekt ITEA-CAFÉ finanziell unterstützt. PuLSE wurde im Software-Variant-Building-Projekt [KMSW00] angewandt, bei dem verschiedene KMU teilnahmen.

Die Methode PuLSE dient der Entwicklung einer domänenspezifischen Softwarearchitektur sowie dazu passender Softwarekomponenten. Die mehrfach verwendbaren Ergebnisse werden neu entwickelt. Von bestehenden Applikationen werden analog zu FODA Merkmale als Anforderungen an die mehrfach verwendbaren Ergebnisse abgeleitet. Der Begriff Merkmal wird jedoch ähnlich allgemein verstanden wie bei ODM.

PuLSE ist eine Methode, die sowohl das Domain-Engineering als auch das Application-Engineering abdeckt. Darüber hinaus bietet sie Unterstützung bei der Pflege der mehrfach verwendbaren Softwareeinheiten. Bei der Entwicklung von PuLSE wurde darauf geachtet, dass die Methode skalierbar und leicht an die Bedürfnisse der Anwender anpassbar ist. Insbesondere die Notation und die Vorgehensweise werden behutsam eingeführt, indem verwendete Notationen und Methoden langsam für das Domain-Engineering angepasst werden. Diese Eigenschaft wurde maßgeblich durch eine Studie über die Anwendung von Produkt-Linien-Konzepten in KMU [KMSW00, MuBa00] beeinflusst, welche das IESE parallel zur Entwicklung von PuLSE durchführte. Die in PuLSE eingesetzten Techniken basieren auf verschiedenen bekannten Domain-Engineering-Methoden. So wurde z.B. für die Analyse der Domäne auf die Methode FAST [WeLa99] aufgebaut. Dabei werden die Anforderungen an eine Familie von Systemen im Rahmen einer Commonality-Analysis ermittelt. Die Entwicklung der domänenspezifischen Softwarearchitektur geschieht inkrementell. Dazu werden zunächst generische Szenarien beschrieben, die anschließend Schritt für Schritt in die domänen-spezifische Softwarearchitektur eingearbeitet werden. Nach der Fertigstellung der domänen-spezifischen Softwarearchitektur erfolgt die Einführung bei der Entwicklung von Applikationen, wie bei den anderen Methoden, abrupt (siehe Abschnitt 2.2.2).

2.4.7 Weitere Methoden

Die in den Abschnitten 2.4.2 bis 2.4.6 aufgezählten Domain-Engineering-Methoden stellen eine Auswahl in der Literatur häufig zitierter Methoden dar. Daneben existieren noch weitere, die im Rahmen dieser Arbeit nicht alle vorgestellt werden können. Die meisten von ihnen folgen dem in Abschnitt 2.2.2 vorgestellten Entwicklungsprozess. Einige der Methoden, wie beispielsweise SYSTHESIS [SPC93], verwenden unterschiedliche Begriffe für die einzelnen Aktivitäten. Teilweise wurden Anpassungen für die Entwicklung spezieller Formen mehrfach verwendbarer Softwareeinheiten oder zur Unterstützung unterschiedlicher Notationen vorgenommen. Die Methode JODA (Joint-Integrated Avionics Working Group Object-oriented Domain Analysis) [Holi93] ist, wenn auch kein Merkmalmodell enthalten ist, der Methode FODA vergleichbar und basiert auf einer objektorientierten Notation. In der Literatur finden sich verschiedene Werke, die versuchen, einen Überblick über die bestehenden Domain-Engineering-Methoden zu geben, und sich gegenseitig ergänzen [FeVe99, CzEi00, MMYA02].

2.4.8 Zusammenfassung von Unterschieden und Gemeinsamkeiten der Methoden

Die in der vorliegenden Arbeit vorgestellten Domain-Engineering-Methoden weisen im Wesentlichen die in Abschnitt 2.2.2 beschriebenen Merkmale auf. Dabei unterscheiden sich die Ansichten über den Begriff Merkmal sowie die Ansätze zur Analyse der Unterschiede und Gemeinsamkeiten von Applikationen innerhalb der Domäne. Die Methoden setzen unterschiedliche Schwerpunkte im Entwicklungsprozess und stellen die entsprechenden Aktivitäten dazu in den Vordergrund. Die Gründe dafür liegen in den unterschiedlichen Ausgangssituationen sowie in den unterschiedlichen Formen mehrfach verwendbarer Softwareeinheiten, die entwickelt werden. So unterscheidet sich beispielsweise das Vorgehen bei der Entwicklung von Softwarekomponenten auf Basis bestehender Applikationen und das Vorgehen bei der Entwicklung einer domänenspezifischen Softwarearchitektur für eine neue Produktlinie. Ferré und Vegas klassifizieren Domain-Engineering-Methoden nach der Form der mehrfach verwendbaren Softwareeinheiten [FeVe99]. Tabelle 2.3 zeigt die wesentlichen Ergebnisse der Klassifikation. Dabei ist anzumerken, dass bei der Entwicklung mehrfach verwendbarer Softwarearchitekturen auch dazu passende Softwarekomponenten entwickelt werden.

Tabelle 2.3: Klassifikation von Domain-Engineering-Methoden nach der Form der zu entwickelnden mehrfach verwendbaren Softwareeinheiten

Form der zu entwickelnden mehrfach verwendbaren Softwareeinheiten	**Domain-Engineering-Methoden**
Mehrfach verwendbare Anforderungen	JODA, SYNTHESIS
Mehrfach verwendbare Softwarekomponenten	Draco
Mehrfach verwendbare Softwarearchitekturen (domänenspezifische Softwarearchitekturen)	FODA / MBSE, DSSA, FAST, PuLSE
Mehrfach verwendbare Arbeitsergebnisse (Assets)	ODM

Die Methoden entstanden in Forschungsabteilungen von Großunternehmen oder in Universitäten und Forschungsinstituten. In allen Fällen standen finanzkräftige Geldgeber hinter den Projekten zur Entwicklung und zur Evaluierung der Domain-Engineering-Methoden. In den USA hat das DoD zusammen mit der Rüstungsindustrie einen entscheidenden Beitrag geleistet. In Europa wurden Domain-Engineering-Projekte von der Europäischen Union sowie von Ministerien der jeweiligen Mitgliedsstaaten finanziert. Die erfolgreiche Anwendung der entwickelten Domain-Engineering-Methoden ist in verschiedene Machbarkeitsstudien [CSPK92, KLW+96, BrCl96, Zalm96, ADH+00] dokumentiert. Die einzige Machbarkeits-

studie, die konkrete Zahlen über Entwicklungsaufwand und Einsparungen nennt, wurde bei Lucent Technologies durchgeführt [ADH+00]. Sie schildert die Anwendung von FAST in 25 Domänen. Die meisten der aufgeführten Studien wurden im Rahmen militärischer Projekte oder in großen Unternehmen durchgeführt. Ausnahmen bilden [KMSW00] und [MuBa00]. Sie betrachten Domain-Engineering-Methoden in KMU. Tabelle 2.4 gibt einen Überblick über die vorgestellten Methoden mit einer Auflistung der Geldgeber sowie der Firmen und Projekte, in denen die Methoden evaluiert wurden.

Tabelle 2.4: Entstehung und Evaluation bestehender Domain-Engineering-Methoden

Methode	**entwickelt von**	**finanziert durch**	**evaluiert bei**
FODA / MBSE	Software Engineering Institute (SEI), Carnegie Mellon University	DoD (Department of Defense, USA)	Army Movement Control Project (DoD), CelsiusTech, Nortel, Telecom Italia
ODM	Organon Motives	DoD	STARS-Projekt (DoD). Lockheed Martin, Hewlett Packard, Rolls-Royce, Logicon
DSSA	IBM Federal Systems Company im Rahmen des DARPA DSSA Programms (unterstützt durch das SEI)	Office of Naval Research (US Navy und US Marines)	IBM Federal Systems Company, Teknowledge, Loral Federal Systems, Honeywell
FAST	Lucent Technologies (Bell Laboratories)	Lucent Technologies (Bell Laboratories)	Lucent Technologies (Bell Laboratories)
PuLSE	Institut für Experimentelles Software Engineering (IESE), Fraunhofer Gesellschaft	Europäische Union, Bundesministerium für Bildung und Forschung	Market Maker Software sowie 6 nicht namentlich genannte KMU

In diesem Kapitel wurde zunächst auf die Ad-hoc-Wiederverwendung ohne den Einsatz von Domain-Engineering eingegangen. Im Anschluss daran erfolgte eine allgemeine Betrachtung des Domain-Engineering. Dabei wurden als Ziele des Domain-Engineering die Entwicklung für die Mehrfachverwendung sowie die Unterstützung von Entwicklern bei den dazu notwendigen Tätigkeiten durch eine strukturierte Vorgehensweise identifiziert. Es folgte die Betrachtung der Auswirkungen bei der Anwendung von Domain-Engineering unter Berücksichtigung der zuvor beschriebenen Merkmale. Anschließend wurden einige spezifische Domain-Engineering-Methoden vorgestellt. Auf Basis dieser Grundlagen wird im nächsten Kapitel die Anwendung von Domain-Engineering in Großunternehmen und in KMU gegenübergestellt sowie der Einsatz zur Entwicklung von Automatisierungssystemen untersucht.

3 Anwendbarkeit von Domain-Engineering-Methoden

In diesem Kapitel werden Eigenschaften von Großunternehmen und KMU gegenübergestellt und für das Domain-Engineering wichtige Aspekte der Unternehmensformen herausgegriffen. Darauf aufbauend wird eine Untersuchung der Anwendbarkeit von Domain-Engineering-Methoden in Großunternehmen sowie in KMU durchgeführt. Außerdem werden Eigenschaften von Automatisierungssystemen dargestellt und die Anwendbarkeit von Domain-Engineering-Methoden für Domänen im Bereich der Automatisierungssysteme diskutiert. Auf Basis der Untersuchungsergebnisse werden spezielle Anforderungen an Domain-Engineering-Methoden für den Einsatz in KMU und die Entwicklung von Automatisierungssystemen abgeleitet.

3.1 Eigenschaften von Unternehmen unterschiedlicher Größenordnungen

Laut der Europäischen Union (EU/EWR-Recht) [EU96] darf ein mittelständisches Unternehmen eine Anzahl von 250 Mitarbeitern und einen Jahresumsatz von 40 Mio. € nicht überschreiten. Deshalb werden in der vorliegenden Arbeit Unternehmen mit mehr Mitarbeitern bzw. einem höheren Jahresumsatz als Großunternehmen bezeichnet. Tabelle 3.1 gibt einen Überblick über die beschriebene Klassifikation.

Tabelle 3.1: Klassifikation von Unternehmen nach Anzahl der Mitarbeiter und des Jahresumsatzes

	Kleines Unternehmen	**Mittelständisches Unternehmen**	**Großunternehmen**
Mitarbeiterzahl	< 50 Personen	< 250 Personen	> 250 Personen
Jahresumsatz	< 7 Mio. €	< 40 Mio. €	> 40 Mio. €

Weitere relevante Kriterien für die Gegenüberstellung von Großunternehmen und KMU werden aus den Zielen und den Eigenschaften des Domain-Engineering nach Abschnitt 2.1 und den Auswirkungen bei der Anwendung von Domain-Engineering laut Abschnitt 2.2.3 abgeleitet:

- Produktpalette: Ausgangspunkt des Domain-Engineering stellen bestehende Applikationen der betrachteten Domäne dar. Eine erfolgreiche Anwendung von Domain-Engineering ist maßgeblich von den Gemeinsamkeiten dieser Applikationen abhängig. Deshalb spielt die Produktpalette, welche alle Applikationen einschließt, die ein Unternehmen entwickelt, für das Domain-Engineering eine wichtige Rolle.

- Finanzsituation: Mit der Einführung von Domain-Engineering sind erhebliche Investitionen verbunden. Deshalb ist die Finanzsituation eines Unternehmens ein wichtiger Indikator für die Möglichkeit, die Einführung von Domain-Engineering zu tragen.
- Organisationsstruktur: Domain-Engineering wird nach einem zweigeteilten Entwicklungsprozess durchgeführt. Diese Zweiteilung beeinflusst die Organisationsstruktur eines Unternehmens und ist damit abhängig von der Flexibilität der Organisationsstruktur.
- Forschungs- und Entwicklungsaktivitäten: Domain-Engineering bringt neue mehrfach verwendbare Software hervor. Das ist mit einer Neuentwicklung gleichzusetzen und damit abhängig vom Umgang eines Unternehmens mit der Forschung und Entwicklung.

Die erläuterten Kriterien dienen im Folgenden der Gegenüberstellung von Großunternehmen und KMU.

Produktpalette

Die Produktpalette eines Unternehmens umfasst alle von ihm angebotenen Produkte. Produkte können zu Produktfamilien oder Anwendungsbereichen zusammengefasst werden. In Großunternehmen wird versucht eine Streuung der Produktpalette zu erreichen. Dadurch entstehen Produktpaletten mit Produkten und Anwendungsbereichen, die sich gegebenenfalls stark unterscheiden. Sie können sich beispielsweise über Anwendungsbereiche wie Unterhaltungselektronik und elektronische Steuergeräte für Kraftfahrzeuge erstrecken. KMU versuchen aufgrund ihrer begrenzten Ressourcen, ihre Produkte in einem kleineren Spektrum zu bündeln. Zum einen ist die Erschließung eines Marktes für Produkte in neuen Anwendungsbereichen aufwändig und riskant. Zum anderen ist die Entwicklungstätigkeit in einem neuen Anwendungsbereich mit zeit- und kostenintensivem Aufbau von Know-how verbunden. Deshalb sind KMU in der Regel höher spezialisiert als Großunternehmen.

Finanzsituation

Die Investition für die Einführung von Domain-Engineering stellt für ein Unternehmen ein finanzielles Risiko dar, das mit der Höhe der Investition steigt. Für zwei Unternehmen, die denselben Betrag investieren, kann das Risiko sehr unterschiedlich sein. In einem Unternehmen mit einem hohen Jahresumsatz ist das Risiko geringer als in einem Unternehmen mit einem niedrigeren Jahresumsatz. In einem Großunternehmen verteilt sich der Jahresumsatz auf viele Anwendungsbereiche innerhalb der Produktpalette. Eine Investition in einen Anwendungsbereich ist - gemessen am Gesamtumsatz - gering. Die Produktpalette von KMU umfasst nur einen bis einige wenige Anwendungsbereiche. Eine Investition in einen Anwendungsbereich ist deshalb – wieder gemessen am Gesamtumsatz - hoch. Sie birgt damit ein höheres Risiko.

Organisationsstruktur

In Unternehmen werden Mitarbeiter in Organisationseinheiten wie Teams, Gruppen oder Abteilungen zusammengefasst. Die Anzahl der dabei entstehenden Hierarchieebenen hängt

unter anderem von der Größe eines Unternehmens ab. Großunternehmen unterhalten in der Regel 5-7 Hierarchieebenen. Dabei ist eine Organisationseinheit für einen oder mehrere Fachbereiche verantwortlich. KMU benötigen in der Regel nur 1-4 Hierarchieebenen und unterteilen diese in Teams und ggf. Abteilungen. Bei kleinen Unternehmen kann sich die Organisationsstruktur auf 2 Hierarchieebenen, z.B. den Geschäftsführer und einige Mitarbeiter, reduzieren.

Forschungs- und Entwicklungsaktivitäten

Die Forschung und Entwicklung in Unternehmen hat zum Ziel, neue Technologien und Methoden zu entwickeln bzw. bestehende für die Entwicklung neuer Produkte nutzbar zu machen. In Großunternehmen werden neue Technologien und Methoden durch Prototypen und Pilotprojekte evaluiert. Dadurch können Auswirkungen auf die Produkte frühzeitig erkannt und Risiken abgeschätzt werden. Sollte sich eine Technologie oder Methode nicht eignen, wird sie nicht in der Produktion eingeführt und die Produkte bleiben davon unbeeinflusst. In KMU übernehmen meist einzelne Mitarbeiter die Einführung neuer Technologien und Methoden. Diese Aufgaben müssen in einem begrenzten zeitlichen Rahmen und parallel zum Tagesgeschäft durchgeführt werden. Prototypen oder Pilotprojekte werden nur bedingt eingesetzt, da diese aufwändig und teuer sind. Neue Technologien und Methoden fließen direkt in die Entwicklung neuer Produkte ein. Stellt sich anschließend heraus, dass diese nicht geeignet sind, kann das schwerwiegende Folgen für das Unternehmen haben.

Zusammenfassend lässt sich sagen, dass der Spielraum von KMU gegenüber Großunternehmen deutlich geringer ist. Das gilt sowohl für die Finanzsituation und die Organisationsstruktur als auch für die Forschungs- und Entwicklungsaktivitäten. Tabelle 3.2 fasst die beschriebenen Eigenschaften der beiden Unternehmensformen zusammen.

Tabelle 3.2: Zusammenfassung der Eigenschaften von Großunternehmen und KMU

	Großunternehmen	**KMU**
Produktpalette	Umfangreich, umfasst verschiedene Anwendungsbereiche	Geringer Umfang, umfasst einen bis einige wenige ähnliche Anwendungsbereiche
Finanzsituation	Mittel verfügbar, Risiko für Investition gering	Wenige Mittel verfügbar, Risiko für Investition hoch
Organisationsstruktur	5-7 Hierarchieebenen	1-4 Hierarchieebenen
Forschungs- und Entwicklungsaktivitäten	Anwendung von Prototyping und Durchführung von Pilotprojekten	Direkte Umsetzung in Produkten, selten Prototyping oder Pilotprojekte

3.2 Domain-Engineering in Großunternehmen

Die Anwendung von Domain-Engineering in Großunternehmen wurde in Studien und Erfahrungsberichten veröffentlicht, die als Grundlage für diese Untersuchung dienen. Dazu kommen Studien aus Militärprojekten, die bezüglich Budget und Mitarbeiterzahl mit Großunternehmen vergleichbar sind. Die Studien und Erfahrungsberichte beschreiben sowohl Ergebnisse aus Pilotprojekten als auch Erfahrungen beim Einsatz in der Produktion. Tabelle 3.3 zeigt eine Übersicht über die in der vorliegenden Arbeit verwendeten Studien und Erfahrungsberichte.

Entwicklungsaufwand

Gemäß Abschnitt 2.2.2 amortisiert sich der in Vorleistung zu erbringende Entwicklungsaufwand für die mehrfach verwendbaren Softwareeinheiten nach ca. 3 bis 4 Anwendungen. Betrachtet man diese Investition vor dem Hintergrund der in Abschnitt 3.1 erläuterten Finanzsituation von Großunternehmen, so wird deutlich, dass sie gemessen am Gesamtumsatz gering ist. Daraus lässt sich schließen, dass Großunternehmen in der Lage sind, die Investitionen für das Domain-Engineering aufzubringen. Dieser Zusammenhang wird auch in [MMYA02] beschrieben sowie durch die in Tabelle 3.3 aufgeführten Studien und Erfahrungsberichte gestützt.

Tabelle 3.3: Studien und Erfahrungsberichte über die Anwendung von Domain-Engineering in Großunternehmen und Militärprojekten

Militärprojekt / Großunternehmen	**Domäne**	**Methode**
STARS [CSPK92]	Army Movement Control	FODA
CelsiusTech [BrCl96]	Steuerungssysteme für Kriegsschiffe und U-Boote inklusive der Steuerung der Waffensysteme und der Kommunikation	FODA
Rolls-Royce [KLW+96]	Steuergeräte zum Starten von Flugzeugtriebwerken	ODM
Hewlett Packard [Corn96]	Steuerungssoftware für Drucker	angepasstes ODM
Telecom Italia [VAM+98]	Service Provisioning Control (SPC)	FODAcom (angepasstes FODA)
Lucent Technologies [ADH+00]	25 unterschiedliche Domänen	FAST

Organisationsstruktur

Die Einführung des zweigeteilten Entwicklungsprozesses erfordert, wie in Abschnitt 2.2.2 beschrieben, mindestens zwei Entwicklungsteams. In Studien und Erfahrungsberichten, die im Rahmen von Pilotprojekten entstanden, wurde das Domain-Engineering von Forschungsbereichen oder Universitäten übernommen. Studien, die bei der Umsetzung realer Projekte durchgeführt wurden, berichten von Umstrukturierungen einzelner Abteilungen und ganzer Unternehmen, wie beispielsweise bei CelsiusTech [BrCl96]. Beide Varianten zeigen, dass Großunternehmen in der Lage sind, ihre Organisationsstruktur an die Anforderungen des zweigeteilten Entwicklungsprozesses anzupassen.

Softwarearchitektur

Laut Abschnitt 2.2.2 ersetzen die mehrfach verwendbaren Softwareeinheiten nach ihrer Entwicklung abrupt die bestehenden Softwarearchitekturen und Implementierungen. Bestehende Softwarearchitekturen werden bei der Entwicklung der domänenspezifischen Softwarearchitektur nicht berücksichtigt. Das damit verbundene Risiko kann in Großunternehmen über die Durchführung von Pilotprojekten und die Entwicklung von Prototypen abgeschätzt werden. Viele der in Tabelle 3.3 aufgeführten Studien und Erfahrungsberichte wurden im Rahmen von Pilotprojekten erarbeitet und stellen für die durchführenden Unternehmen die Entscheidungsgrundlage für die Einführung von Domain-Engineering dar. Die Studien fielen positiv aus und einige der Firmen, wie beispielsweise CelsiusTech, Hewlett Packard und Lucent Technologies, haben die Ergebnisse in ihre Produkte übernommen.

Die in Großunternehmen und im Rahmen von Militärprojekten erstellten Studien und Erfahrungsberichte (vgl. Tabelle 3.3) bescheinigen den entsprechenden Domain-Engineering-Methoden insgesamt eine gute Unterstützung für die untersuchten Domänen. Teilweise werden Anregungen zur Verbesserung oder Optimierung gegeben. Es traten jedoch keine Probleme auf, die den Einsatz der Methoden in Frage stellten. Daraus kann geschlossen werden, dass die betrachteten Domain-Engineering-Methoden für den Einsatz in Großunternehmen geeignet sind. Im folgenden Abschnitt wird untersucht, ob Domain-Engineering auch in KMU anwendbar ist.

3.3 Domain-Engineering in kleinen und mittelständischen Unternehmen

Als Grundlage für die Untersuchung der Anwendbarkeit von Domain-Engineering in KMU dienen ebenfalls Studien und Erfahrungsberichte. Im Vergleich zu Großunternehmen und dem Militär wurden über den Einsatz von Domain-Engineering in KMU nur wenige Studien und Erfahrungsberichte veröffentlicht. Diese sind in Tabelle 3.4 aufgelistet.

Tabelle 3.4: Studien und Erfahrungsberichte über die Anwendung von Domain-Engineering in KMU

Projekt / KMU	**Domäne**	**Methode**
DOOM-Projekt / RESICO [VSG+00]	Management von Ferienhäusern (Rest Houses)	FODA
Software-Variant-Building-Projekt [KMSW00]	Domänen in 6 verschiedenen KMU	PuLSE

Entwicklungsaufwand

Der Entwicklungsaufwand für die Erstellung mehrfach verwendbarer Softwareeinheiten ist laut Abschnitt 2.3.1 im Voraus zu erbringen. Aufgrund des in Abschnitt 3.1 beschriebenen geringen finanziellen Spielraums bedeutet dies für KMU ein großes Risiko. In [KMSW00] wird von einer Studie berichtet, bei der von den sechs teilnehmenden KMU erwartet wird, einen Beitrag von je 18 Personenmonaten (PM) Entwicklungsaufwand verteilt über 2,5 Jahre zu leisten. Der Rest sollte von Beratern erbracht werden. Die teilnehmenden KMU beschäftigten zwischen 2 und 11 Softwareentwickler. Mit den erwähnten 18 PM über 2,5 Jahre errechnet sich ein Anteil von bis zu 30 % der gesamten Entwicklungskapazitäten der teilnehmenden KMU. Diese Ressourcen stehen für das Tagesgeschäft nicht mehr zur Verfügung und verringern bis zur Fertigstellung des Projekts die Produktivität. Es ist deshalb nicht verwunderlich, dass in [KMSW00] von Problemen mit den Ressourcen der KMU berichtet wird.

Organisationsstruktur

Domain-Engineering und der zweigeteilte Entwicklungsprozess wirken sich auf den gesamten Softwareentwicklungsprozess von KMU und die Organisation des Unternehmens als Ganzes aus. Zu viele Änderungen im Softwareentwicklungsprozess gleichzeitig vorzunehmen, ist mit einem hohen Risiko verbunden. Eine erfolgreiche Einführung von Domain-Engineering bedingt eine akkurate Planung, Schulung und organisatorische Unterstützung von außen [VSG+00]. KMU sind aufgrund ihrer begrenzten Ressourcen weniger flexibel bei der Umgestaltung ihrer Organisationsstruktur. Das Entwicklungsteam für das Domain-Engineering muss überwiegend auf Basis des bestehenden Mitarbeiterstamms aufgebaut werden. Da parallel zum Domain-Engineering laufende Projekte bearbeitet werden müssen, sind die freien Ressourcen entsprechend knapp und die Einstellung neuer Mitarbeiter ist nur bedingt möglich. Das macht den Aufbau eines Entwicklungsteams für das Domain-Engineering schwierig bis unmöglich.

Softwarearchitektur

In Abschnitt 2.3.1 wurde beschrieben, dass neu entwickelte, mehrfach verwendbare Softwareeinheiten die bestehenden Softwarearchitekturen und Implementierungen nicht berücksichtigen

und diese abrupt ablösen. Die bestehenden Softwarearchitekturen und Implementierungen stellen für KMU einen besonderen Wert dar, da darin ihr Know-how und ihre Erfahrung gespeichert sind. Die Ablösung einer bewährten Softwarearchitektur und einer funktionierenden Implementierung stellt deshalb ein hohes Risiko dar. Durch die Einführung neuer Technologien, Methoden oder Werkzeuge wird dieses Risiko noch erhöht. Da KMU nicht die Ressourcen besitzen, um Pilotprojekte oder Prototypen durchzuführen, ist eine Abschätzung des Risikos sehr schwierig. Nur äußere Einflüsse wie beispielsweise neue Anforderungen, Konkurrenzdruck oder die auslaufende Unterstützung eingesetzter Technologien, Methoden oder Werkzeuge, können KMU dazu bewegen, dieses Risiko einzugehen.

Die zusammen mit KMU erstellten Studien und Erfahrungsberichte (vgl. Tabelle 3.4) beschreiben teilweise erfolgreiche Anwendungen. Es wird von Problemen berichtet, die durch die begrenzten finanziellen und personellen Möglichkeiten entstehen. Das lässt die Schlussfolgerung zu, dass die betrachteten Domain-Engineering-Methoden für den Einsatz in KMU in ihrer vorliegenden Form nicht geeignet sind. Dem gegenüber steht ein großes Interesse der KMU an der Nutzung von Domain-Engineering, das im Wesentlichen auf die Vorteile beim Einsatz mehrfach verwendbarer Softwareeinheiten zurückzuführen ist.

3.4 Eigenschaften von Automatisierungssystemen

Prozessautomatisierung

Nach [LaGö99a, Göhn05a] ist die Prozessautomatisierung die Automatisierung technischer Prozesse. Ein Prozess ist die Gesamtheit von aufeinander einwirkenden Vorgängen in einem System, durch die Materie, Energie oder Information umgeformt, transportiert oder gespeichert wird. Ein technischer Prozess ist ein Prozess, dessen physikalische Größen mit technischen Mitteln erfasst und beeinflusst werden. Bei den Vorgängen in technischen Prozessen wird unterschieden zwischen:

- kontinuierlichen bzw. dynamischen Vorgängen: Vorgänge, bei denen zeitabhängige, kontinuierliche Prozessgrößen auftreten.
- sequenziellen Vorgängen oder Folgevorgängen: Vorgänge, bei denen Folgen von verschiedenen, unterscheidbaren Prozesszuständen auftreten.
- objektbezogenen Vorgängen oder Stück(gut)vorgängen: Vorgänge, bei denen einzelne, identifizierbare Objekte umgeformt, transportiert oder gespeichert werden.

Eine klare Unterscheidung zwischen diesen Vorgängen ist nicht immer möglich. Der Transport von Werkstücken zwischen zwei Bearbeitungsstationen ist beispielsweise ein objektbezogener Vorgang. Bei der Regelung der Geschwindigkeit des eingesetzten Transportbandes handelt es sich um einen kontinuierlichen Vorgang.

Prozessautomatisierungssysteme

Ein Prozessautomatisierungssystem beinhaltet drei miteinander gekoppelte Arten von Systemen: Ein technisches System, ein Rechner- und Kommunikationssystem sowie das Prozessbedienpersonal [LaGö99a, Göhn05a]. Abbildung 3.1 zeigt den Aufbau eines Prozessautomatisierungssystems.

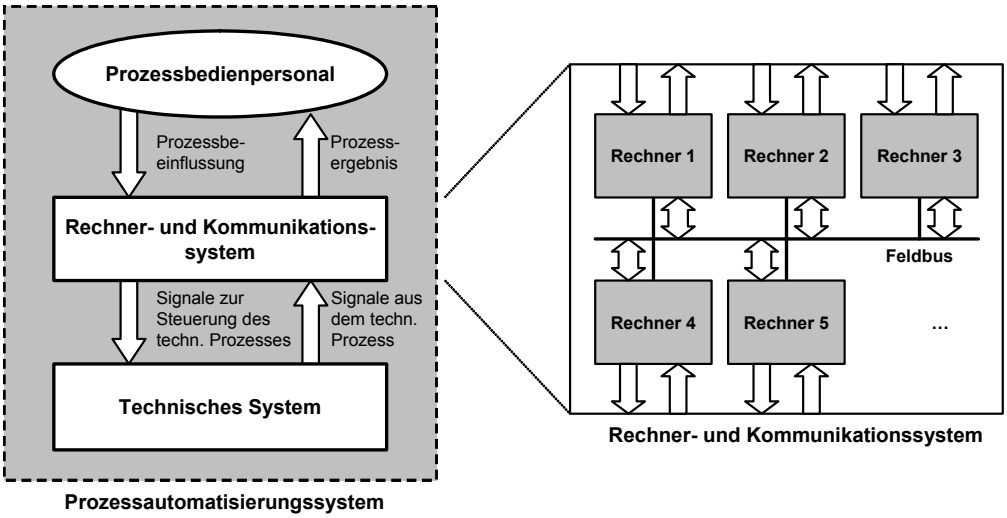


Abbildung 3.1: Aufbau eines Prozessautomatisierungssystems

Technisches System

Bei einem technischen System handelt es sich um ein Gerät, eine Maschine oder eine Anlage, auf der ein technischer Prozess abläuft. Ein chemischer Reaktor ist ein Beispiel für ein technisches System auf dem ein technischer Prozess abläuft, der aus den Teilvorgängen „Befüllung“, „Reaktion“ und „Entleerung“ bestehen kann.

Rechner- und Kommunikationssystem

In einem Rechner- und Kommunikationssystem laufen Informationsprozesse ab. Es beinhaltet beispielsweise Automatisierungscomputer und Bussysteme für die Vernetzung. Es besitzt Schnittstellen zum Prozessbedienpersonal (Prozessbeeinflussung und Prozessergebnis) und zum technischen System (Signale zur Steuerung des technischen Prozesses und Signale aus dem technischen Prozess).

Automatisierungscomputer sind frei programmierbare Digitalrechner. Sie müssen in der Lage sein, Echtzeitbetrieb-Anforderungen zu erfüllen. Das beinhaltet die zeitgerechte Erfassung, Verarbeitung und Ausgabe von Prozessdaten. Außerdem müssen Möglichkeiten zur Ein- und Ausgabe von Prozesssignalen zur Prozessankopplung bestehen.

An der Schnittstelle zum technischen System kommen Sensoren und Aktoren zum Einsatz. Sensoren dienen der Erfassung von Prozessgrößen und deren Umwandlung in eine für Automatisierungscomputer lesbare Form (Prozesssignaleingabe). Aktoren dienen der Umsetzung der

vom Automatisierungscomputer ausgegebenen Informationen in Stelleingriffe im technischen Prozess (Prozesssignalausgabe).

Kommunikationssysteme verbinden sowohl Sensoren und Aktoren mit Automatisierungscomputern als auch Automatisierungscomputer untereinander. Bei einfachen Systemen mit wenigen Sensoren und Aktoren sowie kurzen Leitungen wird ggf. direkt über Leitungsbündel kommuniziert. Feldbussysteme und Netzwerke sind notwendig, wenn viele Sensoren und Aktoren bzw. viele Automatisierungscomputer weit verteilt sind. Es wird unterschieden zwischen Kommunikation auf Feldebene, auf Prozessebene sowie auf Betriebsebene. Auf den verschiedenen Ebenen werden unterschiedliche Anforderungen bezüglich Reaktionszeit und zu übertragender Datenmenge an die eingesetzten Feldbussysteme und Netzwerke gestellt.

Auf den Automatisierungscomputern kommt Automatisierungssoftware zum Einsatz. Sie muss so erstellt sein, dass bei der Datenverarbeitung im Automatisierungscomputer die zeitlichen Anforderungen an die Erfassung der Eingabedaten, an die Verarbeitung im Automatisierungscomputer und an die Ausgabe der Ausgabedaten erfüllt werden. Die zeitlichen Anforderungen sind abhängig von den zeitlichen Abläufen im technischen Prozess.

Prozessbedienpersonal

Das Prozessbedienpersonal überwacht, leitet und bedient den technischen Prozess. Außerdem muss es in Ausnahmesituationen ggf. eingreifen. Zur Überwachung benötigt das Prozessbedienpersonal Informationen über das Prozessergebnis, die auf Anzeigen und Instrumenten dargestellt werden. Zur Bedienung sind Steuer- und Regelgeräte notwendig. Für das Eingreifen in Ausnahmesituationen sind ggf. Handstellgeräte und Not-Aus-Schalter erforderlich.

Bestandteile von Automatisierungssoftwaresystemen

Bei einem Automatisierungssoftwaresystem handelt es sich um die Menge aller Programme, die zur Ausführung einer Automatisierungsaufgabe erforderlich sind. Die Automatisierungsaufgabe lässt sich aus den Aufgaben der Prozessautomatisierung ableiten. Die Grundaufgaben der Prozessautomatisierung sind die Prozessführung und die Prozessüberwachung. Als Prozessüberwachung wird die Überwachung des regulären Prozessablaufs, das Anzeigen des Prozesszustandes, die Diagnose von möglichen Ursachen eines irregulären Betriebsablaufs und die Prozesssicherung bezeichnet. Unter Prozessführung wird die Beeinflussung von Energie- und Massenströmen zum wirtschaftlichen Erreichen eines Prozessergebnisses unter Einhaltung von Randbedingungen, die Steuerung und Regelung einzelner Prozessgrößen und gesamter technischer Anlagen sowie die operative Produktionsführung verstanden.

Die Programme von Automatisierungssoftwaresystemen werden ausführenden Aufgabenbereichen oder organisatorischen bzw. verwaltenden Aufgabenbereichen zugeordnet. Die ausführenden Aufgabenbereiche werden durch so genannte Anwendungssoftware abgedeckt. Dazu gehören z.B. Programme zum Einlesen von Signalen aus dem technischen Prozess sowie

Programme für die Prozessführung und -überwachung. Die organisatorischen und verwaltenden Aufgabenbereiche werden durch die Systemsoftware abgedeckt. Diese umfasst z.B. das Betriebssystem und Treiberprogramme.

Zeitliches Verhalten von Automatisierungssoftwaresystemen

Die zeitlichen Abläufe in einem Automatisierungssoftwaresystem sind mit den zeitlichen Abläufen im technischen Prozess zu synchronisieren. Deshalb ist bei der Erstellung von Programmen für Automatisierungssoftwaresysteme darauf zu achten, dass bei der Datenverarbeitung im Computer die zeitlichen Anforderungen an die Erfassung der Eingabedaten, an die Verarbeitung im Computer und an die Ausgabe der Ausgabedaten erfüllt werden (Echtzeitsystem) [Göhn05a].

Programmierung von Automatisierungssoftwaresystemen

Automatisierungssoftwaresysteme werden analog zu allgemeinen Softwaresystemen durch eine Softwarearchitektur beschrieben, die aus modularen Einheiten sowie deren Beziehungen untereinander besteht [ShGa96] (vgl. Abschnitt 2.2.3). Beispiele für modulare Einheiten sind Funktionen und Prozeduren, abstrakte Datenobjekte, Klassen, Module, usw. [Göhn04].

Automatisierungssoftwaresysteme sind – in Bezug auf die Art der Datenverarbeitung – stets Echtzeitsysteme. Bei der Echtzeitprogrammierung sind zwei Arten von Anforderungen an das zeitliche Verhalten der Datenverarbeitung zu unterscheiden: die Forderung nach Rechtzeitigkeit und die Forderung nach Gleichzeitigkeit [LaGö99a]. Im Allgemeinen lassen sich diese Forderungen durch den Einsatz hinreichend vieler, parallel arbeitender Ein-/Ausgabeeinheiten und durch die nebenläufige Bearbeitung von Rechenprozessen (von einem Echtzeitbetriebssystem gesteuerte Vorgänge der Abarbeitung sequenzieller Programme) erfüllen, falls die Verarbeitungszeiten im Vergleich zu den Zeitabläufen im technischen Prozess klein sind. Zeitbedingungen werden üblicherweise in Form von Einplanungen für die Aktivierung, Ausführung und Beendigung von nebenläufigen Rechenprozessen angegeben [LaGö99b].

Das zeitliche Verhalten von Automatisierungssoftwaresystemen kann vor oder während ihrer Ausführung geplant werden. Die Planung des zeitlichen Verhaltens zyklisch auszuführender Rechenprozesse vor der Ausführung wird synchrone Programmierung genannt. Dabei werden die zyklisch auszuführenden Rechenprozesse über ein Zeitraster synchronisiert. Die Reihenfolge des Ablaufs wird fest vorgegeben. Die Planung des zeitlichen Verhaltens durch ein Organisationsprogramm (Echtzeitbetriebssystem), das während des Ablaufs der Rechenprozesse den zeitlichen Aufruf steuert, wird asynchrone Programmierung genannt. Der Aufruf der Rechenprozesse erfolgt dann, wenn bestimmte Zeitbedingungen (z.B. verbleibende Zeit bis zur Deadline eines Rechenprozesses) erfüllt sind.

Zwischen Rechenprozessen bestehen logische Abhängigkeiten aufgrund der Vorgänge im technischen Prozess (Synchronität zwischen Rechenprozessen und technischem Prozess) sowie

Abhängigkeiten aufgrund der gemeinsamen Benutzung von Betriebsmitteln. Durch die Abhängigkeiten können Verklemmungen oder permanente Blockierungen entstehen, was eine zeitliche Koordinierung der Rechenprozesse notwendig macht. Dies kann beispielsweise durch eine logische Synchronisierung oder eine betriebsmittelorientierte Synchronisierung erfolgen. Bei der logischen Synchronisierung wird der Ablauf der Rechenprozesse an den Ablauf der Vorgänge im technischen Prozess angepasst (Reihenfolge von Aktionen, vorgegebene Zeitpunkte oder Zeitabstände, Reaktion auf Ereignisse). Die betriebsmittelorientierte Synchronisierung erfolgt durch die Einhaltung von Bedingungen bezüglich der Verwendung gemeinsam benutzter Betriebsmittel (Ressourcen). Zur Synchronisierung von Rechenprozessen werden beispielsweise Semaphoren oder das Rendez-vouz-Konzept eingesetzt.

Der Austausch von Daten zwischen Rechenprozessen kann über gemeinsam genutzten Speicher oder über das Versenden von Nachrichten erfolgen.

Beispiele für Automatisierungssoftwaresysteme

Zur Automatisierung von Industrieanlagen, wie beispielsweise verfahrenstechnischen Anlagen, elektrischen Anlagen oder fertigungstechnischen Anlagen, werden häufig speicherprogrammierbare Steuerungen (SPS) als Automatisierungscomputer eingesetzt. SPS werden synchron programmiert [LaGö99a]. Die Software für SPS wird in der Regel in Funktionsbausteine zerlegt, mit denen Funktionen sowie Zugriffe auf das technische System und die Bedien- und Anzeigegeräte realisiert werden.

Bei der Produktautomatisierung, wie beispielsweise bei Waschmaschinen, kommen dagegen meist Mikrocontroller als Automatisierungscomputer zum Einsatz. Sie können sowohl synchron als auch asynchron programmiert werden. Im Bereich der Automobiltechnik werden beide Arten der Programmierung angewandt. Für sicherheitskritische Anwendungen, wie beispielsweise der Automatisierung eines Bremssystems für Automobile ohne mechanische Rückfallebene (Brake-by-Wire), eignet sich die synchrone Programmierung und die Zerlegung in synchrone Softwarekomponenten [GuNä99]. Bei Anwendungen im Komfortbereich eines Automobils (Fensterheber, Scheibenwischer, usw.) wird in der Regel asynchrone Programmierung eingesetzt. Die Strukturierung der Softwarearchitektur kann auf oberster Abstraktionsebene mit Hilfe von Schichten durchgeführt werden. Die Verfeinerung der Struktur kann auf Basis von modularen Einheiten erfolgen. Dieses Vorgehen wird von verschiedenen Standardisierungsprojekten für Software im Automobil, wie beispielsweise ITEA EAST Embedded Electronic Architecture [ITEA04] oder Autosar [Auto05], vorgeschlagen.

Entwicklung von Automatisierungssoftwaresystemen

Das grundsätzliche Vorgehen bei der Entwicklung von Automatisierungssoftwaresystemen unterscheidet vier Entwicklungsphasen: die Anforderungsdefinition, die fachtechnische Lösungskonzeption, den Systementwurf und die Implementierung [Göhn05b]. Bei der

Anforderungsdefinition werden Anforderungen aus dem Blickwinkel der Benutzer (Auftraggeber) beschrieben. Dabei werden die Ziele des Projekts festgelegt sowie die Aufgabenstellung geklärt. Ggf. werden Untersuchungen über die technische und ökonomische Durchführbarkeit des Projekts angestellt. Bei der fachtechnischen Lösungskonzeption wird ein Fachkonzept aus dem Blickwinkel der Technologiefachleute aufgestellt. Dabei erfolgt eine Analyse der Anforderungen sowie des zu automatisierenden technischen Prozesses. Auf Basis dieser Analyse werden Lösungsverfahren (Fachkonzepte) für die gestellten Automatisierungsaufgaben konzipiert. Beim Systementwurf entsteht der Entwurf eines Software-Hardware-System aus dem Blickwinkel der Rechnerfachleute. Dabei werden die Form und die Struktur der Lösung erarbeitet. Es entstehen Beschreibungen der parallel ablauffähigen Vorgänge, der notwendigen Synchronisierung, der Beziehungen der Vorgänge untereinander (Schnittstellen und Daten) und der Beziehungen zur Umwelt (externe Ereignisse, externe Schnittstellen). Außerdem werden die einzusetzenden Geräte und Rechner beschrieben. Bei der Softwareimplementierung erfolgt die Umsetzung des Softwareentwurfs in ein bzw. mehrere Programme. Dabei werden beispielsweise parallel ablauffähige Vorgänge auf Rechenprozesse abgebildet.

3.5 Domain-Engineering für Automatisierungssysteme

Domain-Engineering für Automatisierungssysteme konzentriert sich im Wesentlichen auf den Softwareanteil der Systeme und berücksichtigt die Hardware als Randbedingung. Daher wird im Folgenden der Begriff Automatisierungssystem im Kontext des Domain-Engineering synonym zu Automatisierungssoftwaresystem verwendet.

Beim Domain-Engineering werden bestehende Softwaresysteme bezüglich ihrer Unterschiede und Gemeinsamkeiten analysiert. Werden dabei nicht alle Unterschiede erkannt, besteht das Risiko, dass die Mehrfachverwendbarkeit der Ergebnisse des Domain-Engineering beeinträchtigt wird. Deshalb sind für die Analyse der Unterschiede und Gemeinsamkeiten alle Informationen relevant, mit deren Hilfe Unterschiede aufgedeckt werden können.

Unterschiede von Automatisierungssystemen

Durch Unterschiede im zu automatisierenden technischen Prozess können sich Unterschiede in der Automatisierungsaufgabe sowie in den Einrichtungen, die zur Automatisierung erforderlich sind, ergeben. Das wiederum kann Unterschiede in den Automatisierungssystemen verursachen.

Die Automatisierungsaufgabe kann sich beispielsweise im regulären Prozessablauf, in den zu beeinflussenden Energie- und Massenströmen oder den zu steuernden bzw. zu regelnden Prozessgrößen und technischen Anlagen unterscheiden. Bestehen Unterschiede bezüglich der zeitlichen Abläufe im technischen Prozess, können sich daraus Unterschiede in den zeitlichen Anforderungen an Automatisierungssysteme ergeben. Das wiederum kann sich beispielsweise auf die Einplanung von Rechenprozessen auswirken.

Unterschiede in den Einrichtungen, die zur Automatisierung erforderlich sind, bedeuten Unterschiede im Umfeld der Automatisierungssysteme. Das wird beispielsweise in der eingesetzten Rechnerhardware und Prozessperipherie, den Sensoren und Aktoren, dem Kommunikationssystem, den festverdrahteten Einzelgeräten sowie den Einrichtungen für die Mensch-Prozess-Kommunikation deutlich. Das wiederum kann bei Automatisierungssystemen Unterschiede bei der Prozessgrößenerfassung verursachen. Folgende Aufzählung zeigt Beispiele für Unterschiede innerhalb einer Domäne:

- Heizungsregelung
    - Die Heizungsregelung für eine Wohnung mit 3, 4 oder 5 Zimmern soll automatisiert werden.
    - Die Wärmezufuhr in die Zimmer erfolgt über Heizkörper an der Wand oder im Fußboden bzw. über ein Lüftungssystem.
    - Als Rechnerhardware wird ein Mikrocontroller des Typs HC12, V850 oder C167 eingesetzt (Speichergröße und -Aufteilung, Taktzyklus, Anzahl digitaler und analoger Ein- / Ausgänge, externe Datenbusse, CAN-Bus-Schnittstelle, ...).
    - Zur Erfassung der Umgebungstemperatur werden Temperatursensoren des Typs NTC oder PTC eingesetzt.
- Stückgutsortierung
    - Die Sortierung soll nach einer Codierung (z.B. Strichcode) oder nach Eigenschaften (z.B. Farbe, Größe, usw.) des Stückguts erfolgen.
    - Die Sortierung soll nach 3, 4 oder 10 Klassen vorgenommen werden.
    - Zum Sortieren von Stückgut wird ein Hochregallager, ein Sortierband oder ein X-Achs-Portal eingesetzt.
    - Die Transportbänder für Stückgut unterscheiden sich in ihrer Länge und ihrer Geschwindigkeit.

Unterschiede in der Automatisierungsaufgabe oder in den Einrichtungen zur Automatisierung können Unterschiede im zeitlichen Verhalten der Automatisierungssysteme hervorrufen. Das kann sich beispielsweise auf die Zeitpunkte und die Art der Einplanung von Rechenprozessen oder die geforderten Reaktionszeiten auswirken. Bei der Anwendung von Domain-Engineering für Automatisierungssysteme sind deshalb insbesondere Informationen über bestehende Automatisierungssysteme relevant, die Aufschluss über die genannten Unterschiede geben.

Anwendbarkeit von Domain-Engineering für Automatisierungssysteme

Einige der in Abschnitt 3.2 vorgestellten Studien über die Anwendung von Domain-Engineering-Methoden (vgl. Tabelle 3.3) beschreiben deren Einsatz für die Entwicklung von Automatisierungssystemen. Die Studie [BrCl96] berichtet über einen erfolgreichen Einsatz von FODA zur Entwicklung von Steuerungssoftware für Kriegsschiffe und U-Boote und damit für komplexe, verteilte Automatisierungssysteme. Sie geht jedoch nicht näher auf die Behandlung der speziellen Anforderungen solcher Systeme ein. In [KLW+96] wird der Einsatz von ODM

bei der Entwicklung von Starter-Systemen für Flugzeugtriebwerke beschrieben. Dabei wird auf die eingeschränkten Ressourcen in den betrachteten Automatisierungssystemen eingegangen. Es wird darauf hingewiesen, dass die Konfiguration von mehrfach verwendbaren Softwareeinheiten zur Entwicklungszeit zu geschehen hat, da für eine Konfiguration zur Laufzeit auf dem Zielsystem nicht ausreichend Speicher zur Verfügung steht. Díaz-Herrera untersuchte den Einsatz von Domain-Engineering für Software in eingebetteten Systemen, die eine spezielle Form der Prozessautomatisierungssysteme darstellen [DiMa00, DiGu01]. Die Ergebnisse decken sich mit den Erfahrungen der bereits erwähnten Studien. Díaz-Herrera weist zusätzlich auf eine mangelnde Unterstützung durch Notationen hin, die in der Lage sind, verschiedene Abstraktionsebenen sowie verschiedene Sichten auf den Entwurf von Automatisierungssystemen darzustellen. Die genannten Beispiele lassen den Schluss zu, dass sich Domain-Engineering auch auf Automatisierungssysteme anwenden lässt. Die Domain-Engineering-Methoden bieten jedoch keine explizite Unterstützung für die spezifischen Eigenschaften solcher Software.

Bei Domain-Engineering werden zunächst bestehende Applikationen auf Unterschiede und Gemeinsamkeiten untersucht. (vgl. Abschnitt 2.2). Dabei spielt die Methode zur Analyse der Unterschiede und Gemeinsamkeiten eine wesentliche Rolle. Sie beeinflusst, wo nach Unterschieden und Gemeinsamkeiten gesucht wird, und was als unterschiedlich oder gemeinsam erkannt wird. Die Methode FODA betrachtet bei der Analyse der Domäne nur Merkmale, die für Benutzer bzw. Kunden sichtbar sind (vgl. Abschnitt 2.4.2). Doch viele Eigenschaften der Automatisierungsaufgaben, der Rechnerhardware oder der Prozessperipherie sowie der Echtzeitsoftware bleiben den Benutzern und Kunden verborgen. Damit können bei der Analyse mit der in FODA eingesetzten Definition des Begriffes Merkmal wichtige Eigenschaften von Automatisierungssystemen unentdeckt bleiben (z.B. Prozesssignalerfassung für NTC- oder PTC-Sensoren). Berücksichtigt man die Definition des Begriffes Merkmal nach ODM (vgl. Abschnitt 2.4.3), sind jene Eigenschaften identifizierbar, die für Interessensgruppen (Stakeholder) relevant sind. Diese Interessensgruppen werden im Rahmen der Analyse-Phase von ODM identifiziert. Da zu den Interessensgruppen beispielsweise auch Software- und Hardwareentwickler sowie Technologiefachleute gehören können, werden mit dieser Definition ggf. mehr Unterschiede und Gemeinsamkeiten identifiziert als mit der Definition aus FODA. Es kann jedoch nicht gewährleistet werden, dass alle für Automatisierungssysteme relevanten Unterschiede und Gemeinsamkeiten identifiziert werden.

Wie in diesem Abschnitt gezeigt, bietet Domain-Engineering keine explizite Unterstützung für die Entwicklung von Domänen im Bereich der Automatisierungssysteme. Die Ursache dafür liegt im Wesentlichen in der unzureichenden Analyse bestehender Applikationen, bei der die für Automatisierungssysteme relevanten Informationen nicht vollständig erfasst werden. Fehlen Informationen, besteht beim Entwurf und bei der Implementierung das Risiko, Software-

einheiten zu entwickeln, die nicht mehrfach verwendbar sind. Eine Domain-Engineering-Methode, die entsprechende Unterstützung bietet, ist derzeit nicht bekannt.

3.6 Zusammenfassende Bewertung und Festlegung der Anforderungen

In den Abschnitten 3.2 und 3.3 wurde gezeigt, dass Domain-Engineering-Methoden bisher fast ausschließlich in großen Unternehmen und Militärprojekten eingesetzt werden. Die Erfahrungen, die verschiedene Studien beschreiben, sind überaus positiv. Über den Einsatz von Domain-Engineering-Methoden in KMU sind nur wenige Studien bekannt. Diese berichten über ein großes Interesse der KMU an Domain-Engineering und über Probleme beim Einsatz von Domain-Engineering. Diese Probleme sind im Zusammenhang mit den geringen finanziellen und personellen Ressourcen der KMU zu sehen. Sie lassen sich auf die benötigten Investitionen für den Aufbau einer Domäne und auf die Notwendigkeit zur Bildung eines Teams, das mit dem Domain-Engineering verbundene Aufgaben wahrnimmt, zurückführen. Daraus ergibt sich ein erhebliches Risiko für die KMU. Es ist ersichtlich, dass die Anwendung von Domain-Engineering-Methoden in der Form, wie sie in Kapitel 2 vorgestellt wurden, für den Einsatz in KMU nur bedingt geeignet sind. Tabelle 3.5 fasst die Ergebnisse der Untersuchung der Anwendbarkeit von Domain-Engineering in Großunternehmen und KMU zusammen.

Tabelle 3.5: Bewertung der Anwendbarkeit von Domain-Engineering-Methoden in unterschiedlichen Unternehmensformen

	Großunternehmen	**KMU**
Entwicklungsaufwand	Investition zum Aufbau einer Domäne tragbar	Investition zu hoch und Return-of-Investment zu spät
Organisationsstruktur	Domain-Engineering als Pilotprojekte in Forschungsbereichen oder Forschungsgruppen	Pilotprojekte durch geringe finanzielle Ressourcen nicht durchführbar geringe personelle Ressourcen erschweren die Bildung eines Teams für das Domain-Engineering
Softwarearchitektur und Implementierung	Risiko für die Einführung einer neuen Softwarearchitektur und einer neuen Implementierung durch Pilotprojekt abschätzbar	Risiko für die abrupte Einführung einer neuen Softwarearchitektur und der entsprechenden Implementierung sehr hoch
Bewertung der Anwendbarkeit	gut	schwierig durch geringe finanzielle und personelle Ressourcen

Aus den Ergebnissen der Bewertung können Anforderungen an Domain-Engineering-Methoden für den Einsatz in KMU abgeleitet werden. Da die Probleme bei der Anwendung überwiegend

durch die geringen Ressourcen von KMU verursacht werden, konzentrieren sich die resultierenden Anforderungen auf eine zeitliche Verteilung des Entwicklungsaufwandes für Domain-Engineering. Die folgende Aufzählung zeigt die aufgestellten Anforderungen an Domain-Engineering-Methoden für den Einsatz in KMU.

- Der Entwicklungsaufwand und die damit verbundenen Investitionen müssen gering gehalten und auf einen längeren Zeitraum verteilt werden.
- Eine frühe Nutzung von mehrfach verwendbaren Teilergebnissen aus dem Domain-Engineering muss möglich sein.
- Die Umstellung der Applikationsentwicklung auf den Einsatz mehrfach verwendbarer Softwareeinheiten muss schrittweise erfolgen (keine abrupte Ablösung existierender Softwarearchitekturen)

Die in Abschnitt 2.4 vorgestellten Methoden konzentrieren sich auf die Entwicklung mehrfach verwendbarer Softwarearchitekturen und dazu passender Softwarekomponenten [FeVe99]. Die Entwicklung für die Mehrfachverwendung beginnt mit der Analyse von unterschiedlichen und gemeinsamen Merkmalen bestehender Applikationen. Bei FODA werden dabei beispielsweise für Benutzer sichtbare Merkmale berücksichtigt und ODM konzentriert sich auf Merkmale, die für unterschiedliche Interessensgruppen wichtig sind. Gemäß Abschnitt 3.5 wurden Domain-Engineering-Methoden bereits für die Entwicklung von Automatisierungssystemen eingesetzt. Eine explizite Unterstützung für deren spezifische Eigenschaften bietet keine der in Abschnitt 2.4 vorgestellten Methoden. Um sicherzustellen, dass die Unterschiede und Gemeinsamkeiten von Automatisierungssystemen bei der Domänenanalyse erkannt werden, muss eine Domain-Engineering-Methode alle notwendigen Informationen bezüglich Automatisierungsaufgaben und Einrichtungen zur Automatisierung in die Analyse einbeziehen.

Zur Erfüllung der aufgestellten Anforderungen werden in den folgenden Kapiteln Lösungsansätze vorgestellt. Zunächst wird die Einführung des Evolutionsprinzips zur zeitlichen Verteilung des Entwicklungsaufwandes diskutiert. Anschließend wird das dazu notwendige Konzept zur iterativen Entwicklung mehrfach verwendbarer Softwareeinheiten vorgestellt, das auch die spezifischen Eigenschaften von Automatisierungssystemen berücksichtigt.

4 Evolutionärer Lösungsansatz

Das aus der Natur bekannte Prinzip der Evolution beschreibt die Anpassung von Organismen an deren Lebensraum. Die Organismen verändern sich dabei über lange Zeiträume hinweg in kleinen Schritten. In Kapitel 3 wird für das Domain-Engineering die zeitliche Verteilung des Entwicklungsaufwandes gefordert. Diese Anforderung soll durch die Anwendung des Evolutionsprinzips beim Domain-Engineering erfüllt werden. In diesem Kapitel wird das Evolutionsprinzip vorgestellt und auf das Domain-Engineering übertragen. Anschließend wird ein Lösungsansatz für evolutionäres Domain-Engineering abgeleitet.

4.1 Evolutionsprinzip in der Natur

Darwin und Wallace legten in der Mitte des 19. Jahrhunderts den Grundstein für die Theorie der Evolution von Organismen [Wall1855, DaWa1858]. Diese Theorie beschreibt die ständige Anpassung von Organismen an geografische und geologische Veränderungen ihres Lebensraumes. Ursache für solche Veränderungen können Naturkatastrophen oder globale Klimaänderungen sein. Die Anpassung geschieht durch ständige Mutation und Selektion. Bei der Mutation wird das Erbgut eines Organismus auf zufällige Weise verändert und seine Eigenschaften abgewandelt. Die veränderten Organismen werden sich entweder in ihrem Lebensraum behaupten und überleben oder sie werden aussterben. Die Auswahl der Organismen mit den besten Überlebenschancen bezüglich ihres Lebensraumes wird Selektion genannt. Das Evolutionsprinzip der Natur basiert auf der Anpassung von Organismen durch viele kleine Veränderungen. Bestehende Organismen entwickeln sich dadurch Generation für Generation iterativ weiter. So entstehen über Jahrtausende neue Arten und Gattungen. Durch die Übertragung des Evolutionsprinzips auf das Domain-Engineering wird in diesem Kapitel ein Lösungsansatz für die zeitliche Verteilung des Entwicklungsaufwandes hergeleitet.

4.2 Übertragung des Evolutionsprinzips auf Domain-Engineering

Bei der Mutation wird ein kleiner Teil des Erbguts verändert. So entstehen Organismen, die sich in einem oder in einigen wenigen Merkmalen von der Elterngeneration unterscheiden. Übertragen auf das Domain-Engineering bedeutet Mutation, dass ein abgegrenzter Teil innerhalb einer Domäne verändert wird. In der Natur geschieht sowohl die Auswahl eines solchen Teils als auch seine Veränderung zufällig. Dadurch ist die Mutation nicht effektiv. Es sind viele Versuche notwendig, bis ein Organismus durch geeignete Veränderungen der richtigen Teile angepasst ist. Der Erfolg des Evolutionsprinzips in der Natur lässt sich nur durch die langen Zeiträume begründen, die für die Anpassung zur Verfügung stehen. Laut [Kurz99]

kann eine gezielte Veränderung die Anpassung beschleunigen. Da beim Domain-Engineering nur kurze Zeiträume zur Verfügung stehen, ist bei der Übertragung des Evolutionsprinzips eine gezielte Veränderung notwendig. Daraus folgt für ein evolutionäres Domain-Engineering, dass Teile innerhalb der Domäne gezielt für die Mehrfachverwendung verändert werden müssen.

Die Selektion ist die Auswahl von Organismen anhand ihrer Überlebenschancen. Ist ein Organismus gut an seine Umgebung angepasst, überlebt er. Ist er nicht ausreichend angepasst, stirbt er aus. Domain-Engineering ist die Entwicklung für die Mehrfachverwendung. Selektion übertragen auf das Domain-Engineering bedeutet, dass die Ergebnisse des Domain-Engineering anhand ihrer Mehrfachverwendbarkeit ausgewählt werden. Ergebnisse des Domain-Engineering mit guter Mehrfachverwendbarkeit werden unverändert eingesetzt. Ist die Mehrfachverwendbarkeit nicht ausreichend, werden die Ergebnisse aussortiert oder ihre Mehrfachverwendbarkeit gezielt verbessert. Für das evolutionäre Domain-Engineering ergibt sich daraus die Notwendigkeit, die Mehrfachverwendbarkeit der Ergebnisse zu überprüfen und ggf. Maßnahmen zu ihrer Verbesserung zu ergreifen.

Das iterative Vorgehen stellt in der Natur eine kontinuierliche Veränderung und damit eine ständige Anpassung der Organismen sicher. Dabei entstehen umfangreiche Veränderungen als Summe vieler kleiner Veränderungen. Beim evolutionär durchgeführten Domain-Engineering entstehen durch das iterative Vorgehen schrittweise mehrfach verwendbare Softwareeinheiten. Der Entwicklungsaufwand für das Domain-Engineering wird auf viele Iterationen und damit auf einen längeren Zeitraum verteilt. Dadurch wird auch die abrupte Einführung der mehrfach verwendbaren Ergebnisse bei der Applikationsentwicklung vermieden.

Zusammengefasst ergibt sich bei der Übertragung des Evolutionsprinzips folgendes Bild: Aus der Mutation wird die gezielte Entwicklung mehrfach verwendbarer Softwareeinheiten aus Teilen einer Domäne. Aus der Selektion wird die Überprüfung der Mehrfachverwendbarkeit bestehender Ergebnisse sowie deren Verbesserung abgeleitet. Zusammen mit dem iterativen Vorgehen folgt daraus das in Abbildung 4.1 gezeigte Ergebnis.

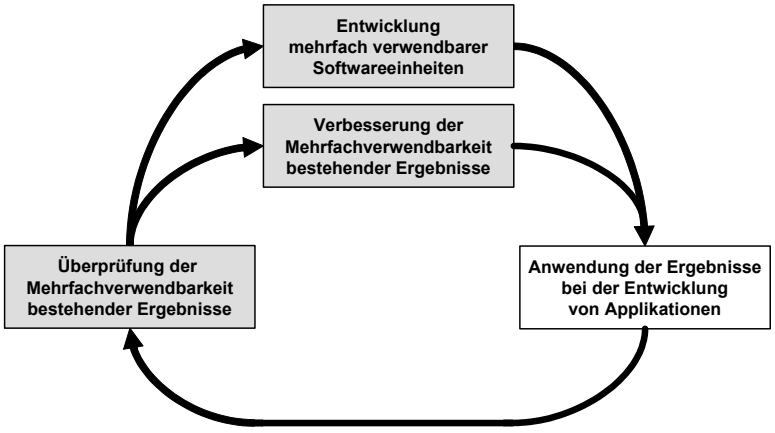


Abbildung 4.1: Übertragung des Evolutionsprinzips auf Domain-Engineering

Mit der Anwendung des Evolutionsprinzips auf Domain-Engineering können die in Kapitel 3 beschriebenen Anforderungen nach der zeitlichen Verteilung des Entwicklungsaufwandes sowie der schrittweisen Entwicklung mehrfach verwendbarer Softwareeinheiten erfüllt werden. Die einzelnen mehrfach verwendbaren Softwareeinheiten stehen sofort nach ihrer Fertigstellung für den Einsatz bei der Entwicklung von Applikationen zur Verfügung. Dadurch wird auch die Forderung nach der frühen Nutzung der Ergebnisse erfüllt.

Das Evolutionsprinzip wirkt sich auf viele Bereiche des Domain-Engineering aus. Die Domain-Engineering-Methoden nach Kapitel 2 folgen dem zweigeteilten Entwicklungsprozess, der auf dem Wasserfall-Modell beruht. Deshalb sind sie für die Anwendung des Evolutionsprinzips ungeeignet.

4.3 Evolutionäres Domain-Engineering

Der Lösungsansatz des evolutionären Domain-Engineering soll die in Kapitel 2 vorgestellten allgemeinen Merkmale des Domain-Engineering aufweisen und gleichzeitig die zeitliche Verteilung des Entwicklungsaufwandes mit Hilfe des Evolutionsprinzips ermöglichen. Zur Beschreibung des Lösungsansatzes wird im Folgenden festgelegt, welches Ziel verfolgt wird und von welchem Ausgangspunkt die Entwicklung der Domäne startet. Die Schritte, die den Weg vom Ausgangspunkt zum gesetzten Ziel beschreiben, werden definiert und deren Auswirkungen bei iterativer Anwendung diskutiert. Aus diesen Schritten werden Anforderungen für ein geeignetes Konzept zur iterativen Entwicklung von Softwareeinheiten abgeleitet.

4.3.1 Ziele des evolutionären Domain-Engineering

Gemäß Abschnitt 2.2 soll mit Domain-Engineering die strukturierte und zielgerichtete Entwicklung mehrfach verwendbarer Softwareeinheiten erreicht werden. Durch die Einführung des Evolutionsprinzips erfolgt diese Entwicklung iterativ. Dabei ist zwischen den Zielen einzelner Iterationen und dem Ziel der gesamten Entwicklung zu unterscheiden. Das Ziel der Iterationen ist die zeitliche Verteilung des Entwicklungsaufwandes. Dazu sind im Rahmen einzelner Iterationen jeweils einzelne Teile der Domäne separat voneinander zu entwickeln, die vor ihrer Entwicklung in der Domäne abgegrenzt und herausgegriffen werden. Das Ziel der gesamten Entwicklung ist die Integration der Ergebnisse aus den einzelnen Iterationen, die wieder zu einem Ganzen zusammenzufügen sind.

Zur Dokumentation der Ergebnisse des evolutionären Domain-Engineering werden aus den in Abschnitt 2.2 eingeführten Formen mehrfach verwendbarer Softwareeinheiten die Softwarekomponenten in Kombination mit der domänenspezifischen Softwarearchitektur ausgewählt. Softwarekomponenten können unabhängig voneinander entwickelt werden. Sie eignen sich damit als Ergebnis der Entwicklung einzelner Teile der Domäne im Rahmen einer Iteration. Die domänenspezifische Softwarearchitektur beschreibt eine mehrfach verwendbare Struktur, die

aus mehrfach verwendbaren Softwarekomponenten aufgebaut ist. Diese spiegeln das Ergebnis des gesamten evolutionären Domain-Engineering wider. Damit unterstützen diese beiden Formen die Ziele des evolutionären Domain-Engineering.

4.3.2 Ausgangspunkt des evolutionären Domain-Engineering

Bestehende Applikationen dienen sowohl für das in Kapitel 2 vorgestellte Domain-Engineering als auch für das evolutionäre Domain-Engineering als Basis für die Entwicklung mehrfach verwendbarer Softwareeinheiten. Jede bestehende Applikation ist durch Anforderungen, ein System-Modell, eine Softwarearchitektur und eine entsprechende Implementierung dokumentiert. Es wird davon ausgegangen, dass bestehende Applikationen bereits mit Hilfe der in Abschnitt 2.1 beschriebenen Ad-hoc-Wiederverwendung entwickelt wurden. Da dabei existierende Lösungen kopiert und angepasst werden, entstehen sowohl Gemeinsamkeiten als auch Unterschiede in den bestehenden Applikationen. Wird die Ad-hoc-Wiederverwendung für die Entwicklung mehrerer Applikationen wiederholt, entsteht eine Menge von Applikationen mit Gemeinsamkeiten und Unterschieden. Eine solche Konstellation von Applikationen ist typisch für KMU und soll als Ausgangspunkt des evolutionären Domain-Engineering dienen.

4.3.3 Schritte einer einzelnen Iteration

Die in Abschnitt 4.2 beschriebenen Schritte, die bei der Übertragung des Evolutionsprinzips auf das Domain-Engineering entstehen, sind iterativ anzuwenden. In diesem Abschnitt wird eine einzelne Iteration betrachtet. Dazu werden die Ziele sowie die Anforderungen an die Schritte einer Iteration beschrieben.

Entwicklung mehrfach verwendbarer Softwareeinheiten

Wie in Abschnitt 4.3.1 skizziert, sollen im Rahmen der Entwicklung mehrfach verwendbarer Softwareeinheiten sowohl eine domänenspezifische Softwarearchitektur als auch mehrfach verwendbare Softwarekomponenten entstehen.

Für die evolutionäre Entwicklung einer domänenspezifischen Softwarearchitektur muss die Zerlegung in Softwarekomponenten schrittweise erfolgen. Das hat zur Folge, dass die Softwarekomponenten und deren Struktur zunächst nicht die gesamte Domäne abdecken. Aus diesem Grund ist im Rahmen der Entwicklung der domänenspezifischen Softwarearchitektur auch eine Integration von Softwarearchitekturen bestehender Applikationen notwendig.

Bei der Zerlegung müssen abgegrenzte Teile mit einem definierten Funktionsumfang und Schnittstellen entstehen, die sich für die Entwicklung mehrfach verwendbarer Softwarekomponenten eignen. Das heißt, die Teile müssen Potenzial besitzen wieder verwendet zu werden. Beim Domain-Engineering erfolgt eine Abschätzung dieses Potenzials auf Basis der in Abschnitt 2.2.2 erläuterten Annahmen. Demnach besitzen Teile dann Potenzial wieder

verwendet zu werden, wenn sie in zukünftigen Applikationen benötigt werden und wenn sie in bestehenden Applikationen häufig eingesetzt wurden. Diese beiden Informationen sind beim Domain-Engineering im Domänenmodell enthalten, das im Rahmen der Domänenanalyse erstellt wird. Diese Phase wird vor dem Domänenentwurf durchgeführt (vgl. Abschnitt 2.2.2). Durch die Betrachtung der gesamten Domäne ist sie sehr aufwändig. Da der Entwicklungsaufwand zeitlich verteilt werden soll, ist auch die Domänenanalyse zeitlich zu verteilen. Im Rahmen eines Evolutionsschrittes sollen nur eine bis einige wenige Softwarekomponenten entwickelt werden. Dazu reichen Informationen über Teile der Domäne aus. Dementsprechend reicht es auch aus, die Domänenanalyse für Teile der Domäne durchzuführen.

Die auf Basis der Zerlegung entstehenden Softwarekomponenten decken jeweils einen Teil der Domäne ab. Dieser Teil wächst über die Iterationen hinweg. Jede neue Softwarekomponente ist in die bestehende domänenspezifische Softwarearchitektur zu integrieren. Dabei sind Schnittstellen und Verknüpfungen der Softwarekomponenten untereinander und mit dem noch nicht abgedeckten Rest zu berücksichtigen. Abbildung 4.2 zeigt beispielhaft die Entwicklung einer Softwarekomponente und ihre Integration in die domänenspezifische Softwarearchitektur.

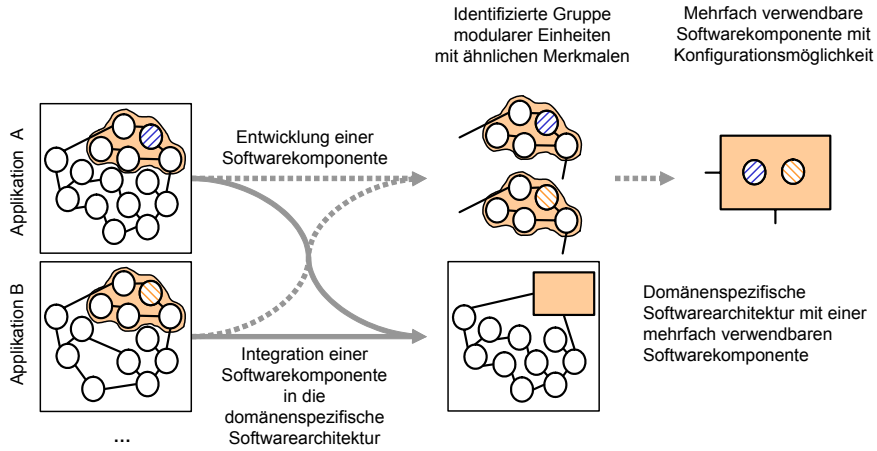


Abbildung 4.2: Entwicklung mehrfach verwendbarer Softwarekomponenten und Integration zu einer domänenspezifischen Softwarearchitektur

Wird das evolutionäre Domain-Engineering im Rahmen des ersten Evolutionsschrittes erstmalig durchlaufen, besteht noch keine domänenspezifische Softwarearchitektur. In diesem Fall ist die Softwarearchitektur einer bestehenden Applikation zu wählen und als domänenspezifische Softwarearchitektur der ersten Generation zu definieren. Sie enthält zu diesem Zeitpunkt noch keine mehrfach verwendbaren Softwarekomponenten. Da die erste Generation der domänenspezifischen Softwarearchitektur als Grundlage für die Entwicklung der ersten Applikation dient, ist es sinnvoll, die Softwarearchitektur einer bestehenden Applikation als domänenspezifische Softwarearchitektur zu wählen, die der nächsten Applikation sehr ähnlich ist.

Die mehrfach verwendbaren Softwarekomponenten sind auf Basis der identifizierten Teile zu entwickeln, die bei der Zerlegung der Domäne entstehen. Nach Abschnitt 2.2.1 stellen diese

Teile eine vertikale Domäne dar, auf die Domain-Engineering angewandt werden kann. Domain-Engineering-Methoden nach Kapitel 2 beinhalten Aktivitäten zur Auswahl einer geeigneten Domäne (vgl. Domain-Scoping), die in diesem Fall nicht anwendbar sind. Es werden Aktivitäten benötigt, die an die Anforderungen beim evolutionären Domain-Engineering angepasst sind. Als Ergebnis müssen mehrfach verwendbare Softwarekomponenten entstehen, welche die in Abschnitt 2.2.1 erläuterten Eigenschaften besitzen.

Anwendung der Ergebnisse bei der Entwicklung von Applikationen

Bei der Entwicklung einer neuen Applikation innerhalb der betrachteten Domänen können die bislang erarbeiteten Ergebnisse eingesetzt werden. Diese beinhalten die domänenspezifische Softwarearchitektur in ihrer aktuellen Generation sowie die bis zu diesem Zeitpunkt entwickelten mehrfach verwendbaren Softwarekomponenten. Bei der Instanziierung werden die Softwarekomponenten durch Konfiguration an die gestellten Anforderungen angepasst. Anschließend werden die instanziierten Softwarekomponenten gemäß der domänenspezifischen Softwarearchitektur in die Applikation integriert. Da die Ergebnisse während der Evolution nicht den gesamten Funktionsumfang abdecken, der zur vollständigen Erstellung einer Applikation notwendig ist, muss ein Teil auf herkömmliche Weise entwickelt werden.

Überprüfung der Mehrfachverwendbarkeit bestehender Ergebnisse

Bei der Entwicklung zukünftiger Applikationen können neue Anforderungen hinzukommen, die von den entwickelten mehrfach verwendbaren Ergebnissen nicht abgedeckt werden. Dies bedeutet eine Beeinträchtigung der Mehrfachverwendbarkeit der betroffenen Softwarekomponenten bzw. der domänenspezifischen Softwarearchitektur. Um die Mehrfachverwendbarkeit der mehrfach verwendbaren Ergebnisse auf längere Sicht zu erhalten, müssen diese überwacht werden. Es sind Überprüfungen der Mehrfachverwendbarkeit notwendig, deren Ergebnisse über das weitere Vorgehen entscheiden. Ist die Mehrfachverwendbarkeit der bestehenden Ergebnisse ausreichend, ist ein unveränderter Einsatz möglich. Reicht sie nicht aus, muss eine Überarbeitung erfolgen. Die Domain-Engineering-Methoden nach Kapitel 2 enthalten keine Aktivitäten zur Überprüfung der Mehrfachverwendbarkeit.

Verbesserung der Mehrfachverwendbarkeit bestehender Ergebnisse

Wurde bei bestehenden Ergebnissen eine unzureichende Mehrfachverwendbarkeit festgestellt, ist deren Mehrfachverwendbarkeit zu verbessern. Die Ergebnisse sind so zu beeinflussen, dass sie den neuen Anforderungen in der Domäne genügen. Wird dieser Zustand erreicht, ist die Mehrfachverwendbarkeit wieder hergestellt. Die Verbesserung der Mehrfachverwendbarkeit, wie sie das evolutionäre Domain-Engineering vorsieht, wird von keiner der in Kapitel 2 vorgestellten Domain-Engineering-Methoden unterstützt.

Bei der einmaligen Anwendung der beschriebenen Schritte wird ein Teil der Domäne identifiziert, der als Grundlage für die Entwicklung mehrfach verwendbarer Software-

komponenten und der Erweiterung der domänenspezifischen Softwarearchitektur dient. Die resultierende domänenspezifische Softwarearchitektur besteht aus den bereits entwickelten Softwarekomponenten sowie einem unbearbeiteten Rest. Erst durch die wiederholte Anwendung vervollständigen sich die Sammlung von Softwarekomponenten und die domänenspezifische Softwarearchitektur. Das geschieht schrittweise - Generation für Generation.

4.3.4 Auswirkungen durch iterative Anwendung der Schritte

Zur Erläuterung der Auswirkungen des iterativen Vorgehens werden die vorgestellten Schritte im Folgenden in mehreren Iterationen durchlaufen und die Ergebnisse beispielhaft dargestellt.

In der ersten Generation wird die domänenspezifische Softwarearchitektur erstellt. Die Softwarearchitektur einer ähnlichen, bestehenden Applikation wird als Basis verwendet. Alle zukünftigen Entwicklungen finden auf dieser Basis statt. Mit der Identifikation potenziell mehrfach verwendbarer Teile, der darauf basierenden Entwicklung einer mehrfach verwendbaren Softwarekomponente sowie der Weiterentwicklung der domänenspezifischen Softwarearchitektur entsteht die erste Generation an mehrfach verwendbaren Ergebnissen. Die resultierende domänenspezifische Softwarearchitektur der ersten Generation enthält eine oder mehrere mehrfach verwendbare Softwarekomponenten. Der Rest wurde bisher noch nicht bearbeitet und ist deshalb nicht für die Mehrfachverwendung vorbereitet. Die mehrfach verwendbaren Softwarekomponenten besitzen Konfigurationsmöglichkeiten, die es erlauben, bei der Instanziierung alle berücksichtigten Variabilitäten zu rekonstruieren. Dieser erste Evolutionsschritt und seine Ergebnisse sind am Beispiel der Entwicklung einer mehrfach verwendbaren Softwarekomponente in Abbildung 4.3 dargestellt. Die Abbildung 4.4 zeigt die Erstellung einer neuen Applikation auf Basis dieser Ergebnisse. Dazu wird die Softwarekomponente durch Konfiguration an die Anforderungen der neuen Applikation angepasst. Der noch nicht mit Softwarekomponenten realisierte Teil der domänenspezifischen Softwarearchitektur wird auf herkömmliche Weise entwickelt.

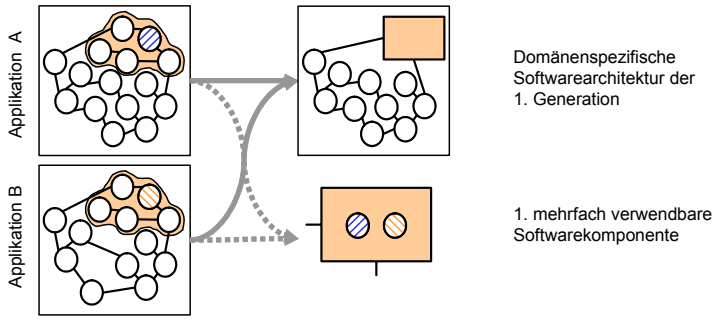


Abbildung 4.3: Entwicklung der 1. Generation mehrfach verwendbarer Softwareeinheiten

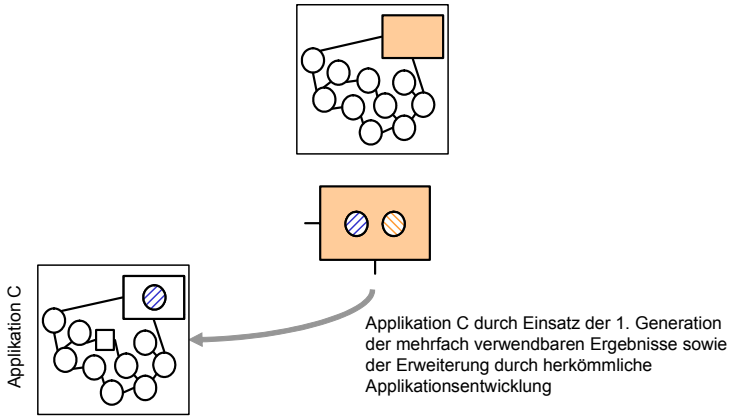


Abbildung 4.4: Anwendung der 1. Generation mehrfach verwendbarer Softwareeinheiten

Ergibt die Überprüfung der Mehrfachverwendbarkeit der bisherigen Ergebnisse der 1. Generation eine ausreichende Mehrfachverwendbarkeit, wird in der folgenden Iteration eine weitere Softwarekomponente erstellt. Bei der Entwicklung der 2. Generation werden zur Identifikation potenziell mehrfach verwendbarer Teile wieder die bestehenden Applikationen untersucht, die jetzt auch die zuletzt entwickelte Applikation beinhalten. Es wird eine mehrfach verwendbare Softwarekomponente erstellt und die 1. Generation der domänenspezifischen Softwarearchitektur zur 2. Generation erweitert. Abbildung 4.5 stellt diesen Vorgang an einem Beispiel dar. Die Erstellung einer neuen Applikation unter Anwendung der 2. Generation der mehrfach verwendbaren Softwareeinheiten ist in Abbildung 4.6 skizziert. Dabei kommt sowohl die neu entwickelte Softwarekomponente als auch die in der ersten Iteration erstellte Softwarekomponente zum Einsatz.

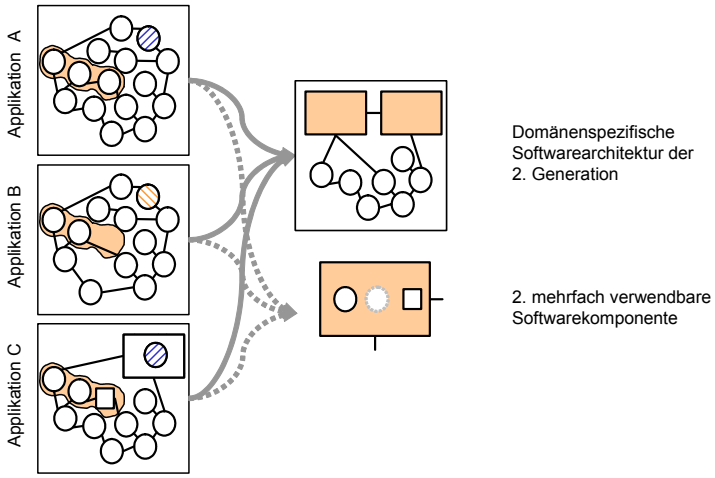


Abbildung 4.5: Entwicklung der 2. Generation mehrfach verwendbarer Softwareeinheiten

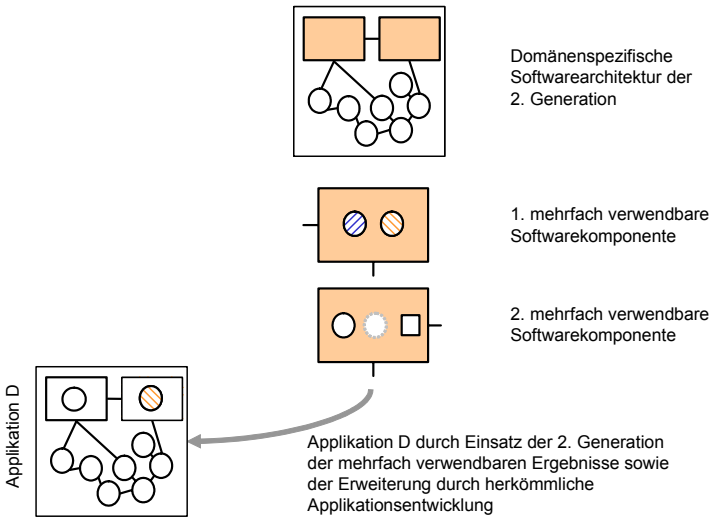


Abbildung 4.6: Anwendung der 2. Generation mehrfach verwendbarer Softwareeinheiten

Treten neue Anforderungen auf, welche durch die bestehenden mehrfach verwendbaren Ergebnisse der 2. Generation nicht erfüllt werden, wird bei der Überprüfung der Mehrfachverwendbarkeit ein Mangel festgestellt. In der folgenden Iteration ist deshalb die Verbesserung der bestehenden Ergebnisse vorzunehmen. Von den Verbesserungen können sowohl die domänenspezifische Softwarearchitektur als auch Softwarekomponenten betroffen sein. Ein Beispiel, bei dem eine Softwarekomponente verbessert werden muss, ist in Abbildung 4.7 dargestellt. Dabei wird eine weitere Alternative gefordert, welche von der bestehenden Softwarekomponente nicht unterstützt wird. Die Softwarekomponente ist deshalb um diese Alternative zu erweitern. Die Ergebnisse der Verbesserung stellen die 3. Generation mehrfach verwendbarer Ergebnisse dar. Bei ihrer Anwendung erfüllen sie die neuen Anforderungen und die Mehrfachverwendbarkeit ist wieder hergestellt.

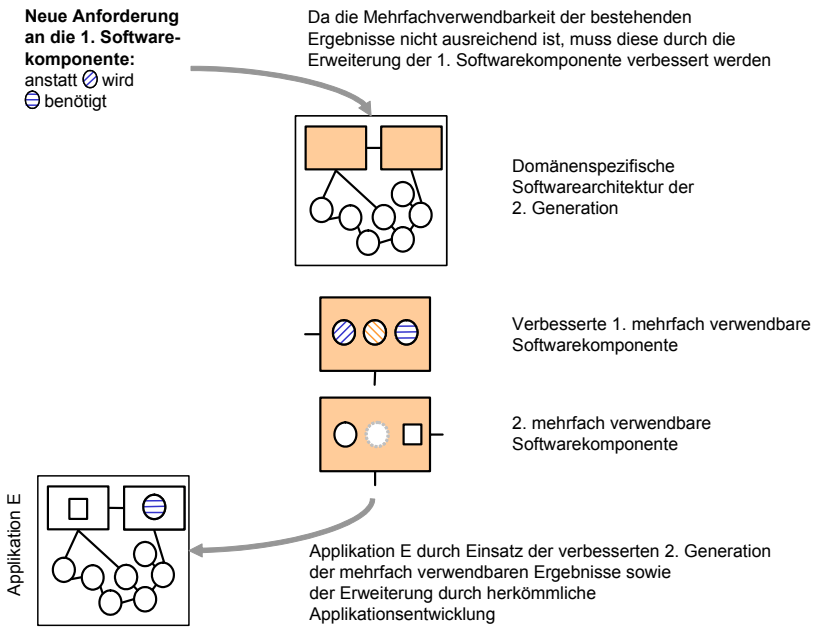


Abbildung 4.7: Verbesserung bestehender Ergebnisse und deren Anwendung

Dieser Abschnitt zeigt die Auswirkungen der einzelnen Schritte über mehrere Iterationen, mit Fokus auf dem Aufbau einer Sammlung mehrfach verwendbarer Softwarekomponenten sowie der schrittweisen Entwicklung der domänenspezifischen Softwarearchitektur.

4.4 Bewertung des evolutionären Lösungsansatzes

Im vorliegenden Kapitel wurde ein Lösungsansatz auf Basis des Evolutionsprinzips vorgestellt. Er beschreibt eine Möglichkeit zur schrittweisen Entwicklung von mehrfach verwendbaren Softwarekomponenten und zum iterativen Aufbau einer domänenspezifischen Softwarearchitektur. Der Lösungsansatz erfüllt die Forderung nach einer zeitlichen Verteilung des Entwicklungsaufwandes sowie der damit verbundenen zeitlichen Verteilung der notwendigen Investitionen. Außerdem sind die Ergebnisse der einzelnen Iterationen direkt nach deren Erstellung bei der Entwicklung neuer Applikationen einsetzbar, was sowohl eine frühe Nutzung von Teilergebnissen gewährleistet als auch die schrittweise Einführung der neuen domänenspezifischen Softwarearchitektur ermöglicht. Damit erfüllt der vorgestellte Lösungsansatz alle in Kapitel 3 aufgestellten Anforderungen bezüglich der Anwendbarkeit in KMU.

Die in Abschnitt 2.4 vorgestellten Domain-Engineering-Methoden orientieren sich im Wesentlichen am zweigeteilten Entwicklungsprozess. Dieser stellt ein Wasserfall-Modell dar, das die iterative Anwendung nicht unterstützt. Des weiteren sind keine Aktivitäten eingeplant, die eine Überprüfung und Verbesserung der Mehrfachverwendbarkeit vorsehen. Es finden sich jedoch zwei Ansätze zur zeitlichen Verteilung des Entwicklungsaufwandes. ODM schlägt vor, nach der Entwicklung der domänenspezifischen Softwarearchitektur die Softwarekomponenten bei Bedarf nach und nach zu implementieren. Begonnen wird mit den Softwarekomponenten, die für die Entwicklung der nächsten Applikation benötigt werden. Dieses Vorgehen ermöglicht die Verteilung der Implementierung. Der Hauptteil des Entwicklungsaufwandes muss in Vorleistung erbracht werden und die Anforderung nach der zeitlichen Verteilung des Entwicklungsaufwandes kann nicht vollständig erfüllt werden. Die Methode PuLSE führt die Entwicklung der domänenspezifischen Softwarearchitektur und der dazugehörigen Softwarekomponenten iterativ durch. Die Anwendung der Ergebnisse erfolgt jedoch erst nach Fertigstellung der gesamten domänenspezifischen Softwarearchitektur und der entsprechenden Softwarekomponenten. Eine frühe Nutzung von Teilergebnissen aus den einzelnen Iterationen wird nicht ermöglicht. Außerdem wird die bestehende Softwarearchitektur abrupt abgelöst.

Keine der in Kapitel 2 vorgestellten Domain-Engineering-Methoden erfüllt alle der in Kapitel 3 gestellten Anforderungen. Um die Anwendung bekannter Domain-Engineering-Methoden zusammen mit dem Evolutionsprinzip zu ermöglichen, sind weit reichende Anpassungen notwendig. Insbesondere das iterative Vorgehen wirkt sich auf viele Bereiche des Domain-Engineering aus. Es wird ein geeignetes Konzept zur Umsetzung des iterativen Vorgehens benötigt. Die Ausarbeitung eines solchen Konzeptes ist Thema des folgenden Kapitels.

5 Konzept zur iterativen Entwicklung mehrfach verwendbarer Softwareeinheiten für Automatisierungssysteme

In diesem Kapitel wird ein Konzept entwickelt, das die Grundlagen des Domain-Engineering nach Kapitel 2, das Evolutionsprinzip nach Kapitel 4 und die Anforderungen zur Entwicklung von Automatisierungssystemen nach Abschnitt 3.4 und 3.5 berücksichtigt. Es wird zunächst die Problemstellung, die aus der iterativen Vorgehensweise und durch die Berücksichtigung von Automatisierungssystemen hervorgeht, erörtert. Anschließend wird das Konzept erarbeitet. Dabei werden die in Abschnitt 4.3.3 eingeführten Schritte getrennt betrachtet. Die Gliederung der Abschnitte des vorliegenden Kapitels ist an diese Schritte angelehnt.

5.1 Problemstellung

Domain-Engineering nach Abschnitt 2.2 folgt einem Wasserfall-Modell, bei dem die einzelnen Aktivitäten zunächst vollständig abgeschlossen werden, um dann die Folgeaktivität basierend auf den Ergebnissen der Vorgängeraktivität durchzuführen. Für die iterative Vorgehensweise soll die Aktivität Domänenentwurf in den Entwurf der domänenspezifischen Softwarearchitektur und die separate Entwicklung von Softwarekomponenten aufgeteilt werden (vgl. Kapitel 4). Beide Tätigkeiten basieren auf den Ergebnissen der Aktivität Domänenanalyse. Bei einer zeitlichen Verteilung der beiden Tätigkeiten müssen Teile der Analyseergebnisse früher zur Verfügung stehen, während andere erst zu einem späteren Zeitpunkt benötigt werden. Die Analyse der Domäne ist deshalb ebenfalls zeitlich zu verteilen. Dabei ist zu berücksichtigen, welche Informationen aus den Analyseergebnissen bereits für die Zerlegung der Domäne benötigt werden und welche Informationen erst für die Entwicklung einzelner Softwarekomponenten notwendig sind. Ein geeignetes Konzept muss die beiden Teile der Analyse jeweils auf die benötigten Informationen der Folgeaktivitäten ausrichten.

Bei der Entwicklung von Automatisierungssystemen sind insbesondere die Vorgänge im zu automatisierenden technischen Prozess sowie die Einrichtungen, die zur Automatisierung notwendig sind, zu berücksichtigen. Sie beeinflussen durch die Synchronisierung der zeitlichen Abläufe im technischen System mit den zeitlichen Abläufen im Automatisierungssystem die Echtzeiteigenschaften von Automatisierungssystemen (vgl. Abschnitt 3.4). Das gilt nicht nur bei der Entwicklung einzelner Applikationen sondern auch bei der Entwicklung domänenspezifischer Softwarearchitekturen und mehrfach verwendbarer Softwarekomponenten. Daraus ergibt sich folgende Problemstellung für das Domain-Engineering für Automatisierungssysteme:

- Domänenanalyse: Bei der Analyse von Unterschieden und Gemeinsamkeiten bestehender Systeme sind die Automatisierungsaufgaben sowie die eingesetzten Einrichtungen zur Automatisierung einzubeziehen.
- Domänenentwurf: Beim Entwurf von Variationspunkten sind geeignete Mechanismen zu wählen, die Unterschiede in parallel ablauffähigen Vorgängen, in der Synchronisierung, in den Beziehungen der Vorgänge untereinander sowie in den Beziehungen der Vorgänge zur Umwelt unterstützen.
- Domänenimplementierung: Bei der Implementierung muss insbesondere die Umsetzung der Variationspunkte aus dem Domänenentwurf möglich sein.

5.2 Ziel und Ausgangspunkt der Entwicklung

Ein Konzept zur iterativen Entwicklung mehrfach verwendbarer Softwareeinheiten für Automatisierungssysteme hat Anforderungen, die aus der iterativen Vorgehensweise resultieren und Anforderungen, welche sich aus dem Anwendungsgebiet der Automatisierungssysteme ergeben, zu erfüllen. Ergebnisse der Entwicklung sollen eine domänenspezifische Softwarearchitektur sowie mehrfach verwendbare Softwarekomponenten gemäß Abschnitt 2.2 sein. Bei der Umsetzung des Konzeptes sollen weitgehend bekannte Mechanismen aus dem Domain-Engineering gemäß Kapitel 2 zum Einsatz kommen.

Es wird davon ausgegangen, dass bereits einige ähnliche Applikationen entwickelt wurden und dabei Ad-hoc-Wiederverwendung gemäß Abschnitt 2.1 zum Einsatz kam. Diese bestehenden Applikationen stellen den Ausgangspunkt für die Entwicklung einer domänenspezifischen Softwarearchitektur und mehrfach verwendbarer Softwarekomponenten dar. In den folgenden Abschnitten wird ein Konzept zur iterativen Entwicklung mehrfach verwendbarer Softwareeinheiten konzipiert. Dabei werden die folgenden Teile gemäß Abschnitt 4.3 getrennt behandelt:

- Identifikation potenziell mehrfach verwendbarer Teile
- Entwicklung mehrfach verwendbarer Softwarekomponenten
- Integration der Softwarekomponenten in die domänenspezifische Softwarearchitektur
- Überprüfung der Mehrfachverwendbarkeit bestehender Ergebnisse
- Verbesserung der Mehrfachverwendbarkeit von Ergebnissen

5.3 Identifikation potenziell mehrfach verwendbarer Teile

5.3.1 Merkmale potenziell mehrfach verwendbarer Teile

Bei der Entwicklung für die Mehrfachverwendung sollen Softwareeinheiten erstellt werden, die für den Einsatz in zukünftigen Applikationen nützlich sind. Ob eine mehrfach verwendbare Softwareeinheit wirklich nützlich ist, hängt vom tatsächlichen zukünftigen Bedarf an dieser

Softwareeinheit ab. Bei Automatisierungssystemen ist dieser Bedarf auch an die zu erfüllenden Automatisierungsaufgaben sowie die einzusetzenden Einrichtungen zur Automatisierung gekoppelt. Besteht beispielsweise Bedarf an einer bestimmten Funktionalität zum Einlesen von Prozesssignalen, muss ein mehrfach verwendbares Teil nicht nur diese Funktionalität erbringen, sondern darüber hinaus die eingesetzte Rechnerhardware, Prozessperipherie, usw. unterstützen.

Die Domänenanalyse nach Abschnitt 2.2.2 kann zur Untersuchung bestehender Applikationen und zukünftiger Trends aufgrund der Anpassung an das Wasserfall-Modell für das zu entwickelnde Konzept nicht genutzt werden (vgl. Abschnitt 5.1). Deshalb werden die bestehenden Applikationen unter Berücksichtigung der folgenden beiden Annahmen des Domain-Engineering betrachtet (vgl. Abschnitt 2.2.2): Die eine Annahme besagt, dass sich Merkmale (Features) von Software dann zur Mehrfachverwendung eignen, wenn sie in bestehenden Applikationen bereits häufig eingesetzt wurden. Die andere Annahme geht davon aus, dass sich Merkmale, die bei der Entwicklung zukünftiger Applikationen häufig benötigt werden, für die Mehrfachverwendung eignen. Die bestehenden Applikationen enthalten Teile, die durch Ad-hoc-Wiederverwendung kopiert und angepasst wurden. Wird ein solcher Teil mehrmals ad hoc wieder verwendet, ist er in ähnlicher oder alternativer Form in verschiedenen Applikationen vorhanden. Übertragen auf die erste Annahme des Domain-Engineering bedeutet dies, dass für mehrmals ad hoc wieder verwendete Lösungen ein Bedarf für den Einsatz in zukünftigen Applikationen angenommen werden kann.

Bei der Ad-hoc-Wiederverwendung wird der Bedarf an bestehenden Lösungen für die Entwicklung der aktuellen Applikation erkannt. Es kann deshalb davon ausgegangen werden, dass für eine mehrfach verwendbare Softwareeinheit Bedarf besteht, wenn sie die Funktionalität der bestehenden Lösungen abdeckt. Die zweite Annahme des Domain-Engineering geht von zukünftigem Bedarf aus. Der Bedarf bei der Entwicklung der aktuellen Applikation besteht sicher. Gelingt es, eine mehrfach verwendbare Softwareeinheit auf Basis einer erkannten Lösung parallel zur aktuellen Applikation zu entwickeln und diese Softwareeinheit bereits in der Applikation anzuwenden, in der Bedarf dafür erkannt wurde, kann der investierte Entwicklungsaufwand zu einem Teil amortisiert werden. Nach Abschnitt 2.3.1 amortisiert sich der gesamte Entwicklungsaufwand für eine mehrfach verwendbare Softwareeinheit nach ca. 3 bis 4 Anwendungen. Das heißt, dass der einmalige Einsatz einer mehrfach verwendbaren Softwareeinheit ca. 1/4 bis 1/3 der Amortisierung ausmacht. Es stellt sich die Frage, wie Teile bestehender Applikationen identifiziert werden können, die sich für die Entwicklung mehrfach verwendbarer Softwareeinheiten eignen.

5.3.2 Erkennung potenziell nützlicher Teile

Das Wissen erfahrener Entwickler über Ad-hoc-Wiederverwendung von Automatisierungssystemen kann für das zu entwickelnde Konzept genutzt werden. Die Entwickler erkennen bei der Ad-hoc-Wiederverwendung potenziell mehrfach verwendbare Teile mit Hilfe ihres Wissens

über bestehende Applikationen, die damit erfüllten Automatisierungsaufgaben sowie die dazu eingesetzten Einrichtungen zur Automatisierung. Bei der Übertragung auf das Konzept ergeben sich zwei Probleme. Zum einen besteht bei der Entwicklung einer Applikation ggf. Bedarf für mehr als eine bestehende Lösung, wodurch mehrere potenziell mehrfach verwendbare Teile erkannt werden. Zum anderen wird ein potenziell nützliches Teil während der Entwicklung derjenigen Applikation erkannt, in der die daraus erstellte Softwarekomponente eingesetzt werden soll. Daraus ergeben sich folgende Vorgaben:

- In einem Iterationsschritt sollen nur eine bis einige wenige Softwarekomponenten entwickelt werden. Als Basis für die Entwicklung sind deshalb ein bis einige wenige Teile aus der Menge der als potenziell mehrfach verwendbar erkannten Teile auszuwählen.
- Mehrfach verwendbare Softwarekomponenten sollen in der Applikation eingesetzt werden, in der Bedarf für diese Softwarekomponente festgestellt wurde. Deshalb muss die Entwicklung mehrfach verwendbarer Softwareeinheiten und die Applikationsentwicklung parallel durchgeführt und synchronisiert werden.

Bei der Auswahl eines potenziell mehrfach verwendbaren Teils ist eine Entscheidung zu treffen, die über Auswahlkriterien oder Prioritäten zu begründen ist. Dazu ist ein Überblick über alle erkannten Teile notwendig. Das bedeutet, dass die Auswahl erst getroffen werden kann, wenn alle möglichen Teile erkannt wurden. Da die potenziell mehrfach verwendbaren Teile erst während der Entwicklung der aktuellen Applikation erkannt werden, liegt ein vollständiger Überblick erst gegen Ende der Implementierungs-Phase vor. Das ist zu spät für die Auswahl eines geeigneten Teils, da die Softwarekomponente auf Basis eines erkannten Teils sowie die Erweiterung der domänenspezifischen Softwarearchitektur nach der Auswahl entwickelt werden muss. Der späteste Zeitpunkt für den Einsatz der mehrfach verwendbaren Ergebnisse in der Applikationsentwicklung ist die Integrations-Phase. Die Fertigstellung der Entwicklung mehrfach verwendbarer Softwareeinheiten muss deshalb vor der Integrations-Phase der Applikationsentwicklung erfolgen. Um Zeit für die Entwicklung der Softwarekomponenten und der domänenspezifischen Softwarearchitektur zu gewinnen, bleibt nur die Auswahl eines erkannten Teils in einer frühen Phase der Applikationsentwicklung. Das greift der Ad-hoc-Wiederverwendung vor und muss unter Umständen ohne den vollständigen Überblick über alle geeigneten Teile geschehen. Der früheste Zeitpunkt für die Auswahl ist nach der Fertigstellung der fachtechnischen Lösungskonzeption gegen Ende der Definitions-Phase. Während der Applikationsentwicklung werden aus der fachtechnischen Lösungskonzeption, in Abhängigkeit vom eingesetzten Softwareentwicklungsprozess für die Applikationsentwicklung, z.B. ein Grobentwurf und ein Feinentwurf erarbeitet, der schließlich implementiert wird. Die Fertigstellung dieser Entwürfe sowie die Implementierung stellen weitere Möglichkeiten für die Auswahl dar. Je später die Auswahl vorgenommen wird, desto mehr Informationen sind verfügbar. Auf der anderen Seite nimmt damit die zur Verfügung stehende Zeit für die Entwicklung von Softwarekomponenten ab. Bei der komponentenbasierten Softwareentwicklung werden anhand der fachtechnischen Lösungskonzeption geeignete Softwarekomponenten für die Entwicklung einer

Applikation ausgewählt [Szyp98, HeCo01]. Die Untersuchung zur Erkennung des Bedarfs an einer Softwarekomponente sollte gegen Ende der Definitions-Phase oder zu Beginn der Entwurfs-Phase der Applikationsentwicklung erfolgen. Abbildung 5.1 zeigt beispielhaft den dargestellten zeitlichen Ablauf. Die Entwicklung einer Applikation ist in drei Phasen aufgeteilt. Parallel dazu wird ein Iterationsschritt gemäß dem Konzept zur iterativen Entwicklung mehrfach verwendbaren Softwareeinheiten für Automatisierungssysteme durchgeführt.

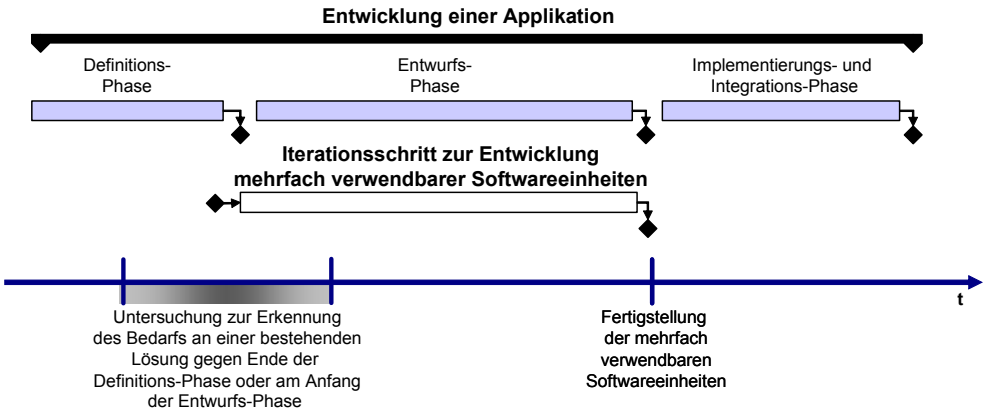


Abbildung 5.1: Parallele Entwicklung einer Applikation und mehrfach verwendbarer Softwareeinheiten

Die domänenspezifische Softwarearchitektur und mehrfach verwendbare Softwarekomponenten werden angewandt, wenn diese nützlich für die Entwicklung der aktuellen Applikation sind. Das ist dann der Fall, wenn sie einen Teil der geforderten Automatisierungsaufgaben der Applikation und damit einen Teil des Automatisierungssystems abdecken sowie kompatibel zu den eingesetzten Einrichtungen für die Automatisierung sind. Bei der Auswahl muss zum einen bekannt sein, für welche Automatisierungsaufgaben Bedarf besteht und welche Einrichtungen zur Automatisierung eingesetzt werden. Zum anderen müssen für die zur Verfügung stehenden Softwarekomponenten die abgedeckten Automatisierungsaufgaben sowie die kompatiblen Einrichtungen zur Automatisierung dokumentiert sein. Mit diesen Informationen lässt sich herausfinden, welche der Automatisierungsaufgaben bzw. Einrichtungen zur Automatisierung noch nicht von bestehenden Softwarekomponenten abgedeckt werden. Ist zudem noch Expertenwissen über bestehende Applikationen vorhanden, ist es möglich, mit überschaubarem Aufwand bestehende Lösungen zu erkennen, die für die Entwicklung der aktuellen Applikation nützlich sind.

Die Entwicklung des zukünftigen Marktes wird anhand der aktuellen Applikation für einen kurzen Zeitraum vorausgesagt. Um Teile zu identifizieren, die in vielen bestehenden Applikationen vorkommen, sind neben der Lösung aus einer bestehenden Applikation weitere Applikationen der Domäne bezüglich dieser Lösung zu vergleichen.

5.3.3 Abgrenzung potenziell mehrfach verwendbarer Teile

Die im vorhergehenden Abschnitt erkannte Lösung ist Teil einer bestehenden Applikation und sie ist nützlich für die Entwicklung der aktuellen Applikation. Wie in Abschnitt 2.2 beschrieben, begründet sich das Potenzial für die Mehrfachverwendung von Software bzw. Teilen von Software auf den Annahmen, dass sie zum einen in vielen bestehenden Applikationen in ähnlicher Form vorhanden sind und dass sie für die Entwicklung zukünftiger Applikationen nützlich sind. Das Vorhandensein der erkannten Lösung in einer einzigen bestehenden Applikation reicht deshalb nicht aus, um eine Aussage über ihr Potenzial für die Mehrfachverwendung zu treffen. Um eine solche Aussage machen zu können, ist zu untersuchen, ob die erkannte Lösung auch in anderen bestehenden Applikationen eingesetzt wurde. Dazu sind bestehende Applikationen bezüglich enthaltener Teile zu analysieren, die der erkannten Lösung ähnlich sind bzw. Alternativen dazu darstellen.

Ähnlich sind sich Teile, wenn sie große Gemeinsamkeiten aufweisen. Es müssen also Teile gefunden werden, die Gemeinsamkeiten bezüglich Automatisierungsaufgabe und den eingesetzten Einrichtungen zur Automatisierung aufweisen. Alternative Teile weisen bei einem direkten Vergleich ggf. keine Gemeinsamkeiten auf. Erst bei der Betrachtung übergeordneter Aufgaben und Zielsetzungen sind Gemeinsamkeiten zu erkennen. Dazu ist eine Generalisierung bzw. Abstraktion der Teile notwendig [Schm97]. Es müssen also Teile gefunden werden, die Gemeinsamkeiten bezüglich der generalisierten bzw. abstrahierten Automatisierungsaufgabe haben oder bei denen Gemeinsamkeiten zu erkennen sind, wenn die einsetzen Einrichtungen zur Automatisierung generalisiert bzw. abstrahiert betrachtet werden. Als ähnlich können beispielsweise Treiberprogramme für die Feldbusse *CAN Highspeed* und *CAN Lowspeed* [Etsch94] bezeichnet werden. Der Zugriff auf die empfangenen bzw. zu sendenden Daten ist gleich. Die Unterschiede liegen in den unteren Schichten der beiden Feldbusprotokolle und werden teilweise von der Hardware gekapselt. Die wenigen Unterschiede in der Software liegen im Detail und werden von den Treiberprogrammen gekapselt. Als alternativ zu CAN Highspeed und CAN Lowspeed können beispielsweise Treiberprogramme für den Feldbus *TTP* [GuNä99] bezeichnet werden. TTP ist ebenfalls ein prozessnahes Kommunikationssystem, besitzt jedoch ein anderes Echtzeitverhalten.

Zum Erkennen ähnlicher Teile wird Wissen über die charakteristischen Merkmale der Automatisierungsaufgaben bzw. die Einrichtungen zur Automatisierung vorausgesetzt. Bei den aufgeführten Beispielen sind dies folgende Merkmale: Zugriff auf die zu empfangenen und zu sendenden Daten von CAN Highspeed und CAN Lowspeed, Echtzeitverhalten von CAN Highspeed, CAN Lowspeed und TTP. Für die Abstraktion wird Wissen über die Zusammenhänge bei Automatisierungsaufgaben und bei Einrichtungen zur Automatisierung vorausgesetzt. Bei den aufgeführten Beispielen sind dies folgende Zusammenhänge: CAN und TTP sind prozessnahe Kommunikationssysteme, ein prozessnahes Kommunikationssystem ist ein Kommunikationssystem und Ethernet ist ein Kommunikationssystem.

Bei der Abgrenzung potenziell mehrfach verwendbarer Teile ist es sinnvoll, zunächst nach ähnlichen Teilen zu suchen. War die Suche erfolgreich, kann eine geeignete Abgrenzung erfolgen. Wenn nicht, ist eine Generalisierung bzw. Abstraktion durchzuführen. Ist diese nicht möglich, ist die Suche erfolglos abzubrechen. Kann die Generalisierung bzw. Abstraktion durchgeführt werden, ist nach Teilen zu suchen, die den generalisierten bzw. abstrahierten Teilen ähnlich sind. Werden keine ähnlichen Teile gefunden, wird dieses Vorgehen wiederholt, bis keine Generalisierung bzw. Abstraktion mehr möglich ist. Das Ergebnis einer Generalisierung bzw. Abstraktion enthält immer die Einheit, nach der zuletzt gesucht wurde. Aufgrund der hierarchischen Beziehung wird das Ergebnis als hierarchische Einheit bezeichnet. Zur Vereinfachung wird ein potenziell mehrfach verwendbares Teil bereits ohne Generalisierung und Abstraktion als hierarchische Einheit bezeichnet. Durch dieses Vorgehen entstehen hierarchische Einheiten mit immer größerem Aufgabenfeld. Abbildung 5.2 zeigt einen Überblick über das beschriebene Vorgehen.

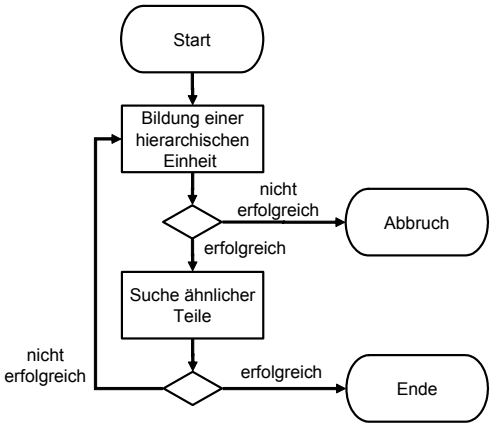


Abbildung 5.2: Vorgehen zur Abgrenzung potenziell mehrfach verwendbarer Teile

Wie in Abschnitt 5.1 diskutiert, beeinflusst ein zu großer Fokus bzw. Funktionsumfang die Anwendbarkeit (Usefulness) einer Softwarekomponente negativ, da damit ihre Flexibilität verloren geht. Eine konkrete Aussage über eine die Grenzen einer sinnvollen Abstraktion kann jedoch nicht getroffen werden. Die resultierende hierarchische Einheit kapselt die als potenziell nützlich erkannte Lösung und ist in vielen oder allen bestehenden Applikationen vorhanden. Ihre Schnittstellen und ihr Verhalten sind durch die gefundenen modularen Einheiten aus bestehenden Applikationen definiert.

5.3.4 Auswirkungen auf die Mehrfachverwendbarkeit

Die Erkennung und Abgrenzung potenziell mehrfach verwendbarer Teile basiert auf den Annahmen, die beim Domain-Engineering dazu dienen, Softwareeinheiten für die Mehrfachverwendung zu entwickeln. Unter Berücksichtigung der zweiten Annahme wird zunächst eine bestehende Lösung ausgewählt, die im Rahmen der aktuellen Applikation benötigt wird. Das Vorgehen stellt sicher, dass eine Softwarekomponente, welche diese Lösung abdeckt, in der

aktuellen Applikation einsetzbar ist. Es ersetzt jedoch keine Marktanalyse, die längerfristige Aussagen über den zukünftigen Bedarf trifft. Nach der Auswahl einer erkannten Lösung wird unter Berücksichtigung der ersten Annahme aus dem Domain-Engineering analysiert, ob ähnliche oder alternative Lösungen in vielen bestehenden Applikationen eingesetzt werden. Dabei werden für die Suche nach alternativen Lösungen hierarchische Einheiten gebildet. Können viele ähnliche bzw. alternative Lösungen gefunden werden, besitzen sie gemäß der ersten Annahme nach Abschnitt 2.2.2 Potenzial für die Mehrfachverwendung.

Das Potenzial für die Mehrfachverwendung der gefundenen Ergebnisse beschreibt deren Bedarf bei der Entwicklung zukünftiger Applikationen. Nach der Definition der Mehrfachverwendbarkeit nach [MMYA02] (vgl. Abschnitt 2.2.1) sind sie damit nützlich (useful), jedoch noch nicht anwendbar (usable). Es fehlen geeignete Konfigurationsmöglichkeiten zur Anpassung an unterschiedliche Anforderungen und Umgebungen. Diese werden bei der Entwicklung mehrfach verwendbarer Softwarekomponenten eingeführt und im folgenden Abschnitt vorgestellt. Die identifizierten Teile stellen Instanzen dar, aus denen gemäß Abschnitt 2.2.2 Typen zu entwickeln sind. Konfigurationsmöglichkeiten müssen auf Basis der Unterschiede dieser Instanzen entworfen und umgesetzt werden.

5.4 Entwicklung mehrfach verwendbarer Softwarekomponenten

In diesem Abschnitt werden mögliche Variabilitäten eines potenziell mehrfach verwendbaren Teils und der ihm zugeordneten Menge ähnlicher und alternativer Teile behandelt. Es wird erörtert, wie diese Variabilitäten bei der Analyse erkannt werden können, wie sie beim Entwurf zu abstrahieren sind und welche Möglichkeiten für die Implementierung bestehen.

5.4.1 Erkennung von Variabilitäten durch Domänenanalyse

In Abschnitt 3.5 wurde beschrieben, dass Unterschiede in Automatisierungssystemen durch Unterschiede in der Automatisierungsaufgabe sowie in den Einrichtungen, die zur Automatisierung erforderlich sind, verursacht werden. In Abbildung 5.3 sind beispielhaft zwei ähnliche Teile dargestellt, bei denen die Variabilitäten hervorgehoben sind. Einige der Variabilitäten haben nur lokale Auswirkungen und werden von modularen Einheiten gekapselt. Andere Variabilitäten wirken sich auf Schnittstellen aus und sind damit auch in der Softwarearchitektur sichtbar.

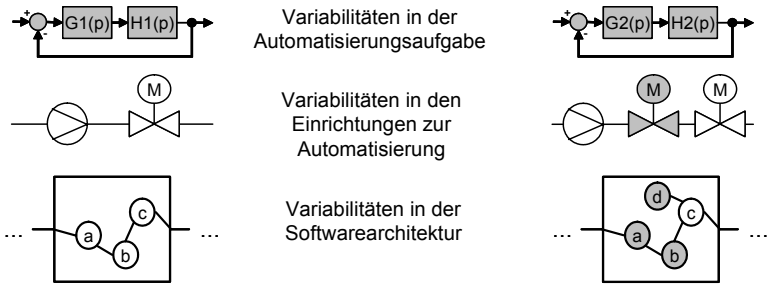


Abbildung 5.3: Beispiele für Variabilitäten potenziell mehrfach verwendbarer Teile

Beim Domain-Engineering wird zum Erkennen von Variabilitäten in der Aktivität Domänenanalyse (vgl. Kapitel 2) eine umfangreiche Sammlung von Informationen über die Merkmale der bestehenden Applikationen innerhalb der Domäne durchgeführt. Darauf aufbauend wird ein Domänenmodell erstellt, das die Unterschiede und Gemeinsamkeiten explizit darstellt. Die Analyse der Domäne ist bei den in Abschnitt 2.4 vorgestellten Methoden jeweils auf die Form der zu entwickelnden mehrfach verwendbaren Softwareeinheiten ausgerichtet. Für die Entwicklung domänenspezifischer Softwarearchitekturen hat sich die *merkmalorientierte Analyse* (feature-oriented Analysis) oder *Merkmalanalyse* (Feature Analysis) durchgesetzt. Wie in Abschnitt 2.4 erläutert, ergeben sich bei der Merkmalanalyse in Abhängigkeit von der Definition des Begriffes Merkmal unterschiedliche Ergebnisse. Bestehende Automatisierungssysteme sind deshalb bei der Domänenanalyse auch aus der Sicht der zu automatisierenden technischen Prozesse sowie aus der Sicht der zur Automatisierung eingesetzten Einrichtungen zu betrachten. Dadurch können die von modularen Einheiten realisierten Automatisierungsaufgaben bzw. Teilaufgaben, die dazu notwendige Synchronisation mit dem technischen Prozess sowie ihre Schnittstellen zu Sensoren und Aktoren als Merkmale erkannt werden. Bei der Domänenanalyse können auf diese Weise alle relevanten Informationen über bestehende Automatisierungssysteme berücksichtigen werden (vgl. Anschnitt 3.5).

Werden die erkannten Merkmale in ein Merkmaldiagramm eingetragen, unterscheidet sich dieses vom Merkmaldiagramm nach [KCH+90] bzw. [CzEi00] durch die Betrachtung der einzelnen Applikationen aus den genannten Sichten. Zur Unterscheidung können die einzelnen Sichten in das Merkmaldiagramm aufgenommen werden. Abbildung 5.4 zeigt beispielhaft die Übertragung von Merkmalen einer hierarchischen Einheit und einer ähnlichen modularen Einheit in ein Merkmaldiagramm. Die Merkmale weiterer modularer Einheiten werden ebenfalls in das Merkmaldiagramm eingetragen, das damit eine Übersicht über alle Unterschiede und Gemeinsamkeiten der hierarchischen Einheit und der Menge der modularen Einheiten bietet.

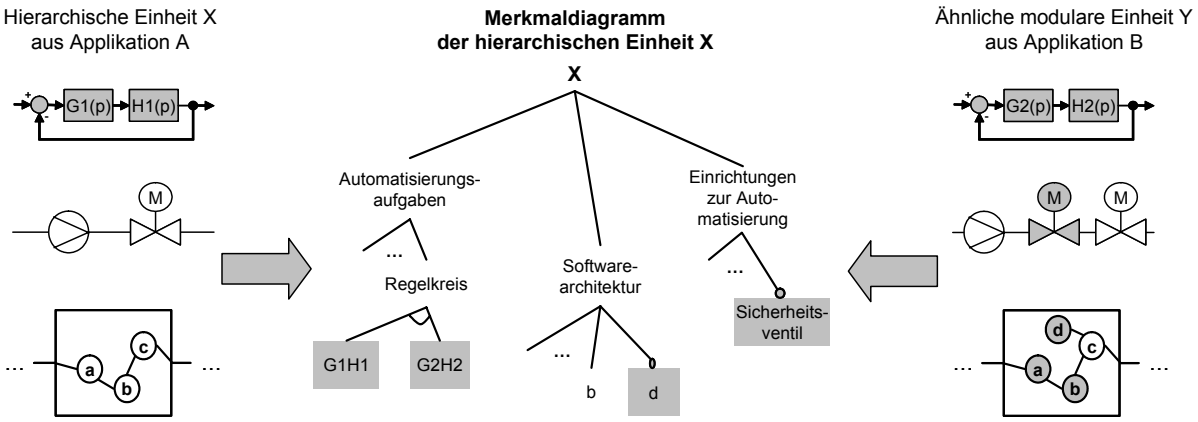


Abbildung 5.4: Merkmaldiagramm aus Teilen bestehender Applikationen

Wie in [Sahb02] gezeigt, hat das in Abschnitt 2.2.3 eingeführte Merkmaldiagramm den Nachteil, dass die entstehende Baumstruktur bei komplexen Domänen umfangreich werden kann und dadurch an Übersichtlichkeit verliert. Zur Lösung dieses Problems können einzelne Äste der Baumstruktur in separaten Diagrammen dargestellt werden.

Die erkannten Unterschiede sind nicht immer unabhängig voneinander. So kann beispielsweise das Vorhandensein eines Merkmals die Existenz eines anderen Merkmals bedingen. Bei der Regelung der Raumtemperatur für unterschiedlich große Räume können beispielsweise die Anzahl der einzusetzenden Temperatursensoren und Heizelemente sowie der Regelalgorithmus von der Raumgröße abhängen. Um diese Abhängigkeiten bei der Analyse einzubeziehen, müssen neben den Unterschieden auch ihre möglichen und nicht möglichen Kombinationen untersucht und dokumentiert werden. In FODA [KCH+90, LeKa04] wird vorgeschlagen, diese Abhängigkeiten in Form von Bedingungen grafisch in das Merkmaldiagramm einzutragen. Das führt bereits bei wenigen Abhängigkeiten zu einem unübersichtlichen Merkmaldiagramm [Sahb02]. Eine zusätzliche Matrix, bei der die Merkmale und deren Abhängigkeiten in Form von Bedingungen eingetragen werden, ist besser geeignet.

Durch die merkmalorientierte Analyse können mit Hilfe der zusätzlichen Sichten die Unterschiede und Gemeinsamkeiten, die aus der Automatisierungsaufgabe bzw. den Einrichtungen zur Automatisierung resultieren, untersucht und erkannt werden. Auf Basis der erkannten Variabilitäten sind Konfigurationsmöglichkeiten für die mehrfach verwendbaren Softwarekomponenten bzw. die domänenspezifische Softwarearchitektur zu entwerfen.

5.4.2 Abstraktion von Variabilitäten durch Domänenentwurf

In Abschnitt 2.2.1 werden mehrfach verwendbare Softwareeinheiten als generische Bausteine beschrieben, aus denen durch Konfiguration unterschiedliche Instanzen gebildet werden. Zur Entwicklung solch generischer Bausteine werden die Variabilitäten bestehender Instanzen abstrahiert und generalisiert [Krue92, Schm97]. Beim Domain-Engineering geschieht dies mit Hilfe der Informationen aus dem Domänenmodell. Zur Erstellung eines generischen Bausteins

sind die Unterschiede zwischen den entsprechenden Instanzen zu abstrahieren. Die Beziehungen der Unterschiede zur Domäne (mandatory, alternative, ...) müssen dabei berücksichtigt werden. Für die Abstraktion sind Konfigurationsmöglichkeiten einzuführen, die bei der Instanziierung das Ableiten einzelner Varianten aus einem generischen Baustein ermöglichen. Für die Realisierung der Konfigurationsmöglichkeiten stehen die in Abschnitt 2.2.3 beschriebenen Mechanismen zur Verfügung. Um zu erkennen, welche der Mechanismen auf welche Konstellation von Unterschieden und Gemeinsamkeiten anzuwenden sind, müssen ihre Auswirkungen für den Anwender betrachtet werden. Variationspunkte der domänenspezifischen Softwarearchitektur erlauben es dem Anwender, an einer Stelle in dieser Softwarearchitektur unterschiedliche Softwarekomponenten einzusetzen. Dazu werden mehrere Softwarekomponenten mit denselben Schnittstellen und einem alternativen Verhalten entwickelt. Dieser Mechanismus ermöglicht es, auch nachträglich weitere Alternativen durch die Entwicklung weiterer Softwarekomponenten mit entsprechenden Schnittstellen und einem alternativen Verhalten zu realisieren. Bestehende Softwarekomponenten müssen nicht geändert werden, was ggf. ihre Wartung und Pflege erleichtert. Die beiden Arten von Variationspunkten, die für Softwarekomponenten eingesetzt werden können, ermöglichen die Konfiguration ihrer internen Softwarearchitektur sowie ihrer Implementierung. Die internen Auswirkungen der Konfiguration können damit gegenüber dem Anwender verborgen werden. Die Variationspunkte der domänenspezifischen Softwarearchitektur eignen sich nicht zur Abstraktion kleiner Unterschiede, da sie Optionen und Alternativen in der Granularität von Softwarekomponenten einführen. Die Mechanismen der Softwarekomponenten können dagegen sowohl für die Abstraktion kleiner als auch großer Unterschiede eingesetzt werden.

Welche der Variationspunkte zur Abstraktion konkreter Variabilitäten eingesetzt werden, ist im Rahmen des Entwurfs der mehrfach verwendbaren Softwarekomponenten zu entscheiden. Unterscheiden sich große Teile der hierarchischen Einheit und der Menge ähnlicher und alternativer modularer Einheiten, ist der Aufwand für die Modellierung eines gemeinsamen Kerns groß und der zu erwartende Nutzen klein. In einem solchen Fall sollte ein Variationspunkt gebildet werden, der die Auswahl unterschiedlicher Softwarekomponenten zur Realisierung dieses Teils zulässt. Unterscheiden sich nur kleine Teile, ist die Kapselung in einer Softwarekomponente vorzuziehen. Die Anpassung erfolgt in diesem Fall über die Konfigurationsmöglichkeiten der Softwarekomponente.

Das Merkmaldiagramm bietet eine detaillierte Beschreibung von Variabilitäten und bildet den Ausgangspunkt für die Entwicklung von Konfigurationsmöglichkeiten. Die Variabilitäten sind zu abstrahieren, wozu ein geeigneter Mechanismus für die Konfiguration entworfen werden muss. Sind keine bis wenige Gemeinsamkeiten erkennbar, ist der Einsatz eines Variationspunktes der domänenspezifischen Softwarearchitektur in Betracht zu ziehen. Weniger umfangreiche Unterschiede können mit den zur Verfügung stehenden Mechanismen der Softwarekomponenten umgesetzt werden. Dadurch entstehen sowohl Konfigurations-

möglichkeiten für die Softwarekomponenten als auch für die domänenspezifische Softwarearchitektur. Abbildung 5.5 zeigt die beschriebenen Zusammenhänge an einem Beispiel.

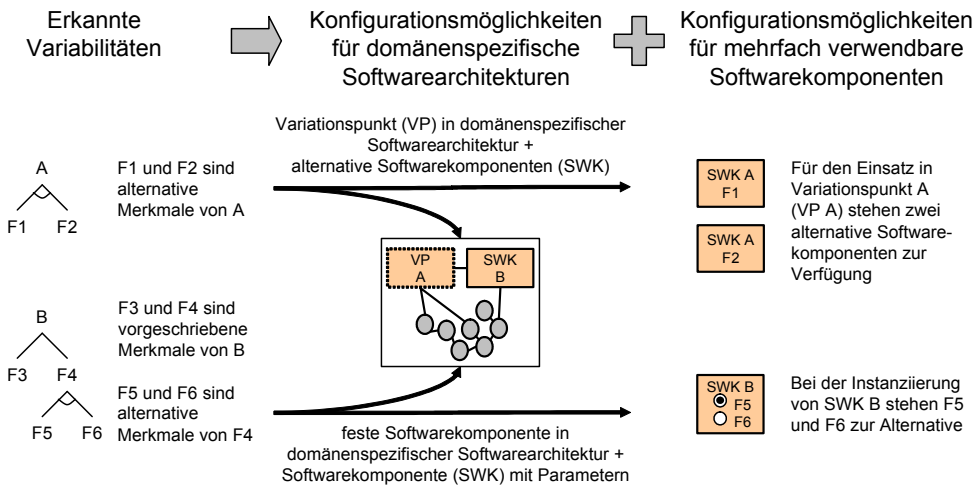


Abbildung 5.5: Auswirkungen der Anwendung verschiedener Mechanismen zur Umsetzung von Konfigurationsmöglichkeiten

Der Zeitpunkt der Entscheidung für eine bestimmte Variante (Konfiguration) wird als *Binding-Time* bezeichnet [CzEi00]. Typische Beispiele für Zeitpunkte zur Bindung von Varianten sind „vor der Übersetzung“ (Compile-Time) und „während der Laufzeit“ (Run-Time). Die zur Realisierung von Variationspunkten eingesetzten Mechanismen sind abhängig von den beschriebenen Zeitpunkten. Die bei objektorientierten Softwarearchitekturen eingesetzte Vererbung ermöglicht beispielsweise eine Bindung von Varianten während der Laufzeit. Um eine Bindung vor der Übersetzung zu ermöglichen, kann bedingte Vererbung mit Hilfe von Template-Klassen [Stro00] eingesetzt werden. Dabei werden nur die Klassen übersetzt, die vor der Übersetzung durch Parameter ausgewählt wurden.

Beim Entwurf einzelner Automatisierungssysteme sind insbesondere parallel ablauffähige Vorgänge, die notwendige Synchronisierung, die Beziehungen der Vorgänge untereinander (Schnittstellen und Daten) und die Beziehungen zur Umwelt (externe Ereignisse, externe Schnittstellen) zu beschreiben [Göhn05b] (vgl. Abschnitt 3.4). Beim Entwurf von Variationspunkten für Automatisierungssysteme müssen diese Aspekte ebenfalls berücksichtigt werden.

Insbesondere beim Einsatz von Mikrocontrollern als Automatisierungscomputer spielt der Speicherbedarf der Software eine wichtige Rolle (Hardwarekosten). Durch die Konfiguration vor der Übersetzung kann bei solchen Systemen der Speicherbedarf optimiert werden, da nur Platz für die gewählten Varianten benötigt wird [KLW+96].

5.4.3 Auswirkungen auf die Mehrfachverwendbarkeit

Das Ergebnis der Abstraktion von Variabilitäten sind Konfigurationsmechanismen für Softwarekomponenten und die domänenspezifische Softwarearchitektur. Sie werden ausgehend von den

in Abschnitt 5.3 als nützlich (useful) identifizierten Teilen bestehender Applikationen entwickelt. Durch die Abstraktion ihrer Variabilitäten entstehen Konfigurationsmöglichkeiten, die es erlauben, die Softwarekomponenten und die domänenspezifische Softwarearchitektur bei der Instanziierung an unterschiedliche Anforderungen anzupassen. Das macht sie für den Einsatz bei der Entwicklung neuer Applikationen einfach anwendbar (usable). Nach [MMYA02] (vgl. Abschnitt 2.2.1) wird die Mehrfachverwendbarkeit von Software über ihre Nützlichkeit (Usability) und ihre Anwendbarkeit (Usefulness) bestimmt. Die mehrfach verwendbaren Softwarekomponenten und die domänenspezifische Softwarearchitektur werden so entworfen, dass sie sowohl nützlich als auch anwendbar sind. Deshalb ist, soweit es nach dem Kenntnisstand bei der Entwicklung beurteilt werden kann, eine gute Mehrfachverwendbarkeit zu erwarten. Kommen in Zukunft Anforderungen an die Domäne dazu oder ändern sich bestehende Anforderungen, wird dies die Mehrfachverwendbarkeit der Softwarekomponenten beeinflussen.

Durch die Berücksichtigung der Automatisierungsaufgabe sowie der zur Automatisierung eingesetzten Einrichtungen können bei der Domänenanalyse die für Automatisierungssysteme relevanten Unterschiede und Gemeinsamkeiten ermittelt werden. Diese bilden die Grundlage für die Entwicklung mehrfach verwendbarer Softwarekomponenten und einer domänenspezifischen Softwarearchitektur für den Einsatz in Automatisierungssystemen.

Bei der Entwicklung von Softwarekomponenten werden Zusammenhänge innerhalb der identifizierten hierarchischen Einheit berücksichtigt. Verknüpfungen zwischen den entstehenden Softwarekomponenten und die daraus resultierende Struktur sind nicht erkennbar. Deshalb ist eine Integration der Softwarekomponenten zu einer domänenspezifischen Softwarearchitektur, wie in Abschnitt 4.3.3 erläutert, durchzuführen.

5.5 Integration der Softwarekomponenten in die domänenspezifische Softwarearchitektur

5.5.1 Aktuelle Generation der domänenspezifischen Softwarearchitektur

Die domänenspezifische Softwarearchitektur besteht gemäß Abschnitt 5.3 aus einem bereits aus mehrfach verwendbaren Softwarekomponenten aufgebauten Teil und einem noch nicht bearbeiteten Teil. Zur Integration einer neuen Softwarekomponente ist ihr Platz innerhalb des noch nicht bearbeiteten Teils zu bestimmen und ihre Verknüpfungen zu bestehenden Softwarekomponenten sowie zu noch nicht bearbeiteten modularen Einheiten sind festzulegen.

5.5.2 Platzierung und Verknüpfung neuer Softwarekomponenten

Die Integration einer neuen Softwarekomponente erfolgt durch das Ersetzen einer modularen Einheit im noch nicht bearbeiteten Rest der domänenspezifischen Softwarearchitektur. Enthält dieser Rest eine modulare Einheit, die einer der Instanzen entspricht, auf deren Basis die Softwarekomponente entwickelt wurde, muss diese für mindestens eine Konfiguration kompatible Schnittstellen besitzen. Enthält der Rest keine solche modulare Einheit, müssen die Position und die Verknüpfungen der Softwarekomponente ggf. von bestehenden Applikationen abgeleitet werden, die eine entsprechende modulare Einheit enthalten. Die vorhandenen Verknüpfungen zu bestehenden Softwarekomponenten oder dem noch nicht bearbeiteten Teil der domänenspezifischen Softwarearchitektur werden in beiden Fällen bei der Integration angepasst. Wurde beim Entwurf der Softwarekomponenten ein Variationspunkt für die domänenspezifische Softwarearchitektur definiert, ist er bei der Integration zu berücksichtigen.

Bei der Integration der Softwarekomponenten sind, wie beim Domänenentwurf (vgl. Abschnitt 5.4.2), parallel ablauffähige Vorgänge, die notwendige Synchronisierung sowie die Beziehungen der Vorgänge untereinander und zur Umwelt zu berücksichtigen.

5.5.3 Abhängigkeiten zwischen Konfigurationsmöglichkeiten

Werden die resultierenden Softwarekomponenten in der Applikationsentwicklung eingesetzt, können sie mit Hilfe der entwickelten Konfigurationsmechanismen angepasst werden. Dabei sind vielfältige Kombinationsmöglichkeiten einzelner Parameter denkbar. Aus der Softwarekomponente B in Abbildung 5.5 können beispielsweise zwei unterschiedliche Instanzen gebildet werden. Ergeben sich Unterschiede in den Schnittstellen oder dem Verhalten, stellt sich die Frage, ob jede dieser Instanzen mit allen Konfigurationen des Variationspunktes A kompatibel ist bzw. zu den geforderten systemweiten Eigenschaften führt. Prinzipiell sind Konfigurationen möglich, die bei der Entwicklung einer einzelnen Softwarekomponente nicht absehbar sind und zu undefinierten Instanzen führen. Zur Beherrschung dieser Problematik ist die Dokumentation gültiger Konfigurationen bei der Entwicklung der Konfigurationsmöglichkeiten notwendig. Das sind insbesondere die, welche zu den ursprünglichen Instanzen führen, aus denen die Softwarekomponenten abstrahiert wurden. Eine vollständige Darstellung der domänenspezifischen Softwarearchitektur muss deshalb neben der Struktur von Softwarekomponenten auch die Dokumentation gültiger und ungültiger Konfigurationen enthalten. Die dazu notwendigen Informationen können in einer Matrix gespeichert werden. Deren Inhalt wird aus den Abhängigkeiten der Merkmale nach Abschnitt 5.4.1 abgeleitet.

5.5.4 Auswirkungen auf die Mehrfachverwendbarkeit

Zusammen mit einer iterativen Vorgehensweise unterstützt die vorgestellte Möglichkeit zur Integration der Softwarekomponenten die schrittweise Erweiterung der domänenspezifischen

Softwarearchitektur. Ausgehend von der Struktur und den modularen Einheiten, die bei der Erstellung der domänenspezifischen Softwarearchitektur aus einer bestehenden Applikation übernommen werden, entsteht eine erweiterte Struktur basierend auf Softwarekomponenten, die nach und nach den unbearbeiteten Rest ablösen. Dieser Vorgang ist in Abbildung 5.6 beispielhaft dargestellt.

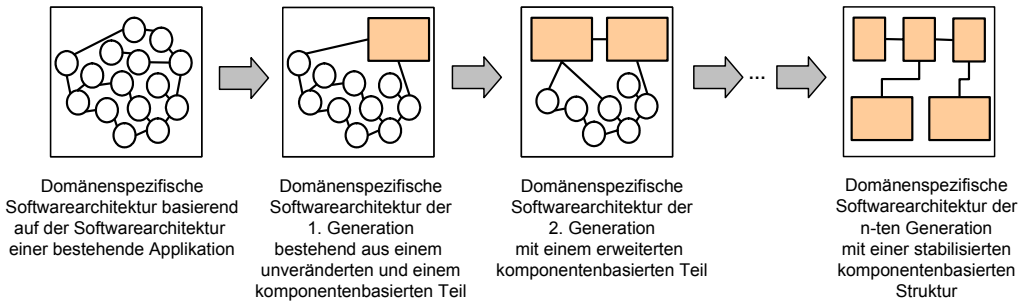


Abbildung 5.6: Evolution der domänenspezifischen Softwarearchitektur

Als Ergebnisse der evolutionären Entwicklung für die Mehrfachverwendung werden eine stabile domänenspezifische Softwarearchitektur und eine Menge dazu passender mehrfach verwendbarer Softwarekomponenten angestrebt (vgl. Abbildung 5.6). Die Entwicklung für die Mehrfachverwendung basiert auf den beiden in Abschnitt 2.2 erläuterten Annahmen. Eine Garantie für deren tatsächliche Mehrfachverwendbarkeit ist dadurch nicht gegeben. Die Entwicklung des Marktes, die Veränderung von Anforderungen sowie des zu automatisierenden technischen Prozesses, die Entwicklung neuer Technologien, die Änderungen von Rechnerhardware, Prozessperipherie, usw. beeinflussen die Mehrfachverwendbarkeit der Ergebnisse. Um auf diese Einflüsse durch Korrekturen der bestehenden Ergebnisse reagieren zu können, muss die Mehrfachverwendbarkeit ständig überwacht und gewartet werden.

5.6 Überprüfung der Mehrfachverwendbarkeit bestehender Ergebnisse

5.6.1 Ziele der Überprüfung

Aufgrund von Veränderungen im Umfeld einer Domäne kann die Mehrfachverwendbarkeit der Softwarekomponenten und der domänenspezifischen Softwarearchitektur beeinträchtigt werden. Um gemäß Kapitel 4 die betroffenen Ergebnisse der Entwicklung aussortieren oder verbessern zu können, muss die Überprüfung der Mehrfachverwendbarkeit die Möglichkeit bieten, Mängel zu erkennen. Die Ergebnisse sind die Grundlage für die Entscheidung über das weitere Vorgehen innerhalb eines Iterationsschrittes.

5.6.2 Messverfahren zur Bestimmung der Mehrfachverwendbarkeit

Wie im vorherigen Abschnitt dargestellt reicht es für die Entscheidung über das weitere Vorgehen aus, einen Mangel an Mehrfachverwendbarkeit zu erkennen. Ein Mangel an Mehrfachverwendbarkeit liegt dann vor, wenn mehrfach verwendbare Softwarekomponenten oder die domänenspezifische Softwarearchitektur bei der Entwicklung neuer Applikationen nicht eingesetzt werden können. Eine quantitative Aussage über den Umfang des Mangels bzw. über die Mehrfachverwendbarkeit ist nicht notwendig. Es wird deshalb keine aufwändige Überprüfung über Metriken und Qualitätsmerkmale gebraucht, wie sie beispielsweise in [Poul94], [Karl95], [DaRi02] oder [WYF03] vorgeschlagen werden.

Für die Verbesserung der Mehrfachverwendbarkeit ist es hilfreich zu wissen, ob es sich um einen Mangel an Nützlichkeit (Usefulness) oder um einen Mangel an Anwendbarkeit (Usability) handelt. Abbildung 5.7 zeigt die Einflussgrößen, die sich negativ auf Nützlichkeit und Anwendbarkeit auswirken und somit die Mehrfachverwendbarkeit negativ beeinflussen. Ein Mangel an Nützlichkeit liegt vor, wenn der Funktionsumfang bei der Entwicklung der aktuellen Applikation nicht benötigt wird. Wird stattdessen eine andere Funktionalität benötigt, ist diese Information für die anstehende Verbesserung ebenfalls wichtig. Sind die bestehenden Ergebnisse nützlich, jedoch nicht anwendbar, bedeutet dies, dass sie mit den verfügbaren Konfigurationsmöglichkeiten nicht an die gestellten Anforderungen und Rahmenbedingungen anpassbar sind. Gründe dafür, dass die Bildung einer passenden Instanz nicht möglich ist, können ungenügende Kombinationsmöglichkeiten von Softwarekomponenten oder nicht erfüllbare neue Anforderungen, neue Automatisierungsaufgaben, neue Einrichtungen zur Automatisierung, usw. sein. In jedem Fall sind für die Entscheidung über die Durchführung von Verbesserungen Informationen zu den Gründen notwendig, die eine Anwendung verhindern. Werden die genannten Informationen bei jedem erfolgreichen und nicht erfolgreichen Einsatz der mehrfach verwendbaren Ergebnisse gesammelt, können Entwickler sowohl bei anstehenden Entscheidungen als auch bei ggf. notwendigen Verbesserungen darauf zurückgreifen.

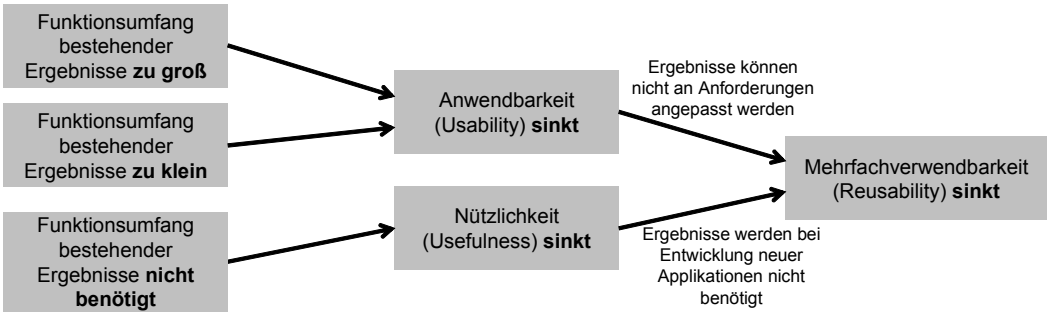


Abbildung 5.7: Einflussgrößen zur negativen Beeinflussung der Mehrfachverwendbarkeit

Für die Sammlung von Informationen über nicht berücksichtigte Anforderungen, Automatisierungsaufgaben und Einrichtungen zur Automatisierung sowie fehlende Konfigurationsmöglichkeiten bietet sich die Führung einer Statistik über die Einsätze der domänenspezifischen

Softwarearchitektur und der bestehenden Softwarekomponenten an. Wird eine schlechte Mehrfachverwendbarkeit der domänenspezifischen Softwarearchitektur sowie der mehrfach verwendbaren Softwarekomponenten festgestellt, ist eine Verbesserung bezüglich dieser Eigenschaft notwendig. Die Ursachen der schlechten Mehrfachverwendbarkeit sind festzustellen und mit geeigneten Maßnahmen zu beheben.

5.7 Verbesserung der Mehrfachverwendbarkeit von Ergebnissen

5.7.1 Ursachen mangelnder Mehrfachverwendbarkeit

Gemäß Abschnitt 2.2.1 beschreibt die Nützlichkeit (Usefulness) die Nachfrage nach mehrfach verwendbaren Ergebnissen bei der Entwicklung zukünftiger Applikationen. Diese Nachfrage ist abhängig von den Anforderungen, die an eine neue Applikation gestellt werden. Bieten die bestehenden Ergebnisse eine geeignete Realisierung zu den gestellten Anforderungen, können sie eingesetzt werden und der Bedarf ist gedeckt. Der Nutzen wird zum einen durch den gegebenen Funktionsumfang und zum anderen durch die Möglichkeiten zur Kombination von Parametern und Softwarekomponenten bestimmt. Ergibt sich durch veränderte Anforderungen oder Rahmenbedingungen zukünftiger Applikationen Bedarf an weiteren Varianten, kann dieser durch die bestehenden Ergebnisse nicht gedeckt werden.

Die Anwendbarkeit (Usability) mehrfach verwendbarer Ergebnisse beschreibt deren Möglichkeiten zur Anpassung an gegebene Anforderungen und Rahmenbedingungen. Die Anpassung geschieht mit Hilfe von Konfigurationsmöglichkeiten. Der Funktionsumfang, den die Ergebnisse bereitstellen, spielt eine wesentliche Rolle. Ist er zu groß und kann nicht durch Konfiguration skaliert werden, sind die mehrfach verwendbaren Ergebnisse nicht für den Einsatz in dieser Applikation geeignet. Das ist insbesondere für Automatisierungssysteme mit begrenzten Ressourcen ein Problem. Ist der Funktionsumfang zu klein, können die Ergebnisse nicht ohne Ergänzungen eingesetzt werden. Das ist der Fall, wenn die bestehenden Ergebnisse beispielsweise nicht alle an sie gestellten Anforderungen erfüllen, nicht mit einer neuen Prozess-peripherie betrieben werden können oder nicht für die Synchronisation mit neuen Vorgängen im technischen Prozess vorbereitet sind.

Die Ursache für einen Mangel an Mehrfachverwendbarkeit kann auch eine Kombination der diskutierten Ursachen für mangelnde Nützlichkeit und mangelnde Anwendbarkeit sein.

5.7.2 Behebung der Ursachen durch Erweiterung bestehender Ergebnisse

Den in Abschnitt 5.7.1 beschriebenen Ursachen mangelnder Mehrfachverwendbarkeit ist gemein, dass es nicht möglich ist, mit den bestehenden Ergebnissen eine Instanz zu bilden, die für die Entwicklung der aktuellen Applikation nützlich ist. Zur Verbesserung der Mehrfachverwendbarkeit werden die bestehenden Ergebnisse um die Möglichkeit zur Bildung der benötigten Instanz erweitert. Dazu wird zunächst eine neue Instanz mit den geforderten Eigenschaften entwickelt. Durch die anschließende Integration der neuen Instanz in die bestehenden Ergebnisse wird die Mehrfachverwendbarkeit der Ergebnisse verbessert. Das zeigt sich zum einen durch die erweiterte Anwendbarkeit und zum anderen durch die hinzugewonnene Nützlichkeit bei der Entwicklung der aktuellen Applikation.

Die zu entwickelnde Instanz muss zum einen die neuen Anforderungen bei der Entwicklung der aktuellen Applikationen erfüllen und zum anderen muss sie in die bestehenden Ergebnisse integrierbar sein. Eine Möglichkeit, eine Instanz mit Gemeinsamkeiten zu entwickeln, ist die Anwendung der bereits in Abschnitt 2.1 erläuterten Ad-hoc-Wiederverwendung. Dazu wird eine Instanz auf Basis der bestehenden Ergebnisse gebildet, welche möglichst viele der neuen Anforderungen erfüllt und damit einen Teil der gesuchten Instanz realisiert. Aus dieser wird dann durch Anpassung gemäß den gestellten Anforderungen die neue Instanz entwickelt. Nach der Entwicklung der neuen Instanz ist diese in die bestehenden mehrfach verwendbaren Ergebnisse zu integrieren. Die Instanz muss durch Konfiguration aus den erweiterten Ergebnissen gebildet werden können. Dieses Ziel wird auch von der in Abschnitt 5.4.2 vorgestellten Abstraktion der Variabilitäten verfolgt. Es liegt nahe, das dort vorgestellte Vorgehen auch zur Integration der neuen Instanz einzusetzen. Dazu müssen die aus der neuen Instanz resultierenden Unterschiede und Gemeinsamkeiten gegenüber den bestehenden Ergebnissen erkannt werden. Hier kommt die Erkennung von Variabilitäten durch die Merkmalanalyse aus Abschnitt 5.4.1 zur Anwendung. Anschließend sind die erkannten Variabilitäten, wie in Abschnitt 5.4.2 beschrieben, zu abstrahieren. Das Resultat sind eine domänenspezifische Softwarearchitektur und Softwarekomponenten, die um die Möglichkeit zur Bildung weiterer Instanzen ergänzt wurden. Abbildung 5.8 zeigt das beschriebene Vorgehen an einem Beispiel.

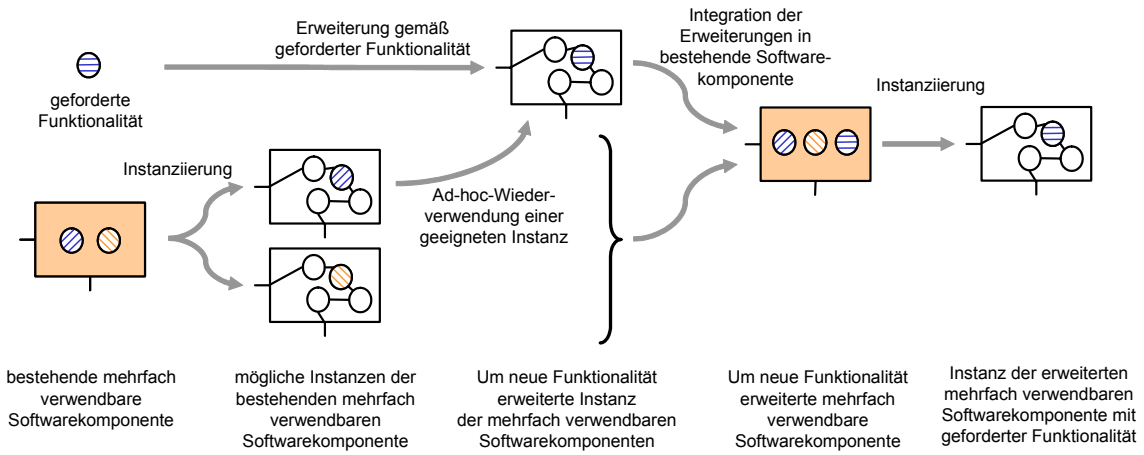


Abbildung 5.8: Erweiterung bestehender Ergebnisse mit Hilfe der Ad-hoc-Wiederverwendung am Beispiel einer mehrfach verwendbaren Softwarekomponente

Um eine Verbesserung der Mehrfachverwendbarkeit durchführen zu können, reicht es aus, einen Mangel an Mehrfachverwendbarkeit rechtzeitig vor dem Einsatz der domänenspezifischen Softwarearchitektur und der Softwarekomponenten zu erkennen. Dann ist es möglich, diesen Mangel noch vor dem Einsatz dieser Ergebnisse zu beheben. Rechtzeitig heißt in diesem Fall, dass der Zeitraum zwischen dem Erkennen des Mangels und dem geplanten Einsatz der mehrfach verwendbaren Ergebnisse für die Durchführung der Verbesserung ausreichen muss. Ein Mangel kann erst erkannt werden, wenn die Anforderungen für eine neue Applikation feststehen. Da dies bei der Applikationsentwicklung während der Definitions-Phase geschieht, kann die Abschätzung der Mehrfachverwendbarkeit frühestens nach dieser Aktivität erfolgen. Der Einsatz der domänenspezifischen Softwarearchitektur und der Softwarekomponenten erfolgt spätestens in der Integrations-Phase der Applikationsentwicklung. So entstehen für die Überprüfung der Mehrfachverwendbarkeit sowie für die ggf. durchzuführende Verbesserung dieselben Rahmenbedingungen für die Synchronisation mit der Applikationsentwicklung, wie für die Identifikation potenziell mehrfach verwendbarer Teile und deren Umsetzung in mehrfach verwendbare Softwarekomponenten (vgl. Abschnitt 5.3.2). Deshalb sind die gleichen Maßnahmen zur Synchronisation auch für die Überprüfung und die Verbesserung der Mehrfachverwendbarkeit notwendig (vgl. Abschnitt 5.3.2).

5.7.3 Auswirkungen auf die Mehrfachverwendbarkeit

Beim Einsatz der mehrfach verwendbaren Softwarekomponenten und der domänenspezifischen Softwarearchitektur können Mängel der Mehrfachverwendbarkeit auftreten, die durch die Überprüfung der Mehrfachverwendbarkeit nach Abschnitt 5.6 festgestellt werden. Mögliche Ursachen dafür sind mangelnde Nützlichkeit (Usefulness) bzw. unzureichende Anwendbarkeit (Usability) der betroffenen Ergebnisse. Die Ursachen können auf fehlende Möglichkeiten zur Bildung einer benötigten Instanz zurückgeführt werden. An diese Instanz werden neue Anforderungen gestellt, die bei der Entwicklung der mehrfach verwendbaren Ergebnisse nicht

berücksichtigt wurden. Um die betreffenden Ergebnisse weiter einsetzen zu können, wurden Maßnahmen zur Verbesserung der Ergebnisse beschrieben. Diese enthalten die Entwicklung einer Instanz, welche die neuen Anforderungen erfüllt. Die neue Instanz wird anschließend in die bestehenden mehrfach verwendbaren Ergebnisse integriert. Aus den verbesserten Ergebnissen kann nun eine Instanz gebildet werden, welche die neuen Anforderungen erfüllt. Damit ist die Mehrfachverwendbarkeit der Ergebnisse wieder hergestellt.

Der tatsächliche Erfolg der Verbesserung zeigt sich während des Einsatzes bei der Entwicklung zukünftiger Applikationen. Auch wenn die Mehrfachverwendbarkeit wieder hergestellt wurde, kann sich diese durch die Veränderung von Anforderungen und Rahmenbedingungen wieder verschlechtern. Zur dauerhaften Erhaltung der Mehrfachverwendbarkeit ist deshalb eine ständige Überwachung und Pflege der mehrfach verwendbaren Ergebnisse erforderlich.

5.8 Auswirkungen des Konzeptes

In Kapitel 4 wurden Anforderungen und Rahmenbedingungen für die Anwendung des Evolutionsprinzips beim Domain-Engineering erarbeitet. Auf dieser Basis entstand im vorliegenden Kapitel ein Konzept zur iterativen Entwicklung von mehrfach verwendbaren Softwareeinheiten für Automatisierungssysteme.

Mit der Identifikation potenziell mehrfach verwendbarer Teile vor der Entwurfs-Phase wird die Forderung nach einer frühen Zerlegung der Domäne erfüllt. Sie ermöglicht die separate Entwicklung der entstehenden Teile. Das ist eine Voraussetzung für das iterative Vorgehen. Die resultierenden Teile sind, soweit dies zum Zeitpunkt ihrer Entwicklung beurteilt werden kann, für die Entwicklung neuer Applikationen nützlich (useful) und durch ihre Konfigurations-möglichkeiten einfach anwendbar (usable).

Bei der Domänenanalyse werden insbesondere die Automatisierungsaufgaben und die zur Automatisierung eingesetzten Einrichtungen berücksichtigt. Das ermöglicht die Analyse der für Automatisierungssysteme relevanten Unterschiede und Gemeinsamkeiten. Damit erfüllt das Konzept eine wichtige Voraussetzung zur Anwendung für Automatisierungssysteme.

Vor dem Einsatz bestehender Ergebnisse wird ihre Mehrfachverwendbarkeit überprüft. Das ermöglicht eine schnelle Reaktion auf die Verschlechterung der Mehrfachverwendbarkeit. Zur Verbesserung werden die bestehenden Ergebnisse um Möglichkeiten zur Bildung neuer, bisher nicht berücksichtigter Instanzen erweitert. Das stellt ihre Mehrfachverwendbarkeit wieder her.

Das vorgestellte Konzept bietet die Voraussetzungen für die iterative Entwicklung mehrfach verwendbarer Softwarekomponenten und einer domänenspezifischen Softwarearchitektur für Automatisierungssysteme. Entwickler benötigen zur Anwendung des Konzeptes Unterstützung und Anleitung für ein strukturiertes Vorgehen. In Kapitel 6 wird dazu eine Methode eingeführt, die auf dem vorgestellten Konzept basiert.

6 Methodik zur Anwendung des iterativen Konzepts

Im vorliegenden Kapitel wird eine Methodik erarbeitet, die beschreibt, welche Aktivitäten ein Entwickler durchführen muss, um das Konzept aus Kapitel 5 anzuwenden. Die Methodik berücksichtigt das Evolutionsprinzip gemäß Kapitel 4 und ermöglicht die Einbeziehung der spezifischen Eigenschaften von Automatisierungssystemen. Es werden zunächst Merkmale von Methodiken zur Softwareentwicklung erläutert. Anschließend wird eine Methodik zur Anwendung des iterativen Konzeptes unter Berücksichtigung dieser Merkmale beschrieben.

6.1 Merkmale von Methodiken zur Softwareentwicklung

Eine Methodik ist eine Anweisung zur methodischen, das heißt zur folgerichtigen und zweckmäßigen Lösung einer Aufgabe [Chro92]. Sind dazu umfangreiche Tätigkeiten notwendig, ist eine Aufteilung der Anweisung in überschaubare Aktivitäten sinnvoll. In diesem Fall ist für jede Aktivität zu beschreiben, welche Produkte zu deren Durchführung benötigt werden (Eingangsprodukte) und welche Produkte entstehen oder verändert werden (Ausgangsprodukte). Umfangreiche Aktivitäten und Produkte können in Teilaktivitäten und Teilprodukte zerlegt werden. Die logische Folge von Aktivitäten und Produkten ergibt einen Produktfluss [BrDr93]. Zur Beschreibung des Produktflusses wird im Folgenden die in Abbildung 6.1 dargestellte Notation eingesetzt. Der Produktfluss folgt einem Vorgehensmodell, wie beispielsweise dem Wasserfall-Modell oder dem Spiral-Modell, das die Reihenfolge des Arbeitsablaufes festgelegt.

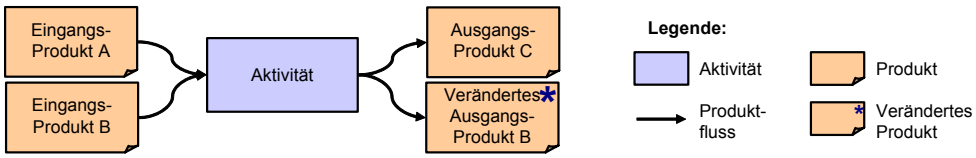


Abbildung 6.1: Notation zur Darstellung des Produktflusses einer Methodik

Ein Teil der Ausgangsprodukte der Aktivitäten sind Bestandteil der Dokumentation. Bei der Dokumentation ist zwischen Systemdokumentation und Benutzungsdokumentation zu unterscheiden. Die Systemdokumentation richtet sich an Entwickler und Personen, die mit der Wartung und Pflege der Software betraut sind. Sie beinhaltet Ergebnisse aus Analyse, Entwurf und Implementierung. Die Benutzungsdokumentation richtet sich an die Benutzer der Software. Sie beinhaltet Informationen zur Installation und zur Benutzung der Software [Göhn04].

6.2 Methodik des evolutionären Domain-Engineering

Die Aktivitäten werden unter Berücksichtigung des in Kapitel 4 eingeführten Evolutionsprinzips sowie dem in Kapitel 5 beschriebenen Konzept gestaltet. Die in Abschnitt 4.3.3 eingeführten

Schritte einer einzelnen Iteration dienen als Ausgangspunkt für die Beschreibung des Konzeptes in Kapitel 5 und bilden die Basis bei der Festlegung der Aktivitäten. Demnach kann im Rahmen eines Evolutionsschrittes parallel zur Applikationsentwicklung die Entwicklung mehrfach verwendbarer Softwareeinheiten oder die Verbesserung bestehender Ergebnisse erfolgen.

6.2.1 Aktivitäten und Produkte

In den Abschnitten 5.3 bis 5.5 wird die Entwicklung mehrfach verwendbarer Softwareeinheiten in drei Schritten beschrieben. Im ersten Schritt werden potenziell nützliche Teile erkannt (vgl. Abschnitt 5.3). Aus der Menge der erkannten Teile sind eines oder mehrere auszuwählen, um darauf aufbauend Softwarekomponenten zu entwickeln. Dabei handelt es sich um Tätigkeiten der Analyse, der Planung und des Entwurfs der domänenspezifischen Softwarearchitektur. Die Tätigkeiten werden drei Aktivitäten zugeordnet. Aus der Erkennung potenziell nützlicher Teile wird die Aktivität „**Mehrfachverwendbarkeitsanalyse**“ abgeleitet. Die Auswahl erkannter Teile wird in der Aktivität „**Planung des Evolutionsschrittes**“ behandelt. Das Ergebnis dieser Aktivität ist unter anderem eine Auswahl von Lösungen, auf deren Basis eine oder mehrere Softwarekomponenten entwickelt werden sollen. Da Softwarekomponenten das Ziel der Entwicklung darstellen, wird aus der Abgrenzung potenziell mehrfach verwendbarer Teile die Aktivität „**Identifikation potenzieller Softwarekomponenten**“. Ihr Ausgangsprodukt ist eine Liste bestehender ähnlicher Instanzen. Auf deren Basis erfolgt die Entwicklung einer mehrfach verwendbaren Softwarekomponente. Dabei werden zunächst Softwarekomponenten erstellt und anschließend in die domänenspezifische Softwarearchitektur integriert (vgl. Abschnitt 5.4). Die notwendigen Tätigkeiten werden in den Aktivitäten „**Entwicklung mehrfach verwendbarer Softwarekomponenten**“ und „**Integration der Softwarekomponenten**“ zusammengefasst. Die mehrfach verwendbaren Ergebnisse stehen am Ende eines Evolutionsschrittes in der jeweils aktuellen Generation für die Entwicklung neuer Applikationen zur Verfügung. Die tatsächliche Mehrfachverwendbarkeit dieser Ergebnisse ist bei jedem Einsatz zu überprüfen (vgl. Abschnitt 5.6). Dies erfolgt während der Aktivität „**Mehrfachverwendbarkeitsanalyse**“. Wird bei der Überprüfung eine mangelhafte Mehrfachverwendbarkeit festgestellt, sind die bestehenden Softwarekomponenten und die domänenspezifische Softwarearchitektur zu verbessern (vgl. Abschnitt 5.7). Dazu wird die Aktivität „**Verbesserung der Mehrfachverwendbarkeit**“ eingeführt. Nach der „Mehrfachverwendbarkeitsanalyse“ steht ab dem 2. Evolutionsschritt die Entscheidung an, ob im Rahmen des aktuellen Evolutionsschrittes eine oder mehrere neue Softwarekomponenten erstellt werden sollen oder ob bestehende Ergebnisse zu verbessern sind. Diese Tätigkeiten werden in die Aktivität „**Planung des Evolutionsschrittes**“ integriert. Die Aktivitäten sind zur Entwicklung einer einzelnen Softwarekomponente ausgelegt. Für die Erweiterung der Domäne um mehrere neue Softwarekomponenten müssen sie mehrmals durchlaufen werden. Tabelle 6.1 gibt einen Überblick über die genannten Aktivitäten sowie ihre Eingangs- und Ausgangsprodukte.

Tabelle 6.1: Aktivitäten und Produkte der Methodik

Aktivität	**Eingangsprodukte**	**Ausgangsprodukte**
Identifikation potenzieller Software-komponenten	Evolutionsschrittplanung, bestehende Applikationen	Liste bestehender Instanzen
Entwicklung mehrfach verwendbarer Software-komponenten	Liste bestehender Instanzen, bestehende Dokumentation zur domänen-spezifischen Softwarearchitektur, bestehende Applikationen, Anforderungen an neue Applikation	Systemdokumentation (Software-komponentenmodell, Software-komponentenentwurf, Software-komponenten-Code-Doku-mentation mit Implementierung) und Benutzungsanleitung der Softwarekomponenten
Integration der Software-komponenten	Dokumentation zu neuen und bestehenden mehrfach verwendbaren Softwarekomponenten, Entwurf und Benutzungsanleitung zur domänen-spezifischen Softwarearchitektur, Liste bestehender Instanzen, bestehende Applikationen	Erweiterte Systemdokumentation (Entwurf der domänen-spezifischen Softwarearchitektur) und Benutzungsanleitung zur domänenspezifischen Softwarearchitektur
Mehrfach-verwendbarkeits analyse	Mehrfachverwendbarkeitsübersicht, Wissen um Ad-hoc-Wiederverwendung, Anforderungen an aktuelle Applikation, bestehende Applikationen, Dokumentation zu bestehenden mehrfach verwendbaren Software-komponenten und zur domänen-spezifischen Softwarearchitektur	Erweiterte Mehrfach-verwendbarkeitsübersicht
Verbesserung der Mehrfach-verwendbarkeit	Evolutionsschrittplanung, Anforderungen an neue Instanz, Dokumentation und Implementierung zu bestehenden mehrfach verwendbaren Softwarekomponenten, Dokumentation zur domänenspezifischen Software-architektur	Erweiterte Dokumentation und Implementierung zu den verbesserten mehrfach verwendbaren Software-komponenten, Dokumentation zur verbesserten domänen-spezifischen Softwarearchitektur
Planung des Evolutions-schrittes	Mehrfachverwendbarkeitsübersicht, Anforderungen an aktuelle Applikation, Wissen um Unternehmen	Evolutionsschrittplanung

Bei der iterativen Entwicklung der Domäne entsteht die Dokumentation ebenfalls iterativ. Einige Ausgangsprodukte entstehen im Rahmen eines Iterationsschrittes, wie beispielsweise beim Softwarekomponentenentwurf und bei der Softwarekomponentenimplementierung, und fließen in die Systemdokumentation der Softwarekomponenten ein. Andere Ausgangsprodukte und die entsprechenden Dokumente werden über die Iterationen hinweg schrittweise vervollständigt. Ein Beispiel dafür ist die domänenspezifische Softwarearchitektur.

6.2.2 Logische Folge der Aktivitäten

Wie in Abschnitt 6.1 erläutert, ergibt sich aus den Abhängigkeiten der Aktivitäten über deren Eingangs- und Ausgangsprodukte eine logische Folge für deren Durchführung.

Den Beginn der Erweiterung um eine oder mehrere neue Softwarekomponenten stellt die „Identifikation potenzieller Softwarekomponenten“ dar. Mit den Ausgangsprodukten kann anschließend die „Entwicklung mehrfach verwendbarer Softwarekomponenten“ und die „Integration der Softwarekomponenten“ durchgeführt werden. Alternativ zur Erweiterung um neue mehrfach verwendbare Softwarekomponenten, sind bestehende mehrfach verwendbare Ergebnisse bezüglich ihrer Mehrfachverwendbarkeit zu verbessern. Als weitere Alternative kann bei der „Durchführung des Evolutionsschrittes“ keine Erweiterung bzw. Verbesserung stattfinden. Das macht insbesondere dann Sinn, wenn keine Ressourcen dafür frei sind. Welche der drei Alternativen im Rahmen der „Durchführung des Evolutionsschrittes“ zu wählen ist, wird in der Aktivität „Planung des Evolutionsschrittes“ entschieden. Diese Aktivität ist deshalb vor der „Durchführung des Evolutionsschrittes“ zu durchlaufen. Die Planung basiert auf dem Ausgangsprodukt der Aktivität „Mehrfachverwendbarkeitsanalyse“. Aus diesem Grund ist sie vor der Aktivität zur „Planung des Evolutionsschrittes“ auszuführen. Da die Überprüfung der Mehrfachverwendbarkeit wiederum auf den Ausgangsprodukten der „Entwicklung neuer Softwarekomponenten“ aufbaut, ergibt sich an dieser Stelle eine Rückkopplung über die Anwendung der mehrfach verwendbaren Ergebnisse, die das geforderte iterative Vorgehen darstellt. Abbildung 6.2 zeigt die resultierende logische Folge der Aktivitäten.

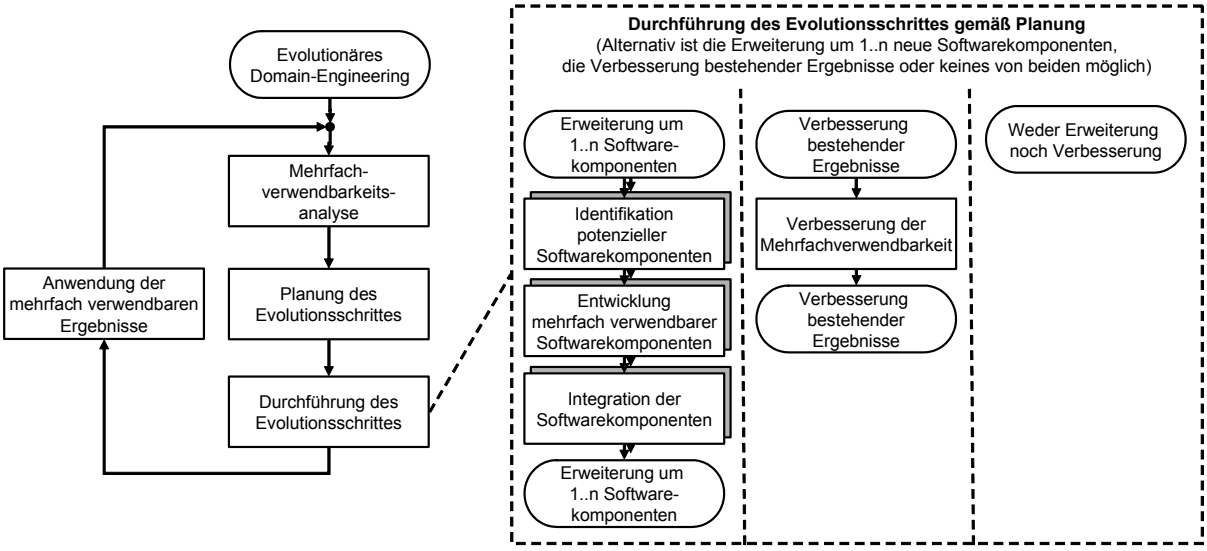


Abbildung 6.2: Logische Folge der Aktivitäten als Flussdiagramm

Beim ersten Durchlaufen der logischen Folge von Aktivitäten bestehen weder Softwarekomponenten noch eine domänenspezifische Softwarearchitektur. Sie werden jedoch für die Überprüfung der Mehrfachverwendbarkeit im Rahmen der „Mehrfachverwendbarkeitsanalyse“ sowie für die „Verbesserung der Mehrfachverwendbarkeit“ benötigt. Bei Aktivitäten, die

Ausgangsprodukte aus Aktivitäten vorhergehender Iterationen als Eingangsprodukte nutzen, ist deshalb zwischen der ersten und den folgenden Iterationen zu unterschieden.

Der Produktfluss der Methodik des evolutionären Domain-Engineering soll einem allgemeinen Vorgehensmodell folgen, welches das iterative Vorgehen unterstützt. Das bisher beim Domain-Engineering verwendete Wasserfall-Modell als Teil des zweigeteilten Entwicklungsprozesses ist nicht geeignet. Bessere Voraussetzungen bietet das Spiral-Modell nach Boehm [Boeh88]. Es repräsentiert den Entwicklungsprozess durch eine Spirale, wobei jede Schleife der Spirale eine abgeschlossene Projektphase zur Erstellung von Teilprodukten darstellt. Für die Erstellung jedes Teilprodukts sind mehrere Schritte zu durchlaufen. Anschließend wird in der nächsten Schleife ein weiteres Teilprodukt entwickelt bzw. ein bestehendes Teilprodukt verfeinert. Das Spiral-Modell erfüllt damit die Anforderungen für die Realisierung einer Methodik zur Anwendung des iterativen Konzeptes. Überträgt man die beschriebenen Aktivitäten auf das Spiral-Modell, ergibt sich daraus eine Spirale, bei der in jeder Schleife ein Evolutionsschritt durchgeführt wird. Als Teilprodukt entsteht jeweils eine neue Generation der Domäne. Das Teilprodukt kann entweder durch die Erweiterung um neue Softwarekomponenten entstehen oder durch die Verbesserung der bestehenden Ergebnisse. Abbildung 6.3 zeigt das resultierende Spiral-Modell.

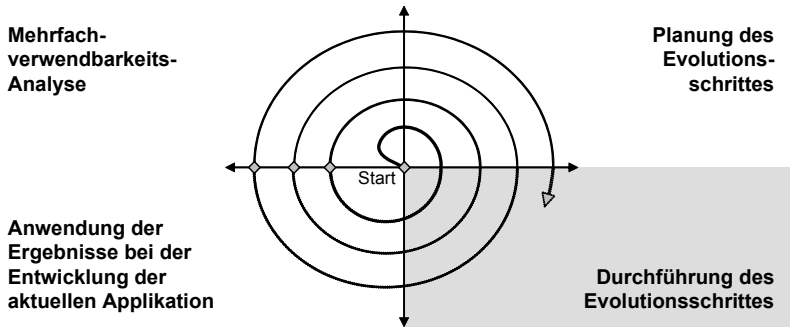


Abbildung 6.3: Methodik des evolutionären Domain-Engineering

Bei der Identifikation potenziell mehrfach verwendbarer Teile wird während der Entwicklung von Applikationen Bedarf an bestehenden Lösungen erkannt (vgl. Abschnitt 5.3.2). Da Produkte zwischen der Applikationsentwicklung und der vorgestellten Methodik ausgetauscht werden, sind die beiden Prozesse zeitlich aufeinander abzustimmen.

6.2.3 Synchronisierung mit der Applikationsentwicklung

Die „Mehrfachverwendbarkeitsanalyse“ ist die erste Aktivität eines Evolutionsschrittes. Sie wird auf Basis des System-Modells aus der Applikationsentwicklung durchgeführt, das gegen Ende der Definitions-Phase erstellt wird. Der Beginn einer Iteration ist von der Fertigstellung der entsprechenden Ausgangsprodukte bei der Applikationsentwicklung abhängig. Umgekehrt kann die Integrationsphase der Applikationsentwicklung erst starten, wenn die Ergebnisse der Durchführung des aktuellen Evolutionsschrittes vorliegen. Abbildung 6.4 zeigt die zeitlichen Zusammenhänge in einem Balkendiagramm. Es ist zu berücksichtigen, dass der Entwicklungs-

aufwand, der zur Durchführung des Evolutionsschrittes benötigt wird, maßgeblich von den bei der „Planung des Evolutionsschrittes“ getroffenen Entscheidungen abhängt.

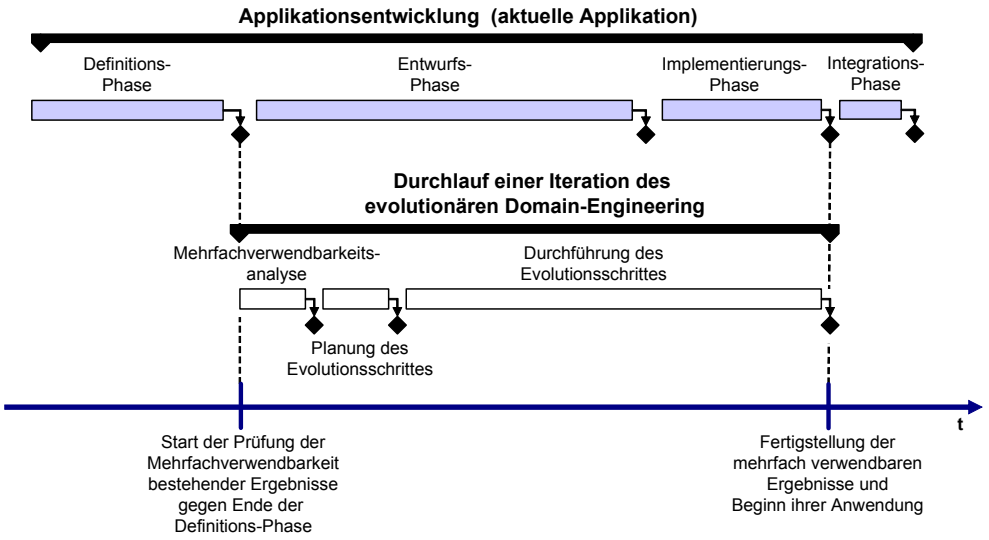


Abbildung 6.4: Synchronisierung zwischen der Applikationsentwicklung und einem Durchlauf des Spiral-Modells

Aus den Aktivitäten und Produkten sowie der logischen Folge der Aktivitäten ergibt sich die zu entwickelnde Methodik des evolutionären Domain-Engineering. Zusätzlich trägt die Synchronisation mit der Applikationsentwicklung der in Abschnitt 2.2.2 beschriebenen Zweiteilung des Entwicklungsprozesses beim Domain-Engineering Rechnung.

6.3 Ergebnis der Einführung einer Methodik zur Anwendung des iterativen Konzeptes

Die vorgestellte Methodik hat die Unterstützung von Softwareentwicklern bei der strukturierten Entwicklung mehrfach verwendbarer Softwareeinheiten zum Ziel. Dazu wird das Konzept zur iterativen Entwicklung mehrfach verwendbarer Softwareeinheiten nach Kapitel 5 gemäß Abschnitt 6.2 umgesetzt. Mit der vorgestellten Methodik wurde eine Möglichkeit geschaffen, den Entwicklungsaufwand zeitlich zu verteilen und die spezifischen Eigenschaften von Automatisierungssystemen zu berücksichtigen.

Die beschriebene Methodik unterstützt die strukturierte Entwicklung mehrfach verwendbarer Softwarekomponenten und einer domänenspezifischen Softwarearchitektur. Sie macht Vorgaben zu den durchzuführenden Aktivitäten, der Reihenfolge ihrer Abarbeitung und der Synchronisierung mit der Applikationsentwicklung. Darüber hinaus werden Produkte beschrieben, welche die Ergebnisse der einzelnen Aktivitäten darstellen. Die Aktivitäten und Produkte sind sehr abstrakt. Sie reichen in der beschriebenen Form nicht aus, um Softwareentwicklern detaillierte Anweisungen zu bieten. Es ist eine Verfeinerung notwendig, die im folgenden Kapitel durchgeführt wird.

7 Verfeinerung von Aktivitäten und Produkten der Methodik

Die in Kapitel 6 eingeführten Aktivitäten der Methodik werden im vorliegenden Kapitel in kleinere, überschaubare Tätigkeiten zerlegt. Bei der Zerlegung wird das Konzept zur iterativen Entwicklung mehrfach verwendbarer Softwareeinheiten für Automatisierungssysteme aus Kapitel 5 berücksichtigt. Die folgenden Abschnitte sind gemäß den Aktivitäten aus Abschnitt 6.2 gegliedert.

7.1 Mehrfachverwendbarkeitsanalyse

7.1.1 Zerlegung in Teilaktivitäten

Die „Mehrfachverwendbarkeitsanalyse“ liefert die Grundlagen für die Entscheidung über den aktuellen Evolutionsschritt. Dabei sind zwei voneinander unabhängige Aspekte zu betrachten. Zum einen sind potenziell mehrfach verwendbare Lösungen in bestehenden Applikationen zu erkennen, auf deren Basis im Rahmen des Evolutionsschrittes neue Softwarekomponenten entwickelt werden können. Zum anderen ist die Mehrfachverwendbarkeit bestehender Ergebnisse zu überprüfen. Deshalb werden für die „Mehrfachverwendbarkeitsanalyse“ die beiden Teilaktivitäten „Erkennung potenziell mehrfach verwendbarer Lösungen“ und „Messung der Mehrfachverwendbarkeit“ durchgeführt. Die beiden Teilaktivitäten können unabhängig voneinander durchgeführt werden.

7.1.2 Erkennung potenziell mehrfach verwendbarer Lösungen

Wie in Abschnitt 5.3.2 erläutert, sind Lösungen bestehender Applikationen zu erkennen, für die Bedarf bei der Entwicklung der aktuellen Applikation besteht. Dazu wird das Wissen erfahrener Entwickler genutzt, die sowohl die Automatisierungsaufgaben und die zur Automatisierung eingesetzten Einrichtungen als auch die in bestehenden Applikationen verwendeten Lösungen kennen. Aus der Ad-hoc-Wiederverwendung ist bekannt, dass für unterschiedliche Teile einer Applikation Lösungen aus verschiedenen bestehenden Applikationen genutzt werden. Für eine systematische Erkennung ähnlicher Problemstellungen und geeigneter Lösungen ist eine Analyse der aktuellen Applikation während der frühen Phase ihrer Entwicklung notwendig. Ziel der Analyse ist es, den Bedarf an bestehenden Lösungen zu erkennen und die entsprechenden Lösungen zu dokumentieren.

Bei der Analyse der aktuellen Applikation prüft ein erfahrener Entwickler, ob in bestehenden Projekten Lösungen zu ähnlichen Problemstellungen existieren. Wird eine Lösung erkannt, die

ggf. durch Anpassungen für die Entwicklung der aktuellen Applikation genutzt werden kann, werden der Bedarf sowie Informationen zum erneuten Auffinden der Lösung dokumentiert. Teile, die durch bestehende Softwarekomponenten abgedeckt werden, sind zu vernachlässigen. Wie in Abschnitt 5.3 erläutert, kann die Suche nach bestehenden Lösungen auch ergebnislos verlaufen und die Menge der erkannten potenziell mehrfach verwendbaren Lösungen bleibt leer.

Zur Beurteilung, ob Problemstellungen bezüglich des zu automatisierenden technischen Prozesses bzw. der einzusetzenden Einrichtungen für die Automatisierung ähnlich sind, sind deren Merkmale zu vergleichen. Mit den Ergebnissen des Vergleichs kann entschieden werden, ob die bestehende Lösung für die Entwicklung der aktuellen Applikation nützlich ist. Um einen strukturierten Vergleich zu ermöglichen, wird ein Fragenkatalog angewandt. In Anhang A findet sich ein Vorschlag für einen Fragenkatalog, der spezifische Fragen für Automatisierungssysteme enthält. Die Gliederung der Fragen sowie die Fragen selbst orientieren sich an den Grundaufgaben der Prozessautomatisierung [Göhn05b] und den technischen Bestandteilen von Prozessautomatisierungssystemen [Göhn05a]. Um domänenspezifische Merkmale beim Vergleich besser berücksichtigen zu können, ist der Fragenkatalog ggf. um domänenspezifische Fragen zu erweitern.

Für die Dokumentation einer erkannten Lösung sind zwei Informationen wichtig. Zum einen muss die Lösung eindeutig identifizierbar sein. Dazu sollte sie einen Namen erhalten, der beispielsweise ihre Funktion bzw. ihre Aufgabe innerhalb der Applikation beschreibt. Zum anderen muss sie später wieder gefunden werden. Deshalb ist die Applikation, in der die Lösung realisiert wurde, und ggf. eine genauere Bezeichnung ihrer Lage festzuhalten. Zur übersichtlichen Darstellung aller erkannten Lösungen bietet sich eine Matrix an, die zeigt, welche Lösungen in welchen bestehenden Applikationen vorhanden sind. Ein Beispiel für eine Applikations-Lösungs-Matrix ist in Abbildung 7.1 dargestellt.

Lösung X ist in Applikation A und Applikation C vorhanden

Lösung Y ist in der modularen Einheit „d“ in Applikation A und in der modularen Einheit „abc“ in Applikation D vorhanden

Applikation / Lösung	Applikation A	Applikation B	Applikation C	Applikation D	...	Häufigkeit
Lösung X	X		X			2
Lösung Y	Modulare Einheit „d“			Modulare Einheit „abc“		2
Lösung Z	X	X	X			3
...						

Abbildung 7.1: Beispiel einer Applikations-Lösungs-Matrix

Die Spalten der Applikations-Lösungs-Matrix repräsentieren die bestehenden Applikationen. In den Zeilen wird jeweils eine erkannte, potenziell mehrfach verwendbare Lösung dargestellt. In

den Feldern der Matrix wird dokumentiert, in welcher bestehenden Applikation die Lösung erkannt wurde. Ggf. werden Angaben über die Platzierung der Lösungen eingetragen.

7.1.3 Messung der Mehrfachverwendbarkeit

Laut Abschnitt 5.6 ist das Ziel der Überprüfung der Mehrfachverwendbarkeit, die tatsächliche Mehrfachverwendbarkeit der domänenspezifischen Softwarearchitektur und der Softwarekomponenten bei deren Einsatz im Rahmen der Entwicklung neuer Applikationen zu ermitteln. Dabei sind die Nützlichkeit (Usefulness) und die Anwendbarkeit (Usability) der Ergebnisse zu untersuchen und in einer Statistik zu dokumentieren. Diese stellt das Ausgangsprodukt der Teilaktivität dar. Für die Entscheidung über das weitere Vorgehen ist von Bedeutung, ob die bestehenden Ergebnisse bei der Entwicklung der aktuellen Applikation eingesetzt werden können. Das ist abhängig vom Bedarf an den Ergebnissen und von den vorhandenen Konfigurationsmöglichkeiten zur Anpassung an die gegebenen Anforderungen. Bei der Verbesserung bestehender Ergebnisse sind die Ursachen der Mängel interessant, wie beispielsweise geänderte Anforderungen oder neue Technologien. Für die Dokumentation der aufgeführten Informationen wird eine Tabelle eingesetzt. Kann eine Instanz der mehrfach verwendbaren Ergebnisse bei der Entwicklung der aktuellen Applikation eingesetzt werden, ist in dieser Tabelle die verwendete Konfiguration zu dokumentieren. Die Tabelle enthält eine Spalte, in der die Applikation festgehalten wird, in der die Ergebnisse zum Einsatz kommen sowie eine Spalte, in der die eingesetzte Konfiguration in Form des genutzten Softwarekomponentensatzes eingetragen wird. Bei fehlgeschlagenen Einsätzen ist in weiteren Spalten die Ursache für das Misslingen und die Konfiguration, welche den gestellten Anforderungen am nächsten kommt, festzuhalten. Festgestellte Mängel werden beschrieben, indem beispielsweise die Anforderungen der aktuellen Applikation aufgeführt werden, die nicht erfüllt werden. Ein Beispiel für eine Statistik über die Mehrfachverwendbarkeit zeigt Abbildung 7.2.

Bezeichnung der Applikation	Erfolgreich eingesetzter SWK-Satz	Mangelhafte SWK bzw. DSSA	Ursache / Beschreibung des Mangels
Applikation 1	A	-	
Applikation 2	D	-	
Applikation 3	-	C	Anforderung /F123/ konnte nicht erfüllt werden
...	...	...	...

Abbildung 7.2: Beispiel einer Statistik über die tatsächliche Mehrfachverwendbarkeit bestehender Ergebnisse

Im Rahmen der Entwicklung der aktuellen Applikation muss untersucht werden, ob die bestehenden mehrfach verwendbaren Ergebnisse eingesetzt werden können. Das ist gemäß Abschnitt 5.6 nach der Definitions-Phase der Applikationsentwicklung zu tun.

7.1.4 Dokumentation

Die Ergebnisse der Teilaktivitäten werden in der Mehrfachverwendbarkeitsübersicht dokumentiert. Sie enthält die Applikations-Lösungs-Matrix und die Statistik über die Mehrfachverwendbarkeit. Das Dokument wird mit jedem Evolutionsschritt erweitert und aktualisiert. Eine Mehrfachverwendbarkeitsübersicht sollte mindestens folgende Inhalte umfassen:

- Historie der Evolution
- Applikationsliste (Applikationen, die ohne / mit Mehrfachverwendung entwickelt wurden) mit kurzer Beschreibung, Ansprechpartner, Auslieferungsdatum, Speicherort, ...
- Applikations-Lösungs-Matrix und ggf. eine genauere Beschreibung der bestehenden Lösungen außerhalb der Matrix
- Statistik über die Mehrfachverwendbarkeit bestehender Ergebnisse und ggf. eine genauere Beschreibung von Ursachen und Mängel außerhalb der Tabelle

7.1.5 Ergebnisse der Verfeinerung

Aus der beschriebenen Vorgehensweise und den entstehenden Ausgangsprodukten wird der in Abbildung 7.3 dargestellte Produktfluss abgeleitet. Die Aktivität „Mehrfachverwendbarkeitsanalyse“ enthält die beiden unabhängigen Teilaktivitäten „Erkennung potenziell mehrfach verwendbarer Lösungen“ und „Messung der Mehrfachverwendbarkeit“. Ihre Ergebnisse werden zusammen in der Mehrfachverwendbarkeitsübersicht dokumentiert. Sie stellt das Ausgangsprodukt der Aktivität „Mehrfachverwendbarkeitsanalyse“ dar und dient als Eingangsprodukt für die Aktivität „Planung des Evolutionsschrittes“.

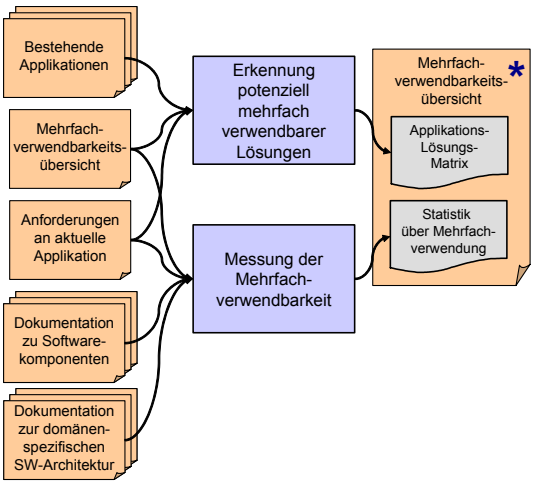


Abbildung 7.3: Aktivitäten und Produkte bei der Mehrfachverwendbarkeitsanalyse

Bei der „Mehrfachverwendbarkeitsanalyse“ der ersten Iteration bestehen noch keine mehrfach verwendbaren Ergebnisse, die überprüft werden können. Deshalb wird in der ersten Iteration nur die Teilaktivität „Erkennung potenziell mehrfach verwendbarer Teile“ durchgeführt.

7.2 Planung des Evolutionsschrittes

In Abschnitt 6.2.2 wurden die drei Alternativen Erweiterung um eine oder mehrere mehrfach verwendbare Softwarekomponenten, Verbesserung der Mehrfachverwendbarkeit bestehender Ergebnisse und Durchführung keiner Aktivität im Rahmen eines Evolutionsschrittes erläutert. In der Aktivität „Planung des Evolutionsschrittes“ wird die Entscheidung getroffen, welche der Alternativen in der aktuellen Iteration durchgeführt wird. Werden keine Mängel bei der Mehrfachverwendbarkeit bestehender Ergebnisse festgestellt, folgt die Erweiterung der mehrfach verwendbaren Softwarekomponenten. Es ist dann zu entscheiden, wie viele Softwarekomponenten im Rahmen des Evolutionsschrittes entwickelt werden und welche der erkannten Lösungen mit Potenzial zur Mehrfachverwendung als Basis dafür dienen. Liegen Mängel bei der Mehrfachverwendbarkeit vor ist festzulegen, ob neue Softwarekomponenten entwickelt oder ob die bestehenden Mängel durch die „Verbesserung der Mehrfachverwendbarkeit“ behoben werden. Es sind die Mängel zu bestimmen, die im Rahmen des Evolutionsschrittes zu beheben sind. Sind die Ressourcen knapp, besteht kein Bedarf an neuen Softwarekomponenten oder liegen keine Verbesserungen an kann auf die Durchführung einer Aktivität verzichtet werden. Eine Zerlegung der Aktivität „Planung des Evolutionsschrittes“ in Teilaktivitäten wird nicht vorgenommen.

Auswahl erkannter Lösungen mit Potenzial zur Mehrfachverwendung

Aus den erkannten Lösungen muss gemäß Abschnitt 5.3.2 eine oder mehrere Lösungen für die weitere Entwicklung ausgewählt werden. Dazu sind geeignete Auswahlkriterien und eine Möglichkeit der Dokumentation der Auswahl zu konzipieren. Die Identifikation potenziell mehrfach verwendbarer Teile basiert gemäß Abschnitt 5.3 auf den Annahmen, dass ein solches Teil in der aktuellen Applikation benötigt wird und dass ein solches Teil in bestehenden Applikationen häufig eingesetzt wurde. Aufgrund des Vorgehens bei der Auswahl wird die erste Annahme für alle erkannten Lösungen als erfüllt vorausgesetzt. Für die Auswahl einer erkannten Lösung dient die zweite Annahme als Grundlage der Auswahlkriterien. Als häufig eingesetzt gilt eine erkannte Lösung dann, wenn Varianten oder Alternativen dieser Lösung in vielen oder allen bestehenden Applikationen vorhanden sind. Die Häufigkeit wird von erfahrenen Entwicklern abgeschätzt und in einer zusätzlichen Spalte der Applikations-Lösungs-Matrix (vgl. Abbildung 7.1) eingetragen. Die Auswahl von Lösungen erfolgt aufgrund ihrer Häufigkeit.

Ist die Identifikation einer Softwarekomponente nicht möglich, ist eine andere Lösung auszuwählen. Sie dient als Ausgangspunkt für einen weiteren Anlauf zur Identifikation einer Softwarekomponente. Beim wiederholten Durchlaufen der Aktivität wird auf die Bewertung der erkannten Lösungen zurückgegriffen. Kann keine geeignete Lösung ausgewählt werden, ist die Entwicklung einer neuen Softwarekomponente erfolglos abzubrechen. Dann wird der Vorgang bei der Entwicklung der nächsten Applikation wiederholt. Verlief die Suche nach potenziell

mehrfach verwendbaren Lösungen ergebnislos, steht die Möglichkeit der Erweiterung um neue Softwarekomponenten nicht zur Auswahl.

Auswahl erkannter Mängel an Mehrfachverwendbarkeit

Wurden bei der Messung der Mehrfachverwendbarkeit mehrere Mängel erkannt, sind die jeweiligen Auswirkungen auf die Anwendbarkeit im Rahmen der Entwicklung der aktuellen Applikation abzuschätzen. Daraus ist eine Gewichtung der erkannten Mängel vorzunehmen. Wurden keine Mängel bezüglich der Mehrfachverwendbarkeit erkannt, steht die Verbesserung der Mehrfachverwendbarkeit bei der zu treffenden Entscheidung nicht zur Auswahl.

Entscheidungsmöglichkeiten

Aus den genannten Möglichkeiten für eine Entscheidung lässt sich der in Abbildung 7.4 dargestellt Entscheidungsbaum ableiten. Eine Entscheidung für die Erweiterung um neue Softwarekomponenten bedingt, dass Lösungen mit Potenzial für die Mehrfachverwendung erkannt wurden. Analog dazu kann eine Verbesserung bestehender Ergebnisse nur erfolgen, wenn mindestens ein Mangel an Mehrfachverwendbarkeit vorliegt. Wurden entweder Lösungen erkannt oder Mängel festgestellt, ist auf Basis der verfügbaren Ressourcen zu entscheiden, wie viele Erweiterungen bzw. Verbesserungen vorgenommen werden. Soll eine Erweiterung um neue Softwarekomponenten erfolgen sind Lösungen auszuwählen, die als Ausgangspunkt für die Entwicklung der Softwarekomponenten dienen. Sollen bestehende Ergebnisse verbessert werden sind Mängel auszuwählen, die zu beheben sind. Wurden sowohl Lösungen erkannt als auch Mängel festgestellt und stehen nicht genügend Ressourcen zur Verfügung alles zu bearbeiten ist die Entscheidung nach wirtschaftlichen Gesichtspunkten zu treffen.

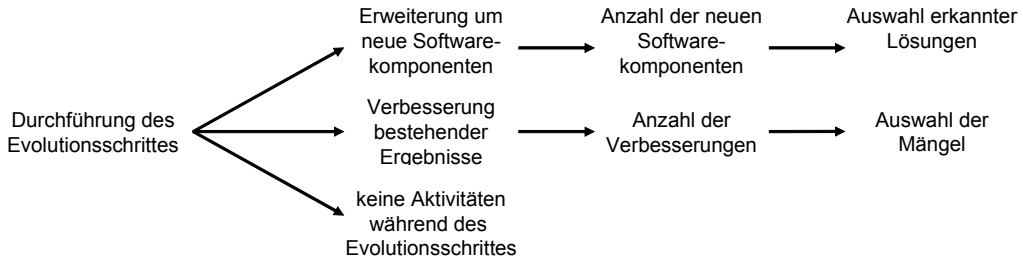


Abbildung 7.4: Entscheidungsbaum für die Planung des Evolutionsschrittes

Wirtschaftliche Grundlagen der Entscheidung

Die Entscheidung zwischen Verbesserung und Erweiterung wird von der Wirtschaftlichkeit der beiden Alternativen beeinflusst. Die Wirtschaftlichkeit hängt ab von der Differenz der Kosten der Verbesserung bzw. Erweiterung und von den durch die Verwendung der Ergebnisse zu erzielenden Einsparungen. Die Höhe der möglichen Einsparungen wird durch die voraussichtliche Zahl der Verwendungen beeinflusst. Die Einsparungen werden durch die Differenz aus dem erwarteten Entwicklungsaufwand ohne Mehrfachverwendung und dem Entwicklungs-

aufwand für die verbesserten bzw. die neu entwickelten Ergebnisse berechnet. Der investierte Entwicklungsaufwand und die Einsparungen, die pro Einsatz der Ergebnisse zu erreichen sind, unterscheiden sich von Fall zu Fall. Abbildung 7.5 zeit drei Beispiele unter Verwendung des in Abschnitt 2.3.1 eingeführten Diagramms zur Darstellung des Entwicklungsaufwandes mit und ohne Mehrfachverwendung. Die Beispiele basieren auf unterschiedlichen Investitionsvolumen für die Erstellung mehrfach verwendbarer Softwareeinheiten und auf unterschiedlichen Werten für Einsparungen bei der Entwicklung von Applikationen mit Mehrfachverwendung. Das Investitionsvolumen kann am Startwert der Geraden für die Entwicklung mit Mehrfachverwendung abgelesen werden. Ihre Steigung stellt den Entwicklungsaufwand pro Applikation dar. Daran kann auch die Einsparung durch die Mehrfachverwendung abgelesen werden. Die Steigung ist kleiner gegenüber der Steigung der Geraden für den Entwicklungsaufwand ohne Mehrfachverwendung. Sobald die Summe der Einsparungen den anfänglichen Mehraufwand aufwiegt, hat sich die Investition amortisiert. Das ist am Schnittpunkt der Geraden mit und ohne Mehrfachverwendung der Fall. Damit wirken sich die Höhe der Investition sowie die Einsparung pro Applikation auf die Amortisierung aus.

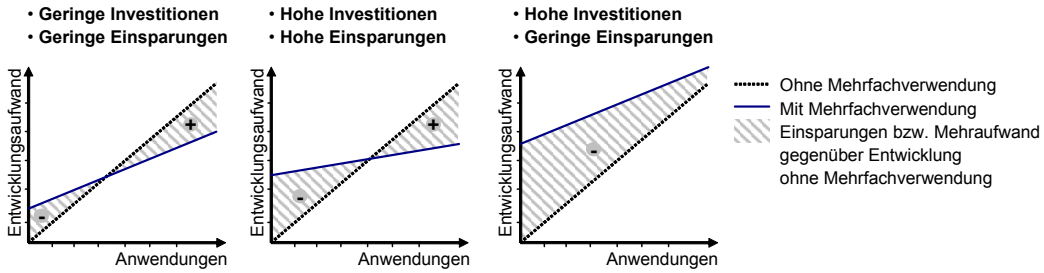


Abbildung 7.5: Entwicklungsaufwand für die Verbesserung und Neuentwicklung mehrfach verwendbarer Ergebnisse

Die Investition und die Entwicklungskosten pro Applikation mit und ohne Mehrfachverwendung können beispielsweise mit COCOMO II [USC04] abgeschätzt werden. Dabei wird der Entwicklungsaufwand mit und ohne Mehrfachverwendung anhand des Source-Code-Umfangs abgeschätzt und daraus die Entwicklungskosten errechnet. Mit diesen Informationen können die Alternativen für die Durchführung des Evolutionsschrittes miteinander verglichen werden. Die Entscheidung wird damit auf Basis wirtschaftlicher Daten begründet.

Die Ergebnisse der „Planung des Evolutionsschrittes“ werden in der Evolutionsschrittsplanung dokumentiert. Bei jedem Evolutionsschritt entsteht ein neues Dokument, das mindestens folgende Inhalte umfassen sollte:

- Ausgewählte Lösungen (mit Begründung)
- Ausgewählte Mängel (mit Begründung)
- Entscheidungsbegründung (z.B. wirtschaftliche Lage, verfügbare Ressourcen, Berechnungen zum Entwicklungsaufwand, ...)
- Entscheidung

Die Aktivität „Planung des Evolutionsschrittes“ wurde gemäß den in Abschnitt 6.2 beschriebenen Tätigkeiten verfeinert. Aus den Ausgangs- und Eingangsprodukten kann der in Abbildung 7.6 dargestellte Produktfluss abgeleitet werden. In Abhängigkeit der getroffenen Entscheidung werden ausgewählte Lösungen, ausgewählte Mängel oder keines der beiden in der Mehrfachverwendbarkeitsübersicht aktualisiert. Die ausgewählten Lösungen werden für die „Erweiterung um 1..n Softwarekomponenten“ benötigt. Die ausgewählten Mängel werden bei der „Verbesserung der Mehrfachverwendbarkeit“ als Eingangsprodukt eingesetzt.

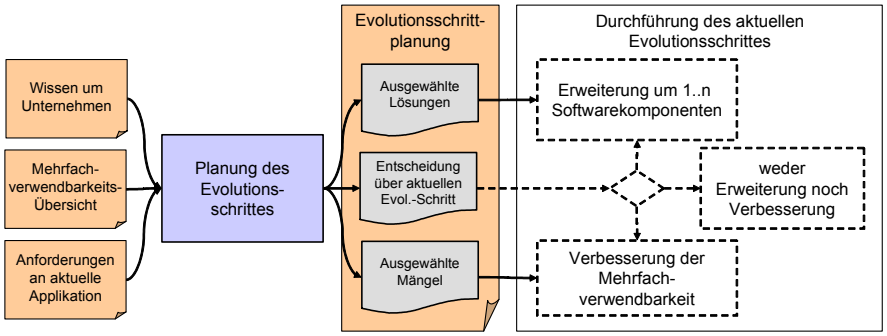


Abbildung 7.6: Teilaktivitäten und Produkte der „Planung des Evolutionsschrittes“

Bei der ersten Iteration bestehen noch keine mehrfach verwendbaren Ergebnisse, die verbessert werden könnten. Die Mehrfachverwendbarkeitsübersicht enthält in diesem Fall noch keine Statistik über die Mehrfachverwendung. Diese steht damit für die „Planung des Evolutionsschrittes“ nicht zur Verfügung und eine Entscheidung kann nur bezüglich der Erweiterung um neue Softwarekomponenten getroffen werden.

7.3 Identifikation potenzieller Softwarekomponenten

Ausgehend von einer als potenziell mehrfach verwendbar erkannten Lösung wird der Funktionsumfang innerhalb der bestehenden Applikationen abgegrenzt, auf dessen Basis eine Softwarekomponente entwickelt wird. Dazu wird das in Abschnitt 5.3.3 vorgestellte Vorgehen zur Bildung hierarchischer Einheiten angewandt.

Um Lösungen in bestehenden Applikationen zu erkennen, die der erkannten potenziell mehrfach verwendbaren Lösung (hierarchische Einheit) ähnlich sind, wird in allen Applikationen der Domäne nach Lösungen gesucht, die ähnliche Automatisierungsaufgaben erfüllen bzw. ähnliche Einrichtungen zur Automatisierung unterstützen. Werden entsprechende Lösungen in weiteren Applikationen gefunden, untermauert dies den Bedarf an einer mehrfach verwendbaren Softwarekomponente mit dem Funktionsumfang der hierarchischen Einheit. Werden keine oder sehr wenige ähnlichen Lösungen in bestehenden Applikationen gefunden, ist durch Generalisierung und Abstraktion eine weitere hierarchische Einheit zu bilden, welche die aktuelle hierarchische Einheit enthält und kapselt. Mit der neuen hierarchischen Einheit wird erneut eine Suche nach ähnlichen Lösungen in den bestehenden Applikationen durchgeführt.

Da die Größe bzw. der Funktionsumfang sowie der Umfang an Konfigurationsmöglichkeiten direkten Einfluss auf die Mehrfachverwendbarkeit einer Softwarekomponente haben (vgl. Abschnitt 2.2.1), ist die Bildung hierarchischer Einheiten rechtzeitig abzubrechen. Spätestens, wenn die hierarchische Einheit die gesamte Applikation umfasst, wird die Suche beendet. Wird eine hierarchische Einheit gefunden, die in vielen oder allen bestehenden Applikationen vorkommt, besitzt sie Potenzial für die Mehrfachverwendung und ist geeignet als Ausgangspunkt für die Entwicklung einer mehrfach verwendbaren Softwarekomponente. Ist die hierarchische Einheit nur in einigen bestehenden Applikationen vorhanden, wird sie gemäß Abschnitt 5.3.1 bei der Entwicklung zukünftiger Applikationen weniger nützlich (useful) sein. Wird keine geeignete hierarchische Einheit gefunden, ist die Suche abzubrechen.

Der Abbruch der Suche hat Auswirkungen auf die Sequenz der durchlaufenen Aktivitäten. Verläuft die Suche erfolgreich, stellt die hierarchische Einheit und die gefundene Menge ähnlicher modularer Einheiten das Ergebnis der Identifikation dar. Verläuft die Suche nicht erfolgreich, kann durch weitere Abstraktion eine neue hierarchische Einheit gebildet werden, mit der die Suche wiederholt wird. Ist die Bildung einer hierarchischen Einheit nicht möglich, entsteht kein Ergebnis und es kann keine Softwarekomponente entwickelt werden. Dann besteht die Möglichkeit, die „Planung des Evolutionsschrittes“ erneut zu durchlaufen und dabei eine andere Lösung auszuwählen. Damit ist die „Identifikation potenzieller Softwarekomponenten“ erneut durchzuführen. Wird bei der „Planung des Evolutionsschrittes“ erkannt, dass keine weiteren Lösungen zur Auswahl stehen, kann im aktuellen Evolutionsschritt keine Erweiterung um neue Softwarekomponenten erfolgen. Das Vorgehen nach dem Abbruch der Aktivität ist in Abbildung 7.7 dargestellt. Eine Zerlegung der Aktivität in Teilaktivitäten ist nicht notwendig.

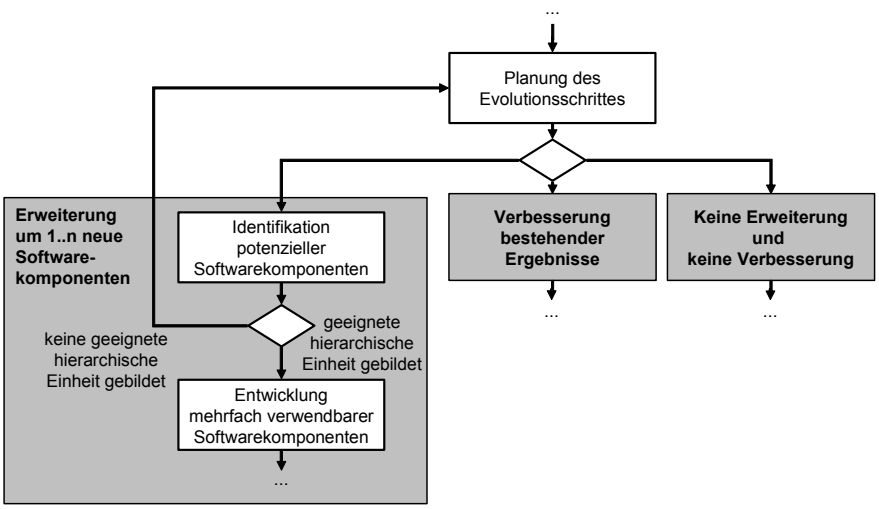


Abbildung 7.7: Vorgehen nach Abbruch der Aktivität „Identifikation potenzieller Softwarekomponenten“

Das Ergebnis der Identifikation potenzieller Softwarekomponenten ist die gefundene hierarchische Einheit sowie eine Menge von ähnlichen modularen Einheiten in anderen Applikationen. Zur Beschreibung der hierarchischen bzw. modularen Einheiten reichen die

Angabe der Applikationen, in denen sie gefunden wurden und ihre Bezeichnung innerhalb der jeweiligen Softwarearchitektur. Eine Darstellung ihrer Eigenschaften oder ihrer Struktur ist in dieser Aktivität nicht notwendig. Dadurch kann auf eine grafische Notation verzichtet werden. Sowohl erfolgreiche als auch erfolglose Suchvorgänge werden in einer Liste dokumentiert. Festgehalten werden der Name der durchsuchten Applikation und das Ergebnis der Suche. Wird eine modulare Einheit gefunden, sind ihr Name und ihre Beziehung zur betrachteten hierarchischen Einheit zu beschreiben. Die Beziehung kann „ähnlich“ oder „alternativ“ sein. Wird keine modulare Einheit gefunden, ist die Beziehung mit „nicht vorhanden“ einzutragen. Zur Dokumentation der Ergebnisse wird die Liste der bestehenden Instanzen eingesetzt. Ein Beispiel für eine Liste der bestehenden Instanzen zeigt Abbildung 7.8.

Name der hierarchischen Einheit: xy_z			
Nr.	**Name der Applikation**	**Name der modularen Einheit**	**Beziehung zur hierarchischen Einheit**
1	Applikation A	xy	ähnliche Lösung
2	Applikation B	xy_z	ähnliche Lösung
3	Applikation C	xy_1	alternative Lösung
4	Applikation D		nicht vorhanden
...	...	...	...

Abbildung 7.8: Beispiel einer Liste bestehender Instanzen

Da die Ergebnisse nur bei der erfolgreichen Suche einer hierarchischen Einheit entstehen, wird auch die Liste bestehender Instanzen nur bei Erfolg erstellt. Ansonsten entsteht kein Ausgangsprodukt der Aktivität.

Aus der Beschreibung der Aktivität „Identifikation potenzieller Softwarekomponenten“ ergibt sich der in Abbildung 7.9 gezeigte Produktfluss.

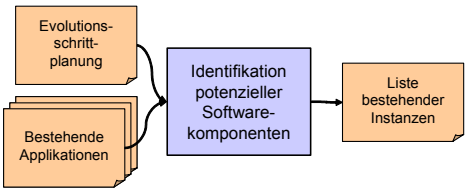


Abbildung 7.9: Produktfluss der Aktivität „Identifikation potenzieller Softwarekomponenten“

Das Dokument „Listen bestehender Instanzen“ stellt das Ausgangsprodukt der Aktivität „Identifikation potenzieller Softwarekomponenten“ dar. Es dient als Eingangsprodukt für die Aktivität „Entwicklung mehrfach verwendbarer Softwarekomponenten“.

7.4 Entwicklung mehrfach verwendbarer Softwarekomponenten

7.4.1 Zerlegung in Teilaktivitäten

Die „Entwicklung mehrfach verwendbarer Softwarekomponenten“ setzt gemäß Abschnitt 6.2 das Konzept zur Entwicklung von Softwarekomponenten aus Abschnitt 5.4 um. Es beinhaltet das Erkennen von Variabilitäten durch die Analyse von Unterschieden und Gemeinsamkeiten, den Entwurf von Konfigurationsmöglichkeiten für Softwarekomponenten und die domänenspezifische Softwarearchitektur sowie die Implementierung der Softwarekomponenten. Das Konzept nutzt die identifizierte hierarchische Einheit und die Menge ähnlicher bzw. alternativer modularer Einheiten als Ausgangspunkt für die Entwicklung einer Softwarekomponente. Können die hierarchische Einheit bzw. eine der modularen Einheiten die Anforderungen der aktuellen Applikation nicht erfüllen, ist vor der Analyse der Unterschiede und Gemeinsamkeiten eine modulare Einheit zu entwickeln, die die Anforderungen erfüllt. Daraus können folgende Teilaktivitäten für die „Entwicklung einer Softwarekomponente“ abgeleitet werden:

- Entwicklung einer neuen Instanz
- Domänenanalyse
- Entwurf von Konfigurationsmöglichkeiten
- Implementierung der Softwarekomponenten

Die einzelnen Tätigkeiten bauen auf den Ergebnissen ihrer Vorgänger auf. Aus den Abhängigkeiten der Eingangs- und Ausgangsprodukte ergibt sich eine logische Folge der Teilaktivitäten, die in Abbildung 7.10 dargestellt ist.

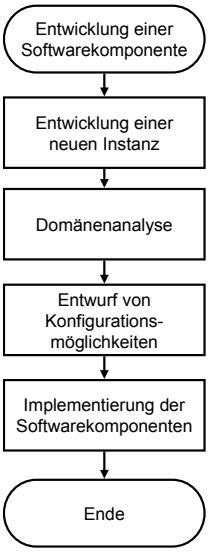


Abbildung 7.10: Logische Folge der Teilaktivitäten zur „Entwicklung einer Softwarekomponente“

In den folgenden Abschnitten werden die vorgestellten Teilaktivitäten beschrieben. Dabei werden Regeln zu deren Durchführung eingeführt sowie Notationen zur Darstellung der resultierenden Ergebnisse bestimmt.

7.4.2 Entwicklung einer neuen Instanz

Erfüllt keine der gefundenen Instanzen alle Anforderungen der Applikation, die aktuell entwickelt wird, ist eine geeignete neue Instanz zu erstellen (vgl. Abschnitt 5.4). Bei der Entwicklung einer neuen Instanz werden neue Funktionen entwickelt. Es entsteht eine neue Variante oder Alternative der bereits dokumentierten modularen Einheiten. Dabei sind die Grenzen der hierarchischen bzw. modularen Einheiten nach Möglichkeit beizubehalten. Wird die neue Funktionalität durch Ad-hoc-Wiederverwendung der hierarchischen Einheit entwickelt, entstehen Unterschiede und Gemeinsamkeiten. Das Ergebnis der Erweiterung ist eine modulare Einheit, welche die Anforderungen der aktuellen Applikation erfüllt und eine weitere Variante oder Alternative zur hierarchischen Einheit darstellt.

7.4.3 Domänenanalyse

In Abschnitt 5.4.1 wird dargestellt, dass zum Erkennen von Variabilitäten durch die Merkmalanalyse bestehende Applikationen bzw. Teile daraus auf Unterschiede und Gemeinsamkeiten zu untersuchen sind. Dieses Vorgehen wird in der Teilaktivität „Domänenanalyse“ umgesetzt. Ausgangspunkt ist die ausgewählte hierarchische Einheit und die Menge der dazu ähnlichen Lösungen, die in der Liste bestehender Instanzen dokumentiert sind. Bei der Analyse sind die Merkmale der Einheiten zusammen mit ihren Beziehungen zur Domäne zu erkennen und zu beschreiben. Die Notation zur Darstellung der Analyseergebnisse wurde in Abschnitt 5.4.1 durch das Merkmaldiagramm eingeführt. Bei dem vorgestellten Konzept werden die Merkmale der hierarchischen Einheiten sowie der ähnlichen bzw. alternativen Lösungen in das Merkmaldiagramm eingetragen. Daraus lässt sich das im Folgenden beschriebene Vorgehen ableiten.

Zu Beginn der Analyse existiert noch kein Merkmaldiagramm. Zur Erstellung werden zunächst die Merkmale der hierarchischen Einheit als Merkmaldiagramm abgebildet. Da bei der Betrachtung genau einer Instanz noch keine Unterschiede erkannt werden können, entstehen auch keine alternativen oder optionalen Merkmale. Anschließend werden die Merkmale der ähnlichen bzw. alternativen Instanzen in das Merkmaldiagramm eingetragen. Ist ein Merkmal einer Instanz bereits im Merkmaldiagramm enthalten, ist zu prüfen, ob die beiden Merkmale identisch sind. Gleiches gilt, wenn bereits alternative oder optionale Merkmale bestehen. Um die Betrachtung bestehender Applikationen bezüglich Automatisierungsaufgabe und der zur Automatisierung eingesetzten Einrichtungen zu unterstützen, wird der in Abschnitt 7.1.2 eingeführte Fragenkatalog (Anhang A) eingesetzt. Er ist ggf. um domänenspezifische Fragen zu erweitern.

Gemäß den Beziehungen der Unterschiede sind alternative oder optionale Merkmale in das bestehende Merkmaldiagramm einzutragen (vgl. Abbildung 5.4). Dabei ist es möglich, dass bestehende Beziehungen geändert werden müssen. Besteht beispielsweise ein optionales Merkmal und es wurde erkannt, dass alternativ dazu ein anderes Merkmal eingesetzt werden kann, ergibt sich eine Oder-Beziehung zwischen diesen beiden Merkmalen. In [CzEi00] ist eine Liste möglicher Umwandlungen von Beziehungen zwischen Merkmalen gegeben.

Zwischen den Unterschieden können gemäß Abschnitt 5.4.2 Abhängigkeiten bestehen. Sie führen bei der Entwicklung von Konfigurationsmöglichkeiten zu erlaubten und nicht erlaubten Kombinationen von Parametern. Es ist für den Entwurf wichtig, die Kombinationen der Merkmale zu kennen, die in den bestehenden Applikationen vorkommen. Diese Zusammenhänge werden in der Merkmal-Applikations-Matrix festgehalten. Abbildung 7.11 zeigt ein Beispiel für eine solche Matrix. Die darin dargestellten Merkmale stammen aus dem in Abschnitt 5.4.1 eingeführten Beispiel (vgl. Abbildung 5.3 und Abbildung 5.4).

Applikation / Merkmal	Applikation A	Applikation B	Applikation C	Applikation D	Applikation E	...
G1H1		X				
G2H2		X		X		
Sicherheitsventil	X		X		X	
d		X				
...						

Abbildung 7.11: Beispiel einer Merkmal-Applikations-Matrix zur Dokumentation von Variabilitäten in bestehenden Applikationen

Die Merkmal-Applikations-Matrix beschreibt, welches Merkmal in welcher bestehenden Applikation vorkommt. Dazu werden die Merkmale in den Zeilen und die Applikationen in den Spalten der Matrix aufgetragen. Ist ein Merkmal in einer Applikation vorhanden, wird dies im entsprechenden Feld der Matrix gekennzeichnet.

7.4.4 Entwurf von Konfigurationsmöglichkeiten

Durch die Abstraktion von Variabilitäten werden, wie in Abschnitt 5.4.2 beschrieben, Konfigurationsmöglichkeiten entworfen. Das Vorgehen wird in der Teilaktivität „Entwurf von Konfigurationsmöglichkeiten“ umgesetzt. Laut Abschnitt 5.4 ist beim Entwurf zum einen zu entscheiden, welche Unterschiede mit welcher Art von Variationspunkten eingeführt werden und zum anderen sind Mechanismen zur Umsetzung dieser Variationspunkte auszuwählen.

Entwurfsentscheidungen

Laut Abschnitt 5.4.2 wird zwischen drei Arten von Variationspunkten unterschieden. Variationspunkte in Form der Austauschmöglichkeit von Softwarekomponenten in der domänenspezifischen Softwarearchitektur ermöglichen die Abstraktion umfangreicher Unterschiede. Variationspunkte, die durch Anpassungen des Source-Codes von Softwarekomponenten realisiert werden, unterstützen die Realisierung von Konfigurationsmöglichkeiten kleineren Umfangs. Variationspunkte, die durch Abstraktion von Unterschieden in der internen Softwarearchitektur von Softwarekomponenten entstehen, ermöglichen den Umgang mit Unterschieden mittleren Umfangs. Beim Entwurf der Konfigurationsmöglichkeiten sind für die im Merkmaldiagramm dokumentierten Unterschiede geeignete Variationspunkte auszuwählen.

Variationspunkte der domänenspezifischen Softwarearchitektur können beispielsweise eingesetzt werden, wenn Unterschiede nahe der Wurzel des Merkmaldiagramms auftreten, da das meist ein Indiz für umfangreiche Unterschiede ist. Ein Beispiel dazu zeigt Abbildung 7.12. Die alternativen Merkmale B, C und D sind Unter-Merkmale der Wurzel. Sie werden durch einen Variationspunkt in Form der Austauschmöglichkeit von Softwarekomponenten in der domänenspezifischen Softwarearchitektur umgesetzt. Dabei entsteht zum einen ein Platzhalter für eine Softwarekomponente in der domänenspezifischen Softwarearchitektur. Zum anderen entstehen drei Softwarekomponenten, die entweder Merkmal B, C oder D sowie deren jeweilige Unter-Merkmale realisieren. Die Softwarekomponenten zur Umsetzung von Merkmal B abstrahieren die alternativen Merkmale E und F. Je nach Umfang der Unterschiede zwischen E und F kann ein Variationspunkt der internen Softwarearchitektur der Softwarekomponente oder ein Variationspunkt des Source-Codes zur Abstraktion eingesetzt werden.

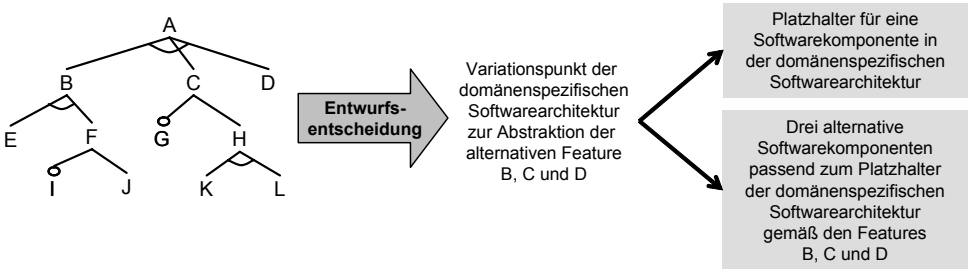


Abbildung 7.12: Beispiel für eine Entwurfsentscheidung beim Entwurf eines Variationspunktes der domänenspezifischen Softwarearchitektur

Die Methode ODM (vgl. Abschnitt 2.4.3) unterscheidet bei der Einführung von Abstraktionen zwischen der reinen Umsetzung der Unterschiede aus dem Domänenmodell und der innovativen Umsetzung, bei der vorausschauend vorgegangen wird. Das heißt, dass bei der Umsetzung der Variationspunkte auch zukünftige Entwicklungen berücksichtigt werden. Aus der Abstraktion dreier alternativer Werte wird dann nicht ein Variationspunkt, der die Auswahl genau eines Wertes erlaubt, sondern ein Variationspunkt, der die Konfiguration einer Konstante innerhalb eines festzulegenden Wertebereiches ermöglicht. Dieses Vorgehen erfordert die Kenntnis der

Domäne. Bei innovativen Erweiterungen besteht das Risiko, dass die entwickelten Funktionen in zukünftigen Applikationen nicht benötigt werden. Der Nutzen des erbrachten Entwicklungsaufwandes ist ungewiss [SCK+96]. Es wird deshalb empfohlen nur innovative Erweiterungen einzubringen, die in den folgenden zwei bis drei Applikationen benötigt werden und ggf. weitere Überarbeitungen der Ergebnisse in Kauf zu nehmen.

Als Folge der Entwurfsentscheidungen entstehen Variationspunkte entsprechend der Unterschiede aus dem Merkmaldiagramm. Diese sind in geeigneter Weise zu dokumentieren. Dazu wird für jede Art der Variationspunkte eine Notation eingeführt.

Dokumentation von Variationspunkten in der domänenspezifischen Softwarearchitektur

Variationspunkte in der domänenspezifischen Softwarearchitektur stellen bei der Instanziierung eine Auswahlmöglichkeit alternativer oder optionaler Softwarekomponenten zur Realisierung definierter Teile der Softwarearchitektur dar (vgl. Abschnitt 2.2). Eine geeignete Notation muss die Modellierung von Softwarekomponenten und ihrer Verknüpfungen ermöglichen. Zur Dokumentation der Variationspunkte sind Möglichkeiten zur Beschreibung von Platzhaltern für alternative oder optionale Softwarekomponenten notwendig. Softwarekomponenten sind gemäß den Variationspunkten, für die sie entwickelt wurden, zu kennzeichnen und die Notation muss die in Abschnitt 3.4 beschriebenen Merkmale von Automatisierungssystemen erfassen können.

Bei der Beschreibung von Softwarearchitekturen wird das Konzept der Architektursichten [Kruc95] verfolgt. Eine Architektursicht ist eine Repräsentation einer Menge von Elementen und deren Beziehungen untereinander [CBBG+02]. Zur Beschreibung einer Softwarearchitektur werden eine bis mehrere Architektursichten dokumentiert. Es wird zwischen statischen und dynamischen Architektursichten unterschieden. Der Fokus liegt entweder auf der Darstellung von Strukturen und Beziehungen modularer Einheiten oder auf der Beschreibung von Abläufen. Der *Components & Connectors View* ist eine statische Architektursicht. Er zeigt eine Menge kooperierender modularer Einheiten mit Laufzeitverhalten [ICGN+04] und deren Beziehungen. Gomaa [Goma01] zeigt eine Möglichkeit auf, UML 2.0 [OMG03a] für die Darstellung eines Components & Connectors Views zu nutzen. Dabei werden Softwarekomponenten durch *Structured Classes* im *Composite Structured Diagram* der UML 2.0 dargestellt. Die Softwarekomponenten kommunizieren über *Ports*, die aus angebotenen und/oder benötigten Schnittstellen bestehen. *Connectors* verbinden diese Ports. Darüber hinaus unterstützt die UML 2.0 Erweiterungsmechanismen, wie beispielsweise *Stereotypes* oder *Tagged Values,* zur Darstellung von Zusatzinformationen in Diagrammen. Diese eignen sich zur Darstellung von Variationspunkten [ZHJ03]. Abbildung 7.13 zeigt Symbole der UML 2.0 zur Darstellung domänenspezifischer Softwarearchitekturen mit Structured Classes und Tagged Values.

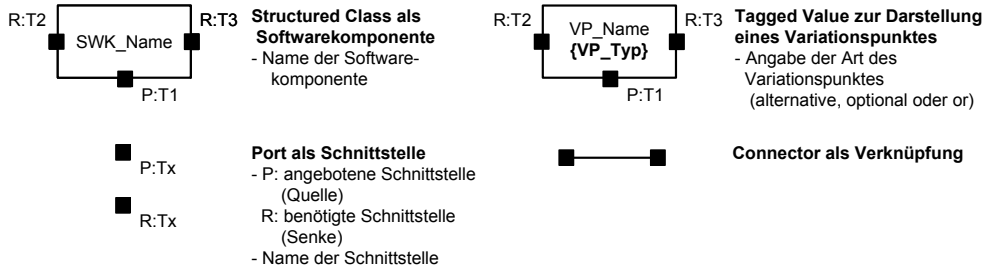


Abbildung 7.13: Notation zur Darstellung der domänenspezifischen Softwarearchitektur basierend auf der UML 2.0

Mit der eingeführten Notation können Softwarekomponenten, deren Beziehungen untereinander und Variationspunkte der domänenspezifischen Softwarearchitektur dargestellt werden. Das ermöglicht die Modellierung der aus dem Merkmaldiagramm abgeleiteten Beziehungen von Softwarekomponenten. Ein Beispiel für eine domänenspezifische Softwarearchitektur mit Variationspunkten wird in Abbildung 7.14 gezeigt. In Variationspunkt A wird das Beispiel aus Abbildung 7.12 mit drei alternativen Softwarekomponenten umgesetzt. Der Variationspunkt T enthält eine einzelne, optionale Softwarekomponente. Damit erfüllt diese Notation alle gestellten Anforderungen zur Darstellung einer domänenspezifischen Softwarearchitektur.

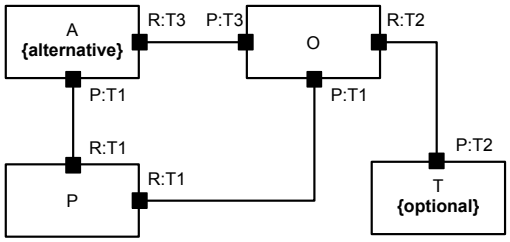


Abbildung 7.14: Beispiel für die Darstellung einer domänenspezifischen Softwarearchitektur

Dokumentation der Variationspunkte von Softwarekomponenten

Für den Entwurf von Variationspunkten innerhalb von Softwarekomponenten sind Mechanismen notwendig, die es erlauben, ihre Softwarearchitektur und ihre Implementierung konfigurierbar zu gestalten. Die notwendigen Mechanismen sind sowohl von den eingesetzten Konzepten als auch von der eingesetzten Programmiersprache abhängig. Bei strukturierter Programmierung müssen beispielsweise andere Mechanismen angewandt werden als bei objekt-orientierter Programmierung. Eine Anpassung der Notation auf die Bedürfnisse der Anwender kann dadurch notwendig werden. In dieser Arbeit wird aufgrund der weiten Verbreitung in der Automatisierungstechnik die Notation UML und die Programmiersprache C++ betrachtet.

Die Notation UML 2.0 [OMG03a] wird beispielsweise für die Darstellung objektorientierter Softwarearchitekturen eingesetzt. Sie bietet jedoch keine grafischen Ausdrucksmittel zur expliziten Darstellung von Variationspunkten einer Softwarearchitektur. In der Literatur sind verschiedene Vorschläge zu finden, wie Variationspunkte dennoch in UML dargestellt werden können. Diese Vorschläge umfassen Entwurfsmuster für bedingte Vererbung nach [CzEi00], die bereits erwähnten Stereotypen zur Kennzeichnung variabler Klassen nach [Goma01] sowie die

in Abschnitt 2.2.3 erläuterten Template-Klassen nach [CzEi00]. Die Anwendung der Entwurfsmuster und der Stereotypen ermöglicht gemäß UML 2.0 die Konfiguration der Variationspunkte zur Laufzeit. Das heißt, dass der Source-Code aller Alternativen und Optionen bei der Kompilierung übersetzt wird. Die Konfiguration mit Template-Klassen lässt dagegen die Konfiguration vor der Kompilierung zu. Begrenzte Ressourcen in Automatisierungscomputern können bei Automatisierungssystemen eine Konfiguration vor der Kompilierung notwendig machen [KLW+96]. Dies wird von den ersten beiden Varianten nicht unterstützt. Zur Dokumentation von Variationspunkten in der Softwarearchitektur von Softwarekomponenten werden deshalb Template-Klassen gewählt.

Abbildung 7.15 zeigt die Anwendung von Template-Klassen an einem Beispiel. Auf der linken Seite der Abbildung ist die Softwarearchitektur der hierarchischen Einheit X dargestellt. Die rechte Seite zeigt das Merkmaldiagramm, das auf Basis der hierarchischen Einheit X sowie ihrer Gemeinsamkeiten und Unterschiede innerhalb der bestehenden Applikationen erstellt wurde. Die modulare Einheit d ist laut Merkmaldiagramm nicht in jeder Ausprägung der bestehenden Applikationen vorhanden und deshalb als optionales Merkmal eingetragen. Dasselbe gilt für die modulare Einheit e. Daraus entsteht die generische Softwarearchitektur der zu entwickelnden Softwarekomponente X unter zu Hilfenahme der Template-Klassen.

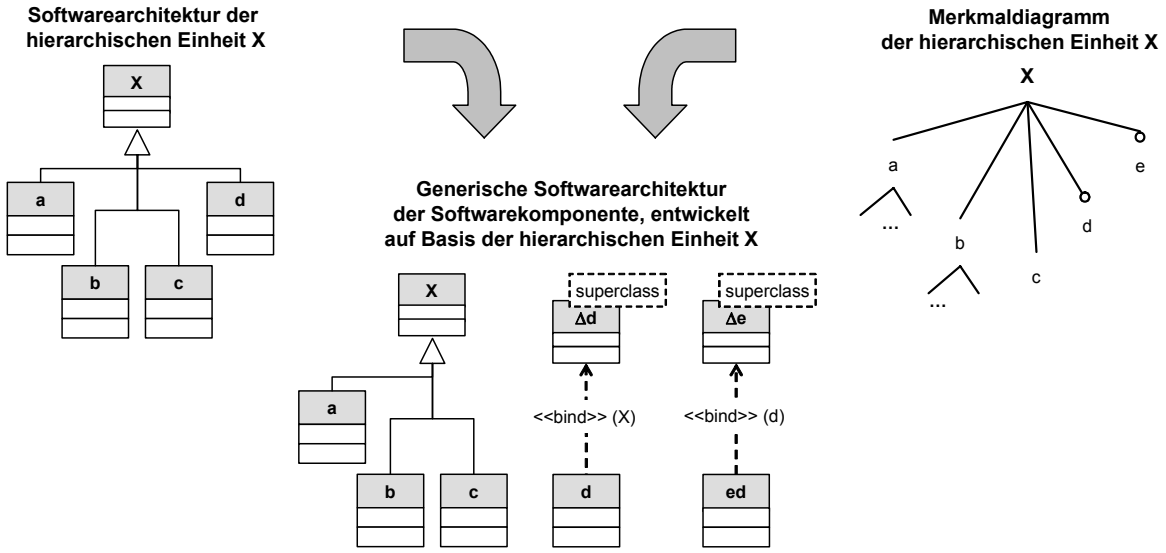


Abbildung 7.15: Beispiel für die Anwendung von Template-Klassen beim Entwurf von Konfigurationsmöglichkeiten für Softwarekomponenten

Über die Sprachkonstrukte zur Umsetzung von Template-Klassen hinaus bietet die Programmiersprache C++ Möglichkeiten zur bedingten Übersetzung durch Präprozessor-Anweisungen [KeRe90]. Mit `#ifdef` können beispielsweise Bedingungen für die Übersetzung eines Abschnittes im Source-Code eingefügt werden. Durch Konstrukte mit `#else` und `#elseif` lassen sich alternative Abschnitte des Source-Codes übersetzen.

Untersuchung der Abhängigkeiten von Konfigurationsmöglichkeiten

Laut Abschnitt 5.4.2 können mit den entworfenen Konfigurationsmöglichkeiten undefinierte Instanzen gebildet werden. Diese entstehen durch die Kombination inkompatibler Merkmale. Zur Beherrschung dieser Problematik wird die Dokumentation gültiger Konfigurationen gefordert. Gültig sind gemäß Abschnitt 5.4.2 die Konfigurationen, die zu den Instanzen führen, aus denen die Softwarekomponenten abgeleitet wurden. Zusätzliche Kombinationsmöglichkeiten bei der Konfiguration sollten nach Bedarf erweitert werden. Dazu sind Bedingungen für die Wahl von Parametern aufzustellen, welche die Freiheitsgrade einschränken und nur bestimmte Kombinationen zulassen. Zur Dokumentation der Bedingungen wird die *Parameter-Kombinations-Matrix* eingeführt, die gültige Kombinationen von Parametern in Parametersätzen zusammenfasst. Abbildung 7.16 zeigt ein Beispiel für eine solche Matrix.

Parameter-Satz / Parameter	Parameter -Satz A	Parameter -Satz B	Parameter -Satz C	Parameter -Satz D	Parameter -Satz E	...
P1 [0,1]	0	1	0	0	0	
P2 [1,2,3]	2	1	3	2	2	
P3[1,2]	1	2	2	1	2	
...		...				

Gültige Kombination von Parametern

Abbildung 7.16: Beispiel einer Parameter-Kombinations-Matrix zur Dokumentation gültiger Instanzen von Softwarekomponenten

Bei der Erstellung der Parameter-Kombinations-Matrix werden auf Basis der Merkmal-Applikations-Matrix aus der Aktivität „Domänenanalyse“ gemäß Abschnitt 7.4.3 gültige Kombinationen von Merkmalen abgelesen. Da die Merkmale den Ausgangspunkt für die Abstraktion der Variabilitäten und damit für den Entwurf der Konfigurationsmöglichkeiten darstellen, ist eine Zuordnung zwischen Parametern bzw. den Werten von Parametern und den ursprünglichen Merkmalen möglich. Gültige Kombinationen von Parametern können in den Spalten abgelesen werden.

Neben der Darstellung der in diesem Abschnitt entworfenen Konfigurationsmöglichkeiten wird auch die Funktionalität der resultierenden Softwarekomponenten in geeigneter Weise dokumentiert.

7.4.5 Implementierung der Softwarekomponenten

In Abschnitt 7.4.4 wurden Konfigurationsmöglichkeiten in Form von Variationspunkten für die domänenspezifische Softwarearchitektur sowie Parameter für die Softwarearchitektur und die Implementierung der Softwarekomponenten entworfen. Es entstanden Abhängigkeiten für die

Instanziierung in Form von Parametersätzen. Bei der Implementierung der Softwarekomponenten sind die Konfigurationsmöglichkeiten in Source-Code umzusetzen.

Die Sprachkonstrukte der Programmiersprache C++ zur Umsetzung von Template-Klassen bieten die Möglichkeit, bei der Instanziierung von Objekten mit Klassen-Optionen deren Struktur, Schnittstellen und Verhalten festzulegen. Die Parameter sind statisch und werden vor der Kompilierung des Source-Codes festgelegt. Nicht ausgewählte Optionen der Template-Klassen werden nicht kompiliert. Folgendes Beispiel zeigt eine Template-Klasse, welche die bedingte Vererbung der Klasse d aus Abbildung 7.15 ermöglicht:

```
// Implementierung der Klasse d
template<class superclass>
class d: public superclass
{
  // d Methoden und Attribute
};
```

Wurde die Klasse e gleichermaßen umgesetzt und besteht eine Klasse X, von der die bestehenden Klassen a, b und c erben, können Instanzen wie in Tabelle 7.1 dargestellt durch Konfiguration angepasst werden. Mit Hilfe der bedingten Vererbung können alle kombinatorischen Möglichkeiten von zwei Optionen bei der Instanziierung realisiert werden. Alternativen werden ähnlich realisiert. Dazu wird eine einzige Template-Klasse erstellt, welche die Möglichkeit bietet, von genau einer Klasse zu erben. Die vererbende Klasse wird aus einer Menge von alternativen Klassen ausgewählt. Unterscheiden sich die Schnittstellen der Alternativen, muss sichergestellt sein, dass andere Teile der Softwarearchitektur, die auf diese Schnittstellen zugreifen, bei der Konfiguration ebenfalls angepasst werden.

Tabelle 7.1: Beispiel für die Konfiguration von Template-Klassen mit parametrisierter Vererbung

Nur Klasse d verfügbar	`d < X >`
Nur Klasse e verfügbar	`e < X >`
Klasse e und Klasse d verfügbar	`e < d < X > >`
Weder Klasse e noch Klasse d verfügbar	`X`

Zur Realisierung von Konfigurationsmöglichkeiten bei der Implementierung werden Präprozessor-Anweisungen [KeRi90, Stro00] eingesetzt. Durch Konstrukte mit `#ifdef` und `#else if` ermöglichen sie die Umsetzung verschiedener Variabilitäten für nahezu alle Bereiche des Source-Code. Ein Beispiel dazu zeigt der folgende Source-Code, bei dem durch

die Code-Zeilen `#define OPTION_NEU` oder `#define ME_B` der eingesetzte Algorithmus zur Berechnung des Rückgabewertes konfiguriert wird:

```
int pot(int base)
{
#ifdef (OPTION_NEU || ME_B)
  return base^2;
#else
  return base * base;
#endif
}
```

Präprozessor-Anweisungen dienen auch zur Prüfung von gültigen Parametersätzen, um falsche Konfigurationen zu unterbinden. Das folgende Beispiel zeigt die Implementierung der Prüfung für zwei abhängige Parameter. Parameter A darf nur konfiguriert werden, wenn Parameter B auch gewählt wurde. Parameter B darf auch alleine gewählt werden. Wird Parameter B durch `#define PARAM_B` konfiguriert und Parameter A nicht, wird beim Kompilieren eine Fehlermeldung ausgegeben.

```
#ifdef PARAM_B
  #ifndef PARAM_A
    #error "PARAM_B kann nicht ohne PARAM_A gewählt werden"
  #endif
#endif
```

7.4.6 Dokumentation

Bei der „Entwicklung mehrfach verwendbarer Softwarekomponenten“ entsteht sowohl eine Systemdokumentation als auch eine Benutzungsdokumentation. Die Systemdokumentation besteht aus vier Dokumenten: der Dokumentation der neuen Instanz, dem Softwarekomponentenmodell, dem Softwarekomponentenentwurf und der Softwarekomponenten-Code-Dokumentation. Als Benutzungsdokumentation ist eine Softwarekomponenten-Benutzungsanleitung zu erstellen. Sowohl die Systemdokumentation als auch die Benutzungsdokumentation entstehen zu jeder Softwarekomponente. Die einzelnen Dokumente umfassen mindestens die im Folgenden beschriebenen Inhalte.

Dokumentation einer neuen Instanz (ggf. mehrere Dokumente):

- Verweis auf die neuen Anforderungen sowie deren Analyse
- Entwurf und Implementierung der neuen Instanz
- Ggf. Übersicht über die Änderungen an bestehenden Lösungen

Softwarekomponentenmodell:

- Liste bestehender Instanzen (aus der Aktivität zur Identifikation potenzieller Softwarekomponenten)
- Merkmaldiagramm und ggf. Erläuterungen zu den einzelnen Merkmalen
- Feature-Applikations-Matrix

Softwarekomponentenentwurf:

- Entwurfsentscheidungen
- Softwarearchitektur der Softwarekomponente (Struktur, Teile der Struktur, Schnittstellen nach außen, Echtzeiteigenschaften, ...)
- Konfigurationsmöglichkeiten (Variationspunkte für die domänenspezifische Softwarearchitektur, Parameter, Parameter-Kombinations-Matrix, ...)

Softwarekomponenten-Code-Dokumentation:

- Schnittstellenbeschreibung
- Source-Code

Softwarekomponenten-Benutzungsanleitung:

- Funktions- und Schnittstellenbeschreibung
- Konfigurationsanleitung
- Integrationsanleitung

7.4.7 Ergebnisse der Verfeinerung

Die Aktivität „Entwicklung neuer Softwarekomponenten“ wird in Teilaktivitäten zerlegt, die in den Abschnitten 7.4.2 bis 7.4.5 detailliert durch Anleitungen und Notationen beschrieben wurden. Es lässt sich der in Abbildung 7.9 dargestellte Produktfluss ableiten.

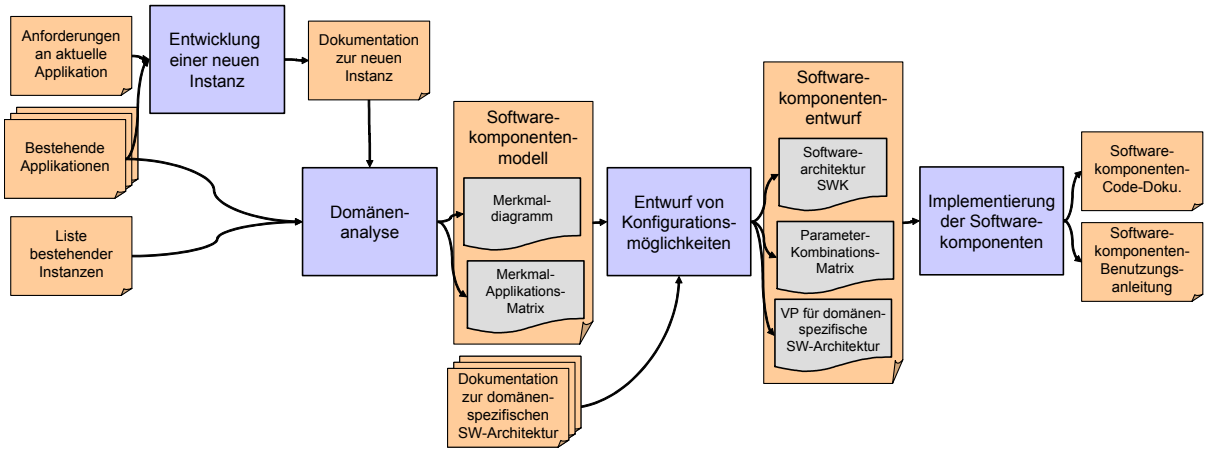


Abbildung 7.17: Teilaktivitäten und Produkte der Aktivität „Entwicklung neuer Softwarekomponenten“

Die neu entwickelten mehrfach verwendbaren Softwarekomponenten stellen das Ausgangsprodukt der Aktivität „Entwicklung mehrfach verwendbarer Softwarekomponenten“ dar. Sie dienen als Eingangsprodukt für die Aktivität „Integration der Softwarekomponenten“.

7.5 Integration der Softwarekomponenten

Die entwickelten Softwarekomponenten müssen entsprechend Abschnitt 5.4.2 in die domänenspezifische Softwarearchitektur integriert werden. Dabei werden die Verknüpfungen der modularen Einheiten genutzt, auf deren Basis die Softwarekomponenten entwickelt wurden. Die bei der Entwicklung der Softwarekomponenten entworfenen Variationspunkte für die domänenspezifische Softwarearchitektur sind bei der Integration zu berücksichtigen. Außerdem sind die Abhängigkeiten zwischen den Softwarekomponenten zu beachten, da sie Schnittstellen durch gegebene Konfigurationsmöglichkeiten beeinflussen können. Eine Zerlegung der Aktivität in Teilaktivitäten ist nicht erforderlich. Die Regeln und Anleitungen zur Durchführung der „Integration der Softwarekomponenten“ werden in diesem Abschnitt eingeführt. Die Notation zur Beschreibung der domänenspezifischen Softwarearchitektur wurde im vorhergehenden Abschnitt erläutert.

Zur Integration wird der Platz in der Struktur der domänenspezifischen Softwarearchitektur gesucht, an dem die neuen Softwarekomponenten eingefügt werden sollen. Dazu werden, wie in Abschnitt 5.4.2 beschrieben, die Lösungen genutzt, aus denen die Softwarekomponenten entwickelt wurden. Anschließend erfolgt die eigentliche Integration unter Berücksichtigung der entworfenen Variationspunkte. Bei vorgeschriebenen und alternativen Softwarekomponenten werden die entsprechenden modularen Einheiten in der domänenspezifischen Softwarearchitektur ersetzt. Bei optionalen Softwarekomponenten müssen Informationen über ihre Position und ihre Verknüpfungen in der domänenspezifischen Softwarearchitektur ggf. aus Lösungen und bestehenden Applikationen abgeleitet werden. Abschließend werden die Verknüpfungen zu bereits vorhandenen Softwarekomponenten und dem noch nicht bearbeiteten Teil der domänenspezifischen Softwarearchitektur erstellt. Die Ergebnisse der Integration werden gemäß Abschnitt 7.4.4 mit UML 2.0 (vgl. Abbildung 7.13) dokumentiert.

Wie in Abschnitt 5.4.2 gezeigt, entstehen bei der Integration der Softwarekomponenten weitere Abhängigkeiten. Konfigurationsmöglichkeiten beeinflussen die Schnittstellen und das Verhalten von Softwarekomponenten. Um bei der Konfiguration der domänenspezifischen Softwarearchitektur die gültige Softwarearchitektur einer Applikation zu erhalten, müssen kompatible Instanzen von Softwarekomponenten kombiniert werden. Bei der Untersuchung der Abhängigkeiten von Konfigurationsmöglichkeiten in der domänenspezifischen Softwarearchitektur sind die Auswahl alternativer und optionaler Softwarekomponenten sowie deren Konfiguration zu berücksichtigen. Die Konfiguration der Softwarekomponenten muss zunächst eine gültige Instanz ergeben. Dazu werden Parametersätze aus der Parameter-Kombinations-Matrix gewählt

(vgl. Abbildung 7.16). Gültige Instanzen der domänenspezifischen Softwarearchitektur bestehen aus einer gültigen Kombination gültiger Instanzen von Softwarekomponenten. Diese werden über Softwarekomponentensätze beschrieben, welche die Parametersätze der ausgewählten Softwarekomponenten enthalten. Zur Dokumentation wird eine Softwarekomponenten-Kombinations-Matrix verwendet, wie sie in Abbildung 7.18 beispielhaft dargestellt ist.

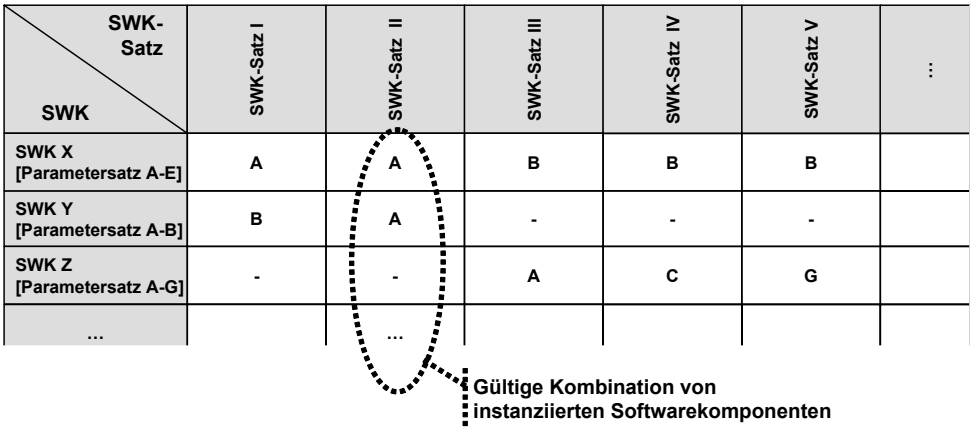


Abbildung 7.18: Beispiel einer Softwarekomponenten-Kombinations-Matrix zur Dokumentation gültiger Instanzen

Bei der „Integration der Softwarekomponenten“ entstehen eine Systemdokumentation und eine Benutzungsdokumentation. Das Dokument „Entwurf der domänenspezifischen Softwarearchitektur“ stellt die Systemdokumentation dar. Zusätzlich ist eine Benutzungsanleitung zur domänenspezifischen Softwarearchitektur zu erstellen. Die Dokumentation wird bei jeder Erweiterung der domänenspezifischen Softwarearchitektur ergänzt und aktualisiert. Die einzelnen Dokumente sollten mindestens die im Folgenden beschriebenen Inhalte umfassen.

Entwurf der domänenspezifischen Softwarearchitektur:

- Entwurfsentscheidungen (z.B. für Integration, Variationspunkte aus der Entwicklung der Softwarekomponenten, ...)
- Domänenspezifische Softwarearchitektur (Struktur, Teile der Struktur, Schnittstellen nach außen, Echtzeiteigenschaften, ...)
- Softwarekomponenten (Liste der bereits verfügbaren Softwarekomponenten, Verweise auf Dokumentation zu den einzelnen Softwarekomponenten, ...)
- Konfigurationsmöglichkeiten (Variationspunkte für die domänenspezifische Softwarearchitektur, Softwarekomponenten-Kombinations-Matrix, ...)

Benutzungsanleitung zur domänenspezifischen Softwarearchitektur:

- Struktur- und Schnittstellenbeschreibung
- Liste verfügbarer Softwarekomponenten mit Verweisen auf deren Dokumentation
- Konfigurationsanleitung
- Integrationsanleitung

Die Aktivität „Integration der Softwarekomponenten“ wird nicht in Teilaktivitäten zerlegt. Der Produktfluss enthält eine Aktivität und die entsprechenden Eingangs- und Ausgangsprodukte. Er ist in Abbildung 7.19 dargestellt.

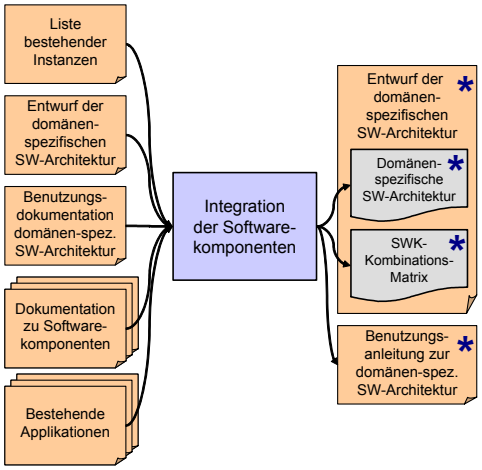


Abbildung 7.19: Eingangs- und Ausgangsprodukte der Aktivität „Integration der Software-komponenten“

Die domänenspezifische Softwarearchitektur fließt zusammen mit den mehrfach verwendbaren Softwarekomponenten in die Applikationsentwicklung ein.

7.6 Verbesserung der Mehrfachverwendbarkeit

Werden bei der „Mehrfachverwendbarkeitsanalyse“ Mängel an den bestehenden Ergebnissen festgestellt, kann alternativ zur Entwicklung neuer Softwarekomponenten die Verbesserung der bestehenden Ergebnisse bezüglich ihrer Mehrfachverwendbarkeit vorgenommen werden. Die dazu notwendigen Tätigkeiten beinhaltet die Aktivität „Verbesserung der Mehrfachverwend-barkeit“. Diese wird im Folgenden in Teilaktivitäten zerlegt, die separat beschrieben werden.

7.6.1 Zerlegung in Teilaktivitäten

Die Maßnahmen zur Verbesserung der Mehrfachverwendbarkeit gemäß Abschnitt 5.7 dienen dazu, die bestehenden Ergebnisse so zu erweitern, dass aus ihnen die geforderte Instanz gebildet werden kann. Die Instanz stellt eine Variante dar, die mit Hilfe der bestehenden Konfigurations-möglichkeiten nicht erstellt werden konnte. Die neuen Variabilitäten, die diese Instanz mit sich bringt, sind in die bestehenden Ergebnisse zu integrieren. Zur Umsetzung des Konzeptes in einer Methodik wird die geforderte Variante entwickelt und in die bestehenden Softwarekomponenten und die domänenspezifische Softwarearchitektur integriert. Die Integration basiert auf der Erkennung von Variabilitäten sowie der Abstraktion der Variabilitäten gemäß den Abschnitten 5.4.1 und 5.4.2. Beides wurde im Rahmen der Aktivitäten „Entwicklung mehrfach verwendbarer Softwarekomponenten“ und „Integration der Softwarekomponenten“ eingesetzt.

Die Aktivität „Verbesserung der Mehrfachverwendbarkeit“ erfordert die folgenden Teilaktivitäten:

- Entwicklung einer neuen Variante
- Erweiterung bestehender Softwarekomponenten
- Erweiterung der bestehenden domänenspezifischen Softwarearchitektur

Durch die Abhängigkeiten der Teilaktivitäten von Ausgangsprodukten ergibt sich eine logische Folge für ihre Durchführung. Diese ist in Abbildung 7.20 als Flussdiagramm dargestellt.

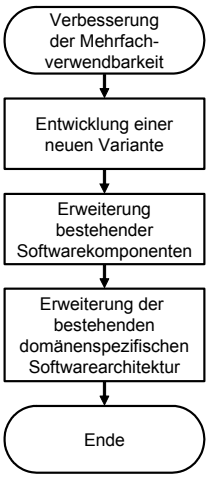


Abbildung 7.20: Logische Folge der Teilaktivitäten zur „Verbesserung der Mehrfachverwendbarkeit“

In den folgenden Abschnitten werden die Teilaktivitäten der „Verbesserung der Mehrfachverwendung“ separat beschrieben.

7.6.2 Entwicklung einer neuen Variante

Laut Abschnitt 5.7.2 ist eine neue Variante unter Anwendung der Ad-hoc-Wiederverwendung auf Basis einer Instanz aus den bestehenden Softwarekomponenten und der aktuellen domänenspezifischen Softwarearchitektur zu erstellen. Zunächst wird eine geeignete Instanz aus den bestehenden mehrfach verwendbaren Ergebnissen gebildet. Sie sollte möglichst viele der gestellten Anforderungen erfüllen. Zur Erfüllung der restlichen Anforderungen ist die Instanz entsprechend anzupassen. Dabei kann beispielsweise eine Skalierung des Funktionsumfangs oder eine Erweiterung der Funktionalität vorgenommen werden. Das Ergebnis ist eine neue Variante, welche die gestellten Anforderungen erfüllt. Um den Überblick über die Anpassungen zu behalten, sind diese in einer Liste zu erfassen. Die Liste der Anpassungen enthält für jede Anpassung einen Vermerk über die geänderte Stelle und die durchgeführten Anpassungen, sodass eine Rekonstruktion der Änderungen möglich ist.

7.6.3 Erweiterung bestehender Softwarekomponenten

In Abschnitt 7.4 wurden die Teilaktivitäten zur Entwicklung neuer Softwarekomponenten beschrieben. Durch die iterative Anwendung entstehen eine oder mehrere Softwarekomponenten inklusive ihrer Dokumentation. Das Ziel der „Erweiterung bestehender Softwarekomponenten“ ist die Integration der in Abschnitt 7.6.2 entwickelten neuen Variante in die bestehenden Softwarekomponenten. Dabei sind alle vorgenommenen Anpassungen zu berücksichtigen. Es können mehrere Softwarekomponenten betroffen sein. Deshalb ist die Integration für alle Softwarekomponenten durchzuführen, an deren Instanzen Veränderungen vorgenommen wurden. Die entsprechenden Softwarekomponenten werden mit Hilfe der Liste der Anpassungen ermittelt. Anschließend werden die einzelnen Softwarekomponenten um Konfigurationsmöglichkeiten gemäß der neuen Instanz erweitert. Dazu werden die in Abschnitt 7.4 beschriebenen Teilaktivitäten durchlaufen und die bestehenden Ausgangsprodukte erweitert.

7.6.4 Erweiterung der bestehenden domänenspezifischen Softwarearchitektur

Die iterative Entwicklung der domänenspezifischen Softwarearchitektur erfolgt durch die Integration neuer Softwarekomponenten. Dies geschieht in der Aktivität „Integration der Softwarekomponenten“ gemäß Abschnitt 7.5. Die domänenspezifische Softwarearchitektur beschreibt die Struktur der Softwarekomponenten. Diese ist nicht zwangsläufig von den Verbesserungen betroffen. Vor der „Erweiterung der bestehenden domänenspezifischen Softwarearchitektur“ ist zu prüfen, ob die „Erweiterung bestehender Softwarekomponenten“ nach Abschnitt 7.6.3 Auswirkungen auf die Struktur hat. Wenn ja, sind die Veränderungen in der domänenspezifischen Softwarearchitektur zu dokumentieren.

7.6.5 Dokumentation

Bei der „Verbesserung der Mehrfachverwendbarkeit“ entsteht keine eigenständige Dokumentation. Es wird vielmehr bei Bedarf die Systemdokumentation und die Benutzungsdokumentation zu den verbesserten Softwarekomponenten und der domänenspezifischen Softwarearchitektur erweitert und aktualisiert. Um eine Nachverfolgbarkeit der Verbesserungen zu gewährleisten, ist insbesondere die Historie der Änderungen und Anpassungen zu pflegen.

7.6.6 Ergebnisse der Verfeinerung

Die Aktivität „Verbesserung der Mehrfachverwendbarkeit“ wird in drei Teilaktivitäten zerlegt, die auf den in den Abschnitten 7.4 und 7.5 vorgestellten Aktivitäten „Entwicklung mehrfach verwendbarer Softwarekomponenten“ und „Integration der Softwarekomponenten“ basieren.

Aus den Teilaktivitäten und ihren Ausgangs- und Eingangsprodukten wird der in Abbildung 7.21 dargestellte Produktfluss der Aktivität abgeleitet.

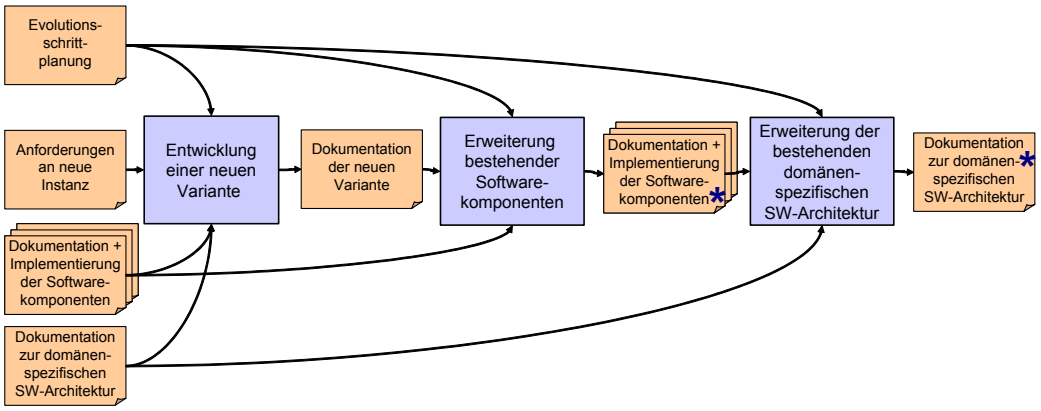


Abbildung 7.21: Teilaktivitäten und Produkte zur „Verbesserung der Mehrfachverwendbarkeit“

Die verbesserten mehrfachverwendbaren Softwarekomponenten sowie die verbesserte domänenspezifische Softwarearchitektur stellen die Ausgangsprodukte der Aktivität „Verbesserung der Mehrfachverwendbarkeit“ dar. Sie fließen zusammen mit den unveränderten Ergebnissen in die Applikationsentwicklung ein.

7.7 Zusammenfassung der Verfeinerung von Aktivitäten und Produkten der Methodik

Die in Kapitel 6 eingeführte Methodik wurde im vorliegenden Kapitel verfeinert. Die in Abschnitt 6.2 eingeführten Aktivitäten wurden in Teilaktivitäten zerlegt. Anschließend erfolgte die Festlegung ihrer logischen Abfolge. Zu jeder Teilaktivität wurden Eingangs- und Ausgangsprodukte eingeführt sowie der resultierende Produktfluss dokumentiert. Zur Durchführung der Teilaktivitäten wurden Regeln oder Anleitungen beschrieben, die Anwender bei der Erstellung der Ausgangsprodukte unterstützen. Bei der Umsetzung flossen das in Kapitel 5 vorgestellte Konzept zur iterativen Entwicklung mehrfach verwendbarer Softwareeinheiten für Automatisierungssysteme sowie Eigenschaften von Domain-Engineering-Methoden gemäß Kapitel 2 ein. Abbildung 7.22 gibt einen Überblick über den Produktfluss aller Teilaktivitäten sowie den zugehörigen Eingangs- und Ausgangsprodukten.

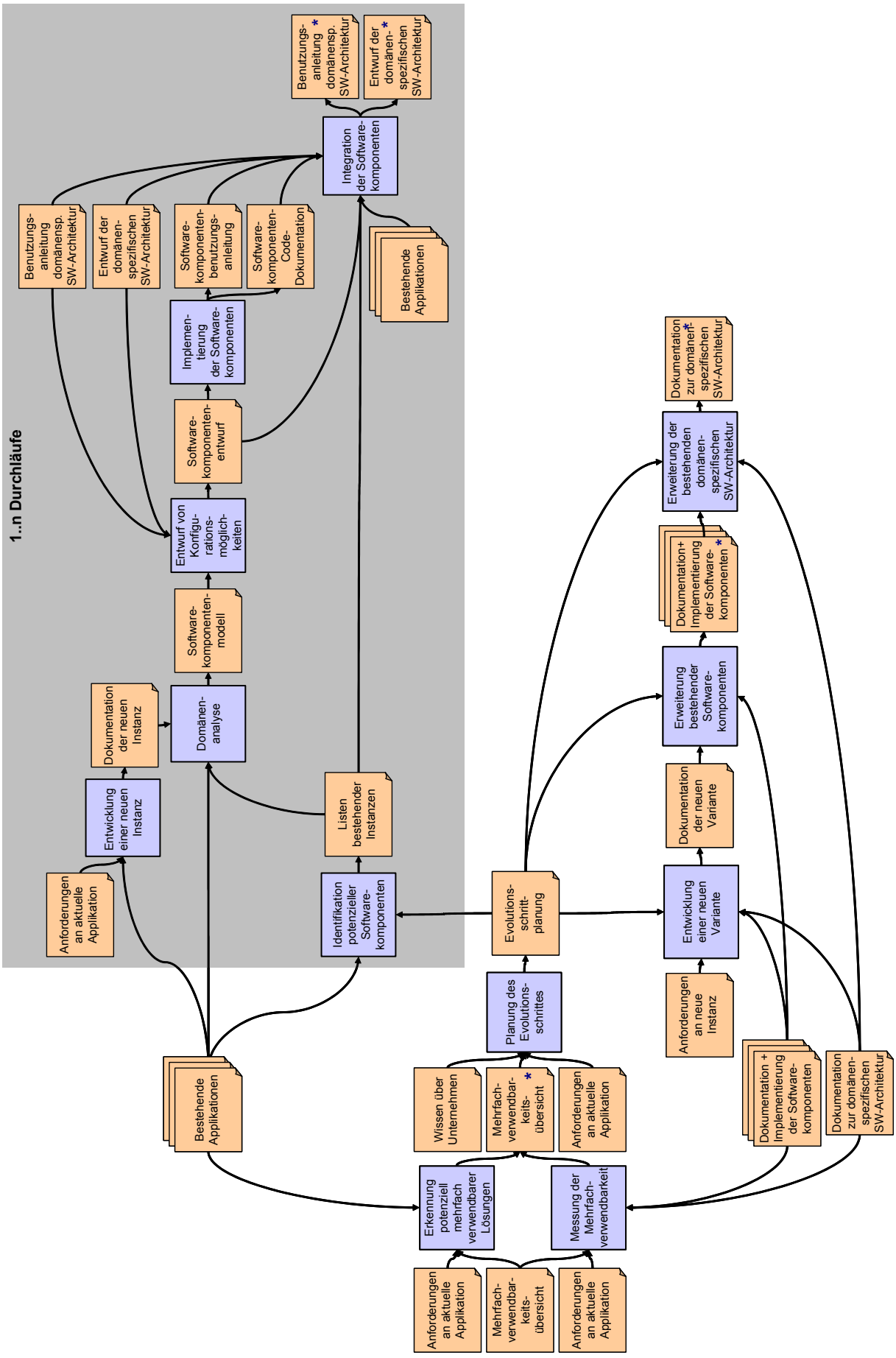


Abbildung 7.22: Überblick über den Produktfluss der verfeinerten Methodik

8 Anwendungsbeispiel Sortieranlagen

In den Kapiteln 4 bis 7 wurde ein iteratives Konzept und darauf aufbauend eine Methodik konzipiert, die das Evolutionsprinzip unterstützt und die Entwicklung mehrfach verwendbarer Softwareeinheiten für Automatisierungssysteme ermöglicht. In diesem Kapitel wird anhand eines Anwendungsbeispiels geprüft, ob die Methodik die in Kapitel 3 aufgestellten Anforderungen erfüllt. In den folgenden Abschnitten wird zunächst eine geeignete Domäne für ein solches Anwendungsbeispiel ausgewählt. Anschließend wird die Anwendung der Methodik über drei Iterationen unter verschiedenen Voraussetzungen dokumentiert.

8.1 Auswahl einer geeigneten Domäne

Als Anwendungsbeispiel wird eine Domäne im Bereich der Automatisierungssysteme benötigt. Die Projekte innerhalb der Domäne müssen einerseits genügend Gemeinsamkeiten aufweisen, um den Nutzen der Mehrfachverwendung erkennen zu können. Andererseits müssen sie unterschiedlich genug sein, um die Herausforderungen bei der Entwicklung für die Mehrfachverwendung darzustellen. Ein Anwendungsbeispiel, das diese Anforderungen erfüllt, ist die Domäne der Sortieranlagen. Die Aufgabe von Sortieranlagen ist die Automatisierung des Sortierens von Werkstücken, Paketen oder ähnlichem Stückgut. Das Stückgut wird gemäß vorgegebener Sortiervorschriften in unterschiedliche Lagerplätze, auf separate Stapel, auf verschiedene Transportbänder oder Ähnlichem abgelegt. Dabei kommen diverse Sortiermechanismen zum Einsatz, wie beispielsweise Weichen oder Hebevorrichtungen.

8.2 Einführung in die Domäne der Sortieranlagen

Es werden drei Hardware-Varianten von Sortieranlagen betrachtet: das Hochregallager, das Sortierband und das 3-Achs-Portal. Modelle dieser Hardware-Varianten wurden am IAS für die Durchführung von Experimenten aufgebaut. Im Folgenden werden die drei Prozessautomatisierungssysteme sowie ihr gemeinsames Automatisierungskonzept beschrieben.

Hochregallager

In einem Hochregallager wird das Stückgut in den Fächern eines Regals abgelegt. Der Sortiervorgang wird von einem Transportarm übernommen, der vor den Fächern platziert wird, um Stückgut einzulagern oder zu entnehmen. Beim Sortiervorgang wird das Stückgut einzeln einem Magazin entnommen und mit Hilfe eines Transportbandes zum Transportarm bewegt. Der Transportarm übernimmt das Stückgut, hebt es vor ein vordefiniertes Fach und legt es dort ab. Das Sortierziel wird auf Basis eines binären Codes berechnet, der am Stückgut angebracht ist und vor dem Sortiervorgang gelesen wird. Einsortiertes Stückgut kann mit dem Transportarm

entnommen und auf ein weiterführendes Transportband abgelegt werden. Für dieses Anwendungsbeispiel stand das Modell eines Hochregallagers mit 5x10 Lagerplätzen zur Verfügung, dessen Aufbau die Abbildung 8.1 zeigt.

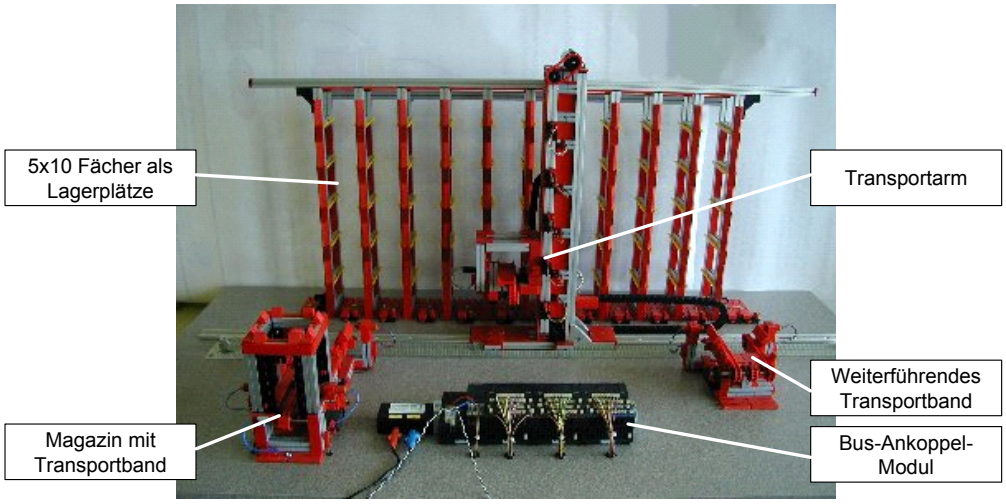


Abbildung 8.1: Sortieranlage Hochregallager

Sortierband

Ein Sortierband transportiert Stückgut über ein Transportband, von dem es über Weichen, Schieber oder Ähnlichem auf weitere Transportbänder oder Lagerplätze geleitet wird. Das Stückgut wird einzeln aus einem Magazin entnommen und auf das Transportband geschoben. Das Sortierband übernimmt das Stückgut vom Transportband und führt es an den Schiebern vorbei. Ist es vor dem richtigen Schieber angekommen, bewegt er das Stückgut vom Transportband auf den entsprechenden Lagerplatz. Das für dieses Beispiel eingesetzte Modell verfügt über ein Transportband, drei Schieber und vier Lagerplätze. Der letzte Lagerplatz befindet sich am Ende des Transportbandes und wird erreicht, wenn keiner der Schieber das Stückgut auf einen anderen Lagerplatz bewegt. Abbildung 8.2 zeigt das Modell des Sortierbandes.

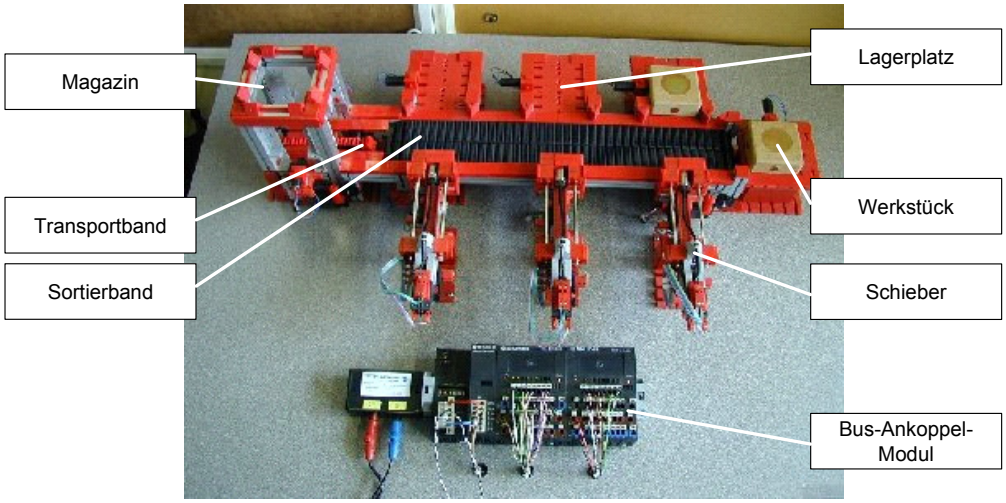


Abbildung 8.2: Sortieranlage Sortierband

3-Achs-Portal

Ein 3-Achs-Portal besteht aus einer Hebevorrichtung für Stückgut, die durch drei getrennt voneinander bewegliche Achsen positioniert wird. Die drei Achsen spannen ein Koordinatensystem auf. Die Hebevorrichtung lässt sich in den Grenzen der Reichweite der jeweiligen Achse frei positionieren. Dadurch kann die Lage der Lagerplätze frei gewählt werden. Das Transportband des 3-Achs-Portals entnimmt das Stückgut einzeln aus einem Magazin. Die Hebevorrichtung nimmt über einen Elektromagneten das Stückgut vom Transportband. Das Stückgut wird über die Z-Achse angehoben und durch die beiden anderen Achsen in X- und Y-Richtung zum gewünschten Lagerplatz gefahren. Nach dem Absenken der Z-Achse wird das Stückgut abgelegt. Das eingesetzte Modell eines 3-Achs-Portals verfügt über drei Lagerplätze, auf denen das Stückgut gestapelt wird. Es ist in Abbildung 8.3 dargestellt.

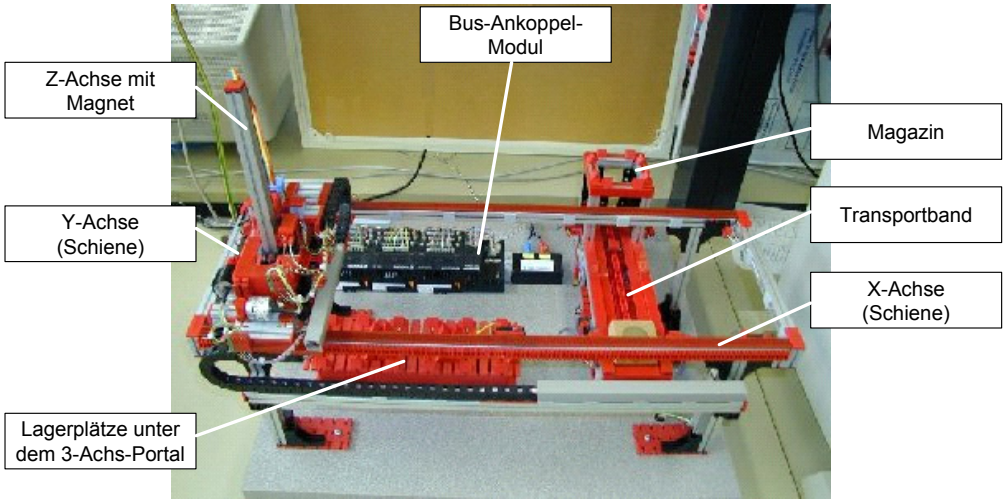


Abbildung 8.3: Sortieranlage 3-Achs-Portal

Automatisierungskonzept

Die Applikationen zur Steuerung der drei Sortieranlagen laufen auf einem Automatisierungscomputer. Über einen Feldbus ist pro Sortieranlage je ein Bus-Ankoppel-Modul mit Ein- und Ausgängen zum Anschluss der Sensoren und Aktoren verbunden. Für das Hochregallager und das Sortierband werden ausschließlich digitale Ein- und Ausgänge eingesetzt. Für das 3-Achs-Portal wird für zwei Achsen zusätzlich ein Modul zur Positionierung von Motoren genutzt. Das Automatisierungskonzept ist in Abbildung 8.4 dargestellt.

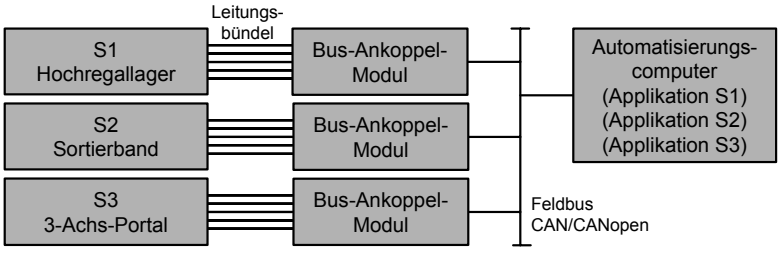


Abbildung 8.4: Automatisierungskonzept der Sortieranlagen

Als Feldbus zur Anbindung der Bus-Ankoppel-Module an den Automatisierungscomputer wird CAN [Etsc94] eingesetzt. Das 3-Achs-Portal nutzt für die Kommunikation mit dem Modul zur Positionierung des Portals in X- und Y-Richtung das auf CAN basierende Protokoll CANopen [CIA04]. Die eingesetzten Protokolle kommunizieren mit unterschiedlichen Übertragungsraten.

8.3 Ausgangspunkt und bestehende Applikationen

Für das Anwendungsbeispiel wird von zwei bestehenden Applikationen ausgegangen. Die erste Applikation ist die Applikation „S1 Hochregallager A“ zur Steuerung des Hochregallagers. Die zweite Applikation ist die Applikation „S2 Sortierband A“ für das Sortierband. Die Applikation „S2 Sortierband A“ entstand mit Hilfe der Ad-hoc-Wiederverwendung auf Basis der Applikation „S1 Hochregallager A“. Die Softwarearchitektur ist in die Schichten *CHRLApplication*, *CSensorState*, *CSecurityMove*, *CBasicMove* und *CCANDriver* sowie die Benutzungsschnittstelle *CHRLDialogBox* und eine Klasse, die den Sortieralgorithmus kapselt, zerlegt. Abbildung 8.5 zeigt die vereinfachte Darstellung der beiden Softwarearchitekturen.

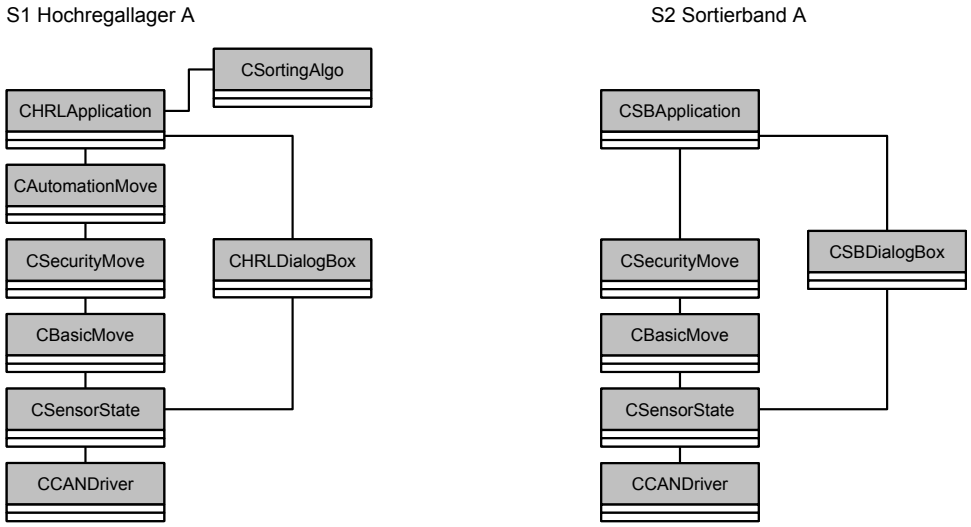


Abbildung 8.5: Softwarearchitekturen der bestehenden Applikationen

Die höheren Schichten der Applikationen beinhalten die Programmlogik, während die unteren Schichten für die Kommunikation mit den Anlagen zuständig sind.

8.4 Evolutionäres Domain-Engineering 1. Generation

In einem neuen Projekt ist eine Applikation zur Automatisierung des Sortierbandes mit dem Namen „S3 Sortierband B“ zu entwickeln. Dabei werden die gleichen Einrichtungen zur Automatisierung eingesetzt, wie bei der Applikation „S2 Sortierband A“. Die Applikationslogik unterscheidet sich aufgrund unterschiedlicher Automatisierungsaufgaben, wie zum Beispiel einem geänderten Sortieralgorithmus. In den unteren Schichten sind durch die Gemeinsamkeiten bei der Hardware auch Gemeinsamkeiten bei der Software zu erwarten. Während der

Durchführung des Projektes soll die erste Softwarekomponente sowie die 1. Generation der domänenspezifischen Softwarearchitektur entwickelt werden. Die Entwicklung wird gemäß der in den Kapiteln 6 und 7 eingeführten Methodik durchgeführt.

Mehrfachverwendbarkeitsanalyse

Die Durchführung der Teilaktivität „Messung der Mehrfachverwendbarkeit“ erübrigt sich, da keine mehrfach verwendbaren Ergebnisse aus vorherigen Iterationen bestehen. Bei der Teilaktivität „Erkennung potenziell mehrfach verwendbarer Lösungen“ gilt es, in den bestehenden Applikationen Teile zu erkennen, die für die Entwicklung der aktuellen Applikation nützlich sind. Das sind insbesondere die unteren Schichten der Applikation „S2 Sortierband A“, wie beispielsweise *CCANDriver*, *CBasicMove* und *CSensorState*. Sie sind in allen bestehenden Applikationen in ähnlicher Form vorhanden und erfüllen viele der Anforderungen der aktuellen Applikation. In der Applikations-Lösungs-Matrix wird festgehalten, welche der genannten Lösungen in welchen bestehenden Applikationen vorkommen. Abbildung 8.6 zeigt einen Ausschnitt aus der Applikations-Lösungs-Matrix für die 1. Generation.

Planung des Evolutionsschrittes

Die Verbesserung bestehender Ergebnisse steht nicht zur Auswahl. Deshalb wird auf Basis der erkannten Lösungen in der Applikations-Lösungs-Matrix eine ausgewählt, aus der eine Softwarekomponente entwickelt werden soll. Als besonders lohnend erscheint die Schicht *CCANDriver*, da diese bei der Anpassung an frühere Applikationen nur wenig geändert werden musste und einen großen Anteil an Gemeinsamkeiten erhoffen lässt. Für die Anpassung der Schichten *CSensorState* und *CBasicMove* mussten umfangreichere Anpassungen vorgenommen werden. Softwarekomponenten, die diesen Funktionsumfang abdecken, erscheinen ebenfalls lohnend, doch aufgrund des größeren Anteils an erwarteten Unterschieden sind sie niedriger zu bewerten als die Schicht *CCANDriver*. Höhere Schichten werden noch niedriger bewertet, da ihre angepassten Anteile noch größer sind. Abbildung 8.6 zeigt einen Ausschnitt aus der Applikations-Lösungs-Matrix, aus der die Lösung *CCANDriver* aufgrund ihrer Bewertung ausgewählt wurde.

Applikation / Lösung	S1 Hochregal-lager A	S2 Sortierband A	S3 Sortierband B	Bewertung
CCANDriver	X	X	X	1
CSensorState	X	X	X	2
CBasicMove	X	X	X	2
...	...	...	...	...

Abbildung 8.6: Ausschnitt aus der Applikations-Lösungs-Matrix mit der ausgewählten Lösung der 1. Generation

Identifikation potenzieller Softwarekomponenten

Bei der gewählten Lösung handelt es sich um eine modulare Einheit einer Applikation, die in allen weiteren Applikationen enthalten ist. Damit kann die modulare Einheit als hierarchische Einheit festgelegt werden. Zur Dokumentation der hierarchischen Einheit werden die ähnlichen modularen Einheiten in den bestehenden Applikationen beschrieben. Dazu dient gemäß Abschnitt 7.1 eine Liste bestehender Instanzen, wie sie in Abbildung 8.7 dargestellt ist.

Name der hierarchischen Einheit: CANDriver			
Nr.	**Name der Applikation**	**Name der modularen Einheit**	**Beziehung zur hierarchischen Einheit**
1	S1 Hochregallager A	CCANDriver	ähnliche Lösung
2	S2 Sortierband A	CCANDriver	ähnliche Lösung
3	S3 Sortierband B	CCANDriver	ähnliche Lösung

Abbildung 8.7: Liste bestehender Instanzen der 1. Generation

Entwicklung mehrfach verwendbarer Softwarekomponenten

Die erste Teilaktivität der „Entwicklung mehrfach verwendbarer Softwarekomponenten“ ist die „Entwicklung einer neuen Instanz“. Dabei ist *CCANDriver* mit Hilfe der Ad-hoc-Wiederverwendung für die neue Applikation zu entwickeln. Bei der anschließenden „Domänenanalyse“ werden anhand des Fragenkatalogs aus Anhang A Unterschiede und Gemeinsamkeiten der modularen Einheiten aus den bestehenden Applikationen ermittelt. Bei der Automatisierungsaufgabe gibt es keine Unterschiede. Durch die Fragen zu den Einrichtungen zur Automatisierung werden Unterschiede bei der CAN-Kommunikation bezüglich Busknoten, Datenobjekten (Länge, Inhalt, ...) und Datenübertragungsraten (vgl. Anhang A Fragen 2.3.1.1.2 ff) aufgedeckt. Beim Vergleich bezüglich der Softwarearchitektur ergeben sich Unterschiede in den Schnittstellen (übergebene Daten). Abbildung 8.8 zeigt einen Ausschnitt aus dem resultierenden Merkmaldiagramm mit den erkannten Unterschieden und Gemeinsamkeiten.

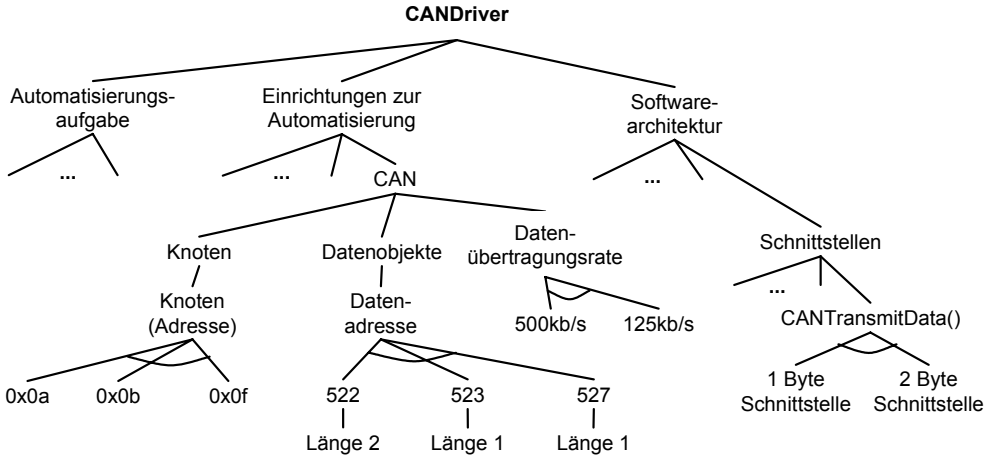


Abbildung 8.8: Merkmaldiagramm der 1. Generation

Die erkannten variablen Merkmale werden den bestehenden Applikationen zugeordnet. Das Ergebnis wird in der Merkmal-Applikations-Matrix dokumentiert, wie sie in Abbildung 8.9 zu sehen ist. Diese Matrix stellt zusammen mit dem Merkmaldiagramm das Ausgangsprodukt der Teilaktivität „Domänenanalyse“ dar.

Applikation / Merkmal	S1 Hochregal-lager A	S2 Sortierband A	S3 Sortierband B
Knotenadresse	0x0a	0x0b	0x0f
Datenadresse	522	523	527
Datenlänge	2 Byte	1 Byte	1 Byte
CANTransmitData()	2 Byte	1 Byte	1 Byte
Datenübertrag-ungsrate	500kb/s	125kb/s	125kb/s

Abbildung 8.9: Merkmal-Applikations-Matrix zur Darstellung der variablen Merkmale

Die gefundenen Unterschiede sind bis auf die Schnittstelle (CANTransmitData()) Konstanten. Bei einem Vorgehen ohne eine innovative, vorausschauende Umsetzung der Unterschiede ergibt sich ein Variationspunkt, der die Auswahl zwischen einer 1-Byte-Schnittstelle und einer 2-Byte-Schnittstelle ermöglicht. Die an die Schnittstelle übergebenen Daten werden über CAN versendet. Wird berücksichtigt, dass CAN das Versenden von 1 bis 8 Byte in einer Botschaft ermöglicht, kann die Schnittstelle für diesen Wertebereich konfigurierbar gemacht werden. Wird dazu ein Array mit 1 bis 8 Byte Länge eingesetzt, resultiert die folgende Schnittstelle, die über die Konstante `CANDRIVER_NR_OF_BYTES` konfigurierbar ist:

```
TStatus CANTransmitData(
                  unsigned char msgData[CANDRIVER_NR_OF_BYTES]
          )
```

Die höheren Schichten, welche diese Schnittstelle verwenden, sind entsprechend anzupassen. Die Konfiguration der Schnittstelle erfolgt über die Konstante `CANDRIVER_NR_OF_BYTES` mit einem Wertebereich von 1 bis 8. Bei der Implementierung der Softwarekomponente wirkt sich der Parameter nicht nur auf die Schnittstelle aus. Bei der Methode `CANTransmitData` wird, wie der nachfolgende Code-Ausschnitt zeigt, Source-Code durch Präprozessor-Anweisungen zur bedingten Übersetzung für die neue Schnittstelle vorbereitet:

```
#if CANDRIVER_NR_OF_BYTES > 0
  event.tagData.msg.data[0] = msgByte[0];
#endif
#if CANDRIVER_NR_OF_BYTES > 1
  event.tagData.msg.data[1] = msgByte[1];
#endif
…
```

Die verbleibenden Unterschiede werden ebenfalls mit Präprozessor-Anweisungen umgesetzt. Bei der Konfiguration der Softwarekomponente sind die Parameter wegen Abhängigkeiten und

Wertebereichen nicht frei wählbar. Abbildung 8.10 zeigt gültige Parametersätze in der Parameter-Kombinations-Matrix.

Parameter-Satz / Parameter	Parameter -Satz A	Parameter -Satz B	Parameter -Satz C
NROFBYTES [1..8]	2	1	1
MSGID [0..4095]	522	523	527
STATIONID [0x0a..0xff]	0x0a	0x0b	0x0f
BITRATE [0..500000]	500000	125000	125000

Abbildung 8.10: Parameter-Kombinations-Matrix der 1. Generation

Integration der Softwarekomponenten

Die modulare Einheit *CCANDriver* ist in allen bestehenden Applikationen vorhanden. Die Integration einer darauf basierenden Softwarekomponente orientiert sich an den Schnittstellen von *CCANDriver*. Da die modularen Einheiten Schnittstellen zum unbearbeiteten Rest der bestehenden Softwarearchitektur und zu einer externen Funktions-Bibliothek besitzen, muss auch die Softwarekomponente *CANDriver* diese Schnittstellen aufweisen. Es entsteht die in Abbildung 8.11 dargestellte domänenspezifische Softwarearchitektur der 1. Generation.

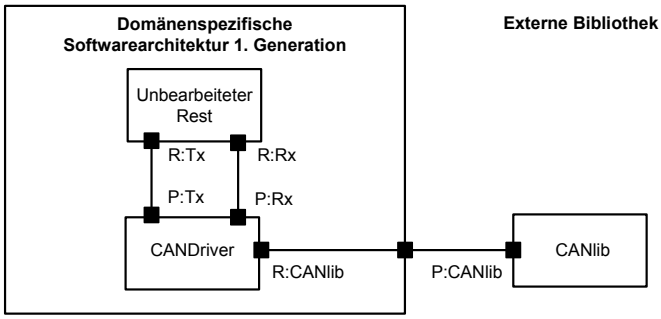


Abbildung 8.11: Domänenspezifische Softwarearchitektur der 1. Generation

Da nur eine Softwarekomponente zur Verfügung steht, gibt es keine Abhängigkeiten für die Kombination von Softwarekomponenten. Die Softwarekomponenten-Kombinations-Matrix enthält drei Sätze von Softwarekomponenten, die den drei Parametersätzen der ersten Softwarekomponente entsprechen. Damit steht die 1. Generation der domänenspezifischen Softwarearchitektur und eine Softwarekomponente für die Applikationsentwicklung zur Verfügung.

8.5 Evolutionäres Domain-Engineering 2. Generation

Im zweiten Projekt ist eine Applikation zur Automatisierung eines Hochregallagers mit dem Namen „S4 Hochregallager B“ zu entwickeln. Dabei sind einige Einstellungen der CAN-

Kommunikation sowie die Abläufe bei der Einlagerung des Stückgutes anzupassen. Es wird nur eine Geschwindigkeit zum Verfahren der Transporteinrichtung benötigt. Die Hardware bleibt im Wesentlichen gleich. Während der Durchführung des Projektes wird eine neue Softwarekomponente sowie die 2. Generation der domänenspezifischen Softwarearchitektur entwickelt.

Mehrfachverwendbarkeitsanalyse

Die 1. Generation der domänenspezifischen Softwarearchitektur und die Softwarekomponente *CANDriver* können durch Konfiguration für die aktuelle Applikation angepasst werden. Die bestehenden Ergebnisse sind bezüglich der aktuellen Applikation sowohl nützlich als auch anwendbar. Das wird in der Statistik über die tatsächliche Mehrfachverwendbarkeit dokumentiert. Abbildung 8.12 zeigt diese Statistik für die 2. Generation.

Bezeichnung der Applikation	Erfolgreich eingesetzter SWK-Satz	Mangelhafte SWK bzw. DSSA	Ursache / Beschreibung des Mangels
S3 Sortierband B	A	-	
S4 Hochregal-lager B	A	-	

Abbildung 8.12: Statistik über die tatsächliche Mehrfachverwendbarkeit der bestehenden Ergebnisse in der 2. Generation

Der Bedarf an bestehenden Lösungen für die Entwicklung der neuen Applikation hat sich gegenüber der letzten Applikation nicht verändert. Die Applikations-Lösungs-Matrix aus der 1. Generation (vgl. Abbildung 8.6) ist deshalb nur um die neuen Lösungen aus der letzten Applikation zu ergänzen.

Planung des Evolutionsschrittes

Da die bestehenden mehrfach verwendbaren Ergebnisse aus den vorherigen Iterationen keine Mängel bei der Mehrfachverwendbarkeit aufweisen, wird bei der „Planung des Evolutionsschrittes“ entschieden, eine neue Softwarekomponente zu entwickeln. Es liegt nahe, eine der beiden Schichten oberhalb von *CANDriver* zu wählen. Die beiden Schichten *CBasicMove* und *CSensorState* sind bezüglich ihres Potenzials gleich zu bewerten. Eine begründete Auswahl kann nicht getroffen werden. Im vorliegenden Anwendungsbeispiel wird *CBasicMove* bestimmt.

Identifikation potenzieller Softwarekomponenten

Es handelt sich bei der gewählten Lösung um eine modulare Einheit, die in allen weiteren Applikationen in ähnlicher Form enthalten ist. Die erkannte Lösung wird als hierarchische Einheit festgelegt. Da alle bestehenden Applikationen eine modulare Einheit mit dem selben Namen enthalten, ist eine weitere Suche nach alternativen Lösungen nicht notwendig. Die hierarchische Einheit sowie die ähnlichen und alternativen modularen Einheiten werden in der Liste bestehender Instanzen dokumentiert.

Bei der Teilaktivität „Entwicklung einer neuen Instanz“ entsteht eine Instanz von *CBasicMove*, welche die Anforderungen der aktuellen Applikation erfüllt. Bei der Anpassung wird eine neue Schnittstelle der Softwarekomponente *CANDriver* eingeführt. Die anschließende „Domänenanalyse“ anhand des Fragenkatalogs aus Anhang A ergibt Unterschiede in der Automatisierungsaufgabe von *CBasicMove*, wie beispielsweise beim Aus- und Einfahren der Schieber (Sortierband), beim Bewegen des Sortierbandes (Sortierband) oder beim Positionieren des Transportarms (Hochregallager). Durch die Fragen zu den Einrichtungen zur Automatisierung werden beispielsweise Unterschiede bei den Sensoren (Endschalter) und Aktoren (Motoren) festgestellt. Die Softwarearchitektur weist Unterschiede in den Schnittstellen auf. Abbildung 8.13 zeigt einen Ausschnitt aus dem resultierenden Merkmaldiagramm.

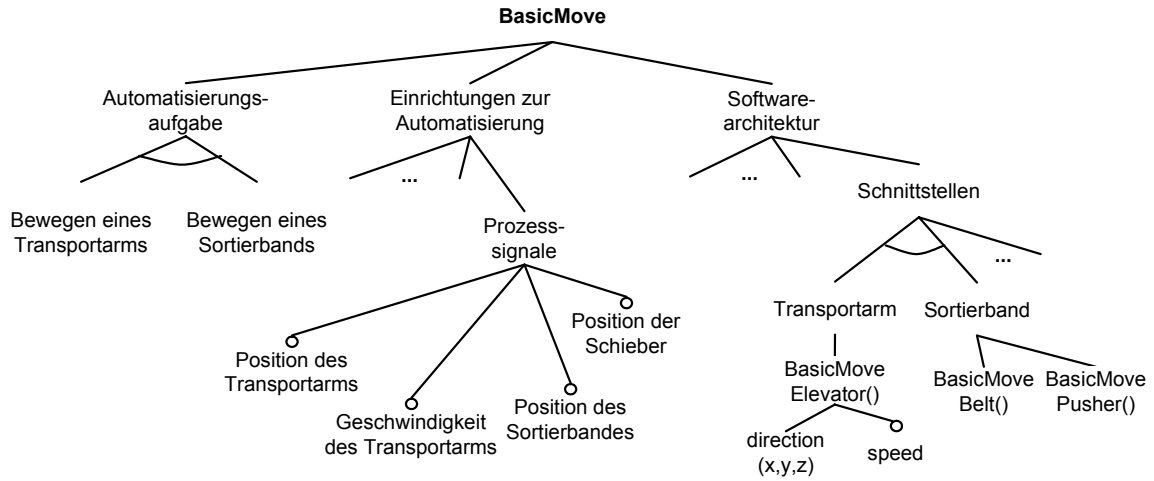


Abbildung 8.13: Merkmaldiagramm der 2. Generation

Durch die Zuordnung der erkannten variablen Merkmale zu bestehenden Applikationen entsteht die Merkmal-Applikations-Matrix, wie sie in Abbildung 8.14 zu sehen ist. Sie lässt erkennen, dass einige Merkmale, wie beispielsweise die „Bewegung eines Transportarms“ sowie die „Bewegung eines Sortierbandes“, nicht zusammen in einer Applikation vorkommen.

Applikation / Merkmal	S1 Hochregal-lager A	S2 Sortierband A	S3 Sortierband B	S4 Hochregal-lager B
Bewegung eines Transportarms	X	-	-	X
Bewegung eines Sortierbandes	-	X	X	-
Motor X (r,l,off)	X	-	-	X
Motor X (r,l,v,off)	X	-	-	-
Motor Y und Motor Z	X	-	-	X
...	...	...	...	...

Abbildung 8.14: Merkmal-Applikations-Matrix der 2. Generation

Beim Entwurf der Softwarekomponente werden zur Abstraktion der alternativen Sortiermechanismen Template-Klassen eingesetzt. Dazu sind die gemeinsamen sowie die alternativen Methoden auf drei neue Klassen zu verteilen, die über eine geeignete Vererbungsstruktur bei der Konfiguration kombiniert werden. Die Gemeinsamkeiten sind in der Klasse *SM_Common* umgesetzt, die Unterschiede in den Klassen *SM_HRL* und *SM_SB*. Abbildung 8.15 zeigt die resultierende Softwarearchitektur für die Softwarekomponente *BasicMove*.

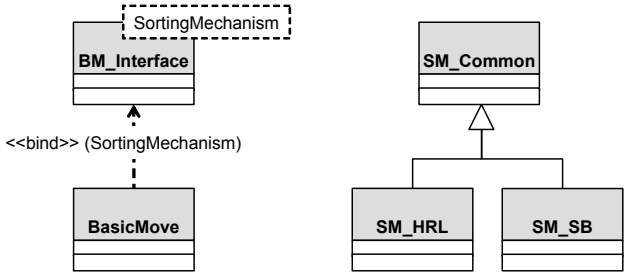


Abbildung 8.15: Softwarearchitektur der Softwarekomponente *BasicMove*

Die Abhängigkeiten zwischen den Parametern der Softwarekomponente werden in einer Parameter-Kombinations-Matrix festgehalten. Die neue Softwarekomponente *BasicMove* wird in die domänenspezifische Softwarearchitektur integriert. Sie besitzt eine Schnittstelle zur bereits bestehenden Softwarekomponente *CANDriver* sowie eine zum noch nicht bearbeiteten Rest der domänenspezifischen Softwarearchitektur. Es ergibt sich die in Abbildung 8.16 dargestellte 2. Generation der domänenspezifischen Softwarearchitektur.

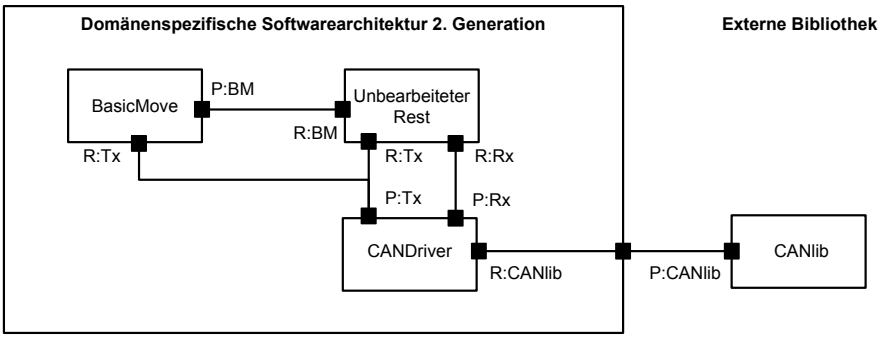


Abbildung 8.16: Domänenspezifische Softwarearchitektur der 2. Generation

Bei zwei Softwarekomponenten ergeben sich in diesem Evolutionsschritt auch Abhängigkeiten bei der Kombination ihrer Instanzen. Diese werden bei der Betrachtung der Schnittstelle deutlich. Wird *BasicMove* beispielsweise für den Einsatz mit einer Schnittstelle von 2 Byte konfiguriert, ist *CANDriver* ebenfalls für 2 Byte anzupassen. Die möglichen Kombinationen werden in der Softwarekomponenten-Kombinations-Matrix dokumentiert. Damit ist die 2. Generation der mehrfach verwendbaren Ergebnisse fertig gestellt und steht zum Einsatz in der aktuellen Applikation bereit.

8.6 Evolutionäres Domain-Engineering 3. Generation

In den ersten beiden Generationen wurden neue Softwarekomponenten entwickelt. Das war möglich, da die Mehrfachverwendbarkeit der bestehenden Ergebnisse ausreichend war. In der 3. Generation wird das 3-Achs-Portal eingeführt. Dadurch ergeben sich neue Anforderungen für die Automatisierung, welche die Mehrfachverwendbarkeit der bestehenden Ergebnisse beeinflussen. Für die neue Applikation mit dem Namen „S5 3-Achs-Portal A“ ist die Mehrfachverwendbarkeit der bestehenden Ergebnisse zu verbessern. Die Verbesserung wird entsprechend der in den Kapiteln 6 und 7 eingeführten Methodik durchgeführt.

Mehrfachverwendbarkeitsanalyse

Bei der Entwicklung der aktuellen Applikation wird festgestellt, dass die bestehenden Softwarekomponenten nicht einsetzbar sind. Das liegt zum einen an der mangelnden Unterstützung des Kommunikationsprotokolls CANopen durch die Softwarekomponente *CANDriver* und zum anderen an der fehlenden Umsetzung des Sortiermechanismus für das 3-Achs-Portal in der Softwarekomponente *BasicMove*.

Der Bedarf an bestehenden Lösungen für die Entwicklung der neuen Applikation hat sich gegenüber den letzten Applikationen nicht verändert. Die Applikations-Lösungs-Matrix aus der 2. Generation ist deshalb nur um die neuen Lösungen aus der letzten Applikation zu ergänzen.

Planung des Evolutionsschrittes

Aufgrund der Mängel wird im Rahmen der Durchführung des Evolutionsschrittes die Verbesserung der Mehrfachverwendbarkeit bestehender Ergebnisse vorgenommen.

Entwicklung einer neuen Instanz

Zur Verbesserung der bestehenden domänenspezifischen Softwarearchitektur und der dazugehörigen Softwarekomponenten ist eine neue Variante zu entwickeln, welche die neuen bzw. veränderten Anforderungen erfüllt. Zunächst wird die Instanz aus den bestehenden Ergebnissen erstellt, die den gestellten Anforderungen am nächsten kommt. Anschließend wird sie an die Anforderungen angepasst. Für die Kommunikation wird bei der neuen Applikation das Protokoll CANopen eingesetzt. CANopen nutzt CAN als Verbindungsschicht. Deshalb wird CAN um die Funktionalität CANopen erweitert, indem eine neue optionale Schicht zur Umsetzung von CANopen oberhalb des *CANDriver* eingefügt wird. Alle Teile der Applikation, die bisher den *CANDriver* nutzen, müssen an die neue Schnittstelle angepasst werden. Die bestehende Softwarekomponente *BasicMove* ist nicht auf den Einsatz für die neue Automatisierungsaufgabe vorbereitet. Die Instanz dieser Softwarekomponente muss deshalb an die Bewegung des 3-Achs-Portals angepasst werden. Während der Entwicklung der neuen Variante wird in der Liste der Anpassungen dokumentiert, welche Instanzen bzw. welche Teile der Instanzen geändert werden.

Nach der Entwicklung der neuen Variante sind die neu hinzugekommenen Unterschiede gemäß Abschnitt 7.6 in die bestehenden Ergebnisse zu integrieren. Welche Teile von den Anpassungen betroffen sind und welche Softwarekomponenten verbessert werden müssen, wird aus der Liste der Anpassungen abgelesen. Entsprechend sind die folgenden Teilaktivitäten zur Erweiterung der Ergebnisse der Domänenanalyse, des Entwurfs und der Implementierung für alle betroffenen Softwarekomponenten durchzuführen.

Erweiterung der Softwarekomponente CANDriver

CAN und CANopen sind prozessnahe Kommunikationssysteme und werden alternativ zueinander eingesetzt. Da CANopen jedoch CAN als Verbindungsschicht nutzt, wird CANopen als Option in das bestehende Merkmaldiagramm eingetragen. Das erweiterte Merkmaldiagramm ist in Abbildung 8.17 dargestellt. Die Merkmal-Applikations-Matrix wird ebenfalls ergänzt.

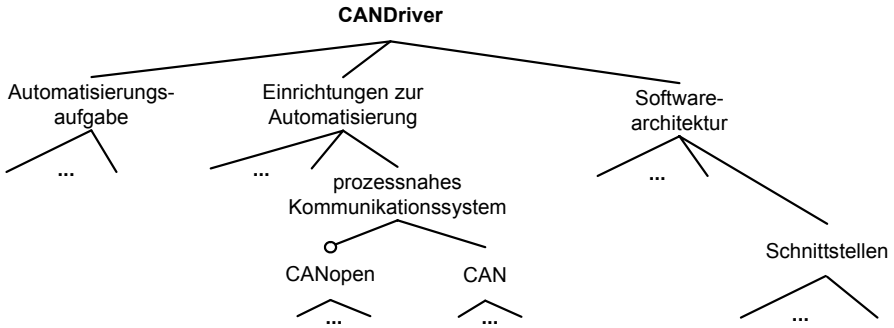


Abbildung 8.17: Erweitertes Merkmaldiagramm der Softwarekomponente *CANDriver*

Der Entwurf der Konfigurationsmöglichkeiten wird gemäß der Analyseergebnisse erweitert. CANopen wird als Option der Softwarekomponente *CANDriver* realisiert. Abbildung 8.18 zeigt die resultierende Softwarearchitektur. Die Implementierung erfolgt gemäß den vorgestellten Mechanismen, ebenso die Erweiterung der Parameter-Kombinations-Matrix.

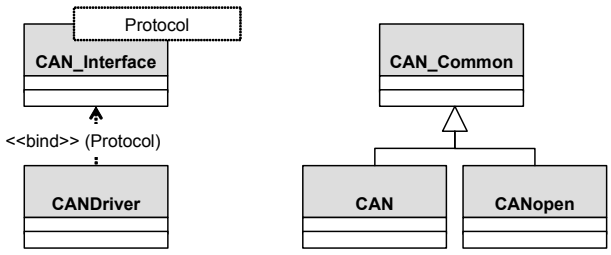


Abbildung 8.18: Erweiterte Softwarearchitektur der Softwarekomponente *CANDriver*

Die Integration der erweiterten Softwarekomponente ergibt keine Änderung in der bestehenden domänenspezifischen Softwarearchitektur. Im Fall der Konfiguration von CANopen als Protokoll ändert sich die Schnittstelle zu den darüber liegenden Schichten.

Erweiterung der Softwarekomponente BasicMove

Die Erweiterung der Softwarekomponente *BasicMove* erfolgt analog zur Softwarekomponente *CANDriver*. Die bestehende Softwarearchitektur der Softwarekomponente wird um einen alternativen Sortiermechanismus für die Bewegung des 3-Achs-Portals erweitert. Die resultierende Softwarearchitektur ist in Abbildung 8.19 dargestellt.

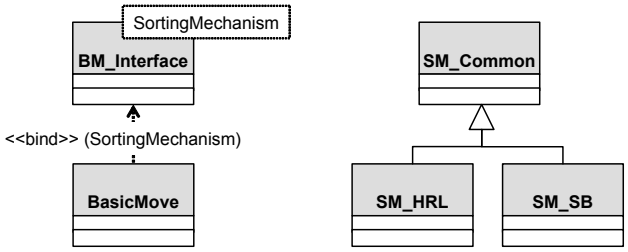


Abbildung 8.19: Erweiterte Softwarearchitektur der Softwarekomponente *BasicMove*

Die Abhängigkeiten zwischen den Softwarekomponenten *CANDriver* und *BasicMove* müssen erweitert werden. Wird das 3-Achs-Portal als Sortiermechanismus gewählt, ist das Protokoll CANopen zwingend erforderlich. Der Zusammenhang wird bei der Erweiterung der Parametersätze und der Softwarekomponentensätze berücksichtigt. Die Softwarekomponenten-Kombinations-Matrix ist um die neuen Möglichkeiten zu erweitern. Damit stehen die verbesserten Softwarekomponenten und die domänenspezifische Softwarearchitektur der 3. Generation für den Einsatz bei der Entwicklung der aktuellen Applikation bereit.

8.7 Ergebnisse und Beschreibung der weiteren Entwicklung

Ergebnisse des Anwendungsbeispiels

Das Anwendungsbeispiel macht die separate Entwicklung der mehrfach verwendbaren Softwarekomponenten deutlich, die nach ihrer Fertigstellung sofort für die Entwicklung von Applikationen zur Verfügung stehen. Die vorgestellte Methodik ermöglicht damit die geforderte zeitliche Verteilung des Entwicklungsaufwandes und die frühe Nutzung von mehrfach verwendbaren Teilergebnissen für die Applikationsentwicklung. Außerdem wird die schrittweise Umstellung der Applikationsentwicklung auf den Einsatz mehrfach verwendbarer Softwarekomponenten und einer domänenspezifischen Softwarearchitektur gezeigt.

Bei der Entwicklung der mehrfach verwendbaren Softwarekomponenten und der domänenspezifischen Softwarearchitektur fließen Informationen über die Automatisierungsaufgaben und die benutzten Einrichtungen zur Automatisierung ein. Die Entwickler werden dabei durch den Fragenkatalog in Anhang A unterstützt.

Beschreibung der weiteren Entwicklung

Die drei vorgestellten Evolutionsschritte des Anwendungsbeispiels zeigen einen Ausschnitt aus der Entwicklung der gesamten Domäne. In nachfolgenden Evolutionsschritten entstehen weitere mehrfach verwendbare Softwarekomponenten sowie eine immer umfangreichere domänenspezifische Softwarearchitektur. Hinsichtlich des Entwicklungsaufwands für evolutionäres Domain-Engineering und Domain-Engineering sowie für die Applikationsentwicklung ergeben sich folgende Zusammenhänge:

- Beim evolutionären Domain-Engineering verteilt sich der Entwicklungsaufwand über mehrere Generationen. Beim Domain-Engineering ist der gesamte Entwicklungsaufwand im Voraus zu erbringen (vgl. Abschnitt 2.3.1).
- Das Einsparungspotenzial durch die Verringerung des Entwicklungsaufwandes bei der Applikationsentwicklung durch die Anwendung der Ergebnisse des evolutionären Domain-Engineering steigt mit jeder neuen Softwarekomponente. Beim Domain-Engineering stehen die im Voraus entwickelten Softwarekomponenten und die domänenspezifische Softwarearchitektur für die Applikationsentwicklung bereits zu Beginn vollständig zur Verfügung. Die Einsparungen beim Entwicklungsaufwand für die Applikationsentwicklung erreichen deshalb gleich zu Beginn ihr Maximum (vgl. Abschnitt 2.3.1).
- Da beim evolutionären Domain-Engineering der Entwicklungsaufwand später investiert wird und die Einsparungen bei der Applikationsentwicklung zu Beginn geringer sind als beim Domain-Engineering, verzögert sich die Amortisierung des investierten Entwicklungsaufwandes im Vergleich zum Domain-Engineering (späterer Return-of-Investment).
- Durch wiederkehrende Aktivitäten zur Projektplanung oder Organisation, wie beispielsweise der Planung des Evolutionsschrittes, ist beim evolutionären Domain-Engineering langfristig mit einem höheren Entwicklungsaufwand zu rechnen als beim Domain-Engineering.

In Abschnitt 2.3.1 wird der Entwicklungsaufwand mit und ohne Domain-Engineering verglichen (vgl. Abbildung 2.6). Abbildung 8.20 zeigt diesen Vergleich erweitert um den Entwicklungsaufwand, der sich durch die Anwendung des evolutionären Domain-Engineering ergibt. Die Unterschiede zwischen evolutionärem Domain-Engineering und Domain-Engineering werden insbesondere an den folgenden Eigenschaften der Kurven deutlich:

- Startwerte der Kurven: Vorleistung an Entwicklungsaufwand bei Domain-Engineering bzw. bei evolutionärem Domain-Engineering

- Steigungen der Kurven:
    - Mit und ohne Domain-Engineering: Entwicklungsaufwand für die Entwicklung neuer Applikationen mit und ohne Mehrfachverwendung
    - Mit evolutionärem Domain-Engineering: Summe des Entwicklungsaufwandes für die Entwicklung neuer Applikationen (mit den verfügbaren Ergebnissen des evolutionären Domain-Engineering) und des Entwicklungsaufwandes für die Durchführung der Iterationen des evolutionären Domain-Engineering. Ist weder eine Erweiterung noch eine Verbesserung der mehrfach verwendbaren Ergebnisse notwendig, entfällt der Entwicklungsaufwand für die Durchführung der Iterationen. Dadurch verringert sich die Steigung.
- Schnittpunkte mit der Kurven für den kumulierten Entwicklungsaufwand ohne Domain-Engineering: Return-of-Investment bei Domain-Engineering bzw. bei evolutionärem Domain-Engineering

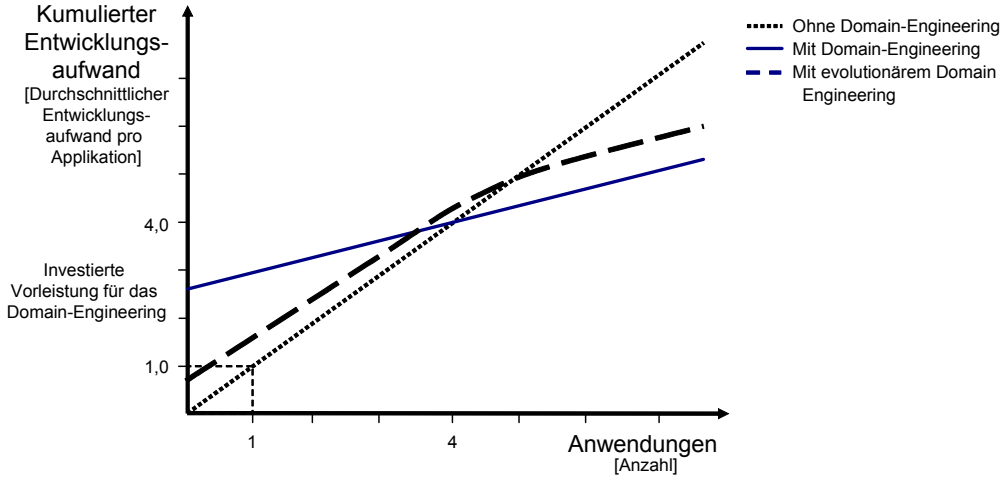


Abbildung 8.20: Entwicklungsaufwand mit und ohne Domain-Engineering sowie mit evolutionärem Domain-Engineering

9 Zusammenfassung und Ausblick

9.1 Ergebnisse

In der vorliegenden Arbeit wurde eine Methodik für das evolutionäre Domain-Engineering zur Entwicklung von Automatisierungssystemen konzipiert. Dem evolutionären Domain-Engineering liegt das Evolutionsprinzip zugrunde. Außerdem fließen spezifische Informationen über Automatisierungssysteme ein.

Das Evolutionsprinzip bedingt ein iteratives Vorgehen bei der Entwicklung von mehrfach verwendbaren Softwarekomponenten und einer domänenspezifischen Softwarearchitektur. Über die Iterationen hinweg werden mehr und mehr Softwarekomponenten separat entwickelt und nacheinander in die domänenspezifische Softwarearchitektur integriert. Im Rahmen der Analyse wird in bestehenden Applikationen nach Lösungen gesucht, die das Potenzial besitzen, mehrfach verwendet zu werden. Dieses Potenzial wird zum einen anhand des Bedarfs an diesen Lösungen bei der Entwicklung zukünftiger Applikationen und zum anderen anhand der Häufigkeit der Verwendung ähnlicher oder alternativer Lösungen in bestehenden Applikationen abgeschätzt. Dabei wird insbesondere auf das Wissen der Entwickler über bestehende Applikationen sowie auf die Systemdokumentation dieser Applikationen zurückgegriffen. Die Lösungen mit einem hohen Potenzial mehrfach verwendet zu werden, dienen als Basis für die Entwicklung mehrfach verwendbarer Softwarekomponenten sowie einer domänenspezifischen Softwarearchitektur. Dazu werden zunächst die Unterschiede und die Gemeinsamkeiten ähnlicher und alternativer Lösungen analysiert. Die gefundenen Unterschiede werden im Rahmen des Entwurfs und der Implementierung durch Variationspunkte abstrahiert. Zur einfachen Anwendung der Ergebnisse werden Konfigurationsmöglichkeiten der mehrfach verwendbaren Softwarekomponenten und der domänenspezifischen Softwarearchitektur eingeführt. Die mehrfach verwendbaren Ergebnisse eines Iterationsschrittes können sofort in die Entwicklung neuer Applikationen einfließen.

Über die Entwicklung neuer mehrfach verwendbarer Ergebnisse hinaus bietet das evolutionäre Domain-Engineering auch Unterstützung bei der Verbesserung der Mehrfachverwendbarkeit bestehender Ergebnisse. Dabei wird bei jedem Evolutionsschritt zunächst die Mehrfach-verwendbarkeit der bisherigen Ergebnisse überprüft. Erkannte Mängel bezüglich der Mehrfach-verwendbarkeit werden in den bestehenden Ergebnissen gezielt behoben.

Bei der Entwicklung mehrfach verwendbarer Softwarekomponenten und domänenspezifischer Softwarearchitekturen für den Einsatz in Automatisierungssystemen sind insbesondere die spezifischen Merkmale von Automatisierungssystemen, wie Echtzeiteigenschaften, Schnitt-stellen zu Rechnerhardware und Prozessperipherie, usw., zu berücksichtigen. Diese spezifischen Merkmale ergeben sich aus den Automatisierungsaufgaben und den Einrichtungen, die zur

Automatisierung eingesetzt werden. Beim evolutionären Domain-Engineering werden die Entwickler bei der Suche nach Unterschieden und Gemeinsamkeiten durch einen Fragenkatalog mit Fragen zu Automatisierungsaufgaben und zu Einrichtungen zur Automatisierung unterstützt.

9.2 Voraussetzungen und Grenzen der Anwendbarkeit

Das evolutionäre Domain-Engineering nutzt bestehende Applikationen als Basis für die Entwicklung mehrfach verwendbarer Softwarekomponenten und einer domänenspezifischen Softwarearchitektur. Je mehr bestehende Applikationen zur Verfügung stehen, desto zuverlässiger kann die Mehrfachverwendbarkeit enthaltener Lösungen für zukünftige Applikationen abgeschätzt werden und desto größer ist die Menge bestehender Lösungen, die für die Entwicklung mehrfach verwendbarer Softwarekomponenten genutzt werden kann.

Das evolutionäre Domain-Engineering nutzt Gemeinsamkeiten bestehender Applikationen innerhalb der Domäne. Diese Gemeinsamkeiten werden in Softwarekomponenten und Softwarearchitekturen für die Entwicklung zukünftiger Applikationen anwendbar gemacht. Weisen die bestehenden Applikationen einer Domäne nur wenige Gemeinsamkeiten auf, müssen umfangreiche Konfigurationsmöglichkeiten bei der Entwicklung der mehrfach verwendbaren Ergebnisse vorgesehen werden. Durch die nötigen Konfigurationsmöglichkeiten steigt der Entwicklungsaufwand, der Nutzen bleibt konstant. Der höhere Entwicklungsaufwand verzögert die Amortisierung. Das bedeutet, dass ein kleiner Anteil an Gemeinsamkeiten innerhalb der betrachteten Domäne den wirtschaftlichen Nutzen des evolutionären Domain-Engineering senkt.

Ein umfangreicher und schneller Wechsel von Anforderungen wirkt sich ebenfalls negativ auf den wirtschaftlichen Nutzen des evolutionären Domain-Engineering aus. Kommen neue Anforderungen hinzu, werden diese im Rahmen der „Verbesserung der Mehrfachverwendbarkeit“ in die bestehenden Ergebnisse eingearbeitet. Der Entwicklungsaufwand dafür ist abhängig von den Auswirkungen der Anforderungen auf Entwurf und Implementierung der bestehenden Ergebnisse. Das heißt, dass Entwicklungsaufwand in die mehrfach verwendbaren Ergebnisse investiert werden muss, um ihre Mehrfachverwendbarkeit aufrecht zu erhalten. Gleichzeitig wird die Entwicklung neuer Softwarekomponenten und die Erweiterung der domänenspezifischen Softwarearchitektur verzögert. Beides behindert eine frühe Amortisierung der mehrfach verwendbaren Ergebnisse und senkt damit den wirtschaftlichen Nutzen des evolutionären Domain-Engineering.

9.3 Bewertung und Anwendbarkeit

In Kapitel 3 wird die Anwendung von Domain-Engineering in KMU als schwierig bewertet. Die Begründung basiert auf dem Entwicklungsaufwand, der Organisationsstruktur sowie der Softwarearchitektur und der Implementierung bei Domain-Engineering. Im Folgenden sind die

wesentlichen Vor- und Nachteile des evolutionären Domain-Engineering gegenüber dem Domain-Engineering bezüglich der Anwendbarkeit in KMU dargestellt:

- Durch die schrittweise Vervollständigung der Bibliothek mehrfach verwendbarer Softwarekomponenten verzögert sich die Verfügbarkeit einer vollständigen Bibliothek. Damit ist der Nutzen bei der Entwicklung neuer Applikationen zunächst gering und wird dann schrittweise vergrößert.
- Durch die zeitliche Verteilung des Entwicklungsaufwandes für die mehrfach verwendbaren Ergebnisse werden die notwendigen Investitionen zeitlich verteilt. Damit wird die finanzielle Belastung zeitlich verteilt.
- Die frühe Nutzbarkeit von mehrfach verwendbaren Teilergebnissen ermöglicht eine frühe Amortisierung dieser Teilergebnisse.
- Die schrittweise Einführung einer domänenspezifischen Softwarearchitektur verringert das Risiko gegenüber der abrupten Einführung einer neuen Softwarearchitektur.
- Die ständige Verbesserung der mehrfach verwendbaren Ergebnisse erlaubt eine schnelle und flexible Anpassung an neue Anforderungen an Applikationen der Domäne. Damit wird die Mehrfachverwendbarkeit der Ergebnisse aufrecht erhalten.

Insbesondere durch die zeitliche Verteilung des Entwicklungsaufwandes und die frühe Nutzung von Teilergebnissen eignet sich das evolutionäre Domain-Engineering besser für den Einsatz in KMU als nicht am Evolutionsprinzip orientierte Domain-Engineering-Methoden.

In Kapitel 3 wird bei Domain-Engineering das Fehlen einer gezielten Unterstützung für Domänen der Automatisierungssysteme festgestellt. Die wesentlichen Vorteile des evolutionären Domain-Engineering gegenüber dem Domain-Engineering bezüglich der Anwendbarkeit für Automatisierungssysteme liegen in der gezielten Unterstützung der Entwickler bei der Analyse bestehender Applikationen bezüglich der Unterschiede und Gemeinsamkeiten, die für Automatisierungssysteme relevant sind. Bei der Suche nach ähnlichen und alternativen Merkmalen hilft ein Fragenkatalog mit Fragen zu Automatisierungsaufgaben und zu Einrichtungen zur Automatisierung.

Durch die Berücksichtigung der für Automatisierungssysteme relevanten Unterschiede und Gemeinsamkeiten erfüllt das evolutionäre Domain-Engineering eine wichtige Voraussetzung für den Einsatz zur Entwicklung von mehrfach verwendbarer Software für Automatisierungssysteme.

9.4 Ausblick

Während der Durchführung der Arbeit und auf Basis der gewonnenen Erfahrungen wurden einige Punkte erkannt, die Ansätze für die Fortführung der Arbeit bieten. Die beiden wichtigsten sind die Erstellung von Implementierungsmustern zur Umsetzung von Template-Klassen sowie die Unterstützung des evolutionären Domain-Engineering durch Softwarewerkzeuge.

Im Rahmen dieser Arbeit wurden Template-Klassen zur Modellierung von Variationspunkten in der Softwarearchitektur mehrfach verwendbarer Softwarekomponenten eingesetzt. Unterstützung zur Umsetzung von Template-Klassen in der Implementierung bietet nur die Programmiersprache C++. Sollen andere Programmiersprachen genutzt werden, wären Implementierungsmuster hilfreich, die beschreiben, wie Template-Klassen in einer Softwarearchitektur bei der Implementierung in der betreffenden Programmiersprache umzusetzen sind.

Die Methodik beinhaltet die Erstellung unterschiedlicher Ausgangsprodukte. Für die grafischen Notationen, wie beispielsweise das Merkmaldiagramm, wären Editoren für den Anwender ebenso hilfreich wie für die Bearbeitung der verschiedenen Matrizen.

Bei der Applikationsentwicklung werden insbesondere die domänenspezifische Softwarearchitektur und die mehrfach verwendbaren Softwarekomponenten eingesetzt. Dabei sind für die jeweilige Applikation geeignete Softwarekomponenten auszuwählen. Zudem sind die Softwarekomponenten und die domänenspezifische Softwarearchitektur passend zu konfigurieren. Für die zentrale Ablage der mehrfach verwendbaren Ergebnisse in einer Bibliothek sowie zur Unterstützung bei der Auswahl passender Softwarekomponenten wäre die Unterstützung durch ein Komponenten-Management-System hilfreich, wie es beispielsweise in [Luce02] beschrieben wird. Zur Unterstützung der Applikationsentwickler bei der Konfiguration mehrfach verwendbarer Ergebnisse wäre eine Werkzeugunterstützung ebenfalls hilfreich. Dazu könnte ggf. das in [Dujm01] beschriebene Werkzeug FIT zur Konfiguration von Komponenten-Frameworks in Betracht gezogen werden.

Anhang: Fragenkatalog

Die Gliederung dieses Fragenkatalogs sowie die Fragen wurden in Anlehnung an [Göhn05a] und [Göhn05b] entwickelt. Die Fragen sind ggf. um domänenspezifische Fragen zu erweitern.

Fragen zu Automatisierungsaufgaben

Nr.	Fragen	Erwartete Antwort(en)	Weiterführende Fragen [Nr.]
1	Welche Grundaufgaben bzw. Teilaufgaben der Prozessautomatisierung werden erfüllt?	Prozesssignalerfassung und -aufbereitung	1.1
		Prozessüberwachung	1.2
		Prozessführung von Fließprozessen	1.3
		Prozessführung von Stück- und Folgeprozessen	1.4
1.1	Welche Prozesssignale sind zu erfassen und aufzubereiten?	Liste mit Prozesssignalen	1.1.1 (für jedes Prozesssignal)
1.1.1	Welche Aufgaben in der Prozesssignalerfassung und -aufbereitung sind zu erfüllen?	Prozesssignaldurchschaltung	1.1.1.1
		AD-Umsetzung	2.2
		Anpassung	1.1.1.2
		Filterung und Glättung	1.1.1.3
		Plausibilitätsprüfung	1.1.1.4
1.1.1.1	Wann erfolgt die Prozesssignaldurchschaltung?	zyklisch	1.1.1.1.1
		azyklisch	1.1.1.1.2
1.1.1.1.1	Welche Merkmale besitzt die zyklische Abfrage?	Zykluszeit für die Abfrage, ...	
1.1.1.1.2	Welche Merkmale besitzt die azyklische Abfrage?	Auftretenswahrscheinlichkeit spontaner Signale, Zeitdauer für ihre Abfrage, ...	
1.1.1.2	Welche Art von Kennlinien sind anzupassen?	lineare Kennlinie	1.1.1.2.1
		nicht-lineare Kennlinie	1.1.1.2.2
1.1.1.2.1	Welche Merkmale besitzt die lineare Kennlinie?	Nullpunktverschiebung, Steigung, Umrechnung in technische Einheit, ...	
1.1.1.2.2	Welche Merkmale besitzt die nicht-lineare Kennlinie?	Approximation der nicht-linearen Funktion durch Polynom-Ansatz (Polynom), Speicherung von Stützstellen der Funktion (Lineare Interpolation)	

1.1.1.3	Wie werden Störeinflüsse eliminiert?	Filterung	1.1.1.3.1
		Glättung	1.1.1.3.2
1.1.1.3.1	Welche Merkmale besitzt die Filterung?	Eingesetzter Filteralgorithmus (z.B. Mittelwertbildung, Kalmanfilter, Kammfilter, ...), Parameter für die Einstellung des jeweiligen Filteralgorithmus, ...	
1.1.1.3.2	Welche Merkmale besitzt die Glättung?	Parameter für die Einstellung der Glättung	
1.1.1.4	Welches Verfahren zur Plausibilitätsprüfung wird eingesetzt?	Statische Plausibilitätsprüfung	1.1.1.4.1
		Dynamische Plausibilitätsprüfung	1.1.1.4.2
		Plausibilitätsprüfung durch Sekundärwertprüfung	1.1.1.4.3
		Plausibilitätsprüfung durch redundante Sensoren	1.1.1.4.4
		Plausibilitätsprüfung durch Mehrfacherfassung	1.1.1.4.5
		Plausibilitätsprüfung durch Vergleich mit Eichwerten	1.1.1.4.6
		Plausibilitätsprüfung anhand eines Prozessmodells	1.1.1.4.7
		Keine Plausibilitätsprüfung	
1.1.1.4.1	Welche Merkmale besitzt die statische Plausibilitätsprüfung?	Physikalisch möglicher Wertebereich, ...	
1.1.1.4.2	Welche Merkmale besitzt die dynamische Plausibilitätsprüfung?	Differenzenquotient, ...	
1.1.1.4.3	Welche Merkmale besitzt die Plausibilitätsprüfung durch Sekundärwertprüfung?	Primärwertzuordnung, Grenzen, Bedingungen und Zusammenhänge für Primärwert und Sekundärwert	
1.1.1.4.4	Welche Merkmale besitzt die Plausibilitätsprüfung durch redundante Sensoren?	Prozesssignale für die 2-aus-3-Auswahl, ...	
1.1.1.4.5	Welche Merkmale besitzt die Plausibilitätsprüfung durch Mehrfacherfassung?	Zeitabstände, ...	
1.1.1.4.6	Welche Merkmale besitzt die Plausibilitätsprüfung durch Vergleich mit Eichwerten?	Eichspannungsquelle, Umschaltzeiten, ...	
1.1.1.4.7	Welche Merkmale besitzt die Plausibilitätsprüfung anhand eines Prozessmodells?	Prozessmodell, ...	
1.2	Welche Prozesse sind zu überwachen?	Liste mit zu überwachenden Prozessen	1.2.1 (für jeden Prozess)
1.2.1	Welcher Zeitraum ist zu überwachen?	Prozessvergangenheit, -gegenwart oder -zukunft	1.2.2

1.2.2	Welche Art der der Prozessüberwachung ist durchzuführen?	signalorientierte Prozessüberwachung (einzelne Prozesssignale)	1.2.2.1
		Informationsorientierte Prozessüberwachung (Zusammenwirken von Prozesssignalen)	1.2.2.2
1.2.2.1	Welche Prozesssignale sind zu überwachen?	Liste mit zu überwachenden Prozesssignalen	1.2.2.1.1 (für jedes Prozesssignal)
1.2.2.1.1	Welcher Art sind die Prozesssignale?	Kontinuierliche Prozesssignale	1.2.2.1.1.1
		Binäre Prozesssignale	1.2.2.1.1.2
1.2.2.1.1.1	Welche Eigenschaften kontinuierlicher Prozesssignale sind zu überwachen?	Einhaltung fester Grenzen	1.2.2.1.1.1.1
		Einhaltung gleitender Grenzen	1.2.2.1.1.1.2
		Einhaltung zeitlicher Änderungsraten	1.2.2.1.1.1.3
		Kombinationen	1.2.2.1.1.1.x (siehe oben)
1.2.2.1.1.1.1	Welche Merkmale besitzt die Prüfung auf Einhaltung fester Grenzen?	Einzuhaltende Grenze, Toleranzen, ...	
1.2.2.1.1.1.2	Welche Merkmale besitzt die Prüfung auf Einhaltung gleitender Grenzen?	Verlauf der einzuhaltenden Grenze, Toleranzen, ...	
1.2.2.1.1.1.3	Welche Merkmale besitzt die Prüfung auf Einhaltung zeitlicher Änderungsraten?	Differenzenquotient, Toleranzen, ...	
1.2.2.1.1.2	Welche Merkmale besitzen die zu überwachenden binären Prozesssignale?	Zuverlässigkeitstabellen, ...	
1.2.2.2	Welches Verfahren zum direkten Modellvergleich wird eingesetzt?	Vergleich durch globale Kenngrößen	1.2.2.2.1
		Vergleich durch Zustandsschätzung	1.2.2.2.2
		Vergleich durch Identifikationsverfahren	1.2.2.2.3
1.2.2.2.1	Welche Merkmale besitzt der Vergleich durch globale Kenngrößen?	Kenngrößenmodell, Kenngrößen, ...	
1.2.2.2.2	Welche Merkmale besitzt der Vergleich durch Zustandsschätzung?	Modell des fehlerhaften Verhaltens, Modell des technischen Prozesses in der idealen, fehlerfreien Anlage, Zustandsbeobachter, geschätzte Größen, ...	

1.2.2.2.3	Welche Merkmale besitzt der Vergleich durch das Identifikationsverfahren?	Charakteristische Änderungen der Systemkoeffizienten bei Fehlern und Ausfällen, Normalwerte der physikalischen Systemkoeffizienten, geschätzte Systemkoeffizienten, Modell des technischen Prozesses mit geschätzten Parametern, Parameterschätzung, ...	
1.3	Welche Fließprozesse sind zu führen?	Liste der Fließprozesse	1.3.1 (für jeden Prozess)
1.3.1	Welche Merkmale haben die zu führenden Fließprozesse?	Differenzial- oder Differenzengleichungen (z.B. als Signalflussplan), Zustandsgleichungen, ...	1.3.2
1.3.2	Welche Eingangsgrößen bestehen, um auf den Prozess einzuwirken?	Liste der Eingangsgrößen	1.3.3
1.3.3	Welche Ausgangsgrößen bestehen, die den Ablauf des technischen Prozesses und dessen Ergebnisse kennzeichnen?	Liste der Ausgangsgrößen	1.3.4
1.3.4	Welche Störgrößen bestehen, die den Ablauf des technischen Prozesses beeinflussen könnten?	Liste der Störgrößen	1.3.5
1.3.5	Welche Strategie zur Prozessführung der Fließprozesse wird angewandt?	Steuerungsstrategie	1.3.5.1
		Regelungsstrategie	1.3.5.2
1.3.5.1	Welche Zusammenhänge zwischen Eingangs- und Ausgangsgrößen bestehen?	Differenzial- oder Differenzengleichungen (z.B. als Signalflussplan), Zustandsgleichungen, ... (siehe auch Merkmale der zu führenden Fließprozesse)	
1.3.5.2	Wie lautet die Führungsübertragungsfunktion und die Störübertragungsfunktion?	jeweilige Übertragungsfunktion	1.3.5.2.1
1.3.5.2.1	Welche Art von Regler wird eingesetzt?	PID-Regler, kaskadierende Regelerstrukturen, Abtastregelung, Zustandsregelungen, Zustandsbeobachter, prädiktive Regelungen, adaptive Regelungen, ...	1.3.5.2.2
1.3.5.2.2	Welche Merkmale besitzt der eingesetzte Regler?	Reglerstruktur, Parameter, ...	
1.4	Welche Stück- und Folgeprozesse sind zu führen?	Liste der Stück- und Folgeprozesse	1.4.1 (für jeden Prozess)

1.4.1	Welche Merkmale haben die zu führenden Stück- und Folgeprozesse?	Zustandsmodelle, ...	1.4.2
1.4.2	Welche Eingangsgrößen bestehen, um auf den Prozess einzuwirken?	Liste der Eingangsgrößen	1.4.3
1.4.3	Welche Ausgangsgrößen bestehen, die den Ablauf des technischen Prozesses und dessen Ergebnisse kennzeichnen?	Liste der Ausgangsgrößen	1.4.4
1.4.4	Welche Störgrößen bestehen, die den Ablauf des technischen Prozesses beeinflussen könnten?	Liste der Störgrößen	1.4.5
1.4.5	Welches Verfahren zur diskreten Steuerung der Stück- und Folgeprozesse wird angewandt?	Verknüpfungssteuerung	1.4.5.1
		Ablaufsteuerung	1.4.5.2
1.4.5.1	Welche zustandslosen Verknüpfungen von Eingangs- zu Ausgangssignalen bestehen?	Logikverknüpfungen	
1.4.5.2	Welche spezifische Abfolge besteht (Zustände und Weiterschaltbedingungen)?	Endliche Automaten, Zustandsgraphen (State Charts), Zustandstabellen, Schrittketten, Petri-Netze, ...	

Fragen zu Einrichtungen zur Automatisierung

Nr.	**Fragen**	**Erwartete Antwort(en)**	**Weiterführende Fragen [Nr.]**
2	Welche Einrichtungen zur Automatisierung werden eingesetzt?	Automatisierungscomputer	2.1
		Prozessperipherie (Sensoren und Aktoren)	2.2
		Kommunikationssystem	2.3
		Einrichtungen für die Mensch-Prozess-Kommunikation	2.4
2.1	Welche Art von Automatisierungscomputer wird eingesetzt?	Speicherprogrammierbare Steuerung (SPS)	2.1.1
		Mikrocontroller	2.1.2
		Industrie PC (IPC)	2.1.3
		Prozessleitsystem	2.1.4

2.1.1	Welche Merkmale haben die Bestandteile der eingesetzten SPS?	Programmspeicher (Größe, ...)	
		RAM-Speicher (Größe, ...)	
		Zeitgeber (Zeitabstände, ...)	
		Schnittstelle zum Programmiergerät (Protokoll, Adresse, ...)	
		Digital- und Analog-Eingabe und -Ausgabe (Anzahl, E/A-Adressen, ...)	
		Sonstige Bestandteile und ihre Merkmale	
2.1.2	Welche Merkmale haben die Bestandteile des eingesetzten Mikrocontroller?	CPU und interner Bus (Registerbreite, ...)	
		Speicher (Größe, Adressierungsart, Anordnung der Daten im Speicher, ...)	
		Zeitüberwachung (Zeitintervall)	
		Taktgeber (Frequenz, Art der Benachrichtigung der Software über das Auftreten eines Taktes, ...)	
		Digitale Ein- / Ausgänge (Anzahl, Adressen, ...)	
		Analoge Eingänge (Anzahl, Adressen, Auflösung, Abtastfrequenz, ...)	
		Serielle Ein-/Ausgabeschnittstellen (Geschwindigkeit, Paritätsprüfung, ...)	
		Impuls-Ein-/Ausgabe (Zeitabstände zwischen den Impulsen, ...)	
		Interrupt-Controller (Freischaltung, Zuordnung der Interrupt-Routinen, ...)	
		Externer Datenbus (Protokoll, Geschwindigkeit, Knotenadresse, ...)	
		Feldbusschnittstelle (Protokoll, Geschwindigkeit, Knotenadresse, ...)	
		Sonstige Bestandteile und ihre Merkmale	

2.1.3	Welche Merkmale haben die Bestandteile des eingesetzten IPC?	Serielle Verbindung, LAN oder Feldbus (Anzahl, Protokolle, Geschwindigkeit, Knotenadressen, ...)	
		SPS mit IPC-Anschaltgruppe und Signal-Ein-/Ausgabegruppen	2.1.1
		Module für die Signal-Ein-/Ausgabe (Anzahl der Module, Moduladressen, Anzahl der E/As pro Modul, E/A-Adressen, ...)	
		Sonstige Bestandteile und ihre Merkmale	
2.1.4	Welche Merkmale haben die Bestandteile des eingesetzten Prozessleitsystems?	Anzeige und Bedienkomponenten	2.1.3
		Prozessnahe Komponenten	2.1.1 und 2.1.3
		Sonstige Bestandteile und ihre Merkmale	
2.2	Welche Prozesssignale werden an den Schnittstellen bereitgestellt?	Liste von Prozesssignalen	
2.2.1	Welcher Art sind die Prozesssignale?	Binäre bzw. digitale Prozesssignale (Eingang)	2.2.1.1
		Binäre bzw. digitale Prozesssignale (Ausgang)	2.2.1.2
		Analoge Prozesssignale (Eingang)	2.2.1.3
		Analoge Prozesssignale (Ausgang)	2.2.1.4
2.2.1.1	Welche Art von Umsetzer wird für die Umwandlung des binären bzw. digitalen Eingangssignals eingesetzt?	Statische Digitaleingabe	
		Dynamische Digitaleingabe	2.2.1.1.1
		Spontane Digitaleingabe	2.2.1.1.2
2.2.1.1.1	Welche Merkmale hat die dynamische Digitaleingabe?	Steigende Flanke, fallende Flanke, beides, ...	
2.2.1.1.2	Welche Merkmale hat die spontane Digitaleingabe?	Zuordnung zu Interrupt, ...	
2.2.1.2	Welche Art von Umsetzer wird für die Umwandlung des binären bzw. digitalen Ausgangssignals eingesetzt?	Spannungsausgabe, Stromausgabe, Ausgabe potenzialfreier Kontaktstellen, ...	
2.2.1.3	Welche Art von Umsetzer wird für die Umwandlung des analogen Eingangssignals eingesetzt?	Integrierender Umsetzer	2.2.1.3.1
		Momentanwertumsetzer	2.2.1.3.2
2.2.1.3.1	Nach welchem Verfahren arbeitet der integrierende Umsetzer?	Dual-Slope Verfahren	2.2.1.3.1.1
2.2.1.3.1.1	Welche Merkmale hat der integrierende Umsetzer nach dem Dual-Slope Verfahren?	Taktfrequenz, Kapazität des Kondensators, Widerstand, Referenzspannung, Eingangsspannung, ...	

2.2.1.3.2	Nach welchem Verfahren arbeitet der Momentanwertumsetzer?	Zählverfahren	2.2.1.3.2.1
		Stufenverfahren	2.2.1.3.2.2
		Parallelumsetzverfahren	2.2.1.3.2.3
2.2.1.3.2.1	Welche Merkmale hat der Momentanwertumsetzer nach dem Zählverfahren?	Taktfrequenz, Wortlänge des digitalen Ausgangs, Eingangsspannung, Einschwingverhalten des Digital-Analog-Umsetzers, Schaltzeit des Komperators, ...	
2.2.1.3.2.2	Welche Merkmale hat der Momentanwertumsetzer nach dem Stufenverfahren?	Taktfrequenz, Wortlänge des digitalen Ausgangs, Eingangsspannung, Einschwingverhalten des Digital-Analog-Umsetzers, Schaltzeit des Komperators, ...	
2.2.1.3.2.3	Welche Merkmale hat der Momentanwertumsetzer nach dem Parallelumsetzverfahren?	Anzahl der Komperatoren, Eingangsspannung, Schaltzeit der Komperatoren, ...	
2.2.1.4	Welche Art von Umsetzer wird für die Umwandlung des analogen Ausgangssignals eingesetzt?	Summation gewichteter Ströme	2.2.1.4.1
		Leiternetzwerk	2.2.1.4.2
2.2.1.4.1	Welche Merkmale hat der Umsetzer nach dem Verfahren der Summation gewichteter Ströme?	Anzahl der Widerstände, Widerstände, ...	
2.2.1.4.2	Welche Merkmale hat der Umsetzer nach dem Verfahren des Leiternetzwerks?	Anzahl der Schalter, Widerstände, ...	
2.3	Welche Kommunikationssysteme werden eingesetzt?	Liste der Kommunikationssysteme	2.3.1 (für jedes Komm.)
2.3.1	Welcher Art ist das eingesetzte Kommunikationssystem?	Prozessnahes Kommunikationssystem	2.3.1.1
		Kommunikationssystem für die Kommunikation zwischen Automatisierungscomputern	2.3.1.2
2.3.1.1	Welches prozessnahe Kommunikationssystem wird eingesetzt?	direkte Anbindung über Leitungsbündel	2.3.1.1.1
		AS-Interface	2.3.1.1.2
		Interbus-S	2.3.1.1.2
		PROFIBUS	2.3.1.1.2
		CAN	2.3.1.1.2
		CANopen	2.3.1.1.2
		Sonstige (ggf. zu erweitern)	2.3.1.1.2
2.3.1.1.1	Welcher E/A-Adresse sind die Daten zugeordnet?	E/A-Adresse	2.2.1
2.3.1.1.2	Welche Knoten sind vorhanden?	Liste der Knoten mit Adresse	2.3.1.1.2.1

2.3.1.1.2.1	Welche Daten werden zwischen welchen Knoten übertragen?	Liste der Datenobjekte (Adresse, Länge, Inhalt, ...)	2.3.1.1.2.2
2.3.1.1.2.2	Mit welcher Geschwindigkeit wird der Bus betrieben?	Baudrate	
2.3.1.2	Welches Kommunikationssystem für die Kommunikation zwischen Automatisierungscomputern wird eingesetzt?	Prozessnahes Kommunikationssystem	2.3.1.1
		Ethernet	2.3.1.2.1
		Sonstige (ggf. zu erweitern)	2.3.1.2.1
2.3.1.2.1	Welches Knoten sind vorhanden?	Liste der Knoten mit Adresse	2.3.1.2.2
2.3.1.2.2	Welche Daten werden zwischen welchen Knoten übertragen?	Liste der Datenobjekte (Adresse, Länge, Inhalt, ...)	
2.4	Welche Einrichtungen für die Mensch-Prozess-Kommunikation werden eingesetzt?	Liste der Einrichtungen	2.4.1
2.4.1	Welche Merkmale besitzen die Einrichtungen für die Mensch-Prozess-Kommunikation?	Prozesszustandsdiagramme, Übersichtsdarstellungen, Diagnose- und Alarmmeldungen, ...	2.4.2
2.4.2	Auf welche Prozesssignale wird über die Einrichtungen für die Mensch-Prozess-Kommunikation zugegriffen (lesend / schreibend)?	Liste der Prozesssignale	

Literaturverzeichnis

[ABFG00] M. Anastasopoulos, J. Bayer, O. Flege und C. Gacek: A Process For Productline Creation and Evaluation: PuLSE-DSSA, Version 2.0, IESE-Report Nr.038.00/E, Version 1.0, 27.06.2000, Institut für Experimentelles Software Engineering, Fraunhofer Gesellschaft, 2000

[ADH+00] M. Ardis, N. Daley, D. Hoffman, H. Siy und D. Weiss: Software product lines: a case study. In Software – Practice and Experience 2000; 30:825-847, Wiley, New York, NY, 2000

[AnGa00] M. Anastasopoulos and C. Gacek. Implementing Product-Line Variabilities. In Proceedings of the Symposium on Software Reusability (SSR 2001), Toronto, Ontario, Canada, Seiten 109 – 117, 18.-20. Mai 2001

[Arra94] G. Arrango: Domain Analysis Methods. In Software Reusability, Ellis Horwood, New York, NY, Seiten 17-49, 1994

[Auto05] Autosar – Automotive Software Architecture. http:/www.autosar.org/, 2005

[Balz96] Helmut Balzert: Lehrbuch der Software-Technik: Software-Entwicklung. Spektrum Akademischer Verlag, Heidelberg, Berlin, Oxford, 1996

[Balz98] Helmut Balzert: Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung. Spektrum Akademischer Verlag, Heidelberg, Berlin, 1998

[Balz99] Heide Balzert: Lehrbuch der Objektmodellierung: Analyse und Entwurf. Spektrum Akademischer Verlag, Heidelberg, Berlin, 1999

[BFK+99] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Wieden und J.-M. DeBaud: PuLSE: A methodology to develop software product lines. In Proceedings of the Symposium on Software Reuse, Seiten 122-131, Mai 1999

[Boeh88] B. Boehm: A Spiral Model of Software Development and Enhancement. In IEEE Computer, Mai 1988, Seiten 61-72, © IEEE 1988

[BrCl96] L. Brownsword und P. Clements: A Case Study in Successful Product Line Development. Technical Report, CMU/SEI-96-TR-016, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1996

[BrDr93] A.-P. Bröhl und W. Dröschel: Das V-Modell: Der Standard für Softwareentwicklung mit Praxisleitfaden. R. Oldenburg Verlag, München, Wien, 1993

[CARDS94] Software Technologie For Adaptable, Reliable Systems (STARS): Domain Engineering Methods and Tools Handbook: Volume I – Methods: Comperhensive Approach to Reusable Defense Software (CARDS). STARS Informal Technical Report, STARS-VC-K017R1/001/00, 31. Dezember 1994

[CBBG+02] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord und J. Stafford: Documenting Software Architectures: Views and Beyond. Addison-Wesley Reading, MA, 2002

[Chro92] G. Chroust: Modelle der Software-Entwicklung. R. Oldenburg Verlag, München, Wien, 1992

[CIA04] CAN in Automation: CANopen. http://www.can-cia.de, 2004

[Clau01] M. Clauß: Modeling Variability with UML. In Proceedings of GCSE2001 Young Researchers Workshop, 2001

[Corn96] P. Cornwell: HP Domain Analysis: Producing Useful Models for Reusable Software. In Hewlett-Packard Journal, August 1996, Seiten 46-54, 1996

[CSPK92] S. Cohen, J. Stanley, S. Peterson und 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, Pittsburgh, PA, 1992

[CzEi00] K. Czarnecki und U. W. Eisenecker: Generative Programming: Methods, Tools, and Applications. Addison-Wesley, Reading, MA, 2000

[DaRi02] F. Dandashi und D. Rine: A Method for Assessing the Reusability of Object-Oriented Code Using a Validated Set of Automated Measurements. In Proceedings of the 2002 ACM Symposium on Applied Computing (SAC), Madrid, Spanien, März 2002

[DaWa1858] Ch. Darwin und A. Wallace: On the Tendency of Species to form Varieties; and on the Perpetuation of Varieties and Species by Natural Means of Selection. Journal of the Proceedings of the Linnean Society, Vol. 3, No. 9, London, Seiten 45-62, August 1858

[DFK98] J.-M. DeBaud, O. Flege und P. Knauber: PuLSE-DSSA – A Method for the Development of Software Reference Architectures. In Proceedings of the 3rd International Workshop on Software Architecture (ISAW-3), November 1998

[Diaz01] J. Díaz-Herrera: Domain Engineering. The Handbook of Software Engineering and Knowledge Engineering (2001) http://www.ksi.edu/seke/hand.html, 2001

[DiGu01] J. Díaz-Herrera und J. Guzmán: Product Lines in the Context of Embedded Systems. In Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001), Toronto, Ontario, Kanada, Mai 2001

[DiMa00] J. Díaz-Herrera und V. Madissetti: Embedded Systems Product Lines. In Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000), Limerick, Ireland, Juni 2000

[Doug98] B.P. Douglass: Real-Time UML: Developing efficient Objects for Embedded Systems. Addison-Wesley, Reading, MA, 1998

[Dujm01] S. Dujmovic: Anwendungsentwicklung mit Komponenten-Frameworks in der Automatisierungstechnik, Dissertation am Institut für Automatisierungstechnik der Universität Stuttgart, 2001

[Etsc94] K. Etschberge: CAN – Controler Area Networl, Hanser Verlag, München 1994

[EU96] Empfehlung der Kommission vom 3.4.1996 (Amtsblatt der Europäischen Gemeinschaften Nr. L 107/4 vom 30.4.1996) iVm. Verordnung (EG) Nr. 1103/97 des Rates vom 17.6.1997

[FeVe99] X. Ferré und S. Vegas: An Evaluation of Domain Analysis Methods. In Proceedings of the 4th CAiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'99). Heidelberg, Juni 1999

[Göhn04] P. Göhner: Skript zur Vorlesung Softwaretechnik I, Institut für Automatisierungs- und Softwaretechnik, Universität Stuttgart, Wintersemester 2004/2005

[Göhn05a] P. Göhner: Skript zur Vorlesung Prozessautomatisierung I, Institut für Automatisierungs- und Softwaretechnik, Universität Stuttgart, Sommersemester 2005

[Göhn05b] P. Göhner: Skript zur Vorlesung Prozessautomatisierung II, Institut für Automatisierungs- und Softwaretechnik, Universität Stuttgart, Sommersemester 2005

[Göhn98] P. Göhner: Komponentenbasierte Entwicklung von Automatisierungs-systemen. In GMA-Kongress 1998, VDI-Berichte Nr. 1397, Seiten 513-521, 1998

[Goma01] H. Gomaa: Modeling Software Product Lines with UML. In Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001), Toronto, Ontario, Kanada, Mai 2001

[GuNä99] M. Gunzert und A. Nägele: Component-Based Development and Verification of Safety Critical Software for a Brake-by-Wire System with Synchronous Software Components. In Proceedings of the International Symposium on Parallel and Distributed Systems Engineering (PDSE99), Los Angeles, CA, 1999

[HeCo01] G. T. Heineman und W. T. Councill: Component-Based Software Engineering: Putting the Pieces Together, 1st Edition, Addison-Wesley, Reading, MA, 2001

[Holi93] R. Holibaugh: Joint Integrated Avionics Working Group (JIAWG) Object-Oriented Domain Analysis Method (JODA). Version 1.3. Technical Report. CMU/SEI-92-SR-003, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1993

[ICGN+04] J. Ivers, P. Clements, D. Garlan, R. Nord, B. Schmerl und JRO Silva: Documenting Component and Connector Views with UML 2.0. Technical Report, CMU/SEI-2004-TR-008, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2004

[ITEA04] ITEA EAST Embedded Electronic Architecture (EEA), http://www.east-eea.net/, 2004

[JoHo01] P. Jost und S. Hoffman: Software Development for Multiple OEMs using Tool Configured Middleware for CAN Communication. SAE World Congress, SAE-2001-01-0110, Detroit, USA, 2001

[Jost00] P. Jost: Identifikation von Komponenten für Domänen in der Automatisierungstechnik. In Tagungsband: Verteilte Automatisierung – Modelle und Methoden für Entwurf, Verifikation, Engineering und Instrumentierung, Magdeburg, März 2000

[Jost03] P. Jost: Identifikation von Softwarekomponenten in kleinen und mittleren Projekten. In GMA-Kongress 2003, VDI-Berichte 1756, 2003

[Karl95] E. Karlsson: Software Reuse: A Holistic Approach. Wiley, New York, 1995

[KCH+90] K. Kang, S. Cohen, J. Hess, W. Nowak und S. Peterson: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report, CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990

[KeRi90] B. Kernighan und D. Ritchie: Programmieren in C. Zweite Ausgabe ANSI C, Prentice-Hall, München, Wien, 1990

[KLM+97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier und J. Irwin: Aspect-Oriented Programming. In Proceedings of the European Conference on Object-Oriented Programming (ECOOP 1997), Finnland, Juni 1997

[KLW+96] T. Kelly, W. Lam, B. Whittle, A. Mowles und R. Rimmer: Diary of a domain analyst: a domain analysis case-study from avionics. In Proceedings of IFIP Working Groups 8.1/13.2 Conference on Domain Knowledge for Interactive System Design, Geneva, May 1996.

[KMSW00] P. Knauber, D. Muthig, K. Schmid und T. Widen: Applying Product Line Concepts in Small and Medium-Sized Companies. In IEEE Software, September/Oktober 2000, Seiten 88-95, © IEEE 2000

[Kruc95] P. Kruchten: The 4+1 View Model of Architecture. In IEEE Software, vol. 12, no. 6, Seiten 42-50, © IEEE 1995

[Krue92] C. Krueger: Software Reuse. In ACM Computing Surveys, vol. 24, no. 2, Seiten 132-183, Juni 1992

[Kurz99] R. Kurzweil: Homo S@piens: Leben im 21. Jahrhundert – Was bleibt vom Menschen?, 4. Auflage, Econ Ullstein List, München, 2001

[LaGö99a] R. Lauber und P. Göhner: Prozessautomatisierung 1, 3. Auflage, Springer-Verlag, Deutschland, 1999

[LaGö99b] R. Lauber und P. Göhner: Prozessautomatisierung 2, Springer-Verlag, Deutschland, 1999

[LeKa04] K. Lee und K. Kang: Feature Dependency Analysis for Product Line Component Design. In Proceedings of the 8th International Conference on Software Reuse (ICSR 2004), Seiten 69 – 85, Madrid, Spain, July 2004

[Luce02] V. de Lucena Junior: Flexible Web-based Management of Components for Industrial Automation, IAS-Forschungsberichte Band 4/2002, Shaker Verlag, 2002

[MBSE97] Software Engineering Institute: Model-Based Software Engineering. Webseite, www.sei.cmu.edu/mbse/, 1997

[MMYA02] H. Mili, A. Mili, S. Yacoub und E. Addy: Reuse-Based Software Engineering: Techniques, Organization, and Controls. Wiley, New York, NY, 2002

[MuBa00] D. Muthig und J Bayer: Helping Small and Medium-Sized Enterprises in Moving Towards Product-Lines. In Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000), Limerick, Irland, Juni 2000

[Neigh80] J. Neighbors: Software Constructing using components. Ph. D. Thesis, (Technical Report TR-160), Department of Information and Computer Science, University of California, Irvine, 1980

[NoWe01] J. Noble und C. Weir: Small Memory Software: Patterns for systems with limited memory. Addison-Wesley, Reading, MA, 2001

[NSK00] U. Nikula, J. Sajaniemi, H. Kälviäinen: A State-of-the-Practice Survey on Requirements Engineering in Small- and Medium-Sized Enterprises. TBRC_RR01, Telecom Business Research Center Lappeenranta, Finnland, 2000

[OMG03a] Object Management Group (OMG): Unified Modeling Language Specification Version 2.0: Super Structure, http://www.omg.org , 2003

[OMG03b] Object Management Group (OMG): UML Profile for Schedulability, Performance, and Time Specification Version 1.0, http://www.omg.org, 2003

[Parn76] D. Parnas: On the Design and Development of Program Families. In IEEE Transactions on Software Engineering, vol. SE-2, no. 1, Seiten 1-9, © IEEE 1976

[Poul94] J. Poulin: Measuring Software Reusability. In the Proceedings of the 3rd International Conference on Software Reuse (ICSR 1994), Rio de Janeiro, Brasilien, November 1994

[Sahb02] R. Sahba: Evaluation of CARTRONIC as Domain Model for Software Component Development. Master Thesis 1836, Institut für Automatisierungs- und Softwaretechnik, Universität Stuttgart, 2002

[Schm97] H. A. Schmid: Systematic Framework Design by Generalization. In Communications of the ACM, vol. 40, no. 10, Seiten 48-51, Oktober 1997

[SCK+96] M. Simos, D. Creps, C. Levine und D. Allemang: Organization Domain Modeling (ODM) Guidebook, Version 2.0. Informal Technical Report for STARS, STARS-VC-A025/001/00, Juni 1996

[SEI04] Software Engineering Institute: Product Line Approach to Software Development. http://www.sei.cmu.edu/plp/plp_init.html, 2004

[SGW94] B. Selic, G. Gullekson und P. Ward: Real-Time Object-Oriented Modeling. Wiley, New York, NY, 1994

[ShGa96] M. Shaw und D. Garlan: Software Architectures: Perspectives on an Emerging Discipline. Prentice-Hall, Upper Saddle River, NJ, 1996

[Simo94] 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

[Somm95] I. Sommerville: Software Engineering. 5th Edition, Addison-Wesley, Reading, MA, 1995

[SoSo99] J. Sodhi und P. Sodhi: Software Reuse: Domain Analysis and Design Process. McGraw-Hill, New York, 1999

[SPC93] Software Productivity Consortium: Reuse-Driven Software Processes Guidebook. Version 02.00.03, Technical Report, SPC-92019-CMC, Software Productivity Consortium, Herndon, VA, 1993

[Stro00] B. Stroustrup: Die C++-Programmiersprache, 4. aktualisierte und überarbeitete Auflage. Addison-Wesley, Reading, MA, 2000

[Szyp98] C. Szyperski: Component Software: Beyond Object-Oriented Programming. Addison-Wesley, Reading, MA, 1998

[Trac94] W. Tracz: LILEANNA Language Reference Manual. Technical Report ADAGELOR9407, Loral Federal Systems - Owego, Juli 1994

[Trac95] W. Tracz: Domain-Specific Software Architecture Pedagogical Example. In ACM SIGSOFT Software Engineering Notes, vol. 20, no. 4, Juli 1995, Seiten 49-62, 1995

[TrCo92] W. Tracz und L. Coglianese: DSSA Engineering Process Guidelines. Technical Report. ADAGE-IBM-9202, IBM Federal Systems Company, December 1992

[USC04] University of Southern California: COCOMO, http://sunset.usc.edu/cse/pub/research/COCOMOII/cocomo_main.html, 2004

[VAM+98] A. Vici, N. Argentieri, A. Mansour, M. d'Alessandro und J. Favaro: FODAcom: An Experience with Domain Analysis in the Italian Telecom Industry. In Proceedings of the 5th International Conference On Software Reuse (ICSR 1998), Victoria, Kanada, Juni 1998

[VSG+00] T. Vernazza, G. Succi, P. Galfione, A. Valerio und P. Predonzani: Moving Toward Software Product Lines in a Small Software Firm: A Case Study. In Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000), Limerick, Irland, Juni 2000

[Wall1855] A. Wallace: On the Law Which Has Regulated the Introduction of New Species. Annals and Magazine of Natural History, Vol. 16, 2nd Series, Seiten 184-196, September 1855

[WeLa99] D. Weiss und C. Lai: Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley, Reading, MA, 1999

[WYF03] H. Washizaki, H. Yamamoto und Y. Fukazawa: A Metrics Suite for Measuring Reusability of Software Components. In Proceedings of Ninth International Software Metrics Symposium (METRICS ‘03), Sydney, Australien, September 2003

[Zalm96] N. Zalman: Making The Method Fit: An Industrial experience in Adopting Feature-Oriented Domain Analysis (FODA). In Proceedings of the Fourth International Conference on Software Reuse, IEEE Computer Society Press, Los Alamitos, CA, 1996, Seiten 233-235, © IEEE 1996

[ZHJ03] T. Ziadi, L. Hélouët und JM Jézéquel: Towards a UML Profile for Software Product Lines. In Prodeedings of the Fifth International Workshop on Product Family Engineering (PFE-5), Seiten 129-139, 2003

Lebenslauf

Persönliche Daten

29.03.1972	geboren in Pforzheim

Schulbildung

1978 – 1982	Grundschule
1982 – 1988	Realschule, Abschluss: Mittlere Reife
1988 – 1991	Technisches Gymnasium, Abschluss: Fachgebundene Hochschulreife

Zivildienst

1991– 1992	Wohnheim für Behinderte der Lebenshilfe e.V. Pforzheim

Studium

1992 – 1998	Studium der Elektrotechnik an der Universität Stuttgart Studienmodell: Regelungs- und Automatisierungstechnik
01.09.1998	Abschluss als Diplom-Ingenieur

Berufstätigkeit

1998 – 2003	Wissenschaftlicher Mitarbeiter am Institut für Automatisierungs- und Softwaretechnik der Universität Stuttgart
seit 2004	Mitarbeiter der Robert Bosch GmbH, Geschäftsbereich Energy and Body Systems, Stuttgart