Kluge, René
Händelstraße 14
99610 Sömmerda
Tel.: 03634/608828
E-Mail: R_Ke@gmx.net
Studiengang: Wirtschaftsinformatik
Matrikel-Nr.: 26083

Integration der Aufwandsschätzung in die Requirements Engineering Phase der Systemfamilienentwicklung

Diplomarbeit
zur Erlangung des akademischen Grades
"Diplom-Wirtschaftsinformatiker"
an der Fakultät für Informatik und Automatisierung
der Technischen Universität Ilmenau

Fachbereich Prozessinformatik
Betreuender Hochschullehrer: Prof. Dr.-Ing. habil. Ilka Philippow
Weiterer Betreuer: Dipl.-Inf. Detlef Streitferdt
Starttermin: 03. September 2002
Abgabetermin: 03. März 2003

Kurzfassung

Das Ziel der vorliegenden Arbeit ist es, Möglichkeiten für eine Integration von Wirtschaftlichkeitsbetrachtungen in den Prozess der Anwendungsentwicklung im Umfeld von Systemfamilien aufzuzeigen.

Den Ausgangspunkt bilden die in einer Systemfamilie dargelegten Merkmale, welche letztendlich die Bestandteile und Funktionalitäten möglicher zu entwickelnder Anwendungen widerspiegeln. Diesen gegenüber treten die in der Anforderungsanalyse erfassten und aufbereiteten Anforderungen seitens eines oder mehrerer Kunden. Diesbezüglich werden verschiedene Szenarien unterschieden, in denen Anforderungen nicht oder lediglich unzureichend durch bestehende Elemente der Systemfamilie abgedeckt und dementsprechende Entwicklungen notwendig werden.

Im Sinne einer ökonomisch begründbaren Entscheidung für oder gegen eines dieser Szenarien wird das Anwendungsgebiet der Aufwandsschätzung vorgestellt. Auf diesen Ausführungen aufbauend, wird das Aufwandsschätzverfahren COCOMOII in die Arbeitsabläufe der Anwendungsentwicklung im Kontext von Systemfamilien in Form von Vorgehensmodellen integriert. Dabei fließen Besonderheiten, welche vorrangig in der Existenz einer Systemfamilie begründet liegen, durch eine Anpassung des Aufwandsschätzverfahrens, in die erarbeiteten Vorgänge ein.

Weiterhin erfolgt, im Hinblick auf einer prototypischen und praxistauglichen Umsetzung, eine Analyse bestehender Datenformate, welche sowohl Merkmale einer Systemfamilie als auch bestehende Anforderungen erfassen können. Aufgrund seines spezifischen Einsatzfeldes und Ursprungs in der eXtensible Markup Language, wird insbesondere auf das Datenformat des Family Oriented Requirements Engineering (FORE) Projekts und bestehenden Möglichkeiten der weiteren Verarbeitung eingegangen. Der abschließend vorgestellte Prototyp umfasst daher sowohl Aspekte einer, dem Umfeld von Systemfamilien angepassten Aufwandsschätzung als auch einer weiteren Verarbeitung deren Ergebnisse mit Hilfe des FORE Datenformats.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Aktuelle Softwareentwicklungen sind geprägt durch steigende Ansprüche seitens der Kunden. Deren Verlangen nach schnellen, qualitativ hochwertigen Produkten zwingt die Hersteller zu neuen, innovativen und effizienten Ansätzen der Softwareerstellung. Dies vollzieht sich zusätzlich vor dem Hintergrund eines zunehmenden Wettbewerbs, woraus sich weitere Anforderungen an die moderne Softwareentwicklung ableiten.

Eine aktuelle und vielversprechende Entwicklung ist der Ansatz der Systemfamilien, welcher in der vorliegenden Arbeit vorgestellt wird. Dies geschieht unter dem Fokus des dargelegten wirtschaftlichen Umfelds. Dabei werden Anforderungen des Kunden ebenso von Interesse sein, wie deren Umsetzung im Prozess der Softwareentwicklung. Im Anschluss wird auf diese Zusammenhänge näher eingegangen und die thematischen Beziehungen vorgestellt.

1.1 Motivation

Im Rahmen der Softwareentwicklung findet der Ansatz der Systemfamilien zunehmende Beachtung. Im Gegensatz zur klassischen Softwareentwicklung, in der einzelne Anwendungen stets neu erstellt werden, sind nun aus einer Systemarchitektur ableitbare Anwendungen von Interesse. Vor allem ökonomische Aspekte der Wiederverwendung bereits entwickelter Software sowie zeit- und kostenorientierte Verbesserungen des Erstellungsprozesses werden dabei berücksichtigt. In der

vorliegenden Arbeit wird der Ansatz der Systemfamilien vorgestellt sowie Grundlagen und notwendige Arbeitsschritte zur Bildung einer derartigen Systemfamilie aufgezeigt.

Vor allem die für eine Softwareentwicklung typische Phase des Requirements Engineering beziehungsweise der Anforderungsanalyse[1] dient im Umfeld von Systemfamilien zur Festlegung der Anforderungen seitens eines Kunden sowie der Darlegung und Aufbereitung dieser für nachfolgende Arbeitsschritte. Eine sinnvolle Notation spielt dabei eine wichtige Rolle. Der Aufbau und die Struktur der Notation muss die Merkmale, durch welche die zu entwickelnden Softwaresysteme beschrieben werden, möglichst genau abbilden, so dass Anforderungen sinnvoll erfasst und widerspruchsfrei geklärt werden können.

In der Verwendung einer Systemfamilie und in der Festlegung der Anforderungen an ein zu entwickelndes System stellt sich meist die Frage nach der Entwicklungsdauer und den Entwicklungskosten. Mit Hilfe aktueller Aufwandsschätzverfahren kann die Wirtschaftlichkeit einer geplanten Softwareentwicklung im voraus überprüft werden. Im Hinblick auf das derzeitige, eingangs des Kapitels aufgeführte Umfeld der Softwareentwicklung werden dadurch ökonomische Entwicklungen planbar und Risiken vorhersehbar. Aufbauend auf einer Notation, welche in der Anforderungsanalyse begründet liegt, können somit Informationen für eine Aufwandsschätzung gewonnen und genutzt werden.

1.2 Zielstellung

Die drei Eckpunkte moderner Softwareentwicklung sind Zeit, Kosten und Qualität, wobei hinsichtlich wettbewerbsorientierter Entwicklungen der Erfolgsfaktor Qualität immer mehr in den Mittelpunkt rückt. Aus Kundensicht stellt hohe Qualität letztendlich die bestmöglichste Erfüllung seiner Anforderungen an das Endprodukt dar. Innerhalb der Anforderungsanalyse einer Entwicklung gilt es somit diese Anforderungen vollständig und widerspruchsfrei zu erfassen. Im Rahmen einer Systemfamilienentwicklung spielt vor allem eine Analyse der Anforderungen im Hinblick auf bereits verfügbare Systembestandteile eine Rolle. Jedoch

[1] im Folgenden wird der Begriff der „Anforderungsanalyse" verwendet

stellt sich in diesem Zusammenhang die Frage: Wie verfährt man mit Anforderungen, die nicht durch vorhandene Bestandteile abgedeckt werden können und wie integriert man diese sinnvoll? Auch rücken bezüglich der Anforderungsanalyse ökonomische Aspekte, also die eingangs erwähnten Faktoren Zeit und Kosten, für den Kunden in den Vordergrund. Somit liegt das Ziel in einer sinnvollen Verbindung der Ergebnisse der Anforderungsanalyse und der Aufwandsschätzung innerhalb einer praxisrelevanten Notation. Aspekte, welche dabei berücksichtigt werden müssen, wären:

- Innerhalb welcher Abläufe einer Entwicklung im Rahmen einer Systemfamilie kann eine Aufwandsschätzung sinnvoll integriert werden?
- Welche Verfahren der Aufwandsschätzung bieten sich für eine sinnvolle Verknüpfung von bereits bestehenden und aufzunehmenden Anforderungen an?
- Welche Notation soll verwendet werden? Welche Ansätze bietet die eXtensible MarkUp Language?

Das Ziel der vorliegenden Arbeit ist somit zunächst eine Analyse der Arbeitsschritte, welche im Zusammenhang mit der Entwicklung einer Systemfamilie bestehen. Des Weiteren müssen aktuelle Entwicklungen der Aufwandsschätzung untersucht und hinsichtlich praxisrelevanter Einsatzmöglichkeiten geprüft werden. Ein nächster Schritt beinhaltet das notwendige Vorgehen einer sinnvollen Verknüpfung von Aufwandsschätzung und Systemfamilien. Dazu müssen bestehende Notationen betrachtet werden, wobei insbesondere aktuelle Entwicklungen und bestehende Vorteile der eXtensible Markup Language in den Vordergrund treten.

2 Der Ansatz der Systemfamilien

Die bisherige Entwicklung von Software orientiert sich stark an einzelnen Softwareprojekten. Den Ausgangspunkt bilden die von einem oder mehreren Kunden geforderten Funktionalitäten, welche in möglichst kurzer Zeit und zu möglichst geringen Kosten in ein fertiges Produkt umgesetzt werden sollen. Oft treten nun innerhalb verschiedener Projekte ähnliche oder gar gleiche Anforderungen in ähnlichen Kontexten auf. Damit stellt sich die Frage, ob eine ständig neue Entwicklung bereits erstellter Lösungen überhaupt notwendig oder sinnvoll ist. In diesem Zusammenhang werden zunehmend Ansätze der Wiederverwendung bestehender Software verfolgt.[1]

Blickt man auf den Bereich industrieller Fertigung, werden Aspekte einer möglichst effizienten Erfüllung von Kundenforderungen in Form sogenannter *Produktlinien* deutlich. Diese fassen aufgrund sich stets wiederholender Kundenwünsche Gemeinsamkeiten der Produkte zusammen. Die dadurch entstehenden Vorteile wiegen derart stark, dass es zunehmend zu Bemühungen einer Übertragung des Ansatzes auf den Bereich der Softwareentwicklung kommt, wobei hier der Begriff der *Systemfamilien* zunehmend Verwendung findet.[2]

Im folgenden Kapitel werden diese Systemfamilien näher untersucht. Dabei liegt der Fokus auf den Gründen für die Bildung sowie auf Ansätzen und Ideen der Umsetzung. Eine nähere Betrachtung erfolgt anhand der Anforderungsanalyse der Softwareentwicklung im Anwendungsfeld der Systemfamilien.

[1] vgl. [CMU/SEI99] S.1,3; [CMU/SEI97] S.19; [Whitey96] S.1
[2] vgl. [Cohen02]

2.1 Grundlagen

Der Ansatz der Systemfamilien basiert auf zahlreichen Überlegungen, welche den Aspekt der Wiederverwendung von Software unterschiedlich aufgreifen. Daher erfolgt zunächst eine Betrachtung wichtiger Grundlagen und Ideen, wobei insbesondere die differenzierte Herangehensweise des Ansatzes der Systemfamilien hervorgehoben werden soll.

2.1.1 Wiederverwendung: Von Einzellösungen zu Systemfamilien

Im Rahmen einer Softwareentwicklung sind die meisten Unternehmen aufgrund der im Laufe der Zeit gewonnenen Erfahrungen und dem erlangten Wissen auf einen bestimmten Kundenkreis spezialisiert. Daher kommt es häufig zu Überschneidungen hinsichtlich Struktur und Einsatzzweck der Endprodukte. Die spezifischen Anwendungsfälle stimmen weitestgehend überein. Die Frage, die sich dabei stellt, ist: „Wie können zentrale Bestandteile der Software, welche stets einen Anteil an den Endprodukten haben, sinnvoll wieder verwendet werden?"

Die Potenziale, welche im Zusammenhang mit der Wiederverwendung von Software stehen, sind vielversprechend:[3]

- Kürzere „time-to-market"[4] Prozesse sorgen für Wettbewerbsvorteile hinsichtlich der zeitlichen Präsenz des Marktauftritts.
- Die dabei resultierende zunehmende Teilnahme des Unternehmens und seiner Produkte am öffentlichen Marktgeschehen sorgt für weitere Vorteile im Wettbewerb.
- Ein Anstieg der Produktivität erfolgt, da der Aufwand in der Erstellung der Endprodukte von kompletter Erstellung auf Zusammenstellung und Anpassung der Software sinkt.

[3] vgl. [Whitey96] S.1ff; [PULSE02] S.2f; [Knauber01]; [Cohen02]

[4] „[...] die Zeit, die von der Initiierung einer Produktidee bis hin zur Markteinführung verstreicht." vgl. [Buchholz96]

- Es kommt zum Anstieg der Qualität der Produkte, da Fehler innerhalb der Erstellung beziehungsweise Zusammenstellung durch wieder verwendbare und bewährte Bestandteile zurückgehen.
- Das Resultat wäre wiederum eine zunehmende Kundenzufriedenheit, was die Grundlage einer längerfristigen Kundenbindung darstellt.

Zur Erreichung derartiger Ziele wurden in der Vergangenheit zunächst Ansätze bezüglich der Wiederverwendung des Programmquellcodes verfolgt, welche ihren Höhepunkt im Rahmen der objektorientierten Softwareentwicklung gefunden haben. Durch die innerhalb der Objektorientierung angewandten Vererbungstechniken können nachträglich Funktionalitäten in bestehende Software integriert werden. Zwar besteht durch diesen Ansatz das Potenzial zu einer Wiederverwendung von Software, jedoch werden in diesem Zusammenhang relativ kleine Ausschnitte des zu entwickelnden Gesamtsystems betrachtet. In komplexen Systemen entsteht somit ein nicht unerheblicher Aufwand in der Handhabung einzelner Objekte.[5]

Weitere Entwicklungen suchten nach Möglichkeiten einzelne Bestandteile des Programmquellcodes zusammenzufassen und eine strukturierte Umgebung zur besseren Verwendung zu schaffen. Daher sollen im Anschluss die Komponentenbasierte Softwareentwicklung und der Ansatz der Frameworkentwicklung vorgestellt werden.[6]

Die Komponentenbasierte Softwareentwicklung:
Innerhalb dieses Ansatzes der Softwareentwicklung entstehen einzelne, für einen jeweiligen Anwendungsfall spezifische Komponenten. Diese werden durch eine möglichst sinnvolle Gruppierung von bereits entwickelten Bestandteilen des Programmquellcodes gebildet. Die Idee dabei ist, dass komplette Anwendungen durch Komposition zusammengesetzt werden können. Zur Abstimmung und korrekten Funktionsweise der Komponenten müssen anschließend zu erstellende Programmzwischenstücke[7] gebildet werden. Der Nachteil dieses Ansatzes besteht jedoch darin, dass eine Komponente herausgelöst aus ihrem Anwendungskontext nur mit

[5] vgl. [Czarnecki01]
[6] vgl. [Northrop03]; [Böllert02]
[7] sog. „Clue Code"

einem gewissen Aufwand in einen anderen Kontext übertragen werden kann.[8]

Der Framework-Ansatz:
Frameworks umfassen eine Architektur und zugehörige Schnittstellen zur Integration von Komponenten. Durch die vorgegebene Struktur wird zudem der Kontroll- beziehungsweise Steuerfluss der resultierenden Anwendung festgelegt. Frameworks bilden somit einen generischen und universellen Rahmen, welcher durch anwendungsspezifische Komponenten spezialisiert wird. Das Problem in der Verwendung besteht jedoch darin, dass bei der Erstellung eines Frameworks ein Kompromiss eingegangen werden muss zwischen vorweg festgelegten und den parametrisierbaren Funktionalitäten. Die Gefahr einer Fehleinschätzung der durch das Framework zu erfassenden Anforderungen besteht dadurch jederzeit. Auch gestaltetet sich das Zusammenspiel mehrerer Frameworks problematisch.

Die Existenz mehrerer Komponenten oder eines Frameworks garantiert somit nicht die eingangs aufgezeigten Erfolge, welche mit der Wiederverwendung angestrebt werden. Insbesondere die dargelegten Probleme mit diesen Ansätzen sprechen für eine umfassende, systematische Betrachtung der Wiederverwendung. Die Konsequenz ist der Ansatz der Systemfamilien. Wiederverwendung wird hier systematisch und strategisch geplant und umgesetzt. Nicht mehr einzelne Frameworks oder Komponenten sondern das Zusammenspiel aller Bestandteile der Endprodukte wird betrachtet. Dies schließt insbesondere Überlegungen ein, welche über die Softwareentwicklung hinausgehen[9] und vorrangig ökonomische geprägte Sachverhalte zum Inhalt haben.[10]

Zunächst werden im Anschluss die im Zusammenhang mit einer Systemfamilie stehenden Begriffe geklärt, woraufhin eine Vorstellung des Aufbaus und der Anwendung einer Systemfamilie erfolgt.

[8] sog. „architectural mismatch": Die Architektur der Komponente entspricht nicht der Architektur des Systems, in welcher sie eingesetzt werden soll. vgl. dazu [CMU/SEI97] S.16ff.
[9] vgl. [CMU/SEI97] S.10,19
[10] siehe Abschnitt 2.3 - Die Anforderungsanalyse im Kontext von Systemfamilien

2.1.2 Begriffsklärung

Bezüglich des Ansatzes der Systemfamilien bestehen zahlreiche unterschiedliche Ausführungen innerhalb der Literatur. Begriffe werden verschieden abgegrenzt und von Autoren teilweise kontrovers verwendet. Daher ist im Folgenden zu beachten, dass die hier dargelegten Definitionen in weiteren Veröffentlichungen der Literatur möglicherweise differenziert gebraucht werden.[11]

Produktlinie
Die Voraussetzung des Ansatzes der Systemfamilien bildet der Begriff der „Produktlinie". Aufgrund des ursprünglich industriellen Kontextes leitet sich eine allgemeine Definition der Produktlinie ab als:[12]

„[...] eine Gruppe von Produkten als gemeinsame, wirtschaftlich erfassbare Zusammenstellung von Eigenschaften, bezogen auf spezielle Bedürfnisse des gewählten Marktes oder Auftrags."

Den Ausgangspunkt stellen demnach Gemeinsamkeiten von Produkten dar, welche vorrangig einem Kunden gegenüber deutlich werden, wobei hauptsächlich ökonomische Belange im Vordergrund stehen. Technische Zusammenhänge werden in Verwendung des Begriffs vernachlässigt.

Systemfamilie
Inzwischen bestehen zahlreiche Ausführungen, welche den Ansatz der Produktlinien auf den Bereich der Softwareentwicklungen übertragen. Dabei fließen ökonomische und technische Ansichten unterschiedlich stark ein. Das Ergebnis ist der Begriff der *Systemfamilie*, welcher die Zuordnung von Produkten aufgrund gemeinsamer technischer Merkmale beinhaltet. Eine Systemfamilie wird im Folgenden verstanden als:[13]

„[...] eine Gruppe von Systemen, welche auf Basis einer Zusammenstellung von gemeinsam verwendbaren Softwarebestandteilen, gebildet wird."

Die Betrachtung von Produktlinien konzentriert sich demzufolge mehr auf die Bedürfnisse und Belange eines bestimmten Marktes sowie des unternehmerischen

[11] vgl. [Czarnecki00] S. 41ff.
[12] vgl. [CMU/SEI99]S.1; S.33ff.
[13] vgl. [Czarnecki01]

Umfelds. Gruppierungen entstehen hinsichtlich der Möglichkeiten zur Vermarktung von Produkten. Dabei ist zu beachten, dass technische Gemeinsamkeiten nicht gegeben sein müssen.

Der Begriff der Systemfamilien hingegen basiert auf dieser gemeinsamen, technischen Grundlage zur Wiederverwendung der Produkte und wird hier als Weiterfassung des Begriffs der Produktlinie verstanden. Unter diesem Blickwinkel ist es möglich, eine Produktlinie als Systemfamilie zu entwerfen, um Vorteile durch Wiederverwendung zu erhalten. Eine Systemfamilie kann dabei auch mehrere Produktlinien umfassen oder eine von mehreren Systemfamilien innerhalb einer Produktlinie darstellen.[14]

Im weiteren Verlauf der vorliegenden Arbeit wird aufgrund der vorrangigen Betrachtung der Möglichkeiten zur Erfassung wieder verwendbarer Bestandteile und aufgrund des Fokus' auf der Softwareentwicklung der Begriff der Systemfamilie verwendet.

Asset

Die Verwendung des Begriffs ist eng mit dem Begriff der Systemfamilie verknüpft. Ein Asset ist:[15]

„[...]jedes greifbare Resultat, das bei der Entwicklung einer Systemfamilie entsteht. Assets können Codestücke, Dokumentationen, Modelle, Anforderungsdokumente, Organisationsanweisungen sein."

Diese Definition betont zusätzlich die differenzierte Betrachtung der Wiederverwendung innerhalb des Ansatzes der Systemfamilien. Wiederverwendung bezieht sich hier somit auf alle Bereiche der Softwareentwicklung.

Domäne

Mit dem Ansatz der Systemfamilie einhergehend ist der Begriff der „Domäne". Dieser auch innerhalb der Literatur recht kontrovers verwendete Begriff soll im Folgenden verstanden werden als:[16]

[14] vgl.[Czarnecki01]; [CMU/SEI99] S.5
[15] vgl. [Whitey96] S.1; [visek]
[16] vgl. [Czarnecki00] S.41f.

„[...]das Anwendungsfeld von Software, welches das gesamte Wissen zur Entwicklung der Software beinhaltet und somit das Potenzial zur Wiederverwendung bildet."

In diesem Zusammenhang wird deutlich, dass die Entwicklung einer Systemfamilie im Kontext einer bestimmten Domäne stattfindet. Wissen und Erfahrungen eines Unternehmens innerhalb spezifischer Bereiche der Softwareentwicklung fließen dabei mit ein.[17]

Merkmale

Im Zusammenhang mit dem Begriff der Systemfamilien ist die Beschreibung dieser durch Merkmale zu nennen.[18]

„Merkmale beinhalten eine kurze, prägnante Beschreibung der Eigenschaften eines Systems."

Grundlage der Merkmale bilden die Anforderungen an zu erstellende Anwendungen. Die Sicht auf die Systemfamilie seitens der Nutzer ist dabei ausschlaggebend. Bezüglich dieser ist zu bedenken, dass Anforderungen seitens potenzieller Kunden in einem gewissen Maße heterogen sind. Somit bestehen in der Realisierung einzelner Anwendungen stets feste, wiederkehrende aber auch neue, spezifische Anforderungen. Innerhalb der Festlegung der Merkmale von Systemfamilien findet daher eine Unterscheidung bezüglich „Gemeinsamkeiten" und „Variabilitäten" statt.[19]

- *Gemeinsamkeiten* (commonalities) definieren die Grundlage für die Wiederverwendung. Der Fokus liegt auf der Kernfunktionalität, also den Merkmalen, welche alle Mitglieder der Systemfamilie aufweisen.
- *Variabilitäten* (variabilities) kennzeichnen die Unterschiede in den Merkmalen der einzelnen Systemmitglieder. Sie beschreiben das zusätzliche Maß an Funktionalität, welches innerhalb einer Domäne notwendig ist oder notwendig werden kann.[20]

[17] vgl. [Czarnecki01]
[18] vgl. [Czarnecki00] S.83f; [Böllert02]
[19] vgl. [Czarnecki00] S.92f.
[20] vgl. [Farcet01]

2.2 Die Entwicklung einer Systemfamilie

Ist die Entscheidung durch ein Unternehmen einmal getroffen, zukünftige Anwendungen auf Basis einer Systemfamilie zu entwickeln, sind zahlreiche, wohl geplante Arbeitsschritte zu deren Umsetzung notwendig. An den folgenden Vorüberlegungen anschließend sollen die zum Aufbau und zur Umsetzung notwendigen Schritte aufgezeigt werden. Die Betrachtung erfolgt dabei vorrangig aus allgemeiner und analytischer Sicht, wobei am Ende eine genauere Beschreibung, anhand der für eine Softwareentwicklung typischen Phase der Anforderungsanalyse vorgenommen wird.

2.2.1 Vorüberlegungen

Wie zurückliegende Ausführungen teilweise schon erkennen lassen, ist der Aufbau einer Systemfamilie mit Aufwand verbunden, welcher über die üblichen Ansätze der Wiederverwendung hinausgeht.[21] Statt der bloßen Nutzung wieder verwendbarer Komponenten oder Frameworks, die zum Teil als Nebenprodukte der klassischen Softwareentwicklung anfallen, wird nun eine ganzheitliche und strategische Betrachtung der Wiederverwendung verfolgt. Dies bedingt jedoch Investitionen, deren Amortisationszeitpunkt[22] ex ante unbekannt ist.[23] Dementsprechend sind zahlreiche Überlegungen im Vorfeld notwendig, wobei besonders die Inanspruchnahme monetärer und personeller Ressourcen zu berücksichtigen ist. Im Hinblick auf die dabei entstehenden wirtschaftlichen Risiken ist die Wahl zwischen einer vorsichtigen und sukzessiven sowie einer vollständig einheitlichen und möglichst zügigen Umsetzung der Systemfamilie zu treffen. Während erstere Strategie die erwähnten wirtschaftlichen Risiken kontrolliert und einschränkt, ist bei zweiterer eine frühe Ausnutzung der Vorteile möglich, welche im Zusammenhang mit Systemfamilien bestehen.[24]

[21] vgl. [PULSE02] S.2; [Cohen02] S. 21ff.
[22] Zeitpunkt ab dem die Gewinne den bis dahin angefallenen Kosten entsprechen vgl. [Gabler00] S.108
[23] vgl. [Whitey96] S.2
[24] vgl. [CMU/SEI97] S.6; [Knauber01]

Innerhalb der Betrachtung einer Systemfamilie erfolgt im Allgemeinen ein zweiteiliges Vorgehen. Zum einen liegt der Schwerpunkt auf der Bildung der Systemfamilie selbst, das sogenannte ,,*domain engineering*". Zum anderen betrifft diese Unterteilung die Konzentration auf der Ableitung konkreter Anwendungen, das sogenannte *„application engineering"*. Beide sollen nun vorgestellt werden.

2.2.2 Domain Engineering - Entwicklung für Wiederverwendung

Technisch betrachtet besteht eine Systemfamilie aus einer Architektur und zahlreichen Komponenten, welche zielorientiert in diese eingebettet werden. Das Domain Engineering umfasst alle Schritte zur Bildung einer innerhalb der bestimmten Domäne allgemeingültigen Architektur und der zugehörigen Assets.[25]

Aufgrund einer Vielzahl von Methoden und Vorgehensmodellen, welche in diesem Zusammenhang bestehen und von zahlreichen Autoren unterschiedlich aufgegriffen und verfolgt werden,[26] kann die im Folgenden vorgestellte Herangehensweise nicht als einflussfreie oder allgemeine Betrachtung gewertet werden. Jedoch sollen, soweit möglich, grundlegende Gedanken aufgezeigt werden. Die zentralen Bestandteile des Domain- und Application Engineerings sind in Abbildung 2.1 dargestellt und werden im Anschluss näher erläutert.[27]

[25] vgl. zu folgenden Ausführungen [Czarnecki00] Chapter3 Domain Engineering; [Czarnecki01]
[26] vgl. [CMU/SEI97] S.7,18
[27] Die Abbildung des Abschnitts sind angelehnt an [Czarnecki00]

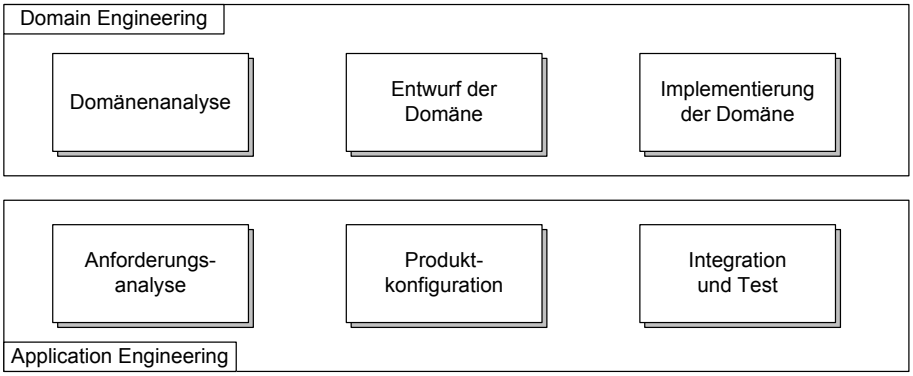


Abbildung 2.1: Überblick: Domain- and Application Engineering

Scoping

Der zwar in Abbildung 2.1 vernachlässigte Bestandteil, jedoch Voraussetzung für den Aufbau einer Systemfamilie, ist die Auswahl der Domäne, das sogenannte *Scoping*.[28] Die Domäne ergibt sich aus dem Anwendungsfeld, in welchem ein Unternehmen bisher tätig war oder aus wettbewerblich geprägten Überlegungen bezüglich der Spezialisierung oder Ausweitung der Geschäftsbereiche. Das Scoping dient vorrangig dem Erkennen relevanter Grenzen, innerhalb derer ein gewisser Aufwand zur Bildung der Systemfamilie gerechtfertigt scheint. Folgende Ausführungen setzen diese Abgrenzung voraus und schließen direkt daran an.

Domänenanalyse (domain analysis)

Innerhalb dieses Schrittes gilt es, sämtliche Systeme beziehungsweise Anwendungen innerhalb einer Domäne zu identifizieren und derart aufzubereiten, dass ein Verständniss der Domäne und ihrer Systeme gegeben ist.

Neben einer Analyse bisher entwickelter Anwendungen werden dazu Experten eines Anwendungsbereichs herangezogen sowie ökonomisch geprägte Überlegungen getroffen.[29] Dabei tritt vorrangig das Feststellen gemeinsamer und variabler Merkmale der Systemfamilie in den Mittelpunkt der Betrachtung. Das Resultat der Domänenanalyse sind Modelle der Domäne und geplanter Anwendungen sowie

[28] vgl. [Czarnecki00] S.42f.
[29] vgl. [CMU/SEI97] S.10,16

vollständig erfasste Gemeinsamkeiten und Variabilitäten. Zur Modellierung der Merkmale existieren zahlreiche Ansätze, wobei sich dahingehende Entwicklungen auf objektorientierte Technologien konzentrieren.[30] Der Teilschritt der Domänenanalyse ist in Abbildung 2.2 skizziert.

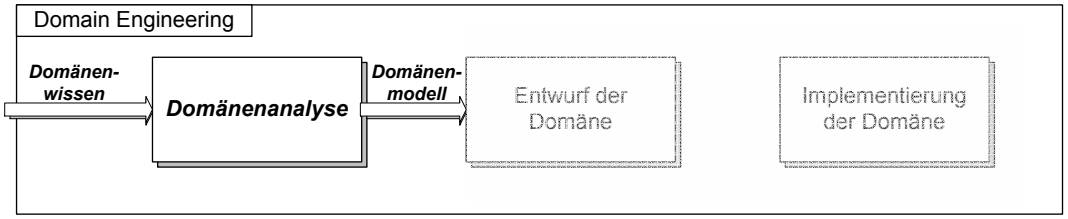


Abbildung 2.2: Die Domänenanlyse

Entwurf der Domäne (domain design)

Das Ziel des Domänenentwurfs ist die Schaffung einer gemeinsamen *Systemarchitektur*, welche die Grundlage der Struktur aller zu erstellenden Anwendungen darstellt. Sie kann als wieder verwendbares Framework beschrieben werden, in welchem das Zusammenspiel aller Assets einer Systemfamilie definiert ist.

Zudem wird in diesem Arbeitsschritt ein *Produktionsplan* erstellt, welcher darlegt, wie eine Anwendung aus der Systemfamilie abgeleitet werden kann. Abbildung 2.3 veranschaulicht den Domänenentwurf.

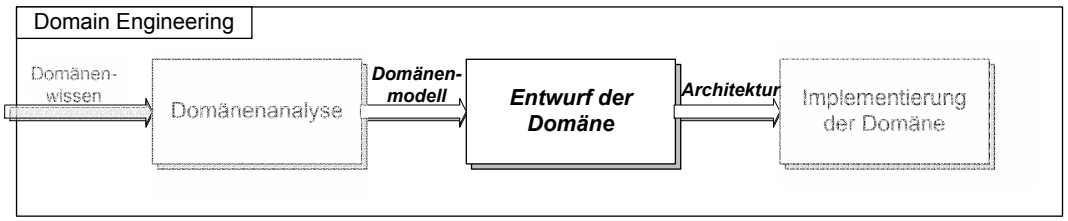


Abbildung 2.3: Der Entwurf einer Domäne

[30] siehe Abschnitt 2.3.1 - Die Erfassung und Modellierung von Merkmalen

Implementierung der Domäne (domain implementation)
Auf Basis der Modelle der Domänenanalyse und der im Domänenentwurf dargelegten Architektur kann in diesem Arbeitsschritt eine konkrete Implementierung erfolgen. Dazu werden Assets in die Architektur eingebunden. Insofern sind Überlegungen notwendig, ob eine vollständig neue Entwicklung notwendig ist oder Bestandteile bereits entwickelter Anwendungen in Form von Komponenten übernommen werden können. Letztendlich sind alle geplanten, wieder verwendbaren Assets erzeugt und Anwendungen aus diesen ableitbar. Abbildung 2.4 zeigt die vorgenommen Arbeitsschritte innerhalb des Domain Engineerings.

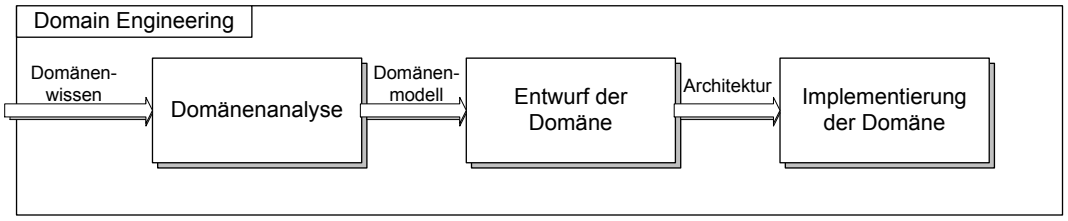


Abbildung 2.4: Implementierung der Domäne

2.2.3 Application Engineering - Entwicklung mit Wiederverwendung

Das Application Engineering im Zusammenhang mit Systemfamilien setzt die Existenz dieser voraus. Anwendungen werden auf Basis der innerhalb des Domain Engineerings erstellten Architektur und den darin enthaltenen Assets gebildet. Wie sich zeigen wird, ergibt sich ein Zusammenspiel des Application- und Domain Engineerings, was wiederum Einfluss auf die Ableitung konkreter Anwendungen hat. Insbesondere soll hier schon einmal auf die zentrale Bedeutung der Anforderungsanalyse hingewiesen werden.[31]

Die Anforderungsanalyse (requirement analysis)
Die Anforderungsanalyse ist ein zentraler Bestandteil jeglicher Softwareentwicklung. Innerhalb dieser gilt es, die von einem Kunden oder einem Markt vorge-

[31] vgl. zu folgenden Ausführungen [Czarnecki01]

gebenen Anforderungen an das zu entwickelnde System entgegenzunehmen und für nachfolgende Arbeitsschritte aufzubereiten. Im Rahmen von Systemfamilien ist dieser Bestandteil von Bedeutung, da auftretende Anforderungen mit den in der Systemfamilie enthaltenen Merkmalen abgeglichen werden müssen. In diesem Zusammenhang besteht eine Wechselbeziehung zwischen Anforderungs- und Domänenanalyse, welche insbesondere dann entsteht, wenn nicht alle Anforderungen durch die Merkmale der Systemfamilie abgedeckt werden. Zunächst erfolgt anhand Abbildung 2.5 eine Einordnung der Anforderungsanalyse:

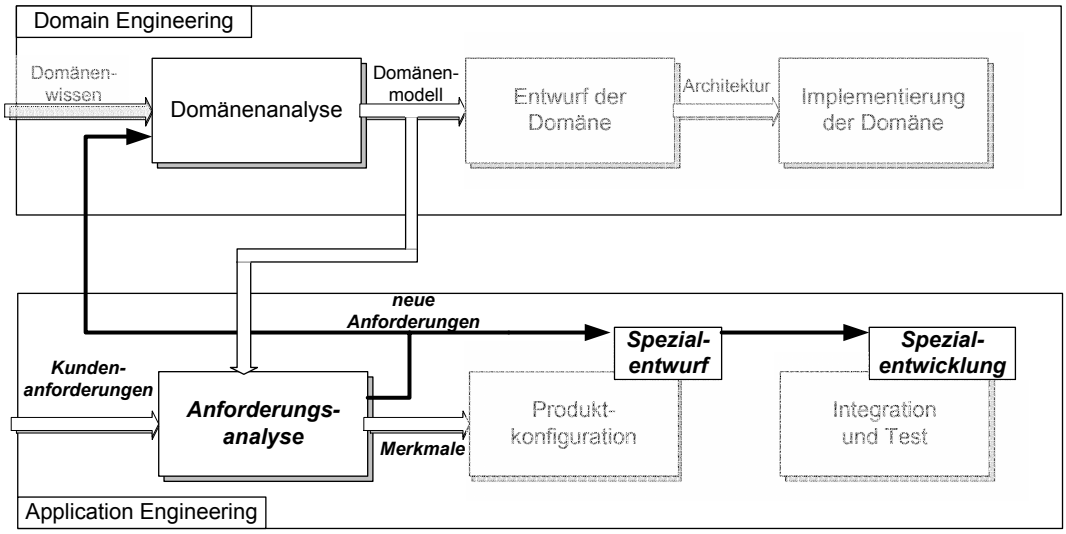


Abbildung 2.5: Die Anforderungsnalyse im Rahmen von Systemfamilien

Somit existieren im Rahmen der Anforderungsanalyse drei Möglichkeiten, inwiefern mit nicht abgedeckten Anforderungen verfahren werden kann, wobei insbesondere Kosten- und Nutzenüberlegungen in den Vordergrund treten:[32]

- Die Anforderungen werden an die Domänenanalyse weitergeleitet, wodurch ein erneuter Durchlauf des kompletten Domain Engineering angestoßen wird.
- Die Anforderungen bleiben nach Absprache mit dem Auftraggeber unberücksichtigt.

[32] vgl. [Böllert02]

- Die Anforderungen werden einmalig und individuell in das zu entwickelnde System eingearbeitet, was einen notwendigen Spezialentwurf und eine Spezialentwicklung zur Folge hat.

Das Ergebnis der Anforderungsanalyse bilden die identifizierten Merkmale als Ausgangspunkt für die Produktkonfiguration.

Produktkonfiguration (product configuration)
Dieser Schritt baut auf den Erkenntnissen des Domänenentwurfs und der Anforderungsanalyse auf. Sowohl die Merkmale, welche innerhalb der Anforderungsanalyse den Anforderungen zugeordnet wurden, als auch die zur Verfügung stehende Architektur zur Auswahl und Einordnung der Assets, werden herangezogen. Abbildung 2.6 stellt die Produktkonfiguration in diesem Zusammenhang dar.

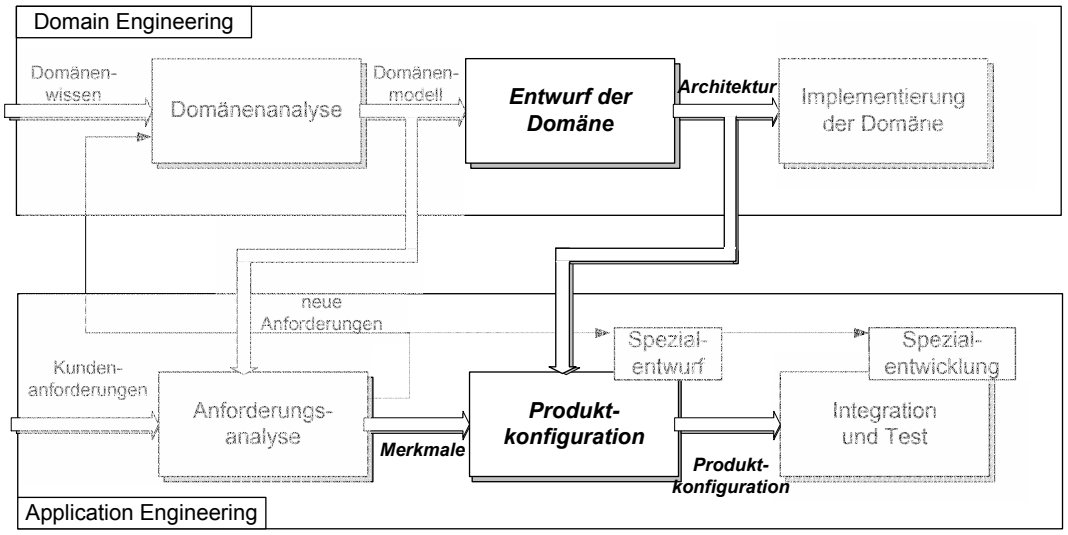


Abbildung 2.6: Die Produktkonfiguration

Durch die sich aus den Merkmalen ergebenden Vorgaben werden die Assets zugeordnet und Anwendung durch Komposition dieser gebildet. Im Falle eines Spezialentwurfs müssen ergänzende und sich integrierende Aspekte berücksichtigt werden und in die Modelle und Implementierungsansätze einfließen.

Integration und Test

Abschließend erfolgt die Umsetzung beziehungsweise Ableitung der Anwendung beziehungsweise des Systems, welches anschließend getestet und an den Kunden ausgeliefert wird. Anwendungen werden in Form von Komponenten entsprechend der in der Produktkonfiguration getroffenen Vorgaben zusammengesetzt. Dabei kommen gegebenenfalls domänenspezifische Sprachen zur Konfiguration der Komponenten untereinander oder Generatoren zum Einsatz, welche ohne manuelle Beisteuerung fertige Anwendungen erstellen sollen. Die vollständige Bearbeitung und die Zusammenhänge aller Bestandteile des Application- und Domain Engineerings sind in Abbildung 2.7 zusammenfassend dargestellt.

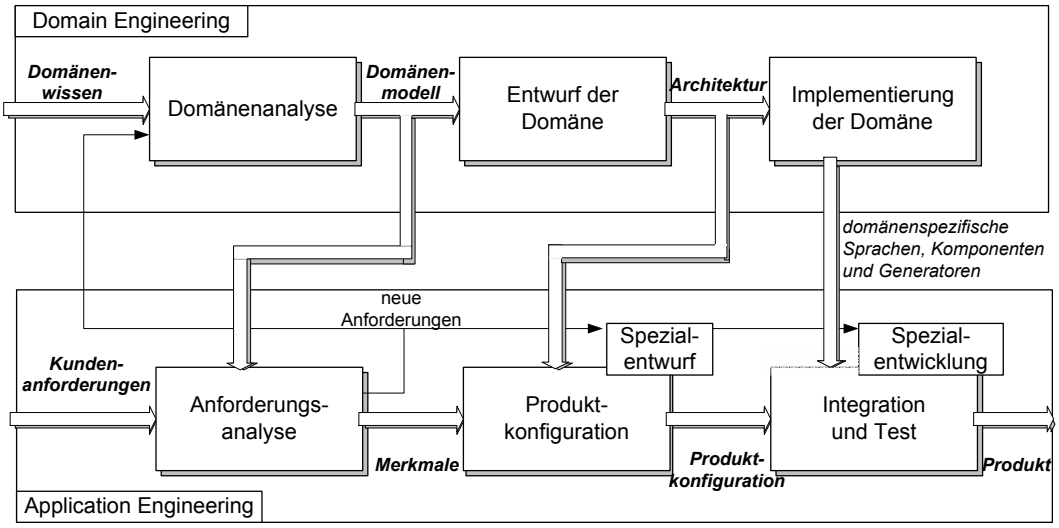


Abbildung 2.7: vollständiges Domain- und Application Engineering

2.3 Die Anforderungsanalyse im Kontext von Systemfamilien

Nach der Darlegung des allgemeinen Ablaufs zur Erstellung einer Systemfamilie und zur Ableitung konkreter Anwendungen sollen nun zentrale Aspekte des Ansatzes näher betrachtet werden. Die Erfassung und Aufbereitung von Anforde-

rungen und die Hinterlegung dieser als Merkmale der Systemfamilie ist Grundlage und Voraussetzung für die erfolgreiche Anwendung des Ansatzes. Diesbezüglich ist das Zusammenspiel der Anforderungs- und Domänenanalyse von besonderem Interesse. Die folgenden Ausführungen beinhalten Methoden zur Erfassung und Modellierung von Merkmalen und umfassen anschließend dabei auftretende Probleme.

2.3.1 Die Erfassung und Modellierung von Merkmalen

Steht das Anwendungsfeld der Systemfamilie, also die Domäne, einmal fest, so gilt die weitere Betrachtung der Auswahl der Merkmale. Neben technischen Überlegungen, welche ohnehin die Grundlage der Systemfamilie darstellen, sollten jedoch weitere Aspekte beachtet werden, um letztendlich den Erfolg des Ansatzes der Systemfamilien zu sichern. Dies betrifft unter anderem Fragestellungen, wie:[33]

- Welche Merkmale sind für den Erfolg eines Markteintritts besonders wichtig und sollten daher frühzeitig implementiert werden?
- Welche Merkmale sind für das längerfristige Überleben des Produkts auf dem Markt wichtig?
- Welche Merkmale sind mit besonderen Risiken behaftet?
- Welche Architektur ist für alle Mitglieder einer Systemfamilie geeignet?
- Wie kann ein Mitglied einer Systemfamilie rationell erstellt werden?
- Wie kann ein Mitglied einer Systemfamilie effizient beschrieben beziehungsweise erstellt werden?

Bei diesen weitgehend ökonomischen Betrachtungen gilt es zwar zunächst, möglichst schnell und effizient auf die Bedürfnisse des Marktes einzugehen, jedoch spielen noch weitere Überlegungen eine Rolle. Neben der resultierenden kostengünstigen, schnellen Entwicklung und damit einhergehenden wirtschaftlichen Vorteilen,[34] erfolgen auch Berücksichtigungen des Aufbaus einer möglichst breiten

[33] vgl. [Czarnecki01]
[34] siehe Abschnitt 4.2 - Bewertung der Kosten: economies of scale

Variantenvielfalt.[35] Ein Unternehmen kann also zunächst schnell in einen Markt eintreten, dort Vorteile im Zusammenhang mit einer Systemfamilie ausnutzen und anschließend neue Märkte erschließen. Vorteile ergeben sich dahingehend in einer sukzessiven Ausweitung und teilweisen Individualisierung der Angebotspalette.[36]

Die Auswahl der Merkmale einer Systemfamilie erfolgt jedoch nicht nur durch das Unternehmen, wie bereits am Zusammenspiel der Anforderungs- und Domänenanalyse erkennbar. Diesbezüglich sind zwei Punkte zu berücksichtigen. Zum einen dienen erfasste Merkmale der Bildung von Modellen und sind somit Ausgangspunkt für die konkrete Umsetzung der Systemfamilie. Zum anderen bilden Modelle oder in anderer Form aufbereitete Merkmale häufig die Basis für die Auswahl von Kundenanforderungen und somit die Grundlage für gegebenenfalls neu zu erfassende Merkmale. Dabei erfolgt wiederum die Berücksichtigung der eingangs aufgezeigten, wirtschaftlich geprägten Fragestellungen durch das Unternehmen. Die Erfassung und Darstellung von Merkmalen gewinnt unter diesem Blickwinkel an Bedeutung.

Darstellung der Merkmale

Innerhalb der anschließenden Ausführungen wird davon ausgegangen, dass eine Domäne bereits besteht und hinreichend abgegrenzt ist. Die dargestellten Möglichkeiten zur Erfassung und Modellierung von Merkmalen basieren weitestgehend auf der *„Feature Oriented Domain Analysis"* (FODA) und der *„Unified Modelling Language"* (UML). Genaue Hintergründe werden dabei vernachlässigt. Vielmehr soll das Verständnis der Vorgehensweise vermittelt werden. FODA stellt eine grundlegende und allgemeingültige Methode innerhalb des Bereichs der Domänenanalyse dar. Es wird ein Vorgehen spezifiziert und konkrete Ergebnisse einzelnen Phasen einer Anwendungsentwicklung mit Bezug zu einer Systemfamilie zugeordnet.[37]

Ausgangspunkt der Erfassung von Merkmalen bildet häufig ein Merkmalkatalog. Bezüglich der Anforderungsanalyse sind Merkmale hier für den Endanwender

[35] siehe Abschnitt 4.2 - Bewertung der Kosten: economies of scope

[36] vgl. [CMU/SEI99] S.27

[37] vgl. zu folgenden Abschnitt [Czarnecki00] S.47ff, Chapter5 Feature Modelling; [Streitferdt00]; [Capilla01]

sichtbare Eigenschaften eines Systems. Dieser Merkmalkatalog umfasst obligatorische und optionale Merkmale als Ausgangspunkt für weitere Entwicklungen. Der Nutzer wählt die gewünschten Eigenschaften eines Systems, welche wiederum dem Systementwickler als konkrete Vorgaben dienen.

Der nächste Schritt ist die Darstellung der Merkmale, insbesondere mit Berücksichtigung der gewählten obligatorischen und optionalen Merkmale. Das Modell steht dabei im Vordergrund, nicht jedoch die Implementierung der Merkmale. In Abbildung 2.8 ist ein Merkmalmodell für eine Anwendung skizziert. Dabei wird die Abwicklung einer Lebensversicherung aus Sicht eines Versicherungsunternehmens dargestellt.

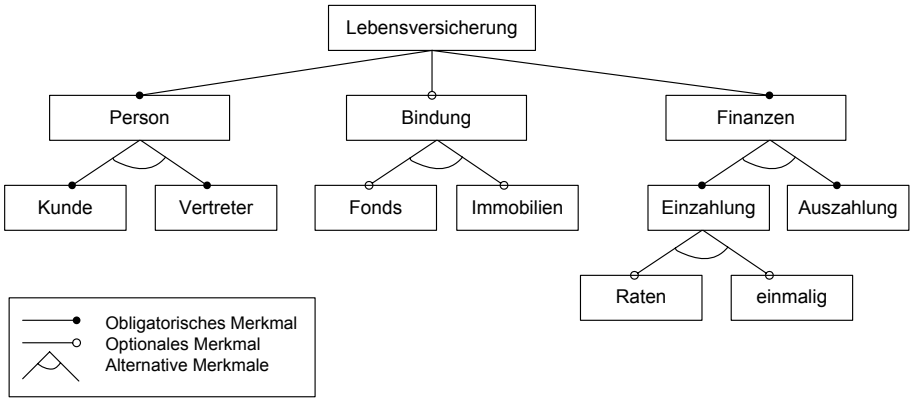


Abbildung 2.8: Merkmalmodell

Der skizzierte Ausschnitt beinhaltet, neben den stets zwingend vorgeschriebenen *Grundfunktionen* der Software, das obligatorische Merkmal „*Einzahlung*". Wie jedoch eingezahlt wird, kann der Kunde selbst entscheiden. Es besteht eine Alternative zwischen einer Einmal- und einer Ratenzahlung. Zudem kann der Kunde optional eine *Zusatzbindung der Versicherung* vereinbaren, wodurch sich sein Auszahlungsbetrag erhöhen kann. Dabei kann dieser zwischen einem *Fonds* oder einer Anlage in *Immobilien* wählen. Die zusätzlich modellierten Merkmale basieren auf den Grundfunktionalitäten der Anwendung.

Das Merkmalmodell bildet die Grundlage für weitere Modelle. In diesem Zusammenhang ist die bereits erwähnte *„Unified Modelling Language"* (UML) her-

vorzuheben. Innerhalb dieser bestehen zahlreiche Modellierungsvarianten, welche die Gegebenheiten aktueller, objektorientierter Programmiersprachen berücksichtigen. Dabei sind die Modelle als Ausgangsbasis für die Implementierung einer Anwendung zu betrachten. In Abbildung 2.9 wurden auf Basis der geforderten Merkmale Klassen gebildet und zueinander in Beziehung gebracht.

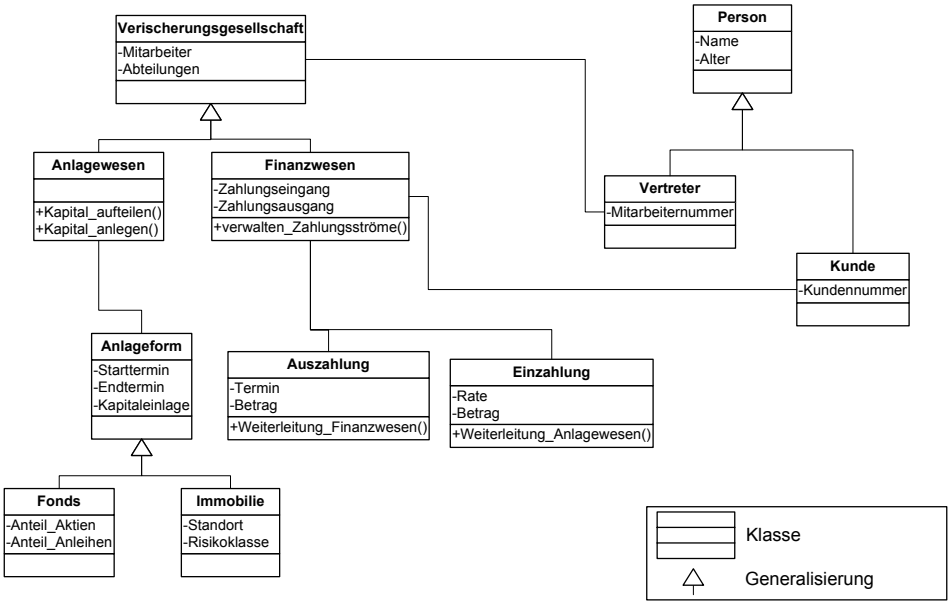


Abbildung 2.9: Auf dem Merkmalmodell aufbauendes Klassenmodell

Aufgrund des flexiblen Zusammenspiels der Merkmale, gestaltet sich der Aufbau einer wie in Abbildung 2.9 dargestellten, statischen Architektur oft schwierig. Ein Merkmal ist nicht auf einen bestimmten Bereich der Architektur beschränkt, sondern wirkt sich auf verschiedene der dargestellten Klassen aus. Ein Klassenmodell dient jedoch der Übersichtlichkeit und Verständlichkeit der Architektur der zugrundeliegenden Software im Hinblick auf Änderungen und Entwicklungen von Anwendungen.

Somit besteht schon ein wichtiger Ansatz zur Implementierung einer Anwendung mit den eingangs geforderten Merkmalen. Im Zusammenhang mit Systemfamilien reicht dies jedoch nicht aus. Die in Abbildung 2.9 dargestellte, statische Archi-

tektur berücksichtigt nicht in anderen Kontexten auftretende Variabilitäten. Ein weiterer Schritt ist daher in Abbildung 2.10 skizziert:

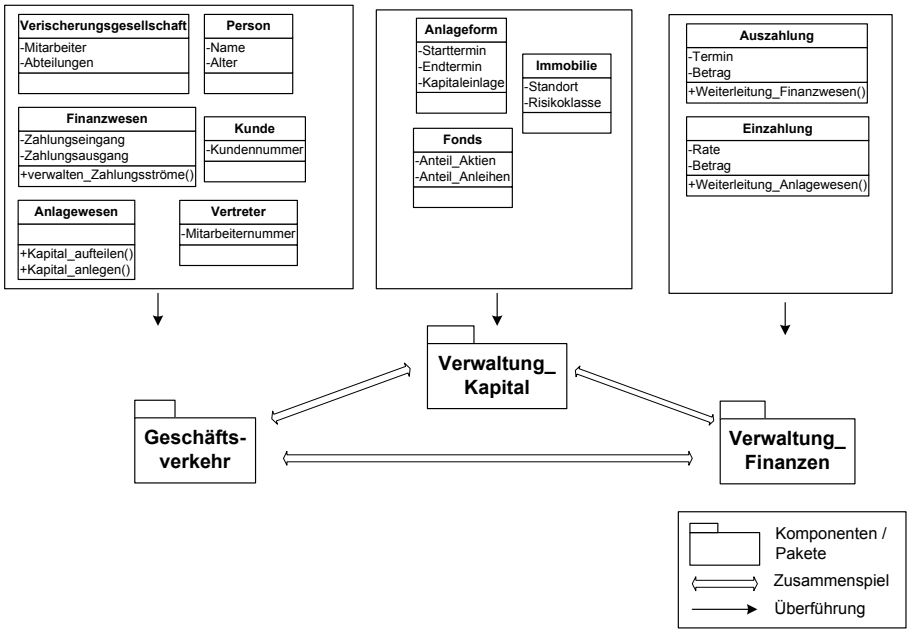


Abbildung 2.10: Darstellung von Variabilitäten anhand eines Komponentenmodells

Die Klassen werden innerhalb von Paketen oder Komponenten zusammengefasst. In der Regel betrifft dies die Klassen, welche letztendlich ein Merkmal realisieren. Die Komponenten können wiederum variabel kombiniert und je nach von Kundenseite gewünschter Funktionalität eingesetzt werden.

Das aufgezeigte Vorgehen stellt einen idealisierten Ablauf dar. Oft treten Probleme und Fehler auf, welche vorrangig in der Komplexität einer Systemfamilie und der Vielzahl möglicher Kundenanforderungen begründet sind. Im Anschluss wird darauf genauer eingegangen.

2.3.2 Das Problemfeld der Anforderungsanalyse

Die meisten Schritte zur Bildung einer Systemfamilie und zur Ableitung konkreter Anwendungen finden in zahlreichen Methoden und Vorgehensmodellen Un-

terstützung. Zwar ist dabei der Bereich der Anforderungsanalyse weitestgehend eingeschlossen, jedoch ergeben sich vor allem hier zahlreiche Fehlerpotenziale.[38]

Ein häufiges Problem stellt die Menge der Anforderungen dar, welche innerhalb einer bestimmten Domäne relevant werden können. Die korrekte Zuordnung gemeinsamer oder variabler Merkmale einer Systemfamilie gestaltet sich mit sinkender Abstraktionsebene zunehmend schwierig. Ein allgemeines Merkmal, welches als gemeinsamer Bestandteil aller Systemmitglieder erfasst wurde, kann sich beispielsweise bei genauer Betrachtung als zu ungenau und am Ende als variable Anforderung herausstellen. Eine zunehmende Aufsplittung sorgt jedoch auch für eine zunehmende Komplexität, was wiederum Kosten- und Nutzenüberlegungen aufwirft. Auch sorgt eine zu allgemeine Darlegung der Merkmale für erhöhte Aufwände in der Erstellung der Anwendung, da nun zusätzliche Implementierungen hinsichtlich der Anforderungen notwendig werden können.[39]

Unter diesem Kontext ist zudem der Umgang mit Anforderungen zu sehen, welche nicht durch die Merkmale der Systemfamilie abgedeckt werden. Falls diese als neue Merkmale in die Systemfamilie übernommen werden sollen, stellt sich die Frage nach deren Erfassung und Integration. Dabei ist zu bedenken, inwieweit es sich unter ökonomischen Gesichtspunkten lohnt, neue Merkmale in die Systemfamilie aufzunehmen oder eventuell eine Spezialentwicklung zu verfolgen. In diesem Zusammenhang sind auch die zunehmend kurzen Lebenszyklen der Endprodukte zu nennen. Ständig finden Weiterentwicklungen statt und neue, teils innovative Eigenschaften werden Bestandteil neuer Anwendungen.[40]

Überlegungen in diesem Zusammenhang führen zu einer sinnvollen Notation der Merkmale und zu einer Integration von Wirtschaftlichkeitsbetrachtungen im Sinne einer Aufwandsschätzung. Im folgenden Kapitel werden daher zunächst relevante Methoden und Verfahren vorgestellt, während im Anschluss daran eine Lösung hinsichtlich der Integration in die dargestellte Anforderungsanalyse angestrebt wird.

[38] vgl. [CMU/SEI99] S.9ff,S.19; [CMU/SEI97] S.6,16f.
[39] vgl. [PULSE02]
[40] vgl. [Cohen02] S.3ff.

3 Aufwandsschätzung von Softwareprojekten

Innerhalb einer Softwareentwicklung erlangen frühzeitige Aussagen über entstehende Zeit- und Kostenaufwendungen eine zunehmende Bedeutung. Dabei ist die Verlässlichkeit der in diesem Zusammenhang auftretenden Aussagen entscheidend, insbesondere in dem schwierigen wettbewerblichen Umfeld, welches die Softwareentwicklung derzeit darstellt.[1] Ergebnisse und Erkenntnisse der Vergangenheit zeigen jedoch, dass komplexe Aufgabenstellungen häufig zu teils fatalen Fehleinschätzungen des Softwareentwicklungsprozesses führen. Daher besteht die Notwendigkeit zu einer geplanten und strukturierten Vorgehensweise einer Aufwandsschätzung.[2] Aus diesem Grund soll eingangs des Kapitels innerhalb grundlegender Ausführungen auf Fehlerpotenziale und häufig anzutreffende Probleme der Aufwandsschätzung eingegangen werden.

Der Bezug einer Aufwandsschätzung zur Entwicklung von Systemfamilien besteht in der Analyse und Abschätzung von Neuentwicklungen oder nicht vorhandenen Systembestandteilen beziehungsweise Assets.[3] Dies beinhaltet vor allem klassische „make-or-buy" Entscheidungen, also die Auswahl zwischen Fremdbezug und Eigenfertigung. Dass aber insbesondere bereits entwickelte Systeme mit Hilfe wieder verwendbarer Assets eine wichtige Grundlage für zahlreiche Aufwandsschätzungen darstellen und somit allein durch die Existenz einer Systemfamilie Ansätze

[1] vgl. [Bredemeier02]
[2] vgl. [Litke96] S.1f.
[3] vgl. [PULSE02] S.2

zur Aufwandsschätzung bestehen, soll durch folgende Ausführungen belegt werden. Abschließend werden am Ende des Kapitels konkrete Verfahren vorgestellt, welche insbesondere Aspekte der Wiederverwendung berücksichtigen und im Sinne der vorliegenden Arbeit sinnvoll erscheinen.

3.1 Die Grundlagen der Aufwandsschätzung

Die Kenntnis und Anwendung der im späteren Verlauf des Kapitels vorgestellten Aufwandsschätzmethoden und -verfahren sichert in der Regel nicht den Erfolg der Schätzung. Vielmehr spielen das Wissen über Fehlerursachen und Probleme der Aufwandsschätzung eine wichtige Rolle. Im folgenden Abschnitt soll daher zunächst auf grundlegende Zusammenhänge eingegangen werden. Anfangs werden zum allgemeinen Verständnis wichtige Begriffe erklärt und abgegrenzt.

3.1.1 Begriffsklärung

Eine Aufwandsschätzung wird im Folgenden verstanden als:[4]

„eine Schätzung des zeitlichen Aufwands eines Softwareprojekts beziehungsweise Projektteils".

Die Ergebnisse dieser Schätzung werden in Einheiten von *Personenstunden* bis zu *Personenjahren* angegeben. Mit Hilfe dieser Werte lassen sich durch die Verwendung fixer Umrechnungsfaktoren Dauer und Kosten eines Softwareprojekts ermitteln.

Weiterhin kann in der Verwendung des Begriffs Aufwandsschätzung zwischen *Aufwandsschätzmethoden* und *Aufwandsschätzverfahren* unterschieden werden:[5]

- Eine Aufwandsschätzmethode stellt einen meist funktionalen Zusammenhang zwischen Produkt und Aufwand her, wobei jede relevante Entwicklungsgröße in diesem Zusammenhang berücksichtigt werden kann.

[4] vgl. [Bundschuh00] S.17
[5] in Anlehnung an [Heinrich97] S.211ff.

- Ein Aufwandsschätzverfahren bedient sich in seiner Anwendung dieser Methoden und stellt letztendlich eine konkrete Vorgehensweise der Schätzung dar.

3.1.2 Umfeld und Probleme der Aufwandsschätzung

Allein die Anzahl existierender und zum Teil hier vorgestellter Schätzmethoden und -verfahren lässt vermuten, dass eine Aufwandsschätzung ein schwieriges Thema in der betrieblichen Praxis darstellt. Der korrekten Prognose von Zeit und Kosten stehen zahlreiche Probleme und Fehlerursachen gegenüber. Somit sind grundlegende Überlegungen vor der Anwendung einer Aufwandsschätzung zu treffen, welche im folgenden Abschnitt kurz vorgestellt werden.[6]

Das Dilemma der Aufwandsschätzung
Als wohl bedeutendste Barriere in der Anwendung einer Aufwandsschätzung wird gewertet, dass eine korrekt durchgeführte Schätzung keinen wettbewerblichen beziehungsweise wirtschaftlichen Erfolg garantiert. Vielmehr können erhebliche betriebliche Defizite, welche sich in hohen Kosten- und Zeitergebnissen einer Schätzung niederschlagen, zu einem Verlust eines potenziellen Kundens führen. Jedoch stellt eine schlechte und ungenaue Schätzung einen gleichermaßen gravierenden Nachteil dar.

In der Betrachtung der Ergebnisse einer Schätzung zeigt sich, dass bei einer zu geringen Schätzung Termine und Kosten in zum Teil nicht unerheblichen Maße überschritten werden und gegebenenfalls unter dem anfallenden Termindruck Qualitätseinbußen hinzunehmen sind. Zum anderen können zu hohe Ergebnisse der Aufwandsschätzung einen Nachteil im Wettbewerb um den Kunden bedeuten und zum Teil eine Ausdehnung der Arbeit auf den zu hoch geschätzten Aufwand bewirken.[7] Dies bedeutet, dass zu hohe Zeit- und Kostenrahmen von den an der Entwicklung beteiligten Personen in der Regel voll ausgeschöpft werden, auch wenn die Entwicklung weitaus schneller und sparender verlaufen könnte.

[6] vgl. zu folgendem Abschnitt [Litke96] S.3-31
[7] Parkinsonsches Gesetz formuliert nach C. Northcote Parkinson, vgl. [Bundschuh00] S. 24

Zur Minimierung des Risikos einer Fehleinschätzung des Aufwands wird gemäß allgemeiner Auffassungen eine Anwendung mehrerer Schätzmethoden und -verfahren zu verschiedenen Zeitpunkten der Softwareentwicklung empfohlen.

Der Konflikt innerhalb der Aufwandsschätzung

Oft sind es politische beziehungsweise psychologische Ursachen, welche für das Scheitern einer Aufwandsschätzung herangezogen werden müssen. Schätzfehler und Fehlurteile werden selten von den Verantwortlichen zugegeben und daher selten korrigiert. Statt dessen wird mit Hilfe aller verfügbaren Kapazitäten versucht, die Ergebnisse der ersten und meist einzigen Schätzung zu bestätigen. Die dabei kaum zu erreichenden Zielvorgaben sorgen in der Regel für qualitative Einbußen der Endprodukte.

Die Genauigkeit der Aufwandsschätzung

Bei der Anwendung der Aufwandsschätzung ist weiterhin zu beachten, dass eine Schätzung immer den Charakter einer Prognose hat, inklusive entsprechender Unsicherheiten. Dies liegt hauptsächlich darin begründet, dass zahlreiche für die Schätzung relevante Einflussfaktoren zu Beginn eines Projekts meist nicht vollständig bekannt sind.[8] Demzufolge nimmt die Unsicherheit bezüglich der Schätzergebnisse mit zunehmenden Projektfortschritt ab. Abbildung 3.1 umfasst eine zeitliche Berücksichtigung dieser Unsicherheiten mittels optimistischer und pessimistischer Werte.[9]

[8] vgl. [Heinrich97] S.212ff.

[9] in Anlehnung an [Bundschuh00] S.25; In der Literatur bestehen differenzierte Annahmen zu den Werten der Schätzfehler und Schätzgenauigkeit vgl. [Boehm00] S.10

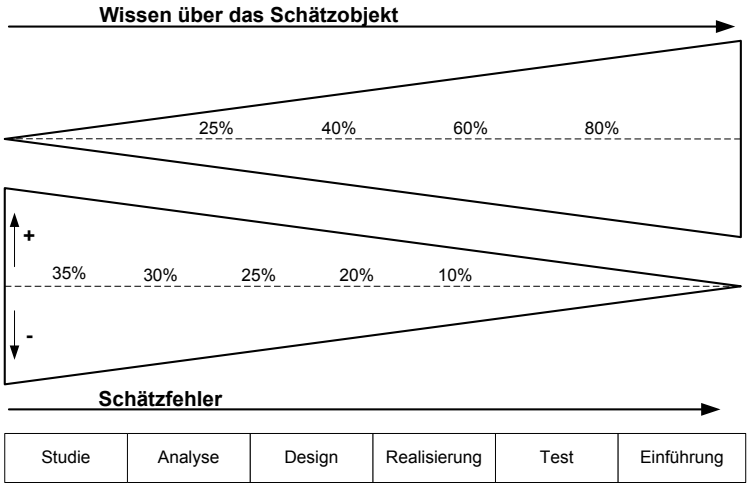


Abbildung 3.1: Abhängigkeit der Aufwandsschätzung vom zeitlichen Einsatz

Wie bereits erwähnt, wird von zahlreichen Autoren eine Wiederholung der Aufwandsschätzung mit zum Teil unterschiedlichen Methoden und Verfahren empfohlen. Dies hat zum einen den Vorteil, dass dieses zu Kontrollmöglichkeiten hinsichtlich der geschätzten Kosten und Termine führt und zum anderen können Unsicherheiten zunehmend ausgeschlossen werden. Aufgrund der bereits dargestellten politischen und psychologischen Probleme, wird dies jedoch in der betrieblichen Praxis häufig vernachlässigt.

Die Ergebnisse einer Aufwandsschätzung

Auch ist bei der Interpretation der Schätzergebnisse Vorsicht geboten. Zwar lassen die Ergebnisse, welche meist in Form von Personenmonaten vorliegen, einen gewissen Spielraum in der personellen Auslastung eines Projekts vermuten, jedoch trifft dies nur bedingt zu. Wie in Abbildung 3.2 skizziert,[10] kann aufgrund der Unteilbarkeit vieler Aufgaben ein zusätzlicher positiver Beitrag durch einen weiteren Mitarbeiter ausgeschlossen werden.[11]

[10] in Anlehnung an [Litke96] S.24

[11] Brooksches Gesetz: *Adding man power to a late project makes it even later* vgl. [Heinrich97] S. 105f,S.211

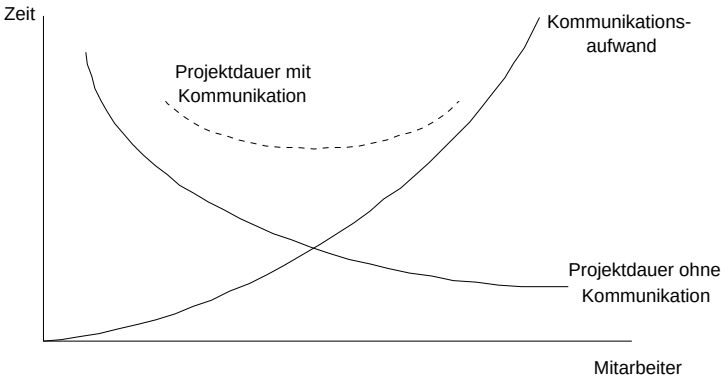


Abbildung 3.2: Brooksches Gesetz

Ab einem gewissen Punkt steigt der Kommunikationsaufwand zur Umsetzung einer Aufgabe derart an, dass die Projektdauer trotz zusätzlicher Mitarbeiter steigt. Dabei ist zu beachten, dass der Kommunikationsaufwand abhängig von organisationellen Rahmenbedingungen und somit von Unternehmen zu Unternehmen unterschiedlich zu werten ist.

Zur Vermeidung von Fehlurteilen wird zudem eine genaue Definition der Schätzergebnisse empfohlen. So sollte beispielsweise bei der Angabe der Schätzergebnisse von Personenstunden bis Personenjahren ein genauer Zusammenhang zwischen diesen Größen dargelegt sein.

3.1.3 Die Anwendung und Umsetzung einer Aufwandsschätzung

Im folgenden Abschnitt sollen zunächst einige Grundgedanken und allgemeine Voraussetzungen zur Anwendung einer Aufwandsschätzung dargelegt werden. Zwar können hier getroffene Aussagen im Widerspruch zu im späteren Verlauf der Arbeit vorgestellten Schätzverfahren stehen, welche ohnehin einen konkreten Ablauf beinhalten, jedoch steht die allgemeine Vorgehensweise hier im Vordergrund.

Dokumentation als Grundlage der Aufwandsschätzung

Der wohl gebräuchlichste Ansatz für eine Aufwandsschätzung ist eine Betrachtung bereits abgeschlossener Softwareprojekte. Erfahrungen aus der Vergangenheit können einen entscheidenden Beitrag für den Erfolg aktueller und zukünftiger Entwicklungen darstellen. Zur Analyse und der darauf aufbauenden Kalkulation von Zeit und Kosten ist eine *Dokumentation* die Basis für die Nachvollziehbarkeit und somit Vergleichbarkeit unterschiedlicher Projekte.[12] Im Allgemeinen wird in diesem Zusammenhang vom Aufbau einer *Wissensbasis* gesprochen, welche durch unterschiedliche Ansätze zur Informationsgewinnung mit Hilfe sogenannter *Einflusskriterien*[13] umgesetzt wird. Auch lassen sich durch eine authentische Dokumentation des verwendeten Aufwandsschätzverfahrens wertvolle Ansätze zu dessen Weiterentwicklung finden, so dass im Laufe der Zeit Potenziale zur Steigerung der Qualität der Aufwandsschätzung bestehen.

Ausgangsgrößen der Aufwandsschätzung

Ausgangspunkt einer Aufwandsschätzung bildet die Bestimmung der voraussichtlichen Projektgröße. Grundlegend wird dazu die Anzahl der Programmcodezeilen verwendet.[14] Da diese Größe den zahlreichen und unterschiedlichen Merkmalen verschiedenster Programmiersprachen nicht gerecht werden kann, werden häufig andere Messgrößen zur Bestimmung des Umfangs des Programmquellcodes herangezogen, wie unter anderem die Anzahl logischer Programmbefehle oder die Anzahl verwendeter Funktionen oder Klassen.[15] Jedoch besteht auch in der Verwendung dieser Größen das Problem, dass innerhalb der Schätzung die technische Realisierung der Programme berücksichtigt werden muss. Daher bestehen Ansätze, in welchen allein die Funktionalität der zu entwickelnden Programme betrachtet wird.[16] Vorrangig betrifft dies die aus Nutzersicht dargestellten Interaktions- und Einflussmöglichkeiten mit der Software. Dazu zählen beispielsweise die im späteren Verlauf der Arbeit vorgestellten *Function Points*.

[12] vgl. [Bundschuh00] S.27ff.
[13] vgl. [Litke96] S.25
[14] in der Regel: Lines of Code [LOC]
[15] vgl. [Stahlknecht99] S. 475f.
[16] vgl. [Thaller00] S.41ff.

Die Einflüsse auf den Aufwand

Neben der Projektgröße sind zahlreiche, für eine Softwareentwicklung relevante Einflussfaktoren von Interesse. Diese oft als *Kostentreiber* bezeichneten Elemente sind bewertbare Kenngrößen, welche innerhalb verschiedener Schätzmethoden und -verfahren unterschiedlich stark berücksichtigt werden. Abbildung 3.3 beinhaltet eine mögliche Klassifizierung der zahlreichen innerhalb einer Softwareentwicklung anzutreffenden Einflussfaktoren.[17]

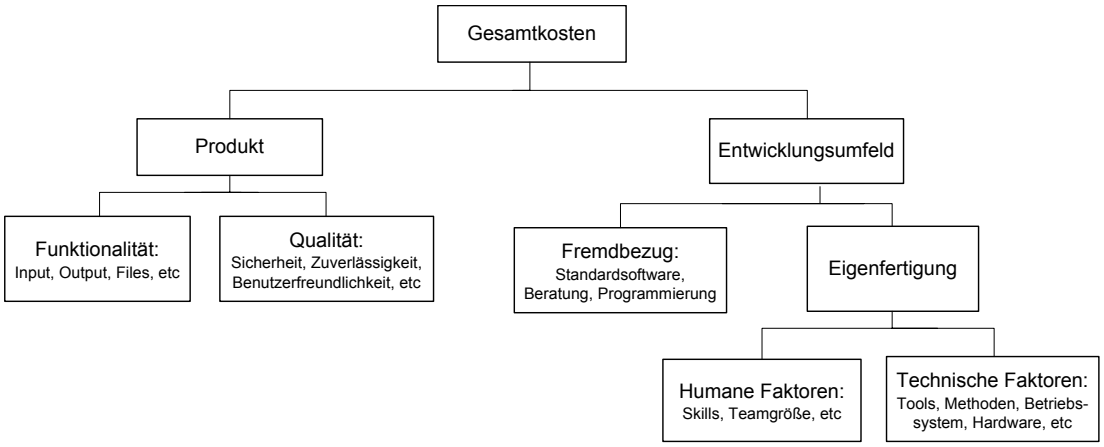


Abbildung 3.3: Struktur der Kostenverursacher

Die Kosten eines Softwareprodukts gliedern sich hier in:[18]

- Kosten durch die Erstellung des Produktes, abhängig von seinem Umfang und der geforderten Qualität sowie
- Kosten durch die Art der Herstellung und Fertigung.

Während die Einflusskategorien *Funktionalität* und *Qualität* durch die Anforderungen des Kunden bestimmt werden, so liegt die Entscheidung nach Art der Fertigung, also die eingangs erwähnte „make-or-buy" Entscheidung, in den Händen des Softwareproduzenten. Auch verdeutlicht die dargestellte Unterteilung den Einfluss humaner Faktoren auf die Kosten beziehungsweise den Aufwand eines

[17] in Anlehnung an [Hürten99] S.3
[18] vgl. zu folgendem Absatz [Hürten99] S. 3ff.

Projekts, was wiederum eine Aufwandsschätzung vor besondere Herausforderungen stellt.

Die Qualität einer Aufwandsschätzung kann demnach damit begründet werden, inwieweit verschiedenste Einflussfaktoren innerhalb eines Schätzverfahrens berücksichtigt werden. Aktuelle Verfahren erlauben neben der Bestimmung der Quantität, anhand des Umfangs des Entwicklungsprojekts, die Festlegung von Komplexität und Qualität mittels Wichtungsfaktoren und spezifischer Merkmalsausprägungen.

Der grundlegende Ablauf einer Aufwandsschätzung

Mit Hilfe von Abbildung 3.4 soll zunächst exemplarisch die bisher dargestellte Anwendung und Umsetzung der Aufwandsschätzung aufgezeigt werden.[19] Der Aufbau entspricht dabei den innerhalb dieses Abschnitts dargelegten grundlegenden Überlegungen.

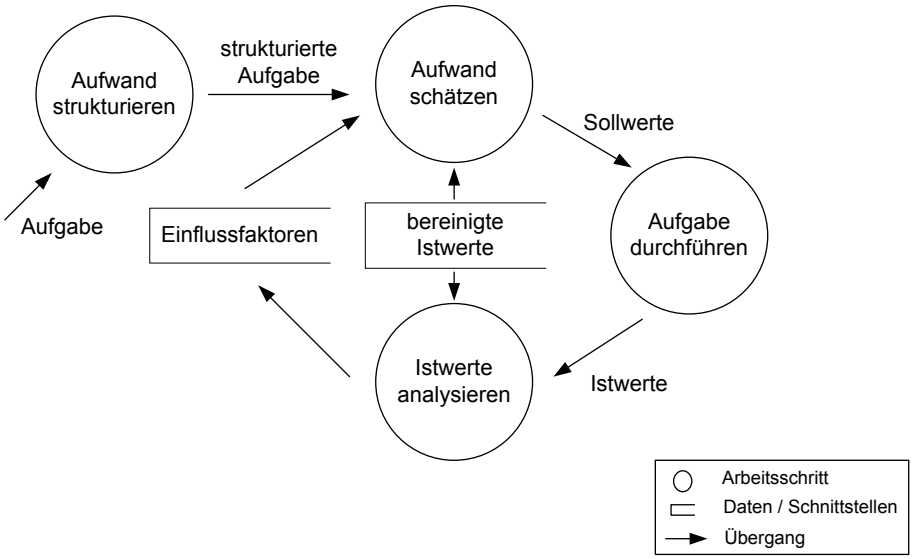


Abbildung 3.4: idealisierter Ablauf der Aufwandsschätzung

[19] in Anlehnung an [Litke96] S.6

Zunächst erfolgt in dem hier dargestellten Schätzprozess die *Strukturierung der Aufgabenstellung*. Dies dient zum einen der Feststellung zu berücksichtigender Einflussfaktoren und zum anderen der möglichst weitgehenden Minimierung der dargestellten Unsicherheit in vorrangig frühen Phasen einer Softwareentwicklung. Das Ergebnis der Strukturierung und zugleich Grundlage der Schätzung bildet hier eine möglichst genau spezifizierte Projektgröße. Die Resultate der Aufwandsschätzung fließen in einem nächsten Schritt als *Sollwerte* in den Entwicklungsprozess ein. Nun beginnt eine dynamische Betrachtung der Aufwandsschätzung. Die in Abbildung 3.4 dargestellte Verfahrensweise beruht auf der Idee einer sich wiederholenden Aufwandsschätzung.[20] Sobald die Sollvorgaben verfehlt werden, findet eine Analyse der *Istwerte* inklusive einer Suche und Berücksichtigung bisher eventuell vernachlässigter *Einflussfaktoren* statt. Im Anschluss wird erneut eine Aufwandsschätzung durchgeführt, wobei neue Soll-Vorgaben ermittelt werden.

Die hier dargestellte Vorgehensweise stellt einen idealisierten Verlauf der Aufwandsschätzung für ein Projekt dar, denn meist berücksichtigen die verschiedenen Aufwandsschätzverfahren nur einen bestimmten, klar definierten Katalog von Einflussfaktoren.

3.2 Aufwandsschätzmethoden

Innerhalb dieses Abschnitts sollen die wichtigsten sowie aktuelle und für die vorliegende Arbeit relevante Schätzmethoden vorgestellt werden.[21] Zunächst werden dazu grundlegende Methoden aufgezeigt, während im Anschluss die darauf aufbauenden Schätzverfahren behandelt werden. Dabei wird keinesfalls ein Anspruch auf Vollständigkeit erhoben, vielmehr sollen grundlegende und praxistaugliche Aspekte der Aufwandsschätzung in den Vordergrund treten. In Abbildung 3.5 erfolgt eine Klassifizierung der im Anschluss beschriebenen Aufwandsschätzmethoden.

[20] siehe Abschnitt 3.1.2 - Umfeld und Probleme der Aufwandsschätzung
[21] vgl. zu folgendem Abschnitt [Burghardt01] S. 85ff.

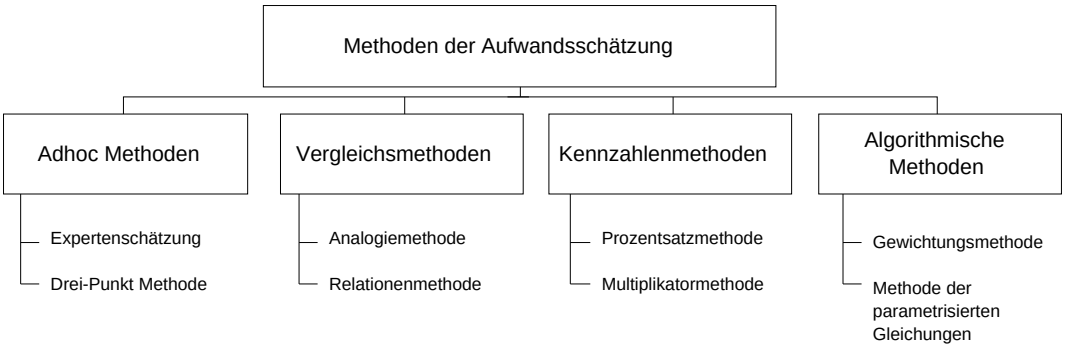


Abbildung 3.5: Klassifizierung der Aufwandsschätzmethoden

3.2.1 Adhoc Methoden

Zunächst sollen hier die Adhoc Methoden als Grundlage der wohl gängigsten betrieblichen Praxis vorgestellt werden. Trotz aller wissenschaftlich fundierter Methoden sind Schätzungen aufgrund des Wissens und der Erfahrungen einzelner oder mehrerer sogenannter Experten häufig die ausschlaggebende Aussage in der Festlegung von Terminen und Kosten. Oft werden diese Methoden daher als *Expertenschätzungen* bezeichnet. Dem Wert der innerhalb dieser Schätzung getroffenen Aussagen wird dabei in der Literatur eine recht unterschiedliche Bedeutung beigemessen. Während einige Autoren gänzlich von diesen Methoden als „Ursache katastrophaler Fehleinschätzungen" abraten,[22] sehen andere in den Adhoc Methoden eine sinnvolle Ergänzung in inhomogenen oder in über die ausschließliche Softwareentwicklung hinausgehenden Projekten.[23] Die folgende Abbildung beinhaltet die schematische Darstellung der unterschiedlichen Adhoc Methoden, welche im Anschluss vorgestellt werden.[24]

[22] vgl. [Litke96] S.32
[23] vgl. [Burghardt01] S.32
[24] in Anlehnung an [Burghardt01] S.108

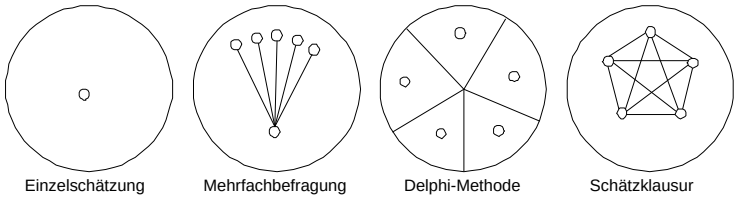


Abbildung 3.6: Formen der Expertenbefragungen

Die Einzelschätzung

Im Rahmen einer Einzelschätzung legt ein einzelner Mitarbeiter, in der Regel ist dies der Projektleiter, den voraussichtlichen Aufwand für ein gewisses Arbeitsvolumen fest. Die Genauigkeit der Schätzergebnisse hängt dabei vom Fachwissen, den Projektkenntnissen und den Erfahrungen des Schätzers ab. Trotz der Gefahren einer einseitigen Problembetrachtung wird der Einzelschätzung ein großer Einfluss in der betrieblichen Praxis beigemessen.

Die Mehrfachbefragung

Diese Form der Expertenschätzung greift auf die Aussagen mehrerer Schätzer zurück. Im Idealfall betrachten diese das zu schätzende Projekt getrennt voneinander und von unterschiedlichen Standpunkten aus. Das Endergebnis wird mittels einer Durchschnittsbildung errechnet. Durch die Anwendung einer Mehrfachbefragung soll eine Verringerung der Vorhersagefehler einer Einzelschätzung bewirkt werden.

Die Delphi Methode

Die Delphi Methode ist eine streng systematisch aufgebaute Befragung mehrerer Experten. Im Gegensatz zur Mehrfachbefragung ist es hier zwingend vorgesehen, dass die Schätzungen getrennt voneinander abgegeben werden. Der grundlegende Ablauf der Delphi Methode gliedert sich wie folgt:

1. Ein Koordinator erläutert jedem Experten die Entwicklungsaufgabe und händigt jedem ein Schätzformular aus.

2. Die einzelnen Experten füllen die Schätzformulare ohne die Möglichkeit zur Absprache getrennt voneinander aus. Als Kontaktperson für fachliche Fragen steht lediglich der Koordinator zur Verfügung.
3. Die Ergebnisse der Schätzung werden vom Koordinator zusammengefasst und den Schätzern vorgelegt.
4. Die Experten korrigieren, wieder voneinander getrennt, ihre Schätzungen.
5. Die Schritte 1 bis 4 werden solange wiederholt, bis sich die Schätzergebnisse der einzelnen Experten in ausreichendem Maße angenähert haben.

Auch hier wird im Anschluss das endgültige Schätzergebnis mittels der Durchschnittsbildung errechnet. Die dargestellte Vorgehensweise der Schätzung sorgt zwar für eine relativ hohe Genauigkeit der Schätzergebnisse, jedoch ist die Delphi Methode mit einem zum Teil beachtlichen Zeitaufwand verbunden.

Die Schätzklausur

Im Gegensatz zur Delphi Methode schätzen innerhalb dieser Methode die Experten gemeinsam und im gegenseitigem Austausch den Aufwand eines Projekts. In einer Diskussion entstehende, überzeugende Argumente und Standpunkte fließen somit in die Ergebnisse der einzelnen Experten ein. Gruppendynamische Aspekte, wie beispielsweise Einflüsse auf die Meinungsbildung durch dominante Gruppenmitglieder, können jedoch zur Verzerrung des Schätzergebnisses führen.

Die Drei-Punkt Methode

Zwar wird die Drei-Punkt Methode[25] selten in einen Zusammenhang mit Expertenschätzungen gebracht, jedoch spielen Überlegungen in der Anwendung dieser Methode auch eine Rolle bei der subjektiven Einschätzung eines Projektaufwands. Grundlage bilden eine optimistische Einschätzung des Aufwands (A_O), eine pessimistische Einschätzung (A_P) und eine Abschätzung des Aufwands, welcher am wahrscheinlichsten zu erwarten ist (A_W). Mit Hilfe einer Wahrscheinlichkeitsdichteverteilung[26] wird der mittlere zu erwartende Aufwand errechnet:

$$A_s = \frac{A_o + 4A_w + A_p}{6}$$

[25] auch unter der Bezeichnung Beta-Methode geläufig
[26] auch Beta-Verteilung genannt

Die Grundlage für die Einschätzungen des optimistischen, pessimistischen und wahrscheinlichsten Aufwands können wiederum die verschiedenen Methoden der Expertenschätzung bilden.

3.2.2 Vergleichsmethoden

Diese Methoden der Aufwandsschätzung greifen auf Erfahrungswerte vergangener Softwareentwicklungen zurück. Mit Hilfe festgelegter Kriterien wird ein aktuelles Projekt anhand seiner Merkmale mit den idealerweise in einer Datenbasis[27] hinterlegten Werten verglichen. Bei weitestgehender Übereinstimmung wird der Gesamtaufwand eines vergangenen Projekts übernommen. Die Vergleiche können jedoch beliebig verfeinert werden, was wiederum höhere Anforderungen an die Dokumentation voraussetzt. Somit können einzelne Bestandteile der zu entwickelnden Software verglichen werden, wodurch in der Regel eine ansteigende Genauigkeit der Ergebnisse erreicht wird. Als Vorteil dieser Methoden wird gewertet, dass bereits Aussagen zu den zu erwartenden Aufwänden in frühen Stadien der Softwareentwicklung getroffen werden können. Als typische Vertreter werden im Anschluss die Analogie- und die Relationenmethode vorgestellt.

Die Analogiemethode

In der Anwendung der Analogiemethode wird mittels der eingangs erwähnten Vergleichskriterien versucht, Aussagen über die Ähnlichkeit verschiedener Projekte zu treffen. Dazu werden anhand projektspezifischer Merkmale sogenannte Leistungsprofile gebildet. In der praktischen Umsetzung werden dabei vor allem Projekte zum Vergleich herangezogen, welche unter ähnlichen Voraussetzungen und Anforderungen realisiert worden sind. Die Einflussfaktoren auf den Projektaufwand[28] stimmen weitestgehend überein. Im Idealfall ist ein Projekt anhand seines Leistungsprofils derart genau beschrieben, dass eine beliebige Granularisierung eines Projekts in Teilprojekte erfolgen kann, da ein Vergleich von Teilprojekten in der Regel eine höhere Genauigkeit der Schätzung zur Folge hat.

[27] siehe 3.1.3 - Dokumentation als Grundlage der Aufwandsschätzung
[28] siehe 3.1.3 - Die Einflüsse auf den Aufwand

Zudem findet eine Beschreibung des Leistungsprofils meist über bestimmte Kenngrößen statt. Dabei stehen hier nicht die Bildung der Kenngrößen im Vordergrund, welche ohnehin auf formel- und zahlenorientierten Methoden beruhen, sondern der Vergleich dieser. So werden in der Regel Wertepaare gebildet, welche den Aufwand und den Umfang zurückliegender Entwicklungsprojekte widerspiegeln. Als gängige Anwendung werden hier exemplarisch die Function Points und deren Darstellung in einer Erfahrungskurve[29] aufgeführt, welche in Abbildung 3.7 dargestellt ist.[30]

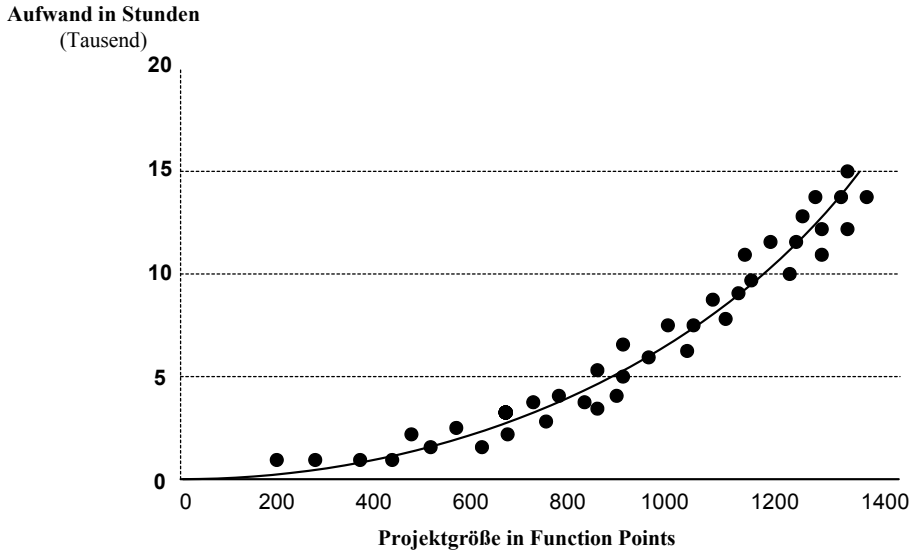


Abbildung 3.7: Vergleich anhand einer Erfahrungskurve

Jeder Punkt stellt ein abgeschlossenes Projekt dar. Die daraus resultierende Kurve kann mit Hilfe mathematischer Methoden der Regressionsanalyse gebildet werden. Somit lassen sich bei Kenntnis der voraussichtlichen Projektgröße zu erwartende Aufwände ableiten, wobei die tatsächlichen Werte ex post in die Erfahrungskurve übernommen werden.

[29] siehe Abschnitt 3.3.1 - Function-Point Verfahren
[30] in Anlehnung an [Hürten99] S. 53ff.

Die Relationenmethode

Im Gegensatz zur Analogiemethode, in welcher der Aufwand lediglich durch einen Vergleich gewisser Kriterien ermittelt wird, liegt der Relationenmethode ein formalisierter Ablauf zugrunde. Die Einflussfaktoren eines Softwareprojekts, welche zum Vergleich heranzuziehen sind, werden hier nach bestimmten *Indizes* gewertet. So wird das zu schätzende Objekt zunächst nach bestimmten Merkmalen untersucht, denen je nach Ausprägung bestimmte Indikatoren zugerechnet werden. Das Resultat bilden Indikatorenleisten als Ausgangspunkt für den Vergleich. Das folgende Beispiel soll die Anwendung der Relationenmethode verdeutlichen:

Den Faktoren Programmiersprache, Programmiererfahrung und Dateiorganisation lassen sich folgende Werte zuordnen:

Programmiersprache	**Programmiererfahrung**	**Dateiorganisation**
PL1=100	5Jahre=80	sequentiell=80
COBOL=120	3Jahre=100	indexsequentiell=120
Assembler=140	1Jahr=140	

Tabelle 3.1: Beispiel von Indizes der Relationenmethode

Die einzelnen Werte umfassen den Einfluss der einzelnen Faktoren auf den Aufwand. So wird ein Softwareprojekt, welches in der Programmiersprache Assembler in sequentieller Dateiorganisation realisiert werden soll und in welchem das Projektteam eine durchschnittliche Erfahrung von einem Jahr besitzt, folgendermaßen gewertet:

Programmiersprache Assembler	Aufschlag von 40 Punkten
Projekterfahrung von einem Jahr	Aufschlag von 40 Punkten
Sequentielle Dateiorganisation	Abschlag von 20 Punkten
Aufschlag/Abschlag bei Vergleich	+60 Punkte

3.2.3 Kennzahlenmethoden

Ähnlich den Vergleichsmethoden setzen die Kennzahlenmethoden ein systematisches Sammeln und Speichern projektspezifischer Daten voraus. Die aus einer Datenbasis ableitbaren projektspezifischen Messgrößen dienen hier jedoch nicht dem Vergleich von Projekten, sondern sind die Grundlage für Kennzahlen. Diese werden zur Bewertung von Schätzgrößen geplanter Entwicklungsprojekte herangezogen.

Die Prozentsatzmethode

Ausgangspunkt der Prozentsatzmethode ist die durchschnittliche Verteilung des Aufwands auf einzelne Phasen eines Softwareprojekts. Dazu wird der Aufwand einer Projektphase als Ausgangsgröße für die Verteilung des Gesamtaufwands herangezogen. Diese allgemeine Vorgehensweise ist in Abbildung 3.8 dargestellt:[31]

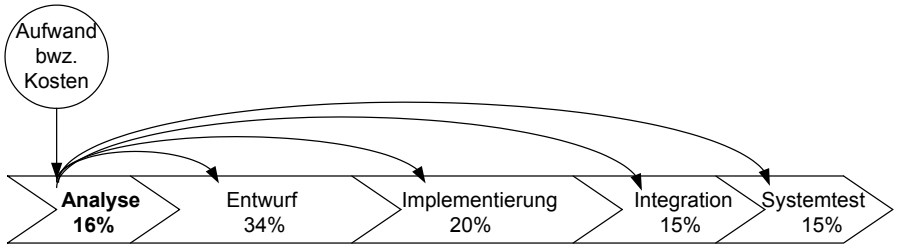


Abbildung 3.8: Prozentsatzmethode

Die Ermittlung des Gesamtaufwands eines Softwareprojekts kann auf zwei Arten erfolgen:

- Zum einen kann durch den Aufwand, welcher durch die Vollendung einer Projektphase bekannt ist, auf die anderen Phasen anhand der Erfahrungswerte vergangener Projekte geschlossen werden.
- Zum anderen kann eine genaue Aufwandsschätzung einer spezifischen Phase vorgenommen werden, woraufhin wiederum auf die restlichen Phasen geschlossen werden kann.

[31] in Anlehnung an [Burghardt01] S.97

Zu berücksichtigen ist bei dieser als schnellen und einfachen Alternative gewerteten Methode die konstante Verwendung des gleichen Phasenmodells der Softwareentwicklung.

Die Multiplikatormethode

Anstatt der Bestimmung des Aufwands werden in der Anwendung der Multiplikatormethode vorrangig die Projektkosten betrachtet. Einem Softwareprojekt werden nach Abschluss, im Verlauf einer Nachkalkulation, spezifische Kennzahlen in Form von *Kosten pro Leistungseinheit* zugerechnet. Die daher oft auch als „Aufwand-pro-Einheit" bezeichnete Methode ermöglicht somit eine genaue Aufspaltung der Gesamtkosten auf verschiedenste Teilbereiche der Softwareentwicklung. Für eine letztendlich durchzuführende Schätzung werden die Kennzahlen mit dem erwarteten Umfang einer abgrenzbaren Leistungseinheit multipliziert.

Die in diesem Umfeld angewandte *Wolverton-Methode* beinhaltet zusätzlich eine Berücksichtigung bezüglich des Softwaretyps und des Schwierigkeitsgrads der Software. Dadurch kristallisieren sich verschiedene Kategorien mit spezifischen Kennzahlen heraus, welche letztendlich die Grundlage zur Bestimmung der Gesamtkosten darstellen.[32]

3.2.4 Algorithmische Methoden

Die Algorithmischen Methoden basieren auf einer Formel oder einem Formelgebilde, dessen Struktur und Konstanten empirisch und meist mit mathematischen Methoden bestimmt worden sind. Grundlegend wird hier ein funktionaler Zusammenhang von Einflussparametern und Aufwand gesucht, woraufhin spätere Entwicklungsvorhaben anhand dieser Zusammenhänge geschätzt werden können. Diese Methoden gelten als aufwändig. Jedoch werden die Ergebnisse als zuverlässig eingestuft.

Methode der parametrischen Schätzgleichungen

Grundlage der Methode der parametrischen Gleichungen ist die Korrelationsanalyse, eine statistische Methode zur Feststellung beziehungsweise Quantifizierung

[32] vgl. [Burghardt01] S.95

der gegenseitigen Abhängigkeiten mehrerer Faktoren.[33] Mit deren Hilfe erfolgt eine Suche nach den Einflussfaktoren, welche den Entwicklungsaufwand maßgeblich beeinflussen. Dazu wird in der Regel eine Analyse einer Vielzahl von idealerweise weitgehend homogenen Projekten durchgeführt. Das Ergebnis bilden meist mittels der Regressionsanalyse gewonnene Funktionen. Diese Herangehensweise gleicht in diesem Zusammenhang dem bereits vorgestellten Ansatz der Function Points.[34]

Die Faktorenmethode

Die Faktorenmethode umfasst ein Wertesystem von Faktoren und Gewichtungszahlen, welche den Ergebnissen der Korrelationsanalyse ähnlich, den quantifizierbaren Einfluss bestimmter Parameter auf den Aufwand ausdrücken. Der Unterschied zur Methode der parametrischen Schätzgleichungen besteht jedoch darin, dass hier eine sowohl subjektive als auch objektive Bewertung der Ergebnisse vorgenommen wird, wodurch diese Methode auch unter der Bezeichnung *Gewichtungsmethode* Anwendung findet.

In Abbildung 3.9 sind beide hier vorgestellten Algorithmischen Methoden dargestellt.[35]

[33] vgl. [Litke96] S.33
[34] siehe Abschnitt 3.2.2 - Vergleichsmethoden
[35] in Anlehnung an [Burghardt01] S.86

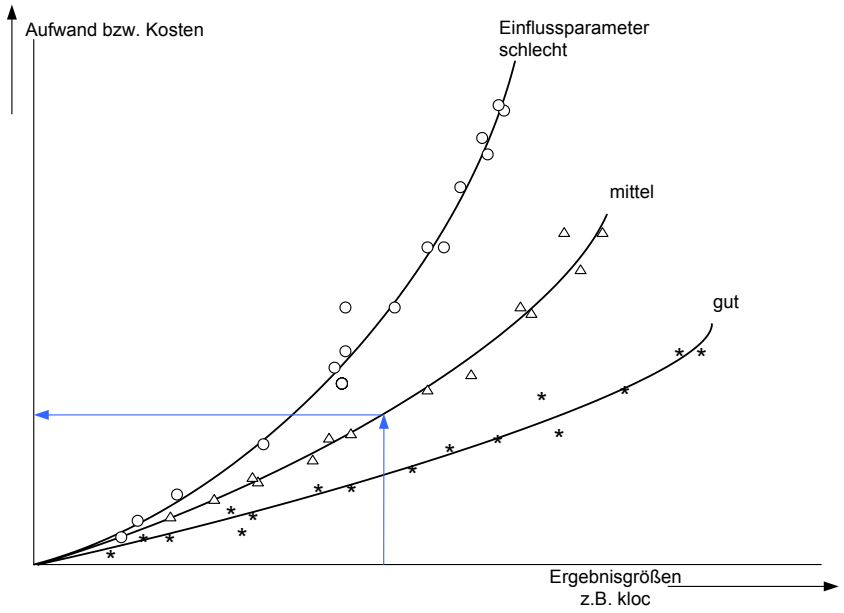


Abbildung 3.9: Überblick: Algorithmische Methoden

Die skizzierten Funktionsverläufe lassen sich zunächst von der Korrelationsanalyse der parametrischen Schätzgleichungen und einer anschließend vorgenommenen Regression herleiten oder beruhen auf einem ermittelten Wertungsschema der Faktorenmethode. Die Wichtung der Einflussparameter hinsichtlich der hier exemplarisch verwendeten Kategorisierungen „gut", „mittel" und „schlecht" fällt jedoch in den Bereich der Faktorenmethode.

3.3 Ausgewählte Aufwandsschätzverfahren

Ein Aufwandsschätzverfahren stellt eine konkrete Anwendung und Umsetzung einer oder mehrerer Aufwandsschätzmethoden dar. Es beinhaltet Vorgehensweisen und Handlungsrichtlinien für den praktischen Einsatz. Der Fokus der hier vorgestellten Verfahren liegt dabei auf der dem Thema der Arbeit betreffenden Relevanz und der Praxistauglichkeit. Dadurch rücken zwei Verfahren in den Vordergrund: das sich lange Zeit bewährte *Function Point Verfahren* sowie das kürz-

lich in seiner zweiten Version veröffentlichte *Constructive Cost Model*. Beide sollen im Anschluss vorgestellt werden. Im Sinne eines im späteren Verlauf der Arbeit zu erarbeitenden Lösungsansatzes und einer prototypischen Umsetzung werden die internen Abläufe und Arbeitsschritte der Verfahren erläutert.

3.3.1 Das Function Point Verfahren

Das Function Point Verfahren, welches 1979 von A.J. Albrecht entwickelt wurde, basiert auf der Faktoren- und Analogiemethode. Insbesondere die Darstellung über eine Erfahrungskurve stellt das Ergebnis dieses Verfahrens dar.[36] Seit 1979 bestehen Standardisierungsbemühungen durch die *International Function Point Users Group*[37] (IFPUG).

Wie bereits beschrieben, sind Function Points eine Alternative zur Quantifizierung der Projektgröße. Statt der üblichen Anzahl der Programmbefehle liegt der Schwerpunkt der Betrachtung hier auf den funktionalen Anforderungen aus Benutzersicht, also dem wahrgenommenen Leistungsumfang einer Anwendung. Function Points sind dadurch unabhängig von der zur Erstellung des Projekts genutzten Technologie. Durch die Art und Weise der Bildung der Function Points eignet sich das Verfahren sowohl für Neuentwicklungen als auch zur Erweiterung beziehungsweise Modifikation vorhandener Softwareprojekte innerhalb meist kommerzieller Systeme.

Zunächst wird im Anschluss die allgemeine Funktionsweise des Verfahrens dargelegt, wobei einzelne Arbeitsschritte näher erläutert werden. Dabei liegt der Fokus nicht allein auf der Bildung der Function Points, vielmehr soll das notwendige Vorgehen innerhalb des Verfahrens vorgestellt werden. Grundlage ist dabei der von der IFPUG veröffentliche Standard 4.1.[38]

Schritt 1: Festlegung des Zähltyps
Innerhalb des Verfahrens wird je nach der geplanten Entwicklung zwischen drei Zähltypen unterschieden:

[36] siehe Abschnitt 3.2.2 - Vergleichsmethoden / Abbildung 3.7
[37] http://www.ifpug.org/
[38] vgl. zu folgenden Ausführungen [Bundschuh00] S.188ff; [Hürten99] S.41ff.

- *Die Neuentwicklung:* Hierbei wird davon ausgegangen, dass ein Anwendungssystem vollständig neu erstellt wird. Gezählt werden die eingebrachte Funktionalität in Form von Function Points.
- *Ein Weiterentwicklungsprojekt*: Hierbei besteht die Möglichkeit, dass Funktionalität einer Software hinzugefügt, geändert oder entfernt wird.
- *Das Anwendungssystem*: Innerhalb dessen wird der Zuwachs oder die Abnahme der Funktionalität beispielsweise durch ein Weiterentwicklungsprojekt berücksichtigt. Auch beinhaltet ein Anwendungssystem eine Schätzung am Anfang und Ende des Projekts, um einen Vergleich von geplanter und realisierter Funktionalität zu erhalten. Durch diese Maßnahmen fließt am Ende einer Entwicklung stets die korrekte Anzahl der Function Points in die Erfahrungsdatenbank ein.

Schritt 2: Festlegung des Umfangs der Zählung und der Systemgrenzen
Der Umfang einer Zählung von Function Points kann ein einziges Anwendungssystem, aber auch mehrere umfassen. Zu beachten ist dabei, dass die technische Betrachtung des zu entwickelnden Systems vernachlässigt wird und lediglich funktionale Aspekte aus Benutzersicht in den Vordergrund treten. Die Festlegung der Systemgrenzen ist insofern von Bedeutung, als dass eine korrekte Zuordnung von Function Points zu systeminternen und -externen Funktionalitäten gewährleistet sein muss. Die dabei aus Benutzersicht erkennbaren Zusammenhänge werden über verschiedene Modellierungsmethoden dargelegt.

Schritt 3: Die Zählung der ungewichteten Function Points
Bevor die einzelnen Abläufe zur Zählung der ungewichteten Function Points (UFP) erläutert werden, soll zunächst Abbildung 3.10 einen Überblick verschaffen und eine Hilfe zur Orientierung über den recht komplexen Arbeitsschritt bieten:

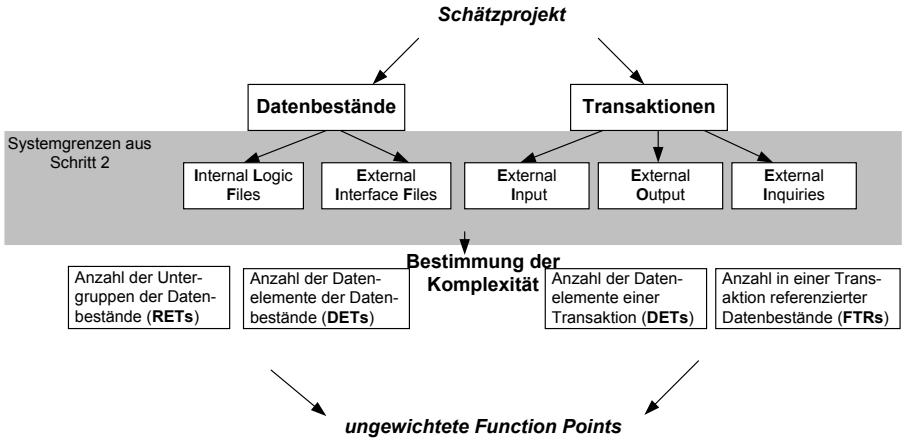


Abbildung 3.10: Die Bestimmung der ungewichteten Function Points (UFP)

Zunächst erfolgt eine Unterteilung nach *fünf Funktionstypen*. Diese werden wiederum Datenbeständen (*„Data Function Points"*) oder Transaktionen (*„Transaction Function Points"*) zugerechnet. Die folgende Tabelle umfasst diese Zuordnung und beinhaltet die einzelnen Funktionstypen:

Data Function Points	**ILF (Internal Logic Files)**: Hierbei handelt es sich um interne Datenbestände, welche innerhalb der festgelegten Systemgrenzen von der Anwendung gepflegt werden. **EIF (External Interface Files)**: Dies sind Datenbestände, welche außerhalb der eigenen Systemgrenzen von anderen Anwendungen gepflegt werden, jedoch von der eigenen Anwendung referenziert werden.
Transaction Function Points	**EI (External Inputs)**: Diese externen Eingabedaten umfassen Prozesse, welche von außerhalb des eigenen Systems auf die Internal Logic Files (ILF) zugreifen. Dies betrifft zum Beispiel das Anlegen eines neuen Datenbestands über eine Dialog-geführte Benutzeroberfläche. **EO (External Output)**: Die externen Ausgabedaten betreffen Prozesse, welche Daten über die eigenen Systemgrenzen hinaus weiterreichen. Durch die vorrangige Betrachtung von Benutzerfunktionen dienen die externen Ausgabedaten vorrangig dazu, dem Benutzer Informationen zu präsentieren. Als Beispiel seien hier Daten genannt, welche Online zur Verfügung gestellt werden. **EQ (External Inquiries)**: Hierbei handelt es sich um externe Abfragen, welche durch Ein- und Ausgaben auf interne Datenbestände, also Internal Logic Files (ILF) zugreifen, diese jedoch unverändert belassen. Auch die externen Abfragen dienen vorrangig der Informationspräsentation. Dazu zählen beispielsweise Daten, welche aus unterschiedlichen Quellen dem Benutzer Informationen präsentieren, ohne die Möglichkeit zur Einflussnahme auf die Daten seitens des Nutzers zu bieten.

Tabelle 3.2: Funktionstypen nach IFPUG 4.1

Zunächst sollen die *Data Function Points* näher erläutert werden. Der in Tabelle 3.2 verwendete Begriff des *Pflegens* umfasst das Anlegen, Ändern, Löschen und den Zugriff auf die Daten. Die Betrachtung der Internal Logic Files und External Interface Files erfolgt wiederum ausschließlich aus Nutzersicht sowie nach der Anzahl der Untergruppen und der Anzahl der Datenelemente:

- Die Anzahl der Untergruppen, hier *„Record Element Type"* (RET) genannt, beschreibt eine vom Nutzer erkennbare Untergruppe von Datenelementen.
- Die Anzahl der Datenelementtypen, der sogenannte *„Data Element Type"* (DET), umfasst alle Datenelemente in einer Geschäftsentität, einer allge-

meinen, vom Rest des Systems abgrenzbaren Zusammenfassung von Datenelementtypen.[39]

Abbildung 3.11 soll die Bestimmung der Untergruppen und Datenelementtypen anhand eines Beispiels verdeutlichen:

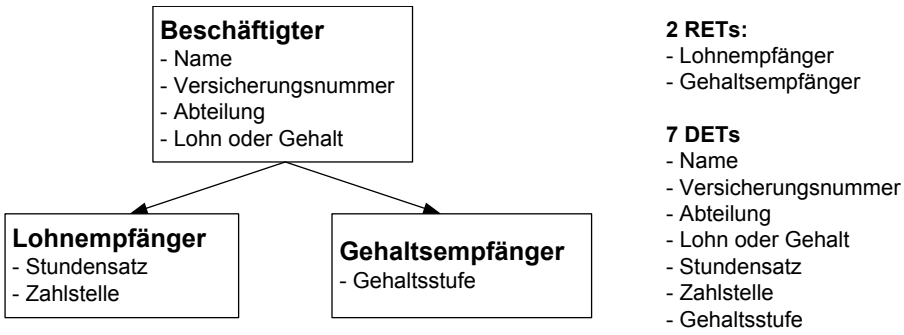


Abbildung 3.11: Die Bestimmung von DETs und RETs am Beispiel

Die Untergruppen der *Benutzergruppe* sind, soweit dies die funktionale Nutzersicht betrifft, *Lohnempfänger* und *Gehaltsempfänger*. Daher gehen hier 2 RETs in die Zählung ein. Die Anzahl der Datenelementtypen ergibt sich als Summe der Datenelemente der hier abgegrenzten Geschäftsentität.

Nachdem den *Internal Logic Files* (ILF) und *External Interface Files* (EIF) die *Record Element Types* (RETs) und *Data Element Types* (DETS) zugeordnet sind, wird deren Komplexität mit Hilfe einer Matrix bestimmt. Dies geschieht mittels der Ausprägungen einfach (low), mittel (average) und hoch (high), wie in Tabelle 3.3 dargestellt ist:

RETs / DETs	**1 - 19 DETs**	**20 - 50 DETs**	**>50 DETs**
1 RETs	low	low	average
2 - 5 RETs	low	average	high
>5 RETs	average	high	high

Tabelle 3.3: Bestimmung der Komplexität der Datenbestände

[39] vgl. [Stahlknecht99] S.187

Dieser Komplexitätsgrad dient wiederum der Zuordnung der Anzahl von Function Points zu den Internal Logic Files (ILF) und den External Interface Files (EIF), wie in Tabelle 3.4 aufgeführt:

	ungewichtete Function Points	
Komplexitätsgrad	**ILF**	**EIF**
low	7	5
average	10	7
high	15	10

Tabelle 3.4: ungewichtete Function Points der Datenbestände

Diese Vorgehensweise soll wiederum anhand des in Abbildung 3.11 skizzierten Beispiels verdeutlicht werden. Der Datenbestand *Beschäftigter* hat gemäß der Einteilung in Tabelle 3.3 den Komplexitätsgrad „low". Nun wird anhand der Systemgrenzen eine Zuordnung gemäß der Internal Logic Files oder der External Interface Files vorgenommen. Dies verdeutlicht wiederum den Einfluss auf die Aufwandsschätzung von der in Schritt 2 vorgenommene Abgrenzung des zu entwickelnden Softwaresystems.

Die Verfahrensweise bei der Bestimmung der ungewichteten Function Points der Transaktionen verläuft analog zu den Datenbeständen. Die Komplexität der Transaktionen hängt hier jedoch von den Faktoren *„File Type Record"* und *„Data Element Type"* ab:

- Die Anzahl der referenzierten Datenbestände, also der externen oder internen Datenbestände, welche für die Abarbeitung der Transaktion benötigt werden, wird als ,,*File Type Record"* (FTR) bezeichnet.
- Die Anzahl der Datenbestände, die sogenannten *„Data Element Type"* (DET), werden ähnlich der Datenbestände der Data Function Points gezählt. Betroffen sind dabei alle aus Nutzersicht erkennbaren Felder, welche von einer Transaktion betroffen sind.

Auch hier werden den einzelnen External Inputs (EI), External Outputs (EO) und External Inquiries (EQ) anhand von Komplexitätsgraden die ungewichteten

Function Points zugerechnet. Das Ergebnis aus Schritt 3 ist somit die Summe der *ungewichteten Function Points* (UFP) aller Datenbestände und Transaktionen.

Schritt 4: Die Ermittlung der Einflussfaktoren
In diesem Schritt wird unter Berücksichtigung von 14 allgemeinen Einflussfaktoren der sogenannte *„Value adjustment factor"* ermittelt, welcher die Grundlage zur Berechnung der gewichteten Function Points darstellt. Diese Einflussfaktoren sind:

1. Datenkommunikation
2. Verteilte Verarbeitung
3. Leistungsfähigkeit
4. Begrenzte Kapazität
5. Transaktionsrate
6. Interaktive Dateneingabe
7. Benutzerfreundlichkeit
8. Interaktive Änderung
9. Komplexe Verarbeitung
10. Wiederverwendbarkeit
11. Installationshilfen
12. Betriebshilfen
13. Mehrfachinstallation
14. Änderungsfreundlichkeit

Anhand einer Skala von 0 für keinen bis 5 für starken Einfluss, wird dieser, hier *„Degree of influence"* (DI) genannt, für jedes dieser Merkmale bewertet. Die Summe der bewerteten Einflüsse ist der Gesamteinflussfaktor, der *„Total Degree of Influence"* (TDI). Dieser wird anschließend zur Berechnung des „Value adjustment factor" herangezogen:

$$VAF = (TDI * 0.01) + 0.65$$

Das Ergebnis hat eine Wertigkeit zwischen 0,65 und 1,35. Für allgemein übliche Softwareprojekte beträgt der „Value adjustment factor" 1.

Schritt 5: Die Berechnung der gewichteten Function Points
In diesem Schritt erfolgt die Berücksichtigung des in Schritt 1 festgelegten Zähltyps. Dieser bestimmt die im weiteren Verlauf herangezogene Formel zur Berechnung. Im Folgenden sollen diese Formeln kurz exemplarisch an einem Weiterentwicklungsprojekt vorgestellt werden:

$$EFP = [(ADD + CHGA + CFP) * VAFA] + (DEL * VAFB)$$

wobei gilt:

- *EFP* (Enhancement Function Points): Function Points eines Weiterentwicklungsprojekts,
- *ADD* (Added Function Points): Hinzugefügte Function Points durch neue Funktionalitäten,
- *CHGA* (Changed After): ungewichtete Function Points. Diese entstehen durch die *Änderung* bestehender Funcktionalitäten. Gezählt werden dabei die Function Points nach der Änderung.
- *CFP* (Conversion Function Points): Function Points aus einer Bestandsübernahme von Altdaten,
- *VAFA* (VAF after): Einflussfaktor, welcher nach der Erweiterung ermittelt wurde.
- *DEL* (Deletd Function Points): ungewichtete Function Points, welche durch das *Löschen* bestehender Funktionalitäten entstehen.
- *VAFB* (VAF before): Der Einflussfaktor vor der durchgeführten Erweiterung.

Diese hier ermittelten Function Points bilden das Ergebnis des Verfahrens. Dabei kann den Function Points mit Hilfe einer Erfahrungskurve ein Aufwand zugerechnet werden.[40] Nach Abschluss des Projekts und mit Kenntnis des Wertepaars Function Point und Aufwand können die neu gewonnenen Erkenntnisse wiederum der Erfahrungskurve hinzugefügt werden.

Schritt 6: Die Dokumentation der Zählung

Die Bedeutung der Dokumentation innerhalb des Function Point Verfahrens ist insbesondere von Interesse, da speziell hier eine Vergleichbarkeit verschiedener Schätzungen gegeben sein muss. Eine Erfahrungskurve bildet sich anhand von

[40] siehe Abschnitt 3.2.2 - Vergleichsmethoden / Abbildung 3.7

Projekten heraus, welche ähnlich aufgebaut sind. Ansonsten wären die Schwankungen der einzelnen Schätzungen zu stark, als dass verlässliche Wertepaare gefunden werden könnten. Auch kann die Bestimmung der Function Points bei Projektende deutlich effizienter und zuverlässiger gestaltet werden, wenn eine korrekte Dokumentation der ersten Zählung vorliegt. Angaben, welche bezüglich einer ordnungsgemäßen Dokumentation empfohlen werden, sind unter anderem:

- Zähltyp,
- Systemgrenzen,
- Datenbestände und Transaktionen,
- Bewertung der 14 Einflussfaktoren,
- die ungewichteten und gewichteten Function Points sowie
- alle eventuell getroffenen Annahmen, die Einfluss auf die Zählung der Function Point haben könnten.

In der Praxis werden in der Regel feste Formulare zur Erfassung der Angaben verwendet. Auch existieren erste Softwareprodukte, welche dies in gewissem Umfang erlauben.

Bewertung des Verfahrens

Das Function Point Verfahren bietet sich insbesondere für eine längerfristige Verfolgung einer Aufwandsschätzung an. Dies ist vorrangig in der Anzahl der Projekte begründet, die nach einer gewissen Zeit die Erfahrungskurve bilden.[41] Auch sollten die Projekte nicht zu stark in den Anforderungen voneinander abweichen, um verlässliche Vergleiche zu ermöglichen. Die Anwendung einer Erfahrungskurve bietet entscheidende Vorteile gegenüber anderen Verfahren. Nicht erfassbare Kostentreiber, wie die eingangs des Kapitels erwähnten *„Humanen Faktoren"* schlagen sich durch die ex post Betrachtung in der Kurve nieder.[42]

[41] i.d.R. sind das 7 Projekte vgl. [Balzert96] S.77

[42] siehe Abschnitt 3.1.3 - Die Einflüsse auf den Aufwand

3.3.2 Weiterführende Varianten des Function Points Verfahrens

Seit dem Bestehen des Function Point Verfahrens gab es zahlreiche Verbesserungen und Weiterentwicklungen. Auf einige Wichtige soll im Folgenden kurz eingegangen werden.

Das Data Point Verfahren

Eine Aufwandsschätzung auf Basis von Data Points stellt ein eigenständiges Schätzverfahren dar.[43] Der zu erwartende Umfang einer zu entwickelnden Software wird anhand eines Datenmodells und der zukünftigen Benutzeroberfläche bestimmt. Für ein Datenmodell werden einzelne Informationsobjekte ermittelt und deren Datenelemente, Attribute und Relationen gemessen. Die Werte der Benutzeroberfläche werden durch die Anzahl der Bildschirmmasken, Berichte und Systemnachrichten bestimmt. Zur Durchführung des Verfahrens werden als Einfluss auf den zu erwartenden Aufwand 8 Qualitätsfaktoren und 10 Projektbedingungen herangezogen.

Das Object Point Verfahren

Dieses Verfahren entstand im Zuge der objektorientierten Softwareentwicklung.[44] Den Ausgangspunkt bildet die Bestimmung der Object Points durch das Zählen von Objekttypen anhand der Bildschirmmasken, Druckausgaben, Informationsobjekten, Datentransfers und Verarbeitungsfunktionen. Anstatt der im Function Point Verfahren genutzten 14, berücksichtigt dieses Verfahren 11 spezifische Einflussfaktoren. Inzwischen bestehen zahlreiche Bemühungen dieses Verfahren weiterzuentwickeln. Jedoch hat sich bislang kein allgemein akzeptierter Standard herausgebildet, so dass hier nur auf die ursprüngliche von der Software AG entwickelte Verfahrensweise eingegangen wurde.

Das MarkII Verfahren

Das MarkII Verfahren baut direkt auf dem Function Point Verfahren auf und ergänzt dieses durch technik- und umgebungsorientierte Parameter sowie durch veränderte Berechnungsregeln. Zusätzlich zu den im Function Point Verfahren

[43] vgl. [Bundschuh00] S.210

[44] vgl. [Bundschuh00] S. 211f. Die Ausführungen sind an das von Maria Oriola 1990 entwickelte Verfahren angelehnt

verwendeten 14 Einflussfaktoren berücksichtigt das MarkII Verfahren 7 Komplexitätsfaktoren.[45]

3.3.3 Das Constructive-Cost-Model (COCOMO)

Das 1981 von Barry W. Boehm vorgestellte Constructive Cost Model basiert auf der Faktorenmethode und der Methode der parametrischen Gleichungen. Es lässt sich beschreiben als eine Kombination von Gleichungen, statistischen Modellen und Schätzungen von Parameterwerten.[46]

Zentrale Ausgangsgröße des Verfahrens bildet die voraussichtliche Anzahl der Programmzeilen, hier genauer *Kilo Delivered Source Instructions* (KDSI) genannt, wozu selbst erstellte, elementare Anweisungen zählen. Der dabei vernachlässigte Aspekt der verwendeten Programmiersprache soll implizit durch die *15 Einflussparameter* berücksichtigt werden. Mittels dieser sogenannten *Kostentreiber* und der durch die KDSI vorzugebenden Produktgröße werden der Aufwand, die Kosten und die Zeit ermittelt.

Das COCOMO Verfahren orientiert sich an mehreren Grundgedanken. Diese sind unter anderem:

- Die Anforderungen an das zu schätzende Projekt werden als konstante Eingangsgrößen betrachtet. Dies liegt vor allem in dem zugrundeliegenden Phasenmodell[47] der Softwareentwicklung begründet, welches davon ausgeht, dass nach der Anforderungsanalyse alle Anforderungen an das Softwareprojekt bekannt sind und feststehen.
- Die Ergebnisse der Schätzung des Aufwands in Personenmonaten[48] gehen von 152 Arbeitsstunden aus. Dieser Wert beruht auf empirischen Erfahrungswerten und kann mittels fixer Umrechnungen in Zeit und Kosten umgewandelt werden.

[45] vgl. [Hürten99] S.37ff; www.uksma.co.uk
[46] vgl. zu folgenden Ausführungen [Heinrich97] S.217, [Litke96] S.142ff.
[47] sog. „Wasserfallmodell"
[48] Angabe in [PM] bzw. [SM] für „staff-months"

- Die Schätzungen klammern gewisse Aufwände aus, wie beispielsweise die Administration oder Umstellung eines Softwaresystems.

Abhängig vom Detaillierungsgrad der durchgeführten Schätzung unterscheidet COCOMO drei Modellvarianten. Diese sind gemäß der chronologischen Reihenfolge ihres Einsatzzeitpunkts:

- **Das Basic Model** betrachtet das Schätzprojekt als Ganzes ohne strukturelle und zeitliche Unterteilung. Der Entwicklungsaufwand wird mit Hilfe einer Grundgleichung berechnet. Als einziger Einflussfaktor wird die Projektgröße in Form von KDSI berücksichtigt. Der Einsatz des Basismodells wird für schnell und frühzeitig durchzuführende Softwareprojekte empfohlen.

- **Das Intermediate Model** berücksichtigt in seiner Anwendung die bereits genannten 15 Einflussfaktoren beziehungsweise Kostentreiber. Jeder dieser Kostentreiber, auf dessen Basis das Projekt gewertet wird, bestimmt einen Multiplikator, welcher wiederum den Einfluss des Kostentreibers auf den Gesamtaufwand berücksichtigt. Das Softwareprojekt wird jedoch auch hier als Ganzes ohne phasenorientierte Differenzierung betrachtet.

- **Das Detail Model** nimmt eine hierarchische Untergliederung des Softwareprojekts in System-, Subsystem-, und Modulebene vor. Der Einfluss der Kostentreiber auf unterschiedliche Komponenten wird hierbei berücksichtigt und es erfolgt eine phasenorientierte Untergliederung.

Des Weiteren umfasst COCOMO drei Entwicklungsmodi beziehungsweise Entwicklungsklassen. Diese bestimmen den sogenannten *„Software-Projekttyp"* und dienen der Zuordnung hinsichtlich der Entwicklungsumgebung.

- **Der Organic mode** umfasst alle als einfach eingestufte Softwareprojekte. Diese sind gekennzeichnet durch eine stabile Entwicklungsumgebung, minimale Innovationen, geringe und stabile Schnittstellen. In der Regel betrifft dies kleine Projekte, welche ohne Termindruck erstellt werden.

- **Der Semidetached mode** ist gekennzeichnet durch Projekte, deren Komplexität zwischen Organic- und Embedded Mode liegt.

- **Der Embedded mode** beschreibt komplexe Softwareprojekte. Dies umfasst Projekte mit engem Termin- und Kostenrahmen, ständigen Innovationen, inflexiblen Schnittstellen der Software sowie hohen Anforderungen hinsichtlich der Verfügbarkeit.

Die dem COCOMO Verfahren zugrundeliegenden mathematischen Konstanten stammen aus den Erfahrungen von 63 Softwareprojekten. Die dabei von Boehm erreichte Konvergenz wurde lange Zeit als gut bis sehr gut eingestuft. Jedoch konnten in der Vergangenheit Daten und Erfahrungen von 1981 nicht mehr den Ansprüchen moderner Softwareentwicklung genügen, so dass es zu Abwandlungen und Weiterentwicklungen des Verfahrens kam. Den Hauptkritikpunkt stellt die Produktgröße in Form von KDSI dar, da sie als einzige Korrelation zum Projektaufwand gesehen wird. Auch müssen die KDSI selbst vor Projektbeginn geschätzt werden, was wiederum aufgrund spezifischer Programmierstile und -sprachen in Frage gezogen werden kann. Aufgrund dieser Kritik beschränken sich die zurückliegenden Ausführungen auf die Grundideen von COCOMO. Ausführlicher soll jedoch im Anschluss die Vorstellung des COCOMOII Verfahrens erfolgen.

3.3.4 Das COCOMOII Verfahren

Das vorläufige Ergebnis der erwähnten Weiterentwicklungen stellt das 1995, ebenfalls von Barry W. Boehm, veröffentlichte COCOMOII Verfahren dar. Auch dieses basiert auf der Faktorenmethode und der Methode der parametrischen Gleichungen, berücksichtigt jedoch moderne Ansätze der Softwareentwicklung.[49]

Auch hier werden drei unterschiedliche Modellvarianten unterschieden. Dabei liegt der Fokus vorrangig auf den verschiedenen Zeitpunkten der Softwareentwicklung, so dass der Detaillierungsgrad der Schätzung implizit in den Submodellen enthalten ist.[50] Diese sind gemäß der chronologischen Reihenfolge ihres Einsatzes:

- Das **Application Composition Model**, welches für den Einsatz in frühen Projektphasen konzipiert ist, berücksichtigt Softwareprojekte, welche mit Hilfe moderner Entwicklungswerkzeuge und -verfahren erstellt werden

[49] vgl. zu folgenden Ausführungen [Boehm00]
[50] siehe Abschnitt 3.3.3 - Das Constructive-Cost-Model (COCOMO)

sollen. Ausgangswert für die Projektgröße sind hier sogenannte Application Points, eine Form der Function Points.

- Das **Early Design Model** wurde ebenfalls für frühe Projektphasen entwickelt, dient aber vorrangig der Evaluation von Architekturen und Entwicklungsstrategien bei unvollständiger Kenntnis des zu schätzenden Projekts. Dieses Submodell beinhaltet bereits *sieben Kostentreiber*, welche in die Berechnung des Aufwands einfließen. Zur Bestimmung der Projektgröße werden hier ähnlich dem ursprünglichen COCOMO Verfahren die Anzahl der Programmbefehle genutzt. Jedoch besteht zudem die Möglichkeit, die Projektgröße anhand *ungewichteter Function Points* zu beschreiben.
- Das **Post Architecture Model** wird als detailliertes Schätzmodell nach der Entwurfsphase verwendet, was bedeutet, dass genaue Informationen über das zu schätzende Projekt verfügbar sein müssen. Zur Anwendung kommen hier 22 *Kostentreiber*. Innerhalb dieses Modells können die Anzahl der Programmbefehle oder ungewichtete Function Points zur Bestimmung der Projektgröße herangezogen werden.

Hinsichtlich der folgenden Ausführungen soll wiederum eine Einschränkung vorgenommen werden. Lediglich das Post Architecture Model wird als angemessen für die eigenen Bedürfnisse kalibrierbar gewertet und findet daher auch meist als einziges Anwendung. Die im Anschluss getroffenen Aussagen gelten daher ausschließlich für das Post Architecture Model des COCOMOII Verfahrens.

Die Ausgangsgleichung:
Die Ausgangsgleichung basiert auf der Anzahl der Programmzeilen, hier *DSI* für *„Delivered Source Instructions"* genannt. Aufgrund der ex anten Unkenntnis des Maßes und da dieses daher wiederum geschätzt werden müsste, können auch ungewichtete Function Points herangezogen werden. Diese werden mittels einer Umrechnungstabelle in DSI umgewandelt. So entspricht beispielsweise ein ungewichteter Function Point in der Programmiersprache „C++" genau 55 DSI.[51] Zunächst soll die Ausgangsgleichung dargestellt werden, woraufhin im Anschluss eine Erläuterung der einzelnen Bestandteile erfolgt:

[51] vgl. [Boehm00] S.20

$$PM_{NS} = A * DSI^E * \prod_{i=1}^{17} EM_i$$

$$wobei : E = B + 0.01 * \sum_{j=1}^{5} SF_j$$

Das angestrebte Ergebnis der Gleichung ist der voraussichtliche Aufwand in Form von Personenmonaten [PM]. Die Werte von $A, B; EM_1, ..., EM_{17}$ sowie $SF_1, ..., SF_5$ sind als gegebene Größen anzusehen, welche in der frei einsehbaren „COCOMOII database" verfügbar sind.[52] A und B im Speziellen sind Konstanten, welche vom zugrundeliegenden Modell abhängen. In dem hier dargestellten *Post Architecture Model* ist:

$$A = 2.94 \text{ und } B = 0.91.$$

Die Werte von EM und SF stellen die bereits eingangs erwähnten Kostentreiber dar:

- EM steht dabei für „*effort multipliers*", also Leistungswerten oder Parametern, welche anhand der Einteilung von Ausprägungen der Kostentreiber multipliziert werden.
- SF bezeichnet „*exponential scale factors*", also gewisse Skalenmaße, welche exponentiell verrechnet werden.

Beide sollen im Folgenden genauer erläutert werden.

Die Effort Multiplier:
Diese Parameter sollen positive sowie negative Einflüsse auf den zu schätzenden Aufwand berücksichtigen. Dazu werden im Post Architectue Model *17* Kostentreiber gemäß eine Klasseneinteilung von „very low" bis „extra high" bewertet. In der weiteren Berechnung werden die im Anschluss des Absatzes vorgestellten *scale factors* mit einbezogen, so dass der Wert der eingangs erwähnten *22* Kostentreiber entsteht.

[52] http://sunset.usc.edu/research/COCOMOII/

Zunächst soll jedoch mit Hilfe von Tabelle 3.5 die Ermittlung der *effort multipliers* vorgestellt werden:

Post Architectur Model

Driver	Bedeutung	Symbol	VL	L	N	H	VH	XH
RELY	Required Software Reliability	EM_1	0.82	0.92	1.00	1.10	1.26	
DAT	Data Base Size	EM_2		0.90	1.00	1.14	1.28	
CPLX	Product Complexity	EM_3	0.73	0.87	1.00	1.17	1.34	1.74
RUSE	Required Reusability	EM_4		0.95	1.00	1.07	1.15	1.24
DOCU	Documentation match to LC needs	EM_5	0.81	0.91	1.00	1.11	1.23	
TIME	Execution Time Constraint	EM_6			1.00	1.11	1.29	1.63
STOR	Main Storage	EM_7			1.00	1.05	1.17	1.46
PVOL	Platform Volatility	EM_8		0.87	1.00	1.15	1.30	
ACAP	Analyst Capability	EM_9	1.42	1.19	1.00	0.85	0.71	
PCAP	Programmer Capability	EM_{10}	1.34	1.15	1.00	0.88	0.76	
PCON	Personal Continuity	EM_{11}	1.29	1.12	1.00	0.90	0.81	
APEX	Application Experience	EM_{12}	1.22	1.10	1.00	0.88	0.81	
PLEX	Platform Experience	EM_{13}	1.19	1.09	1.00	0.91	0.85	
LTEX	Language and Tool Experience	EM_{14}	1.20	1.09	1.00	0.91	0.84	
TOOL	Use of Software Tools	EM_{15}	1.17	1.09	1.00	0.90	0.78	
SITE	Multisite Development	EM_{16}	1.22	1.09	1.00	0.93	0.86	0.80
SCED	Required Development Schedule	EM_{17}	1.43	1.14	1.00	1.00	1.00	

Tabelle 3.5: Die 17 Kostentreiber und die zugehörigen effort multiplier

Wie anhand der Grundformel zu erkennen ist, hat das Ergebnis dieses Arbeitsschritts einen erheblichen Einfluss auf den Gesamtaufwand. Zur korrekten Klassifizierung der einzelnen Kostentreiber existieren zahlreiche Hilfestellungen. Die aktuellen Werte der *effort multipliers* sind in der „COCOMOII database"[53] einsehbar.

Die Skalenfaktoren:
In der Grundgleichung wurde der Parameter E vorgestellt, welcher der Berücksichtigung bestimmter Skaleneffekte dient. Dies bedeutet, dass nun zwischen dem Aufwand eines Softwareprojekts und der Projektgröße mehr als ein einfacher linearer Zusammenhang betrachtet wird. Skaleneffekte[54] umfassen Größenvorteile oder Größennachteile aufgrund der steigenden Größe eines Softwareprojekts. Größenvorteile werden zum Beispiel erreicht, indem ein Team von Softwareentwicklern über einen gewissen Zeitraum Koordinations- und Kommunikationswege

[53] http://sunset.usc.edu/research/COCOMOII/
[54] siehe Abschnitt 4.2.2 - Weiterführende Aspekte: economies of scale

aufbaut und somit die Produktivität steigt.[55] Größennachteile entstehen, indem der Koordinations- und Kommunikationsaufwand aufgrund steigender Projektgröße derart zunimmt, dass die Produktivität sinkt.[56] In Tabelle 3.6 sind die zu verwendenten Skalenfaktoren dargestellt:

scale factors		Very Low	Low	Nominal	High	Very High	Extra High
PREC	Precedentness	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	generally familiar	largely familiar	thoroughly familiar
SF_j		6.20	4.96	3.72	2.48	1.24	0.00
PLEX	Development Flexibility	rigorous	occasional relaxation	some relaxation	general conformity	some conformity	general goals
SF_j		5.07	4.05	3.04	2.03	1.01	0.00
RESL	Architecture/ risk resolution	little (20%)	some (40%)	often (60%)	generally (75%)	mostly (90%)	full (100%)
SF_j		7.07	5.65	4.24	2.83	1.41	0.00
TEAM	Team Cohesion	very difficult interactions	some difficult interactions	basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions
SF_j		5.48	4.38	3.29	2.19	1.10	0.00
	Process maturity						
SF_j		7.80	6.24	4.68	3.12	1.56	0.00

Tabelle 3.6: COCOMOII Skalenfaktoren

Auch hier bestehen wiederum zahlreiche Richtlinien und Hilfsmittel, welche die Kategorisierung erleichtern. Der Sinn in der Anwendung der fünf Skalenfaktoren liegt darin begründet, dass gewisse Rahmenbedingungen einen teilweise nicht unerheblichen Einfluss auf den Entwicklungsaufwand haben.

Das COCOMO Reuse Model

Weiterhin soll hier auf die Wiederverwendung von Programmcode eingegangen werden, da dieser Punkt insbesondere im Zusammenhang mit den innnerhalb der Arbeit vorgestellten Systemfamilien eine wichtige Rolle spielt. Den Ausgangspunkt stellen wiederum die Anzahl der Programmzeilen in *DSI* dar. Dazu werden in diesem Modell die wieder verwendbaren Bestandteile in Programmzeilen umgerechnet. Zunächst sollen auch hier die Ausgangsgleichungen dargelegt werden:

[55] [Boehm00] betrachtet vorrangig die Produktivität. Economies of scale betrachten jedoch die Kosten bei steigender Kapazität, so dass Skalenvorteile i.e.S. erst durch die Umrechnung der Produktivitätskennzahlen zu Kosten zu betrachten sind.

[56] siehe Abschnitt 3.1.2 - Umfeld der Aufwandsschätzung / Abbildung 3.2: Brooksches Gesetz

$$DSI = adaptierteDSI * \left(1 - \frac{AT}{100}\right) * AAM$$

wobei die Grundgleichung:

$$AAF = (0.4 * DM) + (0.3 * CM) + (0.3 * IM)$$

herangezogen wird.

Ist die wertmäßige Ausprägung der AAF kleiner oder gleich 50 berechnet sich AAM wie folgt:

$$AAM = \frac{[AA + AAF * (1 + (0.02 * SU * UNFM))]}{100}$$

ansonsten:

$$AAM = \frac{[AA + AAF + (SU * UNFM)]}{100}$$

Im Folgenden sollen die einzelnen Bestandteile kurz erklärt werden.

AT: Automated Translation ist der prozentmäßige Anteil an Code, welcher automatisiert wieder verwendet werden kann. Dies bedeutet, dass der Code ohne jegliche Anpassung an das neu zu entwickelnde System übernommen wird.

AAM: Adaption Adjustment Modifier betrifft den Code welcher erst durch zusätzlichen Aufwand beziehungsweise Modifikationen wieder verwendet werden kann. Dazu zählen:

- DM: *Design Modifcation* beschreibt den prozentmäßigen Anteil des in das Projekt zu übernehmenden Software-Designs.
- CM: *Code Modifcation* umfasst den prozentmäßigen Anteil des Programmcodes, welcher wieder verwendet werden soll.
- IM: *Integration Modifcation* drückt den prozentmäßigen Anteil des Aufwands aus, welcher bei der Integration wiederverwendbarer Systembestandteile in ein neues System entsteht.

AA: Assessment and Assimilation umfasst den prozentmäßigen Aufwand, welcher durch die notwendige Beurteilung und Anpassung der wieder zu verwendenden Bestandteile entsteht.

SU: Software Understanding beinhaltet die Kriterien Struktur (structure) und Verständlichkeit der Applikation (application clarity) sowie den Grad in welchem Umfang Informationen aus der Software selbst gewonnen werden können.[57] Diese werden wiederum klassifiziert und liefern einen Prozentwert zwischen 10 und 50.

UNFM: Programmer´s unfamilarity beschreibt den Kenntnisstand des Entwicklers im Umgang mit der wieder zu verwendenden Software.

Bewertung COCOMOII

Das relativ neue COCOMOII Verfahren weist insbesondere Vorteile in der Anpassbarkeit hinsichtlich eines spezifischen Softwareprojekts auf. Neben den über Parameter bewertbaren Kostentreibern werden auch Aspekte, welche über die Softwareentwicklung hinausgehen, in Form von Skalenfaktoren berücksichtigt. Zudem können in Bezug auf die hohe Granularität des Post Architecture Models relativ genaue Schätzergebnisse erwartet werden. Aufgrund der empirischen Absicherung der dem Verfahren zugrundeliegenden Gleichungen und Größen werden die Ergebnisse einer Schätzung als verlässlich eingestuft. Auch die freie Verfügbarkeit relevanter Größen innerhalb der „COCOMOII database" dürfte für eine zunehmende Verbreitung sorgen.

3.4 Fazit: Aufwandsschätzung

Die zurückliegenden Ausführungen enthalten zum einen die zahlreichen Schwierigkeiten und Probleme in der Anwendung einer Aufwandsschätzung und zum anderen verdeutlichen sie die Komplexität in der konkreten Umsetzung der Verfahren. Es zeigt sich, dass eine hohe Genauigkeit der Schätzergebnisse mit einem hohen Aufwand der Schätzung selbst einhergeht. Die Entscheidung, welche ein Verfahren dem anderen vorzieht, kann somit nicht klar begründet werden, zumal es selbst in der Literatur an Vergleichsmöglichkeiten mangelt. Im Hinblick

[57] sog. „Self-Descriptness"

auf die praxistaugliche Anwendung bestehen zusätzlich die eingangs des Kapitels erwähnten Probleme, so dass Schwierigkeiten im Umgang mit den Verfahren wiederum Analogieschlüsse und Expertenmeinungen hervorrufen können.[58]

Demnach empfiehlt sich ein genau spezifiziertes Vorgehen der Aufwandsschätzung, welches an die Gegebenheiten der eigenen Softwareentwicklung angepasst ist. Dabei erlangt eine Dokumentation der Ergebnisse ebenso an Bedeutung wie eine konsequente Durchsetzung der gewählten Verfahren, um Fehlerpotenziale von vornherein auszuschließen.

3.5 Aufwandschätzung und Systemfamilien

Unter ökonomischen Gesichtspunkten, welche vorrangig innerhalb einer Aufwandsschätzung betrachtet werden, sind bezüglich der Systemfamilienentwicklung zwei zentrale Kostenblöcke zu unterscheiden:

- Kosten, welche im Zusammenhang mit der Entwicklung der Domäne und ihrer Bestandteile entstehen sowie
- Kosten der Anwendungsentwicklung, wobei im einfachsten Fall lediglich Aufwände bezüglich der Erstellung von Programmcode[59] zur Inbetriebnahme der Softwarebausteine beziehungsweise Komponenten notwendig werden.

Während neue Anforderungen seitens individueller Kundenwünsche mit Hilfe aktueller Schätzverfahren erfasst und dargelegt werden können, stellt insbesondere die Bildung der Assets innerhalb einer Domänenentwicklung ein Problem dar. Den Kostenschätzungen stehen ex ante derart vielgestaltige Nutzenaspekte durch die Kompositionsmöglichkeiten der Bestandteile einer Systemfamilie gegenüber, dass die alleinige Betrachtung des Aufwandes problematisch ist. So besteht beispielsweise die Gefahr, dass eine wichtige Softwarekomponente nicht berücksichtigt wird, da zu hohe Kosten anfangs gegen deren Einsatz in der Systemfamilie sprechen. Des Weiteren besteht das Problem, dass der Nutzen der Komponente

[58] siehe Abschnitt 3.2.1 - Adhoc Methoden
[59] sog. „clue code"

durch den Einsatz betriebswirtschaftlicher Kenngrößen in der Regel nur ex post ermittelt werden kann. Somit stehen in diesem Fall unsichere Schätzungen einem ungewissen Nutzen für die Zukunft gegenüber.

Im Zusammenhang mit aktuellen Entwicklungen sei hier das *„commercial-off-the-shelf (COTS) integration cost model"* erwähnt, welches dem dargelegten Problemen Rechnung trägt.[60] Dieses ist sehr eng mit dem vorgestellten COCOMOII Verfahren verknüpft und basiert ebenfalls auf Wertungen und Verrechnungen verschiedener Kostentreiber. Der entscheidende Nachteil dieses Ansatzes ist jedoch, dass seitens der Integration von COTS-Komponenten davon ausgegangen wird, dass diese keinerlei Veränderung bedürfen. Die Komponenten verfügen lediglich über Schnittstellen zur Steuerung und bieten Funktionalität, welche exakt in das benötigte System übernommen wird. In der Realität sind jedoch meist Anpassungen und Änderungen einzelner Komponenten notwendig, um diese korrekt in die Systemfamilie zu integrieren, so dass dieser Ansatz hier nicht weiter verfolgt werden soll.

In Bezug auf die dargestellte Problematik scheinen demnach Aufwandsschätzungen sinnvoller, welche lediglich die Entwicklung einzelner Anwendungen auf Basis einer Systemfamilie berücksichtigen. Betrachtet man den grundsätzlichen Ablauf der Erstellung von Software im Rahmen von Kundenanforderungen, so geht allen Entwicklungen die Anforderungsanalyse voraus. Innerhalb des Ansatzes der Systemfamilien werden diese Anforderungen mit den Merkmalen dieser abgeglichen. Sind bestimmte Merkmale nicht vorhanden und besteht der Entschluss diese eventuell in die geforderte Anwendung zu implementieren, so werden in der Regel wirtschaftliche Überlegungen notwendig. Den klassischen „make-or-buy" Entscheidungen anschließend, besteht im Zusammenhang mit Systemfamilien zusätzlich die Möglichkeit, neue Anforderung in ein Merkmal der Systemfamilie zu überführen. Bezüglich dieser Entscheidungen kann eine Aufwandsschätzung eine wertvolle Hilfe darstellen, insbesondere im Zusammenhang mit den Investitionen, welche in Verbindung mit dem Aufbau einer Systemfamilie stehen. Die Anwendung eines Schätzverfahrens setzt dahingehend eine Dokumentation vergangener Entwicklungen voraus. Aufgrund des hohen Aufwands und des Risikos innerhalb der Bildung einer Systemfamilie ist dies zumeist gegeben.

[60] vgl. COCOMO database: http://sunset.usc.edu/research/COCOMOII/

Somit bietet sich ein Aufwandsschätzverfahren an, welches bestehende und neu zu implementierende Software gleichermaßen berücksichtigt. Sowohl das Function Point-, als auch das COCOMOII Verfahren beinhalten dahingehende Möglichkeiten. Allgemeinen Empfehlungen folgend, wird jedoch im weiteren Verlauf der Arbeit vorrangig das *COCOMOII* Verfahren und das darin enthaltene *Reuse Model* betrachtet.[61] In Hinblick auf eine prototypische Umsetzung sei dies zusätzlich mit der einfachen Handhabung und der freien Verfügbarkeit des Verfahrens begründet.

3.6 Konkrete Zielstellung

Die bisher dargelegten, theoretisch geprägten Ausführungen stellen die Grundlage für die nun folgende praxisorientierte Herangehensweise dar. Dabei soll deutlich werden, dass der Ansatz der Systemfamilien auf lange Sicht einer einmaligen Gesamtentwicklung vorzuziehen ist. Den Ansatzpunkt bilden die innerhalb der Anforderungsanalyse dargelegten Kundenwünsche im Rahmen einer Anwendungserstellung. Bezüglich dessen, sind mehrere Szenarien im Zusammenhang mit dem Ansatz der Systemfamilien denkbar. Für jedes dieser Szenarien muss demnach ein Vorgehen in Bezug auf eine vorzunehmende Aufwandsschätzung spezifiziert werden, welches den ersten Teil der Zielstellung umfasst. Innerhalb eines zweiten Teils stehen Ansätze zur konkreten, anwendungsorientierten Umsetzung im Vordergrund. Insbesondere die Erfassung, Aufbereitung und Darlegung der Anforderungen in entsprechenden Datenformaten werden im Sinne einer gezielten Nutzung analysiert. In Abbildung 3.12 ist die beschriebene Zielstellung inklusive der erwähnten Szenarien skizziert.

[61] vgl. [Stahlknecht99] S.475

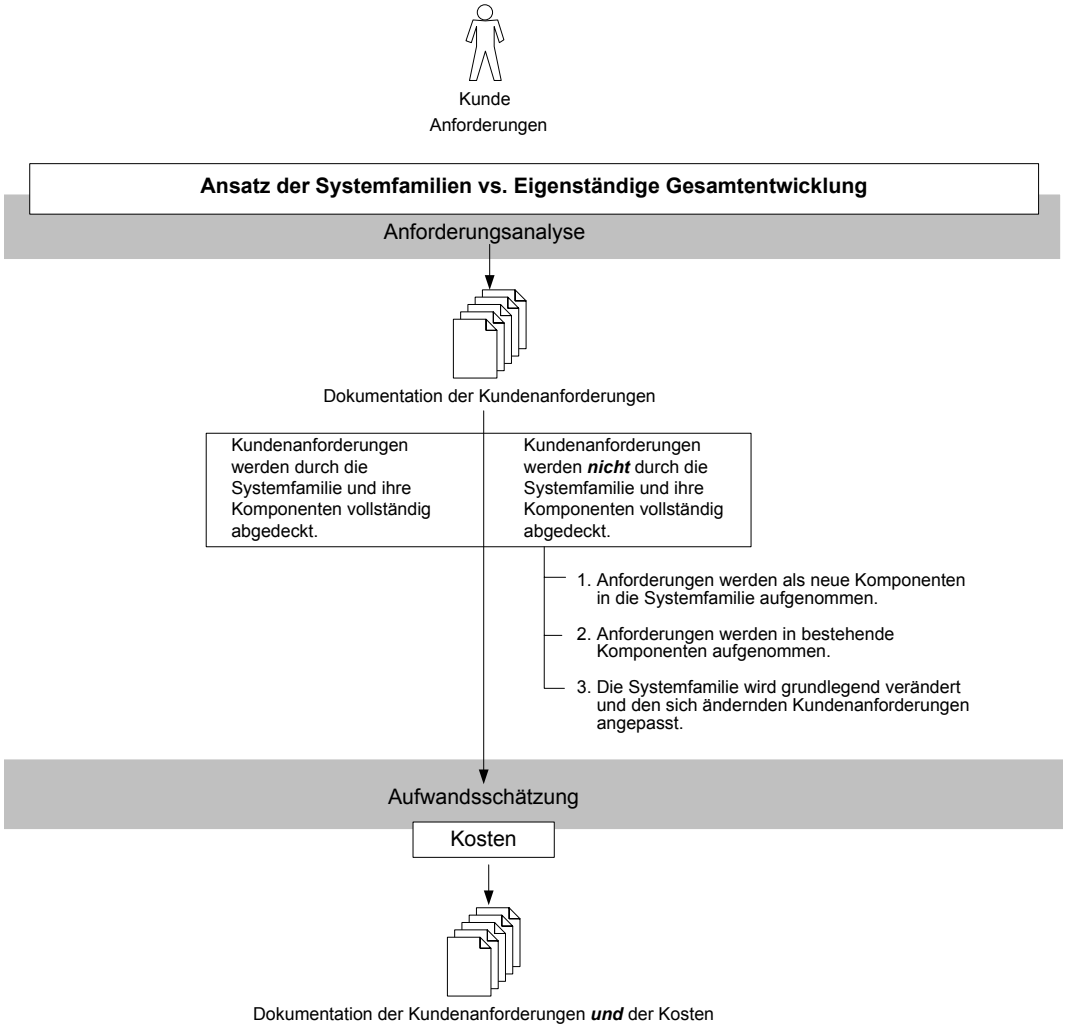


Abbildung 3.12: Entscheidung für oder gegen Systemfamilien

Das angestrebte Resultat bildet eine Dokumentenstruktur, welche Aufwendungen anhand der erfassten Anforderungen widerspiegelt. Im Idealfall sind zudem verschiedene Aufbereitungen des Dokuments möglich, um dem jeweiligen Einsatzzweck entsprechende Präsentationsmöglichkeiten erstellen zu können. Dieser Aspekt fließt in die erwähnte Analyse der Datenformate ein, wobei insbesondere Anwendungsmöglichkeiten der eXtensible Markup Language betrachtet werden. Die letztendlich angestrebten Vorteile wären somit:

- Die dargelegten, geschätzten Aufwendungen bilden die Basis für nachvollziehbare Entscheidungen hinsichtlich Kosten- und Nutzen.
- Durch die mögliche Zuordnung der Kosten werden eigene Defizite und organisatorische Fehlentwicklungen deutlich.
- Der Auftritt gegenüber dem Kunden erscheint aufgrund nachvollziehbarer Kostenaussagen souveräner.

Im weiteren Verlauf der Arbeit sind somit konkrete Überlegungen hinsichtlich der Integrationsmöglichkeiten der Aufwandsschätzung in die Systemfamilienentwicklung notwendig. Es gilt zu untersuchen, inwieweit Ansätze bestehen, Anforderungen und Merkmale sinnvoll zu erfassen und nutzen zu können. Eine, wie bereits erwähnte, sinnvolle Notation ist dabei ebenso entscheidend, wie die Verwendung dieser. Dazu werden im anschließenden Lösungsansatz bestehende Möglichkeiten untersucht und vorgestellt mit dem Ziel der Einordnung der Aufwandsschätzung in den Prozess der Anwendungsentwicklung innerhalb von Systemfamilien. Die zentralen Eckpunkte sind somit zusammengefasst:

- *Wo* und unter welchen Bedingungen kann eine Aufwandsschätzung in die Anwendungserstellung im Umfeld von Systemfamilien integriert werden und
- *Wie* kann dies praktisch umgesetzt werden?

4 Lösungsansatz

Der im Anschluss aufgezeigte Lösungsweg beschreibt ein Vorgehen ausgehend von den Ergebnissen der Anforderungsanalyse. Insbesondere die Dokumentation der Anforderungen und eine sinnvolle Nutzung dieser für eine Aufwandsschätzung spielt in diesem Zusammenhang eine wichtige Rolle. Im Sinne einer möglichst effizienten und weiterführenden Verarbeitung wird dazu die eXtensible Markup Language (XML) betrachtet. Dabei wird eine allgemeine Auszeichnung „DocBook" erläutert, während als konkretes Beispiel im Zusammenhang mit Systemfamilien das Datenformat des „FORE" Projekts vorgestellt wird. Zunächst sollen jedoch Integrationsmöglichkeiten der Aufwandsschätzung in die Entwicklung von Systemfamilien untersucht werden. Dabei wird hier ein allgemein gültiges Vorgehen angestrebt, während eine konkrete Umsetzung am Beispiel des im anschließenden Kapitels vorgestellten Prototyps erfolgt. Eine zentrale Fragestellung wird in diesem Zusammenhang der Detaillierungsgrad und die Anpassbarkeit des Schätzverfahrens an die besonderen Erfordernisse einer Systemfamilie sein. Zudem erfolgt eine Differenzierung hinsichtlich verschiedener Szenarien, welche in diesem Umfeld auftreten können. Abschließend werden anhand der XML Ansätze zur Integration der Ergebnisse der Aufwandsschätzung in das Datenformat der Anforderungsanalyse gesucht. Einen Überblick über die nun folgenden Ausführungen soll Abbildung 4.1 geben:

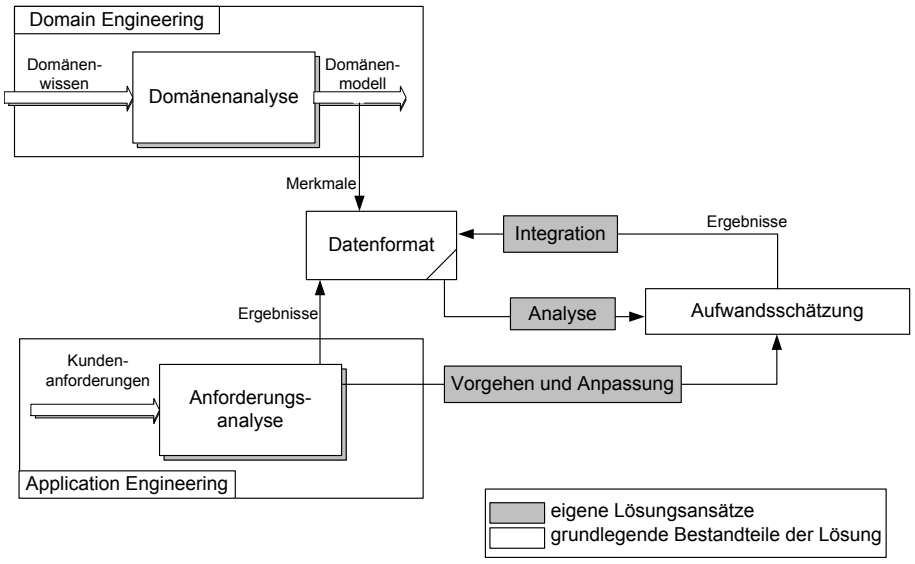


Abbildung 4.1: Lösungsansatz: Überblick

Die an das zweite Kapitel der Arbeit angelehnte Abbildung beinhaltet die vorzunehmenden Arbeitsschritte des Lösungsansatzes und rückt das erwähnte Datenformat in den Mittelpunkt. Zwar wird Wert auf eine weitere Verarbeitung dessen gelegt, jedoch soll hier der Fokus vorrangig auf den einzelnen Bestandteilen des Formats liegen. Zudem soll hier auf die zentrale Bedeutung der Aufwandsschätzung hingewiesen werden. Diese stellt die Grundlage und den Ausgangspunkt aller weiteren Überlegungen dar. Im Anschluss werden daher Ansätze für die Anwendung einer Aufwandsschätzungen innerhalb der typischen, im Kontext von Systemfamilien spezifischen Vorgänge gesucht.

4.1 Szenarien einer Anwendungsentwicklung

Die Entwicklung einer den Kundenwünschen entsprechenden Anwendung im Zusammenhang mit Systemfamilien weicht, wie zurückliegende Ausführungen darlegen, von der klassischen Anwendungsentwicklung ab.[1] Statt der Überführung

[1] siehe Abschnitt 2.2 - Die Entwicklung einer Systemfamilie

der Kundenanforderungen in Prozesse der Programmcodeerstellung muss hier eine differenzierte Betrachtung erfolgen. Insbesondere das Zusammenspiel des *Application-* und *Domain Engineerings* ist dabei von Interesse. Zudem müssen die unterschiedlichen Möglichkeiten, inwieweit Kundenanforderungen, welche nicht durch die Systemfamilie erfüllt werden können, berücksichtigt werden. Zunächst soll dieses teils problematische Umfeld der Systemfamilienentwicklung nochmals in etwas veränderter Form in Abbildung 4.2 aufgegriffen werden, um anschließende Überlegungen darauf aufzubauen.[2]

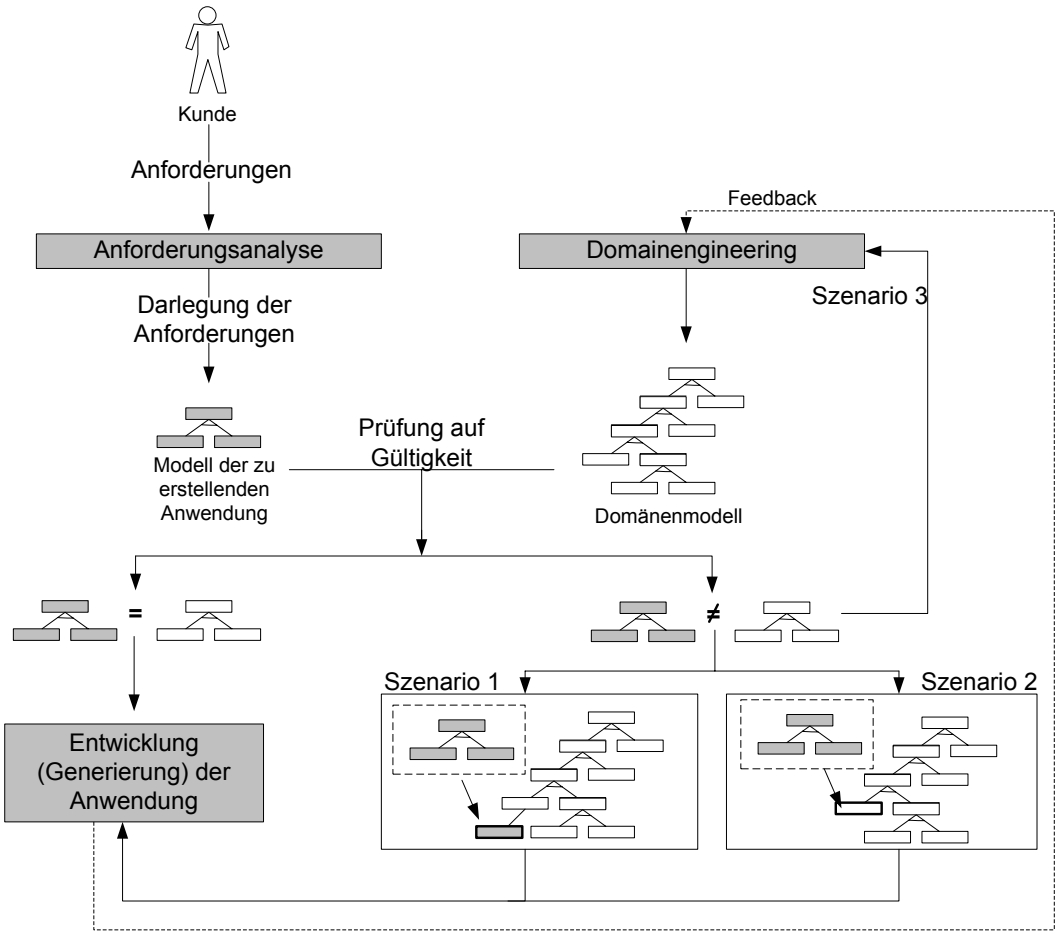


Abbildung 4.2: Zusammenspiel und Vorgehen einer Anwendungsentwicklung im Kontext von Systemfamilien

[2]in Anlehnung an Abschnitt 2.2 - Die Entwicklung einer Systemfamilie; [SEI02]

Wie zu erkennen ist, stellt die besondere Herausforderung in der Entwicklung einer Systemfamilie die möglichst umfassende Erfassung aller relevanten Kundenanforderungen dar. Aufgrund zeitlicher und ökonomischer Ursachen ist dies jedoch nicht immer gegeben, so dass zwischen verschiedenen Möglichkeiten gewählt werden muss, die Anforderungen an eine Software dennoch anzubieten. Daher wird im Folgenden zwischen den in Abbildung 4.2 dargestellten Szenarien unterschieden:

1. Auf Basis der Anforderungen wird ein neues spezifisches Merkmal der Systemfamilie entwickelt und in diese integriert.[3]
2. Innerhalb der in der Systemfamilie bestehenden Merkmale wird nach Ansätzen gesucht, vorhandene Assets den Anforderungen anzupassen.
3. Aufgrund eines veränderten technologischen, wettbewerblichen Umfelds und damit einhergehenden wechselnden Anforderungen müssen grundlegende Veränderungen an der Systemfamilie selbst vorgenommen werden.

In den Mittelpunkt der Betrachtung rücken nun somit begründete Entscheidungen für oder gegen eines der Szenarien. Daher sollen die Möglichkeiten des Aufwandsschätzverfahrens COCOMOII hinsichtlich Anpassbarkeit, Skalierbarkeit und letztendlich Integrierbarkeit in die in Abbildung 4.2 dargestellten Vorgänge aufgezeigt werden.

4.1.1 Szenario 1: Die Integration eines neuen Merkmals

Die innerhalb dieses Teilabschnitts getroffenen Aussagen beruhen auf der Annahme, dass ein neues Merkmal der Systemfamilie erstellt wird und zwar ohne Rückgriffe auf bereits bestehende Software beziehungsweise Assets. Diese Entwicklung entspricht dadurch weitestgehend den Merkmalen einer klassischen Anwendungsentwicklung. Jedoch müssen auch hier wiederum für das Umfeld der Systemfamilien spezifische Aspekte berücksichtigt werden. So kann beispielsweise davon ausgegangen werden, dass aufgrund des Fachwissens, welches Personen im Entwicklungsumfeld einer Systemfamilie besitzen, zahlreiche Aufwände in verminderter

[3] siehe Abschnitt 2.3.1 - Die Erfassung und Modellierung von Merkmalen

Ausprägung auftreten. Sollen verlässliche Aussagen über den Aufwand des neu aufzunehmenden Merkmals getroffen werden, muss das gewählte Schätzverfahren derartige Abweichungen berücksichtigen. Die Skalierbarkeit des COCOMOII Verfahrens und die damit verbundene hohe Schätzgenauigkeit für verschiedenste Softwareprojekte sprechen daher für dessen Einsatz. Die im *Post Architecture Model* erfassten Kostentreiber und Skalenfaktoren müssen den Belangen einer Entwicklung für Systemfamilien entsprechend bewertet und gewichtet werden.

Die dargelegte Analyse soll hier jedoch einen weiterer Aspekt beinhalten. Über die bereits entwickelten Merkmale der Systemfamilie sollten in der Regel derart viele Informationen vorliegen, dass Analogieschlüsse und Urteile beruhend auf dem Wissen über die Funktionsweise existierender Bestandteile als mögliche Schätzmethoden unterstützend herangezogen werden können. Im Sinne eines allgemein gültigen Vorgehens soll hier von folgendem Fall ausgegangen werden: Die Kenntnis des Umfangs bereits entwickelter Softwareelemente sowie die Kenntnis über die Anzahl der Mitarbeiter, welche über einen nachvollziehbaren Zeitraum an diesen gearbeitet haben. Daraus wiederum lassen sich die für eine Aufwandsschätzung elementaren Größen *Personenmonate* [PM] und *Größe des Programmquellcodes* [LOC] ableiten. Das hier angestrebte Ziel ist letztendlich die Aufbereitung einer Erfahrungskurve. Dies erscheint insbesondere dahingehend sinnvoll, da die Softwareentwicklung für eine Systemfamilie in den meisten Fällen unter dem gleichen oder einem ähnlichen Kontext stattfindet und somit besonders in diesem Umfeld Analogieschlüsse zulässt. Dieses Vorgehen kann zusätzlich, entsprechend den Vorgaben des Function Point Verfahrens, beliebig verfeinert werden.[4]

Der angestrebte Vorteil in einer zusätzlichen Schätzung entspricht weitestgehend den Vorteilen des bereits beschriebenen *Function Point Verfahrens*. Dabei wird davon ausgegangen, dass bestimmte Kosten einer Softwareentwicklung derart spezifische Ursachen haben, dass die beispielsweise im COCOMOII-Verfahren verwendeten Kostentreiber und Skalenfaktoren diese nicht erfassen. Die Wahrscheinlichkeit der Aussagen über Aufwand und Kosten des dennoch hier präferierten COCOMOII-Verfahrens können somit anhand einer Erfahrungskurve überprüft werden. Zudem sind frühe Aussagen zu Zeit und Kosten möglich.

[4] siehe Abschnitt 3.3.1 - Das Function Point Verfahren

Zunächst sollen der Ablauf und das Zusammenspiel der bisher beschriebenen Aufwandsschätzverfahren in Abbildung 4.3 skizziert werden, welche an die in Abbildung 4.2 skizzierten Vorgänge anschließen.[5]

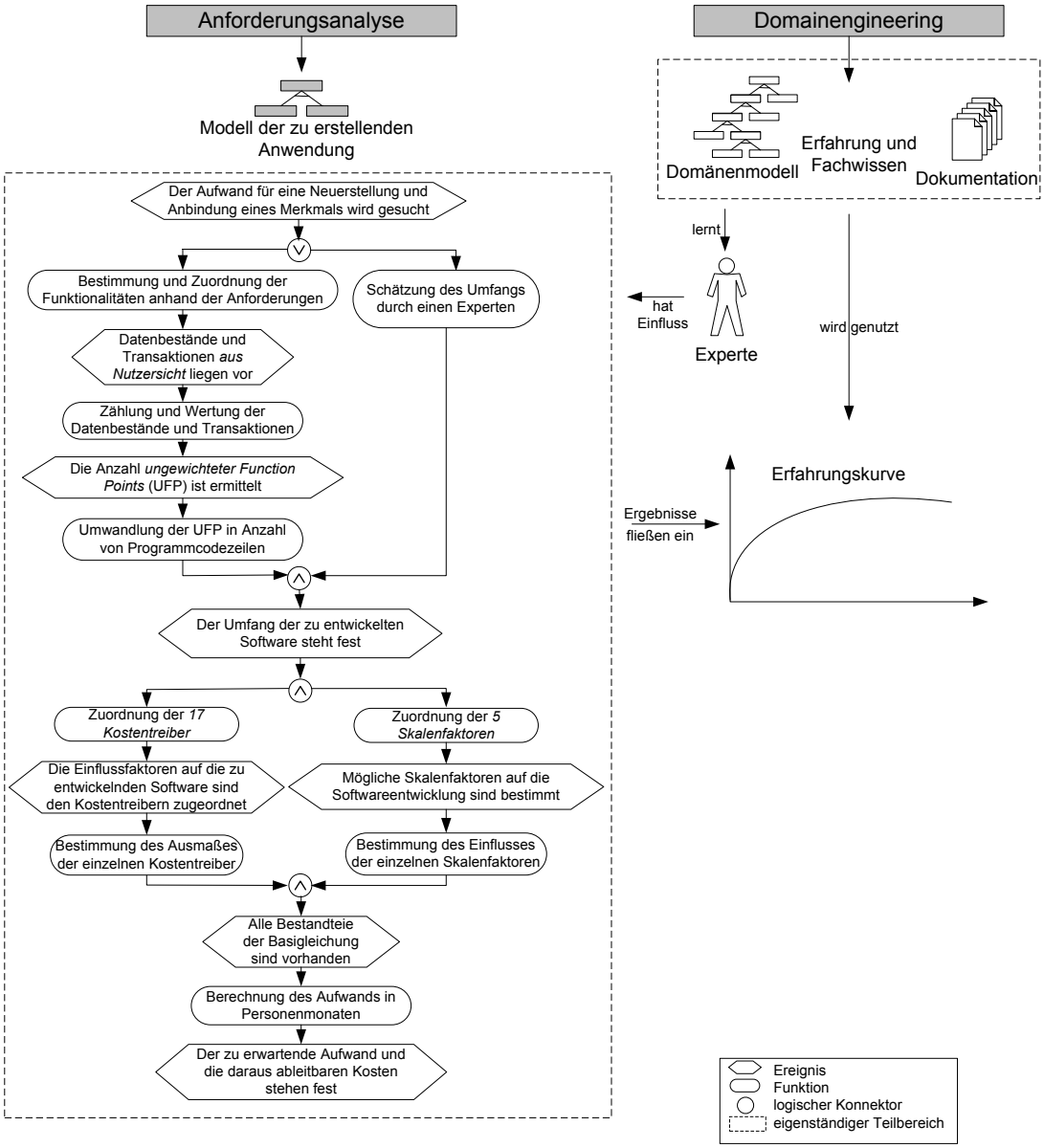


Abbildung 4.3: Szenario 1: Erstellung neuer Software

[5]siehe zur Notation - Anhang B *Ereignisgesteuerte Prozessketten*

Das dargelegte Vorgehen soll nun genauer spezifiziert werden. Wie bereits beschrieben[6] ist der Ausgangspunkt jeder Aufwandsschätzung eine ex ante Bestimmung des Umfangs der zu entwickelnden Software. Während dazu die sequentielle Abfolge der Teilschritte des COCOMOII-Verfahrens die Bestimmung der *ungewichteten Function Points* vorschreibt, besteht zusätzlich die Möglichkeit der Schätzung der Programmgröße seitens eines oder mehrerer *Experten*. Diese begründen ihre Entscheidungen auf Basis der Informationen aus der bereits entwickelten Domäne und dem im Rahmen bisheriger Entwicklungen erlangtem Fachwissen. Liegt der Umfang des zu entwickelnden Programmcodes vor, sind somit bereits frühzeitige Aussagen zu dem zu erwartenden Aufwand anhand der *Erfahrungskurve* möglich. Je nach Anzahl der darin erfassten und abgebildeten Softwareentwicklungen sind dabei verlässliche Aussagen zu erwarten.

Aufgrund der im Rahmen einer zu entwickelnden Lösung geforderten, größtmöglichen Genauigkeit der Schätzergebnisse, wird hier eine weitere Verfolgung des COCOMOII-Verfahrens empfohlen und die Erfahrungskurve als sinnvolle Ergänzung dessen betrachtet. In einem weiteren Schritt gilt es somit die voraussichtlichen Kosten, anhand der in dem Modell der Anwendung erkennbaren Einflussfaktoren, zu bestimmen. Anschließend müssen diese zusammengefasst und gemäß ihres zu wertenden Einflusses, den *Kostentreibern* und *Skalenfaktoren* zugeordnet werden. Weiterhin wird eine Wichtung dieser durchgeführt. Dabei können projektspezifische Gegebenheiten, jedoch auch die Besonderheiten von Entwicklungen im Umfeld von Systemfamilien einfließen. Abschließend wird der zu erwartende Aufwand durch die Verrechnung aller Kostentreiber und Skalenfaktoren anhand des Formelwerks des COCOMOII-Verfahrens ermittelt. Die Zusammenhänge zwischen dem Umfang der Software und dem tatsächlichen Aufwand sollten neben der Übernahme in die Erfahrungskurve in Verbesserungen der Schätzverfahren einfließen.

Die Besonderheiten des hier dargelegten Anwendungsfelds lassen allgemeine Aussagen und Empfehlungen zu internen Arbeitsschritten innerhalb der Aufwandsschätzungen wenig sinnvoll erscheinen. Vielmehr wird hier davon ausgegangen, dass die dargestellte Verwendung der Aufwandsschätzung sowie die Bestimmung, Zurechnung und Wichtung von Kostentreibern und Skalenfaktoren ein Prozess

[6] siehe Abschnitt 3.1 - Die Grundlagen der Aufwandsschätzung

ständiger Anpassungen und Verfeinerungen ist. Die in Abbildung 4.3 skizzierte, modellhafte Abfolge ist daher als Ausgangspunkt für eine Integration der Aufwandsschätzung zu betrachten. Eine Bewertung und Anpassung der Ergebnisse wird im weiteren Verlauf des Kapitels erfolgen.[7]

4.1.2 Szenario 2: Die Anpassung bestehender Elemente

Während innerhalb von Szenario 1 bestehende Softwarebestandteile einer Systemfamilie weitestgehend vernachlässigt wurden, so liegt der Fokus von Szenario 2 vorrangig auf diesen. Anstatt der Integration eines neuen Merkmals, soll bestehende Software nun sinnvoll ergänzt und weiterentwickelt werden. Die Anforderungsanalyse erlangt daher besonders hier eine über die reine Erfassung von Kundenwünschen hinausgehende Bedeutung. Vor allem ist dabei zu berücksichtigen, welche Auswirkungen derartige Eingriffe in den bestehenden Programmcode auf das Zusammenspiel der einzelnen Softwareelemente haben und welche strukturellen Anpassungen dahingehend notwendig werden. Während innerhalb von Szenario 1 derartige Aspekte noch durch das Aufwandsschätzverfahren selbst erfasst werden, so müssen hier Überlegungen hinsichtlich der Wirtschaftlichkeit andere Lösungsansätze verfolgen.

Einen sinnvollen Ansatz bietet das auf dem COCOMOII-Verfahren aufbauende Reuse Model. Dieses berücksichtigt Spezifika von Anpassungs- und Weiterentwicklungsprojekten und ermöglicht die Bestimmung des Umfangs des Programmquellcodes mittels Einflussfaktoren, welche über bisher in Betracht gezogene Aspekte hinausgehen.[8] Anstatt der Ermittlung der Anzahl der Programmcodezeilen als Ergebnis bestimmbarer Funktionalitäten, in Form ungewichteter Function Points, wird nun das *Reuse Model* verwendet. Zunächst soll diese Integration in bisherige Abläufe anhand von Abbildung 4.4 dargestellt werden.

[7] siehe Abschnitt 4.3 - Die Anpassung der Aufwandsschätzung
[8] siehe Abschnitt 3.3.3 Das COCOMOII Verfahren

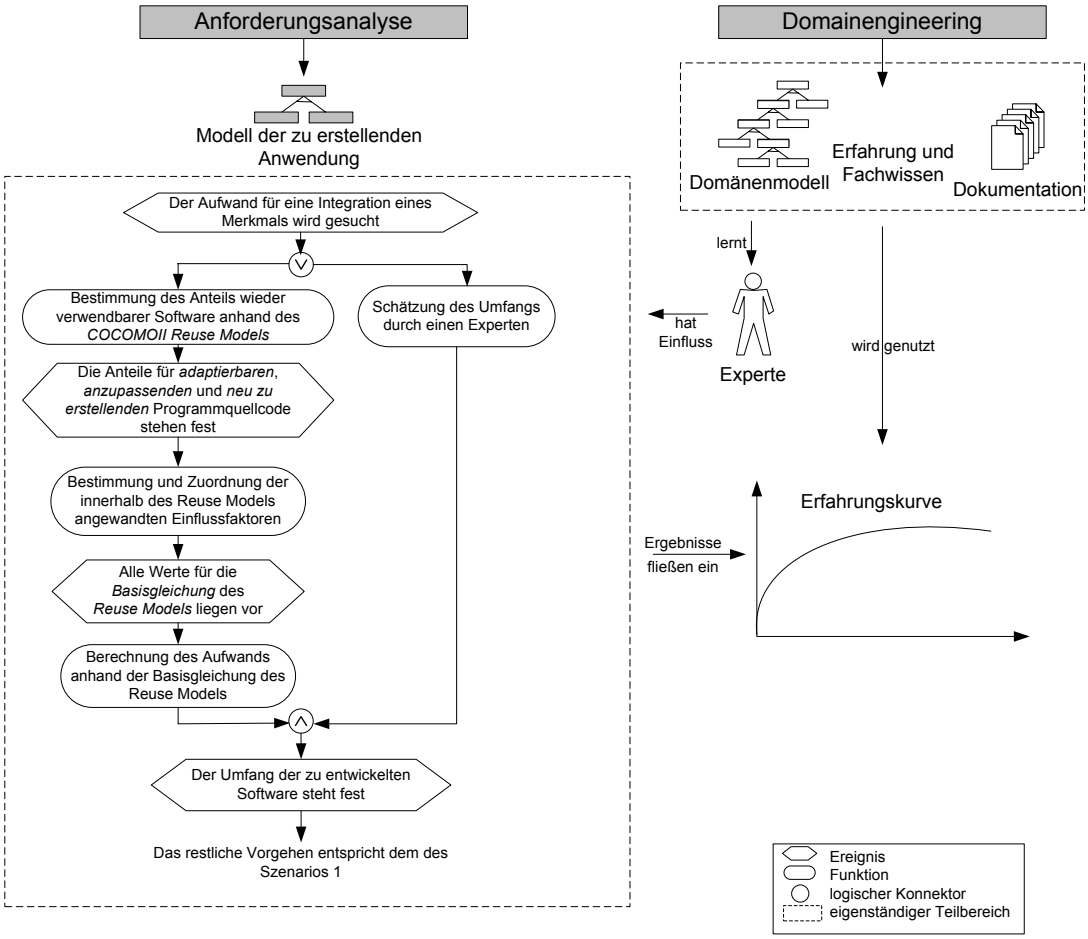


Abbildung 4.4: Szenario 2: Berücksichtigung bestehender Software

Die letztendliche Aufwandsschätzung einer innerhalb dieses Szenarios betrachteten Entwicklung erfolgt daher nicht zwingend im Kontext von Systemfamilien, sondern vorrangig aus Sicht wieder verwendeter Bestandteile des Programmquellcodes und darauf aufbauender Entwicklungen. Daher sollten die Ergebnisse derartiger Schätzungen anfangs geprüft werden und Szenario 1 ähnlich, fortwährende und konsequente Erweiterungen sowie Anpassungen der Schätzprozesse erfolgen. Auch die Ergebnisse dieses Szenarios und das dargestellte Vorgehen werden in späteren Ausführungen analysiert und angepasst.[9]

[9] siehe Abschnitt 4.3 - Die Anpassung der Aufwandsschätzung

4.1.3 Szenario 3: Neuerstellung und Anpassung einer Systemfamilie

In diesem Szenario wird davon ausgegangen, dass neue Anforderungen derart umfassend und vielgestaltig sind, dass eine Umgestaltung bestehender Software nicht mehr ausreichend ist. Dies betrifft Anforderungen, welche über einzelne Kundenwünsche hinausgehen und sind vom Markt oder Wettbewerb hervorgerufene, technologische Erneuerungen in den bisher erstellten Produkten. Normalerweise sind dadurch zwar umfassende Neugestaltungen und Anpassungen der Systemfamilie notwendig, jedoch bleiben die in den Gemeinsamkeiten erfassten Kernfunktionalitäten in der Regel erhalten. Von einer vollständigen Neuerstellung einer Systemfamilie kann daher nur in den seltensten Fällen ausgegangen werden. Dieses hier exemplarisch aufgegriffene Szenario dient deshalb vorrangig einer im späteren Verlauf des vorliegenden Kapitels dargelegten Analyse der Kosten, welche im Zusammenhang mit Systemfamilien bestehen.

Auch hier soll wiederum das COCOMOII-Verfahren Anwendung finden. Komplett neu zu erstellende Softwarebestandteile werden gemäß der vorgeschriebenen Arbeitsschritte geschätzt und kumulativ verrechnet.[10] Werden Softwarebestandteile herangezogen, welche auf Basis von Überlegungen zur Wiederverwendung genutzt werden, so wird hier das Reuse Model empfohlen.[11]

Aufgrund der hohen Risiken und den immensen Aufwendungen, welche im Zusammenhang mit der kompletten Erstellung einer Systemfamilie stehen, rücken Fragen hinsichtlich der Granularität in Verbindung mit der Genauigkeit einer Aufwandsschätzung in den Mittelpunkt. Aufgrund der angestrebten, möglichst verlässlichen Ergebnisse der Schätzung sollte demnach nicht die vollständige Anforderung betrachtet und geschätzt werden, sondern vielmehr einzelne, erkennbare Bestandteile davon hinsichtlich ihrer Funktionalität untersucht werden. Diese Teile werden einzeln geschätzt, um letztendlich den kumulativen Gesamtaufwand der Software zu bilden. Einer hohen Genauigkeit der Schätzung steht somit ein hoher Aufwand der Schätzung selbst gegenüber. Jedoch scheint die Analyse und

[10] siehe Abschnitt 4.1.1 - Szenario 1
[11] siehe Abschnitt 4.1.2 - Szenario 2

Schätzung auf mehreren Ebenen der Software im Hinblick auf das Gesamtprojekt Systemfamilie angemessen.

4.1.4 Bedeutung der Ergebnisse

Innerhalb dieses Abschnitts sollen Besonderheiten und Wertungen hinsichtlich der Ergebnisse der beschriebenen Szenarien aufgegriffen und erläutert werden. Dabei liegt der Fokus auf Ideen und Empfehlungen für die praktische Umsetzung der Lösung.

Die Entscheidung, welches der beschriebenen Szenarien verfolgt werden soll, wird letztendlich ausgehend vom Kundenwunsch auf Basis der geschätzten Kosten getroffen. Dabei muss zusätzlich die Alternative des Zukaufs von Software betrachtet werden. Inwieweit eine Aufwandsschätzung für die Integration dieser angebracht ist, hängt wiederum von spezifischen Gegebenheiten im Zusammenhang mit der gegebenen Systemfamilie ab. Sind dahingehend hohe Aufwendungen zu erwarten, sollte auch hier das COCOMOII-Verfahren in der innerhalb von Szenario 1 beschriebenen Form herangezogen werden.

Weiterhin soll die Bedeutung einer Dokumentation hervorgehoben werden. Erst durch diese kann das für die dargestellten Arbeitsschritte notwendige Fachwissen vermittelt und genutzt werden. Die verwendete Erfahrungskurve setzt dokumentierte, nachvollziehbare Kenngrößen sogar zwingend voraus. Zusätzlich sollten alle vorgenommenen Maßnahmen der Aufwandsschätzung möglichst genau und begründet dokumentiert werden. Es ist davon auszugehen, dass die Aufwandsschätzungen der ersten Anwendungsentwicklungen aufgrund des schwierigen Umfeldes in gewissem Maße von den tatsächlichen Werten abweichen. Eine Dokumentation hilft Fehlerursachen im Schätzprozess zu erkennen, um in der weiteren Nutzung diesen anzupassen und zu verbessern. Eine Dokumentation des Schätzprozesses sollte dabei folgende Punkte umfassen:

- Die Bestimmung des Umfangs des Programmquellcodes anhand ungewichteter Function Points oder des Reuse Models.
- Die Bestimmung der Einflussfaktoren auf den Aufwand anhand des vorliegenden Merkmalsmodells der Anwendung.

- Die Zuordnung der Einflussfaktoren zu den Kostentreibern.
- Die Bewertung und Wichtung der Kostentreiber und Skalenfaktoren.

Zusätzlich soll hier im Falle einer notwendigen, möglichst präzisen Schätzung das Hinzuziehen eines oder mehrerer Experten empfohlen werden. Diese können im Sinn der in Abschnitt 3.2 beschriebenen Formen der Expertenschätzung Einfluss auf die aufgezählten Arbeitsschritte nehmen. Auch sei angemerkt, dass die Granularität der zu schätzenden Software, wie anhand von Szenario 3 beschrieben, beliebig erhöht werden kann.

4.2 Bewertung und Erläuterung der Kosten

Die zurückliegenden Ausführungen beinhalten vorrangig die Ermittlung der Kosten beziehungsweise Aufwendungen, welche bezüglich einer geplanten Softwareentwicklung zu erwarten sind. Das angestrebte Ziel einer eindeutigen Zurechnung von Merkmalen zu Aufwendungen kann mit Hilfe des spezifizierten Vorgehens erreicht werden. In einem nächsten Schritt gilt es nun anhand eines Vergleichs der alternativen Szenarien, insbesondere hinsichtlich einer einmaligen Gesamtentwicklung, die wirtschaftlich sinnvollste Softwareerstellung zu verfolgen. Die alleinige Betrachtung der Ergebnisse einer Aufwandsschätzung wäre im Umfeld von Systemfamilien dahingehend kritisch, da wichtige Zusammenhänge und Besonderheiten berücksichtigt werden müssen. Eine Entscheidung hinsichtlich der zukünftigen Softwareerstellung sollte daher unter anderem die im Folgenden aufgeführten Überlegungen zum Inhalt haben.

4.2.1 Zentrale Bestandteile der Kosten

Ein zentraler Aspekt innerhalb einer im Umfeld der Systemfamilien angestrebten Entwicklung ist die Zurechenbarkeit der Kosten der eigentlichen Anwendungserstellung und der Kosten, welche im Bestehen der Systemfamilie selbst begründet liegen. Jedoch müssen auch die angestrebten und zu erwartenden Einsparungspotenziale Berücksichtigung finden. Beide Größen, sowohl die Investitionen

im Zusammenhang mit der Systemfamilie als auch gerade dadurch entstehende Einsparungen, können bei den Kosten einer einmaligen Gesamtentwicklung vernachlässigt werden. Vielmehr wird hier von weitestgehend konstanten Kosten ausgegangen. Betrachtet man nun die Kosten mit und ohne dem Bestehen einer Systemfamilie anhand mehrerer Entwicklungsprojekte, ergeben sich die in Abbildung 4.5 dargestellten exemplarischen Kostenverläufe.[12]

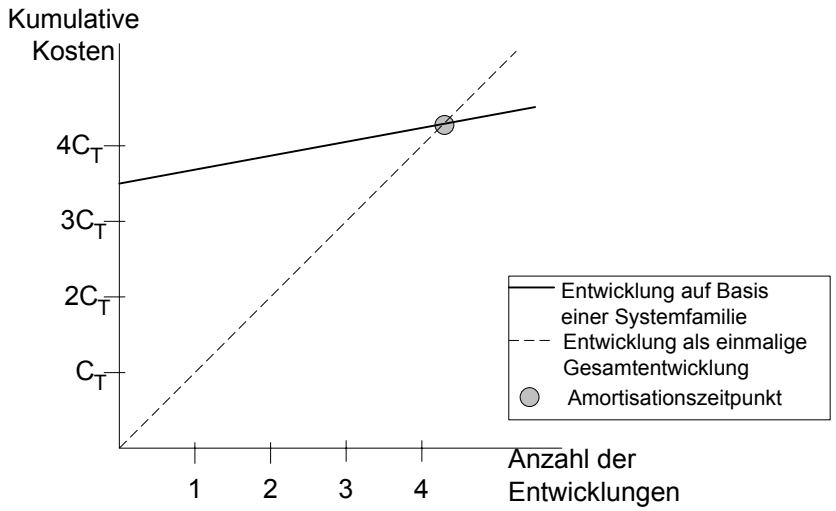


Abbildung 4.5: Vergleich der Kosten

Während die kumulierten Kosten einer einmaligen Gesamtentwicklung konstant ansteigen, so verläuft die Kostenkurve von Entwicklungen im Rahmen von Systemfamilien flacher. Die Ursachen für beide Verläufe sollen nun mathematisch nachgewiesen werden. Dazu werden die relevanten Größen zunächst vorgestellt:[13]

[12] in Anlehnung an [Weiss99] S.46; [Coplien98]
[13] vgl. [Weiss99] S.46ff.

I : die kumulierten Kosten, welche im Rahmen des Domain Engineering als Investition anfallen;

C_T : die Kosten einer einmaligen Anwendungserstellung;

C_F : die Kosten für die Erstellung einer Anwendung auf Grundlage einer Systemfamilie;

S_F : die Einsparungen, welche im Rahmen einer Entwicklung, auf Grundlage einer Systemfamilie, für ein Produkt entstehen.

Weiterhin wird hier davon ausgegangen, dass aufgrund der Einsparungen S_F die Kosten einer einmaligen Gesamtentwicklung höher als die einer Entwicklung auf Basis einer Systemfamilie sind.

$$C_F < C_T$$

Dadurch lassen sich die Kosten für eine Softwareentwicklung im Rahmen einer Systemfamilienentwicklung wie folgt berechnen:

$$C_F = C_T - S_F + \tfrac{I}{n},$$

wobei n die Anzahl bereits entwickelter Softwareprojekte auf Basis der mit der Investition verbundenen Systemfamilie beinhaltet. Mit einer zunehmenden Anzahl von Softwareprojekten amortisieren sich dadurch die Investitionen der Systemfamilie, wodurch sich der im Vergleich zu einer einmaligen Gesamtentwicklung flachere Kurvenverlauf ergibt.

Der Vergleich der Kosten im Rahmen der aufgezeigten Szenarien kann daher nicht allein auf einer einzelnen Entwicklung liegen. Vielmehr sollte sich die Anzahl bisheriger Entwicklungen sowie Anfangsinvestitionen und Einsparungen in den Kostenvergleichen niederschlagen.

Im weiteren Verlauf sollen neben den hier dargestellten Kostenverläufen zusätzliche wirtschaftlich geprägte Betrachtungen hinsichtlich einer geplanten Softwareerstellung vorgestellt werden. Daraufhin werden die dargestellten Zusammenhänge im Laufe einer Anpassung der Aufwandsschätzung berücksichtigt.

4.2.2 Weiterführende Aspekte

Weitere Überlegungen betreffen die Existenz und das Zusammenspiel einzelner Softwarebestandteile. Die Frage, welche im Folgenden betrachtet wird, ist dabei: „*Wann und unter welchen Bedingungen ist eine Softwareentwicklung mit höheren Kosten einer anderen vorzuziehen?*". Um dies beantworten zu können, müssen die Kosten für ein Merkmal im Zusammenhang mit der gesamten Systemfamilie, inklusive zukünftiger Entwicklungen, betrachtet werden.[14] Bezüglich dessen sollen die Grundlagen und Ansätze von „*Verbundvorteilen*"[15] vorgestellt werden. Die darauf aufbauenden, industriell geprägten, theoretischen Konstrukte beschreiben Verbundvorteile als:[16]

„*[...] Kostenersparnisse, welche in der verbundenen Herstellung und dem gemeinsamen Vertrieb von getrennt produzierbaren Gütern begründet liegen.*"

Die Ursachen für niedrigere Kosten sind dabei:

- *Public Input*: Für die Produktion eines Gutes sind Produktionsfaktoren zu beschaffen, die im Unternehmen ohne zusätzliche Kosten für die Produktion eines zweiten Gutes zu nutzen sind.
- *Shared Input*: Die gemeinsame Nutzung von Produktionsfaktoren führt zu einem geringeren Ressourcenverzehr als deren Nutzung in getrennten Unternehmen.
- *Größenvorteile*: Die gemeinsame Nutzung unteilbarer Verfahren oder Investitionsgüter in Forschung, Entwicklung, Konstruktion, Fertigung, Vertrieb, Werbung, etc. ist kostengünstiger als die getrennte.

Wird eine Softwareentwicklung nun als ein Gut angesehen, so können Aufwendungen und Kosten, die im Zusammenhang mit dieser Entwicklung stehen, im Gesamtkontext mit anderen Softwarebestandteilen betrachtet werden. Dies erscheint insbesondere im Umfeld von Systemfamilien sinnvoll, da einmal erstellte Software Einsparungsmöglichkeiten für nachfolgende Entwicklungen beinhaltet.

[14] vgl. [Cohen02]; [Czarnecki01]
[15] economies of scope
[16] vgl. [Kallfass90] S.32ff.

Der in diesem Zusammenhang bestehende Aspekt der Wiederverwendung umfasst letztendlich das Potenzial zur Ausnutzung derartiger Verbundvorteile. Im Rahmen einer Softwareentwicklung gilt es demnach, derartige Zusammenhänge zu erkennen und zu nutzen, was mitunter auch eine Entscheidung für eine Entwicklung bedeutet, welche zunächst mit höheren Kosten verbunden ist.

Weiterführende Fragen beschäftigen sich mit sinnvollen organisatorischen Aspekten im Fokus gezielter wirtschaftlicher Eingriffe. Während im speziellen Fall Verbundvorteile als Ausgangspunkt für unternehmensinterne und entwicklungsspezifische Entscheidungen heranzuziehen sind, so soll hier ein weiterer darüber hinaus gehender Begriff aufgegriffen werden. Die bereits erwähnten Größenvorteile[17] erlangen im Kontext von Systemfamilien eine wichtige Bedeutung. Diese beschreiben:[18]

„*[...] das Sinken der Kosten für ein Gut mit zunehmender Ausbringungsmenge*".

Somit werden auch hier einzelne Softwareentwicklungen betrachtet und nicht eine sich amortisierende Investition wie in Abschnitt 4.2.1 dargestellt. Die Ursachen sind dabei recht vielgestaltig, so dass Größenvorteile exemplarisch anhand von Spezialisierung und Lerneffekten erläutert werden sollen.

Aufgrund mehrmaliger Entwicklungen sich ähnelnder Güter treten bei den beteiligten Mitarbeitern Lerneffekte auf. Diese beherrschen mit zunehmender Anzahl der Entwicklungsprojekte bestimmte Arbeitsschritte zunehmend besser, wodurch sich die Arbeitszeit verkürzt, die Qualität steigt und die Kosten pro Entwicklung sinken. Diese Effekte treten innerhalb komplexer Arbeitsprozesse jedoch nur langsam oder gar nicht auf. Daher ist es sinnvoll die Spezifität der Güter und somit der Arbeitsprozesse zu erhöhen.

Größenvorteile treten demnach erst nach einer gewissen Zeit auf. Im Zusammenhang mit Systemfamilien bedeutet dies, dass eine zunehmende Verkaufszahl von Gütern Größenvorteile hervorrufen kann. Um jedoch sinkende Kosten durch Lerneffekte erzielen zu können, sind Spezialisierungen auf bestimmte Produkte sinnvoll. Im Hinblick auf die dargestellten Szenarien und hinsichtlich der Entscheidung für oder gegen eine mögliche Entwicklung sind Überlegungen bezüglich steigender

[17] economies of scale
[18] vgl. [Carlton00] S.35ff; S.44ff.

Verkaufszahlen spezifischer Softwareprojekte von Interesse. Es ist zu entscheiden, inwieweit beispielsweise eine neue Softwarekomponente die Wahrscheinlichkeit zur Erzielung von Größenvorteilen erhöht und somit niedrigere Kosten zur Folge hat. Im Ergebnis würde die in Abbildung 4.5 dargestellte Kostenkurve für Systemfamilien nicht nur einen flacheren, sondern auch einen fallenden Verlauf aufweisen.

Die zurückliegenden Überlegungen hinsichtlich der Kosten beziehungsweise des Aufwands zeigen, dass die innerhalb der dargestellten Szenarien erreichten Ergebnisse nur unzureichend die angestrebte Zielstellung erfüllen. Daher sind im Folgenden Überlegungen notwendig, welche insbesondere das Umfeld der Systemfamilien berücksichtigen.

4.3 Die Anpassung der Aufwandsschätzung

Die im Sinne der Zielstellung angestrebte Lösung muss von den vorherigen Ausführungen ausgehend zwei zentrale Elemente einer Wirtschaftlichkeitsbetrachtung zum Inhalt haben. Diese sind:

1. Nutzenaspekte und Kosteneinsparungen durch aktuelle Entwicklungen,
2. zusätzliche Aufwendungen aufgrund des Bezugs zu einer Systemfamilie.

Aufgrund dessen stehen nun Ansätze hinsichtlich der Anpassbarkeit und Skalierbarkeit des genutzten COCOMOII Verfahrens an diese Besonderheiten im Mittelpunkt der Betrachtung. Dabei ist insbesondere das bereits erwähnte Reuse Model von Interesse.[19]

4.3.1 Die Berücksichtigung der Wiederverwendung

Wird ein Merkmal entsprechend der innerhalb des Abschnitts 4.1 vorgestellten Szenarien in irgendeiner Form Bestandteil einer Systemfamilie, so vollzieht sich

[19] vgl. zu folgenden Ausführungen [Sneed91] S.102; [Boehm00] S.64ff.

dies in der Regel mit dem Gedanken einer späteren Nutzung dessen. Im Zusammenhang mit Systemfamilien geschieht dies mit dem Ziel einer möglichst wirtschaftlichen Umsetzung verschiedenster Anwendungen und dahingehenden Einsparungsmöglichkeiten. Daher stellt sich die Frage, wie diese Nutzenaspekte innerhalb einer Aufwandsschätzung berücksichtigt werden können. Dazu ist es notwendig, auf die internen, formellen Zusammenhänge des COCOMOII Verfahrens einzugehen. Aufbauend auf den theoretischen Ausführungen des Unterabschnitts 3.3.3 soll das Potenzial der Wiederverwendung anhand des folgenden Beispiels aufgezeigt werden. Die dabei verwendeten Wertigkeiten dienen lediglich der Veranschaulichung der einzelnen Rechnungen und können von tatsächlichen Gegebenheiten mitunter abweichen.

Es wird davon ausgegangen, dass ein Merkmal in eine Systemfamilie integriert wurde. Das Merkmal hatte dabei einen Umfang von 100 LOC. Im Hinblick auf seine spätere Anwendungsentwicklung wird von folgenden Annahmen bezüglich des Programmquellcodes dieses Merkmals ausgegangen:

- 40% müssen aufgrund spezifischer Anforderungen stets neu erstellt werden.
- 30% können ohne zusätzlichen Aufwand übernommen werden.
- 30% müssen angepasst und modifiziert werden.

Ausgehend vom gegebenen Umfang von 100 LOC und dem zu erwartenden Anteil einer Neuerstellung des Programmquellcodes kann die folgende Gleichung angewandt werden:

$$LOC_1 = 0,4 * 100 = 40 LOC_1$$

Weiterhin müssen die 30% des Gesamtumfangs berücksichtigt werden, welche ohne weitere Aufwände übernommen werden. Im Sinne des Reuse Models entspricht dies dem adaptierbaren Quellcode:

$$LOC_2 = 0,30 * 100 = 30 LOC_2$$

Zusätzlich kann mit Hilfe des Reuse Models von Aufwand zur Administration und Verwaltung (AA) von wieder verwendbarer Software ausgegangen werden. Wird

entsprechend den Bewertungskriterien des Modells für AA der Wert 2 angenommen, ergibt sich die ergänzende Berechnung von LOC_2:

$$LOC_2 = 30 * [\frac{2 + 0 * (1 + (0,02 * 0 * 0))}{100}] = 0,6 LOC_2$$

Zur Berücksichtigung des Programmquellcodes, welcher nur nach gewissen Anpassungen innerhalb späterer Anwendungen genutzt werden kann, müssen zusätzliche Annahmen getroffen werden. Entsprechend der innerhalb des Unterabschnitts 3.3.3 aufgeführten Bestandteile gelten im Folgenden die Annahmen:

- Administration und Verwaltung wieder verwendbarer Software **AA = 2**,
- Die Verständlichkeit wieder verwendbarer Software **SU = 10**,
- Die Kenntnis der Software seitens der Entwickler **UNFM = 0.30**,
- Anteil des anzupassenden Designs **DM=10%**,
- Anteil des anzupassenden Quellcodes **CM=20%**,
- Anteil des Aufwands zur Integration **IM=20%**.

Ausgehend von den 30% Anteil am Gesamtumfang des Merkmals ergibt sich die Berechnung:

$$LOC_3 = 0,3 * 100 * [\frac{2 + (0,4 * 10 + 0,3 * 20 + 0,3 * 20) * (1 + (0,02 * 10 * 0,3))}{100}]$$
$$= 5,7 LOC_3$$

Als Ergebnis der aufgeführten Teilschritte entstehen somit insgesamt:

$$LOC_1 + LOC_2 + LOC_3 = 40 + 0,6 + 5,7 = 46,3 LOC_G$$

Der Aufwand für eine weitere Entwicklung, welche zusätzlich das neu in der Systemfamilie aufgenommene Merkmal beinhaltet, würde sich somit nur aus dem Umfang von 46 LOC zusammensetzen. Im Gegensatz dazu stehen die 100 LOC von komplett neu zu erstellenden Programmquellcodes, falls das Merkmal in Zukunft nicht enthalten ist.

4.3.2 Die Berücksichtigung zusätzlicher Aufwände

Berücksichtigt man allein die vorangegangene Betrachtung, so reicht dies in der Regel für eine sinnvolle Beurteilung der Schätzergebnisse aus. Jedoch sollen hier zusätzliche Aufwände berücksichtigt werden, welche besonders im Umfeld von Systemfamilien entstehen und gerade dortige Entwicklungen aufwendiger als einmalige Gesamtentwicklungen gestalten. Dazu werden drei der eigentlichen 17 Kostentreiber des Post Architecture Models genauer vorgestellt.[20]

Developed for Reusability (RUSE)
Dieser Kostentreiber berücksichtigt Aufwände, welche vorrangig durch Entwicklung wieder verwendbarer Software entstehen, wie zum Beispiel eine umfassendere Dokumentation der Software oder zusätzliche Tests und Anpassungen im Zusammenhang mit bestehender Software. Im Sinne der Problemstellung kann dieser Kostentreiber als überdurchschnittlich hoch angesehen werden.

Documentation Match to Life-Cycle Needs (DOCU)
Hier steht die explizite Berücksichtigung einer möglichst umfassenden Dokumentation im Vordergrund. Dies vollzieht sich im Kontext möglichst flexibler und vielgestaltiger Anpassungsmöglichkeiten der Software hinsichtlich sich ändernder Anforderungen. Im Zusammenhang mit Systemfamilien kann dieser Aufwand als überdurchschnittlich angesehen werden.

Required Software Reliability (RELY)
Der Kostentreiber beinhaltet die Sicherheit, welche die zu entwickelnde Software aufweisen muss. Dies umfasst beispielsweise das Maß, in dem sich ein Fehler auf andere Bestandteile der Software auswirkt. Innerhalb von Systemfamilien wird dieser Kostentreiber als hoch bis sehr hoch eingestuft. Die Anwendung und der Einfluss der Kostentreiber soll wiederum exemplarisch aufgezeigt werden.

Ein zu entwickelndes Merkmal umfasst einen voraussichtlichen Umfang von 100 LOC. Es ist zu entscheiden, ob dieses in eine Systemfamilie integriert werden soll und inwieweit dies aus ökonomischer Perspektive sinnvoll erscheint. Weiterhin wird davon ausgegangen, dass alle Kostentreiber einer durchschnittlichen Ausprägung unterliegen und damit vernachlässigt werden können. Lediglich für RUSE,

[20] vgl. [Boehm00]

DOCU sowie RELY werden im späteren Verlauf entsprechende Werte angenommen.

Den Ausgangspunkt für die Berechnung des Aufwands stellt die Basisgleichung des COCOMOII Post Architecture Models dar. Innerhalb der Betrachtung einer einmaligen Gesamtentwicklung ergibt sich folgender Aufwand (PM_E) für das hier behandelte Merkmal:

$$PM_E = 2,94 * 100^{1,1} = 466 PM_E$$

Werden nun zusätzliche Aufwendungen einer Entwicklung im Rahmen von Systemfamilien berücksichtigt, so können die Werte der Kostentreiber RUSE, DOCU und RELY als überdurchschnittlich angesehen werden. Daher werden exemplarisch die Werte RUSE=1,195; DOCU=1,11; RELY=1,18 angenommen. Der Aufwand einer derartigen Entwicklung (PM_S) erhöht sich mit Berücksichtigung der drei Kostentreiber auf:

$$PM_S = 2,94 * 100^{1,1} * 1,195 * 1,11 * 1,18 = 729 PM_S$$

4.4 Zusammenfassung der Ergebnisse

Es stellt sich nun letztendlich die Frage nach der Relevanz und Bedeutung der zurückliegenden Überlegungen. Dazu soll eine auf den innerhalb der Abschnitte 4.3.1 sowie 4.3.2 dargelegten Beispielen zusammenführende Betrachtung erfolgen.

Ausgehend von den Einsparungspotenzialen im Rahmen von Systemfamilien und den zusätzlichen Aufwendungen, welche durch diese hervorgerufen werden, ergibt sich aufbauend auf dem Programmquellcode für jede weitere Anwendungsentwicklung folgende Basisgleichung:

$$PM_Z = 2,94 * 46^{1,1} * 1,195 * 1,11 * 1,18 = 310 PM_Z$$

Werden nun mehrmalige Entwicklungen auf Basis des betrachteten Merkmals gegenübergestellt, ergibt sich folgende Tabelle:

Anzahl der Produkte	Einmalige Gesamtentwicklung ($\sum PM_E$)	Entwicklung für eine Systemfamilie ($PM_S + \sum PM_Z$)	Differenz
1	466	729	- 263 PM
2	932	1039	- 107 PM
3	1398	1349	+ 49 PM
4	1864	1659	+ 205 PM

Tabelle 4.1: Gegenüberstellung mehrmaliger Gesamtentwicklungen und einer Systemfamilie

Legt man die innerhalb des Abschnitts 4.2.1 dargelegten, mathematischen Zusammenhänge diesen Ergebnissen zugrunde, zeigt sich, dass das COCOMOII Verfahren die Erfordernisse einer Anwendungsentwicklung im Kontext von Systemfamilien erfüllen kann.[21] Die getroffenen Aussagen und Berechnungen sind somit für eine Betrachtung des Entwicklungsaufwands relevant und müssen als zusätzliche Arbeitsschritte in die innerhalb des Abschnitts 4.1 skizzierten Vorgehen integriert werden. Dies soll anhand der dargestellten Abläufe innerhalb der anschließenden Abbildung 4.6 erfolgen.

[21] siehe Abbildung 4.5 - Vergleich der Kosten

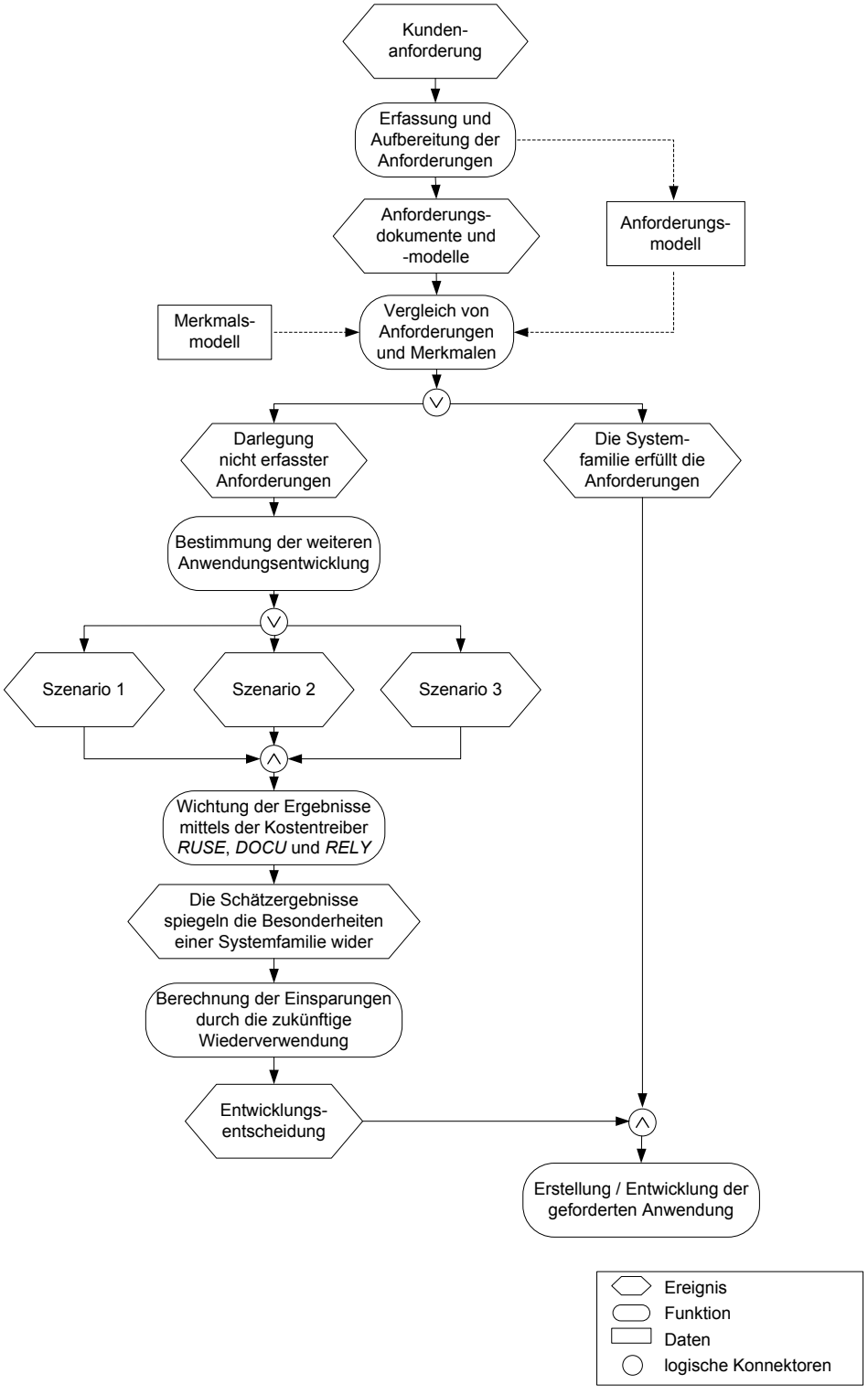


Abbildung 4.6: Das ganzheitliche Vorgehen zur Beurteilung einer Entwicklungsentscheidung

Anhand der zurückliegenden Ausführungen wird deutlich, dass die Kosten einer Softwareentwicklung und damit einhergehende Ursachen und Zusammenhänge komplexe Sachverhalte darstellen. Eine Entscheidung hinsichtlich einer geplanten Entwicklung muss zahlreiche Aspekte berücksichtigen. Daher erlangt eine Dokumentation und die Möglichkeit eines vielgestaltigen Informationszugriffs eine wichtige Bedeutung. Im Sinne einer konkreten Umsetzung umfasst dies ein Datenformat, dessen Integration in vorhandene Arbeitsabläufe sowie dessen Aufbereitungsmöglichkeiten. Damit beschäftigen sich die nun folgenden Ausführungen.

4.5 Die eXtensible Markup Language und ihre Anwendung

Um das in Abschnitt 4.1 dargelegte Vorgehen zu verwirklichen, liegt nun der Fokus auf Möglichkeiten, welche einen sinnvollen Einsatz von Daten der einzelnen Arbeitsschritte gewährleisten. Die Ergebnisse der Anforderungsanalyse müssen erfasst werden und zwar in der Form, dass eine weitere Nutzung dieser im Sinne der dargestellten Szenarien möglich wird. Im Idealfall können für eine Aufwandsschätzung notwendige Informationen direkt aus den bestehenden Daten der Anforderungsanalyse übernommen und anschließend wiederum in diese integriert werden. In diesem Zusammenhang besteht die Notwendigkeit nach einem Datenformat, auf das durch verschiedene Anwendungen zugegriffen werden kann, welches aber dennoch eine umfassende Beschreibung aller notwendigen Informationen erlaubt. Aktuelle Entwicklungen nutzen bezüglich dessen die *„eXtensible Markup Language"* (XML), welche im Anschluss beschrieben werden soll. Dabei wird neben den grundlegenden Eigenschaften der Auszeichnungssprache auf bestehende, für die Lösung relevante Entwicklungen eingegangen.

XML: Bestandteile und Eigenschaften

Zunächst ist es notwendig die hier verwendeten Begriffe zu klären, um ein Verständnis der im weiteren Verlauf der Arbeit angestrebten Lösung zu gewährleisten. Dabei werden bewusst lediglich hier relevante Ausschnitte betrachtet, da das gesamte Anwendungsfeld den Rahmen der für die Lösung heranzuziehenden

Aspekte sprengen würde. Vielmehr sollen wichtige Zusammenhänge hervortreten, welche für die Lösung entscheidend sind und die gewählte Herangehensweise rechtfertigen.[22]

Die Hauptfunktion XMLs ist die logische und inhaltliche Strukturierung von Daten. Der bezüglich dessen herangezogene Begriff der „Auszeichnungssprache" beschreibt:[23]

„*eine spezielle Notation zur Kennzeichnung verschiedener Bestandteile eines Dokumentes*".

Durch Gliederungselemente, welche „*Tags*" genannt werden, entstehen somit logische Gruppierungen und Zusammenhänge der erfassten Daten. Die Bezeichnung der Tags und deren syntaktischer Aufbau ist innerhalb einer „*Document Type Definition*" (DTD) definiert.[24] Eine XML-Datei beruht in der Regel auf einer derartigen DTD, welche eine auf einem bestimmten Einsatzzweck bezogene Auszeichnungen beinhaltet. Eine aktuelle Weiterentwicklung der DTDs stellen *XML-Schemas* dar. Diese erlauben eine präzisere Definition der Tags und können in der gleichen Art und Weise wie eine DTD verwendet werden.[25]

Im engeren Sinne ist XML also die strukturelle Auszeichnung, beziehungsweise das strukturelle *Markup* von Daten. Die wichtigsten Eigenschaften und damit verbundene Vorteile der Sprache sollen nachfolgend zusammengefasst werden:

- XML basiert auf Unicode. Dadurch können sämtliche Zeichen momentan bekannter Sprachen kodiert werden. Innerhalb XML erfasste Daten können somit international genutzt werden.
- Mit Hilfe von DTDs lassen sich beliebige individuelle Strukturierungen von Informationen definieren oder erweitern. Durch die meist logische Bezeichnung der Tags kann in der Regel auf den eigentlichen Inhalt geschlossen werden.

[22] vgl. zu folgenden Ausführungen [Sturm00] S.18ff.
[23] vgl. [Sturm00] S.13
[24] vgl. [Geese00] S.33ff.
[25] vgl. [W3CSchema]

- Innerhalb einer XML-Datei können beliebig viele Informationen aus anderen XML-Dateien eingebunden und dadurch neue, einem Einsatzzweck angepasste Informationen zusammengefasst werden.
- Neben dem Datenaustausch entstehen zahlreiche Anwendungen, welche sich hauptsächlich mit der Speicherung von Informationen beschäftigen.[26] Dies ist in der als relativ einfach gewerteten Handhabung der Daten selbst begründet.

Im weiteren Sinne umfasst die XML jedoch zusätzlich Möglichkeiten zur Aufbereitung und Darstellung der Daten. Dazu existieren standardisierte Schnittstellen sowie auf XML aufbauende Sprachen, welche die in einer XML-Datei enthaltenen Informationen in ein Präsentationsformat überführen können. Als Beispiel sei hier die „*eXtensible Stylesheet Language*" (XSL) aufgeführt, während eine weitere Möglichkeit am Beispiel der Entwicklung des Prototyps verdeutlicht wird.[27] XSL dient der gezielten visuellen Aufbereitung von zunächst in reiner Textform vorliegender Daten.[28] Der in der Verwendung resultierende Vorteil ist, dass die innerhalb einer XML-Datei enthaltenen Daten nicht verändert werden, jedoch beliebige Darstellungen dieser möglich sind. Ein in diesem Zusammenhang zu verwendender Begriff ist der des „*Cross-Media-Publishing*". Dieser umfasst verschiedenste zielgruppenbezogene Publikationen auf Basis ein und derselben XML-Datei. Innerhalb einer praktischen Umsetzung existieren dafür zahlreiche Werkzeuge und Hilfsmittel, welche, soweit bestimmte Voraussetzungen erfüllt sind, eingesetzt werden können. Zur Verdeutlichung der dargelegten Zusammenhänge ist in Abbildung 4.7 eine XML-Datei, der dabei genutzte Teil einer DTD und die Überführung der Daten in eine Präsentationsform mittels der XSL an einem Beispiel skizziert.[29]

[26] vgl. [Enterra02]
[27] siehe Abschnitt 5.1.1 - Das Document Object Model
[28] vgl. [Kay01] S.11ff.
[29] in Anlehnung an [Kay01] S.11

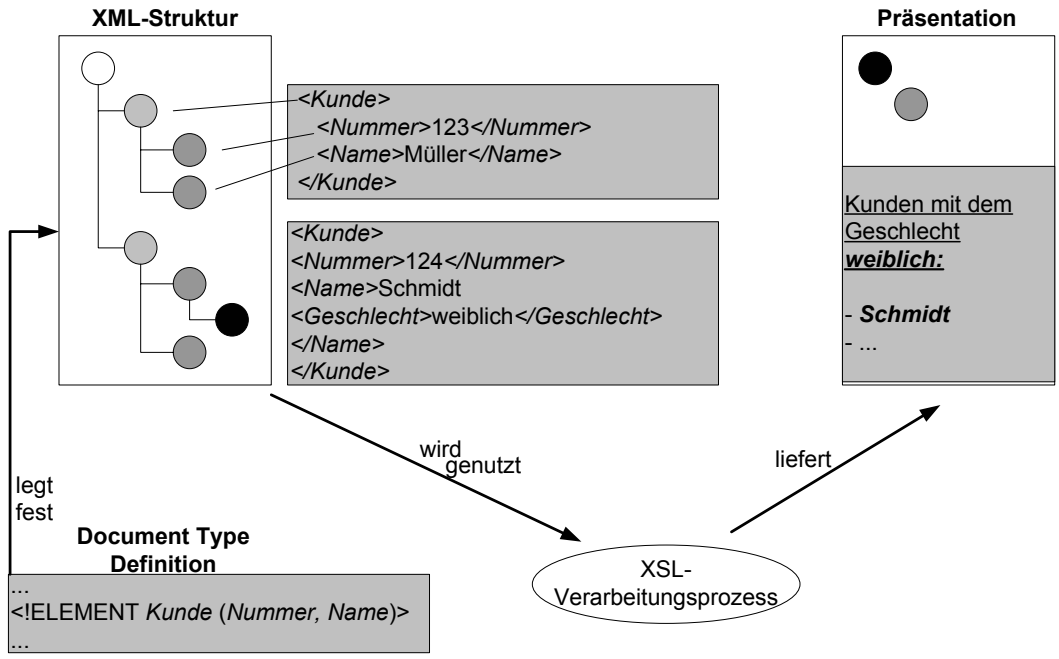


Abbildung 4.7: Zusammenspiel XML-DTD-XSL

Den Ausgangspunkt des dargestellten Beispiels stellt die Definition des Elements *Kunde* innerhalb der DTD dar. Dabei wird festgelegt, dass dieses die Elemente *Nummer* und *Name* beinhaltet. Somit wird die syntaktische Struktur einer auf dieser DTD aufbauenden XML-Anwendung fest vorgeschrieben. Inwieweit die beiden Elemente *Nummer* und *Name* zu verwenden sind, ist zusätzlich in der DTD zu definieren. Zur weiteren Verwendung der in der XML-Datei enthaltenen Daten wird im dargestellten Beispiel ein XSL-Prozess beschrieben. Innerhalb dessen wird auf die in der XML vorliegenden Daten zurückgegriffen, um diese wiederum in ein anderes Format zu übertragen. Damit dies möglich wird, ist die Kenntnis der zu nutzenden XML-Struktur notwendig, welche ihrerseits durch eine DTD vorgeschrieben ist. Zudem können bestehende Werkzeuge zur automatisierten Überführung der XML in ein Präsentationsformat nur genutzt werden, wenn die Struktur der XML bekannt ist. Der Einsatz von Werkzeugen ist notwendig, da eine alleinige, individuelle Überführung durch zu erstellende Lösungen mittels der XSL, eine vollständige Kenntnis des jeweiligen Präsentationsformats voraussetzt,

was zumeist nicht gegeben ist.

Die innerhalb dieses Kapitels angestrebte Lösung muss demnach auf Entwicklungen aufbauen, welche eine XML-Struktur vorweisen, die eine Unterstützung durch aktuelle Hilfsmittel und Werkzeuge gewährleistet. Dazu wird im Anschluss zunächst „*DocBook*" als entstehender Standard für Präsentationen analysiert und vorgestellt, während daraufhin die Dokumentenstruktur eines im Zusammenhang mit dem „*Family Oriented Requirements Engineering*" (FORE) Projekt stehenden Datenformats, als insbesondere im Bereich der Systemfamilien nutzbare Lösung, betrachtet wird.

4.5.1 DocBook - Dokumentation von Soft- und Hardware

Der strukturelle Aufbau der innerhalb einer Softwareentwicklung anfallenden Dokumente gleicht sich in den meisten Fällen. Die für die Gliederung eines Dokuments verwendeten Elemente lassen sich im Sinne eines strukturierten Markups mit Hilfe der XML darstellen. Einen dahingehend angestrebten Standard stellt die SGML-basierte DTD „DocBooks" dar, welche im wesentlichen die hierarchische Gliederung eines Buches umfasst.[30]

Obwohl die Entwicklung DocBooks schon 1991 begann, liegt bis heute keine offiziell verabschiedete Version vor. Norman Walsh, ein Mitglied des gegenwärtig mit der Entwicklung betrauten „DocBook Technical Commitee", verfasste 1999 jedoch eine DTD als Vorschlag für den zu erstellenden Standard. Dieser und darauf aufbauende Veröffentlichungen, einhergehend mit einem wachsenden Anwendungsfeld der XML, sorgen für eine zunehmende Verbreitung DocBooks. Die im Folgenden getroffenen Aussagen zu Aufbau und Inhalt sowie den Möglichkeiten zur Weiterverarbeitung beruhen daher vorrangig auf den Veröffentlichungen durch Norman Walsh und darauf aufbauenden Entwicklungen.

Inhalt und Aufbau

Wie bereits beschrieben, beruht der Aufbau einer DocBook-basierten Anwendung auf den strukturellen Gegebenheiten eines Buches. Die dahingehend innerhalb

[30] vgl. zu folgenden Ausführungen [Walsh99]

der DTD definierten Elemente müssen in der gleichen Form als Auszeichnungen in der zugehörigen XML-basierten Anwendung verwendet werden. In Abbildung 4.8 sind grundlegende Zusammenhänge bezüglich DocBook und dessen Einsatz skizziert.

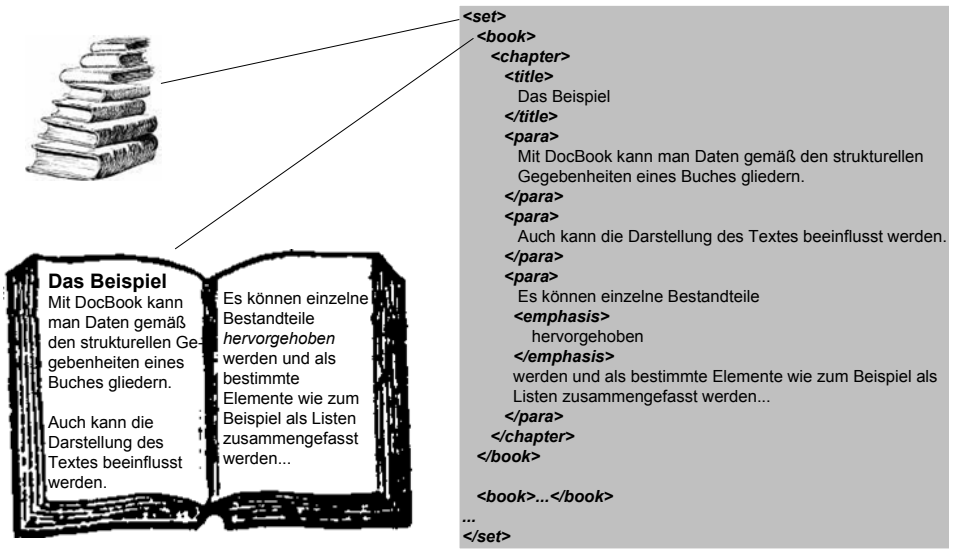


Abbildung 4.8: Ursprung und Anwendung DocBook's

Mit Hilfe von DocBook können demnach komplette Dokumente verfasst und anschließend mittels aktueller Werkzeuge, welche in Zusammenhang mit XML bestehen, aufbereitet werden. Es stellt sich nun jedoch die Frage nach der Anwendbarkeit hinsichtlich der Beschreibung stark technikbasierter Daten, wie sie im Zusammenhang mit Systemfamilien und Aufwandsschätzungen auftreten.

4.5.2 FORE - Datenformat für Systemfamilien

Im Rahmen einer Anwendungsentwicklung auf Basis einer bestehenden Systemfamilie rückt der Vergleich der von einem Kunden hervorgebrachten Anforderungen und der in der Systemfamilie enthaltenen Merkmalen in den Vordergrund. Das

„Family Oriented Requirements Engineering" (FORE) Projekt berücksichtigt diese besondere Bedeutung der Anforderungsanalyse.[31] Das darin enthaltene XML-basierte Datenformat hat zum Ziel, alle Anforderungen in Form von Merkmalen[32] sowie die Merkmale einer Systemfamilie zu erfassen. Dabei können die Zusammenhänge zwischen diesen Merkmalen in einer Form beschrieben werden, dass eine direkte Ableitung von Modellen möglich ist. Der strukturelle Aufbau einer auf FORE aufbauenden Anwendung ist an den Modellierungsvorgaben FODAs angelehnt und in einem XML-Schema vorgegeben. Die Vorteile, welche sich daraus ergeben werden im Folgenden anhand des Aufbaus dargelegt.

Inhalt und Aufbau

Die zentralen Kriterien des Formates sind die darin abbildbaren Anforderungen seitens eines Kunden sowie Merkmale, welche aufgrund dieser für eine Entwicklung in Betracht gezogen werden. Dazu existieren die Gliederungselement „RequirementModel" und „FeatureModel". Die darin enthaltenen Anforderungen beziehungsweise Merkmale werden durch die Tags „FeatureType" und „RequirementType" beschrieben. Ein weiterer wichtiger Bestandteil des Datenformats ist das Element „RelationType". Innerhalb dessen können die Beziehungen beschrieben werden, welche zwischen den beschriebenen Inhalten der Tags „RequirementModel" und „FeatureModel" bestehen.

Der Vorteil in der Anwendung eines XML-Schemas ist, dass ein Element anhand eines Namens und eines Typs sowie seiner wertmäßigen Ausprägung definiert werden kann. Dem Typ eines Elements können wiederum ein oder mehrere Attribute zugewiesen werden oder selbst andere Elemente. Daher wird im Allgemeinen in der Verwendung eines XML-Schemas nicht von Elementen, sondern von Datentypen gesprochen. Durch die genaue Definition der einzelnen Datentypen ist somit die Richtigkeit der zugrunde liegenden XML-Anwendung durch den direkten Bezug zu dem XML-Schema gewährleistet.

[31] vgl. zu folgenden Ausführungen [Streitferdt03]

[32] siehe Abschnitt 2.3.1 - Die Erfassung und Modellierung von Merkmalen

4.5.3 Anwendung im Umfeld von Systemfamilien und Aufwandsschätzung

Die Beschreibung der zurückliegenden Datenformate verdeutlicht zum einen die Relevanz der eXtensible Markup Language und zum anderen zeigt sie die verschiedenen Ansätze und Standardisierungsbemühungen in diesem Umfeld. Anhand des Datenmodells des FORE Projekts wird deutlich, dass eigene spezifische Lösungen oft sinnvoller an individuelle Bedürfnisse anpassbar sind. Daher wird hier von der Verwendung eines allgemeinen Datenformats abgesehen, wie es DocBook letztendlich darstellt. Vielmehr sollen die Bestandteile des FORE Formats und dabei insbesondere die darin beschreibbaren Anforderungen an ein Softwareprojekt genutzt werden.

Dabei liegt der Fokus nicht zwingend auf dem Format selbst. Vielmehr wären Ansätze denkbar, welche auf bestehenden XML-Anwendungen aufbauen. Mittels eigener Entwicklungen könnte direkt auf eine XML-Datei, welche gemäß des FORE-Schemas aufgebaut ist, zugegriffen werden. Die strukturierten Anforderungen können entsprechend einer gewünschten Granularität in eine Aufwandsschätzung überführt werden. Dabei gilt es zunächst, entsprechend dem in Abschnitt 4.1 dargestellten Vorgehen, den Umfang einer Anforderung zu bestimmen. Die dahingehend dargestellten Möglichkeiten sind jedoch derart vielgestaltig, dass hier von der Kenntnis des zu erwartenden Projektumfangs ausgegangen wird.

Inwieweit eine derartige Lösung bewirkt werden kann, soll anhand der folgenden prototypischen Umsetzung dargelegt werden.

5 Die Umsetzung der Lösung am Beispiel eines Prototyps

Auf den vorherigen Kapiteln aufbauend soll nun eine konkrete Umsetzung einer Aufwandsschätzung im Zusammenhang mit Systemfamilien exemplarisch vorgestellt werden. Ähnlich der Herangehensweise im beschriebenen Lösungsansatz erfolgt auch hierbei eine zweiseitige Betrachtung der Problemstellung. Zum einen gilt es, an die für eine Aufwandsschätzung relevanten Daten zu gelangen. Zum anderen ist die Umsetzung des beschriebenen Vorgehens des COCOMOII Verfahrens notwendig.

Zunächst liegt der Fokus auf dem Datenformat FORE als Ausgangspunkt für alle weiteren Arbeitsschritte. Dabei wird das „*Document Object Model*" (DOM) als mögliche Schnittstelle für einen gezielten Zugriff vorgestellt. Im Anschluss werden die erlangten Daten einem detaillierten Aufwandsschätzprozess übergeben. Innerhalb dessen sollen die dargelegten Szenarien einer Anwendungsentwicklung im Kontext von Systemfamilien erfasst und umgesetzt werden. Das angestrebte Ziel ist hier somit, alle notwendigen Arbeitsschritte in Verbindung mit einer möglichst einfachen Handhabung auf Nutzerseite aufzuzeigen.

5.1 Der Zugriff auf XML

Die Grundlage für eine Aufwandsschätzung bildet die Kenntnis der Anforderungen an die zu entwickelnde Software. Aufgrund seines spezifischen Anwendungsfelds erscheint das Datenformat FOREs und die darin erfassbaren Anforderungen

für eine weitere Verarbeitung besonders geeignet. Vor allem dessen Ursprung in der eXtensible Markup Language eröffnet umfassende Möglichkeiten des Zugriffs und Verarbeitung der Daten. Bezüglich dessen soll hier das Document Object Model (DOM) hervorgehoben werden und im Prototyp Verwendung finden.

5.1.1 Das Document Object Model (DOM)

Eine zentrale Stellung in der Entwicklung von XML und darauf aufbauender Entwicklungen nimmt das „World Wide Web Consortium" (W3C) ein. Unter dessen Schirmherrschaft entstanden die allgemein akzeptierten, derzeitig gültigen Standards. Für das Verständnis DOMs ist die Definition XMLs durch das W3C von Interesse:[1]

„*Die Extensible Markup Language beschreibt eine Klasse von Datenobjekten, genannt XML-Dokumente, und beschreibt teilweise das Verhalten von Computer-Programmen, die solche Dokumente verarbeiten.*"

In der Betrachtung von XML kann somit von einer objektorientierten Dokumentenstruktur ausgegangen werden. DOM stellt letztendlich eine plattform- und sprachunabhängige Schnittstelle für den Zugriff auf diese Dokumentenstruktur dar. Es beinhaltet Funktionalitäten zur Manipulation des Inhalts und der Struktur eines XML Dokuments.[2] Diese spiegelt sich in spezifischen Objekten in Form einer Baumstruktur wider. In Abbildung 5.1 ist dieser Zusammenhang an einem Beispiel skizziert.

[1] vgl. [W3C]
[2] vgl. [Sturm00] S.193

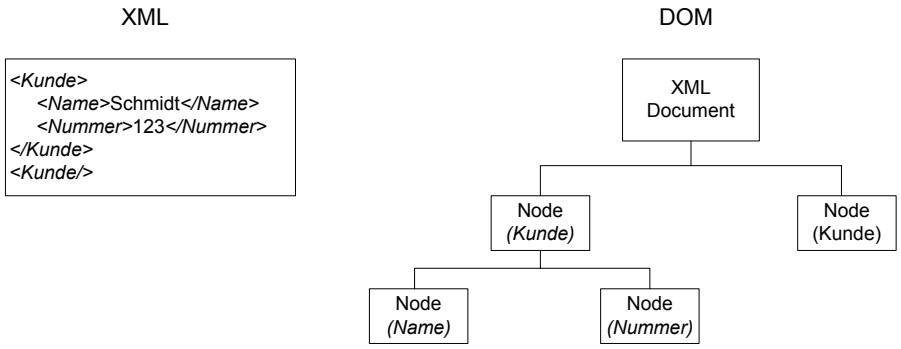


Abbildung 5.1: Zusammenhang XML und DOM

Aus den einzelnen Elementen einer XML leiten sich die Knoten[3] des dargestellten Baumes ab. Ein Knoten ist dabei als ein Verweis auf ein Objekt anzusehen,[4] welches in einer Dokumentenstruktur existieren kann. Auf jeden Knoten kann demnach, mittels bestimmter Methoden, zugegriffen werden.

In der konkreten Umsetzung des Prototyps wird auf das von der Firma Microsoft entwickelte „*XML-Document Object Model*" zurückgegriffen. Dieses stellt eine konkrete Ausprägung der vom W3C verabschiedeten Spezifikation DOMs dar.

5.1.2 Die Umsetzung im Prototyp

Das Ziel in der Verarbeitung des FORE-Datenformats ist der Zugriff auf die Elemente, welche die Anforderungen eines Kunden beinhalten. Im Folgenden wird die im Prototyp implementierte Vorgehensweise vorgestellt, ausgehend von einem XML-Dokument hinzu den darin enthaltenen Anforderungen.[5]

Zunächst wird ähnlich Abbildung 5.1 einer gewählten XML-Datei das DOM-Objekt „*XML-DOMDocument*" zugewiesen. Von Interesse ist nun vorrangig der

[3] engl. „node"
[4] vgl. [Sturm00] S.194
[5] in Anlehnung an [Geese00]; [Sturm00]

Knoten, welcher dem Tag-Element „*RequirementsModel*" entspricht.[6] Um alle Unterelemente dieses Knotens zu erfassen, ist es notwendig, diesem das DOM-Objekt „*DOMNodeList*" zuzuordnen. Dieses als Liste zu behandelnde Objekt wird nun nach den einzelnen Knoten „Requirement" durchsucht, welche dem Objekt „*DOMNode*" entsprechen. Dazu wurde eine rekursive Funktion entwickelt, welche jedes einzelne Objekt nach dem Vorhandensein weiterer Unterelemente überprüft und im gegebenen Fall eine erneute Liste der Funktion übergibt. Im Ergebnis werden alle Anforderung als Baumstruktur innerhalb des Prototyps dargestellt. Dadurch wird zum einen deutlich, wie sich der strukturelle Aufbau der Anforderungen gestaltet. Zum anderen können gezielte Anforderungen einzeln durch einen Nutzer geschätzt werden. In Abbildung 5.2 sind die dargelegten Zusammenhänge zur Verarbeitung der XML-Daten zusammenfassend skizziert.

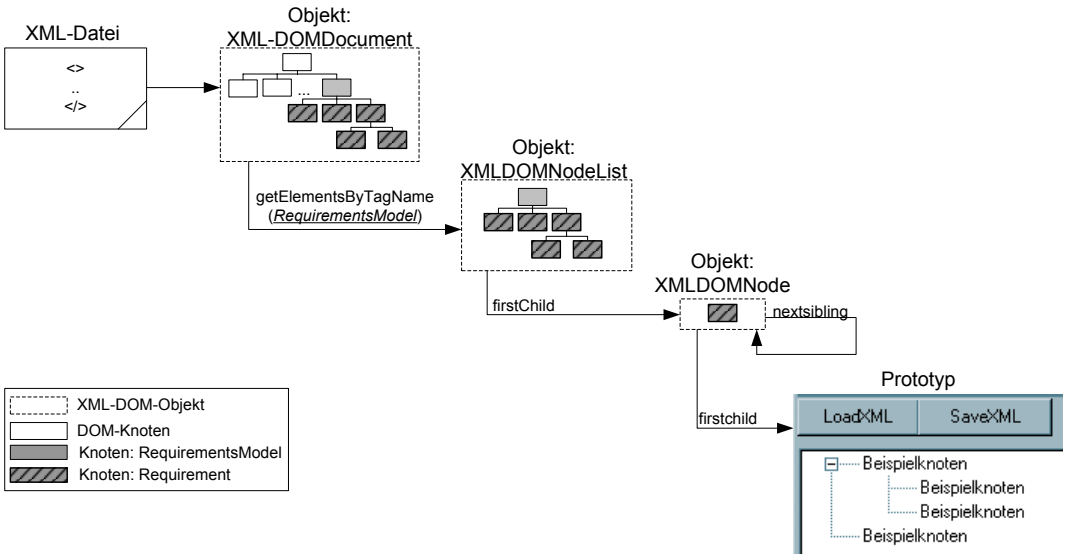


Abbildung 5.2: Nutzung von DOM-Objekten und deren konkrete Umsetzung

Steht der voraussichtliche Umfang der gewählten Anforderung und der zugehörige Aufwand in Personenmonaten fest, werden dem dazu gehörenden XML-Element die Unterelemente beziehungsweise Tags „LOC" sowie „PM" zugeordnet. Die Vor-

[6] siehe Abschnitt 4.5.2 - FORE - Datenformat für Systemfamilien

aussetzung dafür ist eine Aufwandsschätzung, deren Implementierung im Anschluss erläutert wird.

5.2 Die Implementierung der Aufwandsschätzung

Das Problem in der Umsetzung einer Aufwandsschätzung ist der Zwiespalt zwischen einer angestrebten größt möglichen Genauigkeit der Schätzergebnisse und den komplexen Arbeitsschritten zur Durchsetzung dieser. Allein die Ausgangsformeln des COCOMOII Verfahrens sowie alle dazu notwendigen Einflussfaktoren, Kostentreiber und Skalengrößen müssen bestimmt und korrekt zugeordnet werden. Das COCOMOII Verfahren setzt in seiner Anwendung somit spezifisches Fachwissen der einzelnen Arbeitsschritte voraus, um verlässliche Aussagen der Schätzergebnisse erreichen zu können.

5.2.1 Grundlegende Funktionalitäten der implementierten Aufwandsschätzung

Der entwickelte Prototyp erlaubt sowohl schnelle und grobe Schätzungen, anhand gemäß des Umfelds der Systemfamilien angenommener Eingangsgrößen, als auch detaillierte Schätzungen, durch die Implementierung der wesentlichen und wichtigsten Arbeitsschritte des Verfahrens. Die Umsetzung erfolgte vorrangig im Hinblick auf dem innerhalb des Kapitels 4 erarbeiteten Lösungsansatz.

Zunächst gilt die Betrachtung der Bestimmung des voraussichtlichen Umfangs der Software. Dabei besteht die Möglichkeit diesen direkt einzugeben oder den Arbeitsprozess zur Bestimmung der *Ungewichteten Function Points* zu verfolgen. Zweiteres liefert zwar genauere Eingangsgrößen, setzt jedoch die Kenntnis der darin enthaltenen Werte voraus. In Abbildung 5.3 sind beide Möglichkeiten skizziert.

Abbildung 5.3: Die Wahl zwischen einer Expertenschätzung und der Bestimmung der ungewichteten Function Points

Ähnlich wurde mit der Berücksichtigung wieder verwendbarer Softwarebestandteile im Rahmen der gewählten Entwicklung verfahren. Entweder wird der Anteil der wieder zu verwendenden Software als mögliches Ergebnis einer Expertenschätzung direkt bestimmt oder es wird mit Hilfe des notwendigen Fachwissens das Reuse Model des COCOMOII Verfahrens durchgeführt. Zur Bestimmung der einzelnen Größen des Reuse Models wurden die allgemeinen Tabellen und Richtlinien zur Bestimmung der Wertigkeit direkt in den Prototypen übernommen. Im Falle einer einfachen Bestimmung des Softwareanteils wird mit Hilfe dem Anwendungsumfeld von Systemfamilien entsprechenden, angenommenen Werten weiter gerechnet. Abbildung 5.4 beinhaltet eine Umsetzung der Bestimmung des Faktors „*Software Understanding*" als Beispiel für alle anderen Bestandteile des Formelwerks des Reuse Models.

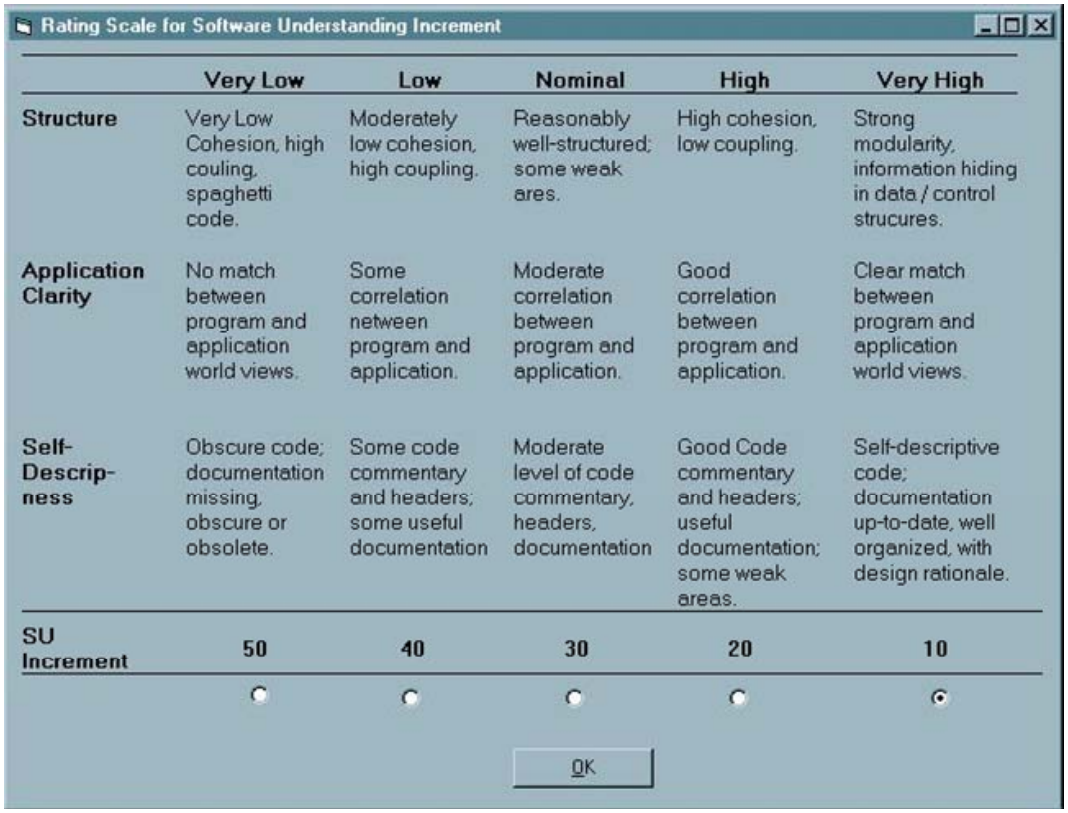


Abbildung 5.4: Beispiel einer Eingabemaske zur Umsetzung des COCOMOII Reuse Models

Zur Bestimmung des Aufwands wurde eine Funktion ermittelt, welche anhand des letztendlich ermittelten Umfangs des Programmquellcodes den notwendigen Aufwand in Personenmonaten berechnet. Dazu wurde die Ausgangsgleichung des COCOMOII Verfahrens herangezogen. Zur Verdeutlichung der verschiedenen umgesetzten Ansätze werden sowohl die Ergebnisse ohne als auch die Ergebnisse mit Berücksichtigung der Wiederverwendung dargestellt. Somit wären bei diesem Stand der Schätzung schon Aufwände bestimmbar. Jedoch sind die innerhalb des Abschnitts 4.3.1 beschriebenen Erweiterungen noch unberücksichtigt.

5.2.2 Die Implementierung weiterführender Funktionalitäten

Somit wurde eine weitere Eingabemaske erstellt, welche die Ermittlung bestimmter Kostentreiber erlaubt und die bisherigen Aufwände je nach ihrer Ausprägung korrigieren lässt. Abbildung 5.5 hat die Umsetzung der innerhalb des Abschnitts 4.3.3 dargelegten Kostentreiber RUSE, RELY und DOCU zum Inhalt.

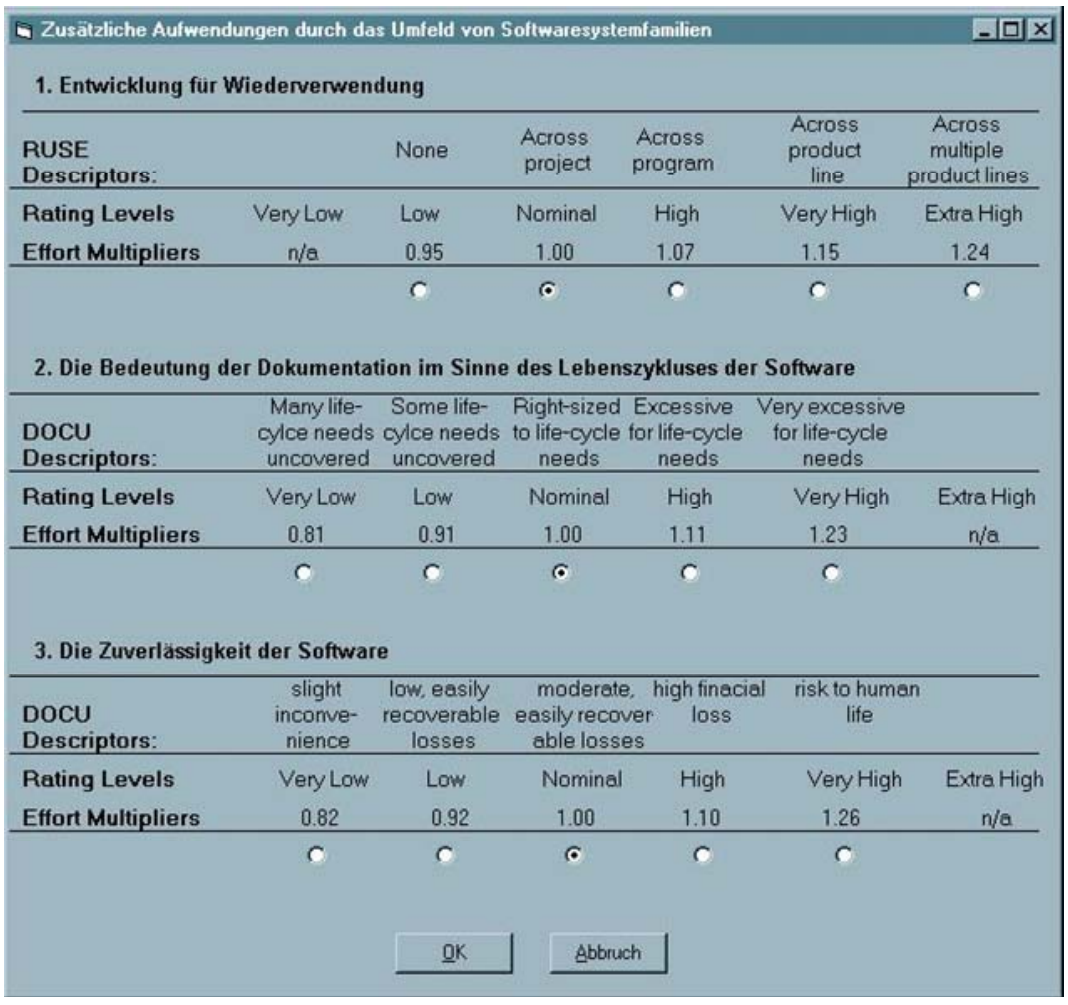


Abbildung 5.5: Die Ermittlung zusätzlicher Einflüsse im Umfeld von Systemfamilien

Zur fundierten Beurteilung des Aufwands im Kontext von Systemfamilien müssen zukünftige Entwicklungen in die Betrachtung mit einfließen. Ein weiterer Be-

standteil des Prototyps ist daher eine Bildschirmmaske, welche eine Beurteilung voraussichtlicher Wahrscheinlichkeiten zukünftiger Entwicklungen erlaubt. Dabei wird vorrangig auf das beschriebene Reuse Model zurückgegriffen, welches in der in Abschnitt 4.4 dargelegten Form Verwendung findet. Die einzelnen Werte ergeben sich wiederum aus dem implementierten Reuse Model. Die folgende Abbildung 5.6 stellt diese Bildschirmmaske dar.

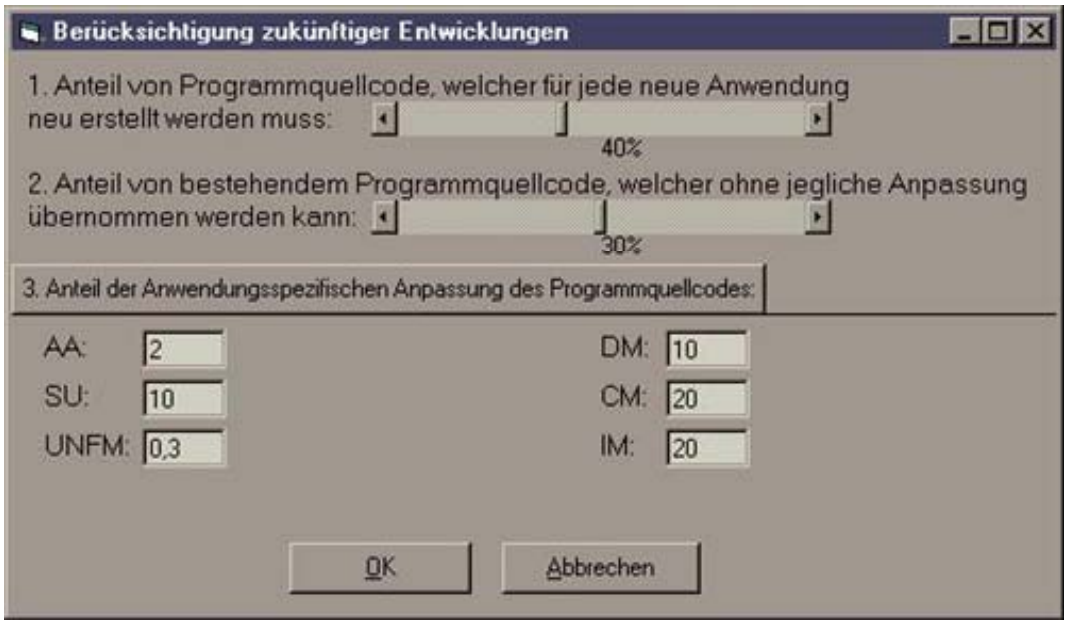


Abbildung 5.6: Bildschirmmaske zur Berücksichtigung zukünftiger Aufwände

Abschließend werden die errechneten Aufwände zusammenfassend dargestellt. Ein Nutzer hat somit die Möglichkeit, entsprechend seiner eigenen ökonomischen Rahmenbedingungen, eine Alternative zu wählen. Der Aufwand sowie der Umfang der gewählten Anforderung werden im Ergebnis als neue Elemente beziehungsweise Tags in das eingangs gewählte XML-Dokument gespeichert. Die Struktur des ursprünglichen Elements „Requirement" wurde somit um die Elemente „LOC" und „PM" erweitert.

6 Zusammenfassung und Bewertung

Die vorliegende Arbeit hatte zum Ziel, eine Aufwandsschätzung für einzelne Anwendungen zu bestimmen, welche sich aus einer Systemfamilie ableiten lassen. In der thematischen Auseinandersetzung war dazu eine Zweiteilung der Arbeit notwendig. Zunächst wurden grundlegende Zusammenhänge und Abläufe in der Entwicklung und Anwendung einer Systemfamilie analysiert und aufbereitet. Dabei zeigte sich, dass hohen Investitionsrisiken in der eigentlichen Erstellung einer Systemfamilie hohen ökonomischen Einsparungspotenzialen zukünftiger Anwendungsentwicklungen gegenüberstehen.

Im weiteren Verlauf wurde von der Existenz einer Systemfamilie ausgegangen und die Anwendungsentwicklung in diesem spezifischen Umfeld näher untersucht. Insbesondere die Fälle, in denen Anforderungen auf Kundenseite nicht durch bestehende Elemente der Systemfamilie abgedeckt werden, waren dabei von Interesse. In diesem Zusammenhang lag der Fokus vorrangig auf der Anforderungsanalyse und der darin enthaltenen Erfassung von Anforderungen und der weiteren Verarbeitung dieser. Es wurde der Ablauf einer derartigen Entwicklung untersucht und die Ergebnisse, wie beispielsweise spezifische Modelle, vorgestellt. Es zeigte sich, dass dem Zusammenspiel der Anforderungsanalyse einer Anwendungsentwicklung und der Konsequenz für die Zusammenhänge in der Systemfamilie eine wichtige Bedeutung beizumessen ist. Vor allem ökonomische Belange sollten daher in den weiteren Ausführungen von Interesse sein.

Die weitere Betrachtung galt den Möglichkeiten einer Aufwandsschätzung als Grundlage für ökonomisch begründbare Entscheidungen hinsichtlich einer Anwendungsentwicklung im Kontext von Systemfamilien. Im Sinne einer allgemeinen Gültigkeit der Ausführungen und über die vorliegende Arbeit hinausgehender Ansätze wurde neben anwendbaren Verfahren auf grundlegende Zusammenhänge und Ursprünge einer Aufwandsschätzung eingegangen. Im Ergebnis zeigte sich, dass eine Aufwandsschätzung komplexe Arbeitsschritte beinhaltet und spezifisches Fachwissen für seine Verwendung voraussetzt. Der Aufwand der Aufwandsschätzung selbst erscheint im Hinblick auf die wirtschaftlichen Aufwendungen in Verbindung mit einer Systemfamilie gerechtfertigt. Daraus entstand die Zielsetzung zur Integration eines konkreten Aufwandsschätzverfahrens in die beschriebene Anforderungsanalyse im Umfeld einer Systemfamilie. Aufgrund seiner Aktualität und freien Verfügbarkeit sowie seiner allgemein akzeptierten Qualität wurde das COCOMOII Verfahren ausgewählt.

In einem zu erarbeitenden Lösungsansatz wurden die Arbeitsschritte einer Anforderungsanalyse durch ein speziell angepasstes Vorgehen des COCOMOII Verfahrens ergänzt. Voraussetzung dafür war die Unterscheidung verschiedener Szenarien bezüglich ihres Auftretens innerhalb einer Anwendungsentwicklung im Kontext von Systemfamilien. Daher wurde für jedes dieser Szenarien ein Vorgehensmodell zur fallspezifischen Aufwandsschätzung entwickelt. In einer anschließenden Analyse der Ergebnisse zeigte sich jedoch, dass wirtschaftliche Belange im Zusammenhang mit einer Systemfamilie nur unzureichend durch die grundsätzlichen Abläufe des COCOMOII Verfahrens abgedeckt werden. Somit wurden eine Anpassung beziehungsweise zusätzliche Abläufe zur Bestimmung des endgültigen Aufwands notwendig. Anhand interner, formeller Zusammenhänge des COCOMOII Verfahrens wurde ein weiteres Vorgehen entwickelt. Das Ergebnis bildet wiederum ein Vorgehensmodell, welches die bisher dargestellten Modelle integriert und erweitert. Anhand einer vorhergehend durchgeführten Beispielrechnung wurden die gewählten Arbeitsschritte im Kontext ermittelter Gesetzmäßigkeiten von Systemfamilien nachgewiesen.

Weiterhin stellte sich die Frage nach der weiteren Verwendung der geschätzten Aufwände. Im Hinblick auf aktuelle Entwicklungen wurde dazu die eXtensible Markup Language und dahingehend existierende Entwicklungen untersucht. Da-

bei zeigte sich, dass das innerhalb des FORE Projekts entwickelte Datenformat sowohl eine Grundlage als auch eine optimale Möglichkeit zur weiteren Verwendung der Schätzergebnisse darstellt. Die in dem XML-basierten Format enthaltenen Anforderungen können direkt für eine Aufwandsschätzung übernommen und dem erarbeiteten Vorgehen übergeben werden. Dabei entstehende Ergebnisse können wiederum in das bestehende Format integriert werden.

Abschließend erfolgte eine prototypische Umsetzung der erarbeiteten Lösung. Hierbei zeigte sich vor allem der Zwiespalt zwischen einer möglichst hohen Schätzgenauigkeit und den damit verbundenen hohen Anforderungen an den Benutzer des Prototyps. Die einzelnen Arbeitsschritte des COCOMOII Verfahrens sind umfangreich und setzen umfassendes Fachwissen voraus. Daher wurde zur Implementierung sowohl auf einfache Eingaben als auch auf die vollständige Berechnung aller Parameter zurückgegriffen. Somit liegt der Kompromiss zwischen hoher Schätzgenauigkeit und einfacher Bedienbarkeit in der Verantwortung des Benutzers. Der Prototyp erlaubt Einblicke in die Abläufe des COCOMOII Verfahrens und berücksichtigt alle im Lösungsansatz erarbeiteten Zusammenhänge.

Insgesamt betrachtet, lässt sich feststellen, dass die ursprünglich angestrebte Zielstellung umgesetzt wurde. Jedoch müssen bezüglich derer einige Einschränkungen vorgenommen werden. So stellt der Ansatz der Systemfamilien einen nicht einheitlich verfolgten Forschungsansatz dar. Die Gültigkeit der erbrachten Lösung soll daher auf die im Laufe der Arbeit dargelegten Modellierungs- und Datenkonstrukte beschränkt werden. Weiterhin sind bei einem praktischen Einsatz einer Aufwandsschätzung die organisatorischen Rahmenbedingungen zu beachten. Aufgrund der hohen Aufwendungen, welche in der betrieblichen Integration der Verfahren und dahingehend notwendiger Voraussetzungen begründet liegen, kann die Wirtschaftlichkeit einer Aufwandsschätzung nicht für alle Einsatzbereiche garantiert werden. Vor allem der Einsatz, der in dieser Arbeit erbrachten Lösung, muss wegen ihrer hohen Spezifität geprüft und gegebenenfalls den Belangen veränderter Rahmenbedingungen angepasst werden.

7 Ausblick

Bezüglich der vorgenommenen Einschränkungen bietet sich das Potenzial für auf dem Lösungsansatz und dem Prototyp aufbauenden Entwicklungen. Insbesondere die dargelegte Kritik an der Komplexität des COCOMOII Verfahrens ist als Ansatzpunkt für Verbesserungen zu sehen. So sollten die im Prototyp herangezogenen, angenommenen Eingangswerte des vereinfachten Vorgehens hinsichtlich ihrer Gültigkeit empirisch überprüft werden. Sind dahingehend einheitliche Größen erkennbar, können die umfangreichen Arbeitsschritte des Verfahrens durch deren Anwendung vereinfacht werden.

Weiterhin sollten andere Aufwandsschätzverfahren hinsichtlich ihrer Tauglichkeit geprüft werden. Zwar bietet das COCOMOII Verfahren aufgrund seiner langen Entwicklung und empirischen Absicherung eine hohe Verlässlichkeit der Ergebnisse, jedoch sollten aufgrund der Spezifität des Anwendungsfeldes von Systemfamilien weitere Verfahren in Betracht gezogen und analysiert werden.[1]

Zudem stellt sich die Frage, inwieweit das Datenformat des FORE Projekts erweitert werden kann und inwieweit die Ergebnisse einer Aufwandsschätzung im Sinne der aufgezeigten Analogiemethode für andere Aufwandsschätzungen genutzt werden können. Eventuell erscheint hinsichtlich einer geplanten, langfristigen Anwendung einer Aufwandsschätzung der erwähnte Aufbau einer Wissensbasis sinnvoll.[2] So ist eine Entwicklung aufbauend auf dem FORE-Datenformat denkbar, welches eine eigene Entwicklung der eXtensible Markup Language darstellt. Im Sinne einer breiten und praxisnahen Anwendung einer Aufwandsschätzung sind somit

[1] vgl. [Jones98] S.131ff.
[2] vgl. [Enterra02]

Entwicklungen hinsichtlich einer Vereinfachung der Verfahren und einer umfassenden Nutzbarkeit der Ergebnisse anzustreben. So sind Aufbereitungsmöglichkeiten des FORE-Datenformats hinsichtlich eines Kundenkontakts ebenso von Interesse, wie die Nutzung der Informationen für innerbetriebliche Prozesse. So können beispielsweise betriebsspezifische Umrechnungswerte für Zeit und Kosten in eine Berechnung mit einfließen und aufbereitet werden. Die darauf aufbauenden betriebswirtschaftlichen Einsatzmöglichkeiten sind derart vielgestaltig, dass sich zahlreiche Entwicklungen daraus ableiten lassen.

Der dargelegte Lösungsansatz und der daran anschließende Prototyp bieten somit die Grundlage potenzieller Weiterentwicklungen. Insbesondere die eXtensible Markup Language bietet dahingehende zahlreiche Möglichkeiten.

Literaturverzeichnis

[Balzert96] Balzert, Helmut : *Software-Entwicklung*; Heidelberg, Berlin, Oxford 1996

[Boehm00] Boehm, Barry W. [et al.]: *Software cost estimation with COCOMOII*; Prentice Hall PTR 2000.

[Böllert02] Böllert, Kai: *Objektorientierte Entwicklung von Software-Produktlinien zur Serienfertigung von Software-Systemen*; Dissertation im Fachgebiet Prozessinformatik der Technischen Universität Ilmenau: 2002.

[Bredemeier02] Bredemeier, Willi: *Die Entwicklung der deutschen Informationswirtschaft bis 2006 Ergebnisse einer Expertenumfrage*; Monitoring Informationswirtschaft 2. Trendbericht 2001/2002; Institute for Information Economics in Zusammenarbeit mit NFO Infratest und im Auftrage des Bundesministeriums für Wirtschaft und Technologie Hattingen: Februar 2002 verfügbar unter: http://www.bmwi.de/textonly/Homepage/download /infogesellschaft/MoniTrend2.pdf

[Buchholz96] Buchholz, Manfred: *Time-to-Market-Management, Zielorientierte Gestaltung von Produktinnovationsprozessen* Stuttgart, Berlin, Köln: Kohlhammer 1996

[Bundschuh00] Bundschuh, Manfred; Fabry, Axel *Aufwandsschätzung von IT-Projekten*; Bonn: MITP-Verlag 2000.

[Burghardt01] Burghardt, Manfred: *Einführung in Projektmanagement: Definition, Planung, Kontrolle, Abschluss; 3. Aufl*; Publicis MCD Verlag 2001.

[Carlton00] CARLTON, D. W.; PERLOFF, J. M.: *Modern industrial organization, 3. Aufl.*; The Addison-Wesley series in economics: Addison-Wesley 2000.

[Capilla01] Capilla, Rafael; Dueñas, Juan C. *Modelling Variability with Features in Distributed Architectures*; in: Software Product-Family Engineering 4th International Workshop; revised papers / PFE 2001, Bilbo Spain, October 3-5 2001; Frank van der Linden (ed.) - Berlin; Heidelberg; New York; u.a.: Springer 2002.

[CMU/SEI97] Bass, Len; Clements, Paul; Cohen, Sholom; Northrop, Linda; Withey, James: *Product Line Practice Workshop Report*; Software Engineering Institute, Carnegie Mellon University, Pittsburgh 1997; verfügbar unter: http://www.sei.cmu.edu/publications/documents/98.reports /98tr007/98tr007abstract.html.

[CMU/SEI99] Bass, Len; Campbell, Grady; Clements, Paul; Northrop, Linda; Smith, Dennis: *Third Product Line Practice Workshop Report*; Software Engineering Institute, Carnegie Mellon University, Pittsburgh 1999; verfügbar unter: http://www.sei.cmu.edu/publications/documents/99.reports /99tr003/99tr003abstract.html.

[Cohen02] Cohen, Sholom *Product Line State of the Practice Report*; Software Engineering Institute, Carnegie Mellon University, Pittsburgh 1999; verfügbar unter: http://www.sei.cmu.edu/pub/documents/02.reports/pdf/ 02tn017.pdf

[Coplien98] Coplien, James; Hoffman, Daniel; Weiss, David: *Commonality and Variability in Software Engineering* Institute of

Electrical and Electronics Engineers 1998 verfügbar unter: http://www.bell-labs.com/user/cope/Mpd/IeeeNov1998/

[Czarnecki00] Czarnecki, Krysztof; Eisenecker, Ulrich W.: *Generative Programming. Methods, Tools and Applications*; Addison-Wesley Professional 2000.

[Czarnecki01] Czarnecki, Krysztof; Eisenecker, Ulrich W.: *Management von Softwaresystemfamilien*; aus: OBJEKTspektrum 2/2001 S. 56ff..

[Enterra02] Enterra Software GmbH: *Wissensmanagement für KMU als Motor für die Zukunft* in aboutIT Ausgabe 44/2002 verfügbar unter: www.aboutit.de/02/44/07.html

[Farcet01] Farcet, Nicolas; Salicki, Serge *Expression and Usage of the Variability in the Software Product Lines*; in: Software Product-Family Engineering 4th International Workshop; revised papers / PFE 2001, Bilbo Spain, October 3-5 2001; Frank van der Linden (ed.) - Berlin; Heidelberg; New York; u.a.: Springer 2002.

[Gabler00] Gabler Wirtschaftslexikon *15. Aufl.* Wiesbaden : Gabler, 2000

[Geese00] Geese, Elmar; Heiliger, Markus: *XML mit VBS und ASP*; Bonn: Galileo Press 2000.

[Heinrich97] Heinrich, Lutz J.: *Management von Informatik-Projekten*; München; Wien; Oldenbourg: Oldenbourg Verlag 1997.

[Hürten99] Hürten, Robert: *Function-Point Analysis - Theorie und Praxis*; Rebbingen-Malmsheim: expert-Verl. 1999.

[Jones98] Jones, T. Capers: *Estimating Software Costs*; New York: McGraw-Hill 1998

[Kallfass90] Kallfaß, Hermann H.: *Großunternehmen und Effizienz*; Wirtschaftspolitische Studien 79; Göttingen: Vandenhoeck u. Ruprecht 1990

[Kay01] Kay, Michael: *XSLT- Programmers Reference 2nd Edition*; Wrox Press Ltd. 2001

[Knauber01] Knauber, Peter [et al.] *Quantifying Product Line Benefits*; in: Software Product-Family Engineering 4th International Workshop; revised papers / PFE 2001, Bilbo Spain, October 3-5 2001; Frank van der Linden (ed.) - Berlin; Heidelberg; New York; u.a.: Springer 2002.

[Litke96] Litke, Hans Dieter: *DV-Projektmanagement: Zeit und Kosten richtig einschätzen*; München; Wien: Hanser 1996

[Northrop03] Northrop, Linda M.: *A Framework for Software Product Line Practice*;
verfügbar unter: http://www.sei.cmu.edu/plp/framework.html

[PULSE02] Fraunhofer Institute for Experimental Software Engineering IESE: *PULSE Produktlinien für Software-Systeme*; verfügbar unter: http://www.iese.fhg.de/PuLSE/pulse_d_2002_04.pdf.

[SEI02] Software Engineering Institute; Carnegie Mellon University: *Application Engineering Process Example* verfügbar unter: http://www.sei.cmu.edu/domain-engineering/appl_eng_example.html

[Sneed91] Sneed, Harry M.: *Softwarewartung und -wiederverwendung* Bd. 1 Softwarewartung; Köln: R.Müller (DV-Praxis-Online) 1991.

[Stahlknecht99] Stahlknecht, Peter; Hasenkamp, Ulrich: *Einführung in die Wirtschaftsinformatik 9.Aufl*; Berlin, Heidelberg, New York u.a: Springer 1999

[Sturm00] Sturm, Jake: *XML-Lösungen programmieren*; Unterschleißheim: Microsoft Press Deutschland 2000

[Streitferdt00] Streitferdt, Detlef; Böllert Kai; Riebisch Matthias: *Feature Modell und Architektur eine Systemfamilie*; in: 45. Internatio-

nales Wissenschaftliches Kolloquium der TU Ilmenau, 6.-10. Oktober 2000.

[Streitferdt03] Streitferdt, Detlef: *Family Oriented Requirements Engineering*; verfügbar unter: http://www.theoinf.tu-ilmenau.de/~streitdf/FORE/.

[Thaller00] Thaller, Georg Erwin: *Software-Metriken einsetzen, bewerten, messen; 2.Aufl.*; Berlin: Verlag Technik 2000

[visek] Virtuelles Software Engineering Kompetenzzentrum verfügbar unter: www.visek.de/servlet/is/2182/

[Walsh99] Walsh,Leonard; Muellner, Norman: *DocBook: The Definitive Guide*; O´ Reilly and Associates, Inc. verfügbar unter: http://www.docbook.org/tdg/en/.

[Weiss99] Weiss, David; Lai, Chi Tau Robert *Software Product-Line Engineering: A Family-Based Software Development Process* Addison-Wesley Professional: 1999

[Whitey96] Whitey, James: *Investment Analysis of Software Assets for Product Lines*; Software Engineering Institute, Carnegie Mellon University, Pittsburgh 1996; verfügbar unter: http://www.sei.cmu.edu/publications/documents/96.reports/96.tr.010.html.

[W3C] World Wide Web Consortium: *Extensible Markup Language (XML) 1.0 (Zweite Auflage)* Empfehlung des W3C, 20. Januar 2002, deutsche Übersetzung von Stefan Mintert verfügbar unter: http://edition-w3c.de/TR/2000/REC-xml-20001006/

[W3CSchema] World Wide Web Consortium: *XML Schema Part 1: Structures*; verfügbar unter: http://www.w3.org/TR/xmlschema-1/

A COCOMOII

A.1 Überblick COCOMOII: Kostentreiber

COCOMOII.2000 Post Architecture Cost Driver Description

Cost-Drivers	Very Low	Low	Nominal	High	Very High	Extra High
RELY	Slight Inconvenience	low, easily recoverable losses	moderate, easily recoverable losses	high, financial loss	risk to human life	
DATA		(testing DB bytes / Pgm SLOC) < 10	10 ≤ D/P < 100	100 ≤ D/P < 1000	D/P > 1000	
CPLX						
RUSE		none	across project	across program	across product line	across multiple product lines
DOCU	Many lifecycle needs uncovered	some lifecycle needs uncovered	correct amount of lifecycle needs	excessive for lifecycle needs	very excessive for lifecycle needs	
TIME			≤ 50 % use of available execution time	70% use	85% use	95% use
STOR			≤ 50 % use of available storage	70% use	85% use	95% use
PVOL		Major change every 12 mo.; minor change every 1mo.	major: 6mo.; minor: 2 wk.	major: 2mo.; minor: 1 wk.	major: 2 wk.; minor: 2 days	
ACAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
PCAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
PCON	48% / year	24% / year	12% / year	6% / year	3% / year	
APEX	≤ 2 months	6 months	1 year	3 years	6 years	
PLEX	≤ 2 months	6 months	1 year	3 years	6 years	
LTEX	≤ 2 months	6 months	1 year	3 years	6 years	
TOOL	edit, code, debug	simple, frontend, backend CASE, little integration	Basic lifecycle tools, moderately integrated	strong, mature lifecycle tools, moderately integrated	strong, mature, proactive lifecycle tools, wells integrated with processes, methods, reuse	
SITE Collocation	International	multi-city and multi-company	multi-city or multi-company	same city or metro area	same building or complex	fully collocated
SITE Communi-cation	Some phone, mail	individual phone, FAX	narrow-band email	wide-band electronic communication	wide-band elect. comm, occasional video conf.	interactive multimedia
SCED	75% of nominal	85% of nominal	100% of nominal	130% of nominal	160% of nominal	

A.2 Auflistung COCOMOII: Kostentreiber

Product Factors

RELY Cost Driver – Required Software Reliability

RELY Descriptors:	slight inconvenience	low, easily recoverable losses	moderate, easily recoverable losses	high, financial loss	risk to human life	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0.82	0.92	1.00	1.10	1.26	n/a

DATA Cost Driver – Database Size

DATA Descriptors:		(testing DB bytes / Pgm SLOC) < 10	10 ≤ D/P < 100	100 ≤ D/P < 1000	D/P > 1000	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0.90	1.00	1.14	1.28	n/a

CPLX Cost Driver – Product Complexity

Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0.73	0.87	1.00	1.17	1.34	1.74

RUSE Cost Driver – Developed for Reusability

RUSE Descriptors:		none	across project	across program	across product line	across multiple product lines
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0.95	1.00	1.07	1.15	1.24

DOCU Cost Driver – Documentation match to Life-Cycle Needs

DOCU Descriptors:	Many lifecycle needs uncovered	some lifecycle needs uncovered	correct amount of lifecycle needs	excessive for lifecycle needs	very excessive for lifecycle needs	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0.81	0.91	1.00	1.11	1.23	n/a

Platform Factors

TIME Cost Driver – Execution Time Constraint

TIME Descriptors:			≤ 50 % use of available execution time	70% use	85% use	95% use
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	n/a	1.00	1.05	1.17	1.46

STOR Cost Driver – Main Storage Constraint

STOR Descriptors:			≤ 50 % use of available storage	70% use	85% use	95% use
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	n/a	1.00	1.05	1.17	1.46

PVOL Cost Driver – Platform Volatility

PVOL Descriptors:		Major change every 12 mo.; minor change every 1mo.	major: 6mo.; minor: 2 wk.	major: 2mo.; minor: 1 wk.	major: 2 wk.; minor: 2 days	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0.87	1.00	1.15	1.30	n/a

Personal Factors

ACAP Cost Driver – Analyst Capability

ACAP Descriptors:	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.42	1.19	1.00	0.85	0.71	n/a

PCAP Cost Driver – Programmers Capability

PCAP Descriptors:	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.34	1.15	1.00	0.88	0.76	n/a

PCON Cost Driver – Personal Continuity

PCON Descriptors:	48% / year	24% / year	12% / year	6% / year	3% / year	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.29	1.12	1.00	0.90	0.81	n/a

APEX Cost Driver – Applications Experience

APEX Descriptors:	≤ 2 months	6 months	1 year	3 years	6 years	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.22	1.10	1.00	0.88	0.81	n/a

PLEX Cost Driver – Platform Experience

PLEX Descriptors:	≤ 2 months	6 months	1 year	3 years	6 years	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.19	1.09	1.00	0.91	0.85	n/a

LTEX Cost Driver – Language and Tool Experience

LTEX Descriptors:	≤ 2 months	6 months	1 year	3 years	6 years	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.20	1.09	1.00	0.91	0.84	n/a

Project Factors

TOOL Cost Driver – Use of Software Tools

TOOL Descriptors:	edit, code, debug	simple, frontend, backend CASE, little integration	Basic lifecycle tools, moderately integrated	strong, mature lifecycle tools, moderately integrated	strong, mature, proactive lifecycle tools, wells integrated with processes, methods, reuse	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.17	1.09	1.00	0.90	0.78	n/a

SITE Cost Driver – Multiside Development

SITE Collocation Descriptors	International	multi-city and multi-company	multi-city or multi-company	same city or metro area	same building or complex	Fully collocated
SITE Communications Descriptors	Some phone, mail	individual phone, FAX	narrow-band email	wide-band electronic communication	wide-band elect. comm, occasional video conf.	interactive multimedia
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.22	1.09	1.00	0.93	0.86	0.80

SCED Cost Driver – Required Development Schedule

SCED Descriptors:	75% of nominal	85% of nominal	100% of nominal	130% of nominal	160% of nominal	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.43	1.14	1.00	1.00	1.00	n/a

A.3 Überblick COCOMOII: Skalenfaktoren

COCOMOII.2000 Post Architecture Scale factor Description

Cost-Drivers	Very Low	Low	Nominal	High	Very High	Extra High
PREC	Thoroughly unprecedented	largely unprecedented	Somewhat unprecedented	generally familiar	largely familiar	thoroughly familiar
FLEX	Rigorous	occasional relaxation	some relaxation	general conformity	some conformity	general goals
RESL	little (20%)	some (40%)	often (60%)	generally (75%)	mostly (90%)	full (100%)
TEAM	very difficult interactions	some difficult interactions	Basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions
PMAT	SW-CMM Level 1 Lower	SW-CMM Level 1 Upper	SW-CMM Level 2	SW-CMM Level 3	SW-CMM Level 4	SW-CMM Level 5
	or the estimated Process Maturity Level					

A.4 Auflistung COCOMOII: Skalenfaktoren

PREC – Precedentedness Rating Levels

Feature	Very Low	Normal/High	Extra High
Organizational understanding of product objectives	General	Considerable	Thorough
Experience in working with related software systems	Moderate	Considerable	Extensive
Concurrent development of associated new hardware and operational procedures	Extensive	Moderate	Some
Need for innovative data processing architectures, algorithms	Considerable	Some	Minimal

FLEX – Development Flexibility Rating Levels

Feature	Very Low	Normal/High	Extra High
Need for software conformance with pre-established requirements	Full	Considerable	Basic
Need for software conformance with external interface specifications	Full	Considerable	Basic
Combination of inflexibilites above with premium on early completion	High	Medium	Low

RESL – Architecture / Risk Resolution Rating Levels

Characteristic	Very Low	Low	Nominal	High	Very High	Extra High
Risk Management Plan identifies all critical risk items, establishes mile-stones for resolving them by Product Design Review (PDR) or Life Cycle Architecture (LCA)	None	Little	Some	Generally	Mostly	Fully
Schedule, budget, and internal mile-stones through PDR or LCA compatible with Risk Management Plan	None	Little	Some	Generally	Mostly	Fully
Percent of develop-ment schedule de-voted to establishing architecture, given general product ob-jectives	5	10	17	25	33	40
Percent of required top software architects available to project	20	40	60	80	100	120
Tool support avail-able for resolving risk items, developing and verifying architectural specs	None	Little	Some	Good	Strong	Full
Level of uncertainty in key architecture drivers: mission, user interface, COTS, hardware, technology, performance.	Extreme	Significant	Considerable	Some	Little	Very Little
Number and criti-cality of risk items	>10 Critical	5-10 Critical	2-4 Critical	1 Critical	>5 Non-Critical	<5 Non-Critical

TEAM – Team Cohesion rating Components

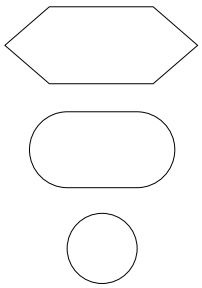
Characteristic	Very Low	Low	Nominal	High	Very High	Extra High
Consistency of stake-holder objectives and cultures	Little	Some	Basic	Considerable	Strong	Full
Ability, willingness of stakeholders to acommo-date other stakeholders' objectives	Little	Some	Basic	Considerable	Strong	Full
Experience of stakeholders in operating as a team	None	Little	Little	Basic	Considerable	Extensive
Stakeholder teambuilding to achieve shared vision and commitments	None	Little	Little	Basic	Considerable	Extensive

PMAT – Ratings for Estimated Process Maturity Level (EPML)

PMAT Rating	Maturity Level	EPML
Very Low	CMM Level 1 (lower half)	0
Low	CMM Level 1 (upper half)	1
Nominal	CMM Level 2	2
High	CMM Level 3	3
Very High	CMM Level 4	4
Extra High	CMM Level 5	5

B Notation: Ereignisgesteuerte Prozessketten

Ereignisgesteuerte Prozessketten stellen ungerichtete Graphen zur Beschreibung von Geschäftsprozessen oder Arbeitsabläufen dar, bei denen die Knoten



beschreiben. Funktionen und Ereignisse werden abwechselnd aufeinander dargestellt, wobei Funktionen:

- von einem oder mehreren Ereignissen ausgelöst werden und
- ein oder mehrere Ereignisse auslösen.

Mit Hilfe der Verknüpfungsoperatoren werden alternativ logische Verknüpfungen zwischen auslösenden und erzeugenden Funktionen und/oder Zuständen zum Ausdruck gebracht:

- UND (Symbol $\wedge$)
- Inklusives ODER (Symbol $\vee$)
- Exklusives ODER (Symbol XOR)

Erklärung über Hilfsmittel

Die vorliegende Arbeit habe ich selbständig und ohne Benutzung anderer als der angegebenen Quellen angefertigt. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Quellen entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Ilmenau, 03.März 2003

René Kluge