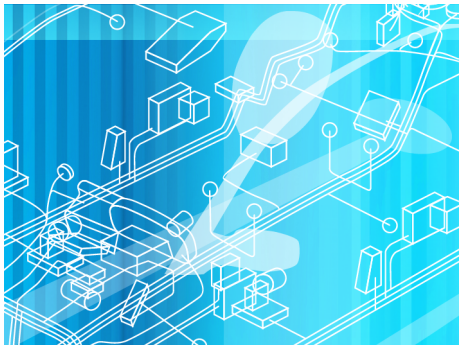
ISST

Fraunhofer Institut Software- und Systemtechnik

Entscheidungsgrundlagen für die Entwicklung von Softwareproduktlinien

Martin Große-Rhode,
Ekkart Kleinod, Stefan Mann



ISST-Bericht 83/07
Oktober 2007

Herausgeber: Fraunhofer-Gesellschaft e. V.
Institut für Software- und Systemtechnik
Leitung: Prof. Dr. Jakob Rehof
Institutsteil Berlin: Mollstraße 1
10178 Berlin
Institutsteil Dortmund: Joseph-von-Fraunhofer-Straße 20
44227 Dortmund

ISSN 0943-1624

Das Projekt VEIA

Das Projekt »Verteilte Entwicklung und Integration von Automotive-Produktlinien« wird vom Bundesministerium für Bildung und Forschung im Rahmen der Forschungsoffensive »Software Engineering 2006« unter dem Förderkennzeichen »01ISF15A« gefördert.

Am Projekt sind vier Partner beteiligt:

- BMW Group
- Fraunhofer ISST
- PROSTEP IMP GmbH
- Technische Universität München

Webseite: http://veia.isst.fraunhofer.de/

Autoren

Dieses Dokument wurde geschrieben von:

Martin Große-Rhode, Ekkart Kleinod, Stefan Mann (Fraunhofer ISST).

Kurzzusammenfassung

Die Entwicklung der Software für E/E-Systeme in Fahrzeugproduktlinien ist in einen jeweils unternehmensspezifischen E/E-Produktentstehungsprozess eingebettet. Aus einem Entwurf des E/E-Systems werden zunächst die funktionalen und nicht-funktionalen Anforderungen an die Software des Gesamtsystems abgeleitet. Für den anschließenden Entwurf der Software für eine Fahrzeugproduktlinie oder ein Fahrzeugprojekt bieten sich zwei grundsätzliche Lösungsmöglichkeiten an: Die Software kann als Softwareproduktlinie konzipiert werden, aus der produktspezifische Varianten generiert werden, oder als eine fixe, produktgenerische Softwarelösung, die alle vorhandenen Varianten in sich enthält und in allen Fahrzeugen installiert wird.

Um zwischen diesen und gegebenenfalls weiteren Lösungsalternativen objektiv entscheiden zu können, werden Bewertungen benötigt, die sich auf die vorliegenden Anforderungen an die Software oder auf Modelle der Entwurfsalternativen abstützen. Dabei sollen die Bewertungen einerseits quantifizierbar, dass heißt durch Metriken belegt sein; andererseits sollen sie bereits früh im Prozess vorgenommen werden können.

Auf der Grundlage des VEIA-Referenzprozesses, in dem die Entwicklungsartefakte (Architekturmodelle) definiert und in eine zeitliche Abfolge gebracht sind, werden in diesem Bericht die Aktivitäten beschrieben, die zu der oben diskutierten Entscheidungsgrundlage führen. Nach einer genaueren Analyse der Entscheidungs- bzw. Bewertungsaufgabe werden geeignete Metriken vorgestellt und mit der Struktur der VEIA-Modelle integriert. Anschließend wird die am weitesten ausgearbeitete Metrik zur Umfangs- und Aufwandsschätzung von Softwareproduktlinien auf die Fallstudie *Condition-Based Service* beispielhaft angewendet.

Inhalt

1	**Einleitung**	5
1.1	Verwendung von Metriken im VEIA-Referenzprozess	5
1.2	Untersuchungsgegenstände	6
1.3	Bewertungsszenario	7
1.4	Anforderungen an die Softwarearchitektur	8
1.5	Metriken	9
1.6	Struktur dieses Dokuments	11
2	**CBS: Ausgangssituation**	13
2.1	Artefakte	13
2.2	Bewertungskriterien	19
3	**Analyse der Bewertungsaufgabe**	23
3.1	Generische und spezifische Softwarekomponenten	23
3.2	Bindungszeit von Varianz	25
4	**Metrikdefinitionen**	31
4.1	Kostenbasiertes Softwareproduktlinienmanagement	32
4.2	Allgemeine Metriken zu Softwareproduktlinien	33
4.3	PFP-Metriken	36
4.4	Metriken auf UML-Modelle am Beispiel »SDMetrics«	62
4.5	Bewertung von Varianz in Architekturen	65
4.6	Servicenutzungsgrade und Kohärenz	70
4.7	Resümee	77
5	**Anwendung der Metriken auf CBS**	79
5.1	Produkte	79
5.2	Anwendung auf die logische Architektur von CBS	80
5.3	Anwendung auf die CBS-Softwarearchitekturen	84
6	**Zusammenfassung und Ausblick**	95
Literatur		97

1 Einleitung

Das Projekt VEIA hat sich das Ziel gesetzt, auf der Grundlage der Konzepte der Produktlinientechnik eine Methode für die verteilte Entwicklung und Integration von Automotive-E/E-Systemen zu erarbeiten, die sich an den konkreten Anforderungen industrieller Entwicklungsprozesse orientiert und praktisch anwendbar ist.

Ein wichtiges Ziel des Projekts ist, eine Entscheidung darüber treffen zu können, ob ein Projekt mit generischen Softwarekomponenten (150 %-Lösung) oder mit produktspezifischen Softwarekomponenten (Produktlinienlösung) umgesetzt werden soll. Diese Entscheidung soll mit Hilfe der im Referenzprozess definierten Artefakte, also der Modelle des Systems getroffen werden. In diesem Sinn soll eine ausführbare Wissensbasis geschaffen werden, die anhand von Metriken den Nutzen des Produktlinienansatzes gegenüber der bisherigen Realisierung zeigt.

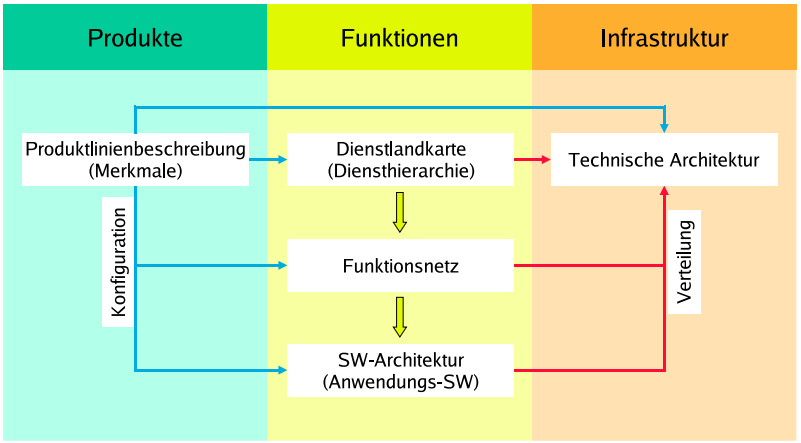
1.1 Verwendung von Metriken im VEIA-Referenzprozess

In der ersten Projektphase wurde der Grobentwurf eines Referenzprozesses definiert, in dem unter anderem die Artefakte zur Erfassung und Modellierung von Architektursichten festgelegt sind, siehe [GEKM07]. Die Artefakte sind (siehe Abbildung 1):

- Produktlinienbeschreibung durch die Produktmerkmale,
- Dienstlandkarte,
- Funktionsnetz,
- Softwarearchitektur,
- Technische Architektur,
- Konfigurationsmodelle,
- Verteilungsmodelle.

Mit Hilfe dieser Artefakte wird ein zu entwickelndes System in den unterschiedlichen Sichten auf unterschiedlichen Abstraktionsniveaus beschrieben. Der Nutzen der Artefakte liegt zum einen in der präzisen Beschreibung des Systems. Darüber hinaus sind mit Hilfe der Architekturmodelle vielfältige Bewertungen des Systems möglich.

Abbildung 1 Artefakte des VEIA-Referenzprozesses.



Bewertungen als Analyseaktivitäten im Referenzprozess waren Untersuchungsgegenstand der zweiten Projektphase, siehe [GKM07a]. In dieser Phase wurde zunächst geklärt, was Bewertungen im Architektur- und Produktlinienkontext sind. Daraus wurde eine konkrete Bewertungsmethodik entwickelt, wie Bewertung und Vergleich von Architekturentwürfen durchgeführt werden sollen. Dabei wurde auch darauf eingegangen, dass Informationen unterschiedlicher Artefakte für Bewertungen notwendig sein können.

Die Ermittlung der Kennzahlen, die für die Bewertung der Entwürfe verwendet werden, erfolgt durch Metriken. Die Metriken wurden in ihrer Bedeutung thematisiert, aber nicht expliziert. An dieser Stelle knüpft die dritte Phase und damit dieses Dokument an.

1.2 Untersuchungsgegenstände

Die Zielfrage der Bewertungen, die nun untersucht wird, ist die Entscheidung zwischen unterschiedlichen Softwarelösungen: zwischen der 150 %-Lösung (mittels generischen Softwarekomponenten) und der Produktlinienlösung (mit spezifischen Softwarekomponenten). Das heißt, es müssen Metriken gefunden und angewendet werden, die Eigenschaften der Softwarearchitektur messen.

Die Softwarearchitektur ist aber nur ein Artefakt des VEIA-Referenzprozesses. Auch aus den anderen Artefakten können Aussagen über die Software gewonnen werden. So enthält z. B. das Funktionsnetz, das als Abstraktion der Softwarear-

chitektur gesehen werden kann, bereits Informationen über Komplexität oder Varianten, die auch in der Softwarearchitektur entstehen werden. Da das Funktionsnetz im Prozess deutlich früher als die Softwarearchitektur erstellt wird bzw. werden soll, sind auf dessen Basis sehr frühe Aussagen über die Software möglich. Des Weiteren hängen die Eigenschaften des modellierten Systems nicht allein von der Software ab, sondern auch von der technischen Architektur und den Konfigurationsmöglichkeiten der Produktmerkmale. Diese Wechselbeziehungen zwischen den Artefakten sind ein Untersuchungsgegenstand dieses Dokuments.

Der zweite Untersuchungsgegenstand sind die Informationen, die den Artefakten übergeben werden. Gerade für die Bewertung von abhängigen Artefakten wie Funktionsnetz und Softwarearchitektur ist es wichtig zu wissen, welche Informationen im Funktionsnetz nötig sind, um qualifizierte Aussagen über die Software treffen zu können.

Der dritte Untersuchungsgegenstand sind die Metriken selbst. Es sollen einige Metriken untersucht und angewendet werden, um den Nutzen der Artefakte nicht nur zu postulieren, sondern um ihn quantifizieren zu können.

Diese drei Untersuchungsgegenstände sind natürlich nicht orthogonal zueinander, sondern ergänzen und beeinflussen einander.

1.3 Bewertungsszenario

Für die Diskussion der Bewertung von Softwarearchitekturentwürfen sei folgende Ausgangssituation gegeben:

1. Der Umfang der Produktlinie, d. h. die Menge der zu entwickelnden E/E-Systeme, ist durch die Produktlinienbeschreibung vorgegeben. Alle für die Softwareentwicklung relevanten Produktmerkmale inklusive der Merkmalsvarianten (alternative und optionale Merkmale) sind erfasst. Inwiefern dies nicht kundensichtbare Merkmale umfasst (z. B. AUTOSAR vs. klassische Standard-Software) und wie letzteres ggf. repräsentiert wird, ist noch zu diskutieren.

2. Die funktionalen Anforderungen an die zu entwickelnde Software inklusive der Funktionsvarianz sind durch ein Funktionsnetz erfasst. Ein Funktionsnetz beschreibt die Funktion des Systems auf einer sehr abstrakten Ebene. Die Beschreibungsmittel für Funktionsnetze und Softwarearchitekturen sind einander sehr ähnlich, jedoch können Funktionsnetze echte Varianz enthalten, Softwarearchitekturen z. B. nach AUTOSAR nicht. Trotzdem kann ein Funktionsnetz als sehr abstrakte Softwarearchitektur verstanden werden.

Auch hier ist noch zu diskutieren, inwiefern Verhaltensinformationen für die Bewertung vorhanden sein müssen. Dies ist aber keine methodische Frage, wie die bzgl. der Merkmale, sondern ergibt sich aus den Anforderungen der Metriken.

3 Die verfügbaren Ressourcen inklusive der Ausstattungsvarianten sind als technische Architektur modelliert. Das heißt, die zu betrachtenden Steuergeräte bzw. Prozessoren und Speicher, deren Kommunikationsverbindungen sowie die optionalen und alternativen Bestandteile sind repräsentiert. Der Detaillierungsgrad des technischen Architekturmodells wird sich wiederum aus den Anforderungen der Metriken ergeben.

4 Gegebenenfalls ist eine Vorgabe für die Verteilung der Funktionen und Informationsflüsse (logische Signalflüsse, Operationsaufrufe) auf die technische Architektur durch ein Verteilungsmodell dargestellt. Diese Vorgabe muss nicht vollständig sein, d. h., es muss nicht für alle Funktionen festgelegt sein, auf welchem Steuergerät sie ausgeführt werden sollen.

5 Die Zuordnung der Optionen und Varianten im Funktionsnetz und in der technischen Architektur zu den Produktmerkmalen ist durch entsprechende Konfigurationsmodelle festgelegt. Das heißt, jeder Varianzpunkt ist mit Produktmerkmalen annotiert, so dass festgelegt ist, welche der Alternativen bzw. ob eine Option für ein bestimmtes Produkt gewählt wird.

1.4 Anforderungen an die Softwarearchitektur

Durch die Funktionsmodelle wird eine Aufgabe gestellt: Eine Menge von Funktionen ist zu implementieren, wobei die technischen Ressourcen vorgegeben sind. Die Lösung ist eine Softwarearchitektur, d. h. ein entsprechendes Modell im Sinne von VEIA bzw. eine (hierarchische) *AUTOSAR Software Component Description*, zusammen mit der Einbettung in die vorgegebenen Modelle. Das heißt:

1 Die Verfeinerungsbeziehung zum Funktionsnetz ist hergestellt und verifiziert: Die Software erfüllt die funktionalen Anforderungen.

2 Die Konfiguration der Software (Auflösung der Varianz) ist durch ein Konfigurationsmodell dargestellt, das mit dem Funktionskonfigurationsmodell im Sinne der Verfeinerung kompatibel ist.

3 Das Modell der Verteilung der Softwarekomponenten und deren Kommunikation (Datenaustausch) auf die Ressourcen ist erstellt und ist kompatibel mit der vorgegebenen Funktionsverteilung.

Die Software muss dabei alle vorgesehenen Varianten realisieren. Sofern Varianten in der Struktur vorgesehen sind, werden diese wie in MOSES [Kle06] dargestellt. Die Darstellung von Varianz im Verhalten ist noch zu diskutieren.

1.5 Metriken

Um verschiedene Lösungen miteinander vergleichen zu können, sollen objektive Entscheidungsgrundlagen geschaffen werden. Dazu wird eine Architekturbewertung im Sinn von VEIA (Phase 2) durchgeführt, vgl. [GKM07a]. Das heißt:

1 Es werden die Kriterien bzw. Messgrößen bestimmt, die zur Bewertung verwendet werden sollen.

2 Zur Ableitung, Interpretation und Gewichtung der Messergebnisse werden ggf. zuvor Zielfragen oder Bewertungsszenarios formuliert.

3 Zu den Messgrößen passende Metriken werden zusammengestellt.

4 Die Metriken werden angewendet und die Messergebnisse gemäß der zuvor bestimmten Interpretation zusammengeführt, um damit eine Entscheidung fällen und begründen zu können.

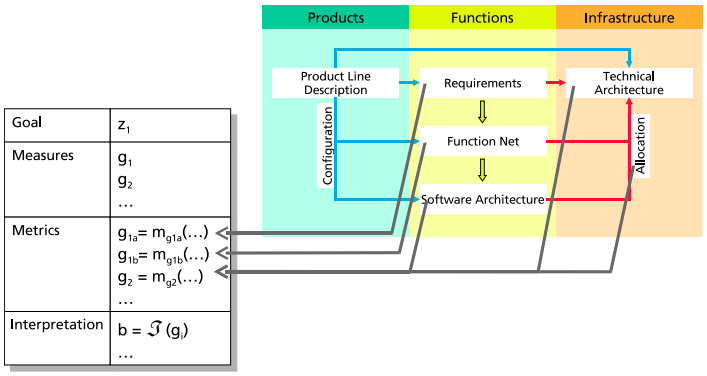
Aus der Definition der Metriken, genauer aus den Eingangsdaten der Metriken, wird abgeleitet, welche Informationen zum Bewertungszeitpunkt vorhanden sein müssen. Diese Informationen sind zum einen in den Modellen erfasste bzw. zu erfassende (Teil-)Produkteigenschaften, zum Beispiel: Anzahl der Softwarekomponenten oder Datenelemente, Anzahl der Buszugriffe, etc. Zum anderen können auch Prozess- oder Organisationseigenschaften Eingangsdaten für eine Metrik sein, die nicht in den betrachteten Modellen erfasst werden, zum Beispiel: Verfügbarkeit von Werkzeugen, Möglichkeit offener Kommunikation vs. Schutz geistigen Eigentums, Erfahrung der Mitarbeiter etc. Diese Eigenschaften fließen beispielsweise in Entwicklungs- oder Testaufwandsabschätzungen ein.

Bei den Produkteigenschaften ist zwischen denjenigen, die bereits in den VEIA- bzw. AUTOSAR-Modellen erfasst werden, und den darüber hinausgehenden Eigenschaften zu unterscheiden. Letztere werden dann als spezifische Bewertungsattribute in das VEIA-Datenmodell integriert, um eine automatische Bewertung mit dem zu erstellenden Werkzeug zu ermöglichen.

Die Messungen (Anwendungen von Metriken) müssen sich nicht auf alle Modelle beziehen (Abbildung 2). Für die Abschätzung des Entwicklungsaufwands, zum Beispiel, reicht (bzgl. der Produkteigenschaften) das Softwaremodell aus. Für die

Bewertung des Echtzeitverhaltens hingegen müssen relativ detaillierte Informationen über die technische Architektur sowie das Verhalten und die Verteilung der Softwarekomponenten vorhanden sein.

Abbildung 2 Metriken und Artefakte.



Die Bewertungen unterscheiden sich auch in der Art (Qualität) der Ergebnisse, die von der Genauigkeit bzw. Art der Eingangsdaten abhängen. Dabei kann es sich u. a. handeln um:

- gemessene Werte aus vergleichbaren Vorgängerprojekten, die als Prognosewerte verwendet werden,
- Werte, die in Unternehmensstandards vorgegeben sind, wie bspw. die Anzahl der Funktionen pro Datei oder die Anzahl der Eingangs- und Ausgangssignale von Funktionen,
- aus der Fachliteratur bezogene Standard- oder Durchschnittswerte,
- präzise Messwerte, die zur Verfügung stehen, weil das betreffende Artefakt bereits zur Verfügung steht (z. B. wenn die Softwarekomponente bereits vollständig in C implementiert ist).

Relativ zu den Bewertungen bzw. den zugehörigen Metriken, die im Prozess angewendet werden, lassen sich damit Reifegrade als Anforderungen an die Modelle bzw. Definitionen von Entscheidungspunkten im Prozess ableiten.

Reifegrad eines Modells *M*

Ein Reifegrad ist immer in Bezug auf eine Bewertung B inklusive einer Metrik M definiert, und zwar folgendermaßen: Um die Bewertung B (mit der Qualität Q) zu ermöglichen, muss das Modell M vollständig bezüglich der Modellelemente E sein

und die Werte W_i der Attribute A_i (in der Qualität Q_i) enthalten. Die Parameter Q, Q_i und A_i werden in der Definition der Bewertung definiert, die Parameter E und A_i in derjenigen der Metrik.

Beispiel: Um eine erste Aufwandsabschätzung zu ermöglichen, muss die Softwarearchitektur so weit modelliert sein, dass alle Ports und Signale (im Sinn eines *freeze*) vorhanden sind und alle Komponenten als invariant, horizontal variant oder vertikal variant gekennzeichnet sind (das Attribut *Varianztyp* ist gesetzt).

Entscheidungspunkt

Ein Entscheidungspunkt ist durch die Bewertungen gekennzeichnet, die an diesem Entscheidungspunkt durchgeführt werden sollen. Dadurch sind die geforderten Modelle und deren Reifegrade bestimmt.

1.6 Struktur dieses Dokuments

Im Dokument wird nachfolgend die eben beschriebene Ausgangssituation auf das Beispiel *CBS* angewendet. In Kapitel 3 wird die gestellte Aufgabe anhand des Beispiels detailliert analysiert.

Damit sind die Vorbereitungen für Kapitel 4 abgeschlossen, das Metriken vorstellt, mit denen die gestellte Aufgabe gelöst werden kann. Dabei werden die Metriken motiviert und erläutert sowie deren theoretischer Hintergrund bezüglich der VEIA-Artefakte beleuchtet.

In Kapitel 5 werden die gefundenen Metriken auf das Beispiel *CBS* angewendet. Damit ist ein erster Vergleich der Metriken und ihrer Aussagekraft möglich.

Den Abschluss bildet die Zusammenfassung mit dem Ausblick auf weitere, offene Themen.

2 CBS: Ausgangssituation

In der Einleitung wurde bereits die Ausgangssituation bezüglich der vorhandenen Modelle und Informationen geschildert. In diesem Kapitel wird die beschriebene Situation anhand des Beispiels CBS illustriert. Parallel zu diesem Dokument entsteht das Dokument »Fallstudie *Condition-Based Service*: Modelle für die Bewertung von logischen Architekturen und Softwarearchitekturen« [GKM07b], in dem alle Modelle sukzessive ausformuliert werden. Daher sind die vorgestellten Artefakte Ausschnitte und nicht das komplett ausformulierte Beispiel. Im Anschluss an die Artefakte werden die Bewertungskriterien rekapituliert.

2.1 Artefakte

2.1.1 Merkmalsmodell / Produktlinienbeschreibung

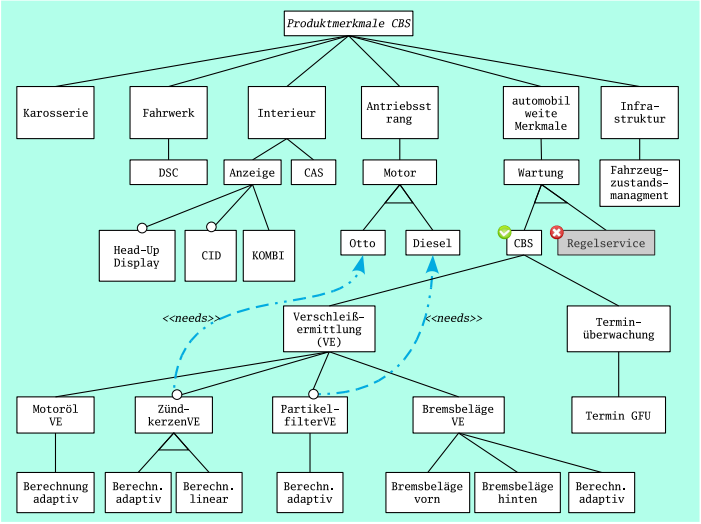
Im Merkmalsmodell werden die Fahrzeugmerkmale erfasst, die relevant für CBS sind. Das sind im Wesentlichen die Verschleißteile bzw. die Fahrzeugkomponenten, die die entsprechenden Umfänge enthalten. Außerdem wird festgelegt, ob und mit welcher Methode (d. h., ob mit Hilfe von Sensorik oder nicht) der Verschleiß eines Verschleißteils erfasst werden soll. Hier werden auch die Produkte festgelegt, die mit der Produktlinie entwickelt werden sollen (Konfigurationsdefinition).

Abbildung 3 zeigt einen Ausschnitt aus dem Merkmalsmodell für CBS. Die Anzeige erfolgt über das Kombigerät und optional über ein *Head-Up-Display* oder ein *Central Information Display (CID)*. Der Motor kann entweder ein Otto- oder ein Dieselmotor sein. Die Wartungsintervalle werden entweder vom CBS oder vom Regelservice berechnet.

Innerhalb des CBS sind Verschleißermittlung und Terminüberwachung Pflicht, bei der Verschleißermittlung gibt es optionale Elemente: Zündkerzenüberwachung oder Überwachung des Partikelfilters. Die Zündkerzen können nur bei Ottofahrzeugen überwacht werden, analog die Partikelfilter nur bei Dieselfahrzeugen. Der Zündkerzenverschleiß kann zusätzlich entweder linear (nur abhängig von der Zeit bzw. gefahrenen Strecke) oder adaptiv (mit Anpassung der Verschleißkurve durch Sensordaten) berechnet werden.

Damit ist die Produktlinie für die Funktion CBS definiert, es können maximal so

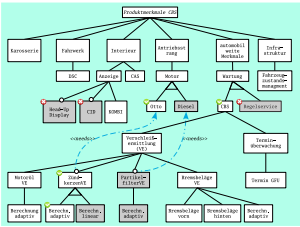
Abbildung 3 Merkmalsmodell für CBS (Ausschnitt).



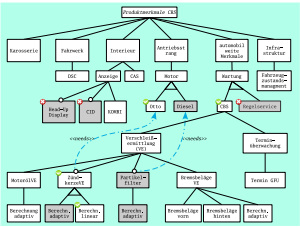
viele Produkte von CBS gebaut werden, wie kombinatorische Möglichkeiten in der Produktlinienbeschreibung vorgesehen sind. Um die Kombinatorik im Folgenden nicht ausufern zu lassen, werden die vorgesehenen Produkte per Definition auf drei beschränkt, die in Abbildung 4 zu sehen sind.

Abbildung 4 CBS-Produkte.

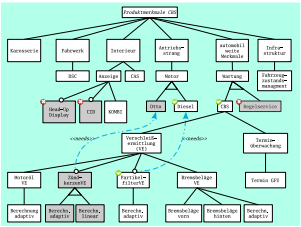
(a) *Produkt P1.*



(b) *Produkt P2.*



(c) *Produkt P3.*



2.1.2 Funktionsnetz

Das Funktionsnetz beschreibt die logische Funktionalität des Systems. Es besteht aus einzelnen Funktionen, die hierarchisiert und vernetzt werden können und

im Zusammenspiel das Verhalten des Systems realisieren. Das Funktionsnetz ist eine sehr abstrakte Sicht auf das Verhalten, das durch die Software letztendlich umgesetzt wird. In diesem Sinn kann das Funktionsnetz als abstrakte Softwarearchitektur angesehen werden.

Abbildung 5 zeigt den zum Merkmalsmodell passenden Ausschnitt des Funktionsnetzes. Auf der linken Seite sind die Schnittstellen zu den Datenquellen zu sehen: Sensoren, Zeitgeber etc.

Die Daten der Sensoren werden durch die CBS-Clients empfangen und aufbereitet. Dabei agieren die Clients als verschleißteilspezifische Überwachungsfunktionen. Die CBS-Clients können sensor-adaptiv oder linear ausgelegt sein.

Die Clients senden die aufbereiteten Daten an die Zentralfunktion `CBS Optimierten Servicetermin ermitteln`, die die Daten zusammenführt und den Fahrzeuggesamtzustand berechnet. Die Zentralfunktion übernimmt auch die Ansteuerung der Datensenken: Anzeigen und, in der Abbildung nicht dargestellt, Fahrzeugschlüssel.

Im Bild ist außerdem noch die Varianzinformation zu sehen, die im Modell abgelegt ist: optionale Funktionen sind gestrichelt, Varianzpunkte besitzen abgerundete Ecken. Die Varianz ist nicht aus dem Funktionsnetz abgeleitet, sondern entspringt dem Merkmalsmodell. Die Verbindung zwischen Merkmalsmodell und Funktionsnetz, das Konfigurationsmodell, ist in der Abbildung der Übersichtlichkeit halber nicht dargestellt.

2.1.3 Infrastruktur

Eine Anforderung an die Implementierung von CBS ist, dass keine zusätzliche Hardware verwendet werden darf. Damit ist die Infrastruktur von CBS, vgl. Abbildung 6, aus folgenden grundlegenden Komponenten aufgebaut: Sensoren, deren Werte benutzt werden; Client-Steuergeräte, die die Werte der Sensoren abfragen bzw. virtuelle Sensoren darstellen; das Kombisteuergerät, auf dem der CBS-Master sowie die linearen Clients laufen sowie Ausgabegeräte wie Displays oder der Fahrzeugschlüssel. Die einzelnen Steuergeräte sind durch Busse miteinander verbunden.

In der Abbildung ist zusätzlich zur Infrastruktur die Verteilung der Funktionen auf die Hardware angegeben. Während das Verteilungsmodell die Aufteilung von Funktionen auf verschiedene Steuergeräte vorsieht, wurde im Beispiel davon nicht Gebrauch gemacht, so dass die Verteilung einfach darzustellen ist. Außerdem

Abbildung 5 Funktionsnetz CBS (Ausschnitt).

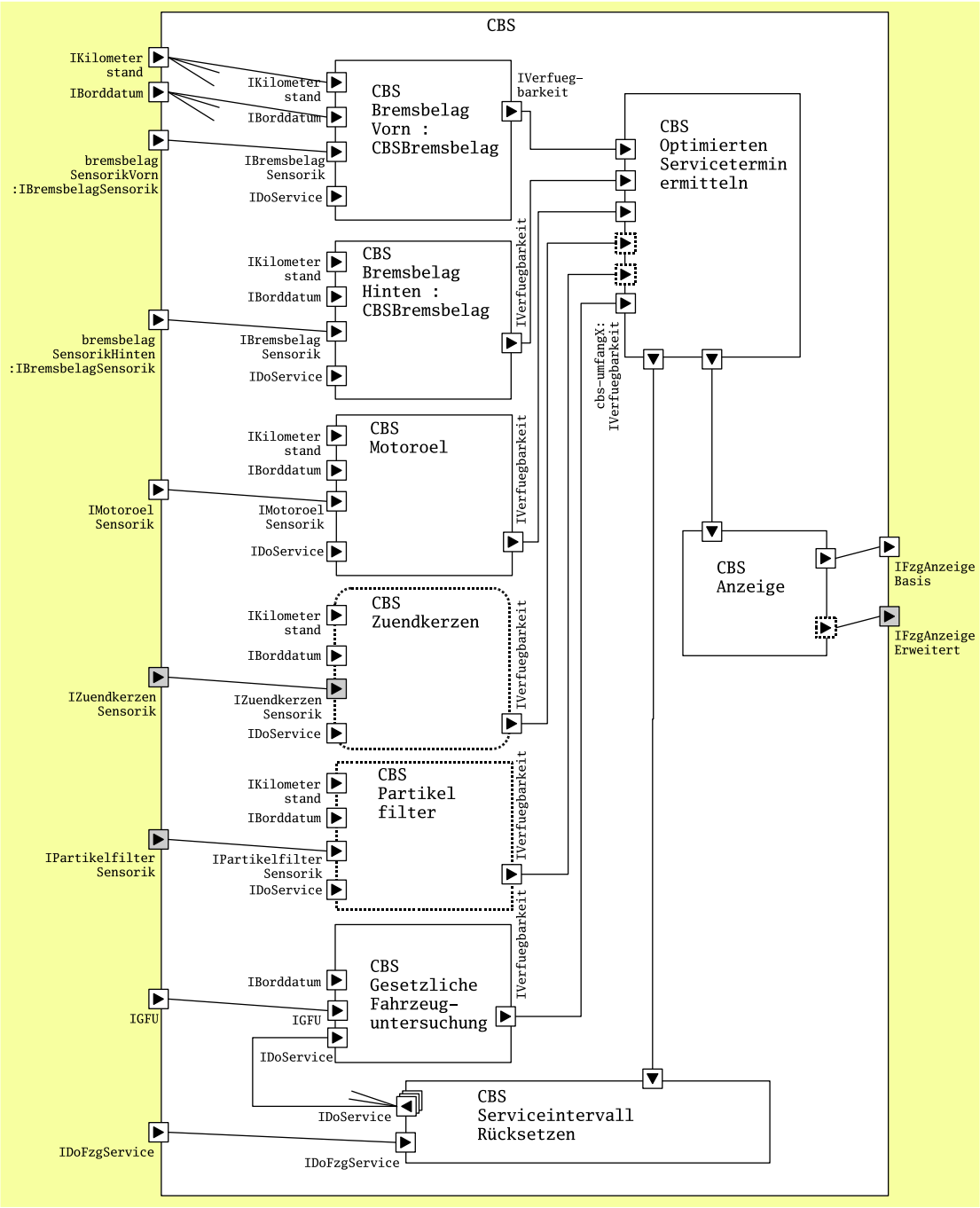
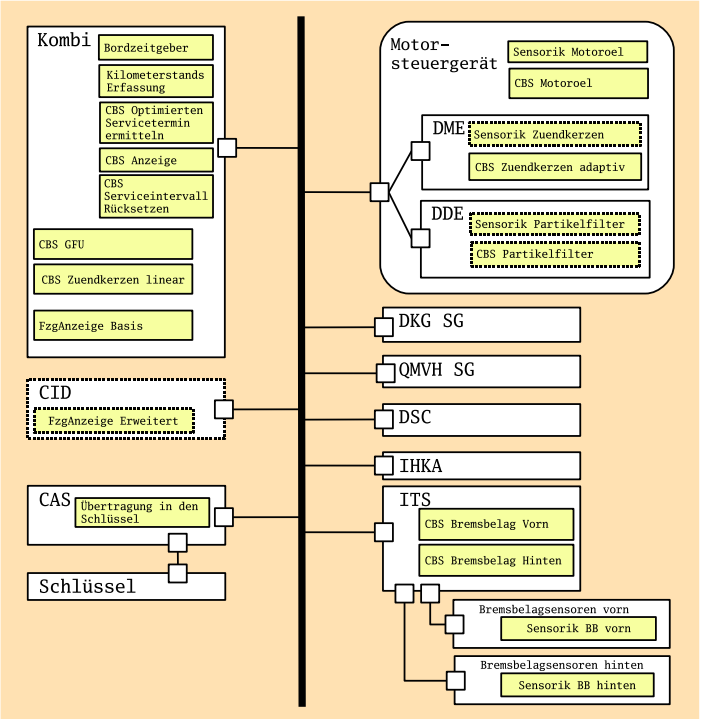


Abbildung 6 Infrastruktur CBS mit Verteilung (Ausschnitt).



sind die Signalpfade zwischen den Steuergeräten eindeutig und können aus der Verteilung der Funktionen abgeleitet werden.

2.1.4 Softwarearchitektur

Für die Softwarearchitektur werden zwei Lösungsalternativen betrachtet. Der wesentliche Unterschied der Lösungen besteht darin, ob generische Softwarekomponenten erstellt werden, die in allen Fahrzeugen (Produkten) installiert werden, oder produktspezifische Softwarekomponenten, die genau die Funktionalität leisten, die in dem jeweiligen Fahrzeug benötigt wird. Beide Lösungen wurden modelliert. Abbildung 7 zeigt die generische Lösung. Jede mögliche Funktion wird durch genau eine Softwarekomponente umgesetzt. Dabei ist zu beachten, dass z. B. der `CBS-Master` (die Realisierung der Zentralfunktion) generisch programmiert werden muss, damit er mit der Situation umgehen kann, dass ein Client fehlt.

Abbildung 7 Softwarearchitektur CBS generisch (Ausschnitt).

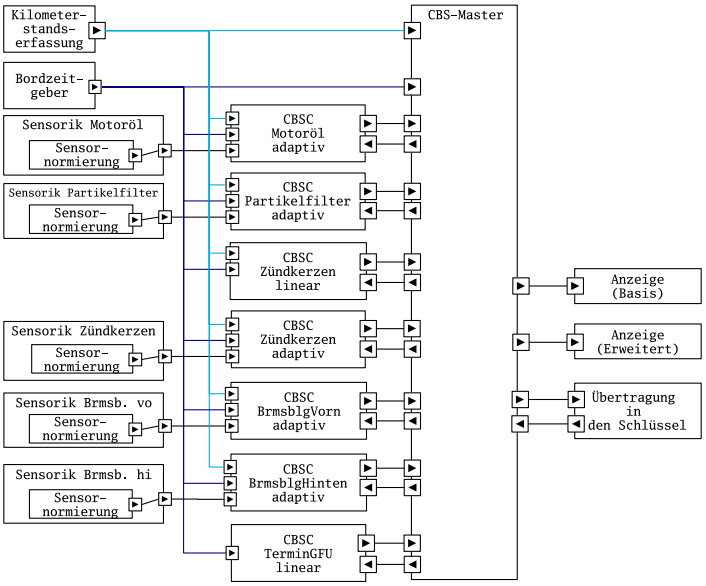
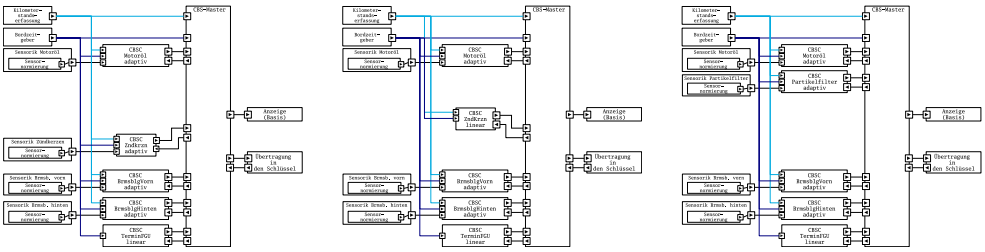


Abbildung 4 zeigt die spezifischen Softwarearchitekturen für die Produkte von Abbildung 4. Es ist zu sehen, dass jeweils nur die Softwarekomponenten in die Architektur übernommen wurden, die für das Produkt relevant sind.

Abbildung 8 Softwarearchitektur CBS spezifisch (Ausschnitt).

(a) *Produkt P1.* (b) *Produkt P2.* (c) *Produkt P3.*



Damit ist die kurze Rekapitulation der Ausgangssituation bezüglich der verwendeten Artefakte von CBS abgeschlossen. Im Anschluss werden Bewertungskriterien und Aufwandstreiber thematisiert, bevor im nächsten Kapitel mit der Analyse der Aufgabe fortgefahren wird.

2.2 Bewertungskriterien

In der zweiten Phase wurden während der Diskussion der Bewertung Kriterien aufgestellt, anhand derer Softwarearchitekturen bewertet werden sollten. Diese Kriterien werden an dieser Stelle noch einmal aufgegriffen und systematisiert.

Zunächst wird eine grundlegende Unterscheidung der Kriterien in Bewertungskriterien der Software und Bewertungskriterien des Umfelds vorgenommen. Mit Hilfe von Softwarekriterien können oft eindeutige Entscheidungen getroffen werden. Die Größe einer Softwarekomponente ist z. B. unstrittig zu bewerten durch »klein=gut« und »groß=schlecht«. Im Gegensatz dazu sind Entscheidungen über Umfeldkriterien nicht so eindeutig. Stabilität z. B. kann in einem Kontext eine positive, in einem anderen Kontext eine negative Eigenschaft sein.

2.2.1 Bewertungskriterien der Software

Nachfolgend sind einige Bewertungskriterien der Software aufgelistet und kurz erläutert.

Ressourcenbedarf
Zum Ressourcenbedarf einer Softwarekomponente gehören z. B.: Codegröße, Laufzeit-Speicherbedarf, Buslast, . . .

Testbarkeit
Testbarkeit ist der Grad, inwieweit die Softwarekomponente Tests unterstützt.

Testaufwand
Der Testaufwand steht in Zusammenhang mit den Testkosten und den Fehlerkosten. Er wird u. a. durch Qualität und Testbarkeit des Testobjekts, die Testinfrastruktur oder die Mitarbeiter-Qualifikation beeinflusst.

Testabdeckung
Die Testabdeckung ist das Verhältnis an tatsächlich getroffenen Aussagen eines Tests gegenüber den theoretisch möglich treffbaren Aussagen. Unabhängig von der konkreten Ausprägung der Testabdeckung ist sie ein wichtiges und bewährtes Bewertungskriterium.

Wartbarkeit
Die Wartbarkeit umfasst z. B Updatefähigkeit, Modularisierung oder Dokumentation.

2.2.2 Bewertungskriterien des Umfelds

Nachfolgend sind einige Bewertungskriterien des Umfelds aufgelistet und kurz erläutert.

Stabilität

Stabilität gibt an, wie stabil eine Softwareproduktlinie sowohl über ihre Laufzeit als auch über die Entwicklungszeit ist. So beeinflussen späte, z. B. zielplattformnahe Entwurfsentscheidungen die Stabilität, aber auch der Ausblick auf Veränderungen in kommenden Fahrzeuggenerationen.

Eine hohe Stabilität sichert langlebige Softwarekomponenten zu. Der Gestaltungsspielraum wird eingeengt, die Flexibilität der Entwickler wird geringer. Dafür steigt der Wiederverwendungsgrad, die Kommunalität der Komponenten erhöht sich. Die langfristige Versorgung mit Softwarekomponenten wird erhöht, der Verwaltungsaufwand für Versionen wird verringert.

Die Stabilität befindet sich also im Spannungsfeld zwischen Stillstand und Bewegung, zwischen Bewährtem und Neuem, zwischen langfristigen und kurzfristigen Lösungen. Daher ist Stabilität nur in ihrem Kontext als anstrebenswert zu definieren. Hersteller und Zulieferer profitieren langfristig von stabilen Produktlinien, Kunden fordern ständige Innovationen. Auch der Zeitpunkt für Innovationen beeinflusst die Stabilität: kundenseitige Software-Updates bedingen andere Konzepte als Updates in der Werkstatt.

Weitere Fragen im Zusammenhang mit Stabilität sind zum Beispiel, wann sich ein Redesign einer Produktlinie lohnt, bzw. bis zu welchem Zeitpunkt evolutionäre Veränderungen besser sind. Eine Rolle können auch organisatorische Zeitpunkte spielen. Zum einen sind einige Zeiten günstiger für Veränderungen als andere, zum anderen gibt es auch innerhalb der Produktentwicklung Bindungszeiten für Varianz, die die Stabilität der Lösung beeinflussen.

Namensraumfestlegungen

Namensraumfestlegungen (englisch *namespaces*), sind zwar ein Kriterium für Stabilität, haben jedoch in der Praxis eine so große Bedeutung, dass ein eigenes Kriterium gerechtfertigt ist.

Namensraumfestlegungen können nur sehr aufwändig beeinflusst werden. Die Ursache hierfür liegt in der besonderen Beziehung von Herstellern und Zulieferern. Die »Herrschaft« über die Namensräume liegt, historisch gewachsen, bei den Zulieferern. Will nun ein Hersteller den Namensraum ändern, bedeutet das Kosten beim Zulieferer und damit beim Hersteller.

Das bedeutet, dass das Kriterium »Namensräume« angibt, welche Kosten die Änderung eines Namensraums voraussichtlich verursachen würde. Diese Kosten sind dann mit denen des Wrapping vorhandener Namensräume zu vergleichen. Daraus ist eine Entscheidung abzuleiten.

Kommunalität / Verbaubarkeit

Die Kommunalität (oder Verbaubarkeit) gibt an, inwieweit eine Softwarekomponente in früheren, späteren oder gleichzeitig existierenden Softwarearchitekturen ohne Änderungen einsetzbar ist. Sie ist daher eine Eigenschaft, die auch bei der Stabilität berücksichtigt wird.

Zertifizierung

Die Zertifizierung von Werkzeugen oder Code ist ein wichtiges Kriterium im Bereich sicherheitsrelevanter Systeme. Da diese Systeme sowie die zu ihrer Herstellung nötigen Werkzeuge zertifiziert werden müssen, wirken sich Änderungen an Code, Methoden oder Werkzeugen direkt über Zertifizierungskosten aus. Diese Kosten umfassen Kosten für Schulungen und die Zertifizierung selbst; aber auch für eventuell nötige Beweise für die Korrektheit von Methoden oder Werkzeugen.

Manipulationssicherheit

Die Manipulationssicherheit dient als Beispiel für ein Kriterium, dessen Beachtung hauptsächlich Kosten verursacht, die mit dem erzielten Nutzen abzuwägen sind. Die Auflösung der Varianz soll als Beispiel dienen: späte Auflösung von Varianz kann durch Laufzeitcodierung oder freischaltbare Codes erfolgen. Damit ein Missbrauch dieser Möglichkeiten ausgeschlossen wird, müssen entsprechende Sicherungsmaßnahmen implementiert werden. Dieser Zusatzaufwand entfällt, wenn die Varianz früher aufgelöst würde.

2.2.3 Kombinierte Betrachtungen

In diesem Abschnitt werden kurz einige Fragen angerissen, die die oben aufgeführten Kriterien im Zusammenhang betrachten.

Codegröße vs. softwarelogistische Aufwände

Die Codegröße ist zunächst im Verhältnis zur Anzahl der Komponenten zu betrachten. Viele kleine Softwarekomponenten bedeuten einen höheren Verwaltungsaufwand. Dieser Verwaltungsaufwand fällt nicht nur beim Hersteller in der Produktion, sondern auch bei Werkstätten an. Die Betrachtung und Bewertung muss daher auch diese Aufwände berücksichtigen.

Zum anderen ist die Beantwortung der Frage, ob eine Produktlinie mittels einer generischen oder mehreren spezifischen Softwarekomponenten realisiert wird, abhängig von der Größe und Anzahl der entstehenden Komponenten. Spezifische Softwarekomponenten sind zwar in der Regel kleiner als generische, müssen aber für jedes Produkt vorgehalten oder generiert werden. Dieser Aufwand ist wiederum sowohl für die Produktion als auch die Werkstätten

zu berücksichtigen. Die Langfristigkeit der Entscheidungen fließt also in die Bewertung ein.

3 Analyse der Bewertungsaufgabe

Die Fachliteratur zur Bewertung von Softwareproduktlinien geht im Allgemeinen von der Annahme aus, dass die Alternative zwischen der Erstellung einer Softwareproduktlinie einerseits und der Erstellung einer dem Umfang der Produktlinie entsprechenden Menge von Einzellösungen andererseits besteht. Diese Situation liegt für die Fallstudie CBS nicht vor: Für jedes Fahrzeug, das heißt für jede mögliche Kombination von zu überwachenden Umfängen je eine eigene Softwarelösung unabhängig von den anderen zu entwickeln, ist offensichtlich keine Option. Die Zielfrage der Fallstudie lautet daher nicht, *ob* eine Softwareproduktlinie erstellt werden soll oder nicht, sondern *wie* diese Softwareproduktlinie beschaffen sein soll.

Um die Frage mit Hilfe von Modellen und Metriken beantworten zu können, muss zunächst genauer analysiert werden, welche Realisierungsmöglichkeiten für Softwareproduktlinien bestehen und wo die für CBS vorgesehenen Lösungen in diesem Lösungsspektrum angesiedelt sind.

3.1 Generische und spezifische Softwarekomponenten

Das Funktionsnetz für CBS ist als Produktlinie mit allen Produktvarianten modelliert. Für die Softwarearchitektur bestehen zwei grundsätzliche Möglichkeiten, die Varianz umzusetzen: Entweder wird eine feste Menge von generischen Softwarekomponenten implementiert, die in allen Fahrzeugen verbaut wird, oder es wird eine größere Auswahl von spezifischen Softwarekomponenten implementiert, aus der für jedes Fahrzeug die individuell passenden ausgewählt und verbaut werden.

Beide Softwarelösungen implementieren alle Produkte der (Funktions-)Produktlinie vollständig. Der wesentliche Unterschied besteht darin, wieviel Software in einem Fahrzeug vorhanden ist. Die beiden theoretischen Extremlösungen sind:

Maximum

Jedes Fahrzeug enthält die gleiche, generische Software. Das heißt, es gibt eine feste Menge von (binären, ausführbaren) Softwarekomponenten, jede mögliche Ausprägung von CBS (jede Funktionsvariante) ist durch diese Menge realisiert und jedes Fahrzeug erhält die komplette Menge. Bei der Maximallösung enthält jedes

Fahrzeug toten Code, da immer mehr Funktionalität vorhanden ist, als benötigt wird.

Die Maximallösung wird auch als *generische Softwarekomponente* bezeichnet. Im Sprachgebrauch von BMW ist oft auch von der *150 %-Lösung* die Rede.

Minimum

Jedes Fahrzeug erhält genau die spezifische Software, die es benötigt. Das heißt, es gibt einen Vorrat von Softwareartefakten (Komponenten und Frameworks) und einen Mechanismus, mit dem die fahrzeugspezifische Software erzeugt werden kann.

Die Minimallösung wird auch als *spezifische Softwarekomponenten* bezeichnet. Im Sprachgebrauch von BMW ist oft auch von der *Produktlinienlösung* die Rede.

Die Minimallösung entspricht der Anforderung an eine Softwareproduktlinie: Die Anforderungen an die Funktionalität der Produktlinie sind durch die Merkmale, deren Kombinationsmöglichkeiten und die damit verbundenen funktionalen Anforderungen gegeben. Die Produktlinienarchitektur als Lösungsarchitektur selbst ist dadurch nicht vorgegeben. Es gibt auch hier theoretische Extreme:

Null-Lösung
: Es gibt für jedes Fahrzeug eine spezifische Menge von Komponenten, das Framework ist leer, der Vorrat von Softwareartefakten ist die Vereinigung der Komponentenmengen. Bezogen auf die Begriffe der Produktlinientechnik bedeutet das: 0 % Gemeinsamkeiten, 100 % Unterschiede.

Optimallösung
: Eine eindeutige Optimallösung gibt es nicht, da es im Allgemeinen keine eindeutige beste Architektur gibt, die alle »objektiven« Gemeinsamkeiten erfasst und die Unterschiede global minimiert.

Vergleich der Lösungen

Im vorigen Kapitel wurden für CBS beide Lösungsmöglichkeiten angegeben: Die Annäherung an die Maximallösung in Form der generischen Softwarekomponente (Abbildung 7) sowie die Annäherung an die Minimallösung in Form der spezifischen Softwarekomponenten für jedes Produkt (Abbildung 8).

Die generische Lösung liegt näher am Maximum, ist aber nicht genau das Maximum, da manche Softwarekomponenten, wie z. B. der Sensor und der Client für die Zündkerzenüberwachung, nur dann ins Fahrzeug gebracht werden, wenn das

zugehörige Steuergerät, die DME, vorhanden ist. Die generische Lösung enthält aber toten Code. So ist z. B. der lineare Zündkerzen-Client immer vorhanden, unabhängig davon, ob zusätzlich ein sensor-adaptiver Zündkerzen-Client oder ein Partikelfilter-Client vorhanden ist. Der sensor-adaptive Zündkerzen-Client macht den linearen Zündkerzen-Client überflüssig, der Partikelfilter-Client schließt jeden Zündkerzen-Client (bzw. seine Funktion) aus.

Da auch bei der generische Lösung fahrzeugspezifische Software verbaut wird, hat sie ebenfalls Produktliniencharakter. Der Unterschied zur Produktlinienlösung liegt im Wesentlichen in der Technik, die verwendet wird, um die fahrzeugspezifische Software zu erstellen. Eine Möglichkeit wäre, Kodierparameter einzusetzen, die die Fahrzeugspezifika abbilden. Eine andere Möglichkeit wäre, das Spezifikum aufgrund der Hardwareauswahl zu bestimmen. Daraus resultieren dann, zumindest in diesem Beispiel, die Unterschiede im Umfang der jeweils fahrzeugspezifischen Software.

Die spezifische Softwarekomponentenlösung soll so gut wie möglich Gemeinsamkeiten erfassen und Unterschiede lokalisieren, also möglichst minimal sein. Ihre vollständige Bestimmung ist erst möglich, wenn die Produkte der Produktlinie definiert sind.

3.2 Bindungszeit von Varianz

Bei beiden Lösungsmöglichkeiten, generischen oder spezifischen Softwarekomponenten, wird die ihnen innewohnende Varianz zu einer bestimmten Zeit aufgelöst. In einem Fahrzeug kann nur eine spezifische Lösung arbeiten. Das heißt, spätestens zur Laufzeit muss genau eine Variante ausgewählt werden, die ausgeführt wird.

Die Auswahl einer Variante wird *Bindung der Varianz* genannt, da die Varianz ab dem Zeitpunkt der Auswahl nicht mehr zur Verfügung steht, sondern gebunden ist. Der Zeitpunkt der Bindung wird *Bindungszeit* (englisch *binding time*) genannt. Da insgesamt alle Varianten realisiert werden müssen, bedeutet das, dass zum Bindungszeitpunkt verschiedene Softwareartefakte erstellt oder verschiedene Teile der Software ausgeführt werden müssen.

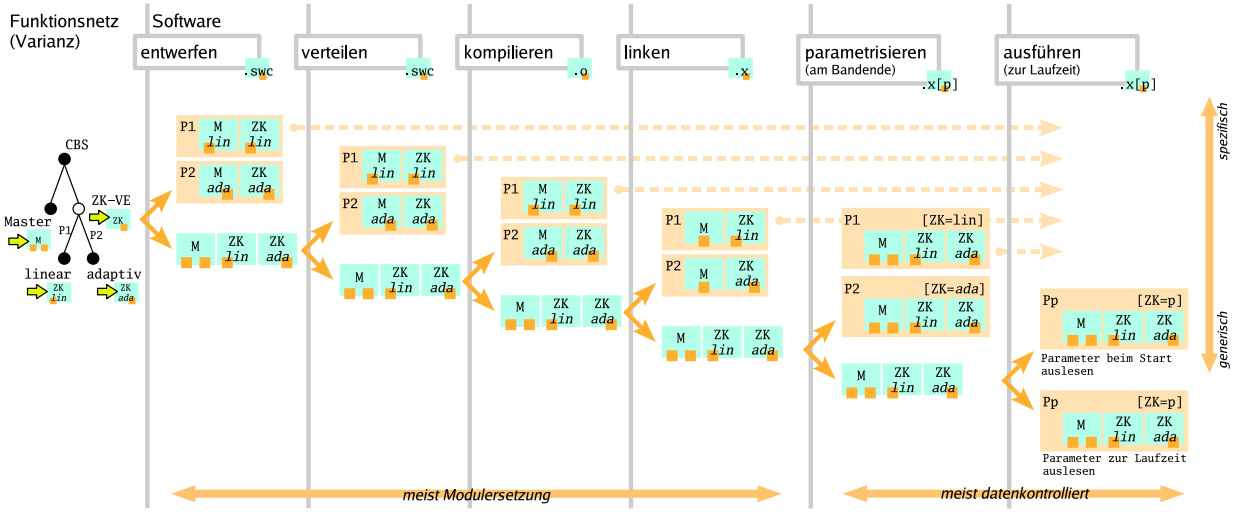
Die Varianz in der Produktlinie zu binden bedeutet nicht, sich bei der Implementierung auf eine Alternative festzulegen. Es müssen immer alle Alternativen implementiert werden. Die Unterschiede zwischen den verschiedenen Bindungszeiten liegen darin, wann (und wie) die Varianz noch intern, innerhalb eines Artefakts

oder einer Artefaktmenge repräsentiert ist, und ab wann mehrere Artefakte (mehrere Artefaktmengen) die verschiedenen Alternativen repräsentieren.

Die Bindungszeiten sind nicht die Entscheidungszeitpunkte im Prozess. Vielmehr ist die Bindungszeit eine Anforderung an die Softwareentwicklung, die von der Implementierung (Codierung) erfüllt werden muss. Für das Beispiel oben bedeutet das, entweder verschiedene Implementierungen für das jeweilige fahrzeugspezifische Verhalten zu entwickeln oder eine Abfrage oder Parametrisierung im Code vorzusehen.

Als Beispiel für diese Diskussion diene ein kleiner Ausschnitt aus CBS (Abbildung 9 ganz links): Als einziger Varianzpunkt wird die Ausprägung des Zündkerzen-Clients als linear (`ZK-lin`) oder adaptiv (`ZK-ada`) betrachtet; als Funktionsumfang der Zündkerzen-Client und der Master. Dadurch sind die funktionalen Anforderungen inklusive der Varianz (Alternative `ZK-lin/ZK-ada` innerhalb der Gesamtfunktion CBS) gegeben. Voraussetzung bzw. Randbedingung sei wieder, dass das Funktionsnetz inklusive der Varianz als funktionale Anforderung vorliegt, ebenso wie die technische Architektur mit der durch die Merkmale bestimmten Varianz (Vorhandensein von Steuergeräten wie DME/DDE etc.) Die Bindungszeitpunkte werden anhand der Entwicklungsartefakte (Modelle) und Stufen des VEIA-Referenzprozesses festgemacht, soweit dies möglich ist. Zum Vergleich und zur Erläuterung werden darüber hinaus die Entwicklungsschritte und -artefakte eines üblichen Softwareentwicklungsprozesses herangezogen.

Abbildung 9 Bindungszeitpunkte für Varianz.



3.2.1 Bindung der Varianz in der Entwicklung

Wird die Varianz in der Entwicklung (im Entwurf) gebunden (*design time*), so geschieht dies während der Entwicklung der ausführbaren Modelle der Softwarearchitektur im VEIA-Referenzprozess. Die betrachteten Modelle sind *AUTOSAR Software Component Descriptions* für die Anwendungskomponenten, im Beispiel werden sie als `.swc`-Files bezeichnet. Die Entwicklung der ausführbaren Modelle entspricht dem Software-Design (C-Files, Klassendiagramme) in der üblichen Softwareentwicklung.

Die Bindung der Varianz in der Entwicklung bedeutet, für jede Variante eine eigene Komponente zu modellieren bzw. zu programmieren. Der Master wird dabei jeweils so modelliert (implementiert), dass er genau mit der passenden Zündkerzen-Variante kommunizieren und arbeiten kann. Es entstehen folgende zwei Produkte:

1. `Master-lin.swc, ZK-lin.swc` für das lineare Produkt
2. `Master-ada.swc, ZK-ada.swc` für das adaptive Produkt

Wird die Varianz nicht zur Entwicklungszeit gebunden so muss der Master so modelliert werden, dass er sowohl mit dem linearen als auch dem adaptiven Client arbeiten kann. Es werden keine Produkte erzeugt, sondern die Varianz bleibt bestehen, die Produktlinie enthält nun drei Komponenten:

- `Master.swc, ZK-lin.swc, ZK-ada.swc`

3.2.2 Bindung der Varianz in der Verteilung

Die Bindung der Varianz in der Verteilung (*allocation time*) greift, wenn die Softwarekomponenten auf die Ressourcen, also die Hardware, die Infrastruktur verteilt werden. Für die Varianzbindung ist wichtig, welche Komponenten miteinander verteilt werden können. Es werden also Produkte gebildet, die sich anhand ihrer Verteilung unterscheiden.

Die drei Komponenten `Master.swc`, `ZK-lin.swc` und `ZK-ada.swc` sind zu verteilen. Im AUTOSAR-Prozess geschieht das, indem entsprechende *System Constraints* angegeben werden; im VEIA-Referenzprozess wird die Verteilungsinformation im Verteilungsmodell *Software* → *Technische Architektur* repräsentiert. Die Bindung der Varianz erzeugt zwei Verteilungsmodelle, bei denen jeweils ein spezifischer Master mit der zugehörigen Zündkerzen-Variante verteilt wird:

1. gemeinsame Verteilung von `Master-lin.swc, ZK-lin.swc` für das lineare Produkt
2. gemeinsame Verteilung von `Master-ada.swc, ZK-ada.swc` für das adaptive Produkt

Wird die Varianz nicht gebunden, bedeutet das, dass alle drei Komponenten verteilt werden, der Master muss wieder entsprechend generisch entworfen werden:

– Verteilung der Komponenten `Master.c, ZK-lin.c, ZK-ada.c`

3.2.3 Bindung der Varianz bei der Generierung / Kompilierung

Die Bindung der Varianz bei der Generierung / Kompilierung (*compile time*) erzeugt die entsprechenden spezifischen Produkte bei der Generierung bzw. bei der Kompilierung. Dafür können Mechanismen wie `if-defs` oder Generator- bzw. Compiler-Parameter verwendet werden, um die Varianz zu binden, dies muss natürlich vorher im Code entsprechend vorgesehen werden. Die Bindung der Varianz erzeugt zwei kompilierte Produkte, jeweils mit spezifischer Master-Komponente:

1. `Master-lin.o, ZK-lin.o` für das lineare Produkt
2. `Master-ada.o, ZK-ada.o` für das adaptive Produkt

Wird die Varianz nicht gebunden, entsteht wieder ein Produkt mit generischer Master-Komponente:

– `Master.o, ZK-lin.o, ZK-ada.o`

3.2.4 Bindung der Varianz beim Linken

Bindung der Varianz beim Linken (*linking time*) bedeutet, dass aus den generischen kompilierten Komponenten spezifische Produkte erzeugt werden. Dies geschieht durch das ausgewählte Linken nur der Komponenten, die in dem Produkt vorkommen sollen. Die Bindung der Varianz erzeugt wieder zwei Produkte, mit jeweils den ausgewählten Komponenten:

1. `Master.x, ZK-lin.x` für das lineare Produkt
2. `Master.x, ZK-ada.x` für das adaptive Produkt

Wird die Varianz nicht gebunden, werden alle Komponenten beim Linken berücksichtigt:

– `Master.x, ZK-lin.x, ZK-ada.x`

3.2.5 Bindung der Varianz durch Parametrisierung

Bei der Bindung der Varianz durch Parametrisierung (*parameterizing time*) wird die Varianz durch einen Parameter aufgelöst, der am Bandende übergeben wird. Dieser Parameter wird auch Kodierparameter genannt. Im Beispiel kann ein Parameter `ZK` übergeben werden, den die Komponenten auswerten können, so dass die Auswahl zwischen linearem und adaptiven Verhalten durchgeführt werden kann. Diese Bindung der Varianz erzeugt zwei Produkte mit unterschiedlichem Parameterwert:

1 `Master.x, ZK-lin.x, ZK-ada.x, ZK=lin` für das lineare Produkt
2 `Master.x, ZK-lin.x, ZK-ada.x, ZK=ada` für das adaptive Produkt

Auch hier gilt wieder: Wird die Varianz nicht gebunden, bleibt das generische Produkt bestehen:

– `Master.x, ZK-lin.x, ZK-ada.x`

3.2.6 Bindung der Varianz zur Laufzeit

Die Bindung der Varianz zur Laufzeit (*runtime*) ist die letzte Möglichkeit, Varianz zu binden. Auch hier gibt es wieder zwei Möglichkeiten: Bindung nur beim Start der Komponente (*startup*) oder die Bindung während der Ausführungszeit (*execution time*). Im Fall der Bindung beim Start, wird der Parameter nur einmal beim Start übergeben, für diese Ausführung ist das Verhalten fest eingestellt:

– `Master.x, ZK-lin.x, ZK-ada.x, param at startup`

Im Fall der Bindung während der Ausführungszeit, reagiert die Komponente fortwährend auf Änderungen des Parameters:

– `Master.x, ZK-lin.x, ZK-ada.x, param at runtime`

3.2.7 Einordnung der CBS-Lösungen

Für CBS wurden zwei Lösungen angegeben: die produktgenerische 150 %-Lösung und die produktspezifische Softwareproduktlinienlösung.

Die 150 %-Lösung besteht ausschließlich aus generischen Softwarekomponenten. Noch zur Laufzeit ist dadurch mehr Funktionalität vorhanden, als im Fahrzeug verwendet werden kann. So sind zum Beispiel alle linearen Umfänge immer im Master implementiert. Die Varianz wird zur Laufzeit dadurch aufgelöst, dass diejenigen Umfänge, die im Fahrzeug nicht vorhanden sind, in der Software nicht abgerufen werden. Allerdings werden nicht alle Softwarekomponenten in jedem Fahrzeug verbaut, sondern nur diejenigen, für die das entsprechende Steuergerät vorhanden ist. Der adaptive Client für die Zündkerzenüberwachung wird nur installiert, wenn das Ottomotorsteuergerät (DME) im Fahrzeug vorhanden ist. Bei der 150 %-Lösung wird ein Teil der Varianz also zur Verteilungszeit gebunden, der Rest zur Laufzeit.

Die Softwareproduktlinienlösung enthält ausschließlich spezifische Softwarekomponenten. Welche Technik zur Generierung der produktspezifischen Software verwendet wird, ist in der Softwarearchitektur nicht festgelegt. Es kann ein Generator für die Konfiguration auf Modellebene, das heißt zur Entwicklungszeit verwendet werden. Die Generierung und damit die konkrete Bindung der Varianz kann aber auch zur Kompilierungs- oder Bindungszeit geschehen.

Allgemein können generische Softwarekomponenten nur entstehen, wenn als Bindungszeiten eine der folgenden Möglichkeiten gewählt wurde:

- Bindung zur Verteilungszeit
- Bindung zur Parametrisierungzeit
- Bindung zur Laufzeit

Spezifische Softwarekomponenten können zu folgenden Zeitpunkten entstehen:

- Bindung zur Entwicklungszeit
- Bindung zur Generierungs-/Kompilierungszeit
- Bindung zur Linkzeit

4 Metrikdefinitionen

In diesem Kapitel werden verschiedene Metriken skizziert, die der Literatur zu Metriken und Produktlinien entnommen wurden und als relevant im Anwendungsfeld des Projekts VEIA erachtet wurden. Die Analyse und Diskussion der Metriken erfolgte dabei nach folgendem Schema:

Formel:
Wie ist die Metrik definiert, wie werden die Maße berechnet?

Eingangsdaten:
Auf welcher Informationsbasis basiert die Metrik? Insbesondere interessiert in diesem Zusammenhang, ob die notwendigen Informationen aus

- dem VEIA- oder AUTOSAR-Datenmodell,
- weiteren Produktdaten (zur Erweiterung des VEIA-Datenmodells),
- sonstigen Daten (Prozess- und Organisationsdaten)

erhoben werden können.

Qualitätsstufen:
Auf welcher Berechnungsgrundlage ist die Metrik definiert und wie gut sind die Ergebnisse der Metrik, insbesondere: ist die Metrik eine Schätzung oder eine Messung?

Folgende Metriken bzw. Arbeiten zu Metriken werden in diesem Kapitel skizziert:

- Kostenbasiertes Softwareproduktlinienmanagement (Abschnitt 4.1)
- Überblick und Kategorisierung von allgemeinen Metriken zu Softwareproduktlinien (Abschnitt 4.2)
- Metriken zur Umfangsmessung und Aufwandsschätzung von Softwareproduktlinien für Automotive-E/E-Systeme (»prozessorientierte Softwaresystemfamilien (PSSF)«) (Abschnitt 4.3)
- Metriken zu strukturellen Eigenschaften in objektorientierten Entwurfsmodellen (am Beispiel von »SDMetrics«) (Abschnitt 4.4)
- Bewertung von Varianz und ihres Einflusses in Architekturen von Softwareproduktlinien (Abschnitt 4.5)
- Metriken zu Servicenutzungsgraden und Kohärenz in Produktlinienarchitekturen (Abschnitt 4.6)

Anmerkung: Die folgenden Abschnitte stellen einen zusammenfassenden Überblick über verschiedene Arbeiten zu Metriken dar, die nicht im Rahmen des VEIA-Projekts entstanden sind und auch nicht von den Autoren des Dokuments erarbeitet wurden. Für die Nachvollziehbarkeit im Rahmen dieses Dokuments wurden Ergebnisse dieser Arbeiten jedoch recht ausführlich übernommen.

4.1 Kostenbasiertes Softwareproduktlinienmanagement

In [SL06a, SL06b] wird der Bedarf festgestellt, Varianz bzgl. der dadurch entstehenden Kosten zu bewerten, da zu viel erlaubte Varianz zu substanziellen Nachfolgekosten während des Lebenszyklus führt.

Als derzeit unzureichend unterstützt und deshalb als Herausforderungen des Produktmanagements für Softwareproduktlinien (SPL) werden u. a. folgende Punkte identifiziert:

- ein unzureichendes Scoping im Lebenszyklus
  (fehlende Dokumentation der Scoping-Entscheidungen, Umgang mit Scoping während Wartung und Evolution)

- hoher Koordinationsaufwand
  Während Wartung und Evolution einer SPL müssen neue oder geänderte Anforderungen integriert werden, die nicht in das initiale Scoping passen und deshalb die Adaption existierender Core-Assets und zusätzliche Varianzpunkte erfordern. Diese Änderungen betreffen geplante Produkte oder auch bereits existierende Produkte, so dass jede Änderung koordiniert werden muss.

- Varianz als Kostentreiber während des gesamten SPL-Lebenszyklus
  (Wartung spezifischer Features bzgl. Integration, Tests, Support etc.)

- »missverstandene Kundenorientierung« bzw. Featureverliebtheit

- fehlende ökonomische Anreize aufgrund fehlender Grundlagen für eine Entscheidungsfindung für oder gegen Varianz (Kostenmodelle für Varianz)

Als Antwort auf die identifizierten Herausforderungen wird ein Ansatz vorgeschlagen, der mehr Transparenz bzgl. der durch Varianz verursachten Kosten ermöglicht und eine fundierte Basis für Entscheidungen im Entwicklungsprozess bereitstellt.

Das vorgeschlagene Varianzmanagement basiert auf der aktivitätsbasierten Kostenschätzung des Kundenwerts der Varianz, um zukünftige Kosten durch Varianz

abzuschätzen, Scoping-Entscheidung zu unterstützen und Varianz strategisch steuern zu können. Der Ansatz verwendet Features als die primären Kostenobjekte.

Das vorliegende Papier beschreibt nur die Ansätze und Problemstellung, die entsprechenden Metriken dazu sind noch nicht entwickelt.

Anwendung im VEIA-Kontext

Unserer Einschätzung nach unterstützt die Anwendung der PFP-Metriken (Abschnitt 4.3) die Umsetzung der oben aufgeführte Anforderungen, weil dort die Charakteristika von Produktlinien (invariante und variable Anteile) in den Metriken explizit berücksichtigt werden und dadurch Umfang und Aufwand, den die Varianz erzeugt, sowie Synergieeffekte bei Wiederverwendung in die Metriken einfließen.

Die in Abschnitt 4.5 skizzierte Arbeit bewertet ebenfalls Varianz bzgl. des Aufwands, sie in einer Architektur zu berücksichtigen bzw. im Nachhinein zu integrieren.

4.2 Überblick und Kategorisierung von allgemeinen Metriken zu Softwareproduktlinien

In [ZC03] wird ein Schema zu Metriken für Softwareproduktlinien vorgestellt. Messungen von Softwareproduktlinien müssen Folgendes adressieren:

- die Entwicklung von Core"=Assets und ihre Nutzung bei der Entwicklung der Produkte in der Produktlinie,
- Charakteristiken der Projektperformanz in der Produktentwicklung,
- gesamte Produktlinienperformanz.

Die Nutzung von Produktlinientechnologien beinhaltet daher drei wesentliche Bereiche für Managementverantwortlichkeiten:

- *Management auf Produktlinienebene* mit der Rolle des *Produktlinienmanagers* und der Verantwortung auf »Unternehmensebene«: Im Fokus steht die gesamte Menge an vergangenen, gegenwärtigen und zukünftigen Produkte, die die Produktlinie ausmachen.
- *Management der Core"=Asset-Entwicklung* mit der Rolle des *Asset Development Managers* und der Verantwortung für die Core-Assets und der zugehörigen Infrastruktur für die systematische Entwicklung der Produkte in einer Produktlinie

– *Management der Produktentwicklung* mit der Rolle des *Produktentwicklungsmanagers*, der für ein »einziges« Produkt verantwortlich ist.

Die Maße in Tabelle 1 stellen eine initiale, nicht vollständige Menge an für Softwareproduktlinien geeigneten Maßen dar. Sie sind entsprechend der drei identifizierten Verantwortlichkeiten zugeordnet. Die in dem Artikel betrachteten Maße sind Maße für Performance, Compliance und Effektivität:

Tabelle 1 Indikatoren und Maße für Softwareproduktlinien [ZC03, Tab. 1].

Objective	Product Line Manager	Asset Development Manager	Product Development Manager
Performance	– Total product development cost – Productivity – Schedule deviation – Time to market – Effort distribution across lifecycle activities – Number of products (past, present, and future) – Trends in defect density	– Cost (effort) to produce core assets – Cost to produce infrastructure – Schedule deviation – Defect density in core assets – Number and type of artifacts in asset library – Core asset quality	– Direct product cost – Defect density in application artifacts – Percent reuse
Compliance	– Mission focus – Architectural conformance – Process compliance	– Mission focus – Process compliance	– Process compliance
Effectiveness	– Return on Investment – Market satisfaction – Market feature coverage	– Return on Investment – Market satisfaction – Market feature coverage	– Customer satisfaction

Anwendung im VEIA-Kontext

Das vorgestellte Schema stellt die aus Managementsicht wesentlichen Rollen und Aufgaben dar, wenn Produktentwicklung auf Basis von Produktlinien durchgeführt wird. Für die Anwendung des Schemas in die Anwendungsdomäne der Automotive-E/E-Systementwicklung sind einige Anpassungen durchzuführen, aufgrund folgender Überlegungen:

Die im Schema vorgegebenen drei Rollen können in der Praxis unter Umstän-

den von einer Person eingenommen werden. Die Rollen werden am *Return of Investment (ROI)* gemessen.

Im Automobilbau werden die Produkte nicht in dem Maße unterschieden, wie in den allgemeinen Softwareproduktlinienansätzen vorausgesetzt wird. Außerdem muss die verteilte Entwicklung von E/E-Systemen beachtet werden. Folgende Analogien zwischen den vorgeschlagenen Rollen und den Verantwortlichkeiten in Automotive-E/E-Entwicklungsprozessen können gezogen werden:

- Die Produktlinie umfasst die baubaren Fahrzeuge in einem Entwicklungsprojekt.

- Assets stellen wiederverwendbare Bausteine dar (verbaubare Komponenten, aber auch wiederverwendbare Konzeptionen wie Modelle, Entwürfe und Muster). Sie bilden die domänenspezifische Anteile in der Produktlinie, die gemeinsam entwickelt und verwendet werden. Als Richtlinie gilt, je mehr wiederverwendbare und gute Assets zur Verfügung stellen, umso besser lässt sich ein Produktlinienansatz fahren.

  Die CBS-Entwicklung kann als Asset-Entwicklung aufgefasst werden, da die Funktionalität in viele Fahrzeuge integriert, aber auch über viele Fahrzeugprojekte hinweg eingesetzt werden soll.

- Die Produkte bzgl. der Rolle des Produktentwicklungsmanagers sind weniger die einzelnen Fahrzeuge, die gebaut werden. Unter Umständen können hier die Entwicklungsbaureihen verstanden werden, die jedoch wiederum Produktliniencharakter haben.

  So wie die Fahrzeugentwicklung im E/E-Bereich heutzutage durchgeführt wird, entspricht die Rolle des Produktentwicklungsmanagers eher einem Steuergeräteverantwortlichen. Als Richtlinie gilt hier, je mehr Assets in einem Steuergerät verwendet werden, umso besser lässt sich ein Produktlinienansatz in den Entwicklungsbaureihe fahren.

  Ziel ist es, CBS als Asset in vielen Steuergeräten und Entwicklungsbaureihen wiederverwenden zu können.

Für die im Papier geforderten Maße bzgl. Wiederverwendungsgrad, Entwicklungskosten und -aufwände lassen sich mit Hilfe der nachfolgend diskutierten Metriken (z. B. die PFP-Metriken in Abschnitt 4.3) erheben.

Das Papier stellt somit einen Anforderungskatalog für wesentliche Rollen und Verantwortlichkeiten aus Prozess-Sicht dar, der den grundsätzlichen Informationsbedarf der Rollen in einer Produktlinienentwicklung beschreibt.

4.3 PFP-Metriken zur Umfangsmessung und Aufwandsschätzung für Softwareproduktlinien

Die Metriken auf Basis von Prozessfamilienpunkten (PFP) zur Umfangsmessung und Aufwandsschätzung für »Prozessorientierte Softwaresystemfamilien« (PSSF) sind im Rahmen des BMBF-Forschungsprojekt »PESOA« entstanden, vgl. [Kie06a, Kie05]. Ein Kurzüberblick wird in [Kie06b, KBS06] gegeben. Als Fallstudie und Basis der Empirie diente die Modellierung einer Benzinmotorsteuerung mit Simulink bei DaimlerChrysler, vgl. [RSW04].

Eingangsdaten

Laut Autor bilden die Informationen aus dem Pflichtenheft[1] und des *Asset Scoping*[2] die notwendige Ausgangsbasis zur Durchführung der Umfangsmessung.

Für die Umfangsmessung sind zudem Simulink-Modelle (Struktur- und Verhaltensmodelle) für die Echtzeitfunktionen eines Benzinmotorsteuergeräts herangezogen worden. Zu den Modellinformationen, die ausgewertet werden, zählen:

- Funktionsspezifikationen bzgl. der Kommunikationsschnittstelle (in Form von Eingangs- und Ausgangssignalen),
- Aussagen, ob es sich um invariante oder variierende Funktionen bzgl. der Produktlinie handelt (gemeinsame vs. variable Bestandteile einer PSSF),
- Aussagen, ob Varianz auf Signalebene vorliegt,
- Verhaltensspezifikation von Funktionen als Statecharts (in Form von Zuständen und Zustandsübergängen),
- Aussagen, inwieweit das Verhalten bgzl. der Produktlinie variierend ist (in Form von variierenden Zuständen und Zustandsübergängen).

[1] *Pflichtenheft* als detaillierte, verbale Beschreibung der Anforderungen an ein neues Produkt, vgl. [Bal01, S. 111]

[2] Das Ergebnis des *Asset Scoping* spezifiziert, welche gemeinsamen und variablen Bausteine einer prozessorientieren Softwaresystemfamilie (PSSF) in wiederverwendbaren Ausprägungen unterstützt werden sollen (vgl. [BKPS04, S. 46]). In VEIA entspricht dies den Aktivitäten zur Erstellung der Produktlinienbeschreibung, vgl. [GEKM07].

Überblick

Im Rahmen der Arbeit sind folgende Softwaremaße hergeleitet worden: unjustierte PFP, justierte PFP und qualitätsjustierte PFP. Unjustierte PFP repräsentieren ein reines Umfangsmaß, wohingegen die (qualitäts-)justierten PFP-Werte als gewichtete Umfangsmaße in erster Linie als Grundlage zur Aufwandsschätzung dienen. Die Aufwandsschätzung erfolgt mittels mathematischem Modell, das auf Basis eines Indikators sowie historischer Erfahrungswerte die Vorhersage künftiger Aufwände ermöglicht.

Umfangsmessung Die PFP-Metriken der Umfangsmessung lassen sich auf Modelle von logischen Architekturen (Funktionsnetze) und auf Softwarearchitekturmodelle anwenden. Es werden strukturelle Informationen aus Architekturmodellen zu Funktionen bzw. Softwarekomponentenbeschreibungen als auch Informationen aus zugehörigen Verhaltensbeschreibungen (Statecharts) herangezogen.

Die Umfangsmessung bzgl. Struktur (Abschnitt 4.3.3) und bzgl. Verhalten (in der Arbeit als *Prozess* bezeichnet) (Abschnitt 4.3.4) erfolgt getrennt. Die berechneten unjustierten PFP-Werte werden anschließend addiert.

Zur Umfangsmessung einer PSSF werden folgende Schritte durchgeführt [Kie06a, S. 58 ff.]:

1. Bestimmung des Zählmodells: Identifikation, ob ein Produkt oder ein Neu- bzw. Weiterentwicklungsprojekt gezählt werden soll,
2. Festlegung des Umfangs der Zählung und der Systemgrenzen (Demarkation),
3. Kategorisierung der Funktionen nach Echtzeitanforderungen (hart vs. weich), nach Lokalität und Wiederverwendung (gemeinsamer vs. variabler Bestandteil einer PSSF),
4. Bestimmung der Komplexität von Echtzeitfunktionen anhand von Strukturbeschreibungen (anhand der Anzahl der Eingangs- und Ausgangssignale),
5. Bestimmung des Umfangsmaßes (unjustierte PFP) für Echtzeitfunktionen anhand von Strukturbeschreibungen unter Berücksichtigung von Wiederverwendung, Komplexität, Lokalität und Erfahrungswerten,
6. Kategorisierung von Prozessen nach Wiederverwendung und Lokalität,
7. Komplexitätsgewichtung der Prozesse anhand der Anzahl der Kanten und Knoten in Statecharts (Komplexitätsmatrix für Echtzeitaspekte),

8 Bestimmung des Umfangsmaßes (unjustierte PFP) für Echtzeitfunktionen anhand ihrer Prozessbeschreibungen unter Berücksichtigung von Wiederverwendung, Lokalität, Komplexität und Erfahrungswerten.

Aufwandsschätzung Um einen Aufwand auf Basis der Umfangsmessung abschätzen zu können, müssen Einflüsse auf die Realisierung einer PSSF ermittelt und entsprechend berücksichtigt werden. Zur Aufwandsschätzung für PSSF werden folgende Schritte durchgeführt [Kie06a, S. 121 ff.; Abb. 43], die in Abschnitt 4.3.5 skizziert werden:

9 Bewertung allgemeiner Aufwandstreiber,
10 Bewertung automotive-spezifischer Aufwandsgeneratoren,
11 Bewertung qualitativer Aufwandsgrößen,
12 Bestimmung justierter PFP-Werte,
13 Bestimmung qualitätsjustierter PFP-Werte,
14 Aufwandsprognose.

Eine Einschätzung der Anwendbarkeit der PFP-Metriken im VEIA-Kontext wird in Abschnitt 4.3.6 gegeben.

4.3.1 Zählmodell (1. Metrik)

Das Zählmodell dient einer Zuordnung zwischen Projekten bzw. Produkten und dem zu ermittelnden Umfangsmaß (siehe zählmodellspezifische Justierung in Abschnitt 4.3.5, Seite 56, vgl. [Kie06a, S. 60 ff.]). Das verwendete Zählmodell definiert als Zähltypen:

- Neuentwicklungsprojekte,
- Wiederverwendungsprojekte (inklusive der Unterscheidung, ob Modifikation einer PSSF oder nur Nutzung einer PSSF) und
- Produktentwicklung auf der Grundlage einer PSSF (Instanziierung).

4.3.2 Demarkation (2. Metrik)

Die Demarkation dient der Festlegung von Grenzlinien, welche die Bestimmung des unjustierten PFP-Werts determinieren und dessen Höhe zentral beeinflussen.

Die Abgrenzung einer PSSF erfolgt anhand

- der Systemgrenzen (Unterscheidung von internen sowie externen Daten bzw. Prozessen und Realisierung einer echtzeit- sowie prozessbezogenen Abgrenzung zwischen den Gemeinsamkeiten und Variabilitäten innerhalb der zu messenden PSSF) sowie
- des Umfangs einer Zählung (anwendungsabhängige Zählgrenze, wodurch ein Über- bzw. Unterschreiten der Systemgrenzen einer einzelnen PSSF ermöglicht wird[3]).

4.3.3 Umfangsmessung von Echtzeitaspekten: Messung von Funktionstypen anhand von strukturellen Merkmalen

3. Metrik: Kategorisierung der Funktion bzgl. Echtzeitanforderungen und Variabilität

Grundlage für die Umfangsmessung ist die Einschätzung der zu analysierenden Funktion, ob sie harten oder weichen Echtzeitanforderungen genügen muss. Des Weiteren wird unterschieden, ob die zu analysierende Echtzeitfunktion zu den variablen[4] oder zu den gemeinsamen Bestandteilen einer PSSF gehört. Die Kombinationen sind in Tabelle 2 dargestellt.

Tabelle 2 Kategorisierung echtzeitorientierter Funktionstypen [Kie06a, Tab. 30].

Funktionstypen	**Wiederverwendung**		**Validierung**	
	Bestandteil der Variabilitäten	**Bestandteil der Gemeinsamkeiten**	**harte Echtzeitanforderung**	**weiche Echtzeitanforderung**
Echtzeitanforderung Variabel Weich (EVW)	+	–	–	+

(Weiter auf der nächsten Seite) ↪ . . .

[3] Der Umfang einer Zählung überschreitet die Systemgrenze einer einzelnen PSSF, wenn im Rahmen des Entwicklungsprojekts mehrere PSSF zusammengeführt werden. Demgegenüber kann der Umfang einer Zählung die Systemgrenze einer PSSF unterschreiten, wenn im Rahmen eines zu messenden Wiederverwendungsprojekts nur eine Untermenge der Bausteine einer PSSF von der Modifikation betroffen ist.

[4] in VEIA: optionale Funktionen bzw. Funktionsalternativen eines Funktionsvarianzpunkts

Funktionstypen	Bestandteil der Variabilitäten	Bestandteil der Gemeinsamkeiten	harte Echtzeitanforderung	weiche Echtzeitanforderung
Echtzeitanforderung Variabel Hart (EVH)	+	–	+	–
Echtzeitanforderung Gemeinsam Weich (EGW)	–	+	–	+
Echtzeitanforderung Gemeinsam Hart (EGH)	–	+	+	–

Eine kurze Charakteristik bzgl. harter und weicher Echtzeitanforderungen ist in Tabelle 3 angegeben (vgl. auch [Kop97, S. 12 f.]).

Tabelle 3 Charakterisierung von harten und weichen Echtzeitanforderungen [Kie06a, Tab. 29].

Merkmal	harte Echtzeitanforderung	weiche Echtzeitanforderung
Spitzenlastenperformance	prognostizierbar	niedrig
Ablaufkontrolle	umgebungsbedingt	computerbedingt
Sicherheit	oft kritisch	oft unkritisch
Redundanztyp	aktiv	kontrollpunktbasiert
Datenintegrität	kurzzeitig	langzeitig
Fehlererkennung	autonom	benutzergeführt

4. Metrik: Klassifizierung der Komplexität von echtzeitorientierten Funktionstypen

Zur Komplexitätsgewichtung der Funktionen (Tabelle 4) wird die Anzahl der verarbeiteten Signale herangezogen, d. h. die Eingangs- als auch die Ausgangssignale der Funktion:

- Eingangssignale (ES): Digitale, analoge oder pulsförmige Signale von Sensoren an ein Steuergerät.
- Ausgangssignale (AS): Stell- und Regelgrößen, welche von einem Steuergerät an Aktuatoren gesendet werden.

Tabelle 4 Klassifizierung der Komplexitität von echtzeitorientierten Funktionstypen, [Kie06a, Tab. 31].

	1 bis 2 ES	**3 bis 5 ES**	**6 oder mehr ES**
1 AS	gering	mittel	hoch
2 AS	mittel	mittel	hoch
3 oder mehr AS	hoch	hoch	hoch

Die in Tabelle 4 dargestellten Komplexitätsgewichte müssen ggf. an den jeweiligen Anwendungskontext angepasst werden; in [Kie06a] waren es *Simulink*-Modelle zur Beschreibung einer Gerätesoftware einer Benzinmotorsteuerung. Diesbezüglich ist die Komplexitätsmatrix für Echtzeitaspekte empirisch abgeleitet.

Weiterhin wurde zur Bewertung nur eine Hierarchieebene (Abstraktionsniveau) zur Bewertung herangezogen, vgl. Argumentation in [Kie06a, S. 94 f.].

5. Metrik: Umfangsmessung für Funktionen anhand ihres Varianztyps

Die in [Kie05, Kie06a] definierten Varianztypen sind nur informell definiert und entsprechen auch nicht vollständig den in VEIA zur Verfügung gestellten Modellierungskonzepten. Sie lassen sich aber folgendermaßen in die VEIA-Konzepte einordnen:

- *horizontal variable Funktion (EVW / EVH)* ≙ optionale Funktion oder alternative Funktion eines Funktionsvarianzpunkts,
- *horizontal gemeinsame Funktion (EGW / EGH)* ≙ (absolut) invariante Funktion,
- *vertikal gemeinsame Funktion* ≙ invariante hierarchische Funktion oder variierende Funktion (Funktionsvarianzpunkt) mit wenig interner Varianz, d. h. ein paar optionalen Signalen (variierende Signale sind wahrscheinlich nicht vorgesehen gewesen),
- *vertikal variable Funktion* ≙ invariante hierarchische Funktion oder variierende Funktion (Funktionsvarianzpunkt) mit »großer« interner Varianz, d. h. mehrheitlich unterschiedlichen Signalmengen.

Für (absolut) invariante Funktionen wird die unjustierte PFP wie folgt berechnet:

Formel 4.1

$$unjustierte\ PFP_{EGW/EGH} = \frac{KG_{g/m/h} * Umrechnungsfaktor_{gem}}{PA}$$

wobei die Variable *PA* für die Produktanzahl[5] und die Variable *KG* für komplexitätsabhängige Korrekturfaktoren für gemeinsame Funktionstypen[6] stehen.

Tabelle 5 Unjustierte PFP für invariante Funktionen, lt. [Kie06a, Tab. 39].

Komplexität	**gering**	**mittel**	**hoch**
unjustierte PFP für EGW	$\frac{KG_g*10}{PA}$	$\frac{KG_m*15}{PA}$	$\frac{KG_h*23}{PA}$
unjustierte PFP für EGH	$\frac{KG_g*15}{PA}$	$\frac{KG_m*23}{PA}$	$\frac{KG_h*35}{PA}$

Für optionale Funktionen oder Funktionsalternativen, die unterschiedlich häufig verbaut werden, wird die unjustierte PFP wie folgt berechnet:

Formel 4.2

$$unjustierte\ PFP_{EVW/EVH} = \frac{KV_{g/m/h} * Umrechnungsfaktor_{var}}{IH}$$

wobei die Variable *IH* für die Implementierungshäufigkeit[7] und die Variable *KV* für komplexitätsabhängige Korrekturfaktoren für variable Funktionstypen[8] stehen.

Tabelle 6 Unjustierte PFP für optionale Funktionen, lt. [Kie06a, Tab. 38].

Komplexität	**gering**	**mittel**	**hoch**
unjustierte PFP für EVW	$\frac{KV_g*5}{IH}$	$\frac{KV_m*7}{IH}$	$\frac{KV_h*10}{IH}$

(Weiter auf der nächsten Seite) ↪ . . .

[5] Die gemeinsamen Assets einer PSSF sind in jedem einzelnen aus der PSSF-Infrastruktur abgeleiteten Produkt vollständig implementiert. Dementsprechend ist die umfangsbestimmende Wiederverwendung dieser gemeinsamen Bausteine durch die aus einer PSSF generierte Produktanzahl determiniert.

[6] Diese Korrekturfaktoren sind organisationsspezifische Erfahrungswerte. Ohne historische Erkenntnisse werden diese ausgeschaltet: $KG_{g/m/h} = 1$.

[7] Je zahlreicher eine EVW / EVH in den zu generierenden Varianten einer PSSF implementiert oder durch diese Assets referenziert wird, desto höher ist die Wiederverwendung jener variablen Funktionstypen.

[8] Diese Korrekturfaktoren sind organisationsspezifische Erfahrungswerte. Ohne historische Erkenntnisse werden diese ausgeschaltet: $KV_{g/m/h} = 1$.

Komplexität	gering	mittel	hoch
unjustierte PFP für EVH	$\frac{KV_g*7}{IH}$	$\frac{KV_m*10}{IH}$	$\frac{KV_h*15}{IH}$

Funktionen mit variierenden Signalmengen Unter Berücksichtigung variabler Mengen an Eingangs- und Ausgangssignalen werden die zuvor angegebenen unjustierten PFPs noch entsprechend gewichtet, um »vertikale Variabilität« zu berücksichtigen. Dazu wird das Verhältnis zwischen der Anzahl an gemeinsamen Signalen und der Gesamtzahl der Signale (*G*) bzw. das Verhältnis zwischen der Anzahl variabler Signale und der Gesamtzahl der Signale (*V*) einer Funktion bestimmt.[9]

Unter Berücksichtigung der variablen Signalmengen einer Funktion wird der unjustierte PFP wie folgt berechnet:

Formel 4.3

$$unj.\ PFP_{vert.\ EVW/EVH} = (1-G)\frac{KV_{g/m/h} * UF_{var}}{IH} + G\frac{KG_{g/m/h} * UF_{gem}}{PA}$$

Formel 4.4

$$unj.\ PFP_{vert.\ EGW/EGH} = V\frac{KV_{g/m/h} * UF_{var}}{IH} + (1-V)\frac{KG_{g/m/h} * UF_{gem}}{PA}$$

wobei

- G = Anteil gemeinsamer Signale, mit $G = \frac{B_G}{C}$
  (B_G = Anzahl gemeinsamer Eingangs- und Ausgangssignale der Echtzeitfunktion, C = Anzahl sämtlicher Eingangs- und Ausgangssignale der Echtzeitfunktion),
- V = Anteil variabler Signale mit $V = \frac{B_V}{C}$
  (B_V = Anzahl variabler Eingangs- und Ausgangssignale der Echtzeitfunktion, C = Anzahl sämtlicher Eingangs- und Ausgangssignale der Echtzeitfunktion),

[9] Auch wenn es in [Kie06a] nicht so dargestellt ist, gilt: $V = 1-G$. Ebenda werden die Funktionen in »vertikal variable Echtzeitfunktionen«, für die der Faktor *G* verwendet wird, und in »vertikal gemeinsame Echtzeitfunktionen«, für die der Faktor *V* verwendet wird, eingeteilt. Jeweils soll gelten: $0 \leq G \leq 1/2$ bzw. $0 \leq V \leq 1/2$. Je näher *G* an $1/2$, desto mehr Gemeinsamkeiten hat ein vertikal variable Echtzeitfunktion. Analog: Je näher *V* an $1/2$, desto mehr Variabilitäten hat eine vertikal gemeinsame Echtzeitfunktion.

- IH = Implementierungshäufigkeit
- PA = Produktanzahl
- KG bzw. KV = komplexitätsabhängiger Korrekturfaktor für gemeinsame bzw. variable Funktionstypen, repräsentiert organisationsspezifische Erfahrungswerte.
- UF_{gem} bzw. UF_{var} = Umrechnungsfaktor für gemeinsame bzw. variable Funktionstypen, in Abhängigkeit sowohl der Echtzeitanforderung als auch der Komplexität, siehe Tabelle 7.

Tabelle 7 Umrechnungsfaktoren UF_{gem} und UF_{var}, lt. [Kie06a, Tab. 41 und 43].

Echtzeitkategorie	Komplexität	Umrechnungsfaktor UF_{gem}	Umrechnungsfaktor UF_{var}
weich	**gering**	10	5
	mittel	15	7
	hoch	23	10
hart	**gering**	15	7
	mittel	23	10
	hoch	35	15

4.3.4 Umfangsmessung von Prozessen: Messung von Funktionstypen anhand der Verhaltensspezifikation

6. Metrik: Kategorisierung von Verhaltensbeschreibungen bzgl. Wiederverwendung und Lokalität

Bzgl. ihrer Verhaltensbeschreibungen werden Funktionen (»Prozesse«) wie folgt kategorisiert (»Lokalitätsfokus«):

- bzgl. der Kommunikationsrichtung: in unidirektionale Eingabe- oder Ausgabeprozesse und in bidirektionale Prozesse,
- bzgl. der Systemgrenze: in interne Prozesse, deren Eingabe, Verarbeitung und Ausgabe sich innerhalb der Systemgrenze vollzieht, und in Prozesse, die über die Systemgrenze Eingaben erhalten bzw. Ausgaben erstellen.

Außerdem erfolgt, wie auch schon bei den Architekturmodellen, die Unterscheidung hinsichtlich der Wiederverwendung in gemeinsame und variable Bausteine einer PSSF.

Die Kombinationen sind in Tabelle 8 dargestellt.

Tabelle 8 Charakterisierung von harten und weichen Echtzeitanforderungen [Kie06a, Tab. 16].

Funktionstypen	**Wiederverwendung**		**Lokalität**			
	Bestandteil der Variabilitäten	**Bestandteil der Gemeinsamkeiten**	**Eingabe außerhalb PSSF-Systemgrenze**	**Ausgabe außerhalb PSSF-Systemgrenze**	**Eingabe innerhalb PSSF-Systemgrenze**	**Eingabe außerhalb PSSF-Systemgrenze**
Prozess Variabel Intern (PVI)	+	–	–	–	+	+
Prozess Variabel Unidirektional (PVU)	+	–	–	+	+	–
	+	–	+	–	–	+
Prozess Variable Bidirektional (PVB)	+	–	+	+	–	–
Prozess Gemeinsam Intern (PGI)	–	+	–	–	+	+
Prozess Gemeinsam Unidirektional (PGU)	–	+	–	+	+	–
	–	+	+	–	–	+
Prozess Gemeinsam Bidirektional (PGB)	–	+	+	+	–	–

7. Metrik: Klassifizierung der Komplexität anhand von Verhaltensbeschreibungen

Zur Komplexitätsgewichtung wird die Anzahl der Knoten und Kanten des Statechart der Funktion herangezogen. Die Komplexität wird in *niedrig*, *mittel* und *hoch* kategorisiert, siehe Tabelle 9.

Tabelle 9 Klassifizierung der Komplexitität von Funktionsverhalten, [Kie06a, Tab. 44].

	1 bis 2 Knoten	**3 bis 5 Knoten**	**6 oder mehr Knoten**
1 bis 5 Kanten	gering	mittel	hoch
6 bis 8 Kanten	mittel	mittel	hoch
9 oder mehr Kanten	hoch	hoch	hoch

Unter Kanten wird jede (Teil-)Transition gerechnet (auch solche zwischen Pseudozuständen).

Zu den Knoten werden hier einfache Zustände, zusammengesetzte Zustände, Unterzustandsautomatenzustände, Startzustände, Endzustände und Pseudozustände gezählt.

Die in Tabelle 9 dargestellten Grenzwerte stammen aus einer von DaimlerChrysler bereitgestellten Studie zur Beschreibung der Variabilität einer Benzinmotorsteuerung (vgl. [RSW04, S. 40 ff.]). Diesbezüglich handelt es sich um eine empirisch abgeleitete Komplexitätsmatrix.

8. Metrik: Umfangsmessung anhand von Verhaltensbeschreibungen von Funktionen

Analog zu Architekturmodellen erfolgt eine Unterteilung der Prozesse in »horizontale« und »vertikale« Gemeinsamkeiten und Variabilitäten. Vgl. hierzu die Diskussionen in [Kie06a, S. 76 ff.]. Die allgemeine Formel für unjustierte PFP für Prozesse lautet:

Formel 4.5

$$unj.\ PFP_{vert.\ PVI/PVU/PVB} = (1-G)\frac{KV_{g/m/h} * UF_{var}}{IH} + G\frac{KG_{g/m/h} * UF_{gem}}{PA}$$

Formel 4.6

$$unj.\ PFP_{vert.\ PGI/PGU/PGB} = V\frac{KV_{g/m/h} * UF_{var}}{IH} + (1-V)\frac{KG_{g/m/h} * UF_{gem}}{PA}$$

wobei

- G = Anteil gemeinsamer Knoten und Kanten im Statechart, mit $G = \frac{B_G}{C}$ (B_G = Anzahl gemeinsamer Knoten und Kanten im Prozess, C = Anzahl sämtlicher Knoten und Kanten im Prozess),
- V = Anteil variabler Knoten und Kanten im Statechart mit $V = \frac{B_V}{C}$ (B_V = Anzahl variabler Knoten und Kanten im Prozess, C = Anzahl sämtlicher Knoten und Kanten im Prozess),
- IH = Implementierungshäufigkeit
- PA = Produktanzahl
- KG bzw. KV = komplexitätsabhängiger Korrekturfaktor zur Berücksichtigung historischer Erfahrungswerte und systemfamilienspezifischer Entwicklungseinflüsse.

– UF_{gem} bzw. UF_{var} = Umrechnungsfaktor für gemeinsame bzw. variable Prozesse, in Abhängigkeit sowohl der Echtzeitanforderung als auch der Komplexität, siehe Tabelle 10.

Tabelle 10 Umrechnungsfaktoren UF_{gem} und UF_{var}, lt. [Kie06a, Tab. 41 und 43].

Prozesskategorie	**Komplexität**	**Umrechnungsfaktor UF_{gem}**	**Umrechnungsfaktor UF_{var}**
PVI	**gering**	10	5
	mittel	15	7
	hoch	23	10
PVU	**gering**	$12\frac{3}{16}$	$5\frac{15}{16}$
	mittel	$18\frac{9}{16}$	$8\frac{5}{16}$
	hoch	$28\frac{7}{16}$	$12\frac{3}{16}$
PVB	**gering**	15	7
	mittel	23	10
	hoch	35	15
PGI	**gering**	10	5
	mittel	15	7
	hoch	23	10
PGU	**gering**	$12\frac{3}{16}$	$5\frac{15}{16}$
	mittel	$18\frac{9}{16}$	$8\frac{5}{16}$
	hoch	$28\frac{7}{16}$	$12\frac{3}{16}$
PGB	**gering**	15	7
	mittel	23	10
	hoch	35	15

Für horizontale Variabilitäten (PVB, PVU, PVI) spielt die Implementierungshäufigkeit (IH) innerhalb der PSSF eine umfangsverringernde Wirkung: Varianten, die häufig wiederverwendet werden, sind günstiger als solche, die nur für ein Produkt zum Zuge kommen. Für horizontale Gemeinsamkeiten ist die Produktanzahl der Produktlinie maßgeblich.

Für vertikale Variabilitäten, die mehr Unterschiede als Gemeinsamkeiten in ihrer Verhaltensbeschreibung in Form von Knoten und Kanten haben, sowie für vertikalen Gemeinsamkeiten, die mehr Gemeinsamkeiten als Unterschiede in ihrer Verhaltensbeschreibung in Form von Konten und Kanten haben, wird der Ähnlichkeitsfaktor in Bezug zur Produktanzahl und zur Implementierungshäufigkeit eingerechnet.

4.3.5 Aufwandsschätzung

Um einen Aufwand auf Basis der Umfangsmessung durchführen zu können, müssen Einflüsse auf die Realisierung einer PSSF ermittelt und entsprechend berücksichtigt werden. Diese werden zur Justierung verwendet. Danach erfolgt die Aufwandsprognose mittels Schätzgleichungen.

9. Metrik: Generelle externe Einflussfaktoren

Die Einflussfaktoren auf den Entwicklungsaufwand werden in generelle Einflüsse und automotive-spezifische Einflüsse aufgeteilt.

Die generellen Einflussfaktoren sind obligatorisch zu erfassen, denn sie ermöglichen eine Berücksichtigung der Auswirkungen des Anwendungsumfelds der Projektaufgabe auf den zu prognostizierenden Projektumfang. Die beispielhaft angegebenen Einflussfaktoren erfassen diese domänenübergreifenden Kostentreiber, ohne dabei Anspruch auf Vollständigkeit zu erheben.

Die 20 exemplarischen Subkriterien bestimmen auf Basis ihrer Ausprägungen den numerischen Einfluss von vier Faktoren und erhöhen oder mindern den zu schätzenden PSSF-Aufwandsindikator. Die einzelnen Subkriterien sind mit einer Frage hinterlegt, deren bejahende oder verneinende Beantwortung eine erhöhende oder mindernde Aufwandsauswirkung hat. Durch Zählung der aufwandserhöhenden Ausprägungen wird der numerische Einflussgrad bestimmt.

Dokumentation

(Beantwortung mit *ja* ist aufwandserhöhend.)

A1 Technisch & Funktional: Sind technische Spezifikationen und / oder Pflichtenhefte zu erstellen?

A2 Prognose: Ist eine Dokumentation des Mess- / Schätzvorgehens vorgesehen?

A3 Integriert: Ist die Erstellung integrierter Dokumentationen beabsichtigt?

A4 Benutzerunterlagen: Sind unterstützende Dokumente für den Endanwender zu erstellen?

A5 Fehler & Test: Werden Testdokumente erstellt und / oder wird ein Fehlerbuch geführt?

Infrastruktur

Der Begriff der Infrastruktur umfasst die »Gesamtheit der Anlagen, Einrichtungen und Gegebenheiten, die den Wirtschaftseinheiten als Grundlage ihrer Aktivitäten vorgegeben sind.« Die Infrastruktur eines PSSF-Projekts bestimmt

die Arbeitsproduktivität und dementsprechend den Entwicklungsaufwand der PSSF.

(Beantwortung mit *nein* ist aufwandserhöhend.)

A6 Entwicklungserfahrung: Sind praktische Erfahrungen in der PSSF-Entwicklung vorhanden?

A7 Domänenkenntnisse: Liegen Kenntnisse im Bereich der Anwendungsdomäne vor?

A8 CASE-Werkzeuge: Sind an die PSSF-Entwicklung angepasste CASE-Werkzeuge verfügbar?

A9 PSSF-Methoden: Kann auf etablierte PSSF-Entwicklungsmethoden zurückgegriffen werden?

A10 CSCW-Methoden: Wird eine CSCW-Anwendung zur Arbeitsunterstützung eingesetzt?

Transitionsprozess

Der Transitionsprozess beschreibt die Wandlung einer Unternehmung vom aktuellen Stand der Softwareentwicklung zu einer PSSF-orientierten Produktentwicklung, welche an die individuellen Organisationsbedürfnisse angepasst ist.

(Beantwortung mit *ja* ist aufwandserhöhend.)

A11 Scope: Ist eine Umgestaltung des aktuellen Scopes notwendig?

A12 Architektur: Muss die bisherige Softwarearchitektur modifiziert werden?

A13 Organisationsstruktur: Erscheint die Umgestaltung von Organisationsstrukturen erforderlich?

A14 Institutionalisierung: Ist die Institutionalisierung einer allgemeinen PSSF-Orientierung obligatorisch?

A15 Reifebestimmung: Sind die PSSF-orientierten Entwicklungsprozesse einer Reifebestimmung zu unterziehen?

Wissensvermittlung

(Beantwortung mit *ja* ist aufwandserhöhend.)

A16 Hilfefunktionen: Werden passive und/oder aktive Hilfefunktionen innerhalb der PSSF realisiert?

A17 Simulationsumgebung: Ist die Entwicklung einer Simulationsumgebung zu Schulungszwecken geplant?

A18 Electronic Learning: Wird eine Wissensvermittlung durch Electronic Learning angestrebt?

A19 Blended Learning: Erfolgt eine Schulung der Anwender durch Blended Learning?

A20 Präsenzveranstaltung: Werden Präsenzveranstaltungen ausgearbeitet und angeboten?

Die durch die PFP-Makroanalyse erfassten, domänenübergreifenden Einflusskriterien der Dokumentation, Infrastruktur, Transitionsprozesse und Wissensvermittlung stellen lediglich eine exemplarische Auswahl allgemeiner Aufwandsgeneratoren der 9. Metrik dar. Demzufolge wurde die herzuleitende Berechnung des PFP-Justierungseinflusses hochflexibel gestaltet, um eine umfassende Modifikation der obigen Referenzkriterien uneingeschränkt zu ermöglichen.

10. Metrik: Automotive-spezifische externe Einflussfaktoren

Zur obligatorischen Justierung der Umfangsmessung von PSSF werden für eingebettete Steuergeräte (ohne Anspruch auf Vollständigkeit) folgende Einflussfaktoren durch jeweils fünf Subkriterien erfasst.

Die einzelnen Subkriterien sind mit einer Frage hinterlegt, deren bejahende oder verneinende Beantwortung eine erhöhende oder mindernde Aufwandsauswirkung hat. Durch Zählung der aufwandserhöhenden Ausprägungen wird der numerische Einflussgrad bestimmt.

Rechenleistung
(Beantwortung mit *ja* ist aufwandserhöhend.)

C1 Sprachniveau: Werden Maschinensprachen anstatt von Hochsprachen verwendet?

C2 Parallelisierung: Erfolgt die Entwicklung parallel ablaufender, verteilter Softwarebausteine?

C3 Design & Algorithmen: Wird das Software"-design und/oder werden die Algorithmen auf eine Laufzeitoptimierung ausgerichtet?

C4 Zugriffszeiten: Erfahren die Zugriffszeiten auf unterschiedliche Speichersegmente eine Berücksichtigung?

C5 Datenlayout: Ist das Layout der adressierten Daten geschwindigkeitsoptimiert festzulegen?

Sicherheit
(Beantwortung mit *ja* ist aufwandserhöhend.)

C6 Überwachung: Erfolgt ein frühzeitiges Überwachen der Sicherheitsanforderungen?

C7 FMEA-Analyse: Wird eine vorausschauende Gefahrenidentifikation durch die FMEA durchgeführt?

C8 FTA-Analyse: Ist die Anwendung der FTA zur Erkennung von Gefahrenursachen eingeplant?

C9 Fahrzeugsubsystem: Ist das Steuergerät Bestandteil eines sicherheitsrelevanten Fahrzeugsubsystems?

C10 Redundanz: Sind berechnete Werte und Konstanten für den Fall eines Sensordefekts redundant vorzuhalten?

C11 Prognosewerkzeuge: Ist ein Werkzeug zur Abschätzung der benötigten Speichergröße verfügbar? (Beantwortung mit *nein* ist aufwandserhöhend.)

C12 Kennlinien & Kennfelder: Erfolgt eine Ressourcenoptimierung von Kennlinien und Kennfeldern? (Beantwortung mit *ja* ist aufwandserhöhend.)

C13 Prioritätsansatz: Erfolgt eine Speicheroptimierung nach Relevanz einzelner Bausteine der Steuersoftware? (Beantwortung mit *ja* ist aufwandserhöhend.)

C14 Virtualisierung: Werden virtuelle Speicher-Ressourcen zur Verfügung gestellt? (Beantwortung mit *nein* ist aufwandserhöhend.)

C15 Caching: Müssen Daten durch Caching-Verfahren zwischengespeichert werden? (Beantwortung mit *ja* ist aufwandserhöhend.)

Analog zu den allgemeinen Aufwandstreibern stellen die drei exemplarischen Einflussfaktoren der Rechenleistung, der Sicherheit und des Speicherumfangs wiederum beispielhafte Referenzparameter der domänenspezifischen PFP-Makroanalyse dar. Folglich wird auch diesbezüglich eine hoch flexible Justierung durch die PFP-Metriken unterstützt.

11. Metrik: Evaluation qualitativer Aufwandsgrößen

Die Metrik zur Bewertung der qualitativen Aufwandsgrößen basiert auf dem ISO/IEC 9126-Qualitätsmodell und berücksichtigt die Auswirkungen der Qualitätsanforderungen an ein Softwareprojekt auf den Umfang der zu entwickelnden PSSF.

Die (domänenübergreifenden) qualitativen Einflüsse werden in sechs Kategorien unterteilt, wobei die Anzahl der Subkriterien unterschiedlich sein kann. Die einzelnen Subkriterien werden als *hoch*, *mittel* oder *gering* bewertet. Der numerische Einflussgrad einer Kategorie (QEG_K) berechnet sich aus der Anzahl der als *hoch* und der Anzahl der als *gering* bewerteten Subkriterien in Relation zur vorgegeben Anzahl der Subkriterien wie folgt:

Formel 4.7

$$QEG_K = \frac{\#hohe\ Auspraegungen - \#geringe\ Auspraegungen}{\#Subkriterien_K}$$

Für die Qualitätsjustierung werden dann die numerischen Einflussgrade von sämtlichen qualitativen Kostengrößen addiert (die Summe der qualitativen Einflussgrade

SQ). Der SQ-Wert kann im Grenzfall zwischen -6 und $+6$ variieren.

Formel 4.8

$$SQ = \sum_{i=1}^{\#K} QEG_i$$

Hier die vorgeschlagenen Kriterien:

Funktionalität

D16 Angemessenheit: Wie hoch sind die Anforderungen zur Bereitstellung von Funktionalitäten, die den spezifizierten Aufgaben und Zielen gerecht werden?

D17 Genauigkeit: Mit welchem Anspruch müssen die vereinbarten Ergebnisse und Leistungen mit dem benötigten Grad an Präzision bereitgestellt werden?

D18 Interoperabilität: In welchem Umfang muss das Softwareprodukt die Fähigkeit besitzen, mit einem oder mehreren Softwaresystemen zu interagieren?

D19 Schutz: Wie bedeutsam ist der Schutz von Informationen und Daten vor unberechtigtem Zugriff sowie die Realisierung eines Zugangsmanagements für autorisierte Nutzer?

D20 Konformität: Mit welchem Anspruchsgrad müssen Standards, Konventionen oder rechtliche Bestimmungen und andere funktionelle Anordnungen eingehalten werden?

Zuverlässigkeit

D21 Reife: Wie hoch sind die qualitativen Ansprüche an das Softwareprodukt, Ausfälle zu vermeiden, die aus internen Softwarefehlern resultieren?

D22 Fehlertoleranz: Wie umfassend ist die Forderung nach einer Beibehaltung der Funktionseigenschaften im Falle von Softwarefehlern oder Schnittstellenverletzungen?

D23 Regenerierung: In welchem Umfang müssen, im Falle eines Softwarefehlers, bestimmte Funktionen wiedererlangt und direkt betroffene Daten zurückgewonnen werden?

D24 Konformität: Wie bedeutsam ist die Einhaltung von Standards, Konventionen oder Verordnungen im Zusammenhang mit Zuverlässigkeitsanforderungen?

Anwendbarkeit

D25 Verständlichkeit: Inwiefern soll der Nutzer bei der Einschätzung unterstützt werden, ob die Software den Anforderungen entspricht und eingesetzt werden kann?

D26 Erlernbarkeit: In welchem Ausmaß hat der Benutzer durch die Software eine Unterstützung zu erfahren, um die Anwendung des Softwareprodukts leichter zu erfassen?

D27 Beherrschbarkeit: In welchem Umfang soll es dem Anwender ermöglicht werden, den Betrieb des Softwareprodukts zu steuern und zu kontrollieren?

D28 Attraktivität: Mit welcher Intensität hat das Softwareprodukt eine visuell fokussierte Anziehungskraft auf den Benutzer auszuüben?

D29 Konformität: Wie komplex ist die Einhaltung von Standards, Konventionen, Entwurfsanleitungen oder Verordnungen zu den Anforderungen der Anwendbarkeit?

Effizienz

D30 Zeitverhalten: In welchem Umfang müssen angemessene Antwort und Bearbeitungszeiten sowie Durchsatzraten während der Ausführung realisiert werden?

D31 Ressourcennutzung: Wie hoch sind die Anforderungen an eine adäquate, mengen- und typbezogene Nutzung von Ressourcen während des Softwarebetriebs?

D32 Konformität: Mit welchem Aufwand ist die Einhaltung von Standards und Konventionen verbunden, die im Kontext von Effizienzanforderungen existieren?

Wartbarkeit

D33 Analysierbarkeit: In welchem Ausmaß soll die Analyse von Defiziten oder Fehlerursachen sowie die Identifikation der zu modifizierenden Softwarebausteine möglich sein?

D34 Änderbarkeit: Wie aufwändig ist die Anforderung einzuschätzen, eine spätere Implementierung von bestimmten Modifikationen zu ermöglichen?

D35 Stabilität: Mit welchem Qualitätsanspruch ist der Vermeidung unerwarteter Effekte, welche aus den Modifikationen des Softwareprodukts resultieren, entgegenzutreten?

D36 Testbarkeit: Wie hoch ist der Aufwand, eine sich an die Modifikation des Softwareprodukts anschließende Gültigkeitsprüfung zu ermöglichen?

D37 Konformität: In welchem Umfang erfolgt die Einhaltung von Standards und Konventionen, die im Zusammenhang mit Aspekten der Wartbarkeit stehen?

Portabilität

D38 Anpassungsfähigkeit: In welchem Umfang muss sich die Software mit denen für ihren originären Zweck angebotenen Mitteln an verschiedene Umgebungen angleichen?

D39 Installierbarkeit: Wie hoch ist der Anspruch an das Softwareprodukt, eine Installation in bestimmten Umgebungen zu ermöglichen?

D40 Koexistenz: Wie komplex ist die Realisierung einer gemeinsamen Existenz und kollektiven Ressourcennutzung mit anderen, unabhängigen Softwareprodukten?

D41 Ersetzbarkeit: In welchem Ausmaß soll die Software eine Substitution anderer Produkte ermöglichen, die in gleicher Zweck-/Umgebungskonstellation existieren?

D42 Konformität: Mit welchem Aufwand ist die Berücksichtigung von Portabilitäts-Standards oder Portabilitäts-Konventionen verbunden?

12. Metrik: Bestimmung justierter PFP-Werte

Generelle Justierung Ein justierter PFP-Wert beschreibt den aufwandsindizierenden Umfang eines PSSF-Projekts auf Basis der durch die domänenübergreifende und domänenspezifische PFP-Makroanalyse ergänzten Ergebnisse der PFP-Mikroanalyse. Diese prozentuale Anpassung des unjustierten PFP-Werts erfolgt auf Grundlage der numerischen Einflussgrade, die im Rahmen der generellen und domänenspezifischen PFP-Makroanalyse ermittelt wurden.

Der Justierungsfaktor J bestimmt den prozentualen Justierungsumfang der durch die PFP-Makroanalyse erhobenen, externen Einflussfaktoren auf den unjustierten PFP-Wert. Die Berechnung von J erfolgt, unter Ausschluss der rein qualitativen Einflussfaktoren, auf Grundlage der folgenden Formel:

Formel 4.9

$$J = \overbrace{\left(SE^G \frac{\frac{EF^G * 2}{100}}{(AF^G * AS_F^G)} + \frac{100 - EF^G}{100} \right)}^{\text{domänenübergreifende Einflüsse}} + \left(\overbrace{\left(SE^D \frac{\frac{EF^D * 2}{100}}{AF^D * AS_F^D} + \frac{100 - EF^D}{100} \right)}^{\text{domänenspezifische Einflüsse}} - 1 \right)$$

wobei die Variablen Folgendes bedeuten:

$EF^{G/D}$ Der Einfluss der Faktoren$^{G/D}$ beschreibt den prozentualen Maximaleinfluss der generellen (G) oder domänenspezifischen (D) Einflussgrößen auf den unjustierten PFP-Wert,[10]

[10] Zur beispielhaften Anpassung an das erweiterbare Grundgerüst der PFP-Makroanalyse, aber

$AF^{G/D}$ Die Anzahl der Faktoren$^{G/D}$ wird durch die Summe der generellen oder domänenspezifischen Einflussgrößen der PFP-Makroanalyse bestimmt.[11]

$AS_F^{G/D}$ Die Anzahl der Subkriterien$^{G/D}$ je Faktor$^{G/D}$ beschreibt die uniformierte Anzahl der Subkriterien, die einem einzelnen generellen oder domänenspezifischen Einflussfaktor zuzuordnen ist.[12]

$SE^{G/D}$ Die Summe der Einflussgrade stellt die summierte Anzahl von aufwandserhöhenden, generellen oder domänenspezifischen Subkriterien dar. (Ein $SE^{G/D}$ charakterisiert einen projektbezogenen Durchlauf der PFP-Makroanalyse, wobei dessen Obergrenze dem Produkt aus $AF^{G/D}$ und $AS_F^{G/D}$ entspricht.)

Durch Einsetzung der spezifischen Werte zu den obigen Variablen in die generische Formel 4.9 kann die für das Grundgerüst der PFP-Makroanalyse spezifische Gleichung zur Bestimmung von J abgeleitet werden. Formel 4.10 zeigt die Gleichung zur Bestimmung von J für das PFP-Grundmodell:

Formel 4.10

$$J = (SE^G * 0,02 + 0,8) + ((SE^D * 0,02 + 0,85) - 1)$$

Aufgrund von 21 verschiedenen SE^G-Ausprägungen und 16 möglichen SE^D-Bewertungen ergeben sich insgesamt 336 erzielbare Kombinationen für J. Die möglichen Werte für J liegen zwischen $0,65$ und $1,35$.

Zählmodellspezifische Justierung In Anlehnung an Zähltypen der *Function Point Analysis* (FPA), die Bewertungen sowohl für Projekte als auch für einzelne

auch aus Kompatibilitätsgründen zur IFPUG FPA, wird in [Kie06a, S. 145] die Variable EF^G auf 20 % gesetzt. Zur Sicherung der Vergleichbarkeit zwischen justierten PFP und justierten FP ist die Höhe des EF^G frei wählbar, insofern der Wert dieser Variable unter 35 % liegt.
Der EF^D wird durch die EF^G-Variable indirekt vorbestimmt. Da die Summe beider Einzelfaktoren 35 % nicht überschreiten darf, ist der EF^D auf 15 % festzusetzen. Das prozentuale Gewicht domänenübergreifender und -spezifischer Einflussfaktoren entspricht somit deren quantitativem Verhältnis untereinander.

[11] $AF^G = 4$: Das domänenübergreifende Referenzmodell der PFP-Makroanalyse ist durch vier externe Einflussfaktoren (Dokumentation, Infrastruktur, Transitionsprozess, Wissensvermittlung; siehe 9. Metrik) gekennzeichnet.
$AF^D = 3$: Das domänenspezifische Grundmodell der PFP-Makroanalyse ist durch drei externe Einflussfaktoren (Rechenleistung, Sicherheit, Speicherumfang; siehe 10. Metrik) gekennzeichnet.

[12] $AS_F^G = 5$: Die domänenübergreifenden Einflussfaktoren des Basismodells der PFP-Makroanalyse wurden in jeweils fünf Subkriterien unterteilt.
$AS_F^D = 5$: Die domänenspezifischen Einflussfaktoren des Referenzmodells der PFP-Makroanalyse wurden in je fünf Subkriterien untergliedert.

Softwareprodukte und damit Zählmodelle für Neuentwicklungsprojekte, Weiterentwicklungsprojekte und Anwendungssysteme unterstützt, kann mit den PFP-Metriken das zu ermittelnde Umfangsmaß ebenfalls Projekten oder Produkten zugeordnet werden.

Die Zähltypen im PFP-Zählmodell sind (vgl. [Kie06a, S. 60 ff.]):

- Neuentwicklungsprojekte
- Wiederverwendungsprojekte (Modifikation vs. Nutzung)
- Produktentwicklung auf der Grundlage einer PSSF (Instanziierung)

Neben der allgemeinen PFP-Größenjustierung kann somit eine zähltypabhängige Gewichtung der unjustierten PFP erfolgen:

- für Neuentwicklungsprojekte: Formel 4.11
- für Wiederverwendungsprojekte als PSSF-Modifikation: Formel 4.12
- für Wiederverwendungsprojekte als PSSF-Nutzung: Formel 4.13
- für die Instanz einer neu entwickelten PSSF: Formel 4.14
- für die Instanz einer modifizierten PSSF: Formel 4.15

Neuentwicklungsprojekte Formel 4.11 ermöglicht die Berechnung des Umfangsmaßes der justierten PFP für PSSF-Neuentwicklungsprojekte unter Berücksichtigung der obigen Ausprägungsformen:

Formel 4.11

$$\textit{justierte } PFP_{Neuentwicklung} = (NPFP + KPFP + MPFP) * J$$

Der justierte PFP-Wert für Neuentwicklungsprojekte ergibt sich aus der mit J multiplizierten Summe der unjustierten PFP für neu entwickelte, aus Legacy-Software konvertierte und nach der Instanziierung manuell angepasste Funktionalitäten.

Folgende Ausprägungsformen fallen unter diesen Zähltyp:

- vollständige Neuentwicklung einer prozessorientierten SSF;
- Neuentwicklung einer prozessorientierten SSF, die (zum Teil) durch eine Migration von bereits bestehenden Softwareprodukten realisiert wird.

Wiederverwendungsprojekte (PSSF-Modifikation) Formel 4.12 beschreibt die Kalkulation von justierten PFP für Wiederverwendungsprojekte, welche eine

PSSF modifizieren:

Formel 4.12

$$justierte\ PFP_{Modifikation} = ((NPFP + GPFPn + KPFP + MPFP) * Jn) + (EPFP * Jv)$$

Die innerhalb der PFP-Makroanalyse erfassten, externen Einflussfaktoren unterliegen in der Ausprägung ihrer Subkriterien zeitlichen Schwankungen. Dementsprechend sind die Entwicklungsbedingungen der ursprünglichen PSSF vom Einfluss der Kostenfaktoren im Rahmen eines späteren Weiterentwicklungsprojekts durch getrennte Justierungsfaktoren voneinander abzugrenzen.

Projekte dieses Zähltyps sind durch durch eine Abwandlung der PSSF gekennzeichnet und können beispielsweise wie folgt ausgeprägt sein:

- Hinzufügen, Löschen oder Modifizieren einzelner Funktionen der PSSF, um auf die Dynamik in den Anforderungen der Domäne zu reagieren.
- Migration von bisher separaten Softwareprodukten, welche durch einen hohen Anteil an Überschneidungen zu den Gemeinsamkeiten der PSSF gekennzeichnet sind.
- Ausschluss von Softwareprodukten aus der prozessorientierten SSF, um den Anteil der Gemeinsamkeiten zu erhöhen und einen höheren Grad an Flexibilität zu erreichen.
- Zusammenschluss von mehreren PSSF, infolge eines hohen Potenzials an wiederverwendbaren Bausteinen zwischen den separaten PSSF.
- Aufspaltung einer PSSF in mehrere PSSF, um Effizienzsteigerungen hinsichtlich einer größeren Flexibilität sowie eines höheren Anteils an Gemeinsamkeiten zu erzielen.

Wiederverwendungsprojekte (PSSF-Nutzung) Die Berechnung des justierten PFP-Werts für Wiederverwendungsprojekte, die durch eine ausschließliche Nutzung der bestehenden PSSF-Infrastruktur gekennzeichnet sind, wird durch Formel 4.13 ermöglicht:

Formel 4.13

$$justierte\ PFP_{Nutzung} = MPFP * J$$

Die Erfassung von Aufwänden generativer Konfigurationstätigkeiten zur automatisierten Instanziierung von Produkten aus einer existierenden PSSF wird durch punkteorientierte Softwaremetriken mit dem Ziel der Umfangsmessung funktionaler Bausteine und externer Einflussgrößen nicht unterstützt. Im Rahmen der

PFP-Analyse erfolgt dementsprechend ein Ausschluss dieser vernachlässigbaren Generierungsaspekte sowie die Erfassung und Justierung von nachgelagerten, manuellen Ergänzungen des instanziierten Softwareproduktes.

Zu diesem Zähltyp zählen Projekte, die eine bestehende PSSF zur automatisierten Generierung von Softwareprodukten verwenden. Die nachstehenden Szenarien beschreiben zwei exemplarische Projektausprägungen dieses PFP-Zählmodells:

- Erstellung von Instanzen der PSSF auf Basis der gemeinsamen Assets;
- Generierung von Softwareprodukten durch das Binden von Variabilitäten an die gemeinsamen Bausteine der PSSF.

Produkt (neu) Der Zähltyp des Produktes wird durch ein abgeschlossenes Neuentwicklungs- oder Wiederverwendungsprojekt initialisiert. Der justierte PFP-Wert eines Softwareprodukts, das aus einer neu entwickelten PSSF erstmalig generiert wurde, ist durch Formel 4.14 zu berechnen:

Formel 4.14

$$\mathit{justierte}\ PFP_{\substack{Produkt \\ (neu)}} = (NPFP + MPFP) * J$$

Produkt (modifiziert) Insofern eine bestehende PSSF durch ein Wiederverwendungsprojekt aktualisiert wird, errechnen sich die justierten PFP einzelner Instanzen gemäß Formel 4.15:

Formel 4.15

$$\mathit{just.}\ PFP_{\substack{Produkt \\ (mod.)}} = ((OPFP + NPFP + GPFPn + MPFP) - (GPFPv + EPFP)) * Jn$$

Verwendete Variablen Zur Kalkulation dieser zählmodellabhängig justierten PFP ist die aktuelle Ausprägung von J und eine zähltypmotivierte Untermenge der folgenden Variablen notwendig:

Entfernte PFP (EPFP)
Unjustierte PFP, die durch das Eliminieren von bisher bestehenden Funktionalitäten der PSSF entstehen.

Geänderte PFP nachher (GPFPn)
Unjustierte PFP, die Änderungen an bestehenden PSSF-Funktionalitäten umfassen und die Situation nach der Modifikation beschreiben.

Geänderte PFP vorher ($GPFPv$)
Unjustierte PFP, die durch Änderungen an existierenden PSSF-Funktionalitäten entstehen und den Zustand vor der Modifikation erfassen.

Justierungsfaktor nachher (Jn)
Justierungsfaktor nach Projektende

Justierungsfaktor vorher (Jv)
Justierungsfaktor vor Projektbeginn

Konvertierte PFP ($KPFP$)
Unjustierte PFP für Funktionen, die der PSSF durch eine Übernahme von Komponenten aus bestehenden Produkten hinzugefügt wurden.

Manuelle PFP ($MPFP$)
Unjustierte PFP für individuelle Funktionalitäten, die nach der Generierung einer PSSF-Instanz an das Softwareprodukt manuell angegliedert werden.

Neu PFP ($NPFP$)
Unjustierte PFP für die neu entwickelten Funktionen der PSSF.

Originäre PFP ($OPFP$)
Unjustierte PFP-Zählung der PSSF vor deren Modifikation.[13]

13. Metrik: Kalkulation qualitätsjustierter PFP-Werte

Ein qualitätsjustierter PFP-Wert ist ein um qualitative Aspekte erweitertes Umfangsmaß, welches die aufwandsindizierende Größe eines prozessorientierten SSF-Projekts beschreibt. Qualitätsjustierte PFP realisieren eine zusätzliche Gewichtung des bereits justierten PFP-Werts auf Basis der numerischen Einflussgrade, die innerhalb der optional durchführbaren, qualitätsorientierten PFP-Makroanalyse zu ermitteln sind.

Qualitätsjustierte PFP-Werte werden wie folgt bestimmt:

Formel 4.17
$$qualitaetsjustierte\ PFP = justierte\ PFP + QJ * unjustierte\ PFP$$

[13] Insofern der OPFP-Wert nicht vorhanden ist, kann er durch die folgende Gleichung bestimmt werden:

Formel 4.16
$$OPFP = \frac{(justierte\ PFP) - (Zaehlung\ der\ PSSF\ vor\ deren\ Modifikation)}{Jv}$$

wobei

qualitativer Justierungsfaktor (QJ)
Der qualitative Justierungsfaktor (QJ) ist die zentrale Größe zur Berücksichtigung der ISO / IEC 9126-konformen Qualitätsfaktoren und wird auf Basis von Formel 4.18 berechnet:

Formel 4.18

$$QJ = \frac{SQ}{6} * \frac{EQF}{100}$$

Einfluss der Qualitätsfaktoren (EQF)
Der EQF beschreibt den prozentualen Maximaleinfluss der qualitativen Einflussfaktoren auf den unjustierten PFP-Wert.[14]

Summe der qualitativen Einflussgrade (SQ)
Der SQ stellt die Summe der numerischen Einflussgrade von sämtlichen qualitativen Kostengrößen dar. Ein SQ kennzeichnet demnach eine projektbezogene Ausführung der qualitativen PFP-Makroanalyse, wobei der SQ-Wert im Grenzfall zwischen minus sechs und plus sechs variieren kann.

14. Metrik: Aufwandsprognosen

Das Vorgehen zur Kostenvorhersage im Rahmen der PFP-Analyse unterstützt die Kalkulation künftiger Entwicklungsaufwendungen auf Basis einer zeitbezogenen Messgröße, die als Grundlage für die Preisfindung dienen kann. Darüber hinaus bestimmen jedoch auch individuelle Kostensätze, Gewährleistungen, Exklusivitätsrechte und Alleinstellungsmerkmale den letztendlichen Preis der Entwicklung oder Modifikation einer PSSF.

Die oben aufgeführten PFP-Umfangsmaße beschreiben eine Vielzahl unterschiedlicher Kostentreiber und stellen die Größe eines Auftragsgegenstands unabhängig von dessen Realisierung dar.

Die Schätzgleichungen zur Berechnung künftiger Aufwendungen bei der Entwicklung oder Modifikation einer PSSF als Personenstunden (PS)[15] basieren auf

[14] Um die Gefahr einer qualitativen Übergewichtung einzuschränken, wird für das PFP-Qualitätsmodell ein EQF-Wert von 15 % vorgeschlagen. Somit ist der Maximaleinfluss der PFP-Makroanalyse auf den unjustierten PFP-Wert mit 50 % zu beziffern, da neben dem in Formel 4.18 dargestellten, 15-prozentigen Qualitätseinfluss zusätzliche 20 % durch generelle und weitere 15 % durch domänenspezifische Einflüsse determiniert werden.
Folglich wird der Wert für den qualitativen Justierungsfaktor durch $QJ = 0,025 * SQ$ bestimmt.

[15] Eine Personenstunde (PS) ist hier definiert als »*one hour of work by one person, including normal personal breaks, but excluding major breaks such as for lunch*«.

empirisch erhobenen Erfahrungswerten und Studien, siehe [Kie06a, Kap. 7].

Schätzgleichung auf Basis unjustierter PFP

Formel 4.19

$$Personenstunden = 2,4199 * unjustierte\ PFP$$

Obwohl dieses Prognosevorgehen eine frühzeitige und grobkörnige Kostenvorhersage gestattet, kann die Anwendung von Formel 4.19 aufgrund einer Nichterfassung von externen Einflussgrößen zu fehlerhaften Prognoseergebnissen führen und ist ausschließlich im Einzelfall vertretbar.

Schätzgleichung auf Basis justierter PFP

Formel 4.20

$$Personenstunden = 2,0987 * justierte\ PFP$$

Die Schätzgleichung basiert ebenso auf historischen Messwerten und ermöglicht die Prognose der notwendigen Personenstunden im Rahmen der Entwicklung oder Modifikation einer PSSF, deren Umfang in justierten PFP vorliegt. Auf Grundlage dieses Größenmaßes erfolgt die Berücksichtigung domänenübergreifender und automobilspezifischer Kostentreiber sowie eine Präzisionssteigerung der Aufwandsvorhersage.

Schätzgleichung auf Basis qualitätsjustierter PFP

Formel 4.21

$$Personenstunden = 2,0534 * qualitaetsjustierte\ PFP$$

Bedingt durch die Fokussierung des qualitätsjustierten Größenmaßes der PFP-Analyse erfolgt ein gewichtender Einbezug der definierten Qualitätsmetriken. Infolge dieser intensiven Justierung des zugrunde liegenden Umfangsmaßes bietet die Formel das Vorgehen mit dem höchsten Genauigkeitsgrad für eine Aufwandsvorhersage der Entwicklung von PSSF im Automotive-Sektor.

4.3.6 Anwendung im VEIA-Kontext

Folgende Anpassungen sind notwendig, um die PFP-Metriken im VEIA-Kontext anzuwenden:

- Die PFP-Metriken können sowohl auf Funktionsmodelle als auch Softwarearchitekturmodelle angewendet werden. Für den jeweiligen Anwendungsfall müssen jedoch die Komplexitätsmatrizen angepasst werden: Ab wann ist eine VEIA-Funktionsspezifikation komplex?
- Die Funktionshierarchie muss dazu entsprechend »flach geklopft« werden. Hier ist, wie bei Kiebusch auch, auszuwählen, welcher Abstraktionsgrad für die Bewertung am besten geeignet ist.
- Für die Justierung und Qualitätsjustierung müssen VEIA-spezifische Aufwandstreiber hinzugezogen werden, ggf. die von Kiebusch vorgeschlagenen durch andere ersetzt werden.
- Die Kennzahlen für die Aufwände (in Personenmonaten) für die Aufwandsschätzung sind ebenfalls anzupassen.
- Die unjustierten PFP-Metriken lassen sich auch als abstraktes Maß für die Größe von Softwarekomponenten verwenden, wenn sie auf die Spezifikation von »verbaubaren« Softwarekomponenten bzw. auf Architekturmodelle für Produkte (durch Konfiguration aus der Produktlinie abgeleitet) angewendet werden. Hierzu muss sowohl PA als auch IH entsprechend gewählt werden. Die Anzahl der Signale bzw. Ports sowie die Anzahl der zu implementierenden Komponenten und die daraus abgeleitete Komplexität sind auch ein Indiz für die zu erwartende Softwarekomponentengröße.

Die PFP-Metriken sind auf eine Fallstudie zum *Condition-Based Service* angewendet worden, die in [GKM07b] angegeben ist. Die Ergebnisse der Bewertung sind in Kapitel 5 beschrieben.

4.4 Metriken auf UML-Modelle am Beispiel »SDMetrics«

SDMetrics [Wüs07] ist ein CASE-Tool, das verschiedene Metriken auf UML-Modelle anwenden kann, um strukturelle Eigenschaften des mit UML festgelegten Entwurfs zu analysieren. Die Metriken stützen sich dabei auf Maße wie Entwurfsgröße (*design size*), Kopplungsgrad (*coupling*) und Komplexität (*complexity*) von UML-Modellierungselementen.

4.4.1 Metriken

In dem CASE-Tool wurden Metriken für die UML-Konstrukte (vgl. [Wüs06]): »Klasse« (*class*), »Schnittstelle« (*interface*), »Paket« (*package*), »Anwendungsfall« (*use case*), »Zustandsmaschine« (*statemachine*), »Aktivität« (*activity*), »Komponente« (*component*) und »Knoten« (*node*) implementiert.

Die Metriken sind in Kategorien unterteilt, die jeweils bestimmte strukturelle Eigenschaften repräsentieren. Folgende Kategorien sind definiert:

Größe Die Größe eines Modellierungskonstrukts wird häufig durch Zählen seiner internen Elemente bestimmt. Zum Beispiel: Anzahl der Operationen einer Klasse, Anzahl der Klassen in einem Paket etc. Ein ähnliches Maß ist auch bei den PFP-Metriken (Abschnitt 4.3) angewendet worden, um daraus die Komplexität abzuleiten.

Kopplung (Coupling) ist der Grad, zu dem Elemente miteinander verzahnt sind.

Eine hohe Kopplung hat einen Einfluss auf Systemqualitätseigenschaften wie Wartbarkeit (die Änderung eines Elements kann Änderungen mit ihm verbundener Elemente erzwingen) und Testbarkeit. Als Entwurfsprinzip gilt, die Kopplung zu minimieren.

Die meisten Kopplungsabhängigkeiten sind gerichtet, z. B. Client-Supplier-Beziehungen zwischen Elementen. Deshalb wird die Kopplung unterschieden in Import-Kopplung (*Use*-Beziehungen) und Export-Kopplung (*Used*-Beziehungen).

Vererbung Metriken bzgl. Vererbung betrachten Aspekte wie:

- Tiefe bzw. Breite der Vererbungsgraphen
- Anzahl der Vorfahren und Nachkommen eines Elements bzgl. der Vererbungshierarchie
- geerbte Größe
- Polymorphie, Methodenüberschreibung (*method overriding*) etc.

Komplexität ist der Grad der Verbundenheit (bzgl. Beziehungen oder Abhängigkeiten) zwischen Elementen.

Kohäsion (cohesion) ist der Grad der »logischen« Verbundenheit zwischen Elementen, zum Beispiel, inwieweit die Methoden einer Klasse dieselben Attribute der Klasse nutzen.

Auf Basis solcher Daten können mit Hilfe von Prognosemodellen Aussagen zu Kosten, Implementierungs- und Testaufwänden etc. geschätzt werden, vgl. hierzu [BW01a, BW01b, BW02].

4.4.2 Anwendung im VEIA-Kontext

Auch wenn die VEIA-Architekturmodelle nicht auf dem objektorientierten Paradigma beruhen, lassen sich doch einige der in diesem Werkzeug implementierten Prinzipien anwenden, zum Beispiel, wenn

- Analogien zwischen Funktionen mit UML-Klassen oder UML-Komponenten gezogen werden,
- Analogien zwischen Funktionsvarianzpunkten und ihren Funktionsalternativen mit der Klassenvererbungshierarchie gezogen werden,
- Analogien zwischen den Kommunikationsbeziehungen der Funktionen mit den Assoziationen zwischen Klassen oder den Konnektoren zwischen UML-Komponenten gezogen werden.

So könnten dann die bereits zur Verfügung stehenden Metriken für VEIA-Produktlinienmodelle nutzbar gemacht werden, zum Beispiel:

- Metriken zur Vererbung für die Bestimmung des Varianzgrads einer Funktion (bzw. eines Produktlinienmodells): Je mehr Funktionsvarianzpunkte und zugehörige Funktionsalternativen dazu spezifiziert sind (≙ Tiefe bzw. Breite des Vererbungsgraphen), desto mehr unterscheiden sich Funktionsausprägungen (bzw. Produkte der Produktlinie) voneinander.

  Diese Informationen können dann genutzt werden, um geeignete Varianzauflösungszeitpunkte bzw. Implementierungstechnologien diesbezüglich zu bewerten. Dazu notwendige zusätzliche Informationen: Häufigkeit der genutzten Varianten (»Implementierungshäufigkeit« bei Kiebusch) und Anforderungen an die Produktflexibilität.

- Komplexität und Kopplungsgrad können dazu genutzt werden, Abhängigkeiten zwischen Funktionsvarianzpunkten zu bewerten: Je höher die Kopplung zwischen Funktionsvarianzpunkten, desto wahrscheinlicher können Varianten nicht frei ausgewählt werden.

Die hier vorgestellte Arbeit kann somit als Ideengeber verstanden werden, Metriken und Maße auf Modelle anzuwenden.

Darüberhinaus lassen sich die Metriken auf die Implementierungsmodelle von CBS, die mit dem Werkzeug *Rhapsody* (i-Logix/Telelogic) erstellt werden, anwenden.

4.5 Bewertung von Varianz in Architekturen

In [Sch03] wird die Frage behandelt, ob sämtliche Varianz einer Produktlinie bereits implementiert werden soll, auch wenn derzeitig zu entwickelnde Produkte diese Varianz noch nicht benötigen, oder ob dies erst geschehen soll, wenn sie benötigt wird.

Es gibt prinzipiell drei Ansätze, Produktlinien (neu) zu entwickeln bzw. ihre Evolution zu planen (vgl. [BKPS04, S. 179 ff.]):

Reaktive Evolution
Varianz, die nicht unmittelbar benötigt wird, wird nicht berücksichtigt.

Leichtgewichtige, architekturzentrierte Evolution
Varianz wird in der Produktlinienarchitektur bereits berücksichtigt, aber noch nicht implementiert.

Proaktive Evolution
Die im Produktlinien-Scoping identifizierte Varianz wird in der Produktlinienarchitektur und in der Entwicklung berücksichtigt und auch implementiert.

Der reaktive und proaktive Ansatz sind die beiden Extrempunkte eines Vorgehens, während der leichtgewichtige Ansatz den Bereich dazwischen repräsentiert.

Um die verschiedenen Ansätze bewerten zu können, wurde ein Kostenmodell aufbauend auf folgenden Begriffen entwickelt:

Varianzpunkt (variation point, vp)
Ein Varianzpunkt korrespondiert mit einer konsistenten Änderung an einem Softwareprodukt aufgrund der Variabilität. Der Varianzpunkt wird dabei als unabhängig von der Anzahl der Stellen in der Implementierung gesehehen, die von der Änderung betroffen sind.

Varianzauswirkungspunkt (variation impact point, vip)
Ein Varianzauswirkungspunkt bezeichnet eine Stelle in der Implementierung (unabhängig vom Auflösungszeitpunkt der Varianz (*binding time*)), die von der Varianz betroffen ist.

Das heißt, es gibt eine 1:n-Beziehung zwischen Varianzpunkt und seinen Varianzauswirkungspunkten.

Befähigung (capability)
Als Basis für das Kostenmodell wird zudem zwischen der Implementierung einer Befähigung bzw. eines Potenzials (cap) und dem Varianzmechanismus (vm) unterschieden. Die Befähigung kann als Einzelimplementierung betrachtet

werden, unabhängig von der Anzahl der VIPs, die für die Integration in das System relevant sind.

Varianzmechanismus (variability mechanism, vm)
Für jeden VIP ist eine VM-Implementierung notwendig.

Folgende Annahmen zu den obigen Begriffen sind getroffen:

- Wird ein Varianzpunkt bereits in der Entwurfsphase der Produktlinienarchitektur berücksichtigt, so verringert sich die Anzahl von Varianzauswirkungspunkten gegenüber einer späteren Einplanung und Integration der Varianz.
- Der Aufwand, die Befähigung für eine Varianz zu implementieren, ist unabhängig von der Softwarearchitektur. Dagegen ist der Aufwand für eine VM-Implementierung direkt proportional zur Anzahl der VIPs.

4.5.1 Parameter für das Kostenmodell

Für das Kostenmodell werden zusätzlich folgende Parameter berücksichtigt:

Anzahl der betrachteten Jahre: y

Anzahl der entwickelten Systeme in diesem Zeitraum: N

Anzahl der entwickelten Systeme bis zu einem bestimmten Zeitpunkt: n

Abschlagsgrad (discount rate): r
Der Abschlagsgrad beschreibt, bis zu welchem Grad Geld über die Zeit abgeschrieben wird, vgl. *discounted cash-flow analysis, [7]*.

Aufwand, um eine Befähigung (cap) zu implementieren: EC
(Dieser Wert wird nur einmal pro Produktlinie benötigt.)

Anzahl der Varianzauswirkungspunkte für die Varianz in der gesamten Infrastruktur: vip

Aufwand für Entwurf und Implementierung eines Varianzmechanismus: $Evm = Evmd + vip * Evmi$
(Annahme: Während die Entwurfskosten als unabhängig von der Anzahl der Umsetzungen eines Varianzmechanismus betrachtet werden kann, gilt dieses nicht für die Implementierungskosten.)

Wahrscheinlichkeit, dass eine bestimmte Varianz zwar nicht im ersten System, aber in einem späteren System aus der Produktlinie benötigt wird: p

Komplexitätskosten (complexity cost): cc
Der Faktor beschreibt die Kosten, die durch Hinzufügen der Varianz in die Architektur verursacht werden. Annahme: Die Wartung von komplexeren Code wird üblicherweise zu höheren Aufwänden. Die Komplexität wird erhöht, wenn viel Varianz früh hinzugefügt wird.

Entwicklungskosten eines Systems aufgrund der Komplexität: $Scc = cc * vip$

Wenn eine Varianz später implementiert wird (im reaktiven Ansatz), so werden zwar initial Kosten gespart, jedoch kommen später verschiedene Kostenarten wieder hinzu:

Kosten-Overhead co
Später durchgeführte Änderungen verursachen typischerweise einen höheren Aufwand (aufgrund von Entwurfsänderungen und Codelöschungen etc.)

co sind die zusätzlichen Kosten für die Implementierung der Varianzmechanismen als Teil der initalen Entwicklung.

Cost retrofit cr
Wird Varianz erst zu einem späteren Zeitpunkt hinzugefügt, so erfordert dies spezifische Änderungen an bereits entwickelten Systeme (z. B. ihre Anpassungen an die existierende Infrastruktur). Diese zusätzlichen Kosten sind pro VIP und pro bereits existierendem System definiert.

Annahme: Mit den oben aufgelisteten Parametern ist es möglich, die Kostenauswirkungen der verschiedenen Produktlinieninitiierungsansätze zu vergleichen.

4.5.2 Kostenmodell für Varianz

Kosten für die Einführung der Varianz in die Produktlinie *EVI* in Abhängigkeit der Einführungszeit und der Anzahl der VIPs

Hierfür gibt es zwei wesentliche Fälle:

1 am Anfang der Produktlinienentwicklung (d. h. bei $n = 0$):

Formel 4.22 $$(n = 0) \bullet EVI(n) = EC + Evm$$

2 zu einem späteren Zeitpunkt der Produktlinienentwicklung:

Formel 4.23 $$(n > 0) \bullet EVI(n) = EC + Evm * co + cr * n * vip$$

Berücksichtigung des »Discounted Cash Flow-Ansatzes« $EVI'(n)$, da die spätere Einführung der Varianz in der zweiten Formel auch ein Prozess über die Zeit ist (mit der Annahme, dass die Systeme gleichmäßig über die Zeit verteilt sind):

Formel 4.24 $$EVI'(n) = EVI(n) * (1 - r)^{(\frac{n}{N} * y)}$$

Vgl. die Analyse in [Sch03, S. 7; Abb. 1].

Anfallende Kosten für *n* Systeme bei einer frühen Varianzberücksichtigung

In den obigen Formeln sind die Kosten für eine spätere Varianzeinführung berücksichtigt, jedoch nicht die Einsparungen, da die Kosten für den Umgang mit der komplexeren Infrastruktur bei der frühen Varianzberücksichtigung noch nicht eingeflossen sind.

Formel 4.25 $$Scc'(n) = Scc * (1 - r)^{(\frac{n}{N} * y)}$$

Kosten für *alle* Systeme nach Berücksichtigung der Varianz:

Formel 4.26 $$Tcc(n) = \sum_{i=n}^{N} Scc'(i)$$

Wird die Varianz mit dem n-ten System berücksichtigt, dann sind die Gesamtkosten dafür wie folgt:

Formel 4.27 $$CV(n) = EVI'(n) + Tcc(n)$$

Für realistische Werte für cc ist der Korrekturwert von $Tcc(n)$ sehr gering, bspw. $0,05$.

Vgl. Analyse [Sch03, S. 9; Abb. 2].

Berücksichtigung der Wahrscheinlichkeit, dass variierende Funktionalität (die Befähigung) benötigt wird

Bisher wurde betrachtet, was passiert, wenn Varianz zu einem bestimmten Zeitpunkt berücksichtigt wird. Nun die Frage, wann dies schließlich passiert.

Ausgehend von der Annahme, dass Varianz erst dann berücksichtigt wird, wenn sie tatsächlich benötigt wird (und nicht vorher), kann der Fokus gesetzt werden auf die Frage: Wann wird die Varianz benötigt? Dies wird als stochastischer Prozess modelliert: Für jedes neue Produkt in der Produktlinie gibt es eine bestimmte Wahrscheinlichkeit p, dass dieses Produkt die zusätzliche Befähigung benötigt. (Beachte: unterschiedliche Funktionalitäten in der Produktlinie haben unterschiedliche Wahrscheinlichkeiten.)

Die Gesamtvarianzkosten $E(CV)$ (*total variability costs E(CV)*:

Formel 4.28

$$E(CV) = \sum_{i=1}^{N} CV(i) * (1-p)^{i-1} * p$$

4.5.3 Resümee der Autoren

Als generelle Richtlinien konnten identifiziert werden [Sch03, S. 15]:

- Späte Implementierung von Varianz macht sich (in einer großen Produktlinie) nur bezahlt, wenn die Wahrscheinlichkeit, dass die Varianz wirklich benötigt wird, sehr gering ist.
- Eine Implementierung von vornherein scheint der kostengünstigste Ansatz zu sein, wenn die Varianzwahrscheinlichkeit nicht zu gering ist.
- Die Entwicklung einer Produktlinienarchitektur, die die Varianz modularisiert, scheint eine Schlüsselrolle zu spielen, um die gesamten Entwicklungskosten geringer zu halten.

4.5.4 Anwendung im VEIA-Kontext

Das oben vorgestellte Berechnungsframework für die Bewertung des Einflusses von Varianz könnte unter Umständen in drei Szenarios Anwendung finden:

1. Bzgl. der in der VEIA-Produktlinienbeschreibung definierten Varianz (die Varianzpunkte in [Sch03]) könnte bewertet werden, welche Auswirkungen (Aufwände) es bedeutet, ein bestehendes Funktionsnetz (logische Architektur) zu ändern. Die im Funktionsnetz modellierte Varianz (Funktionsvarianzpunkte, optionale Funktionen etc.) würden in diesem Fall als Befähigung (*capability*)

bzw. Varianzauswirkungspunkte (VIPs) aus [Sch03] interpretiert werden. Mit hochgerechneten Erfahrungswerten bzgl. der Realisierung von Funktionsnetzen als Software könnten die Metriken justiert und zur Aufwandsschätzung herangezogen werden.

2 Bzgl. des Zusammenspiels von Funktionsnetz und Softwarearchitektur gibt es eine ähnliche Beziehung: hier könnten die Varianz im Funktionsnetz als *VPs* und Varianz in der Softwarearchitektur als *Caps* bzw. *VIPs* gemäß [Sch03] interpretiert werden.

3 Das Szenario der Bewertung des Zusammenhangs von Produktlinienbeschreibung und Softwarearchitektur mit Implementierung entspricht dem originären Fokus des referenzierten Papiers.

Bzgl. der Fallstudie *CBS* könnten somit unterschiedliche Realisierungsszenarien (produktspezifische Softwarekomponenten mittels Softwareproduktlinienansatz vs. produktgenerische Softwarekomponenten mittels 150 %-Ansatz) unter Umständen verglichen werden.

4.6 Servicenutzungsgrade und Kohärenz in Produktlinienarchitekturen

Metriken für Softwareproduktlinienarchitekturen, die Aussagen zu so genannten »Servicenutzungsgrade« (*service utilization*) und Kohärenz treffen, werden in den Arbeiten [DMv01, vDM03, Mis06] vorgestellt.

4.6.1 Servicenutzungsgrad

Metriken zu Servicenutzungsgrade werden in [DMv01, vDM03] zur Bewertung der strukturellen Qualität von Softwareproduktlinienarchitekturen vorgeschlagen.

Unter dem Begriff *Service* werden dabei öffentlichen Methoden, Funktionen oder direkt zugreifbare Datenstrukturen von Komponenten verstanden. Der Servicenutzungsgrad ist dabei ein Maß, das ausdrückt, wie viele der (angebotenen bzw. geforderten) Services (Dienste) einer Komponente auch wirklich in der bewerteten Architektur verwendet bzw. zur Verfügung gestellt werden.

Der Begriff *Dienst* (*service*) ist in den Arbeiten generisch gehalten, er soll auf verschiedene Architekturbeschreibungssprachen (ADLs) anwendbar sein. Demzufolge

kann untersucht werden, inwieweit die vorgeschlagenen Metriken auch auf die VEIA-Modelle [GEKM07] anwendbar sind.

Für Einzelkomponenten

Für eine Einzelkomponente lässt sich der Nutzungsgrad der von ihr angebotenen Dienste (*PSU*) und der von ihr geforderten Dienste (*RSU*) bestimmen. Die weiteren Metriken bauen auf diese beiden Begriffe auf.

Provided Service Utilization (PSU):

Formel 4.29

$$PSU(c) = \frac{P_{actual}(c)}{P_{total}(c)}$$

Required Service Utilization (RSU):

Formel 4.30

$$RSU(c) = \frac{R_{actual}(c)}{R_{total}(c)}$$

Dabei gilt:

P_{actual} ist die Anzahl der von der Komponente *X* angebotenen Dienste, die auch tatsächlich von anderen Komponenten verwendet werden.

P_{total} ist die Anzahl aller von der Komponente *X* angebotenen Dienste.

R_{actual} ist die Anzahl der von der Komponente *X* benötigten Dienste, die auch tatsächlich von anderen Komponenten angeboten werden.

R_{total} ist die Anzahl aller von der Komponente *X* benötigten Diente.

PSU und *RSU* sind kontextabhängig in dem Sinne, dass die konkrete Architektur, in der die bewertete Komponente verwendet wird, bestimmt, welchen Wert die beiden Maße erhalten.

Für Architekturen

Mit den Metriken *CPSU* und *CRSU* wird die interne Kohäsion einer Architektur ermittelt, sozusagen der Grad der Servicenutzung bzw. -bereitstellung für eine Architektur:

Compound PSU (CPSU):

Formel 4.31

$$CPSU = \frac{\sum_{c \in C} P_{actual}(c)}{\sum_{c \in C} P_{total}(c)}$$

Compound RSU (CRSU):

Formel 4.32

$$CRSU = \frac{\sum_{c \in C} R_{actual}(c)}{\sum_{c \in C} R_{total}(c)}$$

Dabei gilt:

P_{actual}, P_{total}, R_{actual}, R_{total} wie vorher.

C ist die Menge der Komponenten in einer Architektur.

Berücksichtigung von Varianz

Um Varianz (XOR-Varianz und Optionalität) in Produktlinienarchitekturen (*PLA*s) zu berücksichtigen, wird die Spanne (Variationsbreite) für *PSU*s und *RSU*s von Komponenten (Menge der *PSU*s bzw. *RSU*s über die Produkte einer Produktlinie) ermittelt.

Die Spanne (Variationsbreite) für die *PSU* einer Komponente *c*, die für alle invarianten Komponenten zu ermitteln ist, wird über die einzelnen Produktarchitekturen *p* berechnet:[16]

Formel 4.33

$$Span_{PSU}(c) = \biguplus_{p \in Products} \frac{P_{actual}(c_p)}{P_{total}(c_p)}$$

[16] Das Symbol $\biguplus$ steht für die Vereinigung in Multimengen, d. h. Mengen, in denen Elemente mehrfach vorkommen können.

Die Spanne (Variationsbreite) für die *RSU* einer Komponente *c*, zu ermitteln für alle invarianten Komponenten, berechnet über die einzelnen Produktarchitekturen *p*, ist:

Formel 4.34

$$Span_{RSU}(c) = \biguplus_{p \in Products} \frac{R_{actual}(c_p)}{R_{total}(c_p)}$$

Die Spanne (Variationsbreite) für *CPSU* einer Produktlinienarchitektur (PLA), berechnet über die einzelnen Produktarchitekturen *p*, ist:

Formel 4.35

$$Span_{CPSU} = \biguplus_{p \in Products} \frac{\sum_{c_p \in C_p} P_{actual}(c_p)}{\sum_{c_p \in C_p} P_{total}(c_p)}$$

Die Spanne (Variationsbreite) für *CRSU* einer PLA, berechnet über die einzelnen Produktarchitekturen *p*, ist:

Formel 4.36

$$Span_{CRSU} = \biguplus_{p \in Products} \frac{\sum_{c_p \in C_p} R_{actual}(c_p)}{\sum_{c_p \in C_p} R_{total}(c_p)}$$

Durchschnitts-PSU bzw. -RSU einer hierarchischen Komponente

Neben der Spanne wird auch der Durchschnittswert berücksichtigt.

Der Durchschnitt der *PSU*s einer Komponente *c* in einer PLA ist:

Formel 4.37

$$Average_{PSU}(c) = \frac{\sum_{p \in Products} \frac{P_{actual}(c_p)}{P_{total}(c_p)}}{\#Products}$$

Der Durchschnitt der *RSU*s einer Komponente *c* in einer PLA ist:

Formel 4.38

$$Average_{RSU}(c) = \frac{\sum_{p \in Products} \frac{R_{actual}(c_p)}{R_{total}(c_p)}}{\#Products}$$

Durchschnitts-PSU bzw. -RSU einer PLA

Der Durchschnitt der *CPSU*s einer PLA ist:

Formel 4.39
$$Average_{CPSU} = \frac{\sum\limits_{p \in Products} \frac{\sum\limits_{c_p \in C_p} P_{actual}(c_p)}{\sum\limits_{c_p \in C_p} P_{total}(c_p)}}{\#Products}$$

Der Durchschnitt der *CRSU*s einer PLA ist:

Formel 4.40
$$Average_{CRSU} = \frac{\sum\limits_{p \in Products} \frac{\sum\limits_{c_p \in C_p} R_{actual}(c_p)}{\sum\limits_{c_p \in C_p} R_{total}(c_p)}}{\#Products}$$

4.6.2 Kohärenz

Die Kohärenz einer Komponente c, $\psi(c)$, wird wie folgt definiert [Mis06]: Anzahl der angebotenen Dienste, die von Clients verwendet werden, gemittelt über alle angeschlossenen Clients der Komponente.

Formel 4.41
$$\psi(c) = \frac{\sum\limits_{w \in W(c)} (\#S(c,w) - 1)}{\#W(c) * (\#P(c) - 1)}$$

wobei

C ... Menge aller Komponenten in einer PLA

$c \in C$ ... Komponente in der PLA

$P(c) = \{s_1, s_2, \ldots\}$ ... Menge der von einer Komponente angebotenen Dienste

$W(c) = \{w_1, w_2, \ldots\}$ ... Menge der Komponenten $w_i \in C$, die Dienste der Komponente c nutzen bzw. erfordern, d. h., $W(c)$ ist die Menge der Clients von c.

$S(c, w)$ ... Menge der Dienste, die von c angeboten und von w tatsächlich genutzt werden.

4.6.3 Anwendung im VEIA-Kontext

Die in [DMv01, vDM03, Mis06] definierten Metriken müssen für VEIA-Artefakte entsprechend angepasst werden und zwar bezüglich:

- des Umgangs mit hierarchischen Komponenten,
- des Service-Begriffs,
- des Umgangs mit Varianz und
- des Kohärenz-Begriffs.

Umgang mit hierarchischen Komponenten

In [Mis06] wird auf flachen Architekturen gerechnet. Hierarchisch zusammengesetzte Komponenten werden in [DMv01, vDM03] nur zum Teil berücksichtigt: In [DMv01] werden zusammengesetzte Komponenten zwar diskutiert, jedoch fehlt in der Definition der Metriken in [DMv01, vDM03] zu den verschiedenen Servicenutzungsgraden jeglicher Bezug zu Kompositionsbeziehungen zwischen Komponenten. Jede hierarchisch zusammengesetzte Komponente könnte jedoch als Architektur betrachtet und als solche bewertet werden. Für die Bewertung der Gesamtarchitektur müssten dann gemäß der Hierarchie die jeweiligen Werte entsprechend zusammengerechnet werden.

Service-Begriff

Die Metriken basieren auf dem Konzept, dass Komponenten *Services* anbieten und benötigen. In VEIA-Funktionsnetzen werden stattdessen Signalflüsse zwischen den Funktionen betrachtet. In AUTOSAR-Softwarearchitekturen [AUT06] wird zwischen Sender-Receiver-Interfaces und Client-Server-Interfaces unterschieden.

Bzgl. Client-Server-Interfaces kann der Service-Begriff 1:1 übernommen werden.

Bzgl. der Abbildung des Service-Begriffs auf Sender-Receiver-Interfaces wird in [vDM03] folgendes vorgeschlagen: »*Even for an event-based ADL, a mapping can be conceived in which provided services are the events to which a component will react and required services are the events emitted by a component.*« Unseres Erachtens muss es eigentlich genau umgekehrt definiert werden: Eingangssignale entsprechen den geforderten Services, weil eine Komponente auf die Eingangssignale im Allgemeinen reagieren *muss*. Ausgangssignale entsprechen den angebotenen Services; andere Komponenten *können* auf diese Ausgangssignale reagieren.

Als VEIA-spezifische Anpassung könnte der *Signalnutzungsgrad* pro Port berechnet werden, ggf. können durch Mittelwertberechnung diese Werte für eine Komponente hoch gerechnet werden. So würden sich Aussagen treffen lassen, inwieweit die Varianzdefinition in Produktlinien in einigen Produkten zu »überflüssigen« Signalen führen würde.[17]

Umgang mit Varianz

Obwohl in [Mis06] Produktlinienarchitekturen fokussiert werden, wird keine Hilfestellung gegeben, wie mit Varianz in Architekturbeschreibungen umzugehen ist. In [DMv01, vDM03] werden optionale Komponenten und Varianzpunktkomponenten berücksichtigt. Dagegen finden optionale Ports, Signale bzw. Konnektoren und variierende Ports keine Berücksichtigung.

Zu überlegen ist, inwieweit ein Nutzungsgrad zur Bewertung herangezogen werden kann, ob und wie gut eine Produktlinie aufgebaut ist bzw. inwieweit die Einzelprodukte voneinander abweichen.

Kohärenz-Begriff

Ob der Kohärenz-Begriff für port- bzw. signal-basierte Architekturbeschreibungen nutzbringend anwendbar ist, muss noch untersucht werden: Es ist davon auszugehen, dass in der Regel nicht alle Ports einer Komponente von den daran angeschlossenen Komponenten genutzt werden, so dass die Kohärenz dieser Komponente im Allgemeinen nicht gegen 1 konvertiert. Siehe hierzu die bereits geführte Diskussion zur Idee des Signalnutzungsgrads im Abschnitt zum Service-Begriff.

[17] Services stellen im Allgemeinen die Interaktionspunkte in serviceorientierten und komponentenbasierten Architekturen dar. In VEIA-Funktionsnetzen, beispielsweise, werden Ports einer Funktion häufig derart modelliert, dass eine spezifische Kommunikation mit anderen Funktionen ermöglicht wird. Daraus lassen sich Aussagen über Datenflüsse ableiten (z. B. Signalflussanalysen). Eine Funktion nutzt in der Regel nicht alle Ports einer anderen Funktion, sondern nur bestimmte. Demzufolge ist die Analogie von *Service:Komponente* in VEIA-Architekturen, zumindest in Funktionsnetzen, eher: *Signal:Port*. Damit ließe sich dann beispielsweise bestimmen, inwieweit überflüssige bzw. sehr wenig verwendete Signale spezifiziert wurden.

4.7 Resümee

Es stehen bereits einige Metriken zur Verfügung, die auf Produktlinien angewendet werden können. Zum Teil arbeiten sie jedoch auf anderen Modellen als im VEIA-Referenzprozess vorgeschlagen. Vor allem der Umgang mit und die Darstellung von Varianz in Produktlinienmodellen ist unterschiedlich.

Um die Metriken auf die Artefakte des VEIA-Referenzprozesses anwenden zu können, ist eine Anpassung der Metriken notwendig.

Außerdem muss der Umgang mit den Begriffen »Produkt« und »Produktlinien« an das Anwendungsgebiet von VEIA – Automotive-E/E-Systeme in einem verteilten Entwicklungs- und Integrationsprozess – angepasst werden. Hier erscheinen die vorliegenden Arbeiten zu generisch.

Bzgl. der PFP-Metriken (Abschnitt 4.3) sind bereits notwendige Anpassungen identifiziert, die direkt oder indirekt einen Einfluss auf die tatsächlich anfallenden Aufwände haben. Hierzu gehören die Kennzahlen für die Komplexitätsgewichtung – die umgekehrt auch Anforderungen definieren, bis zu welcher Detailtiefe modelliert werden muss –, die Benennung und Bewertung der spezifischen Aufwandstreiber sowie die Kennzahlen zur Abschätzung des Aufwands aufgrund von Umfangsmaßen.

Die PFP-Metriken wurden beispielhaft anhand von Funktions- und Softwarearchitekturmodellen ausprobiert, die im Rahmen der VEIA-Fallstudie CBS erstellt wurden. Diese geben einen ersten Einblick zur Interpretation der aus der Anwendung der Metriken erhaltenen Ergebnisse. Jedoch müssen im weiteren Verlauf des Projekts noch mehr Erfahrungen hierzu gesammelt werden. Die Beispielanwendung der PFP-Metriken auf CBS-Modelle ist in Kapitel 5 angegeben.

5 Anwendung der Metriken auf CBS

Im Folgenden werden die Ergebnisse der beispielhaften Anwendung der PFP-Metriken (vgl. Abschnitt 4.3) auf die im Rahmen der VEIA-Fallstudie erstellten Modellierungen der logischen Architektur und der verschiedenen Softwarearchitekturen dargestellt.

Die Fallstudie ist in [GKM07b] angegeben. Auf Basis der logischen Architektur, dessen Bewertung in Abschnitt 5.2 angegeben ist, wurden zwei konkurrierende Softwarearchitekturen erstellt, die ebenfalls beispielhaft mit den PFP-Metriken bewertet (Abschnitt 5.3.1 und Abschnitt 5.3.2) und dann miteinander verglichen wurden (Abschnitt 5.3.3). Ziel war, Aussagen dazu treffen zu können, welche der Softwarearchitekturen *besser* ist.

5.1 Produkte

Im Folgenden werden drei Produkte als Beispiel für die Metrik genutzt. Theoretisch gäbe es noch mehr Produkte, die aber vorerst nicht geplant sind. Hier könnten die Metriken zu justierten PFPs für die verschiedenen Zählmodelle Anwendung finden (z. B. die Metrik für das Szenario »Modifizierung einer PSSF«).

Die drei im Rahmen der Fallstudie betrachteten Produkte sind durch folgende Merkmale ausgezeichnet:

P1: Ottomotor, adaptive Zündkerzenverschleißerfassung, keine erweiterte Anzeige

P2: Ottomotor, lineare Zündkerzenverschleißerfassung, keine erweiterte Anzeige

P3: Dieselmotor, adaptive Partikelfilterverschleißerfassung, keine erweiterte Anzeige

Insgesamt gehen wir also von einer prinzipiellen Produkt(typ)anzahl von $PA = 3$ aus. Die Implementierungshäufigkeit setzen wir im Folgenden in Relation zur Produktanzahl. Für die Implementierungshäufigkeit (IH) der einzelnen Merkmale gehen wir beispielhaft von einer Gleichverteilung von Fahrzeugen mit Otto- und mit Dieselmotoren sowie von einer Gleichverteilung von Fahrzeugen mit adaptiver und mit linearer Zündkerzenverschleißerfassung aus. Unter Berücksichtigung der

Tatsache, dass Zündkerzen nur bei Ottomotoren verbaut werden, kommen wir damit auf folgende Werte:

- Implementierungshäufigkeit für Ottomotoren: $IH_{Ottomotor} = 1/2 * PA$
- Implementierungshäufigkeit für Dieselmotoren: $IH_{Dieselmotor} = 1/2 * PA$
- Implementierungshäufigkeit für Funktionen bzw. Komponenten zur adaptiven Zündkerzenverschleißerfassung: $IH_{ZuendkerzenVE\,adaptiv} = 1/4 * PA$.
- Implementierungshäufigkeit für Funktionen bzw. Komponenten zur linearen Zündkerzenverschleißerfassung: $IH_{ZuendkerzenVE\,linear} = 1/4 * PA$.

In der Modellierung der logischen Architektur wurde die Funktionalität der Bremsbelagverschleißerfassung durch Wiederverwendung realisiert: die Bremsbelagverschleißerfassungsfunktion wird sowohl für die vorderen als auch die hinteren Bremsbeläge verwendet, d. h. in der logischen Architektur zweimal instanziiert. Damit ergibt sich, dass die zugehörigen Funktionen doppelt so häufig wie die Produktanzahl verwendet werden. In den Softwarearchitekturen des Fallbeispiels wurde solch eine Mehrfachverwendung von Komponenten derzeit noch nicht berücksichtigt.

5.2 Anwendung von Metriken auf das Modell der logischen Architektur von CBS

Die Anwendung der Metriken in diesem Abschnitt beziehen sich auf die Modellierung der logischen Architektur in [GKM07b, Kap. 4].

5.2.1 Kiebuschs Unjustierte PFP

Alle Funktion wurden als Echtzeitfunktionen mit weichen Echtzeitanforderungen (EGWs bzw. EVWs) bewertet. Optionale Funktionen und Funktionsalternativen wurden als horizontale EVWs betrachtet. Wenn die Funktionen optionale, abhängige oder variierende Ports haben, dann wurden sie als vertikale EGWs oder EVWs eingestuft. Zur Komplexitätsgewichtung wurden die Signale der Funktionsports gemäß der Portdeklarationen gezählt.

Zur Bewertung wurden alle atomaren Funktionen des Funktionsnetzes herangezogen, d. h. alle Blattknoten der Funktionshierarchie. Demzufolge mussten die in der Hierarchie enthaltenen strukturellen Varianzinformationen (XOR-Varianz und

Optionalität bei hierarchisch zusammengesetzten Funktionen) auf die atomaren Funktionen entsprechend bezogen werden. Die in Tabelle 11 aufgelisteten atomaren Funktionen sind entsprechend ihrer hierarchischen Strukturierung sortiert.

Das Modellierungskonzept der Mehrfachverwendung von Funktionen (durch so genannte »Function-Hooks«) ist im CBS-Funktionsnetz einmal angewendet worden: die Spezifikation der Verschleißbewertung für die vorderen und hinteren Bremsbeläge wurde durch die zweimalige Verwendung der wiederverwendbaren Funktion `CBSBremsbelag` umgesetzt. Dementsprechend positiv wirkt sich der ermittelte Umfang für diese wiederverwendbare Funktion bzgl. der Produktlinie aus.

Die gemäß der 3., 4. und 5. Metrik ermittelten Maße zur Bestimmung der unjustierten Prozessfamilienpunkte (UPFP) als abstraktes Umfangsmaß für die atomaren Funktionen des CBS-Funktionsnetzes sind in Tabelle 11 abgebildet.

Tabelle 11 UPFP für CBS-Funktionsnetz.

Funktion	*#In*	*#Out*	*Kompl.*	UF_{gem}	UF_{var}	*IH* (*#Hooks*)	*V*	*UPFP*	$UPFP_{PA=3}$
`Bremsbelag Verschleiß`	7	3	h	23	–	1 PA (2)	–	11,5 / PA	3,83
`Bremsbelag Verfügbarkeit`	9	4	h	23	–	1 PA (2)	–	11,5 / PA	3,83
`Bremsbelag EinServiceIntervall`	9	4	h	23	–	1 PA (2)	–	11,5 / PA	3,83
`Bremsbelag Lebenszyklus`	9	5	h	23	–	1 PA (2)	–	11,5 / PA	3,83
`Motoröl Verschleiß`	3	1	m	15	–	1 PA	–	15 / PA	5
`Motoröl Verfügbarkeit`	9	4	h	23	–	1 PA	–	23 / PA	7,67
`Motoröl EinServiceIntervall`	9	4	h	23	–	1 PA	–	23 / PA	7,67
`Motoröl Lebenszyklus`	9	5	h	23	–	1 PA	–	23 / PA	7,67
`Zündkerzen linear Verfügbarkeit`	6	4	h	–	10	1/4 PA	–	40 / PA	13,33
`Zündkerzen linear EinServiceIntervall`	9	4	h	–	10	1/4 PA	–	40 / PA	13,33
`Zündkerzen linear Lebenszyklus`	9	5	h	–	10	1/4 PA	–	40 / PA	13, 33
`Zündkerzen adaptiv Verschleiß`	3	1	m	–	7	1/4 PA	–	28 / PA	9,33
`Zündkerzen adaptiv Verfügbarkeit`	6	4	h	–	10	1/4 PA	–	40 / PA	13,33

(Weiter auf der nächsten Seite) ↪ . . .

Funktion	$\#In$	$\#Out$	$Kompl.$	UF_{gem}	UF_{var}	IH $(\#Hooks)$	V	$UPFP$	$UPFP_{PA=3}$
`Zündkerzen adaptiv EinServiceInter-vall`	9	4	h	–	10	1/4 PA	–	40 / PA	13,33
`Zündkerzen adaptiv Lebenszyklus`	9	5	h	–	10	1/4 PA	–	40 / PA	13, 33
`Partikelfilter Verschleiß`	3	1	m	–	7	1/2 PA	–	14 / PA	4,67
`Partikelfilter adaptiv Verfügbarkeit`	9	4	h	–	10	1/2 PA	–	20 / PA	6,67
`Partikelfilter adaptiv EinSer-viceIntervall`	9	4	h	–	10	1/2 PA	–	20 / PA	6,67
`Partikelfilter adaptiv Lebenszyklus`	9	5	h	–	10	1/2 PA	–	20 / PA	6,67
`GFU Verfügbarkeit`	3	2	m	15	–	1 PA	–	15 / PA	5
`GFU EinServiceIn-tervall`	6	2	h	23	–	1 PA	–	23 / PA	7,67
`GFU Lebenszyklus`	6	5	h	23	–	1 PA	–	23 / PA	7,67
`Servicetermin Bestimmen`	12	2	h	23	10	1 PA	4/14	$\frac{135}{7\ \text{PA}}$ = 19,29/PA	6,43
`CBS Anzeige`	1	2	m	15	7	1 PA	1/3	$\frac{37}{3\ \text{PA}}$ = 12,33/PA	4,11
`CBS Serviceintervall Rücksetzen`	4	18	h	23	10	1 PA	12/22	$\frac{175}{11\ \text{PA}}$ = 15,91/PA	5,3
$\sum_f UPFP(f)$								ca. 580,53/PA	ca. 193,51

5.2.2 Kiebuschs 9. Metrik: Allgemeine Aufwandstreiber

Durch eine beispielhafte Bewertung der allgemeinen Aufwandstreiber gemäß der 9. Metrik kommen folgende Werte zustande:

- Dokumentation: 4
- Infrastruktur: 4
- Transitionsprozess: 2
- Wissensvermittlung: 0

$\Rightarrow\ SG^G = 7$

5.2.3 Kiebuschs 10. Metrik: Automotive-Aufwandstreiber

Durch beispielhafte Bewertung der Automotive-spezifischen Aufwandstreiber gemäß der 10. Metrik kommen folgende Werte zustande:

- Rechenleistung: 4
- Sicherheit: 4
- Speicherumfang: 2

$\Rightarrow\ SG^D = 10$

5.2.4 Kiebuschs 12. Metrik: Justierte PFPs

Durch Bewertung der Aufwandstreiber ergibt sich gemäß der 12. Metrik folgender Justierungsfaktor:

- Justierungsfaktor: $J = 0,99$

Als Beispiel der Anwendung einer zählmodellspezifischen Justierung des ermittelten Umfangs sei im Folgenden die Metrik für ein neues Produkt angewendet:

- $\textit{justierte } PFP_{\substack{Produkt \\ (neu)}} = (NPFP + MPFP) * J = (\frac{580,53}{PA} + 0) * 0,99 = \frac{574,72}{PA}$
  (bei $PA = 3$: $191,57$)

5.2.5 Kiebuschs 13. Metrik

Diese Metrik wurde noch nicht angewendet.

5.2.6 Kiebuschs 14. Metrik

Gemäß der 14. Metrik und der oben ermittelten Maße würden folgende Aufwände nötig sein, um die anhand der logischen Architektur angegebene CBS-Produktlinie

zu implementieren:[18]

– Aufwandsprognose auf Basis unjustierter PFP:
$2,4199\,PS * \frac{580{,}53}{PA} = \frac{1404{,}82\,PS}{PA} = \frac{175{,}6\,PT}{PA}$
(bei $PA = 3$: $58,53\,PT$)

– Aufwandsprognose auf Basis justierter PFP:
$2,0987\,PS * \frac{574{,}72}{PA} = \frac{1206{,}65\,PS}{PA} = \frac{150{,}77\,PT}{PA}$
(bei $PA = 3$: $50,26\,PT$)

5.3 Anwendung von Metriken auf CBS-Softwarearchitekturen

Die Anwendung der Metriken in diesem Abschnitt beziehen sich auf die Modellierung der Softwarekomponentenarchitekturen in [GKM07b, Kap. 5].

5.3.1 Produktgenerische Produktliniensoftwarearchitektur für CBS (»CBS-150 %-Softwarearchitektur«)

Siehe hierzu die Modelle in [GKM07b, Abschnitt 5.1].

Kiebuschs unjustierte PFP

Alle Softwarekomponenten wurden als Komponenten mit weichen Echtzeitanforderungen eingestuft. Außerdem wurden alle Softwarekomponenten den Gemeinsamkeiten (*EG*) der Produktlinie zugerechnet, da hier ein produktgenerischer Ansatz verfolgt wird. Zur Komplexitätseinstufung wurde nur die Anzahl der Eingangs- und Ausgangsports von Softwarekomponenten, jedoch nicht die Anzahl der Signale gezählt.

Zur Bewertung wurden alle atomaren Softwarekomponenten der Softwarearchitek-

[18] Gemäß der PFP-Metriken fließen in das Umfangsmaß sowohl Struktur- als auch Verhaltensinformationen ein. In der betrachteten Fallstudie sind die Funktionen jedoch noch nicht ausreichend mit Verhaltensmodellen hinterlegt worden, so dass sich der Aufwand derzeit nur aus der Struktur des Funktionsnetzes herleitet.
PS steht für die Maßeinheit Personenstunde, PT für Personentag.

tur herangezogen, d. h. alle Blattknoten der Hierarchie von Softwarekomponenten. Die produktgenerische Softwarearchitektur wurde so erstellt, dass sie keine explizite strukturelle Varianz enthält, d. h., die strukturelle Varianz der logischen Architektur wurde durch einen 150 %-Ansatz »aufgelöst«. Die in Tabelle 12 aufgelisteten atomaren Softwarekomponenten sind entsprechend ihrer hierarchischen Strukturierung in Sensor-, Client- und Masteranteile sortiert.

Die gemäß der 3., 4. und 5. Metrik ermittelten Maße zur Bestimmung der unjustierten Prozessfamilienpunkte (UPFP) als abstraktes Umfangsmaß für die atomaren Softwarekomponenten der produktgenerischen Softwarearchitektur sind in Tabelle 12 abgebildet.

Tabelle 12 Unjustierte PFP für produktgenerische Produktliniensoftwarearchitektur (CBS-150).

SW-Komponente	*#In*	*#Out*	*Kompl.*	UF_{gem}	UF_{var}	*IH*	*V*	*UPFP*	$UPFP_{PA=3}$
`Kilometerstands-erfassung`	0	1	g	10	–	1 PA	–	10 / PA	3,33
`Bordzeitgeber`	0	1	g	10	–	1 PA	–	10 / PA	3,33
`Sensorik Motoröl`	0	1	g	10	–	1 PA	–	10 / PA	3,33
`Sensorik Partikelfilter`	0	1	g	10	–	1 PA	–	10 / PA	3,33
`Sensorik Zündkerzen`	0	1	g	10	–	1 PA	–	10 / PA	3,33
`Sensorik Bremsbelag vorn`	0	1	g	10	–	1 PA	–	10 / PA	3,33
`Sensorik Bremsbelag hinten`	0	1	g	10	–	1 PA	–	10 / PA	3,33
`CBSC Motoröl adaptiv`	4	1	m	15	–	1 PA	–	15 / PA	5
`CBSC Partikelfilter adaptiv`	4	1	m	15	–	1 PA	–	15 / PA	5
`CBSC Zündkerzen linear`	3	1	m	15	–	1 PA	–	15 / PA	5
`CBSC Zündkerzen adaptiv`	4	1	m	15	–	1 PA	–	15 / PA	5
`CBSC Bremsbelag vorn`	4	1	m	15	–	1 PA	–	15 / PA	5
`CBSC Bremsbelag hinten`	4	1	m	15	–	1 PA	–	15 / PA	5
`CBSC TerminGFU linear`	2	1	g	10	–	1 PA	–	10 / PA	3,33
`CBSM VT1`	2	1	g	10	–	1 PA	–	10 / PA	3,33
`CBSM VT2`	2	1	g	10	–	1 PA	–	10 / PA	3,33
`CBSM VT3`	2	1	g	10	–	1 PA	–	10 / PA	3,33
`CBSM VT4`	2	1	g	10	–	1 PA	–	10 / PA	3,33
`CBSM VT5`	2	1	g	10	–	1 PA	–	10 / PA	3,33

(Weiter auf der nächsten Seite) ↪ . . .

SW-Komponente	$\#In$	$\#Out$	$Kompl.$	UF_{gem}	UF_{var}	IH	V	$UPFP$	$UPFP_{PA=3}$
CBSM VT6	2	1	g	10	–	1 PA	–	10 / PA	3,33
CBSM VT7	2	1	g	10	–	1 PA	–	10 / PA	3,33
CBSM Berechnen	15	17	h	23	–	1 PA	–	23 / PA	7,67
CBSM RÜ1	1	2	m	15	–	1 PA	–	15 / PA	5
CBSM RÜ2	1	2	m	15	–	1 PA	–	15 / PA	5
CBSM RÜ3	1	2	m	15	–	1 PA	–	15 / PA	5
CBSM RÜ4	1	2	m	15	–	1 PA	–	15 / PA	5
CBSM RÜ5	1	2	m	15	–	1 PA	–	15 / PA	5
CBSM RÜ6	1	2	m	15	–	1 PA	–	15 / PA	5
CBSM RÜ7	1	2	m	15	–	1 PA	–	15 / PA	5
Anzeige (Basis)	1	0	g	10	–	1 PA	–	10 / PA	3,33
Anzeige (Erweitert)	1	0	g	10	–	1 PA	–	10 / PA	3,33
Übertragung Schlüssel	1	1	g	10	–	1 PA	–	10 / PA	3,33
Σ_{swc}	64	53						398 / PA	132,67

Kiebuschs 12. Metrik: Justierte PFPs

Durch Bewertung der Aufwandstreiber (die von der Bewertung der logischen Architektur wiederverwendet wurde, siehe Abschnitt 5.2) ergibt sich gemäß der 12. Metrik folgender Justierungsfaktor:

– Justierungsfaktor: $J = 0,99$

Als Beispiel der Anwendung einer zählmodellspezifischen Justierung des ermittelten Umfangs sei im Folgenden die Metrik für ein neues Produkt angewendet:

– $justierte\ PFP_{\substack{Produkt \\ (neu)}} = (NPFP + MPFP) * J = (\frac{398}{PA} + 0) * 0,99 = \frac{394,02}{PA}$
(bei $PA = 3$: $131,34$)

Kiebuschs 14. Metrik

Gemäß der 14. Metrik und der oben ermittelten Maße würden folgende Aufwände nötig sein, um die anhand der produktgenerischen Softwarearchitektur angegebene CBS-Produktlinie zu implementieren:[19]

[19] Es gelten hier die gleichen Einschränkungen wie bei der Berechnung des Umfangsmaßes für das Funktionsnetz, siehe Fußnote 18.

- Aufwandsprognose auf Basis unjustierter PFP:
  $2,4199\,PS * \frac{398}{PA} = \frac{963{,}12\,PS}{PA} = \frac{120{,}39\,PT}{PA}$
  (bei $PA = 3$: $40,13\,PT$)

- Aufwandsprognose auf Basis justierter PFP:
  $2,0987\,PS * \frac{394{,}02}{PA} = \frac{826{,}93\,PS}{PA} = \frac{103{,}37\,PT}{PA}$
  (bei $PA = 3$: $34,47\,PT$)

5.3.2 Produktspezifische Produktliniensoftwarearchitektur für CBS (»CBS-SWPL-Architektur«)

Siehe hierzu die Modelle in [GKM07b, Abschnitt 5.2].

Kiebuschs unjustierte PFP

Alle Softwarekomponenten wurden als Komponenten mit weichen Echtzeitanforderungen eingestuft. Optionale Softwarekomponenten und Alternativen wurden als horizontale EVWs betrachtet. Wenn Softwarekomponenten optionale, abhängige oder variierende Ports besitzen, dann wurden sie als vertikale EGWs oder EVWs eingestuft. Zur Komplexitätseinstufung wurde nur die Anzahl der Eingangs- und Ausgangsports von Softwarekomponenten, jedoch nicht die Anzahl der Signale gezählt.

Zur Bewertung wurden alle atomaren Softwarekomponenten der Softwarearchitektur herangezogen, d. h. alle Blattknoten der Hierarchie von Softwarekomponenten. Die produktspezifische Softwarearchitektur wurde so erstellt, dass sie die strukturelle Varianz der logischen Architektur übernimmt. Demzufolge mussten die in der Hierarchie enthaltenen strukturellen Varianzinformationen (XOR-Varianz und Optionalität bei hierarchisch zusammengesetzten Softwarekomponenten) auf die atomaren Softwarekomponenten entsprechend bezogen werden. Die in Tabelle 13 aufgelisteten atomaren Softwarekomponenten sind entsprechend ihrer hierarchischen Strukturierung in Sensor-, Client- und Masteranteile sortiert.

Die gemäß der 3., 4. und 5. Metrik ermittelten Maße zur Bestimmung der unjustierten Prozessfamilienpunkte (UPFP) als abstraktes Umfangsmaß für die atomaren Softwarekomponenten der produktspezifischen Softwarearchitektur sind in Tabelle 13 abgebildet.

Tabelle 13 Unjustierte PFP für produktspezifische Produktliniensoftwarearchitektur (CBS-SWPL).

SW-Komponente	*#In*	*#Out*	*Kompl.*	UF_{gem}	UF_{var}	*IH*	*V*	*UPFP*	$UPFP_{PA=3}$
Kilometerstands-erfassung	0	1	g	10	–	1 PA	–	10 / PA	3,33
Bordzeitgeber	0	1	g	10	–	1 PA	–	10 / PA	3,33
Sensorik Motoröl	0	1	g	10	–	1 PA	–	10 / PA	3,33
Sensorik Partikelfilter	0	1	g	–	5	1/2 PA	–	10 / PA	3,33
Sensorik Zündkerzen	0	1	g	–	5	1/4 PA	–	20 / PA	6,67
Sensorik Bremsbelag vorn	0	1	g	10	–	1 PA	–	10 / PA	3,33
Sensorik Bremsbelag hinten	0	1	g	10	–	1 PA	–	10 / PA	3,33
CBSC Motoröl adaptiv	4	1	m	15	–	1 PA	–	15 / PA	5
CBSC Partikelfilter adaptiv	4	1	m	–	7	1/2 PA	–	14 / PA	4,67
CBSC Zündkerzen linear	3	1	m	–	7	1/4 PA	–	28 / PA	9,33
CBSC Zündkerzen adaptiv	4	1	m	–	7	1/4 PA	–	28 / PA	9,33
CBSC Bremsbelag vorn	4	1	m	15	–	1 PA	–	15 / PA	5
CBSC Bremsbelag hinten	4	1	m	15	–	1 PA	–	15 / PA	5
CBSC TerminGFU linear	2	1	g	10	–	1 PA	–	10 / PA	3,33
CBSM VT1	2	1	g	10	–	1 PA	–	10 / PA	3,33
CBSM VT2	2	1	g	–	5	1/2 PA	–	10 / PA	3,33
CBSM VT3/4	2	1	g	–	5	1/2 PA	–	10 / PA	3,33
CBSM VT5	2	1	g	10	–	1 PA	–	10 / PA	3,33
CBSM VT6	2	1	g	10	–	1 PA	–	10 / PA	3,33
CBSM VT7	2	1	g	10	–	1 PA	–	10 / PA	3,33
CBSM Berechnen	13	15	h	23	10	1 PA	$\frac{9}{28}$	$\frac{527}{28\ \text{PA}}$	6,27
CBSM RÜ1	1	2	m	15	–	1 PA	–	15 / PA	5
CBSM RÜ2	1	2	m	–	7	1/2 PA	–	14 / PA	4,67
CBSM RÜ3/4	1	2	m	–	7	1/2 PA	–	14 / PA	4,67
CBSM RÜ5	1	2	m	15	–	1 PA	–	15 / PA	5
CBSM RÜ6	1	2	m	15	–	1 PA	–	15 / PA	5
CBSM RÜ7	1	2	m	15	–	1 PA	–	15 / PA	5
Anzeige (Basis)	1	0	g	10	–	1 PA	–	10 / PA	3,33

(Weiter auf der nächsten Seite) ↪ . . .

SW-Komponente	$\#In$	$\#Out$	$Kompl.$	UF_{gem}	UF_{var}	IH	V	$UPFP$	$UPFP_{PA=3}$
`Anzeige (Erweitert)`	1	0	g	–	5	1 PA[20]	–	5 / PA	1,67
`Übertragung Schlüssel`	1	1	g	10	–	1 PA	–	10 / PA	3,33
Σ_{swc}	59	48						$\frac{11111}{28\ \text{PA}}$	132,27
								396,82/PA	

Kiebuschs 12. Metrik: Justierte PFPs

Durch Bewertung der Aufwandstreiber (die von der Bewertung der logischen Architektur wiederverwendet wurde, siehe Abschnitt 5.2) ergibt sich gemäß der 12. Metrik folgender Justierungsfaktor:

– Justierungsfaktor: $J = 0,99$

Als Beispiel der Anwendung einer zählmodellspezifischen Justierung des ermittelten Umfangs sei im Folgenden die Metrik für ein neues Produkt angewendet:

– $justierte\ PFP_{\substack{Produkt \\ (neu)}} = (NPFP + MPFP) * J = (\frac{11111}{28PA} + 0) * 0,99 = \frac{392,85}{PA}$
(bei $PA = 3$: $130,95$)

Kiebuschs 14. Metrik

Gemäß der 14. Metrik und der oben ermittelten Maße würden folgende Aufwände nötig sein, um die anhand der produktspezifischen Softwarearchitektur angegebene CBS-Produktlinie zu implementieren:[21]

– Aufwandsprognose auf Basis unjustierter PFP:
$2,4199\,PS * \frac{11111}{28\,PA} = \frac{960,27\,PS}{PA} = \frac{120,03\,PT}{PA}$
(bei $PA = 3$: $40,01\,PT$)

[20] In der Fallstudie wird die erweiterte Anzeige in keinem der Produkte verwendet. Es ist in den Metriken nicht vorgesehen, Varianten zu planen, die sich nicht in einem Produkt widerspiegeln (es gäbe eine Division durch Null). Da jede vorgesehene Variante mindestens einmal implementiert werden muss, wird hier die Implementierungshäufigkeit mit $1PA$ angenommen.

[21] Es gelten hier die gleichen Einschränkungen wie bei der Berechnung des Umfangsmaßes für das Funktionsnetz, siehe Fußnote 18.

– Aufwandsprognose auf Basis justierter PFP:
$2,0987\,PS * \frac{392{,}85}{PA} = \frac{824{,}48\,PS}{PA} = \frac{103{,}06\,PT}{PA}$
(bei $PA = 3$: $34,35\,PT$)

5.3.3 Vergleich der Softwarearchitekturen

In Tabelle 14 ist eine Übersicht zum Vergleich der produktgenerischen Produktliniensoftwarearchitektur (Spalte »CBS-SW150«) mit der produktspezifischen Produktliniensoftwarearchitektur (Spalte »CBS-SWPL«) angegeben.

In den Spalten »CBS-SWPL: P1/2/3« der Tabelle sind die Kennzahlen für die Einzelprodukte P1, P2 und P3 auf Basis der produktgenerischen Softwarearchitektur angegeben. Die Zeilen »# XXX« der Tabelle geben die Anzahl der jeweils modellierten Elemente in den Softwarearchitekturmodellen wieder und stellen somit die in den Modellen enthaltenen Strukturinformationen dar, die einfach gezählt werden können.

Die beiden unteren Zeilen der Tabelle geben die unjustierten Prozessfamilienpunkte für die beiden Produktlinienarchitekturen bzw. die drei spezifischen Produktarchitekturen an. Die Zeile »UPFP als Indiz für den Aufwand« interpretiert die Werte als Umfangsmaß, die zur Schätzung des Entwicklungsaufwands gemäß der weiteren PFP-Metriken verwendet werden. Dies entspricht der intendierten Verwendung der PFP-Metriken. Die Zeile »UPFP als Indiz für die SW-Größe« interpretiert die UPFP-Werte als Indiz für die Codegröße der zu implementierenden Softwarekomponenten für die Produkte auf Basis des jeweiligen Lösungsentwurfs. Diese Interpretation ist in den Arbeiten zu den PFP-Metriken nicht explizit vorgeschlagen, bietet sich aber im vorliegenden Anwendungskontext an.

Tabelle 14 Vergleich der Softwarearchitekturen.

	CBS-SW150	**CBS-SWPL**	**CBS-SWPL: P1**	**CBS-SWPL: P2**	**CBS-SWPL: P3**
# **Komponenten**	32	30	24	23	24
# **Eingangsports**	64	59	46	45	46
# **Ausgangsports**	53	48	39	38	39
# **Ports**	117	107	85	83	85
# **Komponenten CBS-Clients**	14	14	11	10	11
# **Eingangsports CBS-Clients**	25	25	18	17	18

(Weiter auf der nächsten Seite) ↪ . . .

	CBS-SW150	CBS-SWPL	CBS-SWPL: P1	CBS-SWPL: P2	CBS-SWPL: P3
# **Ausgangsports CBS-Clients**	14	14	11	10	11
# **Komponenten CBS-Master**	15	13	11	11	11
# **Eingangsports CBS-Master**	36	31	26	26	26
# **Ausgangsports CBS-Master**	38	33	27	27	27
UPFP als Indiz für den Aufwand	398 / PA bei PA=3: 132,67	396,82 / PA bei PA=3: 132,27	298	288	298
UPFP als Indiz für die SW-Größe	P1: 328 P2: 318 P3: 328	–	298	288	298

Auf Basis der mit Hilfe der PFP-Metriken ermittelten Werte lassen sich zunächst folgende Alternativen bewerten:

1 Vergleich unterschiedlicher Lösungsentwürfe bei der Produktlinienentwicklung (Dieser Vergleich entspricht der in Abschnitt 1.4 diskutierten Bewertungsaufgabe.)

2 Einzelentwicklung von Produkten gegenüber einer Produktlinienentwicklung (Dieser Vergleich wird in der Fachliteratur zur Softwareproduktlinientechnik überlicherweise angestellt.)

Für den Vergleich können zusätzlich folgende Kriterien und Interpretationen des Umfangsmaßes hinzugezogen werden:

– Schätzung des Entwicklungsaufwands,
– Schätzung der Codegröße der implementierten Softwarekomponenten.

Die beiden Softwarearchitekturmodelle repräsentieren bisher nur eine minimale Produktlinie: in der Produktlinienbeschreibung wurden zwei Varianzpunkte (Otto vs. Diesel sowie lineare vs. adaptive Zündkerzenverschleißerfassung) betrachtet. Darauf aufbauend wurden drei Produkte für die Produktlinie definiert. Daher unterscheiden sich die beiden Produktliniensoftwarearchitekturen nicht sehr in den entsprechenden Maßen, wie in der Tabelle ersichtlich ist. Nichtsdestotrotz lassen sich einige generelle Aussagen ableiten, die durch weitere Fallstudien zu überprüfen sind.

Vergleich der beiden Lösungsentwürfe bei der Produktlinienentwicklung

bzgl. Entwicklungsaufwand

Werden die strukturellen Merkmale der beiden Architekturen (Anzahl der atomaren Softwarekomponenten, Anzahl der Ports) miteinander verglichen, so fällt die produktgenerische Architektur etwas größer aus. Am deutlichsten ist der Unterschied für die Masteranteile: der produktgenerische `CBS-Master` ist für sämtliche potenziell in der Produktlinie vorgesehenen Verschleißumfänge und ihren Varianten ausgelegt, während der `CBS-Master` der produktspezifischen Architektur mit den Varianten bei den Umfängen besser skalierbar ist.

Bei Betrachtung der unjustierten Prozessfamilienpunkte beider Architekturen als Indiz für den Entwicklungsaufwand schneidet die produktgenerische Architektur geringfügig schlechter ab. Zwar werden sämtliche Komponenten für alle Produkte verwendet, was sich im Allgemeinen günstig auf den Aufwand auswirkt, jedoch fallen die Softwarekomponenten etwas komplexer aus, weil die Varianz in die Programmlogik verschoben ist (bspw. durch auszuwertende Konfigurationsparameter beim Systemstart bzw. zur Ausführungszeit; als strukturelles Indiz hierfür gelten die höhere Anzahl an Komponenten sowie die höhere Anzahl an Ports, die eine höhere Komplexitätseinstufung der Komponenten erwirken).

Bei der produktspezifischen Architektur können die Komponenten weniger komplex in der Programmlogik ausgelegt werden (als Indiz hierfür gelten die etwas niedrigere Anzahl von Eingangs- und Ausgangsports der Softwarekomponenten). Der Aufwand in der Generierung solcher Komponenten sowie ihre Verwaltung könnte jedoch auch aufwandserhöhend sein, dies müsste unter Umständen durch Erfahrungswerte bei der Berechnung der UPFPs sowie in der Justierung des Umfangsmaßes einfließen.

Da die betrachtete Produktlinie in der Fallstudie klein ist, fallen die Unterschiede in den UPFPs relativ gering aus. Hier sind noch weitere unterschiedliche Fallstudien nötig, um Erfahrungen zu sammeln, wie sich ein unterschiedlicher Varianzgrad oder die Anzahl der Produkte auf die Bewertung der beiden Softwarearchitekturen auswirkt.

Vergleich der Einzelentwicklung von Produkten mit einer Produktlinienentwicklung

In Tabelle 14 sind die strukturellen Maße als auch die UPFPs für die Einzelprodukte, die idealerweise auf Basis der produktspezifischen Produktliniensoftwarearchitektur erzeugt werden, angegeben. Diese Zahlen können so interpretiert werden, als seien die drei Produkte jeweils in einem eigenständigen Entwicklungsprojekt entstanden. Hiermit lässt sich die in der Produktlinienliteratur häufig gestellte Frage, ob ein Unternehmen seine Produkte in Einzelentwicklungen oder auf Basis eines

Produktlinienansatzes erstellt, relativ eindeutig beantworten: Der Umfang von drei Einzelentwicklungen von insgesamt 884 UPFP im Vergleich zum Umfang auf Basis der Produktlinienentwicklung von 398 $UPFP_{CBS\text{-}150}$ bzw. 396,82 $UPFP_{CBS\text{-}SWPL}$ ist mehr als das Doppelte, bereits bei dieser kleinen Fallstudie. Auch der relative Umfang für ein Produkt fällt auf Basis einer Produktlinie besser aus: 132,67 $UPFP_{CBS\text{-}150}$ bzw. 132,27 $UPFP_{CBS\text{-}SWPL}$ zu 298 $UPFP_{P1,\ P3}$ bzw. 288 $UPFP_{P2}$.

Vergleich der beiden Lösungsentwürfe bei der Produktlinienentwicklung bzgl. Codegröße

Ein weiteres Bewertungskriterium, das in der Automotive-E/E-Entwicklung eine große Bedeutung hat, ist die zu erwartende Codegröße der Softwarekomponenten. Hier kommt zum Tragen, dass einerseits genug Speicher seitens der Hardware im Fahrzeug zur Verfügung stehen muss, um den ausführbaren Softwarebinärcode im RAM, ROM oder Flashspeicher abzulegen, anderseits auch die Flashzeiten, d. h. die Dauer des »Hochladens« der Software ins Fahrzeug während oder nach der Montage des Fahrzeugs, eingeplant werden müssen.

Da das Umfangsmaß sich an strukturellen Eigenschaften der modellierten Softwarekomponenten orientiert, können unjustierte Prozessfamilienpunkte auch als Indiz für die zu erwartende Codegröße der Softwarekomponenten (des Binärcodes bzw. der Speicherbedarf zur Laufzeit) verwendet werden.[22] Diese Interpretation ist in der Zeile »UPFP als SW-Größe« in Tabelle 14 angegeben. Hierzu mussten die UPFPs für jedes Produkt – sowohl auf Basis des produktgenerischen als auch des produktspezifischen Ansatzes – ermittelt werden.

Die UPFP-Werte für die Einzelprodukte auf Basis des produktgenerischen Ansatzes (328 bzw. 318 UPFP) weichen vom UPFP-Wert der generischen Produktlinienarchitektur (398 UPFP) ab, weil sich die Produkte auf Infrastrukturebene unterscheiden: Fahrzeuge mit Ottomotoren besitzen nicht das DDE-Steuergerät, auf dem die Verschleißerfassung für den Partikelfilter installiert worden wäre. Demzufolge fallen die Umfangsmaße für Einzelprodukte in der Interpretation als Indiz für die Codegröße kleiner aus.

Beim Vergleich des produktgenerischen und des produktspezifischen Ansatzes

[22] Diesem Ansatz liegt folgende Annahme zu Grunde: Da für die UPFPs die Anzahl der Eingangs- und Ausgangssignale (bzw. Anzahl der Kanten und Knoten in den Statecharts) berücksichtigt wird, kann davon ausgegangen werden, dass die Größe des Softwarecodes sich proportional zur Anzahl der verarbeiteten Signale (bzw. Anzahl der Kanten und Knoten) verhält. Analoges gilt auch für den Speicherbedarf zur Laufzeit.

zeigt sich, dass sämtliche Produkte auf Basis des produktspezifischen Ansatzes kleiner als die Produkte auf Basis des produktgenerischen Ansatzes ausfallen: z. B. 298 $UPFP_{P1(CBS\text{-}SWPL)}$ zu 328 $UPFP_{P1(CBS\text{-}150)}$. Der Grund hierfür liegt darin, dass kein »toter« Code für Varianten, die nicht zum Zuge kommen, ins Fahrzeug installiert wird. Auch der Speicherbedarf zum Ausführungszeitpunkt ist niedriger, weil weniger CBS-Umfänge vom `CBS-Master` auszuwerten sind.

Resümee

Zusammenfassend lässt sich sagen, dass auf Basis der in der CBS-Fallstudie erfassten Informationen die Lösungsalternative für die Softwareentwicklung auf Basis des produktspezifischen Ansatzes besser als die Lösungsalternative auf Basis des produktgenerischen Ansatzes erscheint. Einschränkend muss aber gesagt werden, dass nicht alle Bewertungskriterien in der oben angestellten Bewertung berücksichtigt wurden. Einige Faktoren, z. B. eine quantifizierte Bewertung der Verwendung der jeweils notwendigen Softwaretechnologien mit ihren Konsequenzen, wurden ebensowenig explizit berücksichtigt. Auch die Feinjustierung der PFP-Metriken konnte noch nicht erfolgen. Bisher wurden nur die vorgeschlagenen Grenzwerte und Erfahrungswerte aus [Kie06a] verwendet.

6 Zusammenfassung und Ausblick

In diesem Bericht wurden Entscheidungsgrundlagen für die Entwicklung von Softwareproduktlinien untersucht. In den beiden vorigen Phasen des Projekts VEIA wurden die Artefakte des Referenzprozesses definiert und ein allgemeiner Rahmen für Architekturbewertungen definiert.

Die Architekturbewertung wurde nun so konkretisiert, dass auf der Grundlage der Artefakte des Referenzprozesses tatsächliche Entscheidungen getroffen werden können. Dazu wurden die Kriterien für die Entscheidung bestimmt und am Beispiel CBS diskutiert. Danach wurden Anforderungen an Metriken aufgestellt, mit denen die Erfüllung der Kriterien gemessen werden kann.

Im Hauptteil der Arbeit wurden bestehende Metriken daraufhin untersucht, ob und wie sie zur Bewertung der Artefakte des Referenzprozesses einsetzbar sind. Dabei musste untersucht werden, welche Informationen im VEIA-Referenzprozess fehlen und welche Anpassungen bzw. Interpretationen der Metriken notwendig sind, um aussagekräftige Ergebnisse zu erhalten.

Es hat sich gezeigt, dass bereits einige Metriken zur Verfügung stehen, die auf Produktlinien angewendet werden können. Zum Teil arbeiten sie jedoch auf anderen Modellen als im VEIA-Referenzprozess vorgeschlagen. Vor allem der Umgang mit und die Darstellung von Varianz in Produktlinienmodellen ist sehr unterschiedlich.

Um die Metriken auf die Artefakte des VEIA-Referenzprozesses anwenden zu können, ist eine Anpassung der Metriken notwendig. Außerdem muss der Umgang mit den Begriffen »Produkt« und »Produktlinien« an das Anwendungsgebiet von VEIA – Automotive-E/E-Systeme in einem verteilten Entwicklungs- und Integrationsprozess – angepasst werden. Hier sind die vorliegenden Arbeiten zu generisch.

Bezüglich einer konkreten Metrik, der PFP-Metrik nach Kiebusch (Abschnitt 4.3), sind bereits notwendige Anpassungen identifiziert, die direkt oder indirekt einen Einfluss auf die anfallenden Aufwände haben. Hierzu gehören die Kennzahlen für die Komplexitätsgewichtung, die Benennung der spezifischen Aufwandstreiber sowie die Kennzahlen zur Abschätzung des Aufwands aufgrund von Umfangsmaßen. Umgekehrt definieren beispielsweise die Kennzahlen für die Komplexitätsgewichtung aber auch Anforderungen an die Modellierung, bis zu welcher Detailtiefe spezifiziert werden muss, bevor die Metriken anwendbar sind.

Die Kiebusch-Metrik wurde beispielhaft anhand von Funktions- und Softwarear-

chitekturmodellen angewendet, die im Rahmen der VEIA-Fallstudie CBS erstellt wurden. Diese geben einen ersten Einblick zur Interpretation der aus der Anwendung der Metriken erhaltenen Ergebnisse. Jedoch müssen im weiteren Verlauf des Projekts noch mehr Erfahrungen hierzu gesammelt werden. Die Beispielanwendung der PFP-Metriken auf CBS-Modelle ist in Kapitel 5 angegeben.

Es kann resümiert werden, dass Aussagen mit vorhandenen Metriken möglich sind. Dennoch sind vielfältige Anpassungen nötig, da zum einen im VEIA-Referenzprozess Daten über das Produktumfeld – Projekt, Prozess, Organisation – nicht erfasst sind, die dementsprechend zusätzlich erhoben werden müssen. Zum anderen sind die Interpretationen, z. B. der Begriffe »Funktion« oder »Varianz« oft weit vom VEIA-Verständnis entfernt. Hier müssen vor allem die Metriken auf Aussagegehalt überprüft und gegebenenfalls angepasst werden.

Literatur

[AUT06] AUTOSAR Development Partnership: *AUTOSAR Software Component Template*. Version 2.0.1. Juni 2006. http://www.autosar.org/download/AUTOSAR_SoftwareComponentTemplate.pdf

[Bal01] *Balzert*, Helmut: *Lehrbuch der Software-Technik: Software-Entwicklung*. Heidelberg u. a. : Spektrum-Akademischer Verlag, 2001

[BKPS04] *Böckle*, Günter (Hrsg.) ; *Knauber*, Peter (Hrsg.) ; *Pohl*, Klaus (Hrsg.) ; *Schmid*, Klaus (Hrsg.): *Software-Produktlinien – Methoden, Einführung und Praxis*. 1. Auflage 2004. dpunkt.verlag http://www.dpunkt.de/spl/

[BW01a] *Briand*, Lionel C. ; *Wüst*, Jürgen: The Impact of Design Properties on Development Cost in Object-Oriented Systems. In: *IEEE Transactions on Software Engineering* 27 (2001), Nr. 11, 963–986. http://www.sdmetrics.com/

[BW01b] *Briand*, Lionel C. ; *Wüst*, Jürgen: Integrating Scenario-based and Measurement-based Software Product Assessment. In: *Journal of Systems and Software* 59 (2001), 3-22. http://www.sdmetrics.com/

[BW02] *Briand*, Lionel C. ; *Wüst*, Jürgen: Empirical Studies of Quality Models in Object-Oriented Systems. In: *Advances in Computers* 59 (2002), 97-166. http://www.sdmetrics.com/

[DMv01] *Dincel*, Ebru ; *Medvidovic*, Nenad ; *van der Hoek*, André: Measuring Product Line Architectures. In: *Proc. 4th Int. WS on Software Product-Family Engineering (PFE 2001)* Bd. 2290. Berlin, Germany : Springer-Verlag, 2001, S. 346–352

[GEKM07] *Große-Rhode*, Martin ; *Euringer*, Simon ; *Kleinod*, Ekkart ; *Mann*, Stefan: Grobentwurf des VEIA-Referenzprozesses / Fraunhofer-Institut für Software- und Systemtechnik, Abt. Verlässliche technische Systeme. Mollstraße 1, 10178 Berlin, Germany, Januar 2007. – ISST-Bericht 80/07. Projekt VEIA. http://veia.isst.fraunhofer.de/

[GKM07a] *Große-Rhode*, Martin ; *Kleinod*, Ekkart ; *Mann*, Stefan: Architekturbewertung / Fraunhofer-Institut für Software- und Systemtechnik, Abt. Verlässliche technische Systeme. Mollstraße 1, 10178 Berlin, Germany, Juli 2007. – ISST-Bericht 82/07. Projekt VEIA. http://veia.isst.fraunhofer.de/

[GKM07b] *Große-Rhode*, Martin ; *Kleinod*, Ekkart ; *Mann*, Stefan: Fallstudie »Condition-Based Service«: Modelle für die Bewertung von logischen Architekturen und Softwarearchitekturen / Fraunhofer-Institut für Software- und Systemtechnik, Abt. Verlässliche technische Systeme. Mollstraße 1, 10178 Berlin, Germany, Oktober 2007. – ISST-Bericht 84/07. Projekt VEIA. http://veia.isst.fraunhofer.de/

[KBS06] *Kiebusch*, Sebastian ; *Bogdan*, Franczyk ; *Speck*, Andreas: An Unadjusted Size Measurement of Embedded Software System Families and its Validation. In: *Software Process Improvement And Practice* 11 (2006), Juni, 435–446. http://dx.doi.org/10.1002/spip.285. – DOI 10.1002/spip.285

[Kie05] *Kiebusch*, Sebastian: Metriken: Definition und Validierung / PESOA-Projekt. 2005. – PESOA-Report No. 14/2005

[Kie06a] *Kiebusch*, Sebastian: *Metriken für prozessorientierte Software-System-Familien: Umfangskalkulation sowie Aufwandsprognose im Electronic Business und Automobilbereich*. Leipzig, Germany, Institut für Wirtschaftsinformatik, Universität Leipzig, Dissertation, 2006. http://www.kiebusch.de/

[Kie06b] *Kiebusch*, Sebastian: Ökonomisches Management von Prozess-Familien. In: *OBJEKTspektrum* 2 (2006), S. 77–84.

[Kle06] *Kleinod*, Ekkart: Modellbasierte Systementwicklung in der Automobilindustrie – Das MOSES-Projekt / Fraunhofer-Institut für Software- und Systemtechnik, Abt. Verlässliche technische Systeme. – ISST-Bericht 77/06. http://veia.isst.fraunhofer.de/

[Kop97] *Kopetz*, H.: *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997

[Mis06] *Misic*, Vojislav B.: Measuring the Coherence of Software Product Line Architectures / Dep. of CS, University of Manitoba. Wiinipeg, Manitoba, Canada, Juni 2006. – TR 06/03

[RSW04] *Richter*, Ernst ; *Schnieders*, Arnd ; *Weiland*, Jens: Prozessanalyse und -modellierung in der Domäne Automotive / BMBF-Forschungsprojekt PESOA. Ulm u. a., Oktober 2004. – PESOA-Report Nr. TR 07/2004. http://www.pesoa.org/

[Sch03] *Schmid*, Klaus: A Quantitative Model of the Value of Architecture in Product Line Adoption / Fraunhofer IESE. 2003. – IESE-Report No. 103.03/E

[SL06a] *Schackmann*, Holger ; *Lichter*, Horst: A Cost-Based Approach to Software Product Line Management. In: *Proc. Int. WS on Software Product Management at 14th IEEE Int. Req. Eng. Conf. (RE2006)*. – Minneapolis/St. Paul, Minnesota, USA, Sep. 12, 2006

[SL06b] *Schackmann*, Holger ; *Lichter*, Horst: Using Costing Information as Decision Support in Variability Management. In: *Clements*, Paul (Hrsg.) ; *Muthig*, Dirk (Hrsg.): *Variability Management – Working with Variability Management*, Fraunhofer IESE. – Proc. of the Workshop held in conjunction with the 10th Software Product Line Conf. (SPLC 2006). IESE-Report No. 152.06/E

[vDM03] *van der Hoek*, André ; *Dincel*, Ebru ; *Medvidovic*, Nenad: Using Service Utilization Metrics to Assess the Structure of Product Line Architectures. In: *Proc. of the 9th Int. Software Metrics Symposium (METRICS'03)*, 2003, S. 298–308. – Sydney, Australia, September 3-5, 2003

[Wüs06] *Wüst*, Jürgen: *SDMetrics User Manual*, Oktober 2006. http://www.sdmetrics.com/

[Wüs07] *Wüst*, Jürgen: *SDMetrics*. CASE-Tool. http://www.sdmetrics.com/. Version: 2007

[ZC03] *Zubrow*, Dave ; *Chastek*, Gary: Measures for Software Product Lines / Software Engineering Institute, Carnegie Mellon University. Pittsburgh, PA, USA, 2003. – Technicla Note CMU/SEI-2003-TN-031. http://www.sei.cmu.edu/publications/documents/03.reports/03tn031.html