UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Entwurf und Evaluierung einer kennzahlenorientierten Gestaltung des Softwareentwicklungsprozesses in produzierenden Betrieben

Der Fakultät für Elektrotechnik, Informatik und Mathematik der
Universität Paderborn
zur Erlangung des akademischen Grades eines
Dr. rer. nat.

eingereichte Dissertation

von
Dipl.-Ing. Andreas Deuter

Eingereicht im März 2018

Erster Gutachter: Prof. Dr. Gregor Engels
Zweiter Gutachter: Prof. Dr. Dennis Kundisch

Danksagung

Ich bedanke mich bei allen Personen, die mich bei der Erstellung dieser Dissertation unterstützten.

Ich bedanke mich bei meinen Gutachtern Prof. Dr. Gregor Engels und Prof. Dr. Dennis Kundisch für ihre stets konstruktive Kritik. Sie gaben mir die wesentlichen Impulse für das wissenschaftliche Arbeiten. Recht herzlichen Dank!

Ich bedanke mich bei meinen ehemaligen Kollegen und Kolleginnen von Phoenix Contact. Insbesondere bedanke ich mich bei Hans-Jürgen Koch, Detlev Kuschke, Werner Neugebauer, Oliver Stallmann, Dr. Tobias Frank und Marco Hogrebe, die mich bei der Durchführung und der Auswertung der Softwaremessungen unterstützten. Bei allen anderen nichtgenannten Kollegen und Kolleginnen bedanke ich mich für die vielen Gespräche über das Dissertationsthema und für ihre wichtigen Anregungen.

Ich bedanke mich bei meinen jetzigen Kollegen und Kolleginnen der Hochschule Ostwestfalen-Lippe für ihre Tipps und Hinweise.

Ich bedanke mich bei meinen Eltern, die mir meine akademische Laufbahn ermöglichten.

Mein innigster Dank gilt meiner Frau Gitta. Letztendlich war es ihr steter Rückhalt, der mir die Kraft gab, diese Dissertation zu Ende zu führen.

Abstract

Increasing digitalization in the industrial sector is requiring ever more intelligent products. Intelligent products are based on mechatronics with an increasing importance of software for the added value of the products. Thus, manufacturing companies developing and supplying intelligent products are challenged to most efficiently design their software developing process as a part of their product development. The demands of this process design are based on both, the strategic and operational goals of the manufacturing company. To successfully monitor the achievement of targets it is essential to implement key figures directly linked to the targets and acquired during the software development process. For this reason it is necessary to place great emphasis on the acquisition of key figures when designing the software development process.

This thesis deals with the question how manufacturing companies can design their software development process in a key figure-oriented way, taking into account that the information needs of the software teams directly involved in the process as well as those of the management shall be fulfilled. Due to their profound knowledge of the software domain, the first target group can be supported by established software key figures; this, however, is not true for the management. For this reason, the main issue of this thesis focuses on the question whether today's production key figures can be used for the software development process. As the manufacturing process is a central and directly value-adding process of each manufacturing company, the management is familiar with production key figures.

The results of this thesis were evaluated in cooperation with the company Phoenix Contact Electronics GmbH, a manufacturing company within the sense of this thesis.

Zusammenfassung

Durch die zunehmende Digitalisierung werden in der Industrie immer mehr intelligente Produkte benötigt. Intelligente Produkte beruhen auf der Mechatronik, wobei der Anteil der Software an deren Wertschöpfung stetig steigt. Produzierende Betriebe, die intelligente Produkte entwickeln und vertreiben, sind folglich aufgefordert, die Softwareentwicklungsprozesse als Teil der Produktentwicklung möglichst effizient zu gestalten. Die Anforderungen an diese Prozessgestaltung leiten sich aus den strategischen und operativen Zielen eines produzierenden Betriebes ab. Um eine Zielerreichung überprüfen zu können, müssen Kennzahlen eingesetzt werden. Sie zeichnen sich dadurch aus, dass sie in einem unmittelbaren Zusammenhang zu den Zielen stehen und dass sie im Softwareentwicklungsprozess erfasst werden. Um Letzteres zu gewährleisten, ist folglich die Kennzahlenerfassung in der Gestaltung des Softwareentwicklungsprozesses zu beachten.

Diese Arbeit behandelt die Fragestellung, wie produzierende Betriebe den Softwareentwicklungsprozess kennzahlenorientiert gestalten können. Dabei wird berücksichtigt, dass sowohl die Informationsbedürfnisse der unmittelbar im Prozess beteiligten Softwareteams als auch die Informationsbedürfnisse des Managements befriedigt werden. Zwar können etablierte Softwarekennzahlen die erste Zielgruppe aufgrund ihres ausgeprägten Wissens über die Softwaredomäne unterstützen, allerdings nicht das Management. Der Fokus dieser Arbeit liegt aus diesem Grund in der Fragestellung, ob heutige Produktionskennzahlen im Softwareentwicklungsprozess eingesetzt werden können. Das Management ist gut mit Produktionskennzahlen vertraut, da der Produktionsprozess ein zentraler, direkt wertschöpfender Prozess eines jeden produzierenden Betriebes ist.

Die Ergebnisse dieser Arbeit wurden in Zusammenarbeit mit der Phoenix Contact Electronics GmbH, einem produzierenden Betrieb im Sinne dieser Arbeit, evaluiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ALM	Application Lifecycle Management
AV	Wertschöpfung
AV_v	Wertschöpfung für eine Softwareversion
BSC	Balanced Scorecard
DR	Design Research
Ch_{d_s}	Single Defect Churn
Ch_{di_s}	Single Internal Defect Churn
Ch_{di_v}	Version Internal Defect Churn
Ch_{dx_p}	Product External Defect Churn
Ch_{dx_s}	Single External Defect Churn
Ch_{file}	File Churn
Ch_{f_s}	Single Feature Churn
Ch_{f_v}	Version Feature Churn
Ch_m	Module Churn
Ch_v	Version Churn
D_v	Entwicklungsdauer einer Softwareversion
DD_{ch_v}	Churn-Fehlerdichte einer Softwareversion
DFR_p	Fehlerbehebungsrate für ein Softwareprodukt
Doc_{f_s}	Dokumentationsgröße einer Softwarefunktion
Doc_{f_v}	Dokumentationsgröße einer Softwareversion
E_{dev_v}	Ist-Aufwand der Entwicklungsaktivitäten für eine Softwareversion
E_{doc_v}	Ist-Aufwand der Dokumentationsaktivitäten für eine Softwareversion
FPR	First Pass Rate
G_{khw}	Zu einer HW-Produktionskennzahl gehörendes Ziel
G_{ksw}	Zu einer SW-Produktionskennzahl gehörendes Ziel
GQM	Goal-Question-Metric
I_{khw}	Zu einer HW-Produktionskennzahl gehörende Interpretation
I_{ksw}	Zu einer SW-Produktionskennzahl gehörendes Interpretation
KB	Kilobyte

K_{HW}	Menge an Hardware(HW)-Produktionskennzahlen
k_{HW}	Eine Hardware(HW)-Produktionskennzahl
K_{SW}	Menge an Software(SW)-Produktionskennzahlen
k_{SW}	Eine Software(SW)-Produktionskennzahl
K_S	Menge an Softwarekennzahlen
P	Produktivität
P_{ch_v}	Churn-Produktivität der Entwicklung einer Softwareversion
P_{doc_v}	Dokumentationsproduktivität einer Softwareversion
P_v	Produktivität für eine Softwareversion
$RGQM$	Reversed-Goal-Question-Metric
Q_{khw}	Zu einer HW-Produktionskennzahl gehörende Frage
Q_{ksw}	Zu einer SW-Produktionskennzahl gehörende Frage
SL	Servicegrad
SL_v	Servicegrad für eine Softwareversion
TRR	Technische Rückläuferrate
TRR_p	Technische Rückläuferrate für ein Softwareprodukt
V_{ch_v}	Churn-Liefergeschwindigkeit einer Softwareversion
V_{doc_v}	Dokumentationsliefergeschwindigkeit einer Softwareversion
WI	Work Item
W_{di_p}	Anzahl an intern entdeckten Fehlern für ein Softwareprodukt
W_{dx_p}	Anzahl an extern entdeckten Fehlern für ein Softwareprodukt

Kapitel 1

Einleitung

1.1 Ausgangssituation und Abgrenzung

Die Digitalisierung der Prozesse in der Industrie hat die Flexibilisierung der Produktion, eine bessere Vernetzung von Entwicklungs- und Produktionsprozessen und folglich die Stärkung der Wettbewerbsfähigkeit produzierender Betriebe zum Ziel. Die technische Grundlage dafür sind intelligente Produkte, die eine weitestgehend selbstorganisierende Produktion und deren Vernetzung zu den angrenzenden Prozessen ermöglichen sollen [Pla17]. In Deutschland wird dieser Trend zur Digitalisierung der Produktionsprozesse als Industrie 4.0 bezeichnet [KWH17]. Produzierende Betriebe, die in dieser Arbeit thematisiert werden, stellen die für die Industrie 4.0 benötigten intelligenten Produkte mit eigenen Produktionsmitteln her. Beispiele für intelligente Produkte im Kontext dieser Arbeit sind speicherprogrammierbare Steuerungen oder regelbare Spannungsversorgungen.

Intelligente Produkte beruhen auf der Mechatronik. Sie werden durch ein Zusammenwirken mehrerer Domänen entwickelt, wie zum Beispiel der Mechanik, der Elektronik und der Softwaretechnik [AG12]. Systems Engineering ist ein methodischer Ansatz für die domänenübergreifende Entwicklung intelligenter Produkte. Der Begriff „Systems Engineering“ ist zwar vielfältig definiert, beschreibt im Wesentlichen jedoch das Management parallellaufender Entwicklungsprozesse in den beteiligten Domänen [BB16]. Eine konkrete Ausgestaltung eines auf den Methoden des Systems Engineering beruhenden Entwicklungsprozesses beschreibt die VDI-Richtlinie 2206 [VDI04]. Der darin beschriebene Entwicklungsprozess baut auf dem V-Modell auf, das aus der Softwareentwicklung [Boe79] stammt und in der VDI 2206 an die Anforderungen an eine domänenübergreifende Entwicklung angepasst wurde. Wie in Abbildung 1.1 gezeigt, erfolgt ausgehend von den Produktanforderungen der domänenübergreifende Systementwurf. Es folgen die domänenspezifischen Entwürfe, die laut der VDI-Richtlinie 2206 meist getrennt in den be-

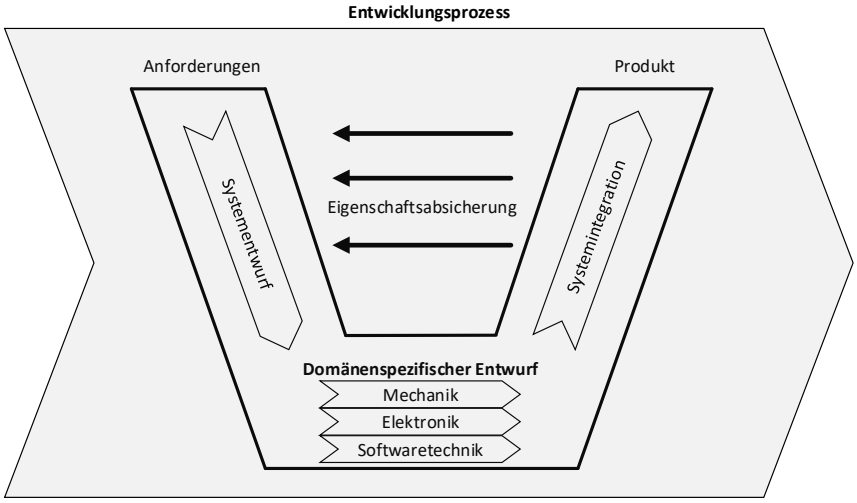


Abbildung 1.1: Domänenübergreifender Entwicklungsprozess nach VDI 2206 [VDI04]

teiligten Domänen erfolgen. In der Systemintegration werden die Ergebnisse der einzelnen Domänen zusammengeführt und die Eigenschaften des intelligenten Produktes geprüft.

Der Umfang der Beteiligung der einzelnen Domänen in der Produktentwicklung hängt von den Merkmalen des konkreten intelligenten Produktes ab, wobei die Softwaretechnik zunehmend eine strategische Rolle einnimmt, denn: „die Analysten sind sich einig, dass seit etlichen Jahren die allermeisten Produktinnovationen in der Software begründet sind“ [Sen14]. Der softwaredomänenspezifische Entwurf wird Softwareentwicklungsprozess genannt.

Es ist die strategische Aufgabe des Managements, also der Führungskräfte in den oberen Hierarchieebenen wie zum Beispiel in der Bereichsleitung oder in der Geschäftsführung, die durch Industrie 4.0 motivierten Prozessänderungen im Produktlebenszyklus intelligenter Produkte im eigenen produzierenden Betrieb zu gestalten. Der Produktlebenszyklus umfasst sowohl die Produktentwicklung als auch die Produktion.

Ausgangspunkt für die Gestaltung aller betrieblichen Prozesse ist die Unternehmensstrategie. Um eine Strategie zu operationalisieren, sind vom Management strategische Ziele zu formulieren, die den Ausgangspunkt für prozessgestaltende Maßnahmen bilden. Für jedes der strategischen Ziele sind Kennzahlen zu definieren, die die Zielerreichung der strategischen Ziele anzeigen [See08]. Um die Kennzahlen auswerten zu können, muss in der Prozessgestaltung deren Erfassung berücksichtigt werden: Der Prozess muss derart gestaltet oder angepasst werden, dass Kennzahlen erfasst werden können.

Die Kennzahlen werden entsprechend den Informationsbedürfnissen des Managements zusammengestellt und aufbereitet. Um die Kennzahlen beurteilen und interpretieren zu können, muss das Management deren *Semantik* kennen. Für den Begriff „Semantik einer

Kennzahl“ gibt es zwar keine allgemein anerkannte Definition, allerdings sind aus der Literatur mehrere Formen für die Beschreibung von Kennzahlen bekannt, zum Beispiel ein Kennzahlenstammblatt [Ben07], ein Kennzahlensteckbrief [Küt10] oder ein Kennzahlenformular [HP07]. Diese möglichen Beschreibungsformen einer Kennzahl enthalten zahlreiche kennzahlenspezifische Informationen wie zum Beispiel Name, Maßeinheit, Wertebereich, Berechnungsgrundlagen und Interpretation. Durch die gesamtheitliche Kenntnis all dieser Informationsinhalte entsteht beim Adressaten einer Kennzahl ein Verständnis über deren Semantik. Diese Semantik ist jedoch oftmals betriebsspezifisch. Damit ist gemeint, dass eine Kennzahl mit demselben Namen in verschiedenen produzierenden Betrieben eine unterschiedliche Bedeutung haben kann.

Abbildung 1.2 veranschaulicht den erläuterten Ablauf, der sich in zwei Phasen gliedert: In der ersten Phase formuliert das Management die strategischen Ziele. Auf deren Basis werden durch verschiedene Stakeholder in einem produzierenden Betrieb Kennzahlen definiert, die für die Überprüfung der Zielerreichung der strategischen Ziele benötigt werden. Sowohl die strategischen Ziele als auch die daraus definierten Kennzahlen sind die Basis für die Gestaltung eines betrieblichen Prozesses. Die Gestaltung des Prozesses erfolgt wiederum von den verschiedenen Stakeholdern. Diese erste Phase ist die *Gestaltungsphase*, deren Abläufe in Abbildung 1.2 mit dünnen Strichen gekennzeichnet sind. In der zweiten Phase werden während der Durchführung des betrieblichen Prozesses die Kennzahlen mit einem Informationsverarbeitungssystem erfasst. Damit ist ein IT-System gemeint, das Daten aus dem betrieblichen Prozess ermittelt, zu Kennzahlen verarbeitet und diese grafisch darstellt [Ben07]. Die grafisch aufbereiteten Kennzahlen werden vom Management interpretiert. Diese zweite Phase ist die *Erfassungsphase*, deren Abläufe in der Abbildung mit dicken Strichen gekennzeichnet sind.

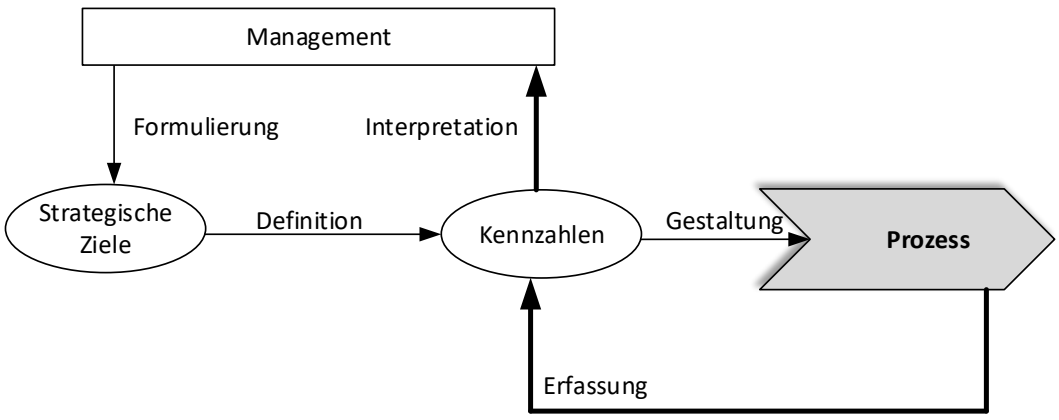


Abbildung 1.2: Schematischer Ablauf einer kennzahlenorientierten Prozessgestaltung

Dieses Vorgehen wird in dieser Arbeit *kennzahlenorientierte Prozessgestaltung* genannt. In den folgenden Abschnitten wird der Stand der kennzahlenorientierten Prozessgestaltung in der Produktion und in der Softwareentwicklung in produzierenden Betrieben als Ausgangspunkt für die Herleitung der Forschungsfrage dargelegt.

1.2 Kennzahlenorientierte Gestaltung der Produktion

Der Produktionsprozess ist ein zentraler Prozess in produzierenden Betrieben, da er direkt wertschöpfend ist. Er beeinflusst in hohem Maße die Herstellkosten, die Lieferzeiten und die Qualität der Produkte und trägt entscheidend zum wirtschaftlichen Erfolg der produzierenden Betriebe bei.

Im Folgenden wird eine betriebsspezifische kennzahlenorientierte Gestaltung des Produktionsprozesses von intelligenten Produkten am Beispiel des Geschäftsbereichs Automatisierung der Phoenix Contact Electronics GmbH, dem Kooperationspartner dieser Arbeit, dargestellt. Die Phoenix Contact Gruppe ist ein weltweit führender Anbieter von elektrischer Verbindungs- und elektronischer Interfacetechnik sowie industrieller Automatisierungstechnik. Im Geschäftsbereich Automatisierung der Phoenix Contact Electronics GmbH werden u.a. intelligente Produkte für die Fabrikautomatisierung entwickelt und gefertigt. Beispiele für solche Produkte sind speicherprogrammierbare Steuerungen und Feldgeräte mit industriellen Netzwerkanschlüssen. Der Verfasser dieser Arbeit arbeitete viele Jahre beim Kooperationspartner. Der im Folgenden ausgeführte Stand der kennzahlenorientierten Prozessgestaltung in der Produktion wurde zum einen in unstrukturierter Form beobachtet und basiert zum anderen auf Gesprächen mit dem Produktionsleiter, dem Geschäftsbereichsleiter und ausgewählten Teamleitern. Einige der Gespräche wurden in Form eines Interviews geführt.

Für die Veranschaulichung der kennzahlenorientierten Prozessgestaltung des Produktionsprozesses dient Abbildung 1.3. Das Management formuliert strategische Ziele, zum Beispiel die Erreichung einer hohen Fertigungsqualität. Basierend auf diesen strategischen Zielen werden verschiedene Produktionskennzahlen definiert, die die Zielerreichung der strategischen Ziele anzeigen. Diese Produktionskennzahlen bilden die Menge K_{HW} (HW steht für Hardware). Ein Beispiel einer Produktionskennzahl ist die sogenannte First Pass Rate (FPR). Die FPR ist das Verhältnis von den bestandenen Fertigungsendtests zu allen durchgeführten Fertigungsendtests. Beim Kooperationspartner wird jedes einzelne gefertigte intelligente Produkt einem Fertigungsendtest unterzogen.

Ableitend aus den strategischen Zielen formulieren die Produktionsteams operative Ziele. Das sind kurz- oder mittelfristige Ziele, die der Steuerung von konkreten Produktions-

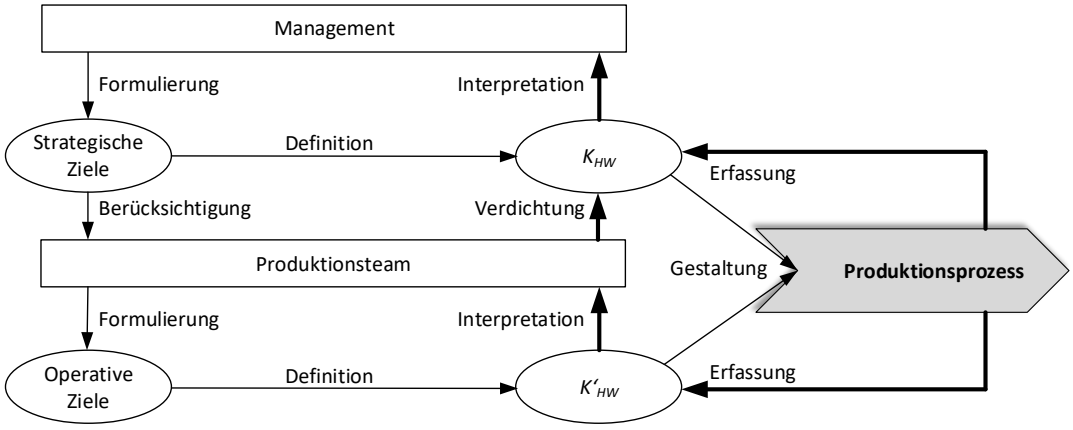


Abbildung 1.3: Beispiel einer kennzahlenorientierten Gestaltung des Produktionsprozesses

linien dienen. Die operativen Ziele sind der Ausgangspunkt für kontinuierliche Verbesserungsmaßnahmen. Kontinuierliche Verbesserungsmaßnahmen sind Teil von kontinuierlichen Verbesserungsprozessen (KVP), die viele kleine Änderungen bewirken, aus deren Summe erhebliche Prozessverbesserungen erreicht werden können [Bec05]. Tabelle 1.1 zeigt ein KVP-Beispiel beim Kooperationspartner.

Neben den kontinuierlichen Verbesserungsmaßnahmen werden auch strategische Maßnahmen realisiert. Diese Maßnahmen müssen anders als die KVP-Maßnahmen vom Management bestätigt werden. Ein Beispiel dafür ist die Beschaffung einer vollautomatisierten Leiterplattenbestückungs- und Lötmaschine im Wert von ca. einer Million Euro.

Basierend auf den operativen Zielen werden mehrere Produktionskennzahlen definiert. Diese Produktionskennzahlen bilden die Menge K'_{HW}. Beim Kooperationspartner sind einige der Produktionskennzahlen, die Teil der Menge K'_{HW} sind, auch Teil der Menge K_{HW}. Damit ist gemeint, dass diese Produktionskennzahlen mit dem gleichen Namen und der gleichen Bedeutung in beiden Mengen verwendet werden. So ist zum Beispiel die FPR Bestandteil in beiden Mengen, jedoch unterscheidet sie sich in dem zeitlichen oder organisatorischen Geltungsbereich: Die Produktionskennzahlen in K'_{HW} zeigen de-

Ist-Stand am 10.04.:	221 Geräte gefertigt mit FPR 87,78 %
Ursachen:	1. Lötfehler Handlötung
	- Rangierplatine nicht richtig verlötet
	2. Testadapter
	- FE-Klammer kontaktiert nicht richtig
Maßnahmen:	1. Mitarbeiter geschult
	2. Adaptierung durch zwei Klammern
Ist-Stand am 27.05.	29 Geräte gefertigt mit FPR 96,67 %

Tabelle 1.1: KVP-Beispiel beim Kooperationspartner

taillierte Informationen zu einem Fertigungslos, zu einem Produktionstag etc. an. Die Produktionskennzahlen in K_{HW} zeigen verdichtete Informationen zu einer Fertigungsstätte, einem Quartal etc. an. K_{HW} ist folglich eine verdichtete Darstellung von K'_{HW}, an deren Aufbereitung die Produktionsteams beteiligt sind. Diese Aufbereitung ändert allerdings nicht die Semantik der Produktionskennzahlen.

Die Produktionskennzahlen in K'_{HW} werden im Produktionsprozess erfasst. Um diese in K_{HW} verdichtet dem Management zur Verfügung zu stellen, wird wie am folgenden Beispiel der FPR gezeigt vorgegangen: Die FPR verschiedener Produktionslinien werden mit Hilfe eines Informationsverarbeitungssystems erfasst und gehen in K'_{HW} ein. Aus diesen verschiedenen FPR wird eine FPR für die Produktionsstätte, in denen die Produktionslinien aufgebaut sind, zusammengeführt und in K_{HW} gezeigt. Die FPR der Produktionsstätte wird folglich nicht im Produktionsprozess erfasst, sondern wird durch eine mathematische Weiterverarbeitung der FPR der Produktionslinien gebildet.

Die Erfassung der FPR ist seit langem in dem Produktionsprozess integriert: Die Produktionsteams tragen in einem Informationsverarbeitungssystem ein, ob ein gefertigtes intelligentes Produkt den Fertigungsendtest bestanden hat, und das Informationsverarbeitungssystem berechnet die FPR. Sowohl die Produktionsteams als auch das Management verwenden seit Jahren die definierten Produktionskennzahlen. Ihre Bedeutung und ihre Interpretation, also ihre Semantik, sind gut bekannt.

Es kann zusammengefasst werden, dass beim Kooperationspartner eine kennzahlenorientierte Gestaltung des Produktionsprozesses etabliert ist. Wie im folgenden Abschnitt aufgeführt wird, gilt dies nicht für die kennzahlenorientierte Gestaltung des Softwareentwicklungsprozesses.

1.3 Kennzahlenorientierte Gestaltung der Softwareentwicklung

Der Prozessablauf für die Entwicklung intelligenter Produkte, die sowohl die Hardware- als auch die Softwareentwicklung umfasst, ist in einem unternehmensweit gültigen Dokument beschrieben. Der Produktentwicklungsprozess basiert auf mehreren Meilensteinen, zu deren Erreichung bestimmte Kriterien zu erfüllen sind.

Die Überwachung der einzelnen Produktentwicklungsprojekte erfolgt mit Hilfe einer Projektmanagementdatenbank. In dieser werden Projektinformationen dokumentiert, wie zum Beispiel Soll-/Ist-Termine der Meilensteine, Plan-/Ist-Aufwände der Produktentwicklungsprojekte und der Status der zu jedem Meilenstein zu erledigenden Aufgaben.

Aus der Projektdatenbank werden Berichte generiert, die u.a. die Abweichungen zwischen Soll- und Ist-Terminen und zwischen Plan- und Ist-Aufwänden projektbezogen darstellen. Diese Berichte werden in regelmäßigen Terminen dem Management vorgestellt.

Das Management spiegelt diese Kennzahlen an den strategischen Zielen und bewirkt daraufhin strategische Maßnahmen. Ein Beispiel einer strategischen Maßnahme ist in der Erläuterung 1.1 beschrieben. Die Softwareteams setzen kontinuierlich Verbesserungsmaßnahmen um, zum Beispiel die Überarbeitung bestehender Programmierrichtlinien oder die Einführung regelmäßiger Code Reviews.

Softwarekennzahlen sind Gegenstand zahlreicher Veröffentlichungen [PP11, SSB10, Kas08, FP97]. Sie können mit Hilfe des Teufelsquadrats kategorisiert werden. Das Teufelsquadrat ist ein Grundmodell der Softwarewirtschaftlichkeit, wobei dessen vier Ecken die Leistungsziele hohe Qualität, hohe Quantität, geringe Entwicklungsdauer und geringe Kosten symbolisieren [Sne87].
Für jede dieser Ecken gibt es Softwarekennzahlen: Für die Quantität gibt es zum Beispiel Function Points [Alb79] oder Lines of Code [SSB10] und für die Qualität gibt es die Fehlerbehebungsrate [PP11] oder die Fehlerdichte [FP97]. Der Kooperationspartner ist bereits in der Lage, Softwarekennzahlen zu Kosten und Entwicklungsdauer zu erfassen, jedoch nicht für Quantität und Qualität. Da jedoch bereits heute der Softwareentwicklungsprozess durch strategische und KVP-Maßnahmen beeinflusst wird, werden für die Prozessgestaltung auch solche Softwarekennzahlen benötigt. Folgendes Beispiel der Einführung eines Werkzeugs zum Architekturmanagement soll der Veranschaulichung dienen: Die Softwareteams hatten analysiert, dass eine entworfene Softwarearchitektur oftmals nicht mit der Implementierung übereinstimmt. Sie kamen zu der Einschätzung, dass dies zu Qualitätsproblemen und zu Verzögerungen in der Freigabe einzelner Softwarefunktionen führte, ohne diese genau quantifizieren zu können. Daraufhin wurden kommerzielle Softwarewerkzeuge für das Architekturmanagement bewertet. Es wurde ein Werkzeug ausgewählt und die Thematik dem Management dargelegt. Dieses folgte der Einschätzung, dass ein derartiges Werkzeug notwendig sei und gab daraufhin die Geldmittel für die Anschaffung und die notwendigen Schulungsmaßnahmen frei. Da jedoch Softwarekennzahlen für die Quantität und die Qualität fehlen, ist es bislang nicht möglich, die Verringerung der Qualitätsprobleme oder die Erhöhung der Verfügbarkeit einzelner Softwarefunktionen zu den zugesagten Terminen zu quantifizieren.

Erläuterung 1.1: Fehlende Softwarekennzahlen beim Kooperationspartner

In diesem bislang praktizierten Vorgehen sind zwar Elemente einer kennzahlenorientierten Prozessgestaltung zu erkennen. Allerdings fehlen zwei wichtige Merkmale einer kennzahlenorientierten Prozessgestaltung des Softwareentwicklungsprozesses:

Zum einen fehlen wesentliche Softwarekennzahlen, die nach dem Stand der Technik für eine Messung des Softwareentwicklungsprozesses benötigt werden. Dies wird in der Erläuterung 1.1 detailliert begründet. Zum anderen fehlt dem Management das Verständnis für die Semantik von Softwarekennzahlen. Dies wurde in verschiedenen Gesprächen bestätigt, in denen zum Beispiel nach der Bedeutung einiger der in Erläuterung 1.1 aufgeführten Softwarekennzahlen gefragt wurde. Die Bedeutung ist dem Management nicht bekannt.

Aufgrund dieser fehlenden Merkmale sind sowohl das Management als auch die Softwareteams nur unzureichend in der Lage, den Softwareentwicklungsprozess zu gestalten. Sie können für die Bewertung der Zielerreichung strategischer und operativer Ziele lediglich die bisherigen Kennzahlen aus der Projektmanagementdatenbank verwenden. Dies reicht in Zukunft nicht mehr aus, da die Bedeutung des Softwareentwicklungsprozesses wächst. Wie bereits in [Deu14] dargestellt, wird der Softwareentwicklungsprozess zu einem zentralen Prozess in produzierenden Betrieben, von dem deren wirtschaftlicher Erfolg mehr und mehr abhängt.

1.4 Forschungsfrage

Der dargestellte Stand der kennzahlenorientierten Gestaltung des Produktionsprozesses und des Softwareentwicklungsprozesses ist spezifisch für einen produzierenden Betrieb. Um einschätzen zu können, inwiefern eine ähnliche Ausgangslage in anderen produzierenden Betrieben anzutreffen ist, wurde eine Umfrage mit Unterstützung der IHK Ostwestfalen zu Bielefeld durchgeführt. Es wurde ein Fragebogen zum Umgang mit Produktionskennzahlen und Softwarekennzahlen erstellt.

Der Fragebogen wurde im Dezember 2015 an die Mitglieder des IHK-Arbeitskreises „IT“ verteilt. Es gab neun Rückläufer. Die Ergebnisse der Umfrage sind Indikatoren für eine vergleichbare Situation der kennzahlenorientierten Prozessgestaltung des Produktions- und des Softwareentwicklungsprozesses in weiteren produzierenden Betrieben, wie die Fragen und Antworten in der Erläuterung 1.2 zeigen.

Frage: Arbeiten Sie im Rahmen ihrer Managementaufgaben mit Produktionskennzahlen, mit denen Sie die Effizienz und die Qualität der Produktionsprozesse in Ihrem Unternehmen beobachten und steuern (z.B. durch die Vorgabe und Überprüfung von Zielwerten)?
Antworten: Diese Frage beantworteten acht Befragte mit „Ja“, ein Befragter mit „Nein“. Die Mehrheit der Teilnehmer wendet demnach Produktionskennzahlen an.
Frage: Arbeiten Sie im Rahmen ihrer Managementaufgaben mit Softwarekennzahlen, mit denen Sie die Effizienz und die Qualität der Softwareentwicklungsprozesse in Ihrem Betrieb beobachten und steuern (z.B. durch die Vorgabe und Überprüfung von Zielwerten)?
Antworten: Diese Frage beantworteten zwei Befragte mit „Ja“, sechs Befragte mit „Nein“ und ein Befragter machte keine Angabe. Die Mehrheit der Teilnehmer verwendet demnach keine Softwarekennzahlen. Da sie keine Softwarekennzahlen verwenden und folglich deren Semantik nicht zu kennen brauchen, ist ihnen die Semantik von Softwarekennzahlen nicht bewusst.

Erläuterung 1.2: Teilergebnisse der IHK-Umfrage

Zur Herleitung des Forschungsbedarfes wird aufgrund der Beobachtungen beim Kooperationspartner und den Antworten in der IHK-Umfrage von folgender Ist- und Zielsituation in produzierenden Betrieben ausgegangen:

Ist-Situation: Die kennzahlenorientierte Gestaltung des Produktionsprozesses ist etabliert. Sowohl das Management als auch die Produktionsteams verwenden Produktionskennzahlen, deren Semantik sie kennen. Die kennzahlenorientierte Gestaltung des Softwareentwicklungsprozesses ist teilweise etabliert. Zwar wirken das Management und die Softwareteams auf den Softwareentwicklungsprozess ein, es werden jedoch wichtige Softwarekennzahlen nicht eingesetzt und folglich wird deren Erfassung nicht in der Prozessgestaltung berücksichtigt. Außerdem ist das Management nicht in der Lage, Softwarekennzahlen zu interpretieren, da es deren Semantik nicht kennt.

Zielsituation: Es ist sowohl die kennzahlenorientierte Gestaltung des Produktionsprozesses als auch die des Softwareentwicklungsprozesses etabliert. Das Management, die Produktionsteams und die Softwareteams kennen die Semantik der für ihren Verantwortungsbereich definierten Kennzahlen. Da die kennzahlenorientierte Gestaltung des Produktionsprozesses bereits etabliert ist, stellt sich die nachfolgende Forschungsfrage als Grundlage dieser Arbeit:

Wie kann der Softwareentwicklungsprozess in produzierenden Betrieben kennzahlenorientiert gestaltet werden?

Um die Zielsituation zu erreichen, sind die Kennzahlen für den Softwareentwicklungsprozess für das Management und die Softwareteams zielgruppenorientiert aufzubereiten. In [DEHW13] wird argumentiert, dass das in der Softwaredomäne vorhandene Wissen über die Softwaremessung „unbedingt zielgruppenorientiert“ zu übersetzen sei. Dies sei notwendig, um die Bewertung von Softwareentwicklungsprozessen in der Praxis zu verbessern. Vor diesem Hintergrund werden aus der Forschungsfrage drei Detailfragen abgeleitet, deren Lösungen zur Verbesserung der Bewertung von Softwareentwicklungsprozessen in produzierenden Betrieben beitragen sollen:

1. Für die Befriedigung der Informationsbedürfnisse des heutigen Managements zu dem Softwareentwicklungsprozess reicht die Verwendung von Softwarekennzahlen, die aus der Literatur bekannt sind, nicht aus, da das Management deren Semantik nicht kennt. Um dieses Problem zu lösen, können drei Handlungsoptionen in Betracht gezogen werden: Erstens könnte das Management das notwendige Wissen aufbauen. Zweitens könnte es Softwaremanager, die die Semantik von Softwarekennzahlen kennen, in den oberen Hierarchieebenen der produzierenden Betriebe etablieren. Drittens könnten bereits heute eingesetzte Produktionskennzahlen im Softwareentwicklungsprozess verwendet werden. In dem Fall müsste die semantische Äquivalenz einer Produktionskennzahl hergestellt werden. Damit ist gemeint, dass eine Produktionskennzahl ihre Bedeutung beibehält, auch wenn sie in der Softwaredomäne verwendet wird. Im letzteren Fall gäbe es zwei Ausprägungen dieser Produktionskennzahl: Die Produktionskennzahl, die die Ergebnisse der Produktion

der Hardware eines intelligenten Produktes anzeigt, und die Produktionskennzahl, die die Ergebnisse der Softwareentwicklung eines intelligenten Produktes anzeigt. Zur Unterscheidung dieser Ausprägungen werden die Begriffe „HW-Produktionskennzahl“ und „SW-Produktionskennzahl“ eingeführt. Die Motivation für die Bearbeitung der dritten Handlungsoption ist die vorhandene Kenntnis des Managements über die Semantik der HW-Produktionskennzahlen. Es wird angenommen, dass die Semantik der äquivalenten SW-Produktionskennzahlen vom Management in kurzer Zeit verinnerlicht werden kann. Konkrete Indikatoren für das Interesse des Managements an der Umsetzung der dritten Handlungsoption sind die Ergebnisse der IHK-Umfrage und ein Gespräch mit dem Geschäftsbereichsleiter des Kooperationspartners. In der IHK-Umfrage wurde die Frage gestellt: „Würde es Ihnen in Ihren Managementaufgaben helfen, Ihnen bekannte Produktionskennzahlen für das Beobachten der Effizienz und Qualität der Softwareentwicklungsprozesse anzuwenden (egal, ob Sie dies für machbar halten oder nicht)?“ Auf diese Frage antworteten sieben Befragte mit „Ja“ und zwei Befragte machten keine Angabe. Der Geschäftsbereichsleiter des Kooperationspartners beantwortet diese Frage mit „Ja“ und betonte explizit sein Interesse an dieser Handlungsoption. Um diese Handlungsoption zu realisieren, stellt sich die Detailfrage: *Wie können SW-Produktionskennzahlen, die die Semantik der äquivalenten HW-Produktionskennzahlen beibehalten, bestimmt werden?*

2. SW-Produktionskennzahlen adressieren das Management und dienen der Überprüfung der strategischen Ziele. Die Softwareteams wie auch die Produktionsteams definieren hingegen die operativen Ziele. Für die Überprüfung der Zielerreichung der operativen Ziele sind Softwarekennzahlen zu bestimmen. Dafür können bekannte Methoden wie zum Beispiel die Goal-Question-Metric-Methode (GQM-Methode) [BW84] angewendet werden. Sobald Softwarekennzahlen als auch SW-Produktionskennzahlen bestimmt wurden, ist der Softwareentwicklungsprozess entsprechend anzupassen. Um die definierten Kennzahlen erfassen zu können, stellt sich die Detailfrage: *Wie sollte der Softwareentwicklungsprozess aufgebaut sein, damit die definierten SW-Produktionskennzahlen und Softwarekennzahlen erfasst werden können?*

3. Die kennzahlenorientierte Prozessgestaltung beinhaltet ein Informationsverarbeitungssystem, das die SW-Produktionskennzahlen und Softwarekennzahlen erfasst und verarbeitet. Die Anwender des Informationsverarbeitungssystems sind das Management und die Softwareteams. Um alle Kennzahlen dem Management und den Softwareteams zuzuführen, stellt sich die Detailfrage: *Wie sollte ein Informationsverarbeitungssystem aufgebaut sein, das SW-Produktionskennzahlen und Softwarekennzahlen erfassen und verarbeiten kann?*

Die Messbarkeit von Softwareentwicklungsprozessen in produzierenden Betrieben ist Thema in mehreren Publikationen. In [SMKN11] wird der Aufbau eines Softwarekennzahlensystems bei der Fa. Ericsson beschrieben. Jedoch ist es das Ziel, die Informationsbedürfnisse von Personen, die für die Softwarequalität verantwortlich sind, zu befriedigen. Daher waren Antworten auf eine mit der Detailfrage 1 vergleichbaren Frage, deren Lösung die Informationsbedürfnisse des Managements befriedigen soll, nicht das Ziel dieser Publikation. In [Kil01] wird das interne Programm der Softwaremessung bei Nokia beschrieben. Zwar werden darin Softwarekennzahlen aus Managementzielen bestimmt, allerdings ist das Management nur für die Entwicklung zuständig, nicht für die Produktion, und es kann nicht erkannt werden, wie die Erfassung von Softwarekennzahlen im Softwareentwicklungsprozess integriert ist. In [ED07] wird neben den Grundlageninhalten zur Softwaremessung auch die Einführung von Function Points bei der Fa. Bosch beschrieben. Function Points, die in Abschnitt 2.1.2.1 vertiefend vorgestellt werden, sind Softwarekennzahlen für die Quantitätsmessung. In der Fallstudie bleibt allerdings offen, wer der Adressat der Function Points ist und ob das Management die Semantik von Function Points kennt, sofern es der Adressat ist.

1.5 Methodische Vorgehensweise

Das Vorgehen dieser Arbeit gliedert sich in die Erstellung der konzeptionellen Grundlagen für die Beantwortung der Forschungsfragen, in eine praktische Evaluierung und in eine Theoriebildung. Es orientiert sich an der methodischen Vorgehensweise des *Design Research* (DR). DR analysiert die Anwendung von entworfenen IT-Artefakten, um Informationssysteme zu verstehen, zu erklären und zu verbessern [IV09]. Trotz der vorhandenen einschlägigen Literatur, zum Beispiel [Ven06, PHBV08, SB12], gibt es jedoch weder eine allgemein gültige DR-Definition noch ein allgemein anerkanntes DR-Modell [IV09, Gol13]. Diese Arbeit orientiert sich daher an den Ausführungen in [Gol13]: DR besteht aus zwei Aktivitäten: dem *Entwurf eines oder mehrerer IT-Artefakte* und der *Theoriebildung*. Die IT-Artefakte, in dieser Arbeit sind es zum Beispiel die Lösungen zu den Detailfragen, haben einen lokalen praktischen Bezug, beispielsweise im konkreten Umfeld eines produzierenden Betriebes. Die lokalen Ergebnisse, die im Entwurf der IT-Artefakte gewonnen werden, werden der Theoriebildung mit dem Ziel zugeführt, daraus allgemein anwendbare Ergebnisse zu bilden. Der Entwurf eines IT-Artefakts setzt sich aus den Phasen *Problemanalyse*, *Erstellen* und *Evaluierung* zusammen. Die Phasen *Erstellen* und *Evaluierung* können mehrfach durchlaufen werden, was in dieser Arbeit auch erfolgte. Abbildung 1.4 zeigt die schrittweise methodische Vorgehensweise dieser Arbeit und die Zuordnung dieser zu den Phasen des *Design Research*.

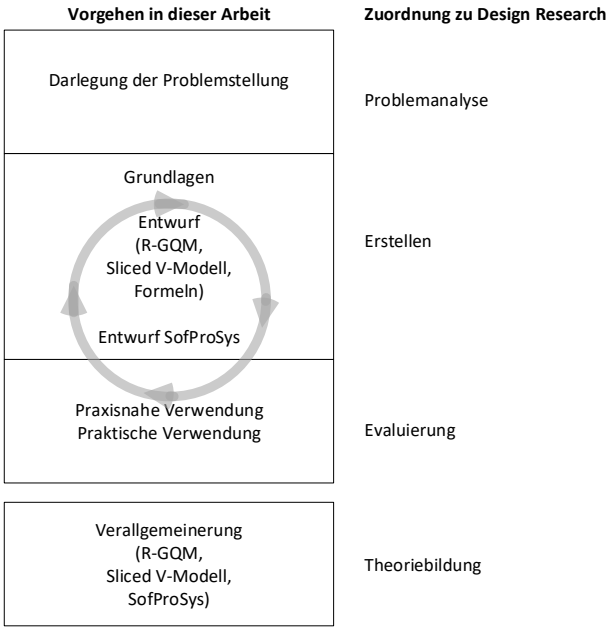


Abbildung 1.4: Vorgehen der Arbeit und Zuordnung zu den DR-Phasen

In der DR-Entwurfsphase *Problemanalyse* wird die Situation beim Kooperationspartner dargestellt und zu einer Problemstellung zusammengefasst. Mit Hilfe der IHK-Umfrage wird die Gültigkeit der Problemstellung in anderen produzierenden Betriebe bewertet.

Die DR-Entwurfsphase *Erstellen* widmet sich zunächst den für diese Arbeit relevanten Grundlagen. Darauf folgt der konzeptionelle Entwurf für die Lösungen zu den Detailfragen: Für die Beantwortung der ersten Detailfrage wird eine auf der GQM-Methode [BW84] aufbauende neue Methode entworfen. Diese Methode wird *Reversed-Goal-Question-Metric-Methode* (RGQM-Methode) genannt. Mit Hilfe der RGQM-Methode können HW-Produktionskennzahlen in der Softwareentwicklung angewendet werden. Im nächsten Schritt werden konkrete HW-Produktionskennzahlen beim Kooperationspartner ausgewählt und mit Hilfe der RGQM-Methode in SW-Produktionskennzahlen transferiert. Um die SW-Produktionskennzahlen erfassen zu können, muss der zugrundeliegende Softwareentwicklungsprozess bestimmte Anforderungen erfüllen. Diese Anforderungen werden formuliert. Weitere Anforderungen ergeben sich aus der Bestimmung von Softwarekennzahlen auf Grundlage der operativen Ziele. Für die Bestimmung der Softwarekennzahlen wird die etablierte GQM-Methode verwendet.

Für die Beantwortung der zweiten Detailfrage wird das sogenannte *Sliced V-Modell* entworfen. Das Sliced V-Modell ist ein Datenmodell, an dessen Gestaltung der Autor der Arbeit maßgeblich beteiligt war. Es beschreibt die im Softwareentwicklungsprozess des Kooperationspartners genutzten Softwareartefakte und deren Zusammenhänge. Im Entwurf des Sliced V-Modells werden die bei der Bestimmung der Kennzahlen formulierten Anforderungen an den Softwareentwicklungsprozess berücksichtigt. Dem Entwurf des

Sliced V-Modells folgen die Berechnungsgrundlagen, das heißt die Formeln für die SW-Produktionskennzahlen und für die Softwarekennzahlen.

Abschließend wird in der DR-Entwurfsphase *Erstellen* ein Prototyp eines Informationsverarbeitungssystems entwickelt, mit dessen Hilfe die dritte Detailfrage beantwortet werden soll. Dieser Prototyp wird *SofProSys* genannt. Dies steht für: Software-Produktionskennzahlen-Informationsverarbeitungs-System.

Abbildung 1.5 zeigt die Zielsituation, die in dieser Arbeit in der Softwaredomäne erreicht werden soll und die sich an der existierenden Situation im Produktionsprozess orientiert (vgl. Abbildung 1.3). Wie in Abbildung 1.2 zeigen die dünnen Striche die Abläufe in der Gestaltungsphase, die dicken Striche die Abläufe in der Erfassungsphase. Das Management legt strategische Ziele fest. Die operativen Ziele werden aus den strategischen Zielen abgeleitet. An der Bestimmung der operativen Ziele sind die Softwareteams aktiv beteiligt. K_{SW} ist die Menge der SW-Produktionskennzahlen, die aus K_{HW}, also aus der bereits existierende Menge an HW-Produktionskennzahlen, in die Softwaredomäne transferiert wird. K_{HW} und folglich K_{SW} werden aus den strategischen Zielen bestimmt und zeigen deren Zielerreichung an. K_{SW} bedient die Informationsbedürfnisse des Managements. K_S ist die Menge der Softwarekennzahlen, die aus den operativen Zielen bestimmt wird und deren Zielerreichung anzeigt. K_S wird von den Softwareteams verwendet.

Folglich verwenden die Softwareteams und das Management unterschiedliche Kennzahlen. Dies ist ein Unterschied zu der im Abschnitt 1.2 erläuterten Situation im Produktionsprozess. Daher müssen sowohl die Erfassung von SW-Produktionskennzahlen als auch die Erfassung von Softwarekennzahlen in der kennzahlenorientierten Gestaltung des Softwareentwicklungsprozesses berücksichtigt werden. Es werden sowohl K_{SW} als auch K_S im Softwareentwicklungsprozess erfasst, wobei auch eine Softwarekennzahl in K_S für die Berechnung einer SW-Produktionskennzahl in K_{SW} verwendet werden könnte.

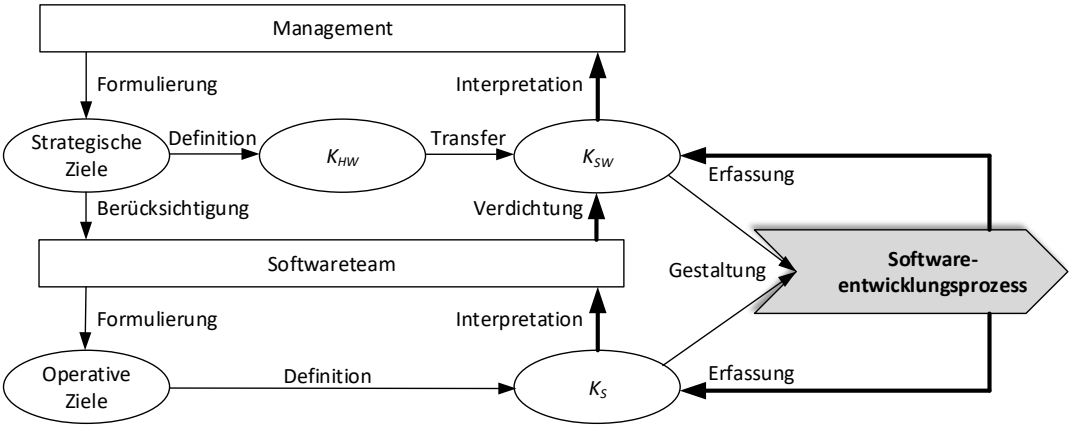


Abbildung 1.5: Zielsituation dieser Arbeit

In der DR-Entwurfsphase *Evaluierung* wird die RGQM-Methode an fünf beim Kooperationspartner eingesetzten HW-Produktionskennzahlen angewendet, aus denen SW-Produktionskennzahlen gebildet werden. Die semantische Äquivalenz der beiden Ausprägungen einer Produktionskennzahl wird bewertet. Die Evaluierung des Sliced V-Modells erfolgt in drei Schritten: durch die Betrachtung der Erfüllung der daran formulierten Anforderungen, durch die Bewertung, ob die Berechnungsgrundlagen für alle Kennzahlen erstellt werden können und durch die Implementierung von *SofProSys* als Nachweis, dass alle Kennzahlen im Sliced V-Modell erfasst werden können. Die Evaluierung von *SofProSys* erfolgt, indem der Prototyp sowohl praxisnah in einem Testumfeld als auch praktisch beim Kooperationspartner verwendet wird. In der praktischen Verwendung werden reale Softwareentwicklungsprozesse untersucht.

Die abschließende Evaluierung, ob die Ergebnisse dieser Arbeit eine Lösung für die Forschungsfrage darstellen, erfolgt zum einen durch den Geschäftsbereichsleiter des Kooperationspartners. Ihm werden die SW-Produktionskennzahlen der realen Softwareentwicklungsprozesse dargestellt und sie werden mit ihm diskutiert. Zum anderen werden die Ergebnisse der Arbeit in der DR-Aktivität *Theoriebildung* verallgemeinert. Es wird erläutert, wie produzierende Betriebe die Ergebnisse dieser Arbeit in ihr betriebliches Umfeld übertragen können und dass diese Ergebnisse somit die Forschungsfrage beantworten.

1.6 Aufbau der Arbeit

Der Aufbau dieser Arbeit orientiert sich an den Aktivitäten des *Design Research* (vgl. Abbildung 1.6). Wie bereits erwähnt, gab es mehrere DR-Iterationen. Sie werden jedoch weder als eigene Kapitel ausgeführt noch explizit in dieser Arbeit beschrieben.

Die Kapitel 1 und 2 widmen sich der DR-Entwurfsphase *Problemanalyse* und den für diese Arbeit relevanten Grundlagen.

Die Kapitel 3, 4, 5, 6 und 7 umfassen die DR-Entwurfsphase *Erstellen* und die DR-Entwurfsphase *Evaluierung*. Es wird die RGQM-Methode entwickelt und es werden betriebsspezifische HW-Produktionskennzahlen des Kooperationspartners ausgewählt, in äquivalente SW-Produktionskennzahlen transferiert und somit die RGQM-Methode evaluiert. Mit Hilfe der GQM-Methode werden Softwarekennzahlen definiert. Es wird das Sliced V-Modell unter Berücksichtigung von Anforderungen, die während der Bestimmung der Kennzahlen formuliert werden, entworfen. Für alle Kennzahlen werden die jeweiligen Berechnungsgrundlagen ermittelt. Es wird *SofProSys* entwickelt und sowohl praxisnah als auch praktisch angewendet. Mit der praktischen Anwendung wird evaluiert, inwiefern die Ergebnisse der Arbeit eine Lösung für die Forschungsfrage darstellen.

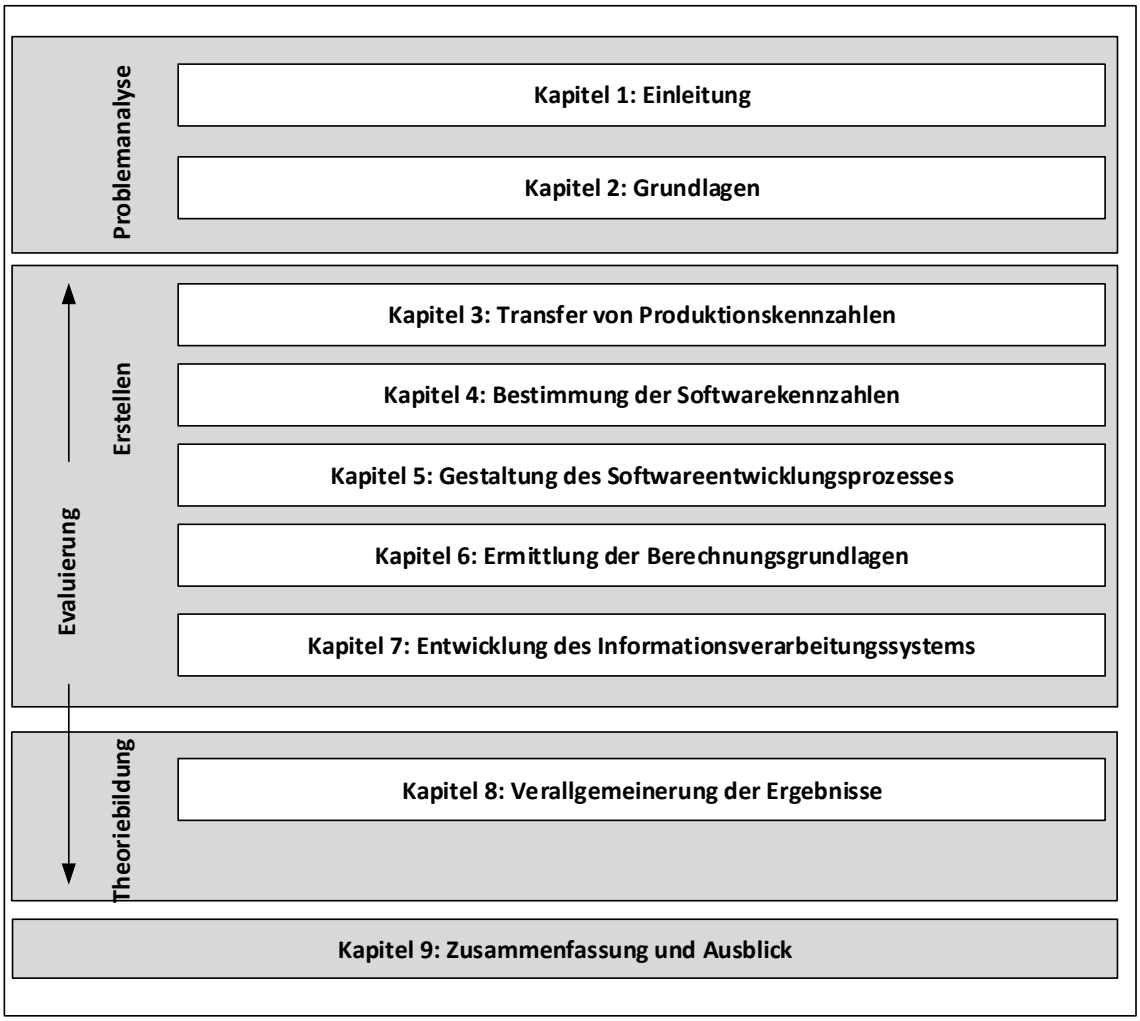


Abbildung 1.6: Aufbau dieser Arbeit

Kapitel 8 widmet sich der DR-Aktivität *Theoriebildung* und verallgemeinert die Ergebnisse der Arbeit. Da die Ergebnisse in nur einem produzierenden Betrieb evaluiert werden, wird hier dargelegt, dass sie sich prinzipiell in andere produzierende Betriebe übertragen lassen.

Das abschließende Kapitel 9, das keiner *Design Research* Aktivität zugeordnet ist, fasst die Inhalte der Arbeit zusammen, erläutert weitere Aktivitäten für Überführung der Ergebnisse der Arbeit in die betriebliche Praxis des Kooperationspartners und skizziert mögliche auf dieser Arbeit aufbauende Forschungsvorhaben.

1.7 Vorveröffentlichungen

Auszüge aus dieser Arbeit wurden in folgenden Publikationen veröffentlicht. Die Publikationen sind chronologisch aufgeführt.

Hauptbezug zu	**Veröffentlichung**
Kap. 5, 6	[Deu12] Deuter, A.: Messung der Software-Produktivität in einem Work Item-basierten V-Modell, In: *Metrikon 2012 - Praxis der Software-Messung*, Shaker Verlag, Aachen, 2012, S. 69-84
Kap. 5	[Deu13] Deuter, A.: Slicing the V-model – Reduced effort, higher flexibility, In: *International Conference on Global Software Engineering (ICGSE)*, 2013, S. 1-10
Kap. 6	[DE14] Deuter, A.; Engels, G.: Measuring the Software Size of Sliced V-model Projects, In: *Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)*, 2014, S. 233-242
Kap. 1	[Deu14] Deuter, A.: Software wird auch im Maschinenbau zur Kernkompetenz, In: *IEE Elektrische Automatisierung+Antriebstechnik* 10 (2014), S. 16-18
Kap. 6	[DK15] Deuter, A.; Koch, H.-J.: Applying Manufacturing Performance Figures to Measure Software Development Excellence, In: *Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)*, 2015, S. 62-77
Kap. 3	[DD15] Deuter, A.; Dreyer, J.: Reversed-GQM: Ein Ansatz zur Wiederverwendung von Kennzahlen, In: *Metrikon 2015 - Praxis der Software-Messung*, Shaker Verlag, Aachen, 2015, S. 3-14
Kap. 3	[Deu16] Deuter, A.: *Software measurement in the context of Industry 4.0*, Workshop in Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2016

Tabelle 1.2: Vorveröffentlichungen zu dieser Arbeit

Kapitel 2

Grundlagen

Dieses Kapitel widmet sich den Grundlagen dieser Arbeit. Für die kennzahlenorientierte Prozessgestaltung wird ein Verständnis von Kennzahlen benötigt. Daher werden in diesem Kapitel zunächst grundlegende Merkmale von Kennzahlen erläutert. Da die Forschungsfrage bzw. die daraus abgeleiteten Detailfragen sowohl Softwarekennzahlen als auch Produktionskennzahlen adressieren, erfolgt eine vertiefende Darstellung dieser beiden Kennzahlenausprägungen. In diesem Zusammenhang werden ebenfalls bekannte Grundsätze für die Gestaltung von Informationsverarbeitungssystemen aufgeführt, da diese für die Lösung der dritten Detailfrage zu beachten sind.

Da die erste Detailfrage einen Transfer von Kennzahlen der Produktionsdomäne in die Softwaredomäne adressiert, wird zu der in der einschlägigen Literatur geführten Diskussion zur Machbarkeit des Transfers von Methoden der Produktionsdomäne in die Softwaredomäne Stellung genommen. Anschließend werden bekannte Beispiele für Kennzahlentransfers genannt und es wird dargelegt, dass diese als Lösung für die erste Detailfrage nicht geeignet sind. Daher werden existierende Methoden bzw. Prozessbeschreibungen, mit denen Kennzahlen für betriebliche Prozesse methodisch bestimmt werden können, dahingehend bewertet, ob sie als Ausgangspunkt für eine neu zu entwickelnde Kennzahlentransfer-Methode geeignet sind.

Abschließend widmet sich dieses Kapitel den Grundlagen, die für die zweite Detailfrage relevant sind: Es werden die wesentlichen Merkmale des Softwareentwicklungsprozesses, der das Ziel der kennzahlenorientierten Prozessgestaltung ist, erläutert und es wird der Softwareentwicklungsprozess des Kooperationspartners dargestellt.

2.1 Kennzahlen

2.1.1 Begriffe und Eigenschaften

Kennzahlen sowie deren Anwendung, Auswahlkriterien und Gestaltungsgrundsätze sind u.a. in [HP07, Ben07, Bro97, Pet08, Hor11, BB10, Küt10] umfangreich beschrieben. Die folgenden Ausführungen erläutern ein für diese Arbeit notwendiges Grundverständnis zu Kennzahlen.

Kennzahlen sind Zahlen, die in verdichteter Form quantitativ oder qualitativ messbare Sachverhalte wiedergeben. Es wird zwischen absoluten Kennzahlen und Verhältniskennzahlen unterschieden (Abbildung 2.1) [Ben07]. Absolute Kennzahlen beschreiben einen Sachverhalt, ohne in Relation zu einer anderen Größe gesetzt zu werden. Sie können gemäß [Ben07] in Einzelzahl, Summe, Differenz oder Mittelwert eingeteilt werden. Addition, Subtraktion und Mittelwertbildung sind nur bei Werten gleicher Einheit möglich.

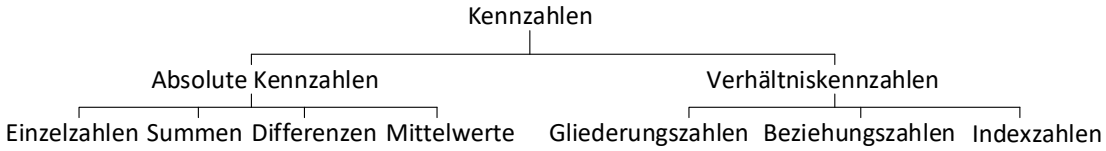


Abbildung 2.1: Arten von Kennzahlen [Ben07]

Verhältniskennzahlen zeigen das Verhältnis von mehreren absoluten Kennzahlen an. Sie können in Gliederungskennzahlen (Anteil einer Größe an einer Gesamtmenge, zum Beispiel Entwicklungskosten zu Gesamtkosten), Beziehungskennzahlen (Verhältnis zweier unterschiedlicher Größen, zum Beispiel Gewinn zu Eigenkapital) und Indexkennzahlen (zeitliche Entwicklung von Größen, zum Beispiel Entwicklungskosten im Jahr 2014 zu Entwicklungskosten im Jahr 2015) eingeteilt werden.

Kennzahlen dienen der Planung, Steuerung und Kontrolle von betrieblichen Aktivitäten (Soll-Ist-Vergleich), lassen Trends erkennen (Ist-Kennwerte im Zeitvergleich) und können für ein internes oder externes Benchmarking herangezogen werden.

Kennzahlen werden von betrieblichen Entscheidungsträgern aller Hierarchieebenen genutzt. Die Entscheidungsträger legen beispielsweise die Soll-Werte einzelner Kennzahlen fest. Weichen die Ist-Werte einzelner Kennzahlen von den Soll-Werten ab, werden Maßnahmen abgeleitet, um die Ist-Werte zu verbessern. Die Wirksamkeit der Maßnahmen wird mit Hilfe der Ist-Werte überprüft. Kennzahlen haben somit eine Informationsfunktion. Kennzahlen werden entsprechend den Informationsbedürfnissen der Adressaten in den verschiedenen Hierarchieebenen aufbereitet. Abbildung 1.3 zeigt ein Beispiel: Darin werden die Produktionskennzahlen beim Kooperationspartner entsprechend den Informations-

bedürfnissen des Managements und der Produktionsteams präsentiert bzw. verdichtet. Sofern die eingesetzten Kennzahlen das Management adressieren, sollte deren Verwendung durch das Management bestätigt werden [BB10].

Um eine Kennzahl eindeutig zu beschreiben, werden alle zu ihr gehörenden Informationen in einem Dokument hinterlegt. Dieses kann ein Kennzahlenstammblatt [Ben07], ein Kennzahlensteckbrief [Küt10] oder ein Kennzahlenformular [HP07] sein. Die Informationen in diesen Dokumenten zeigen die Bedeutung der Kennzahl, also deren Semantik, auf. Um den Begriff „Semantik einer Kennzahl“ für diese Arbeit zu konkretisieren, wird festgelegt, dass die nachfolgenden Informationen die Semantik einer Kennzahl umfassen. Diese Konkretisierung wird benötigt, da für die *Semantik einer Kennzahl* keine allgemein anerkannte Definition bekannt ist. Die benötigten Informationen sind:

- Name
- Maßeinheit
- Wertebereich
- Idealwert
- Möglichkeit der Festlegung von Soll-Werten
- Frage
- Ziel
- Interpretation

Diese Informationsinhalte werden an dem Beispiel der bereits in Abschnitt 1.2 eingeführten First Pass Rate (Name) erklärt: Die FPR ist eine Verhältnisgröße, die in Prozent (%) angegeben wird. Der Wertebereich liegt zwischen 0 % und 100 %, wobei der Idealwert 100 % ist. Ein typischer Soll-Wert, der für eine Produktionslinie individuell festgelegt werden kann, ist 98 %. Die FPR beantwortet die Frage: „Wie ist das Verhältnis von gefertigten Produkten, die fehlerfrei getestet wurden, zu allen gefertigten Produkten?“ Die FPR ist dem strategischen Ziel „Hohe Fertigungsqualität“ zugeordnet. Diese abstrakte Zielformulierung präsentiert die grundlegende Unternehmensstrategie des Kooperationspartners, der sich als Qualitätsanbieter in seinem Markt versteht. Das Ziel ist folglich weder quantifiziert noch terminiert. Der Begriff „Ziel“ bezieht sich in dieser Arbeit auf solche abstrakten Ziele. Die Quantifizierung des Ziels und dessen Terminierung ergibt sich beim Kooperationspartner durch die Formulierung von Soll-Werten für einen festgelegten Zeitraum, wie zum Beispiel: „Für die Werkstattfertigung A ist in 2016 eine durchschnittliche FPR von 96 % zu erreichen.“ Dabei wird die FPR wie folgt interpretiert:

- Im Gegensatz zu rein mechanischen Produkten, die lediglich stichprobenartig geprüft werden, werden alle intelligenten Produkte einem Fertigungsendtest unterzogen. Maximal ist zwar eine FPR von 100 % möglich, typische Ist-Werte liegen jedoch zwischen 95 % und 98 %. Da es sich um hochpreisige intelligente Produkte handelt, werden diejenigen Produkte, die den Fertigungsendtest nicht bestehen, inspiziert und wenn möglich nachbearbeitet. Dies verursacht Nacharbeitskosten. Das Management fordert, die Nacharbeitskosten möglichst gering zu halten und legt für jede Produktionsstätte durchschnittliche jährliche Soll-Werte fest. Die Produktionsteams bestimmen daraus die jährlichen Soll-Werte für die einzelnen Produktionslinien in der Produktionsstätte. Entsprechen die Ist-Werte den Soll-Werten, werden keine Maßnahmen eingeleitet. Sind die Ist-Werte geringer als die Soll-Werte, untersuchen das Management bzw. die Produktionsteams die Ursachen und leiten Maßnahmen ein. Ein Beispiel einer Maßnahme ist bereits in Tabelle 1.1 aufgeführt. Sollten die Ist-Werte für eine Produktionsstätte nicht den Soll-Werten entsprechen, entscheidet das Management gemeinsam mit den Produktionsteams über die Umsetzung strategischer Maßnahmen wie zum Beispiel die Beschaffung von neuen Maschinen oder Mitarbeiterqualifizierungsmaßnahmen.

Wie aus diesem Beispiel zu erkennen ist, beschreibt die Interpretation, wie die Adressaten einer Kennzahl auf die konkret erfassten Ist-Werte reagieren. Die Interpretation ist folglich eine Handlungsbeschreibung des Adressaten. Da diese Handlungsbeschreibung während der Gestaltung der Kennzahl nicht bzw. nur schwierig formulierbar ist, ist sie im operativen Prozess zu beobachten und zu formulieren. Dabei kann die Interpretation personenspezifisch sein: So könnte zum Beispiel ein Manager auf eine Soll-Ist-Abweichung der FPR mit der Einleitung einer Maßnahme reagieren, ein anderer Manager hingegen entscheidet sich für eine weitere Beobachtung der Soll-Werte, ohne eine sofortige Maßnahme einzuleiten. Die Interpretation ist folglich ein nicht eineindeutig beschreibbarer Informationsinhalt einer Kennzahl.

Eine wichtige Rolle für die Erfassung und Verarbeitung von Kennzahlen spielen Informationsverarbeitungssysteme. Dies sind IT-Systeme, die Daten aus einem Prozess ermitteln, zu Kennzahlen verarbeiten und grafisch darstellen [Ben07]. Voraussetzung für die IT-basierte Kennzahlenerfassung ist die Erreichbarkeit der Datenquellen der Kennzahlen über ein IT-Netzwerk. Informationsverarbeitungssysteme reduzieren in der Regel die Kosten der Datenerfassung, -verarbeitung und -auswertung. Außerdem erhöhen sie die Objektivität der dargestellten Kennzahlen, da diese sonst unbewusst oder bewusst manipuliert werden können [Ben07]. Aus diesen Gründen werden bereits heute die Produktionskennzahlen beim Kooperationspartner mit einem Informationsverarbeitungssystem erfasst.

Bei der Gestaltung von Informationsverarbeitungssystemen sind verschiedene Grundsätze zu beachten. In [HP07] werden folgende Gestaltungsgrundsätze empfohlen:

- Hohe Validität: Die Informationsverarbeitungssysteme müssen korrekt messen.
- Berücksichtigung der wesentlichen Kennzahlen.
- Einfluss der Entscheidungsträger: Die in einem Informationsverarbeitungssystem enthaltenen Kennzahlen sollten unmittelbar von Entscheidungsträgern beeinflussbar sein, das heißt die Entscheidungsträger sollten die notwendigen Befugnisse und Kompetenzen haben, Maßnahmen, die die Ist-Werte der Kennzahlen ändern, einzuleiten.
- Ausgewogene Ausgestaltung hinsichtlich der Menge und des Zeithorizonts der Kennzahlen.
- Festlegung von Soll-Werten, um Ist-Werte interpretieren zu können.
- Ausgestaltung unter Kosten-Nutzen-Bedingungen: Der Nutzen des Informationsverarbeitungssystems rechtfertigt die Kosten für dessen Entwicklung und dessen Betrieb.
- Eindeutige Darstellung der Kennzahlen, um Fehlinterpretationen zu vermeiden. Mit Fehlinterpretation ist eine grundsätzlich falsche Interpretation, nicht die oben erläuterte personenspezifische Interpretation gemeint.

Vergleichbare, Informationsverarbeitungssysteme betreffende Gestaltungsgrundsätze werden zum Beispiel in [Bal97, Küt10] formuliert. Da diese grundsätzlich den oben aufgeführten Gestaltungsgrundsätzen ähneln, werden die Gestaltungsgrundsätze aus [HP07] für den später folgenden Entwurf des Informationsverarbeitungssystems als geeignet angesehen und daher berücksichtigt.

2.1.2 Softwarekennzahlen

Im weiteren Verlauf der Arbeit werden die Begriffe „Softwareprodukt“, „Softwareversion“, „Softwarefunktion“ und „Softwaredokumentation“ verwendet. Ein Softwareprodukt im Kontext dieser Arbeit ist eine in einem intelligenten Produkt eingebettete Software oder eine zu einem intelligenten Produkt gehörende Softwareanwendung, zum Beispiel eine App. Eine Softwareversion ist ein definierter Entwicklungsstand eines Softwareproduktes. Der definierte Entwicklungsstand enthält neue Softwarefunktionen. Eine Softwarefunktion ist ein spezifiziertes Verhalten des Softwareproduktes.

Allerdings ist das Softwareprodukt bzw. eine Softwareversion nicht das einzige Ergebnis des Softwareentwicklungsprozesses: In [HKLR84] wird der Begriff „Software“ definiert „als Menge von Programmen oder Daten zusammen mit begleitenden Dokumenten, die für ihre Anwendung notwendig oder hilfreich sind“. Die Bedeutung der Dokumentation wächst mit der zunehmenden Größe von Softwareprodukten, da sie deren Verständnis und deren Weiterentwicklung dient [MD06]. Der Aufwand für die Erstellung der Dokumentation variiert und kann in sicherheitsgerichteten Entwicklungen bis zu 50 % des Gesamtprojektaufwands betragen [MSH+12]. Für die Erläuterung des Begriffs „sicherheitsgerichtet“ wird auf den Abschnitt 2.5.2.1 verwiesen. Folglich sind Dokumente wie beispielsweise die Anforderungs- oder die Testspezifikationen ebenfalls wichtige Ergebnisse des Softwareentwicklungsprozesses. In dieser Arbeit sind mit dem Begriff „Softwaredokumentation“ solche Dokumente gemeint.

Mit Hilfe von Softwarekennzahlen wird der Prozess der Erstellung von Softwareprodukten, Softwareversionen und deren Dokumenten sowie deren quantitative und qualitative Eigenschaften gemessen. Softwarekennzahlen können verschiedenartig kategorisiert werden: In [Kan03] wird zwischen Produkt-, Prozess- und Projektkennzahlen unterschieden. In [SSB10] wird davon ausgegangen, dass Software eine multidimensionale Substanz ist, die in drei Dimensionen messbar ist: Größe, Komplexität und Qualität.

Eine weitere Form der Kategorisierung zeigt das sogenannte Teufelsquadrat, das als „ein Grundmodell der Softwarewirtschaftlichkeit mit vier Ecken“ bezeichnet wird [Sne87]. Die Ecken symbolisieren das Spannungsfeld, in dem sich Softwareentwicklungsprozesse bewegen: Sie sollen eine hohe Qualität und hohe Quantität bei geringer Entwicklungsdauer und geringen Kosten erreichen (Abbildung 2.2).

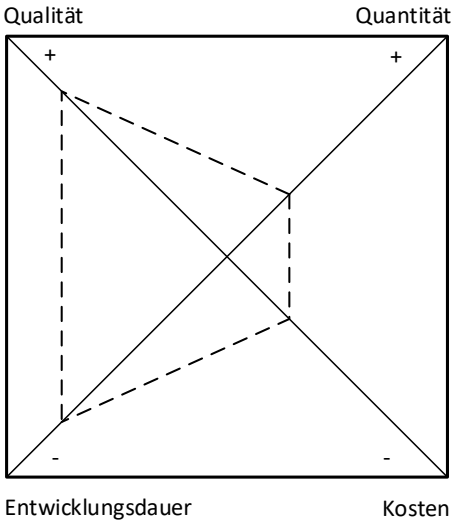


Abbildung 2.2: Teufelsquadrat nach [Sne87]

Das Trapez im Inneren des Quadrats symbolisiert die Wirtschaftlichkeit. Bei Erreichen einer hohen Qualität liegt der Schnittpunkt des Trapezes und der Diagonalen „Qualität-Kosten“ links oben. Bei hohen Kosten befindet sich der Schnittpunkt des Trapezes und dieser Diagonalen im inneren Bereich des Teufelsquadrats und so fort. Abbildung 2.2 zeigt symbolisch einen Softwareentwicklungsprozess, für den eine hohe Qualität, mittlere bzw. geringe Quantität, kurze Entwicklungsdauer und mittlere bzw. hohe Kosten erreicht wurden. Die Attribute „hoch“, „mittel“ etc. sind für einen konkreten Softwareentwicklungsprozess mit Softwarekennzahlen zu quantifizieren.

Im weiteren Verlauf dieser Arbeit werden die Softwarekennzahlen gemäß dem Teufelsquadrat kategorisiert. In den folgenden Abschnitten werden bekannte Softwarekennzahlen erläutert, die für die Quantifizierung der Kategorien des Teufelsquadrats geeignet sind. Alle Softwarekennzahlen in den einzelnen Kategorien sind den absoluten Kennzahlen zuzuordnen.

Zweck dieser Ausführungen ist es, mögliche Softwarekennzahlen für die Menge K_S zu identifizieren, mit denen die Zielerreichung der operativen Ziele des Softwareentwicklungsprozesses beim Kooperationspartner überprüft werden kann (vgl. Abbildung 1.5). In den Erläuterungen wird jeweils bewertet, ob die jeweilige Softwarekennzahl durch ein Informationsverarbeitungssystem erfasst werden kann. Nur solche Softwarekennzahlen sollen für die später folgende Gestaltung des Softwareentwicklungsprozesses beim Kooperationspartner berücksichtigt werden.

2.1.2.1 Quantität

Softwarequantitätskennzahlen, die sich auf ein Softwareprodukt bzw. eine Softwareversion beziehen, können in funktionale und in physikalische Kennzahlen unterschieden werden [BGA14]. Funktionale Kennzahlen zeigen den Funktionsumfang des Softwareproduktes an, wohingegen sich physikalische Kennzahlen auf den inneren Aufbau des Softwareproduktes beziehen. Physikalische Kennzahlen können weiter untergliedert werden und zwar in Quelltextkennzahlen, zum Beispiel Lines of Code, und in Prozesskennzahlen, zum Beispiel die Anzahl geänderter Quelltextzeilen. Allerdings werden mit Prozesskennzahlen, wie sie in [RD13] definiert sind, auch weitere Bearbeitungsaktivitäten, zum Beispiel Anzahl von Commits, gemessen. Solche Kennzahlen können nicht den Softwarequantitätskennzahlen zugeordnet werden. Daher wird der Begriff „Prozesskennzahl“ für diese Arbeit eingegrenzt: Es sind nur Kennzahlen gemeint, die Aussagen über die Änderungen des Quelltextes ermöglichen.

Für Softwarequantitätskennzahlen, die sich auf die Softwaredokumentation beziehen, sind keine expliziten Bezeichnungen bekannt. Sie werden im Folgenden „Dokumentationskenn-

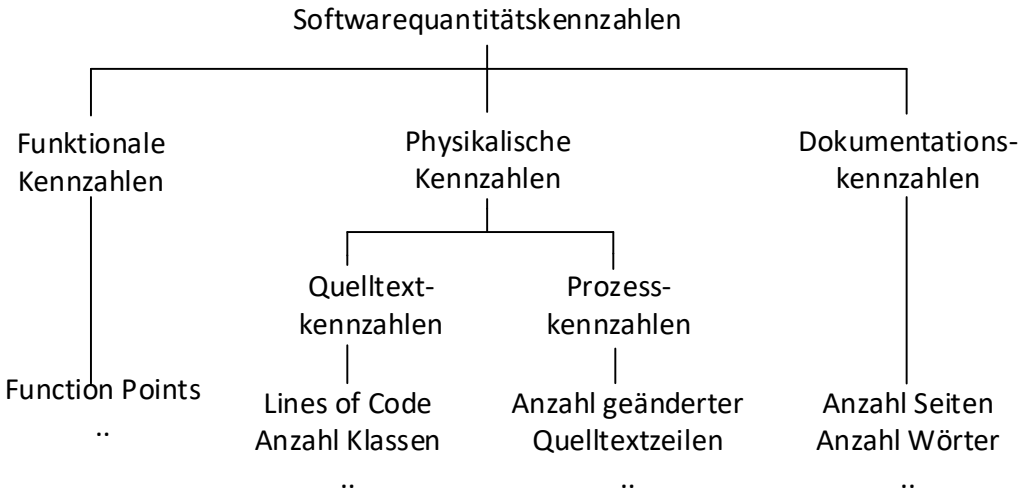


Abbildung 2.3: Kategorien von Softwarequantitätskennzahlen

zahlen“ genannt. Abbildung 2.3 zeigt die erwähnten Kategorien der Softwarequantitätskennzahlen. In den nächsten Abschnitten werden ausgewählte Softwarequantitätskennzahlen aus diesen Kategorien erläutert.

2.1.2.1.1 Funktionale Kennzahlen

Bei der funktionalen Größenmessung steht der Funktionsumfang der Software im Mittelpunkt der Quantitätsmessung [PP11]. Da Function Points die größte Verbreitung aller funktionalen Softwarequantitätskennzahlen erlangt haben [PP11], wird an dieser Stelle nur auf sie eingegangen, um die Idee der funktionalen Größenmessung zu erläutern. Weitere funktionale Kennzahlen sind Use-Case Points [Kar93] oder Object Points [BKK91].

Function Points wurden erstmals von Albrecht in [Alb79] definiert. Function Points nach der Definition von Albrecht wurden in der Norm ISO/IEC 20926 normiert und sind als „IFPUG FPA“ (FPA: Function Point Analyse) bekannt [ISO09]. Ausgehend von den IFPUG FPA entstanden weitere Methoden, mit denen Function Points ermittelt werden können [DA11]. Eine dieser Weiterentwicklungen ist die Norm ISO/IEC 19761 [ISO03], die als COSMIC Function Points (CFP) bekannt ist. Der folgende Absatz fasst das in [Bal00] erläuterte Vorgehen bei der Ermittlung von Function Points zusammen:

Jede Anforderung an ein Softwareprodukt wird einer von fünf Kategorien zugeordnet: Eingabedaten, Abfragen, Ausgabedaten, Datenbestände, Referenzdateien. Anschließend wird jede Produktanforderung in eine von drei Komplexitäten eingeordnet: einfach, mittel, komplex. Für jede Kombination Kategorie/Komplexität wird eine Punktzahl definiert, zum Beispiel wird eine Abfrage mittlerer Komplexität mit vier Punkten gewichtet. Auf

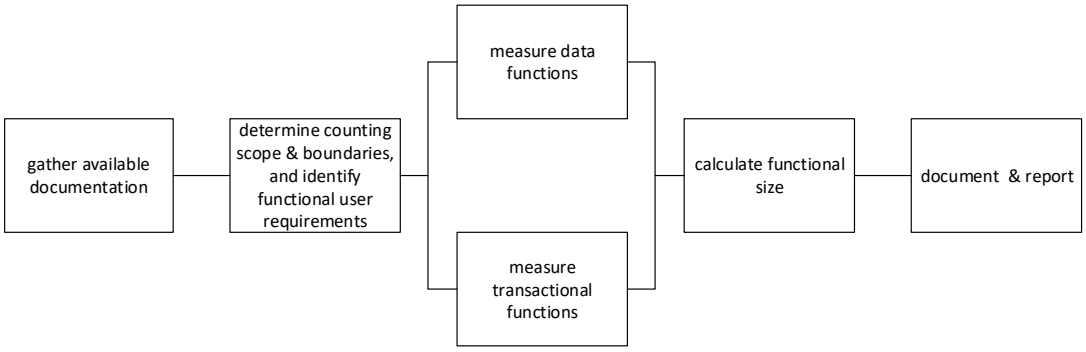


Abbildung 2.4: Prozess zur Ermittlung der Functional Size [ISO09]

diese Art und Weise werden alle Anforderungen an ein Softwareprodukt mit Punkten bewertet. Die Summe aller Punkte sind die Function Points des Softwareprodukts, also dessen funktionale Größe.

Die Anzahl an Function Points wird durch geschulte Experten manuell ermittelt. Die Norm ISO/IEC 20926 beschreibt den aus mehreren Teilaktivitäten bestehenden Prozess für die Ermittlung von Function Points (Abbildung 2.4). In diesem Prozess liegen zwei wesentliche Kritikpunkte an den Function Points begründet: Zum einen ist die erstmalige Anwendung bei bestehenden Softwareprodukten kompliziert und erfordert die Unterstützung von ausgebildeten Function Point Experten [Sym88, Car06]. Obwohl die Anzahl an Function Points objektiv durch das Softwareprodukt vorgegeben wird, zeige die Praxis, dass unterschiedliche Personen unterschiedliche Ergebnisse ermitteln [Abr14]. Zum anderen ist eine IT-basierte Datenerfassung von Function Points schwer umzusetzen [JYW+11, KBNAJ11]. Zwar sind einige Ansätze zur IT-basierten Datenerfassung von Function Points beschrieben [LH11, OBB+14, Kui14], die allerdings die vollständige Modellierung des zu messenden Softwareproduktes, zum Beispiel mit der Unified Modeling Language (UML) [OMG17c], voraussetzen. Solche Modellierungsmethoden sind jedoch in der Praxis nicht immer anzutreffen, beispielsweise ist die UML bei weitem nicht so verbreitet ist, wie allgemein angenommen [Pet13].

2.1.2.1.2 Quelltextkennzahlen

Im Gegensatz zu funktionalen Kennzahlen betrachten Quelltextkennzahlen den inneren Aufbau eines Softwareproduktes, indem sie die Anzahl der enthaltenen Softwareelemente messen. Die bekannteste Messung ist die Zählung von „Lines of Code“ (LOC). Andere Möglichkeiten sind die Zählung der Anzahl an Klassen, an Klassenmethoden und -attributen [SSB10]. Die Messung von Lines of Code ist zwar relativ einfach zu realisieren, allerdings wird ihre Eignung als Quantitätskennzahl in Frage gestellt. Dafür gibt es mehrere Gründe:

- Die Anzahl an LOC für das Lösen einer Programmieraufgabe ist abhängig von der Programmiersprache [PP11].
- Es ist zwischen Zeilen und Anweisungen zu unterscheiden [Car06]. Ein Beispiel ist die Unterscheidung zwischen Logical Lines of Code und Physical Lines of Code [Kas08].
- Ein effektiv programmierter Code ist kürzer als üppige Quelltextkonstrukte, die bezogen auf Ressourcenverbrauch und Rechenintensität kaum optimiert sind: „better programmers do more with less code“ [FPG+04].
- LOC zeigen nicht die Intensität der Bearbeitung des Quelltextes durch die Softwareteams an. Damit ist gemeint, dass die Anzahl an LOC ungefähr konstant bleiben kann, selbst wenn der Quelltext in größerem Maße geändert wird [RD13].

Quelltextkennzahlen können IT-basiert gemessen werden, zum Beispiel durch den Einsatz von Werkzeugen für die statische Quelltextanalyse [PCL17, Klo17, Pol17].

2.1.2.1.3 Prozesskennzahlen

Prozesskennzahlen messen den Bearbeitungsprozess der Erstellung von Softwareprodukten und Softwareversionen. Voraussetzung für die Messung von Prozesskennzahlen ist die Anwendung von Versionsmanagementsystemen. Versionsmanagementsysteme ermöglichen die kontrollierte gemeinsame Bearbeitung und Erstellung von Softwareprodukten unter Verwendung einer zentralen oder dezentralen Datenablage [LEPV10]. Beispiele für Prozesskennzahlen sind Anzahl hinzugefügter oder gelöschter Quelltextzeilen und die Anzahl geänderter Quelltextzeilen [RD13].

Die Anzahl geänderter, gelöschter oder hinzugefügter Quelltextzeilen wird *Churn* genannt [ME98]. Churn wird definiert: “as the sum of the number of lines added, deleted, and modified in the source code.“ [SJS12]. Eine Möglichkeit, den Churn konkret zu messen, ist die Nutzung von sogenannten *Unified Diff Patches* [JAB12]. *Unified Diff Patches* beschreiben den Namen der geänderten Datei und die Hinzufügungen und Löschungen, die in dieser Datei erfolgt sind. Hinzugefügte Zeilen werden mit einem „+“ markiert, gelöschte Zeilen mit einem „-“. Änderungen in einer Zeile werden durch eine Hinzufügung und eine Löschung markiert. Somit lässt sich analysieren, wie viele Quelltextzeilen gelöscht, hinzugefügt oder geändert wurden. Abbildung 2.5 zeigt ein Beispiel eines *Unified Diff Patches*.

Wie erwähnt, ändern sich Quelltextkennzahlen nicht immer wesentlich, insbesondere wenn die Softwareprodukte einen hohen Reifegrad erreicht haben und lediglich von Softwareversion zu Softwareversion verbessert werden. In [RD13] wird hingegen aufgeführt, dass

```
1  Index: Program.cs
2  ===================================================================
3  --- Program.cs  (revision 212459)
4  +++ Program.cs  (revision 212580)
5  @@ -29,8 +29,10 @@
6                 bool testServer = false;
7
8                 //for test purposes
9  -              if (project.Count() == 0)
10 +              if (project == "0")
11                    project = "FirstProject";
12 +              else if (project == "1")
13 +                  project = "SecondProject";
14
15                string output = project + ".csv";
16                bool validProject = true;
17
```

Abbildung 2.5: Beispiel eines *Unified Diff Patches*

sich Prozesskennzahlen selbst bei diesen Voraussetzungen deutlich stärker ändern als Quelltextkennzahlen und dadurch Indikatoren für die Quantität der Softwareänderungen sind. Außerdem seien sie zur Analyse möglicher Fehlerquellen geeignet: In Softwarekomponenten, die stark geändert werden, ist die Wahrscheinlichkeit des Vorhandenseins von Fehlern höher als in Softwarekomponenten, die kaum geändert werden.

Prozesskennzahlen sind IT-basiert erfassbar, da der Quelltext in Versionsmanagementsystemen verwaltet wird. Versionsmanagementsysteme haben Schnittstellen, die sogenannten Application Programming Interfaces (API), die von Informationsverarbeitungssystemen verwendet werden können.

2.1.2.1.4 Dokumentationskennzahlen

Die Softwaredokumentation erfolgt überwiegend in natürlicher Sprache [Sne07], für deren Erstellung in der Regel Texteditoren wie zum Beispiel MS Word verwendet werden. Die Messung der Quantität der Softwaredokumentation ist mit Hilfe dieser Texteditoren möglich. Zum Beispiel können mit MS Word Wörter, Zeichen und Seiten eines Dokuments gezählt werden. Obwohl die Softwaredokumentation ein wichtiges Ergebnis des Softwareentwicklungsprozesses ist und folglich deren Erstellungsprozess stetig verbessert und gemessen werden sollte, sind Dokumentationskennzahlen kaum Gegenstand der Forschung. Ein Beispiel ist aus [MR93] bekannt: Darin werden die erstellten Dokumentenseiten mit der Entwicklungsdauer in ein Verhältnis gesetzt.

Eine andere Möglichkeit der Erstellung der Softwaredokumentation ergibt sich bei Anwendung von IT-Systemen, mit denen sogenannte Work Items erfasst werden können: Work Items beschreiben sowohl zu erledigende Aktivitäten im Softwareentwicklungsprozess, wie zum Beispiel in [TS12], als auch Dokumentationselemente, etwa Produktanforderungen, Testfälle oder Fehlerbeschreibungen, wie zum Beispiel in [Moc03]. Durch das Zählen von

Work Items kann die Quantität der Softwaredokumentation gemessen werden. Die Zählung von Work Items ist IT-basiert möglich, da die verwaltenden IT-Systeme den Zugriff über API implementieren, die von Informationsverarbeitungssystemen verwendet werden können.

2.1.2.2 Kosten

Der überwiegende Teil der im Softwareentwicklungsprozess anfallenden Kosten sind die Personalkosten [Sne87]. Die Personalkosten werden aus den geleisteten Aufwänden, die in der Regel in Stunden angegeben werden, und dem betriebsspezifischen Kostensatz pro Stunde ermittelt. Der betriebsspezifische Kostensatz ist in der Regel abhängig von der Qualifikation des Mitarbeiters (Student, Facharbeiter, Ingenieur etc.) bzw. der durchgeführten Aufgabe (Test nach Anleitung, Entwurf einer Softwarearchitektur etc.). Die Aufwände werden in Soll-Aufwände und Ist-Aufwände unterschieden: Der Soll-Aufwand ist der geplante Aufwand für die Bearbeitung einer Aufgabe im Softwareentwicklungsprozess, zum Beispiel die Implementierung einer Softwarefunktion. Der Ist-Aufwand ist der tatsächlich geleistete Aufwand für die Bearbeitung der Aufgabe. Die Soll-Aufwände bzw. die Ist-Aufwände für die Bearbeitung einzelner Aufgaben werden zu dem Soll-Aufwand bzw. dem Ist-Aufwand für die Realisierung einer Softwareversion summiert.

Weitere Kosten im Softwareentwicklungsprozess entstehen zum Beispiel durch die Beschaffung von Lizenzen und Computern oder durch die Nutzung von Cloud-Services. Da diese Ressourcen in der Regel für die Entwicklung von mehreren Softwareprodukten eingesetzt werden, können diese Kosten nicht direkt einem einzelnen Softwareprodukt zugeordnet werden.

Darüber hinaus fallen indirekte Personalkosten in der Softwareentwicklung an. Dies ist zum Beispiel bei international verteilten Softwareentwicklungen der Fall, bei denen die Teams an mehreren Standorten in unterschiedlichen Ländern an einem Softwareprodukt arbeiten. Die Ursachen für indirekte Personalkosten liegen dabei in den sprachlichen und kulturellen Barrieren [LMT+10]. Indirekte Kosten sind lediglich abschätzbar und nicht direkt messbar.

Die im Softwareentwicklungsprozess geplanten bzw. aufgewendeten Stunden sind IT-basiert erfassbar, sofern sie in einer Datenablage, zum Beispiel zum Verwalten von Projektinformationen, eingetragen werden und die Daten aus der Datenablage mit Hilfe der API erfasst werden können.

2.1.2.3 Entwicklungsdauer

Die Entwicklungsdauer ist der Zeitraum zwischen dem Starttermin und dem Endtermin der Entwicklung einer Softwareversion [BMS02]. Der Starttermin und der Endtermin können für verschiedene Stakeholder des Softwareentwicklungsprozesses unterschiedlich sein. Ein Kunde bzw. ein Auftraggeber könnte den Tag der Auslieferung der Softwareversion als Endtermin betrachten, wohingegen das Entwicklungsteam bzw. der Dienstleister den Tag der internen Freigabe als Endtermin ansehen.

Es ist keine allgemeingültige Definition für den Starttermin bzw. den Endtermin bekannt. Um dennoch betriebsspezifisch die Entwicklungsdauer berechnen zu können, müssen Starttermin und Endtermin betriebsspezifisch definiert werden. Folgende Definitionen werden zum Beispiel in [Kas08] vorgeschlagen:

- Starttermin: Tag der Freigabe der Anforderungen
- Endtermin: Tag der ersten Installation des Softwareproduktes

Unabhängig davon, wie der Endtermin einer Softwareversion definiert ist, sollten nach diesem Termin keine Entwicklungsarbeiten an der Softwareversion erfolgen. Im weiteren Verlauf der Arbeit wird der Begriff „Endtermin“ in diesem Sinne verwendet. Eine Softwareversion enthält neue Softwarefunktionen. Für jede dieser Softwarefunktionen kann jeweils individuell ein *zugesagter Termin* formuliert werden. Der zugesagte Termin ist das einem Kunden genannte Datum, an dem die Auslieferung der Softwarefunktion erfolgen soll.

Alle genannten Termine sind IT-basiert erfassbar, sofern sie in einer Datenablage, zum Beispiel zum Verwalten von Projektinformationen, eingetragen werden und die Daten aus der Datenablage mit Hilfe der API erfasst werden können.

2.1.2.4 Qualität

Die Erfassung von Softwarekennzahlen zu Quantität, Kosten und Entwicklungsdauer ohne eine Erfassung der Softwarequalität ist nutzlos. Egal wie hoch die Quantität, wie kurz die Entwicklungsdauer oder wie gering die Kosten sind – ein minderwertiges Softwareprodukt wird nicht erfolgreich sein: „Without an accompanying assessment of product quality, speed of production is meaningless“ [FP97]. Daher stellen Softwarequalitätskennzahlen einen wesentlichen Bestandteil in der Softwaremessung dar. Die Softwarequalität kann zum einen durch das Erfassen und Verarbeiten von Fehlermeldungen und zum anderen durch das Erfassen und Verarbeiten von sogenannten Softwarequalitätseigenschaften

bestimmt werden. Dabei sollte die Messung der Qualität der Softwaredokumentation in einer gesamtheitlichen Softwarequalitätsmessung berücksichtigt werden.

2.1.2.4.1 Fehlermeldungen

In [SL03] wird ein Fehler wie folgt erläutert: Ein Fehler beschreibt einen Fehlerzustand oder eine Fehlerwirkung. Ein Fehlerzustand ist der inkorrekte Teil eines Softwareproduktes, der die Ursache für eine Fehlerwirkung ist. Eine Fehlerwirkung ist eine Abweichung zwischen einem spezifizierten bzw. einem implizit erwarteten Soll-Verhalten und dem Ist-Verhalten.

Fehler werden zu unterschiedlichen Zeitpunkten und von verschiedenen Stakeholdern entdeckt. Daher gibt es folgende Softwarekennzahlen, die die Anzahl von Fehlern anzeigen (in Anlehnung an [PP11]):

- Anzahl intern entdeckter Fehler: Intern entdeckte Fehler sind Fehler, die im Zusammenhang mit qualitätssichernden Maßnahmen während des Softwareentwicklungsprozesses durch die Softwareteams vor dem Endtermin einer Softwareversion entdeckt werden.
- Anzahl Restfehler: Restfehler sind Fehler, die nach Übergabe der Softwareversion an den Auftraggeber (Kunden) entdeckt werden. Ein Teil der Restfehler wird intern durch die Softwareteams entdeckt, ein anderer Teil wird durch den Auftraggeber (Kunden) entdeckt. Die letztgenannten Fehler sind *extern entdeckte Fehler*.
- Anzahl aller Fehler: Summe aus der Anzahl intern entdeckter Fehler und der Anzahl der Restfehler.

Aus diesen Softwarekennzahlen kann folgende weitere Softwarekennzahl bestimmt werden:

- Fehlerbehebungsrate: Verhältnis von der Anzahl intern entdeckter Fehler und der Anzahl aller Fehler.

Die Fehlerbehebungsrate ist eine Verhältniskennzahl. Zwar wird auf Verhältniskennzahlen in einem eigenen Abschnitt eingegangen (Abschnitt 2.1.2.5). Jedoch wird die Fehlerbehebungsrate in diesem Abschnitt erläutert, um das Verständnis von Fehlermeldungen zu erleichtern. Die Fehlerbehebungsrate ist ein gutes Maß für die Effektivität der internen Tests und sollte über 90 % liegen [SSB10]. Die Anzahl intern entdeckter Fehler sollte bei höherer Testintensität steigen [Sch11]. Daher ist die Fehlerbehebungsrate nur bei einer adäquaten Testdurchführung eine aussagekräftige Softwarekennzahl.

Zeitpunkt der Fehlerentdeckung	Kosten
Entdeckung während der Erstellung der Produktanforderungen	250 $
Entdeckung während der Erstellung des Softwareentwurfs	500 $
Entdeckung während der Programmierung und des Testens	1.250 $
Entdeckung nach dem Endtermin	5.000 $

Tabelle 2.1: Kosten-pro-Fehler-Analyse [Jon17]

Das frühzeitige Entdecken eines Fehlers reduziert die Kosten einer Fehlerbehebung: In [Jon17] sind die in Tabelle 2.1 gezeigten Kosten einer Fehlerbehebung in Abhängigkeit der Phase, in der der Fehler entdeckt wird, aufgeführt. Unabhängig davon, ob die in der Tabelle aufgeführten Geldbeträge für eine bestimmte Projektumgebung tatsächlich zutreffen, kann die dargestellte Tendenz verallgemeinert werden.

Alle entdeckten Fehler werden in der Regel hinsichtlich ihres sogenannten Schweregrades eingeordnet. Eine mögliche Einordnung ist gemäß [PP11] wie folgt:

- Kritischer Fehler: Die Anwendung ist oder wesentliche Funktionen der Anwendung sind nicht verfügbar bzw. nutzbar.
- Schwerer Fehler: Eine wesentliche Funktion ist nicht verfügbar oder liefert inkorrekte Ergebnisse, aber es gibt einen Workaround.
- Leichter Fehler: Eine nicht wesentliche Funktion ist nicht verfügbar oder liefert nicht die richtigen Ergebnisse.
- Trivialer Fehler: Kleinerer Fehler, der die Nutzung der Anwendung nicht wesentlich beeinträchtigt.

Der Schweregrad ist ein sogenanntes Fehlerattribut. Ein Fehlerattribut ist eine den Fehler beschreibende Eigenschaft. Weitere typische Fehlerattribute sind zum Beispiel Name des Melders, Datum der geplanten Behebung und der Status der Fehlerbehebung (geplant, behoben etc.).

Aus der Analyse des Schweregrades der entdeckten Fehler kann eine prozentuale Verteilung des Schweregrades über alle Fehler ermittelt werden. Die prozentuale Verteilung zeigt alle prozentualen Teilwerte an. Ein prozentualer Teilwert zeigt das Verhältnis der Fehler mit einem bestimmten Schweregrad zur Anzahl aller Fehler an. Abbildung 2.6 veranschaulicht diese Erläuterung: Sie zeigt ein Beispiel der prozentualen Verteilung des Schweregrades. Der Teilwert für die kritischen Fehler beträgt 46 %. Um diesen Teilwert interpretieren zu können, sollte die absolute Anzahl an Fehlern bekannt sein.

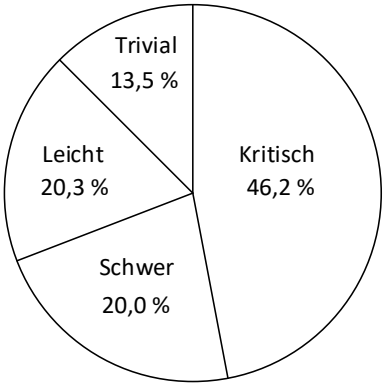


Abbildung 2.6: Erläuterung der prozentualen Fehlerverteilung

Eine IT-basierte Datenerfassung der prozentualen Verteilung bzw. der Anzahl aller Fehler ist möglich, sofern die entdeckten Fehler in Werkzeugen für die Fehlerverwaltung eingetragen werden, zum Beispiel in Bugzilla [Bug17]. Die API dieser Werkzeuge können von Informationsverarbeitungssystemen verwendet werden.

2.1.2.4.2 Qualitätseigenschaften

Die Erfassung und Verarbeitung von Fehlermeldungen unterstützen zwar in der Bewertung der Qualität des Softwareentwicklungsprozesses, sie ermöglichen allerdings keine qualitative Einschätzung des Softwareproduktes. Ein Beispiel einer qualitativen Einschätzung ist die Bewertung, ob ein Softwareprodukt gut nutzbar ist oder nicht. Derartige qualitative Einschätzungen werden erst durch die Definition und die Anwendung von sogenannten *Qualitätsmodellen* ermöglicht. Ein allgemeines Qualitätsmodell ist in der Normenreihe ISO/IEC 25000 definiert [ISO10] (Abbildung 2.7). Gemäß diesem Qualitätsmodell kann die Softwarequalität durch einzelne Charakteristiken, zum Beispiel durch die *Funktionalität*, beschrieben werden.

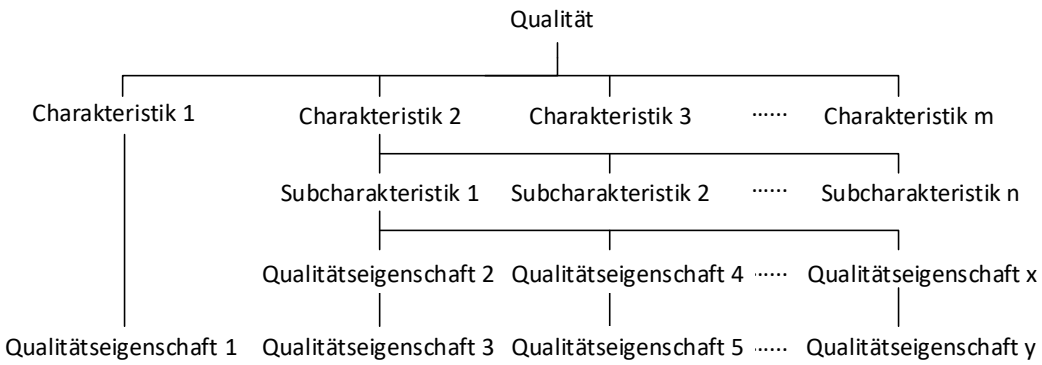


Abbildung 2.7: Allgemeines Qualitätsmodell in Anlehnung an [ISO10]

Den Charakteristiken werden Subcharakteristiken oder Qualitätseigenschaften zugeordnet. Charakteristiken und Subcharakteristiken sind nicht messbar, wohingegen die Qualitätseigenschaften messbar sind.

Die Norm ISO/IEC 25010 definiert außerdem ein Qualitätsmodell, das für die Bewertung des Nutzens einer Software aus Sicht des Anwenders verwendet werden kann. Dieses *Quality in Use* genannte Qualitätsmodell enthält die fünf in der Abbildung 2.8 gezeigten Charakteristiken. Die Norm ISO/IEC 25022 präzisiert das Quality in Use-Modell und ordnet den Charakteristiken und Subcharakteristiken messbare Qualitätseigenschaften zu [ISO16a]. Ein Beispiel einer Qualitätseigenschaft ist die *task time*, die der Charakteristik *efficiency* zugeordnet ist. Sie zeigt die Dauer für die Erledigung einer Aufgabe durch den Benutzer an.

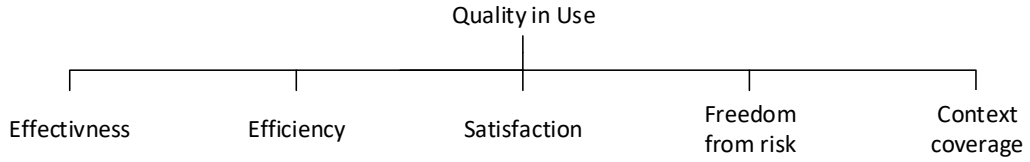


Abbildung 2.8: Quality in Use-Modell der ISO/IEC 25010 [ISO10]

Des Weiteren beschreibt die Norm ISO/IEC 25010 ein Qualitätsmodell für die Anwendbarkeit einer Software. Diese *Software Product Quality* genannte Qualitätsmodell enthält die acht in der Abbildung 2.9 gezeigten Charakteristiken. Die Norm ISO/IEC 25023 präzisiert das Software Product Quality-Modell und ordnet messbare Qualitätseigenschaften den Charakteristiken und Subcharakteristiken zu [ISO16b]. Ein Beispiel einer Qualitätseigenschaft ist die *Functional correctness*, die der Charakteristik *Functional suitability* zugeordnet ist. Sie zeigt die Anzahl der Funktionen an, die korrekte Ergebnisse liefern.

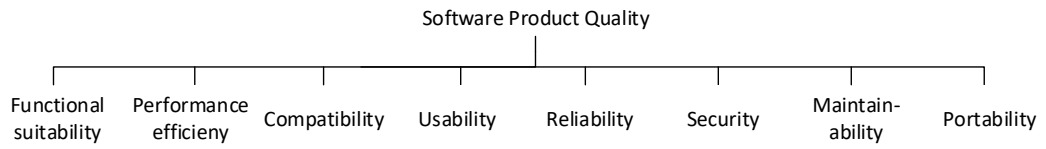


Abbildung 2.9: Software Product Quality-Modell der ISO/IEC 25010 [ISO10]

Die Definition eines Qualitätsmodells ist in der betrieblichen Praxis eine Herausforderung. Obwohl die Normen ISO/IEC 25022 und ISO/IEC 25023 die Qualitätsmodelle der ISO/IEC 25010 präzisieren und messbare Qualitätseigenschaften vorschlagen, können weder die Charakteristiken noch die Qualitätseigenschaften in allen produzierenden Betrieben angewendet werden. Die ISO/IEC 25010 betont daher, dass die Charakteristiken

und die Subcharakteristiken aus den betrieblichen Gegebenheiten abzuleiten und ihnen messbare Qualitätseigenschaften zuzuordnen sind.

In Abhängigkeit ihrer Definition können die Qualitätseigenschaften IT-basiert erfasst werden. Ein Beispiel einer IT-basierten Datenerfassung von Qualitätseigenschaften ist in [DP12] beschrieben: Ausgehend von dem Qualitätsmodell der ISO/IEC 9126 [ISO01], der Vorgängernorm der ISO/IEC 25010, werden Qualitätseigenschaften für die Anzeige der inneren Qualität von Softwareprodukten festgelegt. Ein Beispiel einer IT-basiert erfassbaren Qualitätseigenschaft ist die *file comment ratio*, die der Subcharakteristik *Analysability/Maintainability* im Qualitätsmodell *Software Product Quality* zugeordnet ist.

2.1.2.4.3 Dokumentationsqualität

Da die Softwaredokumentation ein wichtiges Ergebnis des Softwareentwicklungsprozesses ist, sollte sie Gegenstand von qualitativen Bewertungen sein. Der Messung der Qualität der Softwaredokumentation widmen sich Textanalysemethoden wie zum Beispiel in [Leh94, Sne05, GFL+13, Sne15, ASSH16]. In diesen Publikationen wird der Fokus auf die Qualität der Anforderungen an die Softwareprodukte mit dem Ziel gelegt, die Qualität der Anforderungen zu erhöhen. Eine Erhöhung der Qualität der Anforderungen reduziert das Risiko von Widersprüchen und Fehlern in den nachfolgenden Phasen der Softwareentwicklung, etwa im Systementwurf oder in der Implementierung.

Die Bewertung der Qualität der Anforderungen erfolgt toolunterstützt, was bedeutet, dass ein Textanalysetool alle Anforderungen hinsichtlich vorgegebener Kriterien automatisiert prüft. Ein mögliches Kriterium ist die Zählung bestimmter Schlüsselwörter in einer Anforderung, beispielsweise „wenn/dann“, „solange/bis“ etc. [ASSH16]. Je mehr solcher Schlüsselwörter in einer einzigen Anforderung verwendet werden, desto komplexer ist diese. Daraus wird geschlussfolgert, dass die Implementierung dieser Anforderungen schwierig ist. Um die Implementierung zu vereinfachen, sollten daher solche Anforderungen umformuliert bzw. geteilt werden.

Trotz der vorhandenen Publikationen und der darin zu sehenden Bestätigung, dass die Dokumentationsqualität Gegenstand der Forschung ist, ist keine einheitliche Methode hinsichtlich ihrer Messung bzw. ihrer Bewertung zu erkennen. Folglich sind die in den Publikationen genannten Kennzahlen, die die Dokumentationsqualität anzeigen (in [ASSH16] ist das die sogenannte *conjunctive complexity*), spezifisch für die jeweilige Methode.

2.1.2.5 Verhältniskennzahlen

Die meisten der in den Abschnitten 2.1.2.1 bis 2.1.2.4 vorgestellten Softwarekennzahlen sind den absoluten Kennzahlen zuzuordnen (vgl. Abbildung 2.1). Eine Ausnahme ist die Fehlerbehebungsrate, die das Verhältnis der Projektfehlerrate zur Gesamtfehlerrate darstellt. Da die Erläuterung der Fehlerbehebungsrate thematisch in den Kontext der Softwarequalität gehört, wurde sie bereits in Abschnitt 2.1.2.4.1 erläutert.

Aus den anderen erläuterten absoluten Softwarekennzahlen können unterschiedliche Verhältniskennzahlen gebildet werden. In der Literatur sind wiederholt die folgenden Verhältniskennzahlen beschrieben:

2.1.2.5.1 Softwareproduktivität

Es gibt keine allgemein anerkannte Definition des Begriffs „Softwareproduktivität“. Eine von Petersen durchgeführte Analyse über Veröffentlichungen zur Softwareproduktivität zeigt die Existenz vielfältiger Ansätze [Pet11]. Petersen fand insgesamt 586 Beiträge, von denen er 38 im Detail analysierte. Viele dieser Veröffentlichungen enthalten teilweise völlig verschiedene Ansichten darüber, wie die Softwareproduktivität zu erfassen sei.

Für diese Arbeit wird die Softwareproduktivität als das Verhältnis von Softwarequantität und Aufwand betrachtet, wie dies auch in [Kas08, PP11] der Fall ist. Gleichung 2.1 zeigt die aus dieser Definition resultierende allgemeine Berechnungsformel.

$$Produktivität = \frac{Quantität}{Aufwand} \tag{2.1}$$

In Abhängigkeit der verwendeten Quantitätskennzahl sind mögliche Maßeinheiten der Softwareproduktivität entweder $\frac{Function\ Points}{Hour}$, $\frac{LOC}{Hour}$ oder $\frac{Churn}{Hour}$ ($Churn$ ist ein Platzhalter für die Maßeinheit des Churns).

2.1.2.5.2 Liefergeschwindigkeit

Die Liefergeschwindigkeit (engl.: speed of delivery) ist eine Verhältniskennzahl, die die Geschwindigkeit von Softwareentwicklungsprozessen anzeigt [Sym10, PP11]. In agilen Softwareentwicklungsmethoden ist der Begriff „Velocity“ gebräuchlich [HD06]. Die Liefergeschwindigkeit gibt an, welche Softwarequantität in einer bestimmten Zeit erstellt wurde. Gleichung 2.2 zeigt die allgemeine Berechnungsformel.

$$Liefergeschwindigkeit = \frac{Quantität}{Entwicklungsdauer} \tag{2.2}$$

In Abhängigkeit der verwendeten Quantitätskennzahl sind mögliche Maßeinheiten der Liefergeschwindigkeit entweder $\frac{Function\ Points}{Day}$, $\frac{LOC}{Day}$ oder $\frac{Churn}{Day}$.

2.1.2.5.3 Fehlerdichte

Die Fehlerdichte (engl.: defect density) ist das Verhältnis von Quantität und Fehlerraten, zum Beispiel „defects per source line of code“ [NC14]. In [FP97] wird der Fehlerdichte eine so hohe Bedeutung beigemessen, dass sie als „de facto standard measure of software quality“ bezeichnet wird. Gleichung 2.3 zeigt die allgemeine Berechnungsformel.

$$Fehlerdichte = \frac{Anzahl\ Fehler}{Quantität} \tag{2.3}$$

In Abhängigkeit der verwendeten Quantitätskennzahl sind mögliche Maßeinheiten der Fehlerdichte entweder $\frac{Defects}{Function\ Points}$, $\frac{Defects}{LOC}$ oder $\frac{Defects}{Churn}$.

Wie bereits erwähnt, kann ein Fehler in zwei Phasen entdeckt werden:

- In der Phase der qualitätssichernden Maßnahmen des Softwareentwicklungsprozesses
- In der Phase nach der Übergabe der Software an den Auftraggeber, auch „post release“ Phase genannt

Fehlerdichten können jeweils für eine dieser Phasen berechnet werden. Ein Beispiel ist die Kennzahl „Post Release Defect Density“ [Kas08]. Des Weiteren lässt sich die Fehlerdichte über beide Phasen hinweg und über den vollständigen Lebenszyklus der Software ermitteln. Die Fehlerdichte kann lediglich so präzise sein wie die Anzahl der tatsächlich entdeckten Fehler. Die unentdeckten Fehler gehen nicht in die Berechnung der Fehlerdichte ein.

Nachdem in diesem Abschnitt 2.1.2 mögliche Softwarekennzahlen für die Menge K_S identifiziert wurden, werden im nächsten Abschnitt Produktionskennzahlen und deren Einsatzgebiete erläutert.

2.1.3 Produktionskennzahlen

Menschen, Betriebsmittel (Maschinen) und Arbeitsgegenstände sind zentrale Systemelemente des Produktionsprozesses [SBL10]. Die Arbeitsgegenstände sind die Produkte oder Teilprodukte, die von Menschen und Betriebsmitteln bearbeitet werden. Diese Bearbeitung ist in Abläufe gegliedert. Die jeweiligen Abläufe für die Systemelemente sind in Ablaufarten unterteilt [REF92]. Tabelle 2.2 zeigt die Systemelemente und nennt Beispiele für Ablaufarten. Bezogen auf die aufgeführten Systemelemente wurden Produktions-

Systemelement	Beispiele für Ablaufarten
Mensch	Haupttätigkeit
	Nebentätigkeit
	zusätzliche Tätigkeit
Betriebsmittel	Hauptnutzung
	Nebennutzung
	zusätzliche Nutzung
Arbeitsgegenstand	Verändern
	Prüfen
	Liegen

Tabelle 2.2: Ablaufartengliederung der REFA-Methodenlehre [REF92]

kennzahlen definiert, so zum Beispiel in [REF92] und [HJK14]. Sie unterstützen die vorrangige Zielstellung von Produktionsprozessen: Die Einhaltung von Kostenvorgaben bzw. stetige Reduzierung von Kosten. Die in [REF92] und [HJK14] genannten Produktionskennzahlen bezogen auf die Menschen sind zum Beispiel der Zeitgrad und die Arbeitsproduktivität, für die Betriebsmittel sind es die Gesamtanlageneffektivität (engl.: Overall Equipment Effectiveness, kurz: OEE) und der Hauptnutzungsgrad, für die Arbeitsgegenstände die Durchlaufzeit und die Liefertermintreue.

Zeichnet sich die Produktion durch eine hohe Kapitalintensität aus, werden vor allem Produktionskennzahlen zur Steuerung des Einsatzes der Betriebsmittel eingesetzt. Sind die Arbeitskosten die dominierende Kostenart, werden vor allem Produktionskennzahlen in Bezug auf die Menschen ausgewählt. Die ausgewählten Produktionskennzahlen sind abhängig von den betriebsspezifischen Produktionsabläufen. Ein produzierender Betrieb wählt die eingesetzten Produktionskennzahlen individuell für sein Umfeld aus, wobei die Semantik einzelner Produktionskennzahlen betriebsspezifisch sein kann. Damit ist gemeint, dass eine Produktionskennzahl mit demselben Namen in verschiedenen produzierenden Betrieben eine unterschiedliche Bedeutung hat bzw. von einer allgemein anerkannten Definition abweicht. Erläuterung 2.1 zeigt dafür ein Beispiel.

Die Wertschöpfung und die bereits erwähnte *First Pass Rate* sind Produktionskennzahlen, die beim Kooperationspartner verwendet werden. Sie sind, wie alle anderen Kennzahlen, in unternehmensinternen Dokumenten beschrieben. Nicht alle beschriebenen Produktionskennzahlen kommen in allen Produktionslinien zum Einsatz, da sie zum Teil nur für spezifische Produktionsabläufe sinnvoll angewendet werden können oder sich entweder auf kapitalintensive oder personalintensive Produktionslinien beziehen. Aus diesen Produktionskennzahlen werden in dieser Arbeit einige ausgewählt, um sie in die Menge K'_{SW} zu transferieren und in der Softwaredomäne zu nutzen (vgl. Abbildung 1.5).

Beim Kooperationspartner wird die Produktionskennzahl *Wertschöpfung* verwendet. Sie zeigt die geplanten Fertigungskosten auf Basis der Arbeitspläne der Mitarbeiter und der Maschinenauslastungen an. In [Pre08] wird die Wertschöpfung als der Bruttoproduktionswert abzüglich der Kosten für Roh-, Hilfs- und Betriebsstoffe, Abschreibungen, Fremddienstkosten und Kostensteuern definiert. Die Wertschöpfung hat folglich beim Kooperationspartner und in [Pre08] eine unterschiedliche Bedeutung. Dieses Beispiel verdeutlicht die Wichtigkeit des Verständnisses der betriebsspezifischen Semantik einer Produktionskennzahl. Eine betriebsfremde Person würde die Wertschöpfung womöglich falsch interpretieren und diese Fehlinterpretation in der Entscheidung zu prozessgestaltenden Maßnahmen berücksichtigen.

Erläuterung 2.1: Beispiel einer betriebsspezifischen Semantik einer Produktionskennzahl

Inwiefern ein Methoden- bzw. Kennzahlentransfer von der Produktionsdomäne in die Softwaredomäne sinnvoll und möglich ist, wird im folgenden Abschnitt dargelegt.

2.2 Produktion und Softwareentwicklung

Die Anwendung von Methoden der Produktionsdomäne in der Softwaredomäne ist ein in der Wissenschaft und in der Praxis kontrovers diskutiertes Thema. Da diese Arbeit zum Ziel hat, Produktionskennzahlen in der Softwaredomäne anzuwenden, agiert sie genau in diesem Spannungsfeld.

Die Softwareentwicklung ist im Wesentlichen ein manueller Prozess, dessen Erfolg stark von den Erfahrungen und Fähigkeiten der einzelnen Softwareentwickler abhängt [NCK+15]. Automatisierte Abläufe sind zwar im Softwaretest zunehmend anzutreffen, jedoch weniger im Softwareentwurf oder in der Softwareprogrammierung. Jede entwickelte Software ist verschieden. Die Softwareentwicklung ist ein Prozess des „ongoing designs“, d.h. Software wird kontinuierlich konzipiert und implementiert [Ste06]. Allerdings werden in verschiedenen Softwareprodukten durchaus Aufgabenstellungen gelöst, die bereits in anderen Softwareprodukten gelöst wurden. So zeigen die Beobachtungen beim Kooperationspartner, dass bereits gelöste Aufgabenstellungen zu einem späteren Zeitpunkt neu bearbeitet wurden, statt den zu der gelösten Aufgabenstellung gehörenden Quelltext wiederzuverwenden. Beispiele für mehrfach implementierte Aufgabenstellungen sind Druck-, Dateispeicher - oder Dateiladefunktionen.

Der Test einer Softwareversion eines intelligenten Produktes ist dessen Baumusterprüfung zuzuordnen. Eine Baumusterprüfung ist ein Verfahren, bei dem eine benannte Stelle bescheinigt, dass ein repräsentatives Muster des intelligenten Produktes die Bestimmungen von Richtlinien, zum Beispiel der CE-Konformitätsrichtlinie, erfüllt [SB16]. Das bedeutet, dass die Softwareversion zwar im Rahmen der Softwareentwicklung, allerdings nicht für jedes produzierte intelligente Produkt, getestet wird.

Im Gegensatz zu den dargestellten Eigenschaften der Softwareentwicklung wird im Produktionsprozess wiederholt ein identisches Produkt gefertigt. Der Grad der Automatisierung hängt von den Merkmalen des konkreten intelligenten Produktes ab, ist jedoch höher als in der Softwareentwicklung. Ziel der Automatisierung sind deterministische Produktionsprozesse, mit denen kontinuierlich definierte Liefer- und Qualitätsziele erreicht werden können. Diese Ziele, Determinismus und Vorhersehbarkeit, waren bislang und bleiben auch für die Industrie 4.0 gültig [Vol17].

Determinismus und Vorhersehbarkeit sind zweifelsohne gültige Ziele für die Softwareentwicklung. Doch trotz vieler Jahre der Softwareforschung sind Softwareentwicklungsprozesse noch immer wenig vorhersehbar in Bezug auf Kosten, Termine oder Qualität [NCK+15]. Dies kann als ein Grund für die vielfältigen Forschungsarbeiten angesehen werden, in denen der Transfer von Produktionsmethoden in die Softwaredomäne untersucht wird.

Ein Beispiel eines derartigen Methodentransfers ist Kanban. Kanban ist eine Lean Management-Produktionsmethode, die erstmalig bei Toyota eingesetzt wurde [Lik04]. Hiranabe übertrug deren zugrundeliegenden Ideen und Konzepte und entwickelte eine Softwareentwicklungsmethode, die Software-Kanban genannt wird [Hir08]. Software-Kanban wird den agilen Softwareentwicklungsmethoden zugeordnet. Weitere Adaptionen von Lean Management-Produktionsmethoden in der Softwareentwicklung werden in [PW10, SJ12] beschrieben.

Naedele et al. präsentieren einen Ansatz, in dem Konzepte von Manufacturing Execution Systems (MES) in der Softwareentwicklung angewendet werden [NCK+15]. Eine der Aufgaben von MES in der Produktion ist die Datenerfassung und -verarbeitung, um die Produktionsplanung und -steuerung zu optimieren. Naedele et al. haben ein vergleichbares Konzept für die Softwareentwicklung erarbeitet und somit eine Produktionsmethode in die Softwaredomäne übertragen.

Schneidewind argumentiert, es gäbe bereits mehr Schnittmengen zwischen Softwareentwicklung und Produktion als allgemein angenommen [Sch11]. Er vergleicht eine komponentenorientierte Softwarearchitektur mit einer Materialstückliste, sieht im Kompilierprozess einen Fertigungsprozess und vergleicht einen automatisierten Build-Prozess mit einer Fertigungsautomatisierung. Diese Beispiele von Schnittmengen würden zeigen, dass es eine gewisse Ähnlichkeit von Teilprozessen der Softwareentwicklung mit Teilprozessen der Produktion gäbe und somit ein Methodentransfer von der Produktionsdomäne in die Softwaredomäne möglich sei.

Ein weiteres Beispiel eines Methodentransfers ist die Anwendung von Methoden zur Qualitätssteuerung von Prozessen, beispielsweise wie die statistische Prozesskontrolle oder Six Sigma. In verschiedenen Veröffentlichungen wurde diskutiert, ob derartige Methoden

für die Softwaredomäne anwendbar sind [Car94, BBBC09, Sch11, Bin97, RSRL08]. In [Car94, BBBC09, Sch11, RSRL08] wird dies positiv eingeschätzt, wogegen Binder meint, dass Software und Produktion zu unterschiedlich seien und kein Methodentransfer möglich sei [Bin97].

In Zuge von Veröffentlichungen, die im Laufe dieser Arbeit eingereicht wurden, gab es zu diesem Themenkomplex unterschiedliche Kommentare der Gutachter. So teilten nicht alle Gutachter die Einschätzung, dass Methoden aus der Produktion in die Domäne der Softwareentwicklung übertragbar sind. Andere dagegen ermunterten zu mehr Forschungsarbeiten auf diesem Gebiet.

Aus diesen Ausführungen lässt sich schlussfolgern, dass die Anwendbarkeit von Produktionsmethoden in der Softwaredomäne uneinheitlich bewertet wird. Dieser Arbeit liegt die Überzeugung zugrunde, dass die Softwaredomäne von der Produktionsdomäne lernen kann und sollte, da zum einen Determinismus und Vorhersehbarkeit von Prozessen in beiden Domänen zentrale Bedürfnisse sind und zum anderen Determinismus und Vorhersehbarkeit in der Produktion stärker ausgeprägt sind, als in der Softwaredomäne.

Eine Frage dieser Arbeit ist, wie das Management Produktionskennzahlen in der Softwaredomäne nutzen kann, um den Determinismus und die Vorhersehbarkeit in der Softwareentwicklung zu verbessern. Den für die Beantwortung dieser Fragestellung bekannten Grundlagen widmet sich der nächste Abschnitt.

2.3 Transfer und Bestimmung von Kennzahlen

Es sind lediglich wenige Beispiele bekannt, in denen Kennzahlen aus einer Domäne in einer anderen Domäne genutzt werden bzw. in denen ein Kennzahlentransfer zwischen zwei Domänen beschrieben wird.

Die Kennzahl „Produktivität“ ist ein solches Beispiel: Die Produktivität ist eine Kennzahl, die ihren Ursprung in der Domäne der Betriebs- oder Volkswirtschaft hat. Sie wird allerdings auch in der Softwareentwicklung angewendet [Pet11], um die Wirtschaftlichkeit von Softwareentwicklungen zu bewerten. In der einschlägigen Literatur fehlen allerdings Hinweise, dass der Transfer zwischen den unterschiedlichen Domänen auf Grundlage einer methodischen Vorgehensweise erfolgte. Diese Vorgehensweise wäre, sofern nachvollziehbar, für diese Arbeit relevant.

Ein weiteres Beispiel ist aus [FHZ+15] bekannt. Darin werden zum einen die aus der Produktionsdomäne bekannte Lernrate und zum anderen die aus der Entropie bekannte Entropierate für die Qualitätsmessung von ITIL-Prozessen in Cloud-Systemen in der IT-

Domäne verwendet. Diese beiden Kennzahlen werden für die Bestätigung der in [FHZ+15] formulierten Hypothesen angewendet. Allerdings fehlt eine Beschreibung dazu, wie die Kennzahlen ausgewählt und deren Eignung in der IT-Domäne geprüft wurden. Eine nachvollziehbare methodische Vorgehensweise wäre wiederum für diese Arbeit relevant.

Bei diesen beiden Beispielen handelt es sich zwar um einen Kennzahlentransfer, allerdings kann kein methodisches Vorgehen erkannt werden. Folglich ist in dieser Arbeit eine nachvollziehbare Methode für einen Kennzahlentransfer zu entwickeln.

Es gibt jedoch einige Methoden für die methodische Bestimmung von Kennzahlen für betriebliche Prozesse. Dabei werden zunächst strategische bzw. operative Ziele definiert und erst danach werden für diese Ziele Kennzahlen bestimmt. Eine derartige methodische Bestimmung ist notwendig, da Kennzahlen keinem Selbstzweck dienen dürfen. Sie sollten nicht nur deswegen verwendet werden, weil sie messbar sind, sondern die Bestimmung von Kennzahlen sollte sich vielmehr an den Zielen des produzierenden Betriebes orientieren. Zudem birgt die Anwendung von isolierten bzw. zu keinem Ziel gehörenden Kennzahlen die Gefahr von Fehlentscheidungen [Pre08].

Da es bereits einige Methoden für die methodische Bestimmung von Kennzahlen gibt, wird in den folgenden Abschnitten geprüft, ob eine dieser Methoden als Ausgangspunkt für die in dieser Arbeit zu entwerfende Methode des Kennzahlentransfers dienen kann.

2.3.1 Bewertungsgrundlagen

Für die Bewertung, welche Methode als eine sogenannte *Basismethode* für einen Kennzahlentransfer dienen kann, werden in diesem Abschnitt Anforderungen an die Basismethode formuliert. Diese Anforderungen sind das Ergebnis der folgenden Überlegungen:

Es soll eine in einer Ausgangsdomäne vorhandene Kennzahl in eine Zieldomäne transferiert werden. In einem erfolgreichen Transfer muss die in der Ausgangsdomäne gültige Semantik der Kennzahl in der Zieldomäne erhalten bleiben. Gemäß den Ausführungen in Abschnitt 2.1.1 bilden der Name, die Maßeinheit, der Wertebereich, der Idealwert, die Möglichkeit der Festlegung von Soll-Werten, die Frage, das Ziel und die Interpretation die Semantik einer Kennzahl. Nach einem Kennzahlentransfer müssen folglich all diese Informationsinhalte weiterhin für die Kennzahl beschreibbar sein. Des Weiteren müssen sie in der Zieldomäne im Grundsatz identisch zu den Informationsinhalten der Ausgangsdomäne sein. Mit der Formulierung „im Grundsatz identisch“ wird ausgedrückt, dass im Idealfall alle Informationsinhalte in beiden Domänen identisch sind, dies jedoch aufgrund der unterschiedlichen Domänen voraussichtlich nicht immer möglich sein wird. Es ist folglich jeweils individuell zu bewerten, ob ein Informationsinhalt „im Grundsatz identisch“

ist oder nicht. Folgende Kriterien, die jeweils einen Informationsinhalt in der Zieldomäne mit dem dazugehörenden Informationsinhalt in der Ausgangsdomäne vergleichen, werden einer derartigen Bewertung zugrunde gelegt:

- Der Name muss identisch sein.
- Die Maßeinheit muss identisch sein.
- Der Wertebereich muss identisch sein.
- Der Idealwert muss identisch sein.
- Soll-Werte müssen festgelegt werden können, können jedoch domänenspezifisch verschieden sein.
- Die Frage ist im Satzbau identisch, jedoch steht sie im Kontext der jeweiligen Domänen. Mit „im Satzbau identisch“ ist Folgendes gemeint: Wenn die Frage in der Ausgangsdomäne wie folgt aufgebaut ist: „Wie ist das Verhältnis von ... zu ...?“, dann muss dieser Satzbau ebenfalls in der Zieldomäne gelten. Die Frageninhalte, die in ein Verhältnis gesetzt werden, sind allerdings domänenspezifisch.
- Das Ziel muss identisch sein, domänenspezifische Formulierungen sind erlaubt.
- Die Interpretation lässt erkennen, dass der Adressat der Kennzahl in beiden Domänen im Grundsatz identisch auf die Ist-Werte der Kennzahl reagiert. Dies soll am Beispiel der in Abschnitt 2.1 dargestellten Interpretation der FPR in der Produktion erläutert werden: Wenn die FPR in der Softwareentwicklung angewendet wird, dann kann das Management ähnlich wie in der Produktion auf Abweichungen zwischen Soll- und Ist-Werten reagieren. Eine mögliche Maßnahme wäre jedoch zum Beispiel die Beschaffung von neuen Softwarewerkzeugen statt der Beschaffung von neuen Maschinen.

Für den Transfer des Namens, der Maßeinheit, des Wertebereichs und dessen Idealwert sowie der Möglichkeit der Festlegung von Soll-Werten bedarf es keiner expliziten Methode. So könnte zum Beispiel festgelegt werden, dass eine FPR in der Softwaredomäne äquivalent zur FPR in der Produktionsdomäne in % angegeben wird, den Wertebereich von 0 % bis 100 % hat und deren Idealwert 100 % ist. Diese Festlegung ist jedoch nicht ausreichend, da sie lediglich einen Namenstransfer und keinen für diese Arbeit benötigten Kennzahlentransfer bedeuten würde. Es ist offensichtlich, dass ohne den Transfer der weiteren, die Semantik einer Kennzahl beschreibenden Informationsinhalte eine semantische Äquivalenz zwischen der FPR in der Produktion und der FPR in der Software nicht erreicht werden kann.

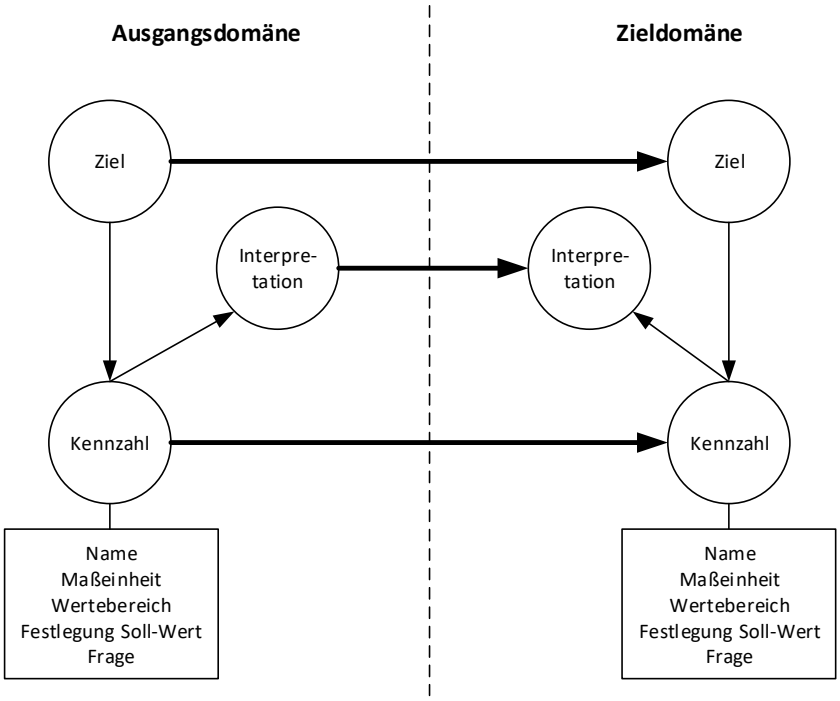


Abbildung 2.10: Grundkonzept eines Kennzahlentransfers

Zwar kann der Satzbau der Frage ebenfalls schlicht übertragen werden, jedoch müssen die domänenspezifischen Frageninhalte durch ein strukturiertes Vorgehen ermittelt werden. Diese Anforderung gilt ebenfalls für die Übertragung des Ziels und der Interpretation. Folglich sind das Ziel und die Interpretation unter Anwendung einer Methode zu transferieren, in der ebenfalls die domänenspezifischen Frageninhalte der Frage ermittelt werden.

Eine Basismethode muss daher einige Eigenschaften aufweisen, die anhand Abbildung 2.10 erläutert werden. Diese Abbildung zeigt das Grundkonzept des zu entwickelnden Kennzahlentransfers:

In der Ausgangsdomäne existiert eine zu transferierende Kennzahl, die in der Zieldomäne wiederverwendet werden soll. Die Kennzahl ist aus einem Ziel, das für die Ausgangsdomäne formuliert ist, abgeleitet und diesem Ziel zugeordnet. Sie wird durch den Namen, die Maßeinheit und den Wertebereich beschrieben. Des Weiteren ist für sie ein domänenspezifischer Soll-Wert vorgegeben und eine Frage formuliert. Die definierte Kennzahl wird im Kontext der Prozessdurchführung erfasst, verarbeitet und durch den Adressaten der Kennzahl interpretiert. Wie in Abschnitt 2.1 dargelegt, ist die Interpretation eine Handlungsbeschreibung der Reaktion des Adressaten auf die Soll-Werte. Da diese Reaktion individuell unterschiedlich sein kann, stellt die Interpretation einen nicht eindeutig beschreibbaren Informationsinhalt einer Kennzahl dar.

Eine Kennzahl kann zwar mehreren Zielen zugeordnet sein, allerdings muss eines der Ziele den Startpunkt des Kennzahlentransfers bilden. Soll diese Kennzahl in der Zieldomäne

eingesetzt werden, ist dies nur möglich, wenn das konkret ausgewählte Ziel sowohl in der Ausgangsdomäne als auch in der Zieldomäne gültig ist. Falls die Kennzahl in der Zieldomäne einem anderen Ziel zugeordnet wird als die Kennzahl in der Ausgangsdomäne, bedient sie nicht die gleichen Informationsbedürfnisse des Managements bzw. der operativen Teams wie die Kennzahl in der Ausgangsdomäne.

Des Weiteren muss die Interpretation sowohl in der Ausgangsdomäne als auch in der Zieldomäne im Grundsatz identisch sein. Wie bereits erläutert, ist damit gemeint, dass der Adressat der Kennzahl auf die Soll-Werte der in der Zieldomäne erfassten Kennzahl in im Grundsatz identischer Art und Weise reagiert wie auf die in der Ausgangsdomäne erfassten Soll-Werte.

Neben dem Ziel und der Interpretation müssen der Name, die Maßeinheit, der Wertebereich und der Idealwert identisch sein. Es ist naheliegend, dass ein Adressat nur dann die transferierte Kennzahl im Grundsatz identisch interpretieren kann, wenn diese Anforderung erfüllt ist. Ebenfalls muss die Möglichkeit der Festlegung eines Soll-Werts erhalten bleiben. Dieser muss nach einem Kennzahlentransfer zwar nicht identisch sein, jedoch sollten die Soll-Werte in der Zieldomäne tendenziell den Soll-Werten in der Ausgangsdomäne ähnlich sein: Der Soll-Wert einer in der Softwareentwicklung angewendete FPR sollte zum Idealwert von 100 % tendieren, und nicht etwa bei 10 % festgesetzt werden. Letzteres würde eine im Grundsatz identische Interpretation nicht zulassen. Abschließend ist es erforderlich, den Satzbau der Frage zu erhalten, um die Kennzahl semantisch äquivalent zu transferieren.

Aus den Ausführungen des Grundkonzepts eines Kennzahlentransfers ergeben sich zwei Anforderungen an die Basismethode. Die Erfüllung dieser Anforderungen ist die Voraussetzung für deren Eignung als Ausgangspunkt für die in dieser Arbeit zu entwerfende Methode eines Kennzahlentransfers.

Anforderung 1 (Ziel-Kennzahl-Zuordnung) *Die Basismethode stellt innerhalb einer Domäne die Zuordnung zwischen einer Kennzahl und einem Ziel her.*

Anforderung 2 (Interpretation-Kennzahl-Zuordnung) *Die Basismethode stellt innerhalb einer Domäne die Zuordnung zwischen einer Kennzahl und deren Interpretation her.*

In den folgenden Abschnitten werden mögliche Basismethoden bewertet. Deren Auswahl erfolgt aufgrund folgender Überlegungen:
Rein finanzwirtschaftliche Kennzahlensysteme wie zum Beispiel das Du-Pont-Schema [Gla03] oder das ZVEI-Kennzahlensystem [ZVE70] werden nicht betrachtet, da die Softwareentwicklung nicht mit Finanzkennzahlen gesteuert werden sollte. Die Steuerung sollte sich vielmehr an den strategischen Zielen eines produzierenden Betriebes orientieren,

welche über die reinen Finanzziele hinausgehen. Eine etablierte Methode für die Steuerung eines produzierenden Betriebes auf Basis solcher Ziele ist die Balanced Scorecard [KN92]. Des Weiteren gibt es in der Domäne der Softwareentwicklung zwei etablierte Methoden bzw. Prozessbeschreibungen für die systematische Bestimmung von Softwarekennzahlen: die Norm ISO/IEC/IEEE 15939 [ISO17] und die Goal-Question-Metric-Methode (GQM-Methode) [BW84]. Da diese Arbeit die kennzahlenorientierte Gestaltung des Softwareentwicklungsprozesses zum Ziel hat, werden neben der Balanced Scorecard die Norm ISO/IEC 15939 und die GQM-Methode ebenfalls näher betrachtet.

Weitere vorhandene Ansätze bzw. Methoden für die systematische Bestimmung von Kennzahlen im Software- und IT-Umfeld, zum Beispiel [Wes99, Küt10, Gau14] werden im Weiteren nicht berücksichtigt, da diese Ansätze bzw. Methoden auf eine der drei genannten Methoden zurückgeführt werden können: In [Wes99] ist die GQM-Methode die Basis für den darin beschrieben 12-stufigen Prozess zur Softwarekennzahlenbestimmung, den Ausführungen in [Küt10] liegt die Balanced Scorecard zugrunde. In [Gau14] wird die COBIT-Methode, eine Methode zur Steuerung und Überwachung der Unternehmens-IT, umfassend beschrieben. Die COBIT-Methode beinhaltet den Aufbau von Kennzahlensystemen. Wie in [Gau14] aufgeführt ist, wird dafür die Balanced Scorecard verwendet.

Im Folgenden werden die drei genannten Methoden in der erwähnten Reihenfolge in einer Vertiefung erläutert, die für eine Bewertung der oben aufgeführten Anforderungen ausreichend ist.

2.3.2 Balanced Scorecard

Lange Zeit wurden für die Unternehmenssteuerung lediglich reine Finanzkennzahlensysteme eingesetzt. Allerdings setzte sich schrittweise die Erkenntnis durch, dass für eine langfristige positive Unternehmensentwicklung neben dem Blick auf die Finanzen auch eine gesamtheitliche Unternehmensbetrachtung erfolgen sollte. Mit dem Ziel einer solchen gesamtheitlichen Betrachtung wurde die Balanced Scorecard (BSC) entwickelt [KN92]. Die Balanced Scorecard ermöglicht die Betrachtung auf ein Unternehmen aus mehreren Perspektiven: aus der Finanzperspektive, aus der Kundenperspektive, aus der internen Prozessperspektive und der Lern-und Entwicklungsperspektive.

Ausgehend von seinen Strategien und Visionen sollte ein Unternehmen für jede dieser Perspektiven eine Mission formulieren, zu deren Konkretisierung Ziele zu formulieren sind. Anhand dieser Ziele sind Kennzahlen zu bestimmen, die die Zielerreichung anzeigen. Auf diese Weise entsteht ein Zusammenspiel aus den Strategien, den Visionen, den Missionen, den Perspektiven, den Zielen und den Kennzahlen, wodurch eine gesamtheitliche Betrachtung auf ein Unternehmen ermöglicht wird (Abbildung 2.11).

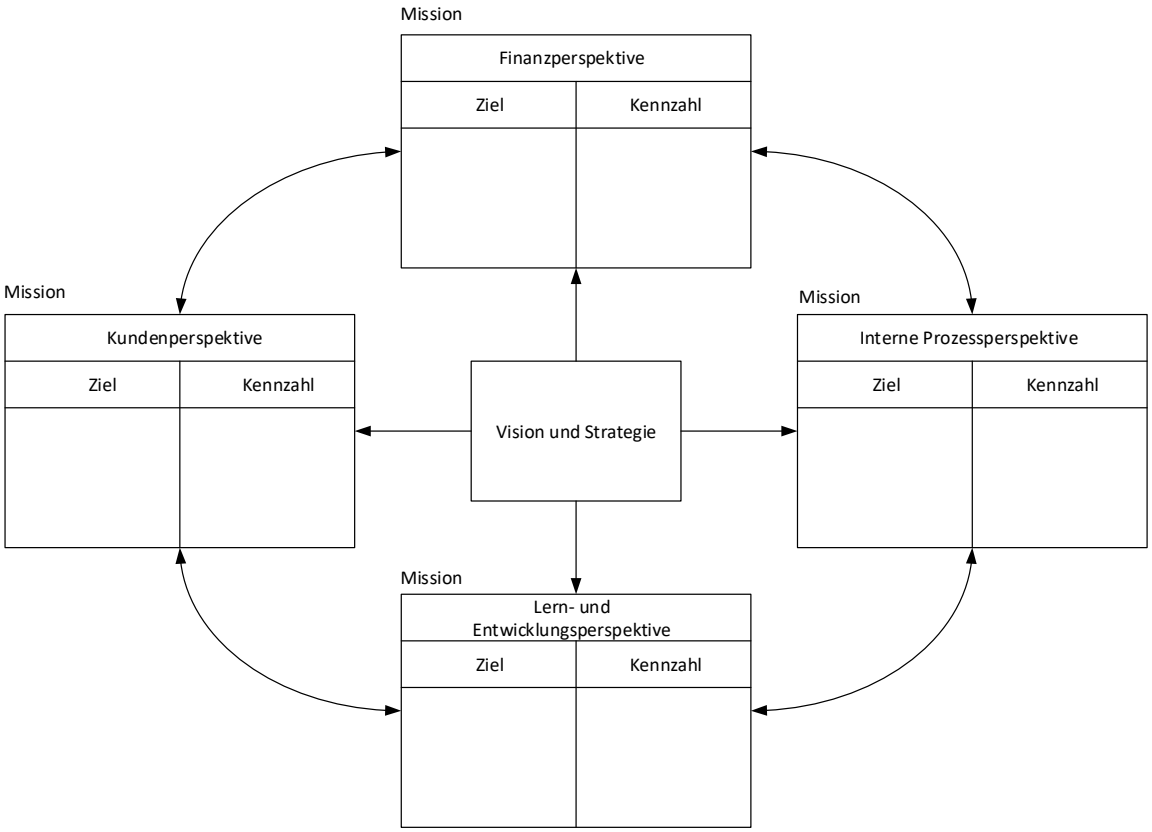


Abbildung 2.11: Modell der Balanced Scorecard in Anlehnung an [KN92, Kap10]

Die Anwendung der Balanced Scorecard ist nicht eingeschränkt auf bestimmte Unternehmens- bzw. Betriebstypen. In der einschlägigen Literatur werden Beispiele ihrer Anwendung u.a. in produzierenden Betrieben [KN92], in Kreditinstituten [FS99], in Softwarefirmen [IPPS02] oder der Anwendung in IT-Abteilungen [Küt10] aufgeführt.

Wie in Abbildung 2.11 zu erkennen ist, wird in der Balanced Scorecard ein eindeutiger Zusammenhang zwischen einem Ziel und einer Kennzahl hergestellt. In [IPPS02] wird zum Beispiel das strategische Ziel „Neue Kunden gewinnen“ in der Kundenperspektive formuliert, dem die Kennzahl „Anzahl an Kunden“ zugeordnet ist.

Des Weiteren ist in der Abbildung 2.11 zu erkennen, dass ein Zusammenhang zwischen der Interpretation und der Kennzahl nicht explizit hergestellt wird. Selbstverständlich muss der Adressat einer Kennzahl diese in irgendeiner Art und Weise interpretieren, um zu bewerten, ob das dazugehörende Ziel erreicht wird bzw. ob Maßnahmen definiert werden müssen, um die Zielerreichung zu ermöglichen. Der Begriff „Interpretation“ wird in dem Modell der Balanced Scorecard jedoch nicht verwendet.

2.3.3 ISO/IEC/IEEE 15939

Die Norm ISO/IEC/IEEE 15939 *Systems and software engineering-Measurement process* (deutsch: System- und Software-Engineering - Messverfahren) ist eine Prozessbeschreibung und widmet sich dem „Measurement Process“. Der Begriff wird definiert als: „process for establishing, planning, performing and evaluating measurement within an overall project, enterprise or organizational measurement structure“. Dies beinhaltet den Aufbau und die Anwendung eines Informationsverarbeitungssystems für Softwarekennzahlen.

Ein Beispiel für die praktische Anwendung der ISO/IEC/IEEE 15939 ist u.a. in [SMKN11] erläutert. In anderen Publikation wird das Paradigma der ISO/IEC/IEEE 15939 unter der Bezeichnung Practical Software Measurement (PSM) verwendet, zum Beispiel in [JLC12]. Aufgrund der Ähnlichkeiten von PSM und der ISO/IEC/IEEE 15939, kann die Norm als die Standardisierung von PSM betrachtet werden.

Die Norm ISO/IEC/IEEE 15939 enthält zum einen relevante Begriffsdefinitionen der Softwaremessung, zum anderen beschreibt sie das Prozessmodell zum Aufbau und zur Anwendung eines Informationsverarbeitungssystems für Softwarekennzahlen (Abbildung 2.12). Die Kreise zeigen alle Prozessaktivitäten, wobei das grau hinterlegte Rechteck die zentralen Prozessaktivitäten umrandet. Die Pfeile zeigen den Datenfluss. Das weiß hinterlegte Rechteck symbolisiert den zu messenden Prozess, aus dem die Daten generiert werden.

Das Prozessmodell unterstützt bei der Identifizierung von Aufgaben, die notwendig sind, um Softwarekennzahlen zu definieren, auszuwählen und zu verbessern. In der Norm wer-

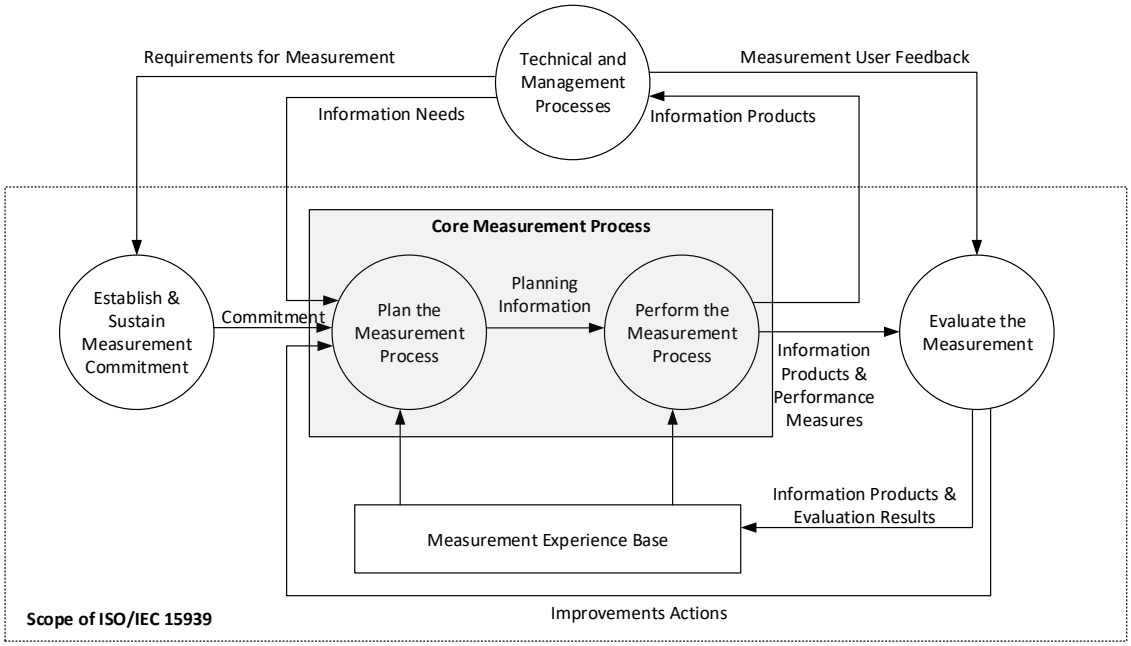


Abbildung 2.12: Prozessmodell der Softwaremessung nach ISO/IEC/IEEE 15939 [ISO17]

den jedoch keine konkreten Softwarekennzahlen vorgeschlagen. Im Folgenden werden die Aktivitäten des Prozessmodells erläutert:

Establish & Sustain Measurement Commitment: Es wird der Bereich für die durchzuführenden Messungen identifiziert sowie die Bereitschaft des Managements eingeholt, Ressourcen für den Aufbau und für die kontinuierliche Verbesserung von Softwarekennzahlensystemen dauerhaft bereitzustellen und die Messergebnisse aktiv für eine Prozesssteuerung anzuwenden. Die Personalressourcen für den Aufbau des Informationsverarbeitungssystems werden zugewiesen.

Plan the Measurement Process: Es werden die Informationsbedürfnisse der Organisationseinheit ermittelt. Sie sind die Basis für die Bestimmung der Kennzahlen. Die Verfahren für die Messung, für die Datenspeicherung und für die Anzeige der Kennzahlen werden definiert und die Bewertungskriterien für die Kennzahlen erstellt.

Perform the Measurement Process: Es werden die Messverfahren in den zu messenden Prozess integriert. Dies kann bewirken, dass der Prozess ggf. angepasst werden muss, damit die Messverfahren durchgeführt werden können. Die Daten werden erfasst, zu Kennzahlen verarbeitet und die Kennzahlen werden den Adressaten der Kennzahlen zugeführt.

Evaluate the Measurement: Die Adressaten sichten und bewerten die Ist-Werte der Kennzahlen. Gegebenenfalls werden Prozessverbesserungsmaßnahmen definiert und umgesetzt.

Die ISO/IEC/IEEE 15939 betont die Bedeutung der Relation der ausgewählten Softwarekennzahlen zu den Informationsbedürfnissen der Organisationseinheiten, wie beispielsweise einem produzierenden Betrieb. Die Informationsbedürfnisse entstehen in den technischen und in den Managementprozessen und basieren auf den Zielen der Organisationseinheiten. Der Zusammenhang zwischen den Informationsbedürfnissen und dem Prozess wird in einem Informationsmodell hergestellt (Abbildung 2.13). Die Informationsbedürfnisse werden durch ein Informationsprodukt befriedigt. Letzteres enthält einen oder mehrere Indikatoren und dessen Interpretation. Es bildet sich aus einer oder mehreren, in der Norm als „abgeleitet“ bezeichnete Kennzahlen („derived measures“), die sich wiederum aus Basiskennzahlen („base measures“) bilden, die im Prozess erfasst werden.

Wie in Abbildung 2.13 zu erkennen, wird in der Norm ein Zusammenhang zwischen Zielen (den Informationsbedürfnissen) und Kennzahlen hergestellt. Dabei handelt es sich nicht um eine direkte Zuordnung, sondern mehrere Kennzahlen bilden Indikatoren, die die Informationsbedürfnisse befriedigen. Die Interpretation ist ein wichtiger Teil des Informationsmodells und wird daher in der Norm mehrfach referenziert. Die Interpretation basiert auf Indikatoren, die jeweils auf mehreren Kennzahlen aufbauen.

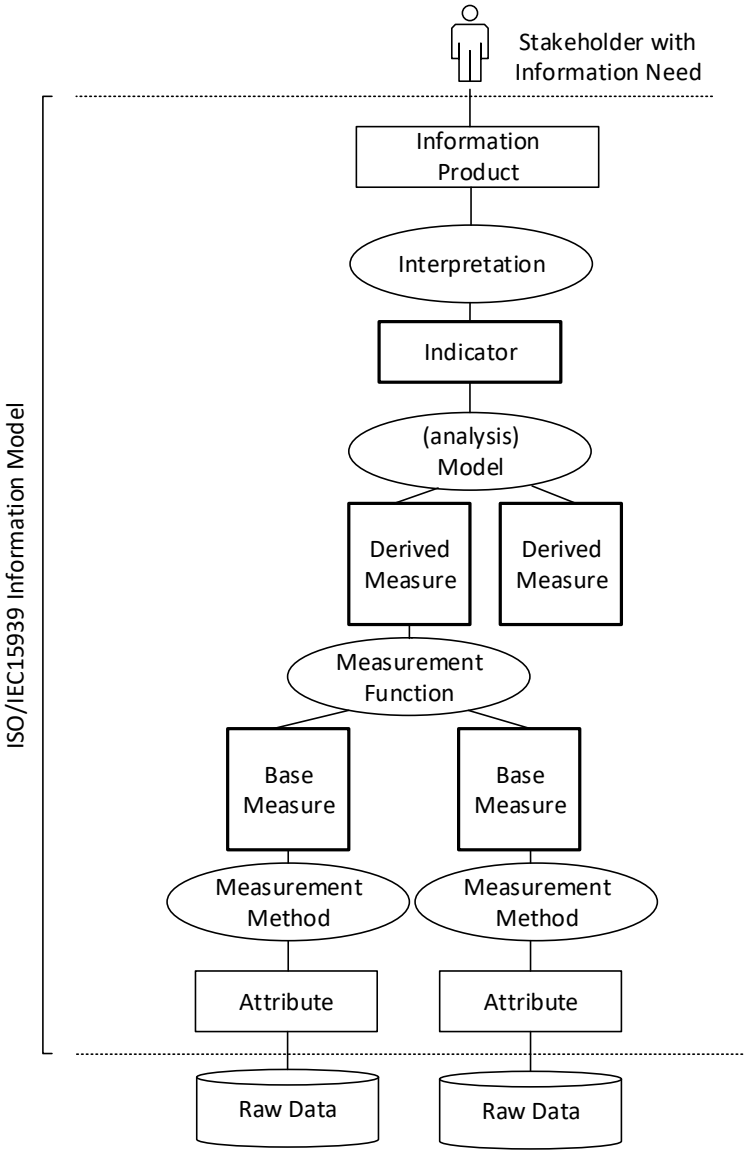


Abbildung 2.13: Informationsmodell der ISO/IEC/IEEE 15939 [SMKN11]

2.3.4 GQM-Methode

Der Goal-Question-Metric-Methode (GQM-Methode) [BW84] wird in der einschlägigen Literatur eine hohe Bedeutung beigemessen: Sie sei die Grundlage für jede Softwaremessung [SSB10] bzw. es sei nachgewiesen, dass sie eine besonders effektive Methode zur Auswahl von Softwarekennzahlen ist [FP97]. Beispiele für die praktische Anwendung der GQM-Methode sind u.a. in [FLM+98, SB99, Wes99, Moh08, HK15] aufgeführt.

Die GQM-Methode stellt die strategischen bzw. die operativen Ziele der Softwaremessung in den Vordergrund. Ausgehend von diesen Zielen werden Fragen (Questions) formuliert, die mit Kennzahlen beantwortet werden sollen. Erst danach werden die konkreten Kennzahlen (Metrics) festgelegt. Die Vorgehensweise der Ermittlung der Ziele, Fragen und Kennzahlen erfolgt schrittweise, wobei die Beschreibung der Vorgehensweise in den un-

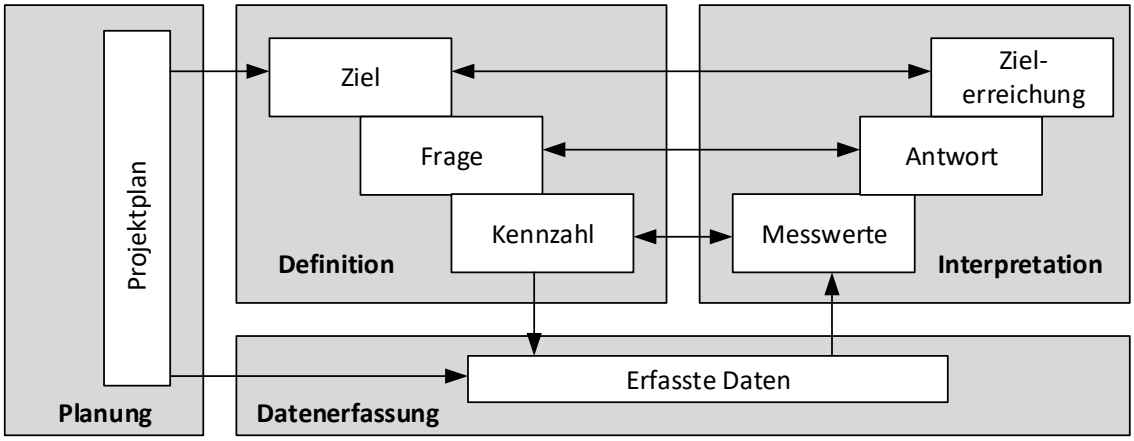


Abbildung 2.14: Die Phasen der GQM-Methode nach [SB99]

terschiedlichen Publikationen nicht einheitlich ist. Zum Beispiel werden in [Bal97] sechs Schritte beschrieben, in [SB99] werden vier Phasen erläutert. Im Kern ist die Vorgehensweise jedoch ähnlich, daher wird im Folgenden nur eine mögliche Vorgehensweise erläutert, und zwar die in [SB99] beschriebene. Nach dieser werden während der Anwendung der GQM-Methode vier Phasen durchlaufen: Planung, Definition, Datenerfassung und Interpretation (Abbildung 2.14).

Planungsphase: In der Planungsphase werden ein Projektteam etabliert, ein Projektplan erstellt, der zu verbessernde Prozess identifiziert und das Einverständnis des Managements zum geplanten Vorgehen eingeholt. In der Planungsphase ist es wichtig, die in dem Softwareentwicklungsprozess beteiligten Mitarbeiter einzubinden. Es ist zu erklären, wieso die Messungen durchgeführt werden und dass eine aktive Mitarbeit die Voraussetzung für eine erfolgreiche Durchführung der GQM-Methode ist.

Definitionsphase: In der Definitionsphase werden die zu messenden Kennzahlen identifiziert. Dabei werden zunächst die Ziele der Messung definiert. Danach werden Fragen erfasst, die die Ziele verfeinern und die mit konkreten Kennzahlen beantwortet werden können. Abbildung 2.15 zeigt diesen Top-Down-Ansatz. Des Weiteren wird der Softwareentwicklungsprozess danach bewertet, ob er die Messung aller Kennzahlen unterstützt. Falls nicht, sind die dafür notwendigen Anpassungen zu beschreiben und umzusetzen.

Datenerfassungsphase: Für jede Kennzahl wird bestimmt, wie sie erfasst wird. Die Datenerfassung kann sowohl manuell als auch in elektronischer Form erfolgen. Es sollten zunächst Probemessungen getätigt werden, auf deren Basis die Methoden der Datenerfassung aktualisiert werden. Alle Messungen sollten im Kontext eines Informationsverarbeitungssystems erfolgen, das in dieser Phase aufzubauen ist. Es werden alle Messwerte erfasst.

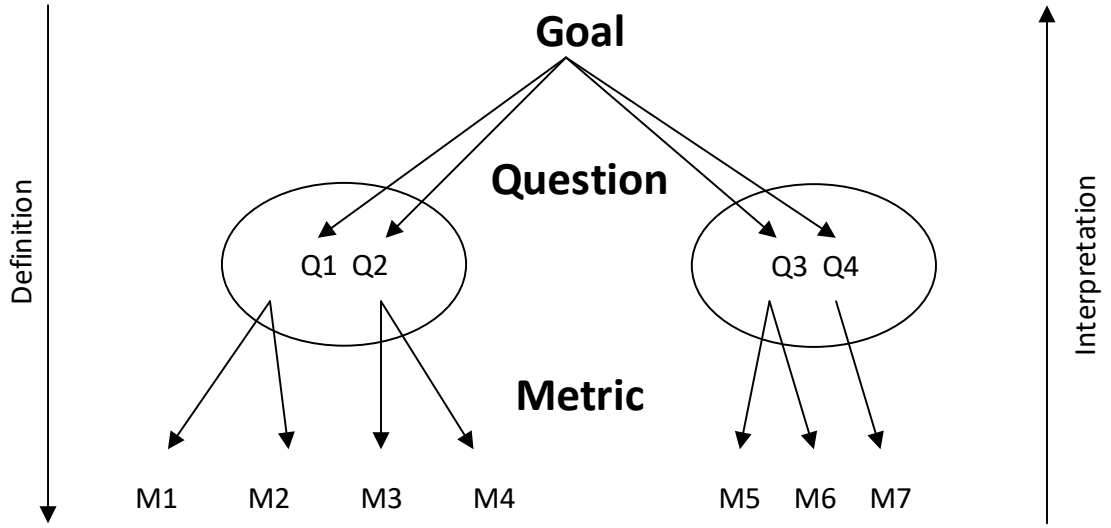


Abbildung 2.15: Definitionsphase der GQM-Methode nach [SB99]

Interpretationsphase: Die erfassten Ist-Messwerte werden vom Adressaten der Kennzahlen interpretiert. Dies erfolgt in umgekehrter Reihenfolge der Kennzahlendefinition: Die Messwerte beantworten die Fragen und mit den beantworteten Fragen wird die Zielerreichung überprüft (Abbildung 2.15). Neben der Interpretation der Messwerte erfolgt eine Bewertung des GQM-Prozesses. Hierbei sind die beteiligten Personen zu befragen, um deren Einschätzungen für mögliche Anpassungen in den nachfolgenden Messprozessen zu berücksichtigen.

Wie in der Abbildung 2.15 zu erkennen ist, stellt die Anwendung der GQM-Methode top-down einen eindeutigen Zusammenhang zwischen einem Ziel und einer Kennzahl her. Des Weiteren ist zu erkennen, dass die Kennzahlen bottom-up interpretiert werden, um die Erreichung der Ziele zu bewerten. Letzteres erfolgt in der Interpretationsphase (vgl. Abbildung 2.14).

2.3.5 Bewertung

Im Folgenden wird bewertet, ob die dargestellten Methoden bzw. Prozessbeschreibungen die im Abschnitt 2.3.1 formulierten Anforderungen an eine Basismethode für den in dieser Arbeit zu entwickelnden Kennzahlentransfer erfüllen. Tabelle 2.3 zeigt die Bewertung: Ein „+“ bedeutet, dass die Anforderung gut erfüllt wird. Ein „o“ bedeutet, dass die Anforderung mit Einschränkung erfüllt wird.

Die GQM-Methode erfüllt beide Anforderungen ohne Einschränkung. Sie stellt jeweils eindeutig einen Zusammenhang zwischen einem Ziel und einer Kennzahl bzw. zwischen

Anforderung	BSC	ISO/IEC/IEEE 15939	GQM
A1: Ziel-Kennzahl-Zuordnung	+	o	+
A2: Interpretation-Kennzahl-Zuordnung	o	o	+

Tabelle 2.3: Bewertung der Methoden bzw. Prozessbeschreibungen

einer Interpretation und einer Kennzahl her. Im Informationsmodell der ISO/IEC/IEEE 15939 sind zwar die Kennzahlen die Basis für Indikatoren, die die Informationsbedürfnisse, also die Ziele, bedienen, allerdings fehlt die direkte Zuordnung zwischen einer Kennzahl und einem Ziel. Diese Zuordnung erfolgt implizit, daher wird der Grad der Erfüllung der Anforderung A2 mit einem „o“ bewertet. Jedoch wird im Vergleich zur GQM-Methode diese Zuordnung als schwächer bewertet. In dem Modell der Balanced Scorecard wird der Begriff „Interpretation“ nicht erwähnt. Folglich fehlt eine explizite Zuordnung zwischen einer Kennzahl und einer Interpretation. Dies ist zwar in der Praxis implizit gegeben, und daher wird die Erfüllung der Anforderung A2 mit einem „o“ bewertet. Der Grad der Erfüllung wird allerdings schwächer bewertet als der Grad der Erfüllung durch die GQM-Methode.

Aus diesen Gründen wird die GQM-Methode als Basismethode für die Entwicklung einer Methode für den Transfer von Produktionskennzahlen in die Domäne der Softwareentwicklung ausgewählt. Bevor im Abschnitt 2.5 der Softwareentwicklungsprozess betrachtet wird, widmet sich der nächste Abschnitt dem Prozess der Produktentstehung.

2.4 Produktentstehungsprozess

2.4.1 Definition und Eigenschaften

Produkte entstehen im Rahmen eines Produktentstehungsprozesses. Der Produktentstehungsprozess ist Teil des Produktlebenszyklus, dessen Phasen und Tätigkeiten Abbildung 2.16 zeigt. Der Produktentstehungsprozess umfasst die im gestrichelten Rechteck eingerahmten Phasen Anforderungen, Produktplanung, Entwicklung und Prozessplanung. Sein Resultat sind sowohl das intelligente Produkt als auch die Produktionsunterlagen [ES09].

Intelligente Produkte beruhen auf der Mechatronik und entstehen durch das Zusammenwirken mehrerer Disziplinen zum Beispiel der Mechanik, der Elektronik (beides ist die Hardware des Produktes) und der Softwaretechnik. Das Attribut *intelligent* bringt zum Ausdruck, dass die Software ein dominanten Anteil zum Wert eines Produktes beiträgt.

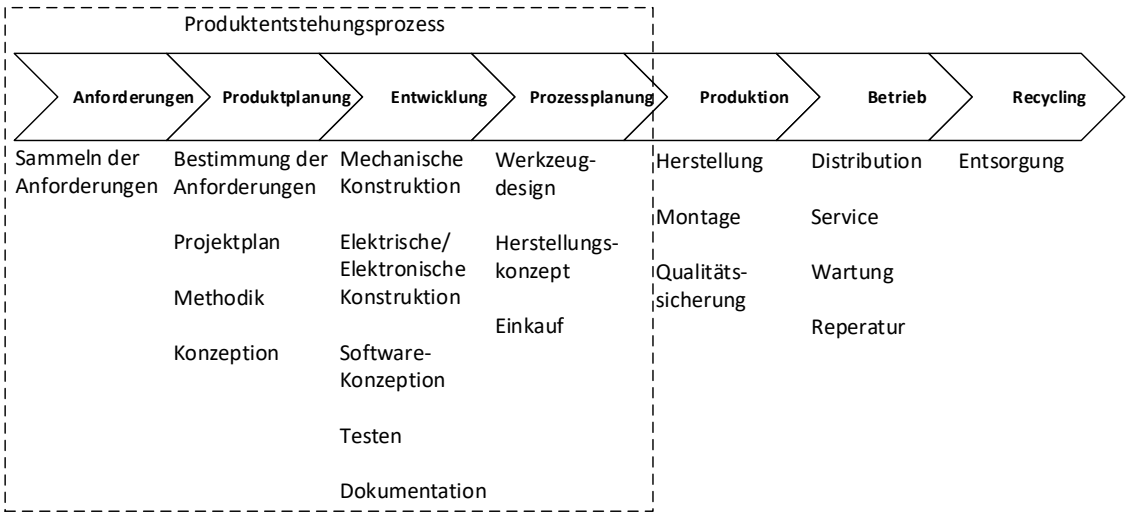


Abbildung 2.16: Phasen und Tätigkeiten des Produktlebenszyklus gemäß [ES09]

In der Hardware- und Softwareentwicklung haben sich spezifische Entwurfstechniken etabliert. Für die Hardwareentwicklung sei stellvertretend die VDI-Richtlinie 2221 [VDI93] genannt. Bekannte Entwicklungsmethoden der Softwareentwicklung sind das Wasserfall-Modell, das V-Modell und Scrum, auf die in Abschnitt 2.5 eingegangen wird. Für einen umfangreichen Überblick über bekannte disziplinspezifische Entwurfsmethoden wird auf [ERZ14] verwiesen.

Mechatronische Produkte erfordern aufgrund des Zusammenwirkens verschiedener Disziplinen einen interdisziplinären Entwurfsansatz. Der Komplexitätsgrad der jeweiligen Entwicklungsaufgabe ist abhängig von der Komplexität des konkreten Produkts [Ben07]. Unter dem Begriff *Komplexität* wird sowohl der innere Zusammenhang von Produkten als auch deren Vielgestaltigkeit verstanden. Der innere Zusammenhang wird durch die Konnektivität ausgedrückt und die Vielgestaltigkeit durch die Varietät [Bru91]. Abbildung 2.17 veranschaulicht den Begriff Komplexität von Produkten.

Durch die wachsende Komplexität intelligenter Produkte infolge ihrer höheren Konnektivität und Vielgestaltigkeit, wächst zunehmend der Komplexitätsgrad der Entwicklungsaufgaben. Im sogenannten *Systems Engineering* wird eine Lösung gesehen, den stetig wachsenden Komplexitätsgrad der Entwicklung intelligenter Produkte zu beherrschen.

2.4.2 Systems Engineering

Systems Engineering ist ein facettenreicher Begriff, was sich in einer relativ großen Anzahl an Definitionen äußert [Tsc16]. Im Wesentlichen wird darunter das Management parallellaufender Entwicklungsprozesse verschiedener Disziplinen über den gesamten Produktlebenszyklus verstanden [BB16]. Es gibt zahlreiche Systems Engineering-Buchliteratur, zum

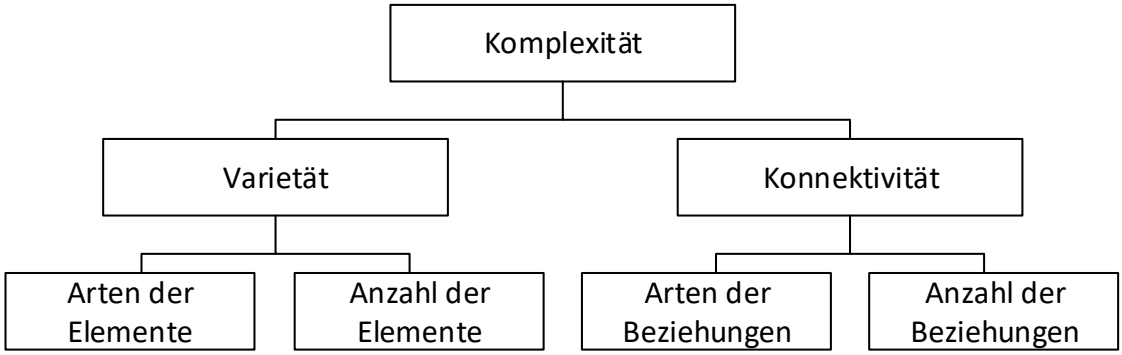


Abbildung 2.17: Komplexität von Produkten nach [Bru91]

Beispiel [Wei14, HdFV15, BB16] und einige der Themenfelder des Systems Engineering sind in internationalen Normen standardisiert, zum Beispiel in der ISO/IEC/IEEE 42010 [ISO11] und in der ISO/IEC/IEEE 15288 [ISO15].

Die dem Systems Engineering zugrundeliegende Philosophie gliedert sich in das Systemdenken und in ein Vorgehensmodell [Tsc16]. Das Ergebnis des Systemdenkens ist die fachliche Beschreibung des Systems, das Vorgehensmodell beschreibt das Vorgehen in der Systementwicklung.

Ziel der fachlichen Beschreibung des Systems ist zum einen die Definition der Korrelationen zwischen Systemanforderungen, Funktionen, Verhalten und Struktur des Systems [EKM17]. Ein zentraler Systems Engineering-Gedanke ist es, die genannten Aspekte zunächst unabhängig von einer technischen Lösung zu beschreiben. Damit soll eine frühzeitige Festlegung auf eine konkrete Umsetzung vermieden werden, also ob zum Beispiel eine Produktfunktion durch Hardware oder Software realisiert wird. Eine frühzeitige Festlegung, zum Beispiel auf bekannte Lösungen, verhindert womöglich neue kreative Ansätze.

Zum anderen sollen im Prozess der Systemmodellierung alle Stakeholder frühzeitig in den Produktentstehungsprozess eingebunden werden. Darin liegt ein weiterer Vorteil des Systems Engineering: Eine konsequente Umsetzung von Systems Engineering erfordert eine ständige Kommunikation der Stakeholder untereinander. Missverständnisse, zum Beispiel zwischen Hardwareentwickler und Softwareentwickler bezogen auf die Realisierung einer Produktfunktion, die gegebenenfalls zu Projektverzögerungen führen würden, können vermieden werden. Allerdings ist in vielen Unternehmen ein gesamtheitliches Bewusstsein für Systems Engineering noch nicht zu erkennen, obwohl der Nutzen von Systems Engineering in der Praxis anerkannt ist bzw. teilweise nachgewiesen werden kann [Elm08, GCW+13].

Die Themenfelder des Systems Engineering sind vielfältig: Der in der VDI-Richtlinie 2206 dargestellte Entwurfsprozess zeigt zum Beispiel das Anforderungsmanagement, den Systementwurf und die Systemintegration als Themenfelder (vgl. Abbildung 1.1)[VDI04]. Die

Beschreibung eines Systems im klassischen Systems Engineering bzw. dessen Ergebnisdokumentation der Themenfeldern erfolgt dokumentenbasiert. Das sogenannte Modell Based System Engineering (MBSE) als eine Weiterführung des klassischen Systems Engineering basiert hingegen auf digitalen Modellen, die entlang des Produktentstehungsprozesses integriert werden [EKM17].

Die digitalen Modelle sind zum einen systemübergreifend. Ein systemübergreifendes Modell wird Systemmodell genannt. Für die Systemmodellierung gibt es verschiedene Modellierungssprachen, zum Beispiel die System Modeling Language (SysML) [OMG17b] und die Spezifikationstechnik CONSENS, die aus den Arbeiten in [ADG+09] entstanden ist. Zum anderen sind die digitalen Modelle disziplinspezifisch. Beispiele dafür sind die M-CAD-Modelle in der Mechanik und die UML-Modelle in der Softwareentwicklung. Abbildung 2.18 zeigt schematisch den Zusammenhang zwischen den Modellarten.

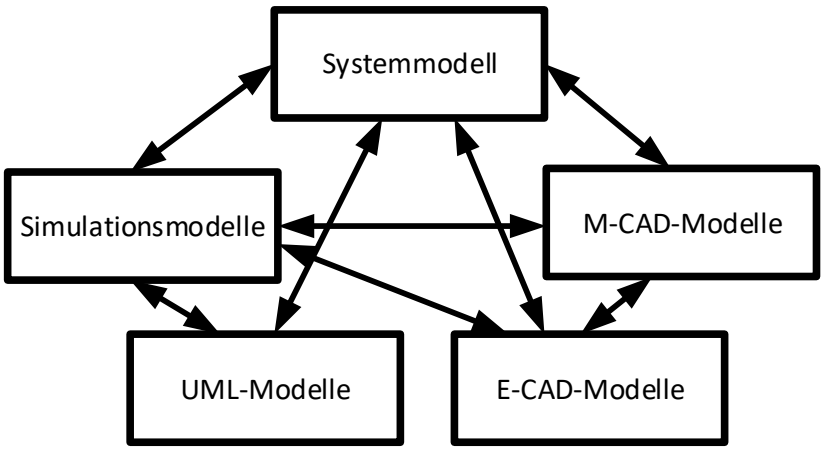


Abbildung 2.18: Digitale Modelle in der Produktentwicklung gemäß [EKM17]

Die aktuellen Herausforderungen des MBSE bestehen darin, die verschiedenen Modellarten konsistent über den Produktlebenszyklus zu verwalten und Modellinformationen bidirektional zwischen den verschiedenen Modellarten zu übertragen, um doppelte Dateneingaben und folglich Inkonsistenzen zu vermeiden. Als eine Lösung für diese Herausforderungen werden Product Lifecycle Management (PLM)-Systeme angesehen. PLM-Systeme sind „IT-Lösungen zur Umsetzung des für den Prozess der Produktentstehung notwendigen Daten-Backbone“ [ES09]. Dem Management von Systemmodellen in PLM-Systemen widmete sich zum Beispiel das Forschungsprojekt mecPro^2, dessen Ergebnisse in [EKM17] veröffentlicht wurden. Allerdings lag der Fokus auf der effizienten Verwaltung von SysML-Modellen in PLM-Systemen. Folglich sind in mecPro^2 keine Lösungen für den bidirektionalen Informationsaustausch zwischen dem Systemmodell und den diszplinspezifischen Modellen entstanden. Einem alternativen Ansatz widmete sich das Forschungsprojekt CRYSTAL [Cry17]. In diesem Forschungsprojekt wurde der Datenaustausch zwischen

den verschiedenen Modellen bzw. zwischen den verschiedenen IT-Systemen, in denen die Modelle verwaltet werden, auf Basis der OSLC-Spezifikation (Open Services for Lifecycle Collaboration) realisiert [OSL17]. Es wurden zwar einige Implementierungen realisiert, dennoch stellt der abschließende Bericht zur Einschätzung des Projektes fest, dass die Kopplung von Modellen und IT-Systemen eine Herausforderung in der Praxis bleibt.

Da in diesen aktuellen Forschungsarbeiten keine grundsätzlichen Lösungen für die eingangs genannte Herausforderung erarbeitet werden konnten, ist es naheliegend zu erwarten, dass diese Herausforderungen auch in der Praxis ungelöst sind. Folglich kann für diese Arbeit nicht davon ausgegangen werden, dass Systemmodelle, die mit Methoden des Systems Engineering entwickelt werden, in die Modelle und Artefakte der domänenspezifischen Disziplinen konsistent übertragen werden. Vielmehr werden die Informationen aus den Systemmodellen, sofern Systemmodelle vorhanden sind, manuell in die domänenspezifischen Modelle und Artefakte übertragen. Die einzelnen Domänen sind in der Praxis nach wie vor relativ autarke Disziplinen im Rahmen des Produktentstehungsprozesses. Dies gilt auch für die Softwareentwicklung. Die in dieser Arbeit getätigte fokussierte Betrachtung des Softwareentwicklungsprozesses als einen domänenspezifischen Entwicklungsprozess mit stark wachsender strategischer Bedeutung ist daher gerechtfertigt.

2.5 Softwareentwicklungsprozess

2.5.1 Definition und Eigenschaften

Ein Softwareentwicklungsprozess hat zum Ziel, in einem vereinbarten zeitlichen Rahmen eine definierte Aufgabenstellung, und zwar die Erstellung eines Softwareproduktes bzw. einer Softwareversion, durch einen bekannten Personenkreis zu erledigen. Um die Chancen zu erhöhen, dass die Entwicklungsziele bezogen auf Qualität, Quantität, Entwicklungsdauer und Kosten (vgl. Teufelsquadrat in Abschnitt 2.1.2) erreicht werden, „sollte jede Softwareentwicklung in einem festgelegten organisatorischen Rahmen erfolgen“ [Bal00]. Dieser organisatorische Rahmen wird „Vorgehensmodell“ genannt.

Ein Metamodell für die Beschreibung von Softwareentwicklungsprozessen bzw. Vorgehensmodellen ist das Software Process Engineering Metamodel (SPEM) [OMG17a]. Motivation für die Erstellung von SPEM war die Notwendigkeit, ein einheitliches Beschreibungsformat für die Vielzahl an existierenden Vorgehensmodellen zu schaffen, da deren unterschiedliche Beschreibungsformate eine Austauschbarkeit und Vergleichbarkeit kaum ermöglichen.

Das SPEM-Metamodell ist in UML modelliert. Es beschreibt mehrere Pakete, die verschiedene Elemente enthalten (Abbildung 2.19). Das Paket *MethodContent* enthält diejenigen

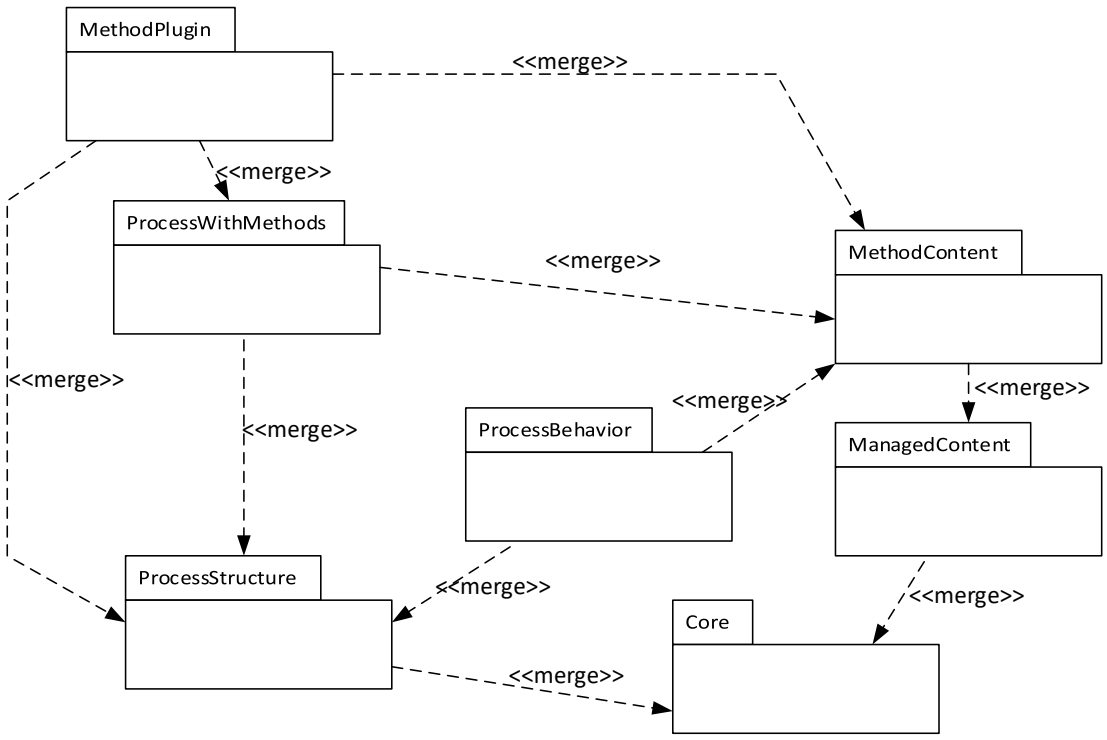


Abbildung 2.19: Struktur des SPEM-Metamodells gemäß [OMG17a]

Elemente, mit denen die Prozesse eines Vorgehensmodells definiert werden. Zur Modellierung von Prozessaufgaben, von Prozessrollen und von Prozessergebnissen sind die in Tabelle 2.4 aufgeführten Elemente vorgesehen.

Obwohl das SPEM-Metamodell zahlreiche weitere Elemente enthält, soll nicht tiefer darauf eingegangen werden, da im weiteren Verlauf dieser Arbeit lediglich auf die drei in Tabelle 2.4 aufgeführten Elemente zur Beschreibung eines Vorgehensmodells Bezug genommen wird.

Eine Studie aus dem Jahr 2013 belegt, dass die Anwendung von Vorgehensmodellen, zumindest in Deutschland, de facto durchgängig etabliert ist [KL14]. Lediglich 2,1 % der befragten Personen gaben an, dass in ihrer Organisation keines der zur Auswahl angebotenen Vorgehensmodelle zur Anwendung kommt. Gleichzeitig wird in dieser Studie deutlich, dass sehr viele verschiedene Vorgehensmodelle angewendet werden: Das Spektrum reicht von agilen Vorgehensmodellen wie zum Beispiel Software-Kanban [Hir08] oder Scrum [Sch95] bis hin zu den als traditionell einzuordnenden Vorgehensmodellen, wie beispielsweise dem Wasserfall-Modell oder dem V-Modell [Boe79].

Traditionelle Vorgehensmodelle werden auch als schwergewichtig bezeichnet [Han10]. Der Grund für diese Attributierung soll am V-Modell erläutert werden:
Das V-Modell ist eine Erweiterung des Wasserfall-Modells mit integrierter Qualitätssicherung durch definierte Verifikations- und Validationsaktivitäten [Bal97]. Abbildung 2.21

Element	Beschreibung
RoleDefinition	Dieses Element beschreibt eine Prozessrolle, die im Softwareentwicklungsprozess aktiv ist. Die Beschreibung umfasst die Fähigkeiten, die Kompetenzen und die Verantwortungen der diese Prozessrolle ausfüllenden Person. Beispiele für jemanden, der diese Position innehaben kann, sind ein Programmierer oder ein Tester.
TaskDefinition	Dieses Element beschreibt eine Prozessaufgabe, die durch eine Prozessrolle erledigt wird. Sie beschreibt die Eingaben der Prozessaufgabe und deren Ausgaben. Beispiele für eine Prozessaufgabe sind die Erstellung eines Dokuments oder eine Programmieraufgabe.
WorkProductDefinition	Dieses Element beschreibt die Ausgabe einer Prozessaufgabe. Dies ist zum Beispiel ein Dokument oder ein Teil eines Quelltextes.

Tabelle 2.4: Ausgewählte Elemente des SPEM-Package *MethodContent* [OMG17a]

zeigt ein Beispiel eines V-Modells. Die Phasen auf der linken Seite des „V“ entsprechen den Phasen des Wasserfall-Modells, wobei die Ergebnisse einer Phase vorliegen müssen, bevor die nächste Phase beginnen kann. Die Ergebnisse einer Phase sind Dokumente. Die Stärken des V-Modells liegen in der abgestuften Vorgehensweise von der Anforderungsspezifikation über das Design bis hin zur Implementierung. Jeder diese Stufen auf der linken Seite des „V“ ist eine dedizierte Phase des Tests auf der rechten Seite des „V“ zugeordnet. Softwareentwicklungen nach dem V-Modell haben einen starken Fokus auf qualitätssichernden Maßnahmen.

Die Schwergewichtigkeit des V-Modells ergibt sich aus diesem dokumentenbasierten Ablauf an Phasen: Dokumente können je nach Softwareentwicklungsprojekt sehr umfangreich sein und folglich kann ihre vollständige Erstellung lange Durchlaufzeiten eines Softwareentwicklungsprozesses bewirken. Außerdem leiten sich die Aufgaben in den weiteren Phasen aus den erstellten Dokumenten ab. Sollten sich während des Softwareentwicklungsprozesses Änderungen zum Beispiel an den Anforderungen ergeben, so bewirken diese Änderungen eine Nachbearbeitung aller abhängigen Dokumente, eine überarbeitete Projektplanung und gegebenenfalls Änderungen der geplanten Endtermine. Die Planung ist nämlich eine zentrale Eigenschaft traditioneller Vorgehensmodelle: Alle Prozessaktivitäten werden inhaltlich und zeitlich geplant, bevor an ihnen gearbeitet wird [Som12].

Bis in den frühen 1990er Jahren war die Ansicht weit verbreitet, dass nur eine sorgfältige Planung die Basis für eine erfolgreiche Softwareentwicklung ist. Allerdings setze

sich zunehmend die Erkenntnis durch, dass der Planungsaufwand traditioneller Vorgehensmodelle für kleinere und mittlere Softwareprojekte zu groß ist und den gesamten Softwareentwicklungsprozess beherrscht [Som12]. Daher wurden agile Vorgehensmodelle vorgeschlagen, die auch leichtgewichtige Vorgehensmodelle genannt werden. Ziel der agilen Vorgehensmodelle ist eine Fokussierung auf die Entwurfs- und Implementierungsaktivitäten. Die zugrundeliegenden Paradigmen sind im *Manifest für Agile Softwareentwicklung* beschrieben [Man17]. Die prinzipielle Vorgehensweise agiler Vorgehensmodelle wird im Folgenden am Beispiel von Scrum erläutert. Die Erläuterungen sind eine Zusammenfassung aus [Som12].

Scrum wurde erstmalig in [Sch95] präsentiert. Das Vorgehen in Scrum besteht aus drei Phasen: einer allgemeinen Planungsphase, einer Serie an Sprint-Zyklen und einer Projektabschlussphase (Abbildung 2.20). In der allgemeinen Planungsphase werden die übergeordneten Ziele des Softwareentwicklungsprojektes und die grobe Softwarearchitektur festgelegt. In der Projektabschlussphase wird das Softwareentwicklungsprojekt vervollständigt, zum Beispiel durch eine Softwaredokumentation, und es erfolgt eine Projektretrospektive, in der die positiven und negativen Aspekte des Projektes aufgearbeitet werden. Das Herzstück von Scrum bilden die Sprint-Zyklen.

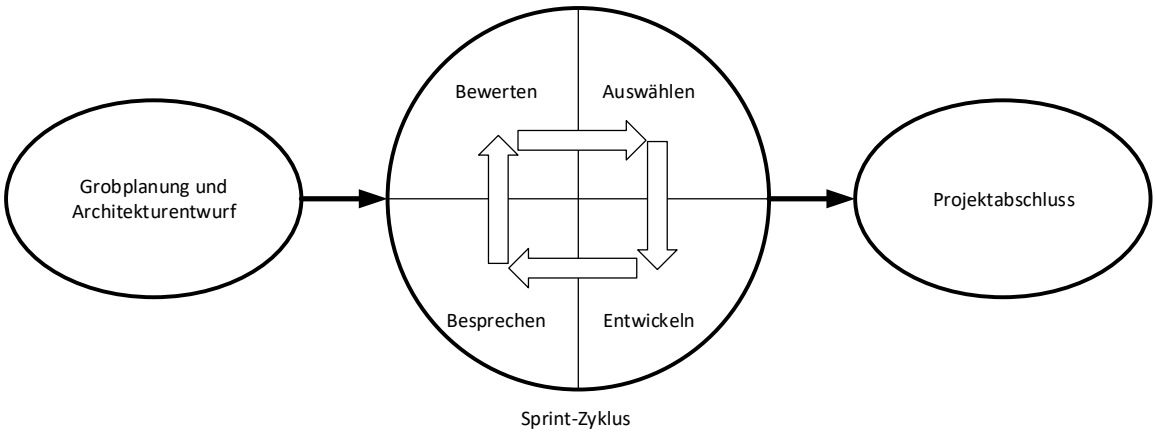


Abbildung 2.20: Die Phasen in Scrum gemäß [Som12]

Ein Sprint ist eine Planungseinheit in der ausgesuchte Arbeitspakete bearbeitet werden. Ein Sprint hat immer eine feste Dauer, meistens zwei oder vier Wochen. Am Ende eines Sprints müssen die Arbeitspakete erledigt sein, so dass immer eine funktionsfähige Software verfügbar ist. Die in einem Sprint zu bearbeitenden Arbeitspakete leiten sich aus dem Produkt-Backlog ab, der Sammlung an Anforderungen an das Softwareprojekt. Die Anforderungen werden Produkt-Backlog-Items genannt. In jedem Sprint werden Produkt-Backlog-Items aus dem Produkt-Backlog ausgewählt und in das Sprint-Backlog überführt. Ein großer Vorteil von Scrum gegenüber den traditionellen Vorgehensmodellen besteht in der Flexibilität der Auswahl der Produkt-Backlog-Items: Im Verlauf eines

Scrum-Projektes wird die Reihenfolge der Bearbeitung der Produkt-Backlog-Items bestimmt, ursprünglich angedachte Produkt-Backlog-Items können verworfen werden und neue, erst während des Projektverlaufs generierte Anforderungen können in das Produkt-Backlog aufgenommen werden. Die Basis für diese Entscheidungen (Reihenfolge, Löschung bzw. Hinzufügen von Anforderungen) ist die funktionsfähige Software, die zum Ende jedes Sprints verfügbar ist und mit den Auftraggebern bzw. Stakeholdern des Softwareentwicklungsprojektes besprochen wird. Die Produkt-Backlog-Items sind folglich zentrale Artefakte der Steuerung eines Scrum-Projektes. Die Idee der Steuerung eines Softwareentwicklungsprozesses mit Hilfe von Items wird in dem später folgenden Entwurf des Sliced V-Modells aufgegriffen.

In [KL14] wird betont, dass die meisten Organisationen das jeweils verwendete Vorgehensmodell an ihre Bedürfnisse adaptieren. Dies wird „tailoring“ genannt. Es kann folglich davon ausgegangen werden, dass jede Organisation, jede Firma oder jeder produzierende Betrieb ein spezifisches Vorgehensmodell anwendet. Dies gilt auch für den Kooperationspartner: Er setzt ein „getailortes“ V-Modell der DIN EN 61508-3 ein. Das V-Modell der DIN EN 61508-3 wird im nächsten Abschnitt erläutert.

In der Softwareentwicklung arbeiten viele Personen zusammen, die unterschiedliche Aufgaben haben, wie zum Beispiel Projektleitung, Programmierung oder Testen. Für die Zusammenarbeit in der Softwareentwicklung werden zunehmend sogenannte „Collaboration Tools“ eingesetzt, die insbesondere in verteilten Softwareentwicklungen benötigt werden [LEPV10, CA01]. Dies sind Werkzeuge, mit denen bestimmte Aufgaben während der Softwareentwicklung durchgeführt werden, zum Beispiel Versionsmanagement, Fehlerverfolgung oder Anforderungsmanagement. „Verteilte Softwareentwicklung“ bedeutet zum einen eine global verteilte Softwareentwicklung über mehrere Länder und zum anderen eine Verteilung über mehrere Standorte, die relativ nahe beieinanderliegen. So finden zum Beispiel die Softwareentwicklungen beim Kooperationspartner an zwei Standorten statt, die lediglich 30 km auseinanderliegen. Selbst Softwareentwicklungen, die an einem Standort stattfinden und bei denen die beteiligten Personen in unterschiedlichen Gebäuden arbeiten, kann als eine verteilte Softwareentwicklung angesehen werden.

Der Zugriff von verteilten Arbeitsplätzen auf alle im Softwareentwicklungsprozess entstehenden Daten, die zeitgleiche Bearbeitung von Dokumenten, E-Mail-Benachrichtigungen bei Änderungen von Daten, die zentrale Speicherung von Daten und vieles mehr sind Grundfunktionen von Collaboration Tools. Sie vereinfachen die Zusammenarbeit in verteilten Softwareentwicklungen, da alle beteiligten Personen jederzeit auf die aktuellsten Daten zugreifen können.

Während sich etliche Collaboration Tools auf eine bestimmte Facette der Softwareentwicklung konzentrieren, also zum Beispiel nur auf das Anforderungsmanagement wie IBM

Rational DOORS [DOO17], haben Systeme für das sogenannte „Application Lifecycle Management“ (ALM-Systeme) das Ziel, den gesamten Softwareentwicklungsprozess zu unterstützen. ALM-Systeme sind werkzeugbasierte Lösungen für die Koordination von Softwareentwicklungsaktivitäten sowie für das Management von Softwareartefakten, zum Beispiel von Produktanforderungen oder von Testfällen [Juk11]. Ein ALM-System kann folglich als ein umfassendes Collaboration Tool angesehen werden.

Da Collaboration Tools in der Regel den Zugriff auf die gespeicherten Daten IT-basiert über API ermöglichen, können Informationsverarbeitungssysteme auf diese Daten zugreifen. Die Nutzung von Collaboration Tools vereinfacht folglich die IT-basierte Erfassung von Softwarekennzahlen.

2.5.2 Softwareentwicklungsprozess beim Kooperationspartner

2.5.2.1 V-Modell der DIN EN 61508-3

Der Softwareentwicklungsprozess beim Kooperationspartner orientiert sich an der DIN EN 61508. Die DIN EN 61508 ist eine aus acht Teilen bestehende Normenreihe für die Entwicklung von sicherheitsgerichteten Produkten. Sicherheitsgerichtete Produkte werden entworfen, um gefahrbringende Zustände zu verhindern [DKE10b]. Ein gefahrbringender Zustand ist ein Zustand, in dem Menschen und/oder Maschinen gefährdet sind. Die Anforderungen an die Softwareentwicklung und das V-Modell sind im dritten Teil der Normenreihe, der DIN EN 61508-3, beschrieben [DKE10a].

Der Kooperationspartner nutzt die DIN EN 61508 seit Beginn der Entwicklung der ersten sicherheitsgerichteten Produkte. In der Softwareentwicklung wurde die Norm zunächst nur für die Entwicklung sicherheitsgerichteter Softwareprodukte verwendet. Später wurde entschieden, das V-Modell der DIN EN 61508-3 auch für die Entwicklung nichtsicherheitsgerichteter Softwareprodukte zu verwenden.

Das V-Modell der DIN EN 61508-3 ist in mehrere Phasen aufgeteilt, in denen definierte Aktivitäten stattfinden (Abbildung 2.21). In der DIN EN 61508-3 heißt es: „Die Ergebnisse der Aktivitäten im Softwaresicherheitslebenszyklus müssen dokumentiert werden“. Die während einer Phase entstandenen Dokumente sind die Ausgaben dieser Phase, die gleichzeitig die Eingaben für die nächste Phase darstellen. Beispiele für Ausgaben sind die Spezifikation der Anforderungen an die Sicherheit der Software und die Spezifikation des Softwaresystementwurfs.

Die inhaltliche Tiefe der Dokumentation wird zwar nicht durch die Norm vorgeschrieben, in der Praxis wird jedoch detailliert dokumentiert. Der Grund dafür ist, dass sicherheitsgerichtete Produkte erst verkauft werden dürfen, nachdem unabhängige Prüfinstitute ein

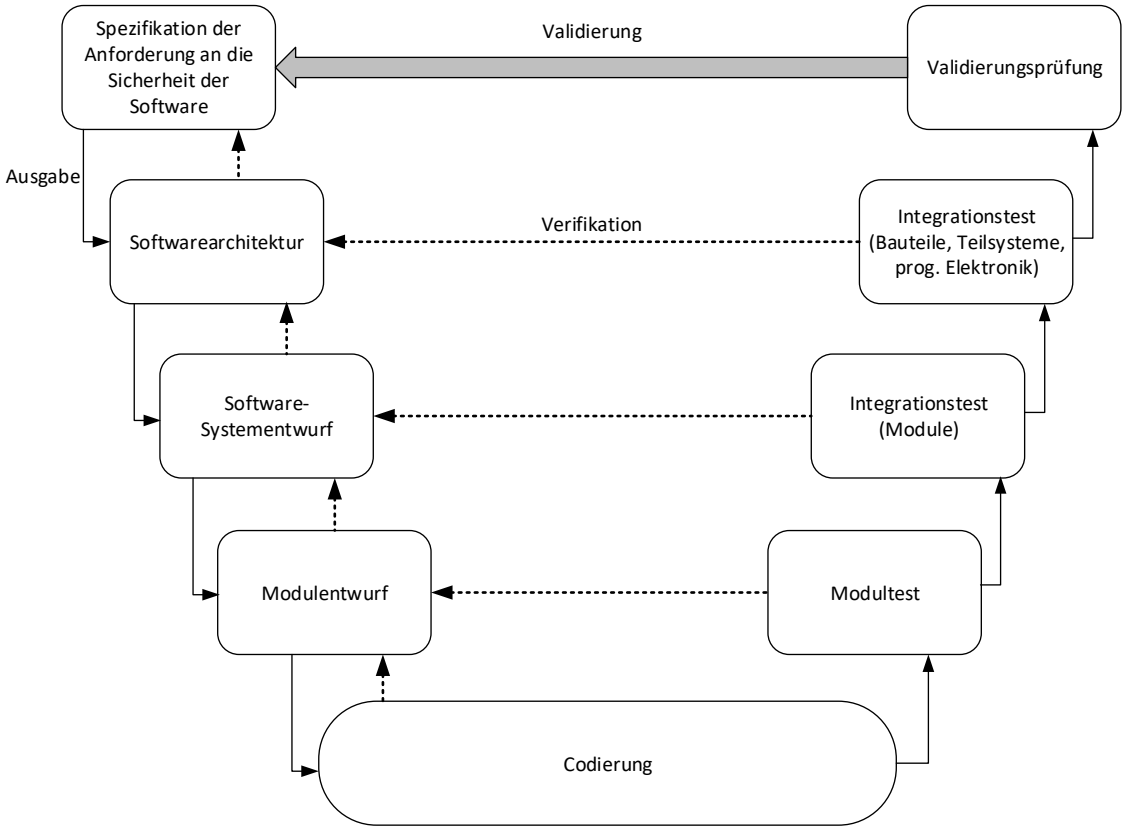


Abbildung 2.21: V-Modell gemäß DIN EN 61508-3 [DKE10a]

Produktzertifikat erteilt haben. Die Prüfinstitute vergeben das Zertifikat erst nach einer Prüfung der Vollständigkeit und Korrektheit der Dokumentation und der Traceability.

Die DIN EN 61508-3 fordert eine Vorwärts- und eine Rückwärtstraceability zwischen allen Eingaben und Ausgaben. Traceability „refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction“ [GF94]. Ein aktives Traceability-Management erhöht die Qualität von Softwareentwicklungen [WP10]. Bei Anwendung des V-Modells der DIN 61850-3 muss eindeutig nachvollziehbar sein, wie die Ergebnisse einer Phase in den Ergebnissen der nächsten Phase berücksichtigt werden, wie zum Beispiel eine Anforderung in der Softwarearchitektur berücksichtigt wird.

2.5.2.2 Anpassungen an das DIN EN 61508-3 V-Modell

Beim Kooperationspartner wurden die von der DIN EN 61850-3 geforderten Dokumente ursprünglich mit MS Word erstellt. Um die Traceability nachzuweisen, wurden einzelne Abschnitte in den Dokumenten mit einer eindeutigen Identifikationsnummer (ID) markiert. Diese IDs wurden in Traceability-Matrizen eingetragen, die mit MS Excel erstellt wurden. Eine Traceability-Matrix zeigt den Zusammenhang zwischen den IDs zweier Dokumente. Tabelle 2.5 zeigt das Konzept einer Traceability-Matrix.

Dokument A: \ Dokument B:	ID101	ID242	ID345	ID556	ID789
ID100		x			
ID223	x				
ID320			x		x
ID894				x	

Tabelle 2.5: Schematisches Konzept einer Traceability-Matrix

Diese Vorgehensweise erwies sich als aufwendig und fehleranfällig, insbesondere wenn mehrere tausend IDs in der Entwicklung eines Softwareproduktes verwaltet werden mussten. Zu jeder Änderung in einem Dokument musste über die Traceability-Matrizen analysiert werden, welche Textpassagen in den nachfolgenden Dokumenten von dieser Änderung betroffen sein könnten. Jede neue ID musste in die Traceability-Matrizen überführt werden. Da diese und ähnliche Bearbeitungsschritte mehrere Minuten dauern konnten, empfanden die Mitarbeiter dieses Vorgehen als mühsam. Außerdem kam es aufgrund der Größe der Dokumente und der Traceability-Matrizen regelmäßig zu Fehlern (zum Beispiel indem ein Kreuz in der Traceability-Matrix falsch gesetzt wurde), sodass zeitintensive Dokumentenreviews durchgeführt werden mussten.

Zudem erwies sich das teamübergreifende Dokumentenmanagement mit MS Word als kompliziert. Die Dokumente wurden in einem Netzwerklaufwerk gespeichert. Die Mitarbeiter wurden nicht automatisch über Änderungen an den Dokumenten informiert, sondern der Mitarbeiter, der eine Änderung vorgenommen hatte, musste die anderen Mitarbeiter informieren. Zudem war ein gleichzeitiges Bearbeiten von Dokumenten nicht möglich.

Aus den aufgeführten Gründen benötigte der Kooperationspartner eine Lösung, um die bestehenden Nachteile in der Anwendung des V-Modells zu beheben. Die Auswahl eines alternativen Vorgehensmodells wurde nicht in Betracht gezogen, da ein Teil der Softwareprodukte zertifiziert wird und Voraussetzung für die Zertifizierung die Anwendung des V-Modells der DIN EN 61508-3 ist. Der Kooperationspartner hatte daher geprüft, wie die aufgeführten Nachteile beseitigt werden könnten. Als Ergebnis dieser Prüfung wurde die Einführung eines ALM-Systems beschlossen und in internen Prozessbeschreibungen wurde definiert, wie das V-Modell der DIN EN 61508-3 in dem ALM-System zu realisieren ist. Des Weiteren wurde im Zuge dessen das Versionsmanagementsystem gewechselt. Mittlerweile werden folgende Collaboration Tools eingesetzt:

- ALM-System: *Polarion ALM* [Sie17]
- Versionsmanagementsystem: *Subversion* [Apa17]

Die Einführung des angepassten V-Modells erfolgte schrittweise über mehrere Jahre. Während seiner Gestaltung wurde die Erfassung von Kennzahlen nicht berücksichtigt, da zu dem Zeitpunkt der Anpassung sämtliche Entwicklungskennzahlen in der im Abschnitt 1.3 erläuterten Projektmanagementdatenbank erfasst wurden. Wie bereits begründet, ist dieses Vorgehen für die Softwareentwicklung nicht mehr ausreichend.

2.6 Zusammenfassung

Dieser Arbeit liegt die Überzeugung zugrunde, dass die Domäne der Softwareentwicklung von der Produktionsdomäne lernen kann und grundsätzlich Methoden und Kennzahlen von der Produktionsdomäne in die Softwaredomäne transferiert werden können.

Kennzahlen sind Zahlen, die in verdichteter Form quantitativ oder qualitativ messbare Sachverhalte wiedergeben. Für ihr Verständnis muss ein Adressat einer Kennzahl deren Semantik verstehen. Da aus der Literatur keine einheitliche Definition für die Semantik einer Kennzahl bekannt ist, wird für diese Arbeit definiert, dass der Name, die Maßeinheit, der Wertebereich, der Idealwert, die Möglichkeit der Vergabe von Soll-Werten, das Ziel, die Frage und die Interpretation die Semantik einer Kennzahl beschreiben.

Kennzahlen werden mit Hilfe von Informationsverarbeitungssystemen erfasst und verarbeitet. Im Entwurf eines Informationsverarbeitungssystems sind verschiedene Gestaltungsgrundsätze zu berücksichtigen, die im Abschnitt 2.1.1 aufgeführt sind.

Es sind keine Beispiele bekannt, in denen Produktionskennzahlen in der Domäne der Softwareentwicklung genutzt werden. Folglich fehlt eine Methode, mit der HW-Produktionskennzahlen in die Domäne der Softwareentwicklung transferiert werden können. Diese Aufgabenstellung ist in dieser Arbeit zu lösen. Es gibt mehrere Methoden bzw. Prozessbeschreibungen, mit denen sich Kennzahlen zielorientiert bestimmen lassen, von denen drei näher betrachtet wurden: die Balanced Scorecard, die Norm ISO/IEC/IEEE 15939 und die GQM-Methode. Grundsätzlich sind alle drei Methoden bzw. Prozessbeschreibungen als Basismethode für eine neu zu entwickelnde Kennzahlentransfermethode geeignet. Da allerdings die GQM-Methode die Anforderungen an eine Basismethode am besten erfüllt, wird sie als Basismethode gewählt. Die neu zu entwickelnde Methode muss gewährleisten, dass die Semantik der HW-Produktionskennzahlen nach dem Kennzahlentransfer erhalten bleibt.

Vorgehensmodelle bilden den organisatorischen Rahmen von Softwareentwicklungsprozessen. Bei der praktischen Umsetzung von Vorgehensmodellen sollten sogenannte Collaboration Tools angewendet werden. Deren Einsatz verbessert die Zusammenarbeit im Softwareentwicklungsprozess und vereinfacht die Datenerfassung mit Hilfe eines Informa-

tionsverarbeitungssystems. Der Kooperationspartner wendet ein auf der DIN EN 61508-3 basierendes V-Modell unter Einbeziehung eines ALM-Systems und eines Versionsmanagementsystem an. Allerdings wurde in der Gestaltung dieses angepassten V-Modells die Erfassung von Kennzahlen nicht berücksichtigt. Die Aufgabenstellung, das V-Modell entsprechend zu gestalten und die Erfassung von SW-Produktionskennzahlen und Softwarekennzahlen zu ermöglichen, ist in dieser Arbeit zu lösen.

Für die Erreichung der in der Abbildung 1.5 gezeigten Zielsituation ist es außerdem notwendig, die Softwarekennzahlen in der Menge K_S aus den operativen Zielen zu bestimmen. Dafür wären grundsätzlich wiederum die Balanced Scorecard, die ISO/IEC/IEEE 15939 und die GQM-Methode geeignet. Da letztere als Basismethode für eine Kennzahlentransfermethode ausgewählt wurde, wird festgelegt, sie auch für die Bestimmung der Softwarekennzahlen zu verwenden, um eine einheitliche Methode für das Lösen mehrerer Aufgabenstellungen dieser Arbeit einzusetzen.

Kapitel 3

Transfer von Produktionskennzahlen

Dieses Kapitel widmet sich der ersten Detailfrage dieser Arbeit: *Wie können SW-Produktionskennzahlen, die die Semantik der äquivalenten HW-Produktionskennzahlen beibehalten, gebildet werden?* Es wird eine auf der GQM-Methode aufbauende Methode entwickelt, mit der Produktionskennzahlen in die Domäne der Softwareentwicklung transferiert werden können. Diese Methode wird Reversed-Goal-Question-Metric-Methode (RGQM-Methode) genannt. Des Weiteren wird ein Anwendungsbeispiel der RGQM-Methode erläutert. Die Inhalte dieses Kapitels entstanden iterativ in den Design Research-Entwurfsphasen *Erstellen* und *Evaluierung* (vgl. Abschnitt 1.5).

Erste Konzepte der RGQM-Methode wurden in [DD15] veröffentlicht und in einem Workshop einem Fachpublikum zur Diskussion gestellt [Deu16]. Abbildung 3.1 zeigt, welche Inhalte in diesem Kapitel behandelt werden und an welcher Stelle diese Inhalte zur Erreichung der Zielsituation dieser Arbeit beitragen. Das graue Rechteck markiert den Inhalt: den Transfer der Kennzahlen in K_{HW} nach K_{SW} mit Hilfe der RGQM-Methode.

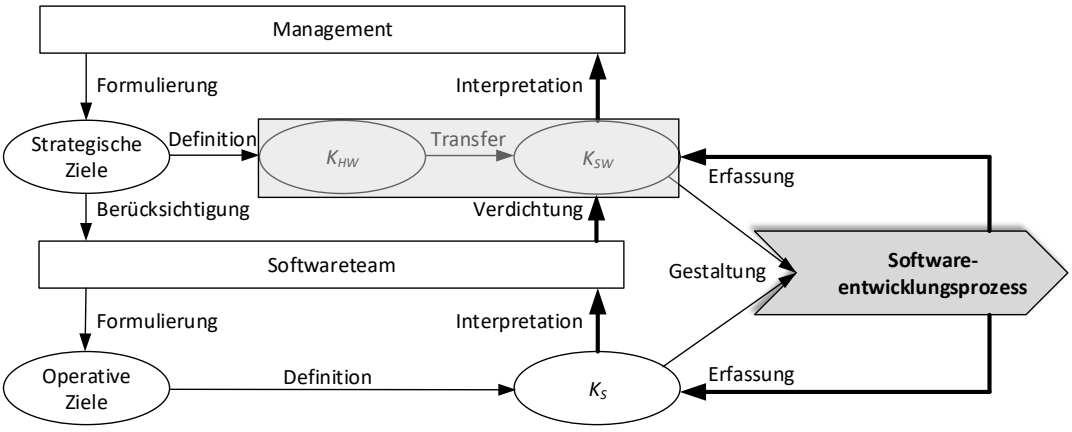


Abbildung 3.1: Inhalt des Kapitels 3 in Bezug auf die Zielsituation dieser Arbeit

3.1 RGQM-Methode

Im Abschnitt 2.3.1 wurde das Grundkonzept des angestrebten Kennzahlentransfers erläutert und in Abbildung 2.10 schematisch dargestellt. Mit einer Basismethode ist eine existierende Methode für die Bestimmung von Kennzahlen gemeint. Damit mit der Basismethode das Grundkonzept realisiert werden kann, muss sowohl ein eindeutiger Zusammenhang zwischen Ziel und Kennzahl als auch zwischen Interpretation und Kennzahl hergestellt werden (vgl. Anforderungen A1 und A2 in Abschnitt 2.3.1).

Für die Auswahl der Basismethode wurden drei Methoden näher betrachtet: die Balanced Scorecard, die Norm ISO/IEC/IEEE 15939 und die GQM-Methode. Wie im Abschnitt 2.3.5 aufgeführt, erfüllen zwar alle drei Methoden beide Anforderungen, die Balanced Scorecard und die Norm ISO/IEC/IEEE 15939 jedoch nur eingeschränkt. Die GQM-Methode erfüllt beide Anforderungen ohne Einschränkung. Sie wird daher als Basismethode für die im folgenden Abschnitt beschriebene Methode zum Kennzahlentransfer ausgewählt.

3.1.1 Konzept

Wie bereits in Abschnitt 2.3.4 erläutert, spielen in der GQM-Methode das Ziel und die Interpretation einer Kennzahl folgende Rollen: das Ziel ist der Ausgangspunkt für die Bestimmung einer Kennzahl, die Interpretation unterstützt bei der Überprüfung der Zielerreichung. Mit Hilfe der in diesem Abschnitt entworfenen RQGM-Methode werden sowohl das Ziel als auch die Interpretation einer HW-Produktionskennzahl in die Domäne der Softwareentwicklung übertragen. Durch diese Übertragung entsteht eine SW-Produktionskennzahl.

Die RQGM-Methode übernimmt das in Abschnitt 2.3.4 beschriebene Phasenmodell der GQM-Methode mit den vier Phasen Planung, Definition, Datenerfassung und Interpretation (Abbildung 3.2). Der Planungsphase und der Interpretationsphase wird das Vorgehen der Planungsphase bzw. der Interpretationsphase der GQM-Methode zugrunde gelegt. Das bedeutet, dass sich in diesen beiden Phasen die RGQM-Methode nicht von der GQM-Methode unterscheidet. In der Datenerfassungsphase werden die SW-Produktionskennzahlen erfasst. Die genaue Implementierung der Datenerfassung hängt von den einzelnen Kennzahlen ab.

In der RGQM-Definitionsphase wird der Weg der zielorientierten Bestimmung von Kennzahlen der GQM-Methode genutzt, jedoch wird dieser Weg zunächst in entgegengesetzter Richtung durchlaufen. Abbildung 3.3 zeigt schematisch die RGQM-Definitionsphase, die eine konkrete Umsetzung des im Abschnitt 2.3.1 erläuterten Grundkonzepts eines Kennzahlentransfers darstellt (vgl. Abbildung 2.10).

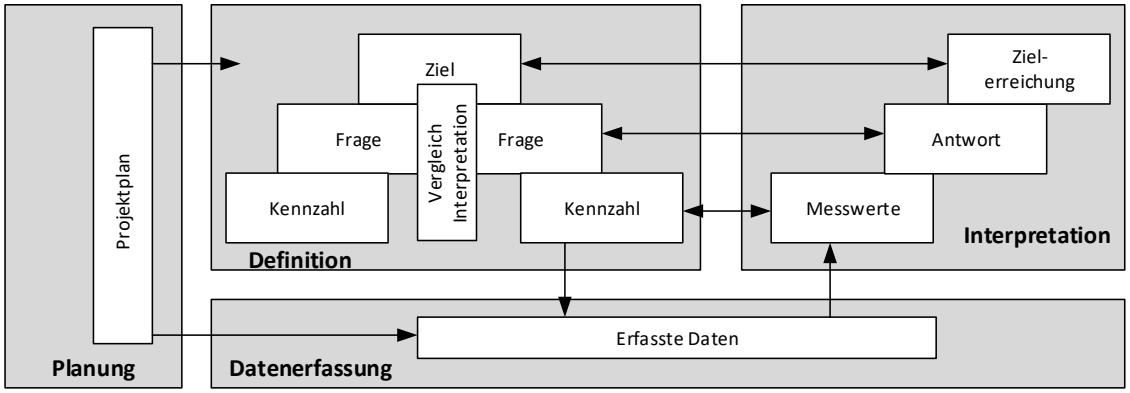


Abbildung 3.2: Phasen der RGQM-Methode

Die gestrichelte Linie symbolisiert das logische Ergebnis des Kennzahlentransfers: Eine existierende HW-Produktionskennzahl k_{HW} wird in eine SW-Produktionskennzahl k_{SW} transferiert. Die durchgezogenen Linien symbolisieren den Abarbeitungspfad des Kennzahlentransfers. Dabei wird ausgehend von k_{HW}die dazugehörende Frage Q_{khw} und das dazugehörende Ziel G_{khw} für den Produktionsprozess ermittelt. Es wird geprüft, ob das Ziel nicht nur für den Produktionsprozess gültig ist, sondern auch auf den Softwareentwicklungsprozess übertragen werden kann (G_{ksw}). Damit ein Ziel übertragen werden kann, sollte es abstrakt formuliert sein.

Wie in Abschnitt 2.1.1 erläutert, ist ein abstraktes Ziel weder terminiert noch quantifiziert. Konkret formulierte Ziele sind dagegen terminiert und quantifiziert und stehen somit in direktem Bezug zur Produktion. Daher ist ein Transfer konkreter Ziele nicht möglich.

Falls das Ziel übertragen werden kann, wird im Anschluss das zielorientierte Bestimmen einer Kennzahl gemäß GQM-Methode angewendet und die Frage Q_{ksw} formuliert. Im

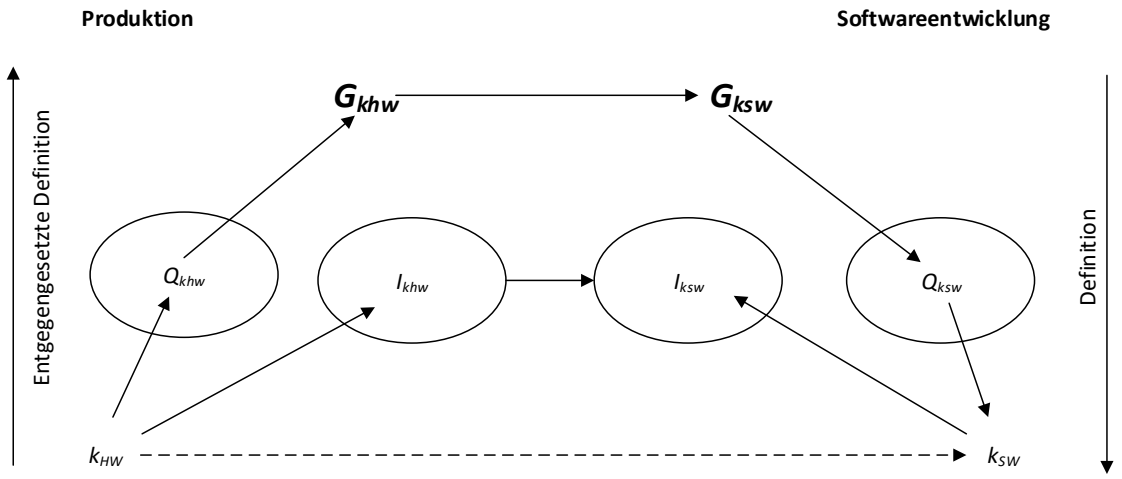


Abbildung 3.3: RGQM-Definitionsphase

Unterschied zur GQM-Methode steht jedoch die Kennzahl bereits fest, mit der die Frage beantwortet wird: Es ist die SW-Produktionskennzahl k_{SW}. Anders als bei der GQM-Methode muss folglich eine neue Kennzahl nicht bestimmt werden. Abschließend wird bewertet, ob die Interpretation I_{ksw} der SW-Produktionskennzahl im Grundsatz identisch zur der Interpretation I_{khw} der HW-Produktionskennzahl ist. Nur wenn all diese Schritte erfolgreich abgearbeitet werden können, ist die Produktionskennzahl transferierbar.

Die RGQM-Definitionsphase setzt sich aus acht aufeinanderfolgenden Bearbeitungsschritten zusammen, die im folgenden Abschnitt erläutert werden.

3.1.2 Bearbeitungsschritte

Abbildung 3.4 ordnet die acht RGQM-Bearbeitungsschritte in die RGQM-Definitionsphase ein. Sie beginnt mit der Identifizierung einer HW-Produktionskennzahl k_{HW} (①) und endet mit der Ermittlung der Interpretation der SW-Produktionskennzahl k_{SW} (⑧).

Für die Durchführung der RGQM-Methode ist ein Team verantwortlich, das in der RGQM-Planungsphase zusammengestellt wird (vgl. Abschnitt 2.3.4). Im Folgenden werden die RGQM-Bearbeitungsschritte zwar im Passiv beschrieben, dennoch ist ein aktive Durchführung der RGQM-Methode durch das Projektteam bzw. durch eine benannte Person gemeint.

Bearbeitungsschritt 1 (Identifizierung der HW-Produktionskennzahl k_{HW}):
Alle im Produktionsprozess eingesetzten HW-Produktionskennzahlen werden gesichtet. Es wird analysiert, welche dieser HW-Produktionskennzahlen das Management verwendet und folglich Teil der Menge K_{HW} sind (vgl. Abbildung 1.3). In der Regel werden nicht

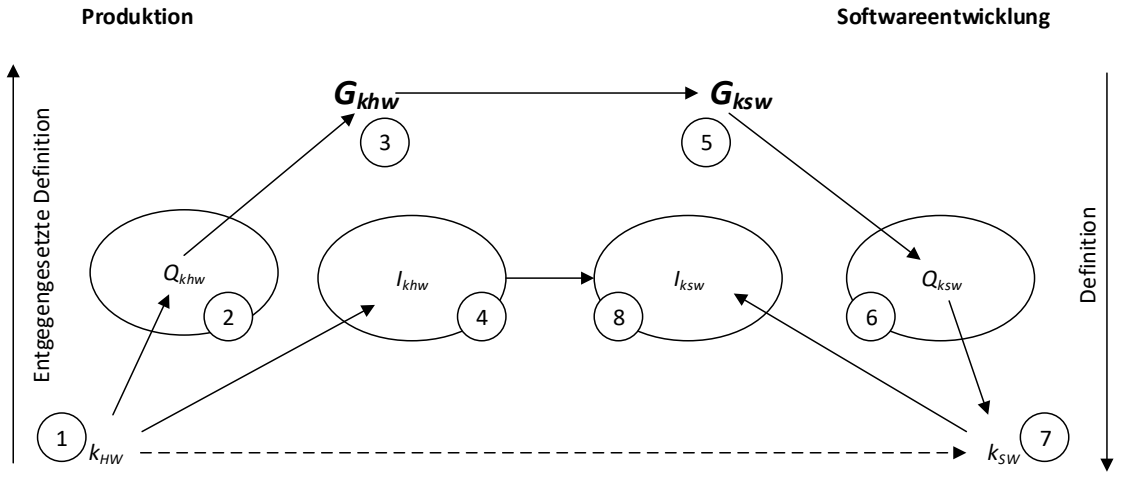


Abbildung 3.4: Bearbeitungsschritte in der RGQM-Definitionsphase

alle in einem produzierenden Betrieb definierten HW-Produktionskennzahlen vom Management genutzt. Dies ist zum Beispiel der Fall, wenn sie ausschließlich der Überprüfung operativer Ziele dienen und folglich Teil der Menge K'_{HW} sind.

Anhand der vom Management verwendeten HW-Produktionskennzahlen ist zu entscheiden, für welche der HW-Produktionskennzahlen der RGQM-Bearbeitungsprozess gestartet wird. Insbesondere wenn die RGQM-Methode erstmalig angewendet wird, sollte die Anzahl der ausgewählten HW-Produktionskennzahlen begrenzt werden, um Erfahrungen in der Anwendung der RGQM-Methode zu sammeln. Zwar stehen dann dem Management noch nicht alle benötigten Kennzahlen zur Steuerung des Softwareentwicklungsprozesses zur Verfügung. Dennoch sind, anders als vor der erstmaligen Anwendung der RQGM-Methode, bereits einige SW-Produktionskennzahlen verfügbar, was als eine Verbesserung der Ausgangssituation angesehen wird.

Bearbeitungsschritt 2 (Identifizierung der zu k_{HW} gehörenden Frage Q_{khw}):
Im nächsten Schritt wird Q_{khw} ermittelt. Dies ist die produktionsspezifische Frage, die mit k_{HW} beantwortet wird. Falls k_{HW} mit der GQM-Methode hergeleitet wurde, existiert bereits die Frage Q_{khw}. Sie kann folglich durch Sichtung der vorhandenen GQM-Unterlagen (Protokolle etc.) ermittelt werden. Da die GQM-Methode ihren Ursprung in der Softwaredomäne hat, ist es zwar unwahrscheinlich, dass die GQM-Methode in der Produktionsdomäne verwendet wird, auszuschließen ist dies jedoch nicht. Falls k_{HW} nicht mit der GQM-Methode hergeleitet wurde bzw. keine Frage existiert, wird die Frage ermittelt.

Bearbeitungsschritt 3 (Identifizierung des zu Q_{khw} gehörenden Ziels G_{khw}):
G_{khw} ist das für den Produktionsprozess formulierte strategische Ziel, dem die Frage Q_{khw} zugeordnet ist. Falls k_{HW} und folglich Q_{khw} ursprünglich mit der GQM-Methode hergeleitet wurden, werden wiederum die existierenden GQM-Unterlagen genutzt, um G_{khw} zu notieren. Andernfalls wird unter Einbindung des Managements das strategische Ziel G_{khw} ermittelt, dem Q_{khw} zugeordnet ist. Falls es zwar strategische Ziele gibt, aber keines davon Q_{khw} zugeordnet werden kann, muss explizit das strategische Ziel G_{khw} formuliert werden. Es wird angenommen, dass die Formulierung eines strategischen Ziels G_{khw} möglich ist. Andernfalls wäre die Nutzung von k_{HW} in Frage zu stellen: Sie wird zwar vom Management genutzt, aber ihre Erfassung dient keinem strategischen Ziel.

Bearbeitungsschritt 4 (Identifizierung der Interpretation I_{khw}):
Aus Abschnitt 2.1.1 ist bekannt, dass die Interpretation keinen eineindeutig beschreibbaren Informationsinhalt einer Kennzahl darstellt. Sie ist vielmehr eine Handlungsbeschreibung des Adressaten der Kennzahl, der auf die Ist-Werte der Kennzahl reagiert. In diesem RGQM-Bearbeitungsschritt wird folglich festgestellt, wie das Management die Ist-Werte der HW-Produktionskennzahl k_{HW} für die Überprüfung der Erreichung des dazugehörenden strategischen Ziels verwendet. Dies können Soll-/Ist-Wert-Vergleiche, Grenzwertüber-

prüfungen oder Trendanalysen sein. Des Weiteren wird ermittelt, wie das Management reagiert, wenn Ziele nicht erreicht werden. So kann es zum Beispiel ad-hoc-Maßnahmen einleiten, wenn die Ist-Werte bestimmte Grenzwerte über- oder unterschreiten.

In diesem RQGM-Bearbeitungsschritt wird identifiziert, welches Verständnis das Management von k_{HW} hat, für wie wichtig es k_{HW} für die Produktionssteuerung erachtet und wie es auf abweichende Ist-Werte reagiert.

Bearbeitungsschritt 5 (Prüfung der Gültigkeit von G_{ksw}):
Mit diesem Schritt beginnt der Transfer von k_{HW} in die Softwaredomäne. Es wird bewertet, ob das zu k_{HW} gehörende strategische Ziel G_{khw} auch, aus Sicht des Managements, auf den Softwareentwicklungsprozess übertragen werden sollte. Falls ja, müssen gegebenenfalls produktionsspezifische Begriffe, sofern sie in G_{khw} verwendet werden, durch softwarespezifische Begriffe ersetzt werden. Folglich wird aus G_{khw} ein für die Softwaredomäne angepasstes Ziel G_{ksw}. Ein Beispiel für ein übertragbares strategisches Ziel mit angepassten domänenspezifischen Begriffen ist:

G_{khw}: Verkürzung der durchschnittlichen Durchlaufzeiten in der *Produktion*
G_{ksw}: Verkürzung der durchschnittlichen Entwicklungsdauer in der *Softwareentwicklung*

Sollte das Management zu der Einschätzung gelangen, dass G_{khw} nicht für den Softwareentwicklungsprozess gilt, ist der RGQM-Bearbeitungsprozess für die HW-Produktionskennzahl k_{HW} beendet. Sie kann nicht in die Softwaredomäne transferiert werden.

Bearbeitungsschritt 6 (Formulierung der softwarespezifischen Frage Q_{ksw}):
Falls G_{khw} auf den Softwareentwicklungsprozess übertragen werden soll, wird im nächsten Schritt die Frage formuliert, die die SW-Produktionskennzahl beantwortet. Es wird bewertet, wie der Satzbau der Frage Q_{khw} in den Satzbau der Frage Q_{ksw} überführt werden kann. Wie im Abschnitt 2.3.1 aufgeführt ist es notwendig, dass der Satzbau der Frage Q_{khw} übernommen wird. Die Frageninhalte in der Frage Q_{ksw}, beispielsweise „Wie ist das Verhältnis von...zu..?“, dürfen dagegen spezifisch für die Domäne der Softwareentwicklung sein. Um die Frageninhalte der Frage Q_{ksw} zu ermitteln, werden die Frageninhalte der Frage Q_{khw} in softwaredomänenspezifische Frageninhalte abgebildet. Dieser Vorgang wird als „Mapping“ bezeichnet. Das Mapping jedes einzelnen Frageninhaltes wird begründet.

Bearbeitungsschritt 7 (Ermittlung der Berechnungsgrundlagen von k_{sw}):
Da k_{hw} bereits im Produktionsprozess erfasst und vom Management verwendet wird, sind die dazugehörenden Berechnungsgrundlagen definiert. Diese sind abhängig von der Kennzahl selbst und von der Frage, ob es sich um eine absolute Kennzahl oder eine Verhältniskennzahl handelt. Da k_{sw} in einer anderen Domäne als k_{hw} erfasst wird, kann die Berechnungsgrundlage für k_{sw} von der Berechnungsgrundlage für k_{hw} abweichen.

Während der Erstellung der Berechnungsgrundlagen wird bewertet, ob die für k_{hw} geltenden Semantikmerkmale *Maßeinheit*, *Wertebereich*, *Idealwert* und *Möglichkeit der Festlegung von Soll-Werten* auf k_{sw} übertragbar sind. Dies erfolgt auf Basis einer konkreten Berechnungsformel. Ist zum Beispiel k_{hw} eine Verhältniskennzahl, die in % angegeben wird und den Wertebereich von 0 % bis 100 % hat, muss die Berechnungsformel für k_{sw} die Übertragung der aufgeführten Semantikmerkmale ermöglichen. In Gleichung 3.1 ist die Berechnungsformel für dieses Beispiel gezeigt.

$$k_{sw} = \frac{A_{cond}}{A_{all}} \tag{3.1}$$

mit:

A_{all}	Alle Artefakte
A_{cond}	Artefakte, für die eine Bedingung gilt

k_{sw} ist ebenfalls eine Verhältniskennzahl, die in % angegeben wird. Es ist naheliegend, dass das Semantikmerkmal *Name* ohne gesonderte Prüfung übertragen wird.

Bearbeitungsschritt 8 (Ermittlung der Interpretation I_{ksw}):
k_{sw} kann transferiert werden, wenn die RGQM-Bearbeitungsschritte 1 bis 7 erfolgreich sind und wenn k_{sw} eine im Grundsatz identische Interpretation erlaubt wie k_{hw}. Wie bereits in Abschnitt 2.3.1 erläutert, ist damit gemeint, dass das Management auf die Soll-Werte von k_{sw} ähnlich reagieren würde wie auf die Soll-Werte von k_{hw}. Um dies zu prüfen, wird mit dem Management erörtert, wie es k_{sw} für die Überprüfung der Zielerreichung strategischer Ziele verwenden würde. Diese Reaktion wird notiert. In deren Beschreibung können softwaredomänenspezifische Begriffe verwendet werden. Dieser zunächst theoretischen Betrachtung folgt eine praktische Evaluierung. Diese ist jedoch erst möglich, wenn ein Informationsverarbeitungssystem verfügbar ist und damit Ist-Werte von k_{sw} erfasst werden.

Nach Abarbeitung aller RGQM-Bearbeitungsschritte hat die SW-Produktionskennzahl k_{sw} die meisten der im Abschnitt 2.1.1 definierten Semantikmerkmale der HW-Produktionskennzahl k_{hw} übernommen und zwar: *Name*, *Maßeinheit*, *Wertebereich*, *Idealwert*, *Möglichkeit der Festlegung von Soll-Werten*, *Ziel* und *Interpretation* (hier mit der Einschränkung im Grundsatz identisch).

Die *Frage*, die mit der SW-Produktionskennzahl beantwortet wird und die ein weiteres Semantikmerkmal darstellt, ist zwar im Satzbau identisch, jedoch können die Frageninhalte verschieden sein. Obwohl folglich eine andere Frage generiert wird, kann ein Manager mit der SW-Produktionskennzahl in gleicher Art und Weise arbeiten, wie mit der dazugehörenden HW-Produktionskennzahl. Dies ist möglich, weil:

- die Fragestellung durch den gleichen Satzbau erhalten bleibt, wobei domänenspezifische Frageninhalte verwendet werden,
- und weil beide Ausprägungen der Produktionskennzahl den Zielen G_{khw} bzw. G_{ksw} zugeordnet sind, wobei G_{ksw} das transferierte G_{khw} ist,
- und weil beide Ausprägungen der Produktionskennzahl im Grundsatz identisch interpretiert werden.

Für die Vertiefung des Verständnisses der RGQM-Methode wird im nächsten Abschnitt ein Anwendungsbeispiel beim Kooperationspartner dargestellt.

3.2 Anwendungsbeispiel

3.2.1 Einführung

In dem Anwendungsbeispiel werden fünf HW-Produktionskennzahlen transferiert. Die folgende Erläuterung orientiert sich an der Reihenfolge der RGQM-Bearbeitungsschritte: In jedem RGQM-Bearbeitungsschritt werden jeweils alle fünf HW-Produktionskennzahlen genannt. Folglich wird nicht für jede einzelne HW-Produktionskennzahl jeweils ein RGQM-Bearbeitungsschritt erläutert.

Zunächst werden die RGQM-Bearbeitungsschritte 1 bis 6 dargestellt. Die RGQM-Bearbeitungsschritte 7 und 8 werden im Abschnitt 6.3 erläutert, da für deren Nachvollziehbarkeit das Datenmodell des Softwareentwicklungsprozesses bekannt sein sollte, welches in Abschnitt 5.2.2 beschrieben wird. Im Laufe des Anwendungsbeispiels entstehen Anforderungen an den Softwareentwicklungsprozess, die in dessen Gestaltung zu berücksichtigen sind. Sie müssen erfüllt werden, damit die SW-Produktionskennzahlen von einem Informationsverarbeitungssystem erfasst werden können.

3.2.2 RGQM-Bearbeitungsschritte

3.2.2.1 RGQM-Bearbeitungsschritt 1

Der Geschäftsbereichsleiter beim Kooperationspartner wurde befragt, welche der in den unternehmensinternen Dokumenten beschriebenen HW-Produktionskennzahlen er im Rahmen seiner aktuellen Managementtätigkeit für die Steuerung und Überwachung von Produktionsprozessen nutzt. Aus den von ihm genannten sieben HW-Produktionskennzahlen wurden die in Tabelle 3.1 gezeigten HW-Produktionskennzahlen ausgewählt.

Nr.	HW-Produktionskennzahl	Beschreibung
$k10.1.1_{hw}$	First Pass Rate [%]	Verhältnis von gefertigten Produkten, die den Fertigungsendtest bestehen, zu allen gefertigten Produkten
$k11.1.1_{hw}$	Technische Rückläuferrate [%]	Verhältnis von aus technischen Gründen von Kunden zurückgeschickten Produkten zu allen gefertigten Produkten
$k12.1.1_{hw}$	Servicegrad [%]	Verhältnis termingerecht gefertigter Auftragspositionen zu allen gefertigten Auftragspositionen
$k13.1.1_{hw}$	Wertschöpfung [€]	Die auf Basis von Arbeitsplänen der Mitarbeiter und Maschinenauslastungen ermittelten Fertigungsplankosten, multipliziert mit der Anzahl gefertigter Produkte
$k13.2.1_{hw}$	Produktivität [€/h]	Verhältnis von Wertschöpfung zu Ist-Stunden der Mitarbeiter

Tabelle 3.1: Auswahl der HW-Produktionskennzahlen

Diese Auswahl basiert zum einen auf der vom Geschäftsbereichsleiter eingeschätzten Wichtigkeit der HW-Produktionskennzahlen. Zum anderen erfolgte die Auswahl aufgrund seines Wunsches, den Einsatz dieser HW-Produktionskennzahlen im Softwareentwicklungsprozess zu evaluieren. Zum Beispiel begründete er den Wunsch an den Servicegrad wie folgt:
Der Geschäftsbereichsleiter sei zwar in der Lage, die Lieferqualität von großen Softwarefunktionen, die in der Regel von Großkunden gefordert werden, einzuschätzen. Jedoch könne er die Lieferqualität der vielen kleineren Softwarefunktionen nicht bewerten, da ihm diese kleineren Softwarefunktionen nicht bekannt seien. Die großen Softwarefunktionen werden den Großkunden zu Terminen geliefert, an deren Vereinbarung der Geschäftsbereichsleiter mitgewirkt hat. Durch regelmäßige Abstimmungen mit den Großkunden und den Softwareentwicklungsteams kann er einschätzen, ob diese Termine tatsächlich eingehalten wurden bzw. wie groß die Lieferverzögerungen sind. Die Erwartung an die SW-Produktionskennzahl Servicegrad sei, dass er damit auch die Lieferqualität der vielen kleinen Softwarefunktionen bewerten könne. Genau dies ist mit der HW-Produktionskennzahl *Servicegrad* für die Produktion möglich, wobei es sich dabei um Auftragspositionen statt um Softwarefunktionen handelt. In ähnlicher Art und Weise äußerte er Wünsche für die anderen ausgewählten HW-Produktionskennzahlen.

Wie im Folgenden aufgeführt, gibt es für die First Pass Rate, für die Technische Rückläuferrate und den Servicegrad je ein zugeordnetes strategisches Ziel ($G10_{khw}$, $G11_{khw}$,

$G12_{khw}$). Die Wertschöpfung und die Produktivität werden einem strategischen Ziel $G13_{khw}$ zugeordnet, beantworten dabei jedoch unterschiedliche Fragen. Die Nummerierung dieser beiden HW-Produktionskennzahlen beginnt daher mit der Ziffer 13. Wie in Abschnitt 4.2 begründet wird, erfolgt eine 10er-Nummerierung, um strategische von operativen Zielen unterscheiden zu können.

Wie bereits in Abschnitt 1.2 erläutert, sind die beim Kooperationspartner eingesetzten HW-Produktionskennzahlen betriebsspezifisch. Damit ist gemeint, dass die Auswahl und ihre Semantik betriebsspezifisch sind. Die fünf HW-Produktionskennzahlen in Tabelle 3.1 bestätigen die Erläuterung: Für die First Pass Rate, die Technische Rückläuferrate und den Servicegrad ist nicht bekannt, ob sie mit diesem Namen und der in der Tabelle 3.1 aufgeführten Beschreibung in anderen produzierenden Betrieben verwendet werden. Die Wertschöpfung und die Produktivität sind Kennzahlen, deren allgemein bekannte Definitionen von den Beschreibungen in Tabelle 3.1 abweichen. In der Erläuterung 2.1 in Abschnitt 2.1.3 wurde bereits die betriebsspezifische Semantik der Wertschöpfung beschrieben, die von der allgemein anerkannten Definition abweicht. Die Produktivität ist allgemein als eine betriebs- bzw. volkswirtschaftliche Kennzahl bekannt, die das Verhältnis von Aufwand und Ergebnis anzeigt. Diese Definition weicht von der Bedeutung der Produktivitätskennzahl beim Kooperationspartner ab.

3.2.2.2 RGQM-Bearbeitungsschritte 2 bis 4

Um die zu den HW-Produktionskennzahlen gehörenden Fragen, Ziele und Interpretationen zu bestimmen, fanden mehrere Interviews mit einem Gruppenleiter der Produktion und dem Geschäftsbereichsleiter statt. Dies erfolgte in drei Schritten: Zunächst wurden die Fragen ermittelt und danach das Ziel formuliert. Im dritten Schritt wurde von den Gesprächsteilnehmern erläutert, wie sie auf die Ist-Werte der HW-Produktionskennzahlen reagieren und so deren Interpretation erfragt.

Die auf diese Art und Weise ermittelten Fragen, Ziele und Interpretationen der einzelnen HW-Produktionskennzahlen werden in den folgenden Abschnitten aufgeführt. Wie in Abschnitt 2.1.1 erläutert, werden die Ziele abstrakt formuliert und repräsentieren die Unternehmensstrategie. Die Ziele sind folglich weder quantifiziert noch terminiert. Eine Quantifizierung bzw. Terminierung erfolgt jeweils im spezifischen Kontext einer Produktionslinie, für die bestimmte Soll-Werte in einer bestimmten zeitlichen Periode, zum Beispiel in einem Monat, zu erreichen sind. Wie in den folgenden Abschnitten ersichtlich wird, ist die Frage, die eine HW-Produktionskennzahl beantwortet, im Prinzip identisch zu deren Beschreibung in Tabelle 3.1. Das ist naheliegend für Kennzahlenbeschreibungen: Die Beschreibung sollte möglichst präzise darstellen, welche Information eine Kennzahl anzeigt und welche Frage folglich durch die Kennzahl beantwortet wird.

3.2.2.2.1 First Pass Rate

Die Frage, die durch die First Pass Rate (FPR) beantwortet wird, lautet:

- $Q10.1_{khw}$: Wie ist das Verhältnis von gefertigten Produkte, die fehlerfrei getestet wurden, zu allen gefertigten Produkten?

Das zur First Pass Rate und zu der Frage gehörende Ziel lautet:

- $G10_{khw}$: Hohe Fertigungsqualität

Die First Pass Rate wird wie folgt interpretiert:

- Wie bereits erwähnt, werden alle gefertigten intelligente Produkte im Gegensatz zu rein mechanischen Produkten, die lediglich stichprobenartig getestet werden, einem Fertigungsendtest unterzogen. Zwar ist eine maximale FPR von 100 % möglich, typische Ist-Werte liegen jedoch zwischen 95 % und 98 %. Da es sich um hochpreisige intelligente Produkte handelt, werden diejenigen Produkte, die den Fertigungsendtest nicht bestehen, inspiziert und wenn möglich nachbearbeitet. Dies verursacht Nacharbeitskosten. Das Management fordert, die Nacharbeitskosten möglichst gering zu halten und legt für jede Produktionsstätte durchschnittliche jährliche Soll-Werte fest. Die Produktionsteams bestimmen daraus die jährlichen Soll-Werte für die einzelnen Produktionslinien in der Produktionsstätte. Wenn die Ist-Werte den Soll-Werten entsprechen, werden keine Maßnahmen eingeleitet. Sind die Ist-Werte geringer als die Soll-Werte, untersuchen das Management bzw. die Produktionsteams die Ursachen und leiten Maßnahmen ein. Ein entsprechendes, auf eine konkrete Produktionslinie bezogenes Beispiel wurde bereits in Tabelle 1.1 in Abschnitt 1.2 gezeigt. Sollten die Ist-Werte für eine Produktionsstätte nicht den Soll-Werten entsprechen, entscheidet das Management gemeinsam mit den Produktionsteams über die Umsetzung strategischer Maßnahmen, beispielsweise die Beschaffung von neuen Maschinen oder die Durchführung von Mitarbeiterqualifizierungsmaßnahmen.

3.2.2.2.2 Technische Rückläuferrate

Die Frage, die durch die Technische Rückläuferrate (TRR) beantwortet wird, lautet:

- $Q11.1_{khw}$: Wie ist das Verhältnis von ausgelieferten Produkten, die aufgrund eines technischen Defekts reklamiert werden, zu allen gefertigten Produkten? (Anmerkung: Es gibt auch andere Gründe für Reklamationen, zum Beispiel eine falsche Lieferung. Diese Reklamationen gehen nicht in die TRR ein.)

Das zur Technischen Rückläuferrate und zu der Frage gehörenden Ziel lautet:

- $G11_{khw}$: Hohe Kundenzufriedenheit

Die Technische Rückläuferrate wird wie folgt interpretiert:

- Die TRR beträgt idealerweise 0 %. Sie liegt meist unter 5 %. Jedes reklamierte Produkt wird inspiziert und wenn möglich repariert. Selbst wenn ein reklamiertes Produkt nicht repariert wird, entstehen Nacharbeitskosten durch die Inspektion. Das Management fordert, die Nacharbeitskosten durch eine niedrige TRR möglichst gering zu halten. Es legt für jede Produktionsstätte durchschnittliche jährliche Soll-Werte fest. Die Produktionsteams leiten daraus die jährlichen Soll-Werte für die einzelnen Produktionslinien in der Produktionsstätte ab. Die Maßnahmen bei Abweichungen von Soll- und Ist-Werten ähneln den Maßnahmen, die bei der Interpretation der FPR aufgeführt wurden.

3.2.2.2.3 Servicegrad

Die Frage, die durch den Servicegrad beantwortet wird, lautet:

- $Q12.1_{khw}$: Wie ist das Verhältnis von Auftragspositionen, die zum Bestätigungstermin geliefert wurden, zu allen Auftragspositionen?

Das zum Servicegrad und zu der Frage gehörende Ziel lautet:

- $G12_{khw}$: Hohe Lieferqualität

Der Servicegrad wird wie folgt interpretiert:

- Die Produktion bestätigt dem Vertrieb die Liefertermine für die eingegangenen Auftragspositionen. Das Management fordert, diese Termine einzuhalten. Im Idealfall beträgt der Servicegrad 100 %. In den meisten Fällen liegen die Werte über 95 %. Das Management legt für jede Produktionsstätte durchschnittliche jährliche Soll-Werte fest. Die Produktionsteams leiten daraus die jährlichen Soll-Werte für die einzelnen Produktionslinien in der Produktionsstätte ab. Bei Abweichungen zwischen Soll- und Ist-Werten findet eine Untersuchung der Ursachen statt. Die Ursachen können sehr unterschiedlich sein. So können zum Beispiel benötigte Bauteile für die Fertigung der Auftragspositionen gefehlt haben oder die Priorität von Auftragspositionen wurde geändert. In Abhängigkeit von den Ursachen werden ggf. geeignete Maßnahmen eingeleitet, zum Beispiel eine Erhöhung des Lagerbestands von Bauteilen.

3.2.2.2.4 Wertschöpfung

Die Frage, die durch die Wertschöpfung beantwortet wird, lautet:

- $Q13.1_{khw}$: Wie hoch sind die Fertigungsplankosten (ohne Berücksichtigung der Materialkosten) der produzierten Ist-Menge?

Das zur Wertschöpfung und zu der Frage gehörende Ziel lautet:

- $G13_{khw}$: Hohe Fertigungsrentabilität

Die Wertschöpfung wird wie folgt interpretiert:

- Die Fertigungsplankosten werden pro Produkttyp ermittelt. Die Wertschöpfung gibt die Fertigungsplankosten der produzierten Ist-Menge an. Die Fertigungsplankosten für ein Produkt werden bei der Gestaltung des Produktlistenpreises berücksichtigt. Sie müssen zur Wettbewerbsfähigkeit des Produktes beitragen und eine Marge pro Produkt erlauben. Daher werden die Fertigungsplankosten bzw. die zugrundeliegenden Fertigungsabläufe kontinuierlich angepasst, um die Wertschöpfung zu senken und somit die Wettbewerbsfähigkeit des betreffenden intelligenten Produktes zu erhöhen. Diese Anpassungen erfolgen während der gesamten Lebenszeit des intelligenten Produktes. Die Wertschöpfung ist zudem eine Eingangskennzahl für die Berechnung der Produktivität.

3.2.2.2.5 Produktivität

Die Frage, die durch die Produktivität beantwortet wird, lautet:

- $Q13.2_{khw}$: Wie ist das Verhältnis von der Wertschöpfung zu den Ist-Stunden, die der Fertigung des Produktes direkt zugeordnet werden können?

Das zur Produktivität und zu der Frage gehörende Ziel ist identisch mit dem der Wertschöpfung und lautet:

- $Q13_{khw}$: Hohe Fertigungsrentabilität

Die Produktivität wird wie folgt interpretiert:

- Der Verlauf der Produktivität wird in Trends angezeigt. Der Trend sollte gleichbleibend oder steigend sein. Sinkende Ist-Werte der Produktivität sind ein Indikator dafür, dass die Produktionsmitarbeiter einen wachsenden Teil ihrer Arbeitszeit nicht wertschöpfend einsetzen. In diesem Fall wird untersucht, welche Ursachen es dafür gibt. Diese Ursachen werden gezielt adressiert und beseitigt.

3.2.2.3 RGQM-Bearbeitungsschritt 5

Nach der Erfassung der Fragen, Ziele und Interpretation der ausgewählten HW-Produktionskennzahlen wurde mit dem Geschäftsbereichsleiter bewertet, ob die jeweiligen Ziele auf die Softwareentwicklung übertragen werden können. Dies wurde vom Geschäftsbereichsleiter bestätigt: Alle ermittelten Ziele sind für den Kooperationspartner zentrale strategische Ziele, die sowohl für die Produktion als auch für die Softwareentwicklung gelten. Lediglich das Wort *Fertigung* müsse durch das Wort *Softwareentwicklung* ausgetauscht werden. Daraufhin wurden folgende strategischen Ziele für die Softwareentwicklung formuliert:

- $G10_{ksw}$: Hohe Softwareentwicklungsqualität
- $G11_{ksw}$: Hohe Kundenzufriedenheit
- $G12_{ksw}$: Hohe Lieferqualität
- $G13_{ksw}$: Hohe Softwareentwicklungsrentabilität

Wie ersichtlich ist, wird die Abstraktionsebene der Zielformulierung nicht geändert. Das heißt, die Ziele für die Softwareentwicklung sind wie die Ziele für die Produktion weder quantifiziert noch terminiert. Eine Quantifizierung bzw. Terminierung erfolgt wiederum jeweils in einem spezifischen Kontext, zum Beispiel im Kontext der Entwicklung einer Softwareversion.

3.2.2.4 RGQM-Bearbeitungsschritt 6

Da die Ziele übertragen werden können, wurden im nächsten Bearbeitungsschritt die von diesen Zielen ableitbaren softwaredomänenspezifischen Fragen formuliert, die von diesen Zielen abgeleitet werden können. Beantwortet werden diese Fragen mit den jeweiligen SW-Produktionskennzahlen, also der First Pass Rate, der Technischen Rückläuferrate usw.

Wie in Abschnitt 2.3.1 erläutert, müssen der Satzbau der ursprünglichen Frage und der Satzbau der neuen Frage identisch sein, um die Semantik der HW-Produktionskennzahl

zu erhalten. Dies war bei allen Fragen möglich und wurde berücksichtigt. Die Frageninhalte müssen allerdings durch softwaredomänenspezifische Begriffe ersetzt werden. Dabei wurde jeweils ein Frageninhalt einer Ausgangsfrage in einen Frageninhalt einer Zielfrage gemappt. Die folgende Auflistung zeigt und erläutert das festgelegte Mapping:

- Gefertigte Produkte, die den Fertigungsendtest bestehen: Dieser Frageninhalt wird auf die *Quelltextänderungen, die der Implementierung von Softwarefunktionen zugeordnet werden können*, gemappt. Idealerweise bestehen alle gefertigten Produkte den Fertigungsendtest und alle Quelltextänderungen dienen der Implementierung neuer Softwarefunktionen. Folglich symbolisieren beide Frageninhalte Prozessaktivitäten, die im ersten Anlauf erfolgreich realisiert wurden. Allerdings gibt es fehlerhaft gefertigte Produkte und Quelltextimplementierungen, die der Behebung von Fehlern zuzuordnen sind. Beides verursacht zu vermeidende Nacharbeitskosten.

- Alle gefertigten Produkte: Dieser Frageninhalt wird auf *alle Quelltextänderungen* gemappt. Darin sind Quelltextänderungen, die Fehlerbehebungen zugeordnet werden können, enthalten.

- Aus technischen Gründen von Kunden zurückgeschickte Produkte: Dieser Frageninhalt wird auf *Quelltextänderungen, die der Implementierung extern entdeckter Fehler zugeordnet werden können*, gemappt. Beide Frageninhalte verursachen zu vermeidende Nacharbeitskosten.

- Termingerecht gefertigte Auftragspositionen: Dieser Frageninhalt wird auf die *Softwarefunktionen, die zum zugesagten Termin geliefert wurden*, gemappt. Eine Softwarefunktion wird als Äquivalent einer Auftragsposition angesehen und sollte wie eine Auftragsposition termingerecht geliefert werden.

- Alle gefertigten Auftragspositionen: Dieser Frageninhalt wird auf *alle Softwarefunktionen* gemappt. Darin sind die nicht zum zugesagten Termin gelieferten Softwarefunktionen enthalten.

- Fertigungsplankosten der Ist-Menge: Dieser Frageninhalt wird auf die *geplanten Kosten für die Entwicklung neuer Softwarefunktionen* gemappt. Die geplanten Softwareentwicklungskosten werden als Äquivalent der Fertigungsplankosten angesehen, da in letzteren die Materialkosten nicht enthalten sind.

- Ist-Stunden der Mitarbeiter: Dieser Frageninhalt wird auf die *Ist-Stunden, die der Entwicklung der Softwareversion direkt zugeordnet werden können*, gemappt. Beide Arten von Ist-Stunden zeigen den tatsächlichen Aufwand der beteiligten Mitarbeiter an und werden daher als äquivalent angesehen.

Auf Basis dieses Mappings wurden im Anschluss folgende Fragen formuliert (in den Klammern steht die jeweils dazugehörende SW-Produktionskennzahl):

- $Q10.1_{ksw}$(First Pass Rate): Wie ist das Verhältnis von Quelltextänderungen, die der Implementierung von Softwarefunktionen zugeordnet werden können, zu allen Quelltextänderungen?
- $Q11.1_{ksw}$(Technische Rückläuferrate): Wie ist das Verhältnis von Quelltextänderungen, die der Implementierung extern entdeckter Fehler zugeordnet werden können, zu allen Quelltextänderungen?
- $Q12.1_{ksw}$(Servicegrad): Wie ist das Verhältnis von Softwarefunktionen, die zum zugesagten Termin geliefert wurden, zu allen gelieferten Softwarefunktionen?
- $Q13.1_{ksw}$(Wertschöpfung): Wie hoch sind die geplanten Kosten für die Entwicklung neuer Softwarefunktionen?
- $Q13.2_{ksw}$(Produktivität): Wie ist das Verhältnis von der Wertschöpfung zu den geleisteten Ist-Stunden, die der Entwicklung der Softwareversion direkt zugeordnet werden können?

Um diese Fragen mit den jeweiligen SW-Produktionskennzahlen beantworten zu können, muss der Softwareentwicklungsprozess folgende Anforderungen erfüllen, die in dessen Gestaltung zu berücksichtigen sind:

Anforderung 1 (Zuordnung Quelltextänderungen) *Der Softwareentwicklungsprozess muss es ermöglichen, Quelltextänderungen sowohl der Implementierung neuer Softwarefunktionen als auch der Behebung intern und extern entdeckter Fehler zuzuordnen und diese Zuordnungen zu erfassen.*

Anforderung 2 (Termine Softwarefunktionen) *Der Softwareentwicklungsprozess muss es ermöglichen, neue Softwarefunktionen individuell mit einem zugesagten Termin zu markieren und diesen zugesagten Termin sowie den Freigabetermin der Softwarefunktion zu erfassen.*

Anforderung 3 (Soll-/Ist-Stunden) *Der Softwareentwicklungsprozess muss es ermöglichen, die Soll-Aufwände für die Entwicklung von Softwarefunktionen und die Ist-Aufwände, die für die Entwicklung einer Softwareversion geleistet wurden, zu erfassen.*

Es reicht aus, die Aufwände zu erfassen, da sich die Kosten direkt aus den Aufwänden bestimmen lassen.

Die Erfüllung dieser Anforderungen durch den Softwareentwicklungsprozess ist die Voraussetzung dafür, dass ein Informationsverarbeitungssystem die fünf SW-Produktionskennzahlen erfassen kann.

3.2.2.5 RGQM-Bearbeitungsschritte 7 und 8

Im RGQM-Bearbeitungsschritt 7 werden die Berechnungsgrundlagen der SW-Produktionskennzahlen erstellt. Die Berechnungsgrundlagen der in diesem Kapitel bestimmten SW-Produktionskennzahlen werden allerdings erst in Abschnitt 6.3 aufgeführt. Um diese nachzuvollziehen, sollte das Datenmodell des Softwareentwicklungsprozesses bekannt sein, welches in Abschnitt 5.2.2 erläutert wird. Im RGQM-Bearbeitungsschritt 8 wird überprüft, ob die SW-Produktionskennzahlen eine im Grundsatz identische Interpretation wie die der jeweiligen HW-Produktionskennzahlen erlauben. Die Interpretation jeder einzelnen SW-Produktionskennzahl wird gemeinsam mit deren Berechnungsgrundlagen in Abschnitt 6.3 aufgeführt. Zudem wird in Abschnitt 6.3 die semantische Äquivalenz der beiden Ausprägungen einer Produktionskennzahl bewertet. Um diese Bewertung nachvollziehen zu können, sollte wiederum das Datenmodell des Softwareentwicklungsprozesses bekannt sein.

Während der Anwendung der RQGM-Methode wurden die Anforderungen A1 bis A3 formuliert, die der Softwareentwicklungsprozess beim Kooperationspartner erfüllen muss. Weitere Anforderungen ergeben sich in dem Prozess der Bestimmung von Softwarekennzahlen, dem sich das nächste Kapitel widmet.

Kapitel 4

Bestimmung der Softwarekennzahlen

Für die in dieser Arbeit zu erreichende Zielsituation der kennzahlenorientierten Gestaltung des Softwareentwicklungsprozesses sind neben den SW-Produktionskennzahlen auch Softwarekennzahlen zu bestimmen. Während die SW-Produktionskennzahlen das Management adressieren, erfüllen die Softwarekennzahlen die Informationsbedürfnisse der Softwareteams. In diesem Kapitel wird das prinzipielle Vorgehen für die Bestimmung der Softwarekennzahlen und ein Anwendungsbeispiel beim Kooperationspartner erläutert. In dem Anwendungsbeispiel werden weitere Anforderungen formuliert, die in der Gestaltung des Softwareentwicklungsprozesses berücksichtigt werden müssen. Die Inhalte dieses Kapitels entstanden iterativ in den Design Research-Entwurfsphasen *Erstellen* und *Evaluierung* (vgl. Abschnitt 1.5).

Abbildung 4.1 zeigt, welche Inhalte in diesem Kapitel behandelt werden und an welcher Stelle diese Inhalte zur Erreichung der Zielsituation dieser Arbeit beitragen. Die graue Fläche markiert den Inhalt: die Herleitung der operativen Ziele und die Bestimmung der Softwarekennzahlen.

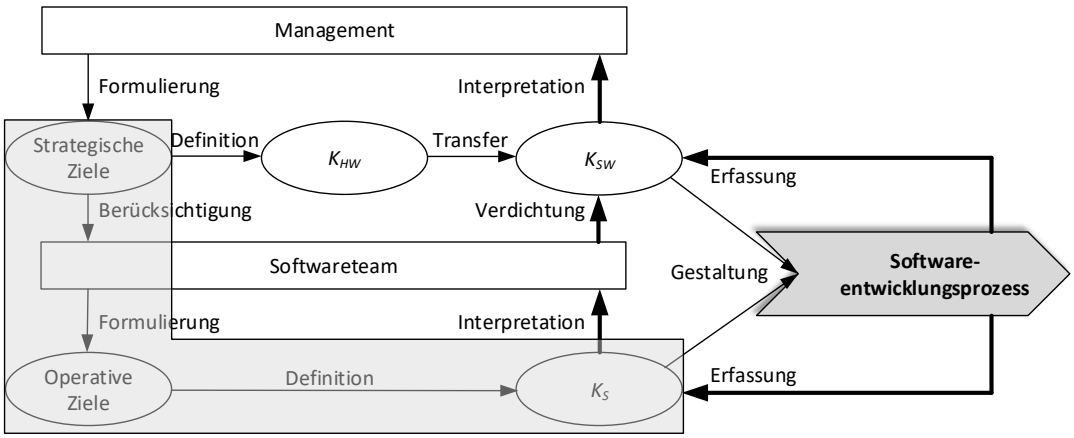


Abbildung 4.1: Inhalt des Kapitels 4 in Bezug auf die Zielsituation dieser Arbeit

4.1 Vorgehen

Ausgangspunkt für die kennzahlenorientierte Gestaltung des Softwareentwicklungsprozesses sind die die vom Management formulierten strategischen Ziele. Um diese zu operationalisieren, werden daraus unter Einbindung der Softwareteams operative Ziele abgeleitet. Wie bereits in Abschnitt 2.1.1 dargelegt, sind die strategischen Ziele im Kontext dieser Arbeit von abstrakter Natur und weder quantifiziert noch terminiert. Ein Beispiel ist das Ziel „Hohe Fertigungsqualität“. Für diese Arbeit wird diese abstrakte Formulierung der strategischen Ziele ebenfalls für die Formulierung der operativen Ziele verwendet. Ein im Folgenden aufgeführtes Anwendungsbeispiel lautet „Leistungseffiziente Programmierung“.

Um strategische in operative Ziele zu überführen, können verschiedene Ansätze gewählt werden, von denen drei im Folgenden kurz dargestellt werden:

- Hierarchische Balanced Scorecards (vgl. Abschnitt 2.3.2): In [FS99] wird dargelegt, dass jeweils eine individuelle Balanced Scorecard für jede Hierarchieebene eines produzierenden Betriebes erarbeitet werden kann. Dabei sind die strategischen Ziele in der Balanced Scorecard der Managementebene und die operativen Ziele in den Balanced Scorecards der unteren Hierarchieebenen aufgeführt. So entsteht ein Netz miteinander verwobener Balanced Scorecards. Bei Anwendung dieses Ansatzes müssten die Softwareteams, also die Hierarchieebene der Softwareentwicklung, ebenfalls eine eigene Balanced Scorecard entwickeln.

- GQM$^+$Strategies® (vgl. Abschnitt 2.3.4): Dieses ist eine auf der GQM-Methode aufbauende Methode, mit der Ziele über mehrere Unternehmenshierarchien verknüpft werden [BHL$^+$07]. Durch die Anwendung von GQM$^+$Strategies® entsteht ein Modell, das den Zusammenhang aller Ziele, folglich auch der strategischen und operativen Ziele, und die für die Zielüberprüfung notwendigen Messaktivitäten zeigt [HMT09]. Bei Anwendung dieses Ansatzes können die Softwareteams schrittweise die operativen Ziele aus den strategischen Zielen ableiten.

- Pragmatische Vorgehensweise: Neben der Anwendung eines dieser beiden methodischen Ansätze kann die Ableitung der operativen Ziele aus den strategischen Zielen auch pragmatisch erfolgen, indem die Softwareteams unter Kenntnis der strategischen Ziele ihre operativen Ziele definieren. Dieser Definitionsprozess erfolgt mit geeigneten, im Betrieb bewährten Methoden, beispielsweise der Durchführung von Workshops.

Welche der genannten Methoden anzuwenden ist, hängt von den Erfahrungswerten eines produzierenden Betriebes im Umgang mit diesen Methoden ab. Damit ist gemeint,

dass ein produzierender Betrieb den Ansatz hierarchischer Balanced Scorecards nutzen kann, sofern bereits Erfahrungen mit der Balanced Scorecard auf einer Hierarchieebene vorhanden sind. Liegen schon Erfahrungen mit der GQM-Methode vor, kann die darauf aufbauende GQM$^+$Strategies® genutzt werden.

Ziel dieser Arbeit ist es nicht, diese oder andere geeignete Methoden für die Überführung von strategischen in operative Ziele zu bewerten. Daher erfolgt hier kein Vorschlag für oder gegen eine der drei genannten Möglichkeiten. Es wird allerdings betont, dass der Prozess der Zielableitung für die kennzahlenorientierte Gestaltung des Softwareentwicklungsprozesses notwendig ist.

Unabhängig davon, welcher Ansatz von einem produzierenden Betrieb gewählt wird, stehen als Ergebnis des Ansatzes die formulierten operativen Ziele zur Verfügung. Um sicherzustellen, dass für alle strategischen Ziele jeweils mindestens ein operatives Ziel formuliert wurde bzw. dass jedes operative Ziel einen Bezug zu einem strategischen Ziel hat, sollte die Relation der Ziele grafisch dargestellt werden. Eine exemplarische Darstellung dieses Zusammenhangs zeigt die Abbildung 4.2.

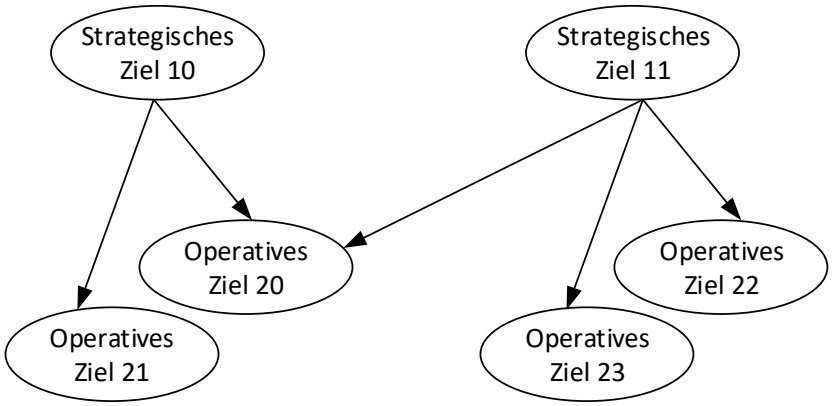


Abbildung 4.2: Relation zwischen strategischen und operativen Zielen

Damit in einer solchen Grafik schnell erkennbar ist, ob es sich um ein strategisches oder um ein operatives Ziel handelt, ist eine Syntax zu definieren, die eine Unterscheidung ermöglicht. Beispielsweise können unterschiedliche Symbole, G_S vs. G_O, oder jeweils ein anderer Nummernkreis genutzt werden. Letzteres ist in der Abbildung 4.2 der Fall.

Da sich sowohl strategische als auch operative Ziele ändern können, sollte die Relation zwischen diesen beiden Zieltypen in regelmäßigen zeitlichen Abständen, beispielsweise einmal pro Jahr, bewertet und ggf. überarbeitet werden.

Der Formulierung der operativen Ziele für die Softwareentwicklung folgt die Bestimmung der Softwarekennzahlen, mit denen die Zielerreichung der operativen Ziele überprüft werden kann. Für diesen Prozess der Softwarekennzahlenbestimmung sollten anerkannte Me-

thoden wie die GQM-Methode (vgl. Abschnitt 2.3.4) oder die Norm ISO/IEC/IEEE 15939 (vgl. Abschnitt 2.3.3) angewendet werden. Nach Anwendung dieser Vorgehensweise stehen dem produzierenden Betrieb die Softwarekennzahlen zur Verfügung, deren Erfassung durch den Softwareentwicklungsprozess ermöglicht werden muss. In dem Prozess der Kennzahlenbestimmung sollte abgewägt werden, ob bereits existierende Softwarekennzahlen (vgl. Abschnitt 2.1.2) für die Überprüfung der Zielerreichung geeignet oder ob betriebsspezifische Softwarekennzahlen neu zu definieren sind.

4.2 Anwendungsbeispiel

In diesem Abschnitt wird die Bestimmung der Softwarekennzahlen beim Kooperationspartner erläutert. Zunächst wird dargelegt, wie die operativen Ziele bestimmt wurden, und die Relation zwischen den operativen Ziele zu den strategischen Zielen wird hergestellt. Im Anschluss wird das Vorgehen bei der Bestimmung der Softwarekennzahlen erläutert und die in diesem Prozess bestimmten Softwarekennzahlen werden aufgeführt.

4.2.1 Bestimmung der operativen Ziele

Um die operativen Ziele aus den strategischen Zielen beim Kooperationspartner herzuleiten, wurde eine pragmatische Vorgehensweise zugrunde gelegt. Die strategischen Ziele sind diejenigen Ziele, die mit Hilfe der RGQM-Methode ermittelt wurden (vgl. Abschnitt 3.2.2.3). Dies sind:

- $G10_{ksw}$: Hohe Softwareentwicklungsqualität
- $G11_{ksw}$: Hohe Kundenzufriedenheit
- $G12_{ksw}$: Hohe Lieferqualität
- $G13_{ksw}$: Hohe Softwareentwicklungsrentabilität

Aus diesen Zielen hat der Verfasser dieser Arbeit zunächst mehrere operative Ziele abgeleitet und diese mit einigen Stakeholdern aus der Softwareentwicklung besprochen. Als Ergebnis dieser Diskussionen wurden fünf operative Ziele für die Softwareentwicklung formuliert.

Um die Nummern für die strategischen und operativen Ziele zu vergeben, wird das oben erwähnte Prinzip eines unterschiedlichen Nummernkreises genutzt: Die strategischen Ziele erhalten eine 10er-Nummer, den operativen Zielen wird eine 20er-Nummer zugewiesen.

Im Folgenden werden die definierten operativen Ziele für die Softwareentwicklung genannt und erläutert:

$G20_{ks}$ Leistungseffiziente Programmierung:
Die Programmierung ist eine Hauptaufgabe in der Softwareentwicklung. Mit dem Begriff „leistungseffizient“ ist ein möglichst ausgewogenes, sich kontinuierlich verbesserndes Verhältnis der Softwarequantität zu den Kosten und zur Entwicklungsdauer gemeint.

$G21_{ks}$ Leistungseffizientes Dokumentieren:
Das Dokumentieren ist eine wesentliche Aufgabe im Softwareentwicklungsprozess (vgl. Abschnitt 2.1.2.1). Wie bei den Entwicklungsaktivitäten ist auch hier mit dem Begriff „leistungseffizient“ ein möglichst ausgewogenes, sich kontinuierlich verbesserndes Verhältnis der Quantität der erstellten Softwaredokumentation zu den Kosten und zur Entwicklungsdauer gemeint.

$G22_{ks}$ Gezielte Bearbeitung von Qualitätsschwerpunkten:
Bislang sind beim Kooperationspartner Qualitätsschwerpunkte nicht systematisch erkennbar. Qualitätsschwerpunkte zeigen auf, welche Softwarequalitätseigenschaften (vgl. Abschnitt 2.1.2.4.2) in einem Softwareprodukt zu verbessern sind, mit welcher Priorisierung diese Verbesserungen zu bearbeiten sind und welche Ursachen in der Programmierung für Softwarequalitätseinschränkungen verantwortlich sind. Bislang erfolgen derartige Einschätzungen unsystematisch in Teammeetings, indem die Teammitglieder beispielsweise einschätzen, dass einige Berechnungszeiten zu lange dauern. Sie ändern daraufhin das Softwareprodukt, um diese Einschränkungen zu beheben. Diesen Einschätzungen fehlt jedoch eine quantitativ auswertbare Datenbasis.

$G23_{ks}$ Objektive Bewertung der Fehlermenge:
Entdeckte Fehler führen regelmäßig zu Diskussionen zwischen Vertriebs- und Entwicklungsmitarbeitern. Damit ist gemeint, dass Vertriebsmitarbeiter immer wieder wegen einzelnen von Kunden entdeckten Fehlern die Leistungsfähigkeit der Softwareentwicklung in Frage stellen. Objektiv nachweisbare Qualitätsverbesserungen, mit denen derartige Diskussionen versachlicht werden könnten, fehlen.

$G24_{ks}$ Reduzierung der Fehlermenge:
Die Anzahl an Fehlern zu reduzieren, ist eine kontinuierliche Aufgabe im Softwareentwicklungsprozess. Bislang ist es möglich, die Anzahl entdeckter Fehler zu quantifizieren. Allerdings ist diese Anzahl nur aussagekräftig, wenn sie in Relation zur Softwarequantität gesetzt wird. Bislang ist es nicht möglich, diese Relation herzustellen.

Alle operativen Ziele stehen in Relation zu einem oder mehreren strategischen Zielen, wobei es für jedes strategische Ziel mindestens ein operatives Ziel gibt. Abbildung 4.3 zeigt die Zielrelationen des Anwendungsbeispiels.

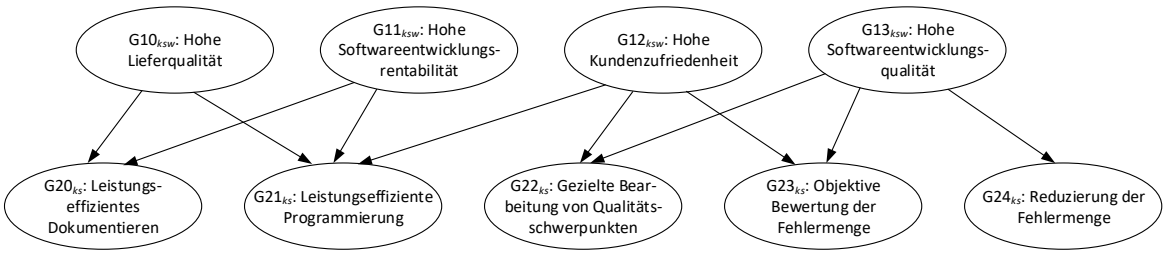


Abbildung 4.3: Relationen zwischen strategischen und operativen Zielen

4.2.2 Bestimmung der Softwarekennzahlen

Der Formulierung der operativen Ziele folgt die Bestimmung von dazugehörenden Softwarekennzahlen. Dafür wird die GQM-Methode verwendet. Die einschlägige Literatur misst der GQM-Methode eine hohe Bedeutung bei und beschreibt deren erfolgreiche Anwendung in der Praxis. Dies ist ein Grund dafür, dass beim Kooperationspartner die GQM-Methode für die Bestimmung von Softwarekennzahlen ausgewählt wurde. Der zweite Grund ist, dass die RGQM-Methode auf der GQM-Methode aufbaut und somit beim Kennzahlentransfer ähnlich wie bei der Bestimmung der Softwarekennzahlen vorgegangen werden kann.

Für jedes operative Ziel wurden mehrere Fragen formuliert und für jede Frage wurde mindestens eine Softwarekennzahl bestimmt. Dieser Prozess, der der GQM-Definitionsphase zuzuordnen ist (vgl. Abbildung 2.14), wurde vom Verfasser dieser Arbeit durchgeführt. Die Prozessergebnisse wurden von mehreren Stakeholdern aus der Softwareentwicklung geprüft.

In den folgenden Abschnitten werden die Ergebnisse des Prozesses und einige zum Verständnis notwendige Erläuterungen aufgeführt. Auf eine vertiefende Darstellung wird verzichtet, da lediglich eine anerkannte Methode angewendet wird und keine neuen wissenschaftliche Erkenntnisse gewonnen werden.

Für die Erläuterung der folgenden Softwarekennzahlen wie Churn, intern und extern entdeckte Fehler, Liefergeschwindigkeit etc. wird auf den Abschnitt 2.1.2 verwiesen.

4.2.2.1 Leistungseffiziente Programmierung

Um das abstrakt formulierte Ziel „leistungseffiziente Programmierung“ zu quantifizieren, sind folgende Fragen zu beantworten:

- $Q20.1_{ks}$: Wie viel Software wurde erstellt?
- $Q20.2_{ks}$: Wie viele Stunden wurden für die Erstellung der Software aufgewendet?
- $Q20.3_{ks}$: Wie lange hat das Softwareentwicklungsprojekt gedauert?

- $Q20.4_{ks}$: Wie hoch ist die Produktivität bezogen auf die erstellte Software?
- $Q20.5_{ks}$: Wie hoch ist die Liefergeschwindigkeit bezogen auf die erstellte Software?

Für jede dieser Fragen wurde je eine Softwarekennzahl identifiziert, mit der die jeweilige Frage beantwortet werden kann:

- $k20.1.1_s$: Churn
- $k20.2.1_s$: Aufwand für die Entwicklungsaktivitäten
- $k20.3.1_s$: Entwicklungsdauer
- $k20.4.1_s$: Churn-Produktivität
- $k20.5.1_s$: Churn-Liefergeschwindigkeit

Um die Softwarequantität zu bestimmen, wurde der Churn ausgewählt. Wie in Abschnitt 2.1.2.1.3 erläutert, ist der Churn zum einen ein aussagekräftiges Maß für die Menge der bearbeiteten Software (es werden die Änderungen des Quelltextes berücksichtigt), zum anderen kann der Churn automatisiert gemessen werden. LOC zeigen hingegen nicht die Menge der tatsächlichen Softwarebearbeitungen an. Function Points können im Umfeld des Kooperationspartners nicht automatisiert gemessen werden, da die dafür notwendigen Voraussetzungen nicht vorhanden sind: Es fehlen über den Produktlebenszyklus konsistente UML-Modelle aller Softwareprodukte.

Der Churn beschreibt die Menge aller Quelltextänderungen an einer Softwareversion bzw. an einem Softwareprodukt. Er ist entsprechend zuzuordnen. Der Softwareentwicklungsprozess muss folglich eine solche Zuordnung ermöglichen. Da bereits die ähnliche Anforderung A1 die Zuordnung von Quelltextänderungen umfasst, wird folgende ergänzende Anforderung an den Softwareentwicklungsprozess formuliert, die in dessen Gestaltung berücksichtigt werden muss:

Anforderung 4 (Erweiterte Zuordnung Quelltextänderungen) *Der Softwareentwicklungsprozess muss es ermöglichen, alle Quelltextänderungen einer Softwareversion zuzuordnen und diese Zuordnung zu erfassen. Durch eine derartige Zuordnung zu einer Softwareversion ist es auch möglich, die Quelltextänderungen einem Softwareprodukt zuzuordnen.*

Um die weiteren Softwarekennzahlen ermitteln zu können, muss der Softwareentwicklungsprozess ebenfalls die Anforderungen A5 und A6 erfüllen.

Anforderung 5 (Entwicklungsaufwand) *Der Softwareentwicklungsprozess muss es ermöglichen, den Ist-Aufwand der Entwicklungsaktivitäten zu erfassen.*

Anforderung 6 (Start und Ende) *Der Softwareentwicklungsprozess muss es ermöglichen, den Starttermin und den Endtermin einer Softwareversion zu erfassen.*

4.2.2.2 Leistungseffizientes Dokumentieren

Um das abstrakt formulierte Ziel „leistungseffizientes Dokumentieren“ zu quantifizieren, sind folgende Fragen zu beantworten:

- $Q21.1_{ks}$: Wie viel Softwaredokumentation wurde erstellt?
- $Q21.2_{ks}$: Wie viele Stunden wurden für die Erstellung der Softwaredokumentation aufgewendet?
- $Q21.3_{ks}$: Wie lange hat das Softwareentwicklungsprojekt gedauert?
- $Q21.4_{ks}$: Wie hoch ist die Produktivität bezogen auf die erstellte Softwaredokumentation?
- $Q21.5_{ks}$: Wie hoch ist die Liefergeschwindigkeit bezogen auf die erstellte Softwaredokumentation?

Für jede dieser Fragen wurde je eine Softwarekennzahl identifiziert, mit der die jeweilige Frage beantwortet werden kann:

- $k21.1.1_s$: Anzahl an Work Items (vgl. Abschnitt 2.1.2.1.4)
- $k21.2.1_s$: Aufwand für die Dokumentationsaktivitäten
- $k21.3.1_s$: Entwicklungsdauer (wird von $G20_{ks}$ übernommen)
- $k21.4.1_s$: Dokumentationsproduktivität
- $k21.5.1_s$: Dokumentationsliefergeschwindigkeit

Die Auswahl der Kennzahl *Anzahl an Work Items* erfolgte aufgrund der Art und Weise der Implementierung des V-Modells beim Kooperationspartner: Wie bereits in Abschnitt 2.5.2.2 erläutert, wird ein ALM-System verwendet, in dem einzelne Dokumentationsartefakte, zum Beispiel Anforderungen oder Testfälle, als Work Items verwaltet werden. Auf die Details der Verwaltung wird in Kapitel 5 eingegangen. Diese Art der Erfassung der Dokumentationsgröße zeigt nicht an, wie viel Text in einem Work Item formuliert wurde. Da Work Items kleine Informationseinheiten sind, umfasst der Text in der Regel wenige Zeilen. Es wäre zwar möglich, die Anzahl der Work Items mit einer durchschnittlichen Anzahl an Zeilen zu multiplizieren. Da bislang keine Erfahrungen existieren, wie viele

Zeilen durchschnittlich in Work Items eingetragen werden, wird darauf verzichtet und nur die Anzahl an Work Items als Softwarekennzahl bestimmt.

Um diese Softwarekennzahlen erfasst zu können, muss der Softwareentwicklungsprozess die Anforderungen A7 und A8 erfüllen.

Anforderung 7 (Softwaredokumentation) *Der Softwareentwicklungsprozess muss es ermöglichen, Work Items einer Softwareversion zuzuordnen und die jeweils erstellte Anzahl an Work Items zu erfassen. Durch eine derartige Zuordnung zu einer Softwareversion ist es auch möglich, die Work Items einem Softwareprodukt zuzuordnen.*

Anforderung 8 (Dokumentationsaufwand) *Der Softwareentwicklungsprozess muss es ermöglichen, den Ist-Aufwand der Dokumentationsaktivitäten zu erfassen.*

4.2.2.3 Gezielte Bearbeitung von Qualitätsschwerpunkten

Um Qualitätsschwerpunkte gezielt bearbeitet zu können, ist die Frage zu beantworten:

- $Q22.1_{ks}$: Wo liegen die Qualitätsschwerpunkte?

Für die Beantwortung der Frage $Q22.1_{ks}$ müssen alle entdeckten Fehler kategorisierbar sein, um eine prozentuale Verteilung der Qualitätsschwerpunkte feststellen zu können. Jeder Fehler muss mit Attributen beschrieben werden können, die die Softwarequalitätseigenschaften, die Fehlerursachen und den Schweregrad des Fehlers anzeigen. Die oben genannte Frage wird mit folgenden Softwarekennzahlen beantwortet:

- $k22.1.1_s$: Prozentuale Verteilung von Softwarequalitätseigenschaften
- $k22.1.2_s$: Prozentuale Verteilung von Fehlerursachen in der Programmierung
- $k22.1.3_s$: Prozentuale Verteilung von Fehlerschweregraden

Wie in Abschnitt 2.1.2.4.1 erläutert, ist die prozentuale Verteilung keine einzelne Softwarekennzahl, sondern zeigt die Teilwerte der einzelnen Fehlereinordnungen an.

Um die prozentualen Verteilungen erfassen zu können, muss der Softwareentwicklungsprozess die Anforderung A9 erfüllen.

Anforderung 9 (Fehlerattribute) *Der Softwareentwicklungsprozess muss es ermöglichen, Fehlerattribute zu erfassen.*

4.2.2.4 Objektive Bewertung der Fehlermenge

Um objektiv bewerten zu können, wie viele Fehler durch interne qualitätssichernde Maßnahmen und wie viele erst durch die Kunden entdeckt wurden, ist die Frage zu beantworten:

- $Q23.1_{ks}$: Wie ist das Verhältnis von intern zu extern entdeckten Fehlern?

Um diese Frage beantworten zu können, wurden folgende Softwarekennzahlen identifiziert:

- $k23.1.1_s$: Anzahl intern entdeckter Fehler
- $k23.1.2_s$: Anzahl extern entdeckter Fehler
- $k23.1.3_s$: Fehlerbehebungsrate

Damit diese Softwarekennzahlen erfasst werden können, muss der Softwareentwicklungsprozess die Anforderung A10 erfüllen.

Anforderung 10 (Fehlerzuordnung) *Der Softwareentwicklungsprozess muss es ermöglichen, Fehler einer Softwareversion zuzuordnen und diese in intern und extern entdeckte Fehler zu unterscheiden. Durch eine derartige Zuordnung zu einer Softwareversion ist es auch möglich, die Fehler einem Softwareprodukt zuzuordnen.*

4.2.2.5 Reduzierung der Fehlermenge

Um Qualitätsverbesserungen objektiv nachweisen zu können, ist die Frage zu beantworten:

- $Q24.1_{ks}$: Wie viele Fehler pro Softwaremengeneinheit wurden entdeckt? Da der Churn für die Bestimmung der Softwaremengen ermittelt wurde, wird die Frage präzisiert: Wie viele Fehler pro KB Churn wurden entdeckt?

Für die Beantwortung dieser Frage wurden die folgende Softwarekennzahl identifiziert:

- $k24.1.1_s$: Churn-Fehlerdichte

Zur Erfassung der Churn-Fehlerdichte ist keine weitere Anforderung durch den Softwareentwicklungsprozess zu erfüllen, da die Erfassung durch die Erfüllung der vorherigen Anforderungen möglich ist.

Durch die Anwendung der in Kapitel 3 beschriebenen RQGM-Methode und der in diesem Kapitel erläuterten Bestimmung der Softwarekennzahlen wurden die Anforderungen A1 bis A10 erfasst. Nur wenn der Softwareentwicklungsprozess beim Kooperationspartner diese Anforderungen erfüllt, ist die Erfassung aller bestimmten SW-Produktionskennzahlen und Softwarekennzahlen möglich. Der Gestaltung des Softwareentwicklungsprozesses unter Berücksichtigung dieser Anforderungen widmet sich das nächste Kapitel.

Kapitel 5

Gestaltung des Softwareentwicklungsprozesses

Dieses Kapitel widmet sich der zweiten Detailfrage dieser Arbeit: *Wie sollte der Softwareentwicklungsprozess aufgebaut sein, damit die definierten SW-Produktionskennzahlen und Softwarekennzahlen erfasst werden können?* Es wird das prinzipielle Vorgehen der Gestaltung des Softwareentwicklungsprozesses und ein Anwendungsbeispiel beim Kooperationspartner erläutert. Darin wird das sogenannte Sliced V-Modell entworfen. Die Inhalte dieses Kapitels entstanden iterativ in den Design Research-Entwurfsphasen *Erstellen* und *Evaluierung* (vgl. Abschnitt 1.5). Erste Konzepte des Sliced V-Modells wurden in [Deu12, Deu13] veröffentlicht.

Abbildung 5.1 zeigt, welche Inhalte in diesem Kapitel behandelt werden und an welcher Stelle diese Inhalte zur Erreichung der Zielsituation dieser Arbeit beitragen. Die graue Fläche markiert den Inhalt: die Gestaltung des Softwareentwicklungsprozesses, um darin die Kennzahlen für K_{SW} und K_S erfassen zu können.

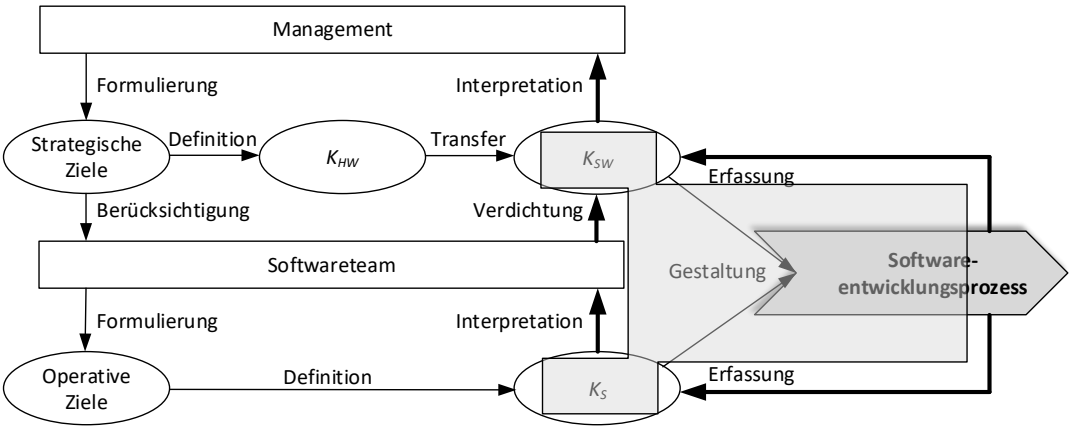


Abbildung 5.1: Inhalt des Kapitels 5 in Bezug auf die Zielsituation dieser Arbeit

5.1 Vorgehen

Wie in Abschnitt 2.5.1 dargelegt, beschreibt ein Vorgehensmodell den organisatorischen Rahmen des Softwareentwicklungsprozesses. Mit der Aktivität *Gestaltung des Softwareentwicklungsprozesses* ist folglich entweder die Auswahl eines Vorgehensmodells oder die Änderung eines bereits vorhandenen Vorgehensmodells gemeint.

In dieser Arbeit kann keine Empfehlung für ein bestimmtes Vorgehensmodell gegeben werden. Die Auswahl ist vom jeweiligen produzierenden Betrieb individuell zu tätigen. Um allerdings Kennzahlen aus dem Softwareentwicklungsprozess IT-basiert erfassen zu können, sollten für die Implementierung des gewählten Vorgehensmodells ein bzw. mehrere Collaboration Tools genutzt werden (vgl. Abschnitt 2.5.1).

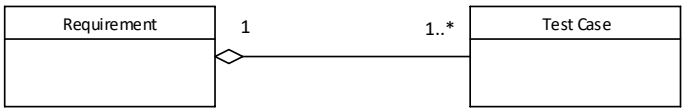
Die Gestaltung des Softwareentwicklungsprozesses berücksichtigt zum einen die spezifischen Bedürfnisse des produzierenden Betriebes und zum anderen die Anforderungen an den Softwareentwicklungsprozess, die bei der Bestimmung der SW-Produktionskennzahlen und der Softwarekennzahlen formuliert wurden. Ein Beispiel für die Berücksichtigung betriebsspezifischer Bedürfnisse ist die Definition von betriebsspezifischen Rollen und ein dafür angepasstes Rechtemanagement in einem Collaboration Tool.

Um die Anforderungen an den Softwareentwicklungsprozess erfüllen zu können, die bei der Bestimmung der SW-Produktionskennzahlen und der Softwarekennzahlen formuliert wurden, muss ein Datenmodell erstellt werden. Es beschreibt die im Softwareentwicklungsprozess entstehenden Softwareartefakte, wie zum Beispiel Produktanforderungen oder Testfälle. Diese durch das Datenmodell beschriebenen Softwareartefakte sind der *WorkProductDefinition*-Klasse des SPEM-Metamodells (vgl. Abschnitt 2.5.1) zuzuordnen.

Um das Datenmodell zu beschreiben, wird UML als eine etablierte Modellierungssprache verwendet. Konkret wird das UML-Klassendiagramm genutzt: Die Softwareartefakte und deren Beziehungen werden darin anschaulich dargestellt. Auf diese Art und Weise entsteht ein Datenmodell, das die Anforderungen erfüllt, die während der Bestimmung der SW-Produktionskennzahlen und der Softwarekennzahlen formuliert wurden. Das bedeutet auch, dass in dem Datenmodell alle benötigten UML-Relationen enthalten sind.

Erläuterung 5.1 zeigt ein Beispiel, wie eine Anforderung an den Softwareentwicklungsprozess in einem Datenmodell berücksichtigt wird.

Bei der Bestimmung von Softwarekennzahlen wurde folgende Kennzahl bestimmt: „Durchschnittliche Menge von Testfällen, die einer Produktanforderung zugeordnet sind“. Um diese Kennzahl zu erfassen, muss in der Gestaltung des Softwareentwicklungsprozesses Folgendes ermöglicht werden: *Im Softwareentwicklungsprozess muss jeder Testfall einer Produktanforderung zugeordnet werden können.*
Diese Anforderung an den Softwareentwicklungsprozess muss das Datenmodell berücksichtigen. Die folgende Abbildung zeigt den Auszug eines UML-Klassendiagramms, in dem dies realisiert ist. Wie zu erkennen ist, gibt es eine UML-Aggregationsbeziehung zwischen einer Produktanforderung (Requirement) und einem Testfall (Test Case).



Die weiteren gezeigten Eigenschaften des Softwareentwicklungsprozesses, wie zum Beispiel, dass zu jeder Produktanforderung mindestens ein Testfall existieren muss, können sich aus betriebsspezifischen Bedürfnissen an den Softwareentwicklungsprozess ergeben, die dieses vorschreiben.

Erläuterung 5.1: Anforderungen an den Softwareentwicklungsprozess und Datenmodell

Die Bewertung, ob das Datenmodell alle formulierten Anforderungen erfüllt, erfolgt mehrstufig durch:

1. eine manuelle Prüfung des Datenmodells,
2. die Erstellung der Berechnungsgrundlagen aller Kennzahlen,
3. und die Implementierung eines Informationsverarbeitungssystems.

Um ein Informationsverarbeitungssystem implementieren zu können, ist eine vollständige Modellierung des Softwareentwicklungsprozesses unter Anwendung weiterer Klassen aus dem SPEM-Metamodell, wie zum Beispiel der *RoleDefinition*-Klasse oder der *TaskDefinition*-Klasse, nicht nötig. Eine vollständige Modellierung wird zwar jedem produzierenden Betrieb empfohlen. Jedoch ist es für die Kennzahlenerfassung unerheblich, wer die Arbeitsergebnisse erzeugt hat und wie derjenige dabei vorgegangen ist, denn es werden nur die Arbeitsergebnisse benötigt, die dem Datenmodell entnommen werden können.

Das in diesem Abschnitt erläuterte Vorgehen kam beim Kooperationspartner zum Einsatz. Im folgenden Abschnitt wird das entwickelte Datenmodell vorgestellt.

5.2 Sliced V-Modell

5.2.1 Begriff und Anforderungen

Wie in Abschnitt 2.5.2.2 erläutert, setzt der Kooperationspartner ein auf dem V-Modell der DIN EN 61508-3 beruhendes Vorgehensmodell ein. Im Zuge einer Prozessverbesserungsmaßnahme wurde das Verfahren, das die Nutzung von MS Word-Dokumenten und deren Ablage auf Netzwerklaufwerken vorsah, schrittweise durch die Nutzung eines ALM-Systems ersetzt. In diesem ALM-System werden alle Dokumente und Entwicklungsinformationen eingetragen. Des Weiteren wurde im Rahmen der Einführung des ALM-Systems ein vorhandenes Versionsmanagementsystem gegen ein neues ausgetauscht. In internen Prozessbeschreibungen ist die Realisierung des V-Modells der DIN EN 61508-3 in dem ALM-System und in dem Versionsmanagementsystem erläutert. Diese internen Prozessbeschreibungen erklären zwar die Aktivitäten im Umgang mit den beiden Systemen und benennen die dafür verantwortlichen Personen, es fehlt jedoch ein Datenmodell, das die im Softwareentwicklungsprozess entstehenden Softwareartefakte beschreibt. Um ein Informationsverarbeitungssystem aufzubauen, wird das Datenmodell benötigt, damit die Berechnungsgrundlagen der SW-Produktionskennzahlen und Softwarekennzahlen definiert und durch ein Informationsverarbeitungssystem implementiert werden können.

Wie im vorherigen Abschnitt dargelegt, erfolgt die Beschreibung des Datenmodells in Form von UML-Klassendiagrammen. In dem Datenmodell muss das V-Modell der DIN EN 61508-3 berücksichtigt werden, da es das gewählte Vorgehensmodell des Kooperationspartners ist. In dem Datenmodell muss folglich zum einen eindeutig erkennbar sein, dass die Ergebnisse der einzelnen Phasen des V-Modells, wie beispielsweise eine Anforderungsspezifikation, berücksichtigt werden. Zum anderen müssen die Validations- und Verifikationsbeziehungen der DIN EN 61508-3 erhalten bleiben (vgl. Abbildung 2.21). Aufgrund dieser Randbedingung wurde in dieser Arbeit keine weitere Option des inhaltlichen Aufbaus des Datenmodells geprüft.

Da das Datenmodell auf dem V-Modell der DIN EN 61508-3 aufbaut, wird es als dessen Verfeinerung betrachtet. Diese Verfeinerung wird *Sliced V-Modell* genannt. Das Sliced V-Modell beschreibt die Dokumente der DIN EN 61508-3, die allerdings keine traditionellen Dokumente darstellen. Vielmehr sind es Container für Work Items, in denen die eigentlichen Dokumenteninhalte eingetragen werden. Die Work Items sind untereinander verbunden und bilden dokumentenübergreifend „Scheibchen“ von Work Items. Diese „Scheibchen“ begründen den Namen *Sliced V-Modell*.

Um alle bestimmten SW-Produktionskennzahlen und Softwarekennzahlen erfassen zu können, werden bei der Erstellung des Sliced V-Modells die in den Abschnitten 3.2.2.4 und

4.2.2 ermittelten Anforderungen an den Softwareentwicklungsprozess berücksichtigt. Im Folgenden werden alle Anforderungen zusammenfassend aufgeführt:

A1 Zuordnung Quelltextänderungen:
Der Softwareentwicklungsprozess muss es ermöglichen, Quelltextänderungen sowohl der Implementierung neuer Softwarefunktionen als auch der Behebung intern und extern entdeckter Fehler zuzuordnen und diese Zuordnungen zu erfassen.

A2 Termine Softwarefunktionen:
Der Softwareentwicklungsprozess muss es ermöglichen, neue Softwarefunktionen individuell mit einem zugesagten Termin zu markieren und diesen zugesagten Termin sowie den Freigabetermin der Softwarefunktion zu erfassen.

A3 Soll-/Ist-Stunden:
Der Softwareentwicklungsprozess muss es ermöglichen, die Soll-Aufwände für die Entwicklung von Softwarefunktionen und die Ist-Aufwände, die für die Entwicklung einer Softwareversion geleistet wurden, zu erfassen.

A4 Erweiterte Zuordnung Quelltextänderungen:
Der Softwareentwicklungsprozess muss es ermöglichen, alle Quelltextänderungen einer Softwareversion zuzuordnen und diese Zuordnung zu erfassen.

A5 Entwicklungsaufwand:
Der Softwareentwicklungsprozess muss es ermöglichen, den Ist-Aufwand der Entwicklungsaktivitäten zu erfassen.

A6 Start und Ende:
Der Softwareentwicklungsprozess muss es ermöglichen, den Starttermin und den Endtermin einer Softwareversion zu erfassen.

A7 Softwaredokumentation:
Der Softwareentwicklungsprozess muss es ermöglichen, Work Items einer Softwareversion zuzuordnen und die jeweils erstellte Anzahl an Work Items zu erfassen.

A8 Dokumentationsaufwand:
Der Softwareentwicklungsprozess muss es ermöglichen, den Ist-Aufwand der Dokumentationsaktivitäten zu erfassen.

A9 Fehlerattribute:
Der Softwareentwicklungsprozess muss es ermöglichen, Fehlerattribute zu erfassen.

A10 Fehlerzuordnung:
Der Softwareentwicklungsprozess muss es ermöglichen, Fehler einer Softwareversion zuzuordnen und diese in intern und extern entdeckte Fehler zu unterscheiden.

5.2.2 Eigenschaften

Das Sliced V-Modell enthält die nachfolgend aufgeführten Artefakte. Die Auflistung zeigt zunächst den Namen der Artefakte und in welchem IT-System (ALM-System oder Versionsmanagementsystem) die Artefakte verwaltet werden. In weiteren Verlauf dieses Abschnitts werden die Artefakte detailliert erläutert.

- Storage (ALM-System)
- Document (ALM-System)
- Work Item (ALM-System)
- Link (ALM-System)
- Repository (Versionsmanagementsystem)
- Revision (Versionsmanagementsystem)
- Baseline (ALM-System)

Abbildungen 5.2 und 5.3 zeigen das Datenmodell, anhand dessen die aufgeführten Artefakte erläutert werden. Abbildung 5.2 stellt die Vererbungszusammenhänge der Artefakte dar, Abbildung 5.3 gibt die Beziehungen zwischen den Artefakten wieder.

Das *Storage* ist das zentrale Artefakt des Datenmodells, mit dem alle weiteren Artefakte verbunden sind. Das Storage existiert über den gesamten Lebenszyklus des Softwareproduktes. Beim Anlegen des Sliced V-Modell Storage enthält es noch keine der anderen Artefakte, diese werden nach und nach hinzugefügt.

Ein *Document* enthält die Dokumentation, die in einer bestimmten Phase des V-Modells der DIN EN 61508-3 erstellt wird. In einem Sliced V-Modell gibt es verschiedene *Document*-Typen. Um eine neue Softwareversion zu entwickeln, wird zunächst ein *Feature Set Document* erstellt. Es enthält alle zu entwickelnden neuen Softwarefunktionen (die *Features*). Die sich aus den neuen Softwarefunktionen ergebenen konkreten Produktanforderungen an die Softwareversion werden in einem oder mehreren *Requirements Specification Documents* eingetragen. Es folgen die weiteren Dokumente, die die DIN EN 61508-3 vorschreibt: *System Design Document*, *Module Design Document* und mehrere *Test Documents*. Um Aufgaben zu verwalten, die während der Entwicklung der neuen Softwareversion zu erledigen sind, wird jeweils mindestens ein *Task Document* angelegt. Die Fehler, die während der internen qualitätssichernden Maßnahmen entdeckt werden (vor dem Endtermin einer Softwareversion), und die Fehler, die extern von Kunden (nach dem Endtermin einer Softwareversion) gefunden werden, werden jeweils in getrennte *Defect*

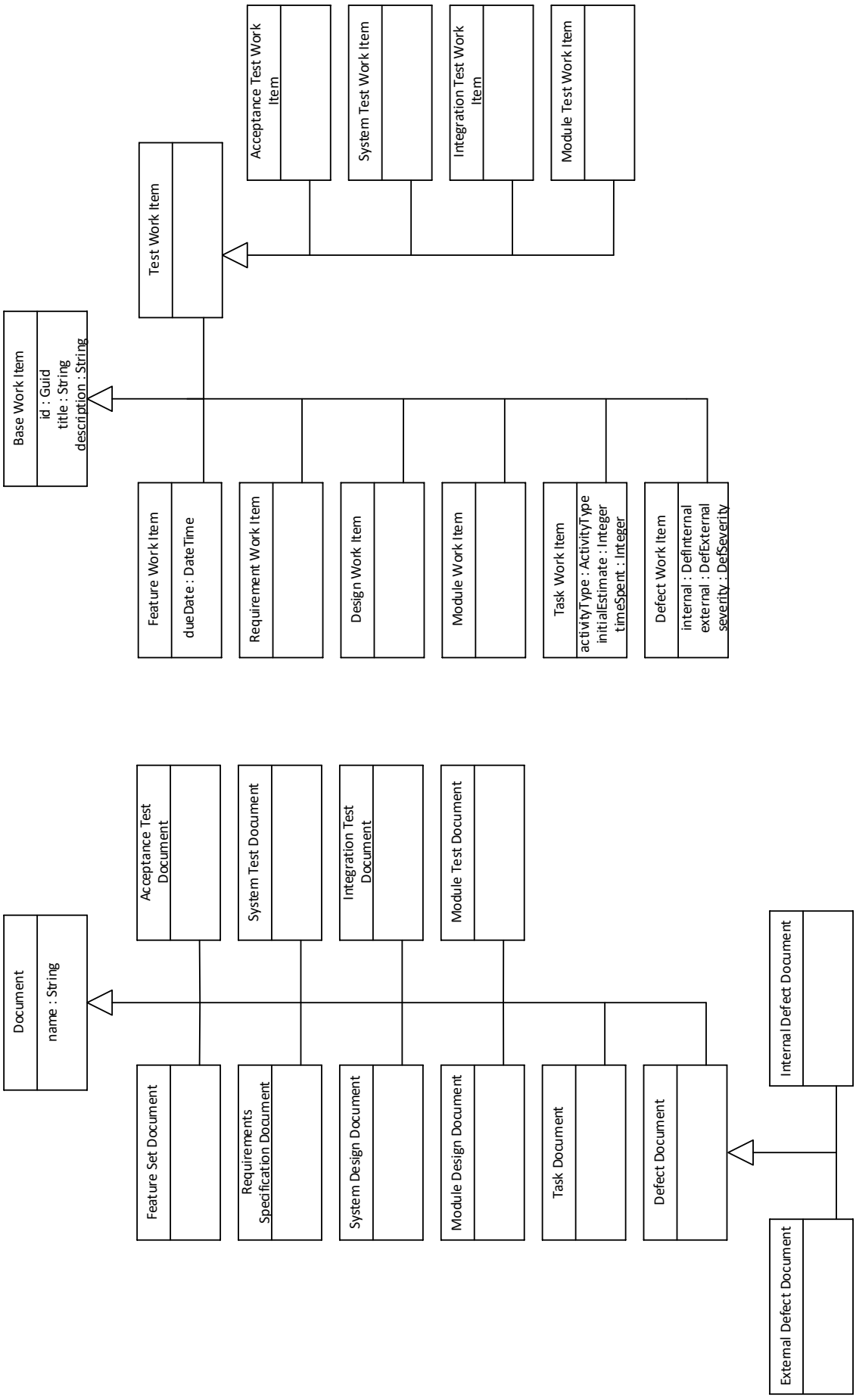


Abbildung 5.2: Vererbungszusammenhang der Sliced V-Modell-Artefakte

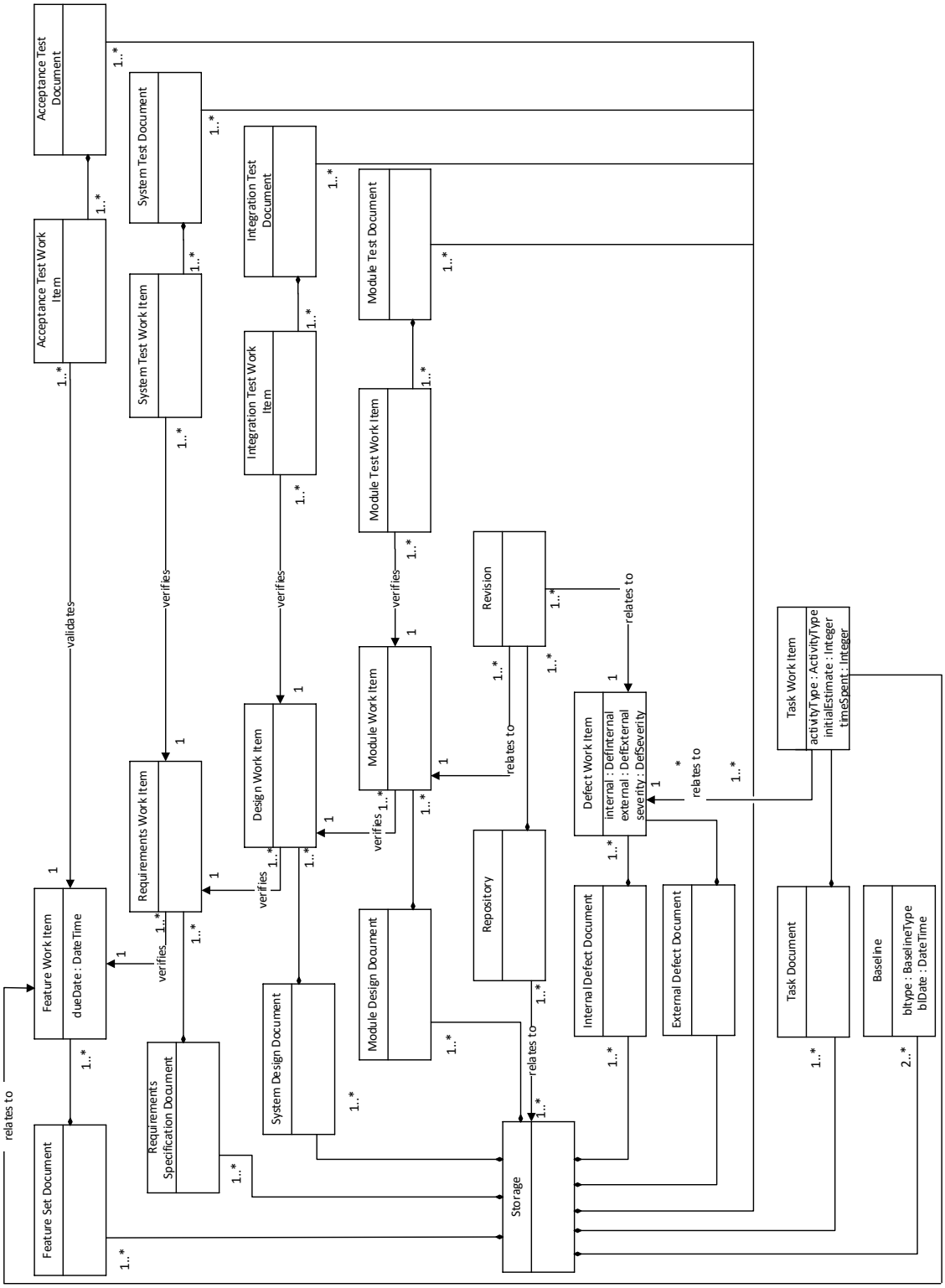


Abbildung 5.3: Beziehungen zwischen den Sliced V-Modell-Artefakten

Documents eingetragen: Es gibt für jede Softwareversion ein *Defect Document* für die intern entdeckten Fehler und für jedes Softwareprodukt, also in jedem *Storage*, ein oder mehrere Dokumente für die extern entdeckten Fehler.

Die Namen der *Feature Set Documents*, der *Defect Documents* und der *Task Documents* werden gemäß festgelegten Regeln gebildet. Dadurch können sie durch ein Informationsverarbeitungssystem identifiziert werden. Folgende Dokumentennamen zeigen Beispiele für die Regeln:

- Features_V1_20: Enthält alle Features, die zur Softwareversion V1.20 umzusetzen sind. Ein *Feature Set Document* wird für jede Softwareversion angelegt.
- Tasks_V1_20: Enthält alle projektspezifischen Aktivitäten, die für die Softwareversion V1.20 zu erledigen sind. Ein *Task Document* wird für jede Softwareversion angelegt.
- Defects_V1_20: Enthält alle Fehler, die während der internen qualitätssichernden Maßnahmen für die Softwareversion V1.20 entdeckt wurden. Dieses Dokument wird *Internal Defect Document* genannt und wird für jede Softwareversion angelegt.
- ExtDefects_All: Enthält alle extern entdeckten Fehler. Dieses Dokument wird *External Defect Document* genannt. Es gibt mindestens ein Dokument für ein Softwareprodukt.

Die *Documents* enthalten Work Items und sind folglich Container von Work Items. Work Items wurden in Abschnitt 2.1.2.1.4 eingeführt: Sie beschreiben sowohl zu erledigende Aktivitäten im Entwicklungsprozess als auch Dokumentationsartefakte, wie zum Beispiel Produktanforderungen, Testfälle oder Fehlerbeschreibungen. In einem Sliced V-Modell werden verschiedene Typen von Work Items verwendet, die in Tabelle 5.1 aufgeführt sind.

In einem *Document* dürfen nur Work Items eines Typs vorkommen. Ein Beispiel: Ein *Requirements Specification Document* (eine Anforderungsspezifikation) enthält nur *Requirements Work Items*. Allerdings kann es mehrere *Requirements Specification Documents* geben.

Jeder Work Item-Typ besitzt mindestens die in Tabelle 5.2 aufgeführten Attribute, die vom *Base Work Item* vererbt werden. Sie werden *Standardattribute* genannt. Darüber hinaus verfügen einige Work Item-Typen über weitere Attribute, die in den nächsten Absätzen erläutert werden.

Für das Verständnis der Work Item-Typen ist in Tabelle 5.3 ein Beispiel aufgeführt. Es bezieht sich auf eine neue Softwarefunktion einer SPS-Software, die sogenannte optische

Work Item-Typ	Beschreibung
Feature	Beschreibung einer Softwarefunktion
Requirement	Detaillierte Produktanforderung zu einer Softwarefunktion
Design	High Level-Softwaredesign, zum Beispiel Softwarearchitektur
Module	Low Level-Softwaredesign in Ergänzung zum Quelltext und zu Programmkommentaren
Test	Basistyp für die Test Work Item-Typen
Acceptance Test	Testbeschreibung eines Abnahmetests
System Test	Testbeschreibung eines Systemtests
Integration Test	Testbeschreibung eines Integrationstests
Module Test	Testbeschreibung eines Modultests
Defect	Fehlerbeschreibung
Task	Zu erledigende Dokumentations- bzw. Entwicklungsaktivitäten

Tabelle 5.1: Work Item-Typen im Sliced V-Modell

Profinet-Diagnose. Profinet ist ein industrielles Ethernet-Protokoll, über das eine SPS mit sogenannten Profinet-Geräten Daten austauschen kann. Diese neue Softwarefunktion wird in einem *Feature Work Item* vermerkt. Um diese Softwarefunktion zu realisieren, müssen mehrere Anforderungen, mehrere Designerweiterungen, mehrere Testfälle und mehrere Aufgaben formuliert und umgesetzt werden. In Tabelle 5.3 ist jeweils nur ein Beispiel für einen Work Item-Typ bzw. nur für einen vom *Test Work Item* abgeleiteten Typen gezeigt.

Ein *Feature Work Item*, das in ein *Feature Set Document* eingetragen wird, enthält zusätzlich zu den Standardattributen das Attribut *dueDate*. In diesem Attribut wird notiert, zu welchem Termin die Fertigstellung einer Softwarefunktion zugesagt ist.

Vom *Test Work Item* sind weitere Work Item-Typen für jede sogenannte Teststufe abgeleitet. Eine Teststufe ist „eine Gruppe von Testaktivitäten, die gemeinsam ausgeführt und verwaltet werden“ [SL03]. Beispiele einer Teststufe im V-Modell der DIN EN 61508-3 sind der Modultest oder der Integrationstest. Die abgeleiteten Work Item-Typen sind: *Acceptance Test Work Item*, *System Test Work Item*, *Integration Test Work Item* und *Module Test Work Item*.

Attributname	Typ	Beschreibung
id	Guid	Identifikator, um ein Work Item eindeutig im ALM-System identifizieren zu können. Wird automatisch generiert
title	String	Titel des Work Items
description	String	Inhalt des Work Items (Text, Bilder)

Tabelle 5.2: Attribute des *Base Work Items*

Work Item-Typ	**Attribut *title***	**Attribut *description***
Feature	Optische Profinet-Diagnose	Optische Profinet-Diagnose gemäß Profinet-Spezifikation V2.3
Requirement	Anzeige Profinet-Gerät im SPS-Display	Wenn ein Profinet-Gerät einen Alarm der optischen Diagnose meldet, muss die Adresse des Profinet-Geräts im SPS-Display angezeigt werden
Design	Systemarchitektur Optische Diagnose	Erweiterung UML-Klassendiagramm um Klasse *OptDiagData* (siehe UML-Werkzeug)
Module	Auswertung Profinet-Alarm	Implementierung der Funktion *receiveOptAlarm*
System Test	Anzeige Profinet-Gerät	Schritt 1: Starten der SPS Schritt 2: Reduzierung der optischen Leistung zwischen Gerät 1 und 2 Erwartetes Ergebnis: Die Adresse des Profinet-Geräts 2 wird angezeigt
Defect	Falsche Adresse des Profinet-Geräts	Die Adresse des Profinet-Geräts 1 wird anstatt der Adresse des Profinet-Geräts 2 angezeigt
Task	Erweiterung der Systemarchitektur	Check-Out Klassendiagramm, Bearbeiten, Check-In

Tabelle 5.3: Beispiele für Work Item-Typen

Das *Defect Work Item* besitzt zusätzlich zu den Standardattributen folgende weitere Attribute, mit denen verschiedene Stakeholder eines Softwareproduktes ihre Sicht auf einen Fehler ausdrücken können:

- Ein Softwareentwickler trägt in dem Attribut *internal* die von ihm erkannten Ursachen des entdeckten Fehlers ein. Dieses Attribut hat den Datentyp *DefInternal*. Dies ist ein Aufzählungstyp, dessen Wertemenge [SSR+08] entnommen ist.
- Die Bewertung eines Anwenders des Softwareproduktes, hinsichtlich der Frage, welche Qualitätseigenschaft des Softwareproduktes durch den entdeckten Fehler beeinträchtigt ist, wird in das Attribut *external* eingetragen. Dieses Attribut hat den Datentyp *DefExternal*. Dies ist ein Aufzählungstyp, dessen Wertemenge an die Qualitätscharakteristiken der ISO/IEC 25010 angelehnt ist.
- Ein Produktverantwortlicher nutzt das Attribut *severity* und bewertet damit den Schweregrad des Fehlers. Dieses Attribut hat den Datentyp *DefSeverity*. Dies ist ein Aufzählungstyp, dessen Wertemenge [PP11] entstammt.

Tabelle 5.4 zeigt die genannten Aufzählungstypen und deren Wertemengen. Die Bedeutung der Werte in den Wertemengen wird im Anhang A erläutert. Um die extern entdeckten Fehler einer Softwareversion zuzuordnen, enthält das *Defect Work Item* ein weiteres Attribut, *swVersion* vom Typ *String*. Darin wird die Softwareversion eingetragen, in der der Fehler entdeckt wurde.

Das *Task Work Item* besitzt zusätzlich zu den Standardattributen die in Tabelle 5.5 gezeigten Attribute. Die Zeitbasis für diese Attribute sind Stunden. Das Attribut *initialEstimate* dient der Planung einer Aufgabe. Das Attribut *timeSpent* gibt den für diese Aktivität erbrachten Aufwand wieder.

Aufzählungstyp	**Beschreibung**	**Wertemenge**
DefInternal	Bewertung eines Fehlers aus Sicht eines Entwicklers	Algorithm, method Assignment, initialization Checking Data External interface Internal interface Logic Non-functional Timing, optimization Other
DefExternal	Bewertung eines Fehlers aus Sicht eines Anwenders	Documentation Functionality Handling Optic Performance Stability
DefSeverity	Bewertung eines Fehlers aus Sicht eines Produktverantwortlichen	Critical Major Neutral Minor Trivial

Tabelle 5.4: Aufzählungstypen der Attribute des *Defect Work Items*

Um Dokumentationsaktivitäten und Entwicklungsaktivitäten zu unterscheiden, enthält das *Task Work Item* das Attribut *activityType* vom Typ *ActityType*. Dies ist ein Aufzählungstyp mit der Wertemenge *{documentation, development}*. Dokumentationsaktivitäten umfassen alle Aufgaben für die Erstellung und die Pflege der Entwicklungsdokumentation, zum Beispiel *Erstellen einer Testspezifikation*. Bei Entwicklungsaktivitäten

Attributname	Typ	Beschreibung
initialEstimate	Integer	Initial abgeschätzte Zeit in Stunden, die notwendig ist, um die Aktivität zu erledigen
timeSpent	Integer	Tatsächliche Zeit in Stunden, die für die Erledigung der Aktivität aufgebracht wurde

Tabelle 5.5: Zeitattribute des *Task Work Items*

handelt es sich um alle Aufgaben für die Realisierung des Softwareproduktes, zum Beispiel *Programmieren einer Klassenmethode*.

Work Items können untereinander in Beziehung gesetzt werden. Eine derartige Beziehung ist ein *Link*. Ein Link zeigt einen Informationszusammenhang der beiden verbundenen Work Items an und hat einen Typ und eine Richtung. Die DIN EN 61508-3 definiert die Traceability-Regeln *Validierung*, *Verifikation* und *Ausgabe*. Um diese Regeln im Sliced V-Modell umzusetzen, werden die Linktypen *validates* und *verifies* definiert, mit denen Work Items verlinkt werden können. Da die Traceability-Regel der DIN EN 61508-3 *Ausgabe* dieselben Artefakte verbindet wie die Traceability-Regel *Verifikation* (vgl. Abbildung 2.21), nur in entgegengesetzter Richtung, wird diese Traceability-Regel im Sliced V-Modell nicht genutzt. Für das Traceability Management bietet sie keinen inhaltlichen Mehrwert: Da die Work Items in einem Sliced V-Modell über mindestens einen Link in Beziehung gesetzt werden, ist die Traceability entlang des V-Modells gewährleistet. Das Verwalten von Traceability-Matrizen entfällt (vgl. Tabelle 2.5), da ALM-Systeme in der Lage sind, Traceability-Berichte automatisiert zu erstellen. Der Aufwand des Traceability-Managements reduziert sich somit auf das Setzen der Links und auf die Auswertung der Traceability-Berichte.

Da die Dokumente in Work Items aufgeteilt werden und die Work Items dokumentenübergreifend verlinkt sind, bilden sich dokumentenübergreifende „Scheibchen“ (Abbildung 5.4). Wie schon erwähnt, begründet dies den Namen *Sliced V-Modell*. Das ausgehend von einem *Feature Work Item* gebildete „V“ wird *V-Slice* genannt.

Das Sliced V-Modell stellt einer Verfeinerung des V-Modells dar. Der in Abschnitt 2.5.1 erläuterte Nachteil des V-Modells, demnach eine nächste Entwicklungsphase erst begonnen werden kann, wenn ein vollständiges Dokument erstellt ist, entfällt im Sliced V-Modell. Die Steuerung der Entwicklungsphasen erfolgt im Sliced V-Modell nicht dokumentenorientiert, sondern Work Item-orientiert. Damit ist gemeint, dass zwar einzelne Work Items innerhalb einer V-Slice in einer bestimmten Reihenfolge zu erstellen sind, allerdings erfolgt die Bearbeitung einzelner V-Slices unabhängig voneinander.

Grundsätzlich kann im Sliced V-Modell jede Softwarefunktion einzeln und unabhängig von anderen Softwarefunktionen spezifiziert, implementiert und getestet werden.

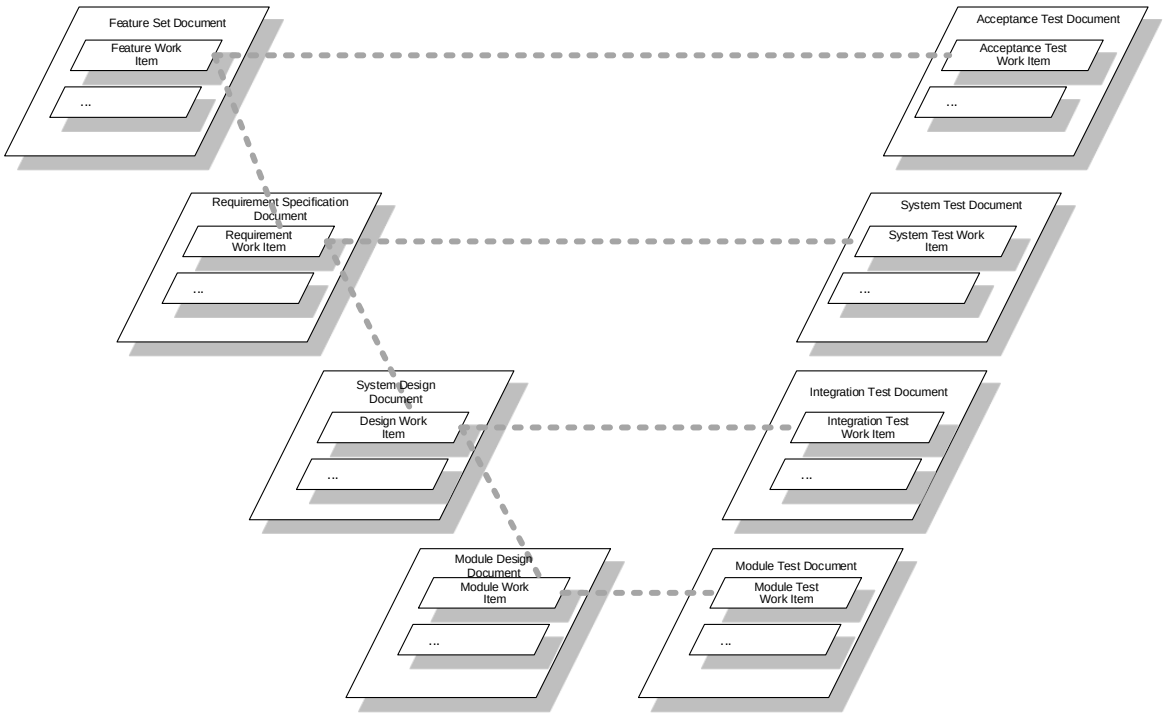


Abbildung 5.4: Schematische Darstellung von V-Slices

Abbildung 5.5 zeigt exemplarisch drei Softwarefunktionen, die jeweils in einer V-Slice bearbeitet werden. Für jede Softwarefunktion wird ein Feature Work Item erzeugt, das den Startpunkt einer einzelnen V-Slice darstellt. Wie in der Abbildung gezeigt, können die V-Slices zeitlich versetzt bearbeitet werden.

Die Steuerung der Bearbeitung der V-Slices bzw. der einzelnen Work Items innerhalb einer V-Slice erfolgt durch die Task Work Items. Sie können, ähnlich wie die Backlog-Items in Scrum, erfasst und innerhalb eines Sprint-Zyklus bearbeitet werden. Task Work Items können vor jedem Sprint-Zyklus neu definiert, verworfen oder geändert werden. Da jede V-Slice ein „Mini-V“ darstellt, kann das Sliced V-Modell als ein agiles Vorgehensmodell angesehen werden, in dem die Qualitätseigenschaften des V-Modells integriert sind.

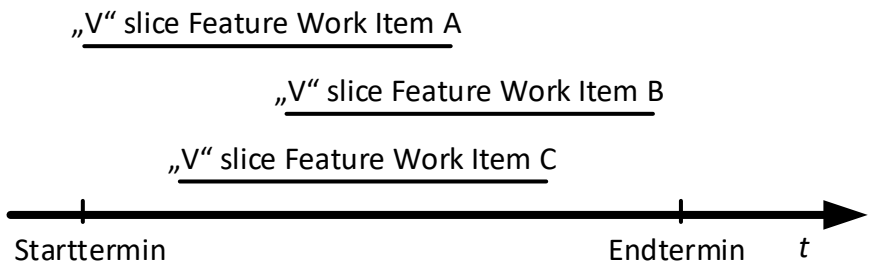


Abbildung 5.5: Bearbeitungsabfolge einzelner V-Slices

Ein *Repository* ist die Datenablage eines Versionsmanagementsystems [Pop13]. Eine *Revision* ist eine im Repository gespeicherte Änderung, wie zum Beispiel die Änderung an einer Quelltext-Datei. Die im Repository gespeicherten Revisionen sind entweder mit einem *Module Work Item* oder mit einem *Defect Work Item* verlinkt.

Das Sliced V-Modell Storage und damit alle darin enthaltenen Work Items und Dokumente wird zu definierten Zeitpunkten mit Hilfe einer *Baseline* „eingefroren“. Eine Baseline ist „a reference configuration from which to identify and to control change“ [DCKV08]. Sie enthält die vollständige Dokumentation des Sliced V-Modell Storage zu dem Zeitpunkt, an dem die Baseline erstellt wurde. Die Inhalte der Dokumente und der Work Items sind folglich abhängig von der Baseline. Abbildung 5.6 zeigt dies exemplarisch anhand einiger Dokumenttypen im Sliced V-Modell.

Eine Baseline wird jeweils zum Starttermin und zum Endtermin einer Softwareversion erstellt. Der Tag der Erstellung wird im Attribut *blDate* des Typs *Baseline* gespeichert. Die genaue Definition von „Starttermin“ und „Endtermin“ ist jeweils durch die Stakeholder festzulegen (vgl. Abschnitt 2.1.2.3). Um diese Unterscheidung an dem im Klassendiagramm gezeigten Typ *Baseline* vorzunehmen, besitzt dieser Typ das Attribut *blType* vom Typ *BaselineType*. Dieses ist ein Aufzählungstyp mit der Wertemenge *{start, end}*. Im Folgenden wird eine *Baseline*, die in diesem Attribut den Wert {start} gesetzt hat, Start-Baseline genannt. Eine *Baseline*, die in diesem Attribut den Wert {end} gesetzt hat, wird End-Baseline genannt.

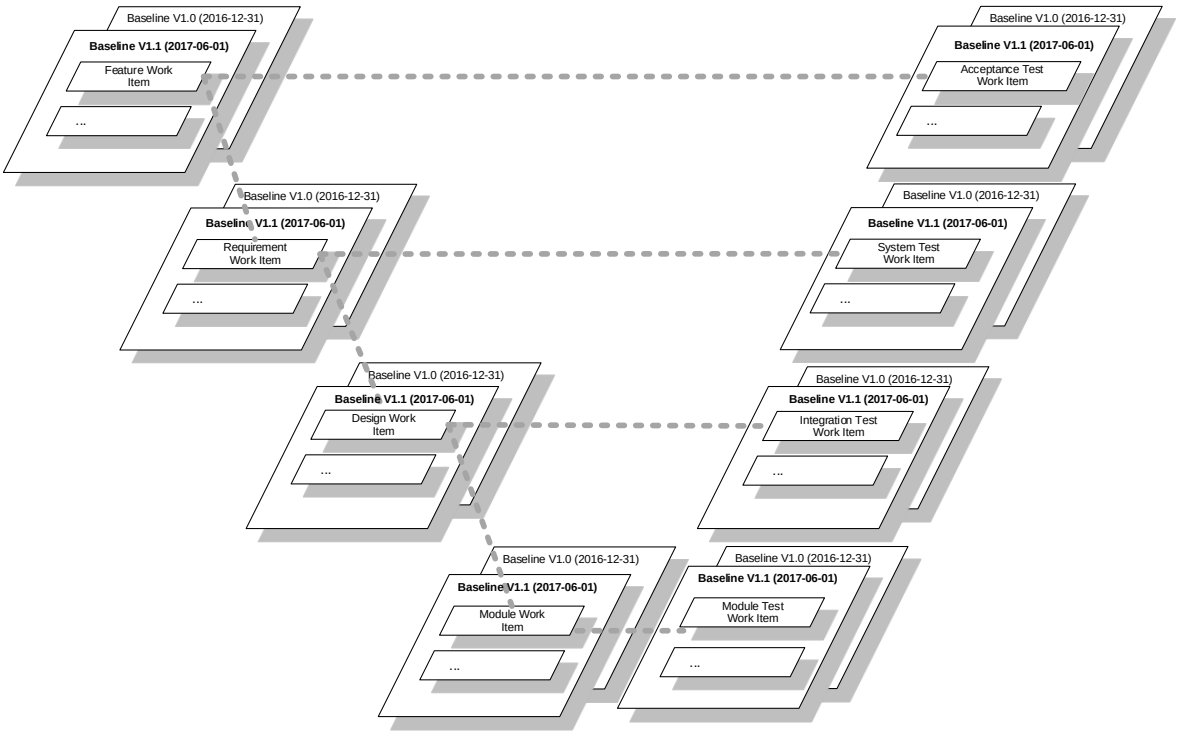


Abbildung 5.6: Baselines im Sliced V-Modell

5.2.3 Bewertung

Um zu bewerten, ob das Sliced V-Modell geeignet ist, die im Anwendungsbeispiel bestimmten SW-Produktionskennzahlen und Softwarekennzahlen zu erfassen (vgl. Abschnitt 3 und 4), erfolgt als erstes eine manuelle Prüfung, ob die Anforderungen A1 bis A10 erfüllt werden. Diese Prüfung wird in den nächsten beiden Kapiteln durch zwei weitere Prüfschritte, nämlich Erstellung der Berechnungsgrundlagen aller Kennzahlen und Implementierung eines Informationsverarbeitungssystems, vervollständigt. Die manuelle Prüfung der Erfüllung der Anforderungen ergibt folgende Ergebnisse:

A1 Zuordnung Quelltextänderungen und A4 Erweiterte Zuordnung Quelltextänderungen: Jede Revision im Repository ist entweder mit einem *Defect Work Item* oder einem *Module Work Item* verlinkt. Letztere sind immer mit einem *Feature Work Item* indirekt verlinkt, das heißt, sie sind mit *Design Work Items* verlinkt, die mit *Requirements Work Items* verlinkt sind, welche wiederum mit *Feature Work Items* verlinkt sind. Somit lässt sich eindeutig nachvollziehen, welche Quelltextänderungen aufgrund einer Fehlerbehebung bzw. für die Umsetzung einer neuen Softwarefunktion erfolgten. Sowohl die *Defect Work Items* als auch die *Module Work Items* können einer Softwareversion zugeordnet werden. Die Menge aller *Feature Set Documents* sind einem Sliced V-Modell Storage zugewiesen, das ein Softwareprodukt repräsentiert. Alle Quelltextänderungen lassen sich somit einer Softwareversion und folglich dem Softwareprodukt zuordnen. Die Unterscheidung zwischen intern und extern entdeckten Fehlern findet über die *Internal Defect Documents* bzw. *External Defect Documents* statt.

A2 Termine Softwarefunktionen: Die Zieltermine einer Softwarefunktion werden in das Attribut *dueDate* eines *Feature Work Items* eingetragen. Der Endtermin einer Softwareversion wird mit Hilfe einer End-Baseline vermerkt.

A3 Soll-/Ist-Stunden: Der Soll-Aufwand einer Softwarefunktion wird im Attribut *initialEstimate* eines *Feature Work Items* eingetragen. Die Summe aller Einträge in diesem Attribut ist der Soll-Aufwand aller für eine Softwareversion geplanten Softwarefunktionen. Die tatsächlich geleisteten Aufwände werden im Attribut *timeSpent* eines *Task Work Items* vermerkt. Jedes dieser Work Items ist einem *Feature Set Document* zugeordnet. Somit kann der Ist-Aufwand einer Softwareversion ermittelt werden.

A5 Entwicklungsaufwand und A8 Dokumentationsaufwand: Die tatsächlich geleisteten Aufwände werden im Attribut *timeSpent* in den *Task Work Items* vermerkt. Diese Work Items werden durch das Attribut *activityType* in Entwicklungs- und Dokumentationsaktivitäten unterschieden. Da jedes dieser Work Items einem *Feature Set Document* zugeordnet ist, können der Ist-Entwicklungsaufwand und der Ist-Dokumentationsaufwand einer Softwareversion ermittelt werden.

A6 Start und Ende: Der Starttermin und der Endtermin einer Softwareversion werden durch die Start-Baseline und die End-Baseline vermerkt. Die Entwicklungsdauer kann aus der Differenz der jeweiligen Kalendertage ermittelt werden.

A7 Softwaredokumentation: Ausgehend von den *Feature Work Items* in einem *Feature Set Document* sind alle für eine Softwareversion erstellten Work Items untereinander verlinkt. Sie lassen sich somit dieser Softwareversion zuordnen und können durchgezählt werden. Die intern entdeckten Fehler können aufgrund der festgelegten Syntax des Dokumentennamens des jeweiligen *Internal Defect Documents* ebenfalls einer Softwareversion zugeordnet werden. Alle Work Items gehören zu einem Sliced V-Modell Storage und lassen sich folglich einem Softwareprodukt zuordnen.

A9 Fehlerattribute: Die *Defect Work Items* besitzen die Attribute *internal*, *external* und *severity*. In diesen Attributen werden jeweils die für die einzelnen Kategorien definierten Werte eingetragen, aus denen die prozentualen Teilwerte aller Fehlerattribute berechnet werden können.

A10 Fehlerzuordnung: Alle intern entdeckten Fehler sind in einem *Internal Defect Document* als *Defect Work Items* eingetragen und können somit einer Softwareversion zugeordnet werden. Alle extern entdeckten Fehler sind in einem *External Defect Document* eingetragen und lassen sich folglich dem Softwareprodukt zuordnen. Die Anzahl der intern und extern entdeckten Fehler kann gezählt werden und auf deren Basis ist es möglich, die Fehlerbehebungsrate zu berechnen.

Die manuelle Prüfung zeigt, dass das Sliced V-Modell alle Anforderungen erfüllt, um durch ein Informationsverarbeitungssystem die bestimmten SW-Produktionskennzahlen und Softwarekennzahlen erfassen zu können. Da alle Daten eines Sliced V-Modell Storage in einem ALM-System bzw. in einem Versionsmanagementsystem gespeichert werden, ist eine IT-basierte Datenerfassung und -verarbeitung realisierbar. Wie in Abschnitt 2.1 erläutert, sollten Kennzahlen automatisiert erfasst und verarbeitet werden, da so die Kosten der Datenerfassung reduziert und die Objektivität der Kennzahlenwerte erhöht werden. Voraussetzung dafür ist die IT-basierte Erreichbarkeit der Datenquellen der Kennzahlen, was durch die Nutzung eines ALM-Systems und eines Versionsmanagementsystems in einem Sliced V-Modell gegeben ist. Sowohl ALM-Systeme als auch Versionsmanagementsysteme ermöglichen den Zugriff auf ihre Daten über API, die durch andere IT-Systeme verwendet werden können.

Nachdem das Datenmodell des Softwareentwicklungsprozesses entwickelt wurde, können die Berechnungsgrundlagen der SW-Produktionskennzahlen und der Softwarekennzahlen erstellt werden. Diesem Themenkomplex widmet sich das nächste Kapitel.

Kapitel 6

Ermittlung der Berechnungsgrundlagen

Dieses Kapitel widmet sich den Berechnungsgrundlagen der Kennzahlen. Es werden die Notwendigkeit von Berechnungsgrundlagen, das prinzipielle Vorgehen bei deren Erstellung und die Berechnungsgrundlagen der in den vorherigen Kapiteln bestimmten SW-Produktionskennzahlen und Softwarekennzahlen gezeigt. Diese Berechnungsgrundlagen setzen die Anwendung des Sliced V-Modells voraus. Zunächst werden die Berechnungsgrundlagen der Softwarekennzahlen und danach die der SW-Produktionskennzahlen aufgeführt, da einige SW-Produktionskennzahlen aus Softwarekennzahlen gebildet werden. Die Inhalte dieses Kapitels entstanden iterativ in den Design Research-Entwurfsphasen *Erstellen* und *Evaluierung* (vgl. Abschnitt 1.5).

Abbildung 6.1 zeigt die Inhalte dieses Kapitels und an welcher Stelle diese Inhalte zur Erreichung der Zielsituation dieser Arbeit beitragen. Die grauen Rechtecke markieren den Inhalt: die Berechnungsgrundlagen der Kennzahlen in K_{SW} und K_S.

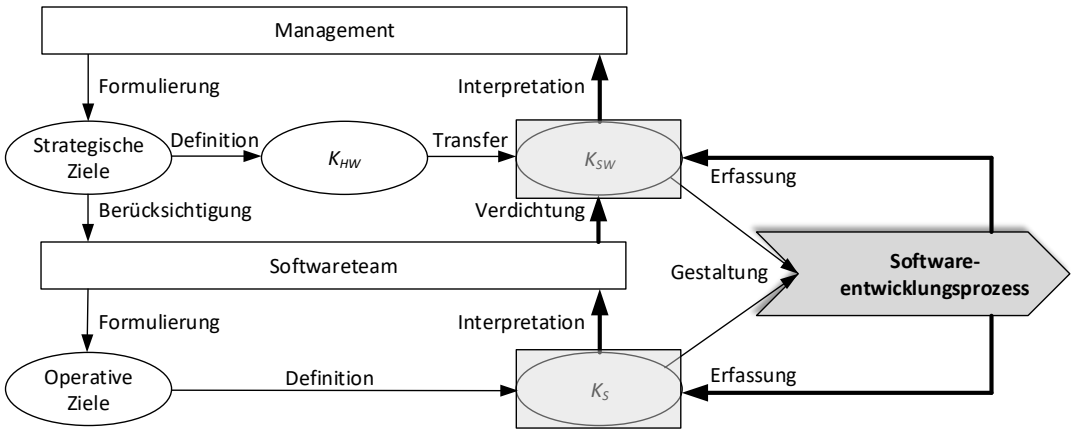


Abbildung 6.1: Inhalt des Kapitels 6 in Bezug auf die Zielsituation dieser Arbeit

6.1 Notwendigkeit und Vorgehen

Um ein Informationsverarbeitungssystem zu implementieren, ist es erforderlich, die Berechnungsgrundlagen aller Kennzahlen zu ermitteln. Aus den Berechnungsgrundlagen muss hervorgehen, auf welchen Datenquellen die Berechnung der Kennzahlen beruht. Es können folglich nur Datenquellen genutzt werden, die tatsächlich in dem Datenmodell des Softwareentwicklungsprozesses vorhanden sind. Wie in Abschnitt 5.1 erläutert, ist die Erstellung der Berechnungsgrundlagen ein Teil der Überprüfung, ob ein Datenmodell die Anforderungen an einen Softwareentwicklungsprozess erfüllt, wie sie bei der Bestimmung der Kennzahlen formuliert wurden.

Des Weiteren zeigen die Berechnungsgrundlagen, wie ein Informationsverarbeitungssystem die Daten aus den Datenquellen zu Kennzahlen verarbeiten soll. Dies gilt sowohl für die absoluten Kennzahlen, die in Einzelzahlen, Summe, Differenz und Mittelwert eingeteilt werden, als auch für Verhältniskennzahlen, die das Verhältnis von zwei oder mehr absoluten Kennzahlen anzeigen (vgl. Abschnitt 2.1.1). Um in einem Informationsverarbeitungssystem die Berechnung einer Verhältniskennzahl implementieren zu können, muss zunächst die Erfassung der darin eingehenden absoluten Kennzahlen implementiert werden.

Bei der Erstellung der Berechnungsgrundlagen der absoluten Kennzahlen wird das zugrundeliegende Datenmodell analysiert und die dazugehörenden Datenquellen identifiziert. Dies ist ein manueller Vorgang, für den keine systematische Methode bekannt ist. Sind die Datenquellen analysiert, wird eine konkrete mathematische Formel ermittelt, mit der die jeweilige Kennzahl aus den Datenquellen berechnet wird: Sie zeigt die Additions-, Differenz- oder Mittelwertberechnung (vgl. Abschnitt 2.1.1).

Um die Berechnungsgrundlagen der Verhältniskennzahlen zu ermitteln, ist eine Analyse des Datenmodells nicht notwendig, da in deren mathematischen Formeln lediglich die absoluten Kennzahlen berücksichtigt werden und somit keine direkte Datenerfassung aus dem Datenmodell erfolgt.

Die Berechnungsgrundlagen sind durch ein Review von geeigneten Experten zu prüfen. Dafür können Review-Methoden aus dem Methodenbaukasten für statische Testverfahren, beispielsweise informelles Review oder Walkthrough, angewendet werden [SL03].

In den folgenden Abschnitten werden für das Anwendungsbeispiel beim Kooperationspartner die Berechnungsgrundlagen aller bestimmten Softwarekennzahlen und SW-Produktionskennzahlen erläutert. Deren Erstellung orientiert sich an dem dargestellten Vorgehen. In allen Formeln wird auf die Angabe von Maßeinheiten verzichtet.

6.2 Berechnungsgrundlagen der Softwarekennzahlen

Alle nachfolgenden Softwarekennzahlen werden jeweils für eine Softwareversion berechnet und sollten zum Endtermin der Softwareversion erfasst werden. Dies ermöglicht die kontinuierliche Beobachtung des Softwareentwicklungsprozesses von Softwareversion zu Softwareversion. Zwar wäre es möglich, die Softwarekennzahlen zu jedem beliebigen Zeitpunkt nach dem Endtermin zu erfassen, da die Softwareartefakte dauerhaft in dem ALM-System bzw. dem Versionsmanagementsystem gespeichert werden und der Bezug der Softwareartefakte zu einer Softwareversion durch das Sliced V-Modell eindeutig hergestellt wird. Um jedoch den Softwareentwicklungsprozess kontinuierlich zu verbessern, sollte die Bewertung der Softwarekennzahlen unmittelbar nach dem Endtermin erfolgen.

Eine Ausnahme bildet die *Fehlerbehebungsrate* (vgl. Abschnitt 2.1.2.4.1): Sie wird nur für das Softwareprodukt berechnet. Der Grund dafür liegt darin, dass die in einer Softwareversion enthaltenen Fehler zum Teil erst Jahre nach dem Endtermin der Softwareversion entdeckt werden. Da zum Endtermin noch keine extern entdeckten Fehler bekannt sein können und folglich die Fehlerbehebungsrate immer 100 % beträgt, ist eine Erfassung der Fehlerbehebungsrate bezogen auf eine Softwareversion am Endtermin nicht sinnvoll. Zwar werden extern entdeckte Fehler einer Softwareversion zugeordnet, indem das Attribut *swVersion* an einem *Defect Work Item* entsprechend gefüllt wird (vgl. Abschnitt 5.2.2) und folglich wäre es möglich, nachträglich die Fehlerbehebungsrate für eine Softwareversion zu berechnen und zu bewerten. Allerdings ist zu diesem Zeitpunkt, der deutlich nach dem Endtermin der Softwareversion liegt, das Wissen über die Entwicklung dieser Softwareversion in den beteiligten Softwareteams nicht mehr so präsent wie an dem Endtermin selbst. Die Definition und Durchführung von kontinuierlichen Verbesserungsmaßnahmen wäre in dem Fall nicht ohne weiteres möglich, da sich die Rahmenbedingungen des Softwareentwicklungsprozesses bereits geändert haben können. Geänderte Rahmenbedingungen sind zum Beispiel neue Mitarbeiter im Softwareteam.

Die Fehlerbehebungsrate ist daher als ein Indikator für den Erfolg aller qualitätssichernden Maßnahmen zu betrachten, die im Laufe der Entwicklung eines Softwareproduktes erfolgten. Sie sollte in regelmäßigen zeitlichen Abständen erfasst werden, zum Beispiel in festen Zeitabstände, wie zum Beispiel einmal im Quartal, oder jeweils an einem Endtermin einer Softwareversion.

6.2.1 Churn

Der Churn als Softwarekennzahl wurde in Abschnitt 4.2.2.1 bestimmt. Mit dem Churn kann die Frage beantwortet werden, wie viel Software erstellt wurde. Da die Erstellung

der Software mehreren Aktivitäten zugeordnet werden kann, zum Beispiel den Arbeiten an einer Softwarefunktion oder der Behebung eines Fehlers, werden im Folgenden mehrere Churn-Kennzahlen eingeführt.

Im Sliced V-Modell können die Quelltextänderungen, die aufgrund der Umsetzung einer Softwarefunktion oder einer Fehlerbehebung erfolgten, eindeutig der Softwarefunktion oder dem Fehler zugeordnet werden. Zur Erläuterung soll die Abbildung 6.2 dienen: Ausgehend von dem *Feature Work Item*, in dem die Softwarefunktion formuliert ist, können die verlinkten Work Items schrittweise erkannt werden. Am Ende dieser Work Item Links befinden sich die *Module Work Items*. Sie sind mit den Revisionen verlinkt, die die Quelltextänderungen enthalten. Die *Defect Work Items* sind unmittelbar mit den Revisionen verlinkt.

Zur Messung des Churn wird ein *Unified Diff Patch* genutzt. Dies ist ein Text (vgl. Abbildung 2.5), dessen Größe in Kilobyte [KB] angegeben wird. Der Churn, der einer einzelnen Softwarefunktion zugeordnet wird, wird *Single Feature Churn* genannt. Die Gleichungen 6.1 und 6.2 zeigen dessen Berechnung. Es werden zunächst die Churns, die die Quelltextänderungen zwischen den Ständen einzelner Dateien anzeigen, erfasst und summiert. Die Dateistände sind Teil der Revisionen, die mit *Module Work Items* verlinkt sind. Die Summe bildet den *Module Churn*. Danach werden alle *Module Churns* summiert, die über eine V-Slice einem *Feature Work Item* zugeordnet werden können. Die V-Slice wird dabei von unten nach oben verfolgt.

$$Ch_m = \sum_{i=0}^{k} Ch_{file_i} \tag{6.1}$$

$$Ch_{f_s} = \sum_{j=0}^{l} Ch_{m_j} \tag{6.2}$$

mit:

- Ch_{file} *File Churn*: Summe der Größe der *Unified Diff Patches* für eine einzelne Datei (eine einzelne Datei kann mehrfach geändert werden)
- Ch_m *Module Churn*: Summe aller *File Churns*, die dem *Module Work Item* zugeordnet sind
- Ch_{f_s} *Single Feature Churn*: Summe aller *Module Churns*, die dem *Feature Work Item*zugeordnet sind
- k Anzahl aller Dateien in einer Revision, die mit einem *Module Work Item* verlinkt sind
- l Anzahl an *Module Work Items* in einer V-Slice

Ch_m und Ch_{f_s} haben die Maßeinheit KB (Kilobyte).

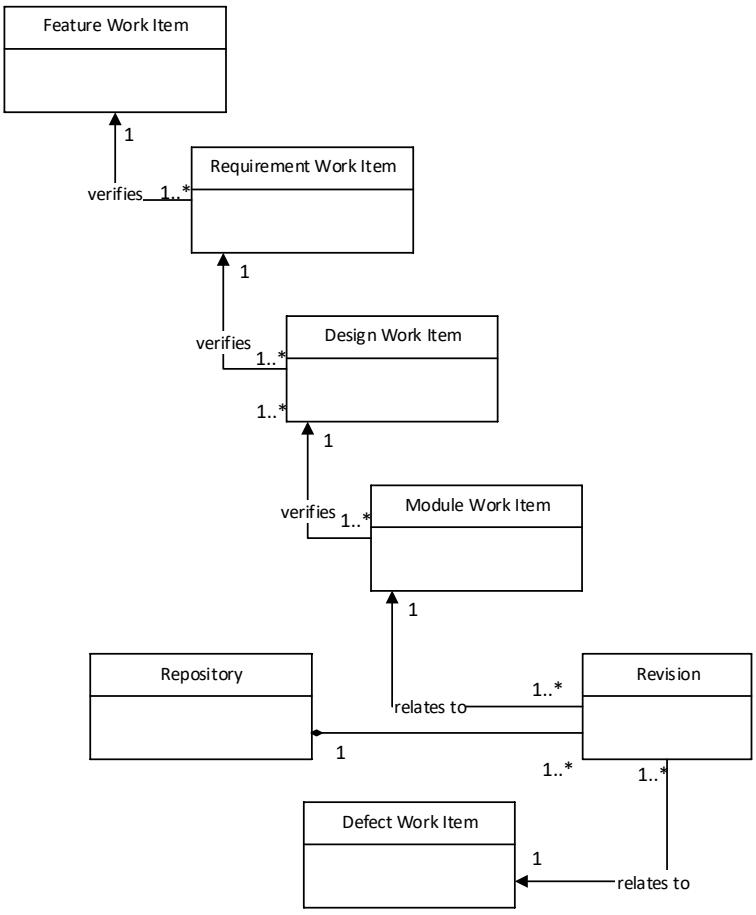


Abbildung 6.2: Zuordnung von Quelltextänderungen zu Softwarefunktionen

Für die Veranschaulichung dieser Gleichungen soll Abbildung 6.3 dienen. Sie zeigt die Ermittlung eines *Module Churn*: Um Ch_{file} zu berechnen, werden zunächst die Churns zwischen den Ständen der Dateien, die in den Revisionen geändert wurden, ermittelt. Zum Beispiel wird der Churn zwischen dem Stand der Datei A in Revision 3478 und dem Stand der Datei A in Revision 2604 sowie der Churn zwischen dem Stand der Datei C in Revision 3478 und dem Stand der Datei C in Revision 1974 ermittelt. Da der Churn jeweils zwischen zwei aufeinanderfolgenden Dateiständen ermittelt wird, ist für die niedrigste Revision mit der Nummer 1974, die mit dem analysierten *Module Work Item* verlinkt ist, festzustellen, in welcher Revision die Datei C zuletzt geändert wurde (in der Abbildung schematisch für die Datei C durch die gepunktete Linie dargestellt). Da diese Revision nicht mit dem *Module Work Item* verlinkt ist, muss ein Informationsverarbeitungssystem die entsprechenden Informationen aus dem Versionsmanagementsystem auswerten.

Nicht jede Datei in einer Revision ist eine Quelltextdatei, sondern sie kann auch eine Konfigurationsdatei sein (xml, ini etc.), die für die Umsetzung einer Softwarefunktion benötigt wird. Bei der praktischen Ausgestaltung des Informationsverarbeitungssystems ist zu entscheiden, welche Dateitypen in der Berechnung von Ch_{f_s} zu berücksichtigen sind.

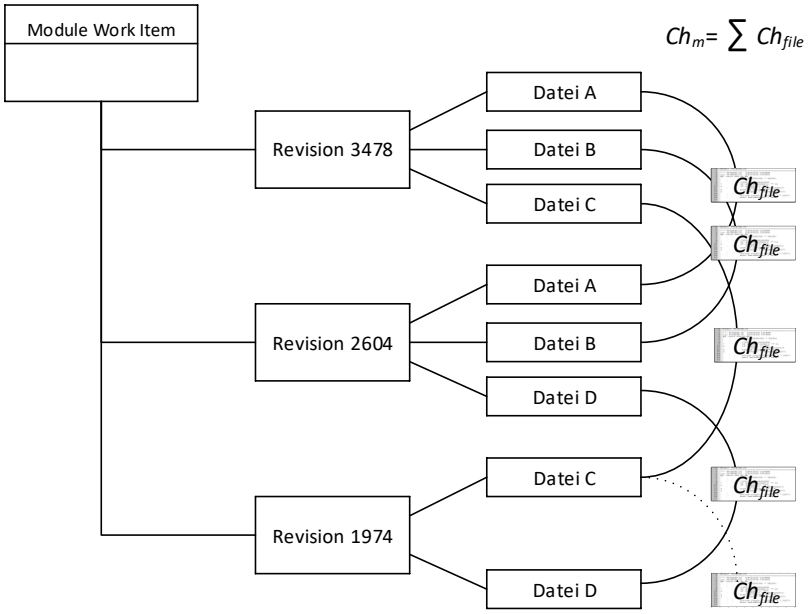


Abbildung 6.3: Ermittlung des *Module Churn*

Wie bereits in Abschnitt 2.1.2.1 im Bild 2.5 dargestellt, zeigt ein *Unified Diff Patch* sowohl Hinzufügungen und Änderungen als auch Löschungen an. Löschungen sind als Arbeitsergebnisse einer Softwareentwicklung anzusehen und sollten gemessen werden: Beispielsweise können Optimierungen des Quelltexts zur Entfernung von nicht mehr benötigten Quelltextfragmenten führen.

Auf die gleiche Weise wird der Churn ermittelt, der einer Fehlerbehebung zugeordnet wird. Er wird *Single Defect Churn* genannt und wird wie in Gleichung 6.3 berechnet.

$$Ch_{d_s} = \sum_{i=0}^{m} Ch_{file_i} \tag{6.3}$$

mit:

- Ch_{d_s} *Single Defect Churn*: Summe aller *File Churns*, die dem Work Item vom Typ „defect“ zugeordnet sind
- m Anzahl der Dateien in einer Revision (für ein *Defect Work Item*)

Ch_{d_s} hat die Maßeinheit KB (Kilobyte).

Wie in Abschnitt 2.1.2.4.1 aufgeführt, werden Fehler in intern entdeckte Fehler und in extern entdeckte Fehler unterschieden. Aufgrund dieser Unterscheidungsmöglichkeit wird der Begriff *Single Defect Churn* präzisiert: Der *Single Defect Churn* eines intern entdeckten Fehlers wird *Single Internal Defect Churn* (Ch_{di_s}) genannt, der *Single Defect Churn* eines extern entdeckten Fehlers heißt *Single External Defect Churn* (Ch_{dx_s}).

Bis zu diesem Punkt sind die Churns einer einzelnen Softwarefunktion bzw. einem einzelnen Fehler zugeordnet. Während der Entwicklung einer Softwareversion werden mehrere Softwarefunktionen umgesetzt und mehrere Fehler entdeckt. Folglich können Churn-Kennzahlen für eine Softwareversion ermittelt werden. Diese ergeben sich aus den Summen der Churn-Kennzahlen einzelner Softwarefunktionen bzw. einzelner Fehler, wie die Gleichungen 6.4 und 6.5 zeigen.

$$Ch_{f_v} = \sum_{i=0}^{n} Ch_{f_{si}} \tag{6.4}$$

$$Ch_{di_v} = \sum_{j=0}^{o} Ch_{di_{sj}} \tag{6.5}$$

mit:

- Ch_{f_v} *Version Feature Churn*: Summe aller *Single Feature Churns* der *Feature Work Items* in einem *Feature Set Document*
- Ch_{di_v} *Version Internal Defect Churn*: Summe aller *Single Internal Defect Churns* der *Defect Work Items* in einem *Internal Defect Document*
- n Anzahl der *Feature Work Items* in einem *Feature Set Document*
- o Anzahl der *Defect Work Items* in einem *Internal Defect Document*

Ch_{f_v} und Ch_{di_v} haben die Maßeinheit KB (Kilobyte).

Wie in Abschnitt 5.2.2 erläutert, werden in einem Sliced V-Modell Storage die extern entdeckten Fehler in ein dediziertes Dokument, das *External Defect Document*, eingetragen. Die darin eingetragenen Fehler sind keiner einzelnen Softwareversion, sondern dem Softwareprodukt zugeordnet. Daher zählt der Churn der Behebung extern entdeckter Fehler nicht zur Entwicklung der Softwareversion, sondern zum Softwareprodukt. Gleichung 6.6 zeigt die Berechnung.

$$Ch_{dx_p} = \sum_{i=0}^{p} Ch_{dx_{si}} \tag{6.6}$$

mit:

- Ch_{dx_p} *Product External Defect Churn*: Summe aller *Single External Defect Churns* der *Defect Work Items* in einem *External Defect Document*
- p Anzahl der *Defect Work Items* im *External Defect Document*

Ch_{dx_p} hat die Maßeinheit KB (Kilobyte).

6.2.2 Aufwand für die Entwicklungsaktivitäten

Der Aufwand für die Entwicklungsaktivitäten als Softwarekennzahl wurde in Abschnitt 4.2.2.1 bestimmt.

Wie in Abschnitt 5.2.2 erläutert, werden im Sliced V-Modell die geplanten und die tatsächlichen Aufwände in den Attributen *initialEstimate* und *timeSpent* der *Task Work Items* eingetragen. Aus den Werten in diesen Attributen werden die Soll- und Ist-Aufwände der Entwicklung einer Softwareversion ermittelt. Gleichung 6.7 zeigt deren Berechnung.

$$E_{dev_v} = \sum_{i=0}^{q} TS_{dev_i} \tag{6.7}$$

mit:

TS_{dev}	Wert im Attribut *timeSpent* in einem *Task Work Item* bei dem das Attribut *activityType* auf den Wert *development* gesetzt ist
E_{dev_v}	Ist-Aufwand der Entwicklungsaktivitäten für eine Softwareversion
q	Anzahl an *Task Work Items* im *Task Document*, bei denen das Attribut *activityType* auf den Wert *development* gesetzt ist

TS_{dev} und E_{dev_v} haben die Maßeinheit h (Stunden).

6.2.3 Entwicklungsdauer

Die Entwicklungsdauer als Softwarekennzahl wurde in Abschnitt 4.2.2.1 bestimmt. Die Entwicklungsdauer einer Softwareversion ist die absolute Anzahl an Tagen, die zwischen dem Tag, an dem die Start-Baseline erstellt wurde, und dem Tag, an dem die End-Baseline erstellt wurde, vergangen sind. Im Sliced V-Modell wird das jeweilige Datum im Attribut *blDate* des Typs *Baseline* gespeichert. Gleichung 6.8 zeigt die Berechnung.

$$D_v = B_{E_v} - B_{S_v} \tag{6.8}$$

mit:

D_v	Entwicklungsdauer einer Softwareversion
B_{E_v}	Tag der Erstellung der End-Baseline einer Softwareversion
B_{S_v}	Tag der Erstellung der Start-Baseline einer Softwareversion

D_v hat die Maßeinheit d (days).

6.2.4 Churn-Produktivität

Die Churn-Produktivität als Softwarekennzahl wurde in Abschnitt 4.2.2.1 bestimmt. Die Churn-Produktivität einer Softwareversion ist das Verhältnis der Summe des *Version Feature Churn* und des *Version Internal Defect Churn* zum Ist-Aufwand der Entwicklungsaktivitäten an dieser Softwareversion, wie Gleichung 6.9 zeigt.

$$P_{ch_v} = \frac{Ch_{f_v} + Ch_{di_v}}{E_{dev_v}} \tag{6.9}$$

mit:

P_{ch_v} Churn-Produktivität der Entwicklung einer Softwareversion

Die Maßeinheit der Churn-Produktivität ist $\frac{KB}{h}$ (Kilobyte pro Stunde).

6.2.5 Churn-Liefergeschwindigkeit

Die Churn-Liefergeschwindigkeit als Softwarekennzahl wurde in Abschnitt 4.2.2.1 bestimmt. Die Churn-Liefergeschwindigkeit einer Softwareversion ist das Verhältnis der Größe des Churn und der Entwicklungsdauer der Softwareversion. Bei der Berechnung der Churn-Liefergeschwindigkeit werden entsprechend Gleichung 6.10 sowohl der *Version Feature Churn* als auch der *Version Internal Defect Churn* berücksichtigt.

$$V_{ch_v} = \frac{Ch_{f_v} + Ch_{di_v}}{D_v} \tag{6.10}$$

mit:

V_{ch_v} Churn-Liefergeschwindigkeit einer Softwareversion

Die Maßeinheit der Churn-Liefergeschwindigkeit ist $\frac{KB}{d}$ (Kilobyte pro Tag).

6.2.6 Anzahl an Work Items

Die Anzahl an Work Items als Softwarekennzahl wurde in Abschnitt 4.2.2.2 bestimmt. In einem Sliced V-Modell können die Work Items gezählt werden, die mit einem *Feature Work Item* verlinkt sind. Diese Anzahl gibt die Dokumentationsgröße einer Softwarefunktion wieder. Gleichung 6.11 zeigt die Berechnung.

$$Doc_{f_s} = 1 + \sum_{i=0}^{r_s} i + \sum_{i=0}^{d_s} i + \sum_{i=0}^{m_s} i + \sum_{i=0}^{t_s} i \tag{6.11}$$

mit:

Doc_{f_s}	Dokumentationsgröße einer Softwarefunktion
r_s	Anzahl an *Requirements Work Items* in einer V-Slice
d_s	Anzahl an *Design Work Items* in einer V-Slice
m_s	Anzahl an *Module Work Items* in einer V-Slice
t_s	Anzahl an Work Item-Typen, die vom *Test Work Item* abgeleitet sind, in einer V-Slice

Doc_{f_s} hat die Maßeinheit *WI* (Work Items).

Das *Feature Work Item* geht als Summand 1 in die Berechnung der Dokumentationsgröße einer Softwarefunktion ein, da es ein Bestandteil einer V-Slice ist. Abbildung 6.4 veranschaulicht die Berechnung von Doc_{f_s}.

Die Dokumentationsgröße einer Softwareversion ist die Summe der Dokumentationsgrößen aller Softwarefunktionen, die in dieser Softwareversion implementiert wurden. Sie wird gemäß Gleichung 6.12 ermittelt.

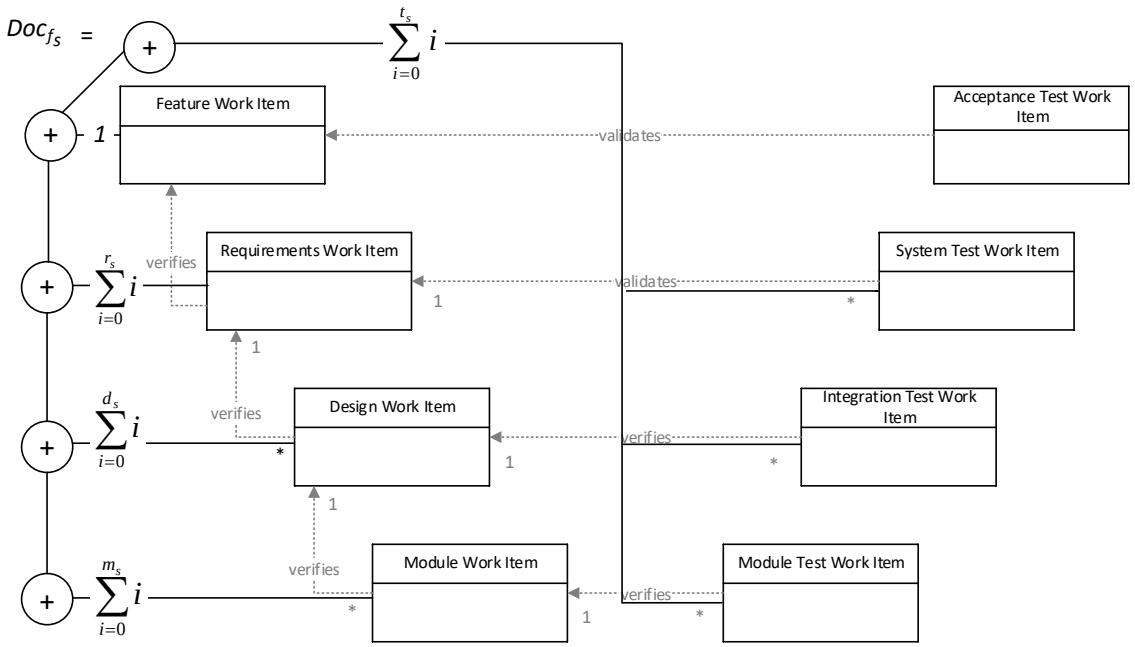


Abbildung 6.4: Ermittlung der Dokumentationsgröße einer Softwarefunktion

$$Doc_{f_v} = \sum_{i=0}^{r} Doc_{f_{s_j}} \tag{6.12}$$

mit:

Doc_{f_v}	Dokumentationsgröße einer Softwareversion
r	Anzahl aller *Feature Work Items* in einem *Feature Set Document*

Doc_{f_v} hat die Maßeinheit *WI* (Work Items).

6.2.7 Aufwand für die Dokumentationsaktivitäten

Der Aufwand für die Dokumentationsaktivitäten als Softwarekennzahl wurde in Abschnitt 4.2.2.2 bestimmt. Diese Softwarekennzahl wird in der gleichen Art und Weise wie der Aufwand für die Entwicklungsaktivitäten erfasst, jedoch werden die *Task Work Items* ausgewertet, bei denen das Attribut *activityType* auf den Wert *documentation* gesetzt ist. Gleichung 6.13 zeigt die Berechnung.

$$E_{doc_v} = \sum_{i=0}^{s} TS_{doc_i} \tag{6.13}$$

mit:

TS_{doc}	Wert im Attribut *timeSpent* in einem *Task Work Item*, bei dem das Attribut *activityType* auf den Wert *documentation* gesetzt ist
E_{doc_v}	Ist-Aufwand der Dokumentationsaktivitäten für eine Softwareversion
s	Anzahl an *Task Work Items* im *Task Document*, bei denen das Attribut *activityType* auf den Wert *documentation* gesetzt ist

TS_{doc} und E_{doc_v} haben die Maßeinheit h (Stunden).

6.2.8 Dokumentationsproduktivität

Die Dokumentationsproduktivität als Softwarekennzahl wurde in Abschnitt 4.2.2.2 bestimmt. Die Dokumentationsproduktivität einer Softwareversion ist das Verhältnis der Dokumentationsgröße einer Softwareversion zu den für die Softwareversion geleisteten Ist-Aufwänden der Dokumentationsaktivitäten. Gleichung 6.9 zeigt die Berechnung.

$$P_{doc_v} = \frac{Doc_{f_v}}{E_{doc_v}} \tag{6.14}$$

mit:

P_{doc_v} Dokumentationsproduktivität einer Softwareversion

Die Maßeinheit der Dokumentationsproduktivität ist $\frac{WI}{h}$ (Work Items pro Stunde).

6.2.9 Dokumentationsliefergeschwindigkeit

Die Dokumentationsliefergeschwindigkeit als Softwarekennzahl wurde in Abschnitt 4.2.2.2 bestimmt. Die Dokumentationsliefergeschwindigkeit einer Softwareversion ist entsprechend Gleichung 6.15 das Verhältnis der Dokumentationsgröße einer Softwareversion zu der Entwicklungsdauer der Softwareversion.

$$V_{doc_v} = \frac{Doc_{f_v}}{D_v} \tag{6.15}$$

mit:

V_{doc_v} Dokumentationsliefergeschwindigkeit einer Softwareversion

Die Maßeinheit der Dokumentationsliefergeschwindigkeit ist $\frac{WI}{d}$ (Work Items pro Tag).

6.2.10 Prozentuale Verteilung von Fehlerattributen

Die prozentuale Verteilung von Fehlerattributen wurde in Abschnitt 4.2.2.3 bestimmt. Dort sind drei verschiedene prozentuale Verteilungen genannt: die der Softwarequalitätseigenschaften, die der Fehlerursachen in der Programmierung und die der Fehlerschweregrade. Da alle drei Varianten identisch berechnet werden, zeigt dieser Abschnitt exemplarisch die Berechnungsgrundlage für eine der Varianten.

Wie in Abschnitt 2.1.2.4.1 erläutert, zeigt die prozentuale Verteilung alle prozentualen Teilwerte an. Ein prozentualer Teilwert gibt das Verhältnis der Fehler mit einer bestimmten Einordnung zur Anzahl aller Fehler wieder. Ein Beispiel eines prozentualen Teilwerts für die Fehler, die mit dem Schweregrad *kritisch* markiert sind, zeigt die Gleichung 6.16.

$$SV_{cr} = \frac{\sum_{i=0}^{cr} i}{\sum_{i=0}^{di} i} \tag{6.16}$$

mit:

di	Anzahl an *Defect Work Items* in einem *Internal Defect Document*
cr	Anzahl an *Defect Work Items* in einem *Internal Defect Document*, die den Wert „critical“ im Attribut „severity“ enthalten
SV_{cr}	Prozentualer Teilwert für *Defect Work Items* in einem *Internal Defect Document*, die den Wert *critical* im Attribut *severity* enthalten

SV_{cr} ist eine Verhältnisgröße, die in % angegeben wird.

Die Berechnung aller weiteren prozentualen Teilwerte erfolgt äquivalent zur gezeigten Berechnung. Auf die Darstellung jeder einzelnen Gleichung wird daher verzichtet.

6.2.11 Anzahl intern entdeckter Fehler

Die Anzahl intern entdeckter Fehler wurde in Abschnitt 4.2.2.4 bestimmt. Die intern entdeckten Fehler werden in den einzelnen *Internal Defect Documents* eingetragen. Die Anzahl aller für ein Softwareprodukt intern entdeckten Fehler ist folglich die Summe der Fehler in den einzelnen *Internal Defect Documents*. Gleichung 6.17 zeigt die Berechnung.

$$W_{di_p} = \sum_{i=0}^{u}(\sum_{i=0}^{di} i) \tag{6.17}$$

mit:

W_{di_p}	Anzahl an intern entdeckten Fehlern für ein Softwareprodukt
u	Anzahl aller *Internal Defect Documents* in einem Sliced V-Modell Storage

W_{di_p} hat die Maßeinheit *WI* (Work Items).

6.2.12 Anzahl extern entdeckter Fehler

Die Anzahl extern entdeckter Fehler wurde in Abschnitt 4.2.2.4 bestimmt. Die extern entdeckten Fehler werden in die einzelnen *External Defect Documents* eingetragen. Die Anzahl aller für ein Softwareprodukt extern entdeckten Fehler ist folglich die Summe der Fehler in den einzelnen *External Defect Documents*. Gleichung 6.18 zeigt die Berechnung.

$$W_{dx_p} = \sum_{i=0}^{w}(\sum_{i=0}^{dx} i) \tag{6.18}$$

mit:

W_{dx_p}	Anzahl an extern entdeckten Fehlern für ein Softwareprodukt
dx	Anzahl an *Defect Work Items* in einem *External Defect Document*
w	Anzahl an *External Defect Documents* in einem Sliced V-Modell Storage

W_{dx_p} hat die Maßeinheit WI (Work Items).

6.2.13 Fehlerbehebungsrate

Die Fehlerbehebungsrate wurde in Abschnitt 4.2.2.4 bestimmt. Sie ist das Verhältnis der Fehler, die im Zusammenhang mit qualitätssichernden Maßnahmen während der Entwicklung intern entdeckt werden, zu den Fehlern, die nach dem Endtermin einer Softwareversion vom Auftraggeber entdeckt werden (Abschnitt 2.1.2.4). Wie zu Beginn des Abschnitts 6.2 erläutert, wird die Fehlerbehebungsrate nicht für eine einzelne Softwareversion ermittelt, sondern für das Softwareprodukt. Gleichung 6.19 zeigt ihre Berechnung.

$$DFR_p = \frac{WI_{di_p}}{WI_{di_p} + WI_{dx_p}} \tag{6.19}$$

mit:

DFR_p	Fehlerbehebungsrate für ein Softwareprodukt

DFR_p ist eine Verhältnisgröße, die in % angegeben wird.

6.2.14 Churn-Fehlerdichte

Die Churn-Fehlerdichte wurde in Abschnitt 4.2.2.5 bestimmt. Sie zeigt an, wie viele intern entdeckte Fehler pro KB Churn entdeckt wurden. Bei der Berechnung der Churn-Fehlerdichte für eine Softwareversion werden entsprechend Gleichung 6.20 sowohl der *Version Feature Churn* als auch der *Version Internal Defect Churn* berücksichtigt.

$$D_{ch_v} = \frac{\sum_{i=0}^{di} i}{Ch_{f_v} + Ch_{di_v}} \tag{6.20}$$

mit:

DD_{ch_v}	Churn-Fehlerdichte einer Softwareversion

DD_{ch_v} hat die Maßeinheit $\frac{WI}{KB}$ (Work Items pro Kilobyte).

6.3 Berechnung der SW-Produktionskennzahlen

Im Abschnitt 3.2 wurde begonnen, die RGQM-Methode an einem Anwendungsbeispiel zu demonstrieren und die Anwendung der RGQM-Bearbeitungsschritte 1 bis 6 wurde erläutert. Für das Anwendungsbeispiel steht die Erläuterung der RGQM-Bearbeitungsschritte 7 und 8 aus. Diese erfolgt in diesem Abschnitt. Wie in Abschnitt 3.1.2 erläutert, werden im RGQM-Bearbeitungsschritt 7 die Berechnungsgrundlagen der SW-Produktionskennzahlen ermittelt und im RGQM-Bearbeitungsschritt 8 wird überprüft, ob die SW-Produktionskennzahlen die Interpretation der dazugehörenden HW-Produktionskennzahlen beibehalten.

In den folgenden Abschnitten werden zunächst die Berechnungsgrundlagen und die Interpretationen der SW-Produktionskennzahlen erläutert, in Abschnitt 6.4 wird die Semantik der beiden Ausprägungen einer Produktionskennzahl verglichen. In einem Gespräch mit dem Geschäftsbereichsleiter des Kooperationspartners wurde geprüft, ob die Interpretation beibehalten wird. Vorab wurden ihm die zu den SW-Produktionskennzahlen gehörenden Fragen und die jeweiligen Berechnungsformeln vorgestellt und erläutert.

Alle nachfolgenden SW-Produktionskennzahlen werden jeweils für eine Softwareversion berechnet und sollten zum Endtermin der Softwareversion erfasst werden. Dies ermöglicht die kontinuierliche Beobachtung des Softwareentwicklungsprozesses von Softwareversion zu Softwareversion. Eine Ausnahme bildet die Technische Rückläuferrate: Sie wird nur für das Softwareprodukt erfasst. Die Begründung ist identisch wie bei der Softwarekennzahl *Fehlerbehebungsrate*: Fehler in einer Softwareversion werden zum Teil erst Jahre nach dem Endtermin der Softwareversion entdeckt. Daher ist eine Berechnung der Technischen Rückläuferrate zum Endtermin einer Softwareversion nicht möglich. Sie ist wie die Fehlerbehebungsrate als ein Indikator für den Erfolg aller qualitätssichernden Maßnahmen zu betrachten, die im Laufe der Entwicklung eines Softwareproduktes erfolgten. Sie sollte ebenfalls in regelmäßigen zeitlichen Abständen erfasst werden. Die dafür möglichen Optionen wurden bereits zu Beginn des Abschnitts 6.2 erläutert.

6.3.1 First Pass Rate

Die Frage, die die First Pass Rate beantwortet soll, lautet: *Wie ist das Verhältnis von Quelltextänderungen, die der Implementierung von Softwarefunktionen zugeordnet werden können, zu allen Quelltextänderungen?*

Um die Größe der Quelltextänderungen zu messen, wird der Churn verwendet. Wie in Abschnitt 6.2.1 gezeigt, gibt es mehrere Churn-Kennzahlen: Der *Version Feature Churn* zeigt die Größe der Quelltextänderungen an, die für die Implementierung einer Software-

funktion vorgenommen wurden. Der *Version Internal Defect Churn* entspricht der Größe der Quelltextänderungen, die für die Fehlerbehebung intern entdeckter Fehler vorgenommen wurden. Diese beiden Softwarekennzahlen bilden die Gesamtgröße des Churns einer Softwareversion, der *Version Churn* (Ch_v) genannt wird. Abbildung 6.5 zeigt schematisch den Aufbau des *Version Churn*.

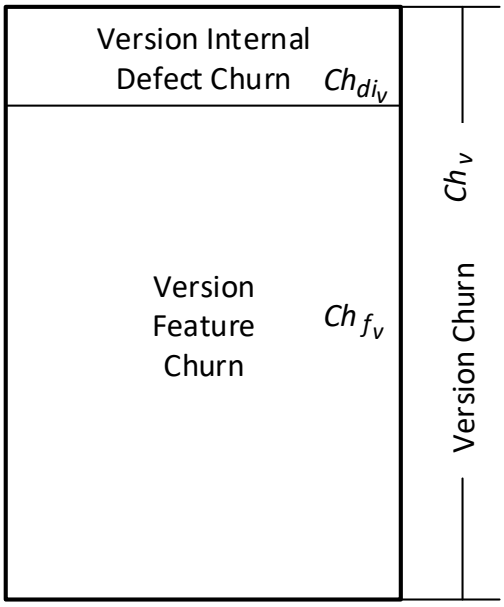


Abbildung 6.5: Aufbau des *Version Churn*

Diese Churn-Kennzahlen bilden die Basis für die Beantwortung der Frage und die Berechnung der First Pass Rate, die in Gleichung 6.21 gezeigt wird.

$$FPR_v = \frac{Ch_{f_v}}{Ch_{f_v} + Ch_{di_v}} \tag{6.21}$$

mit:

FPR_v First Pass Rate für eine Softwareversion

FPR_v ist eine Verhältnisgröße, die in % angegeben wird.

Interpretation: Der Idealwert der First Pass Rate ist 100 %. Ein hoher Ist-Wert der First Pass Rate zeigt an, dass die Fehler, die durch qualitätssichernde Maßnahmen intern entdeckt wurden, wenig Quelltextänderungen verursachen bzw. dass wenige Fehler entdeckt wurden. Beides sind Indikatoren für eine hohe Softwareentwicklungsqualität und stehen damit für geringe Nacharbeitskosten, die durch fehlerbereinigende Programmieraktivitäten anfallen.

Es wird davon ausgegangen, dass die Ist-Werte der First Pass Rate in der Softwareentwicklung kleiner sein werden als die Ist-Werte der First Pass Rate in der Produktion. Dies

wird damit begründet, dass der Automatisierungsgrad in der Produktion höher ist als in der Softwareentwicklung. Wie bereits in Abschnitt 2.2 aufgeführt, wird in der Produktion wiederholt ein identisches Produkt gefertigt, während die Softwareentwicklung einen Prozess des „ongoing design“ darstellt, der durch eine starke manuelle Bearbeitung geprägt ist. Manuelle Tätigkeiten können als fehleranfälliger angesehen werden als automatisierte Bearbeitungsvorgänge. Unabhängig von den tatsächlichen Ist-Werten ist es möglich, Abweichungen von den Soll-Werten zu erkennen und über die Umsetzung von Maßnahmen zur Verbesserung der First Pass Rate zu entscheiden.

6.3.2 Technische Rückläuferrate

Die Frage, die die Technische Rückläuferrate beantworten soll, lautet: *Wie ist das Verhältnis von Quelltextänderungen, die der Implementierung extern entdeckter Fehler zugeordnet werden können, zu allen Quelltextänderungen?*

Der *Product External Defect Churn* zeigt die Größe der Quelltextänderungen an, die für die Fehlerbehebung extern entdeckter Fehler vorgenommen wurden. Diese Churn-Kennzahl wird nicht für eine einzelne Softwareversion ermittelt, sondern für das Softwareprodukt.

Die Größe der Quelltextänderungen für ein Softwareprodukt, die durch die Entwicklung neuer Softwarefunktionen und die Behebung intern entdeckter Fehler erfolgen, wird durch die Summe aller *Version Feature Churns* und aller *Version Internal Defect Churns* angezeigt. Die jeweiligen Summenkennzahlen werden *Product Feature Churn* (Ch_{f_p}) bzw. *Product Internal Defect Churn* (Ch_{di_p}) genannt. Deren Addition ergibt den sogenannten *Product Churn* (Ch_p). Abbildung 6.6 zeigt schematisch den Aufbau des *Product Churn*.

Die genannten Churn-Kennzahlen bilden die Basis für die Beantwortung der Frage und die Berechnung der Technischen Rückläuferrate, die in Gleichung 6.22 gezeigt wird.

$$TRR_p = \frac{Ch_{dx_p}}{\sum_{i=1}^{w}(Ch_{f_{vi}} + Ch_{di_{vi}}) + Ch_{dx_p}} \tag{6.22}$$

mit:

TRR_p	Technische Rückläuferrate für ein Softwareprodukt
w	Anzahl an Softwareversionen

TRR_p ist eine Verhältnisgröße, die in % angegeben wird.

Interpretation: Der Idealwert der Technischen Rückläuferrate ist 0 %. Ein kleiner Ist-Wert der Technischen Rückläuferrate zeigt an, dass extern entdeckte Fehler wenig Nach-

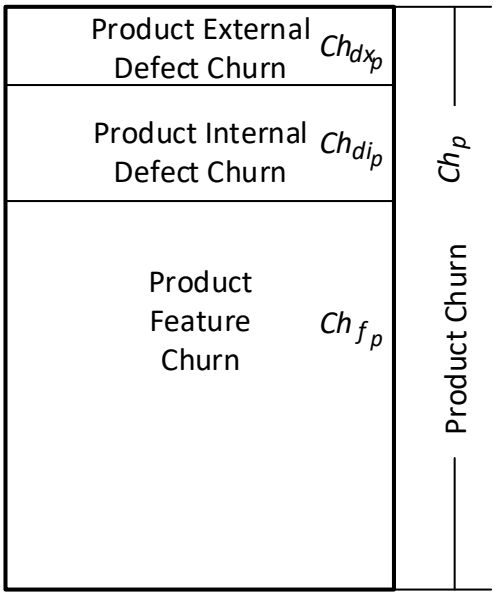


Abbildung 6.6: Aufbau des *Product Churn*

arbeiten bewirkt haben bzw. dass wenige Fehler entdeckt wurden und somit die Nacharbeitskosten für fehlerbereinigende Programmieraktivitäten gering waren.

Es kann an dieser Stelle nicht eingeschätzt werden, wie stark die Ist-Werte der Technischen Rückläuferrate in der Softwareentwicklung von den Ist-Werten der Technischen Rückläuferrate in der Produktion abweichen werden. Die Ist-Werte in der Softwareentwicklung könnten die heutigen geringen Ist-Werte in der Produktion erreichen, sofern die Qualität der ausgelieferten Software ausreichend hoch ist. Durch einen Vergleich von Soll- und Ist-Werten kann das Management über die Umsetzung von Maßnahmen zur Verbesserung der Technischen Rückläuferrate entscheiden.

6.3.3 Servicegrad

Die Frage, die der Servicegrad beantworten soll, lautet: *Wie ist das Verhältnis von Softwarefunktionen, die zum zugesagten Termin geliefert wurden, zu allen gelieferten Softwarefunktionen?*

Im Sliced V-Modell enthalten die *Feature Work Items* das Attribut *dueDate*, in dem der zugesagte Termin der Softwarefunktion notiert wird. Um den Servicegrad einer Softwareversion zu ermitteln, werden alle in einem *Feature Set Document* enthaltenen *Feature Work Items*, in denen das *dueDate* dem Datum der End-Baseline entspricht bzw. nach dem Datum der End-Baseline liegt, mit allen *Feature Work Items* in ein Verhältnis gesetzt.

Gleichung 6.23 zeigt die dazugehörende Berechnung:

$$SL_v = \frac{\sum_{i=0}^{fc_v} i}{\sum_{i=0}^{f_v} i} \tag{6.23}$$

mit:

SL_v	Servicegrad für eine Softwareversion
f_v	Anzahl an Softwarefunktionen, die in einer Softwareversion realisiert wurden (erledigte *Feature Work Items* in dem *Feature Set Dokument* der entsprechenden Softwareversion)
fc_v	Anzahl an Softwarefunktionen, die in einer Softwareversion und zum zugesagten Termin realisiert wurden (*Feature Work Items*, bei denen das Datum im Attribut *dueDate* nach dem Endtermin liegt bzw. mit dem Endtermin identisch ist)

SL_v ist eine Verhältnisgröße, die in % angegeben wird.

Interpretation: Der Idealwert des Servicegrades ist 100 %. Das würde bedeuten, dass alle Softwarefunktionen einer Softwareversion spätestens zum zugesagten Termin freigegeben sind. Ein niedriger Ist-Wert des Servicegrades ist ein Indikator dafür, dass mehrere Softwarefunktionen nicht zum zugesagten Termin freigegeben sind und somit die Lieferqualität beeinträchtigt ist. Ebenso wie ein niedriger Wert des Servicegrads in der Produktion, zeigt ein niedriger Ist-Wert des Servicegrades in der Softwareentwicklung nicht an, wie groß die Lieferverzögerungen sind. Durch einen Vergleich von Soll- und Ist-Werten kann das Management über die Umsetzung von Maßnahmen zur Verbesserung der Ist-Werte entscheiden.

6.3.4 Wertschöpfung

Die Frage, die die Wertschöpfung beantworten soll, lautet: *Wie hoch sind die geplanten Kosten für die Entwicklung neuer Softwarefunktionen?*

Der geplante Entwicklungsaufwand für die Umsetzung einer neuen Softwarefunktion, die im Sliced V-Modell mit einem *Feature Work Item* beschrieben wird, wird in dem Attribut *initialEstimate* in den *Task Work Items* eingetragen, die mit dem *Feature Work Item* verlinkt sind. Bei der Ermittlung der Wertschöpfung werden sowohl die geplanten Entwicklungsaktivitäten, also die *Task Work Items* mit dem Wert *development* im Attribut *activityType*, als auch die geplanten Dokumentationsaktivitäten, also die *Task Work Items*

mit dem Wert *documentation* im Attribut *activityType*, berücksichtigt. Um die geplanten Softwareentwicklungskosten zu berechnen, werden alle Werte im Attribut *initialEstimate* summiert und mit einem betriebsspezifischen Stundensatz multipliziert, wie Gleichung 6.24 zeigt:

$$AV_v = \sum_{i=0}^{x} IE_{f_i} \cdot H_r \tag{6.24}$$

mit:

AV_v	Wertschöpfung für eine Softwareversion
IE_f	Geplanter Aufwand einer Aktivität, die mit der Realisierung einer Softwarefunktion zusammenhängt (Wert im Attribut *initialEstimate* eines *Task Work Items*, das mit einem *Feature Work Item* verlinkt ist)
H_r	Betriebsspezifischer Stundensatz
x	Anzahl aller *Task Work Items*, die mit *Feature Work Items* verlinkt sind, die in einem *Feature Set Document* stehen

Die Maßeinheit für AV_v ist € (Euro).

Interpretation: Die Wertschöpfung in der Softwareentwicklung entspricht den Entwicklungsplankosten einer Softwareversion. Die aus den geplanten Aufwänden der einzelnen Aktivitäten resultierende Wertschöpfung in der Softwareentwicklung hat ebenso wie die Wertschöpfung in der Produktion keinen Wertebereich: Ihre Größe hängt von den konkreten Softwarefunktionen bzw. dem zu fertigenden Produkt ab. Äquivalent zur Wertschöpfung in der Produktion ist die Wertschöpfung in der Softwareentwicklung kein Indikator dafür, ob die Softwarefunktionen einen Kundennutzen bieten.

Die Wertschöpfung in der Softwareentwicklung kann in der Preisbildung der Softwareprodukte berücksichtigt werden: Die Einnahmen aus den verkauften Softwareprodukten sollten idealerweise höher als die Wertschöpfung sein.

Äquivalent zur Wertschöpfung in der Produktion wird die Wertschöpfung in der Softwareentwicklung benötigt, um die Produktivität zu berechnen.

6.3.5 Produktivität

Die Frage, die die Produktivität beantworten soll, lautet: *Wie ist das Verhältnis von der Wertschöpfung zu den geleisteten Ist-Stunden, die der Entwicklung der Softwareversion direkt zugeordnet werden können?*

Die Ist-Stunden der Entwicklung einer Softwareversion entsprechen in einem Sliced V-Modell der Summe der Werte in dem Attribut *timeSpent* aller *Task Work Items*, die für eine Softwareversion angelegt wurden. Dies sind der Aufwand der Entwicklungsaktivitäten (Abschnitt 6.2.2) und der Aufwand der Dokumentationsaktivitäten Abschnitt 6.2.7. Diese Ist-Stunden werden nach Gleichung 6.25 in ein Verhältnis zu der Wertschöpfung gesetzt.

$$P_v = \frac{AV_v}{E_{dev_v} + E_{doc_v}} \tag{6.25}$$

mit:

P_v Produktivität für eine Softwareversion

Die Maßeinheit für P_v ist $\frac{€}{h}$ (Euro pro Stunde).

Interpretation: Die Produktivität sollte in einem Team tendenziell unverändert bleiben oder steigen. Sinkt die Produktivität, könnte das ein dafür Indikator sein, dass die Mitarbeiter zu einem wachsenden Teil ihrer Arbeitszeit nicht an der Umsetzung neuer Softwarefunktionen arbeiten. Da in der Softwareentwicklung keine Maschinenkosten in der Wertschöpfung berücksichtigt werden, beträgt der ideale Wert der Produktivität: *Betriebsspezifischer Stundensatz/h* oder höher. Ein Beispiel: Wenn der betriebsspezifische Stundensatz 70 € beträgt, ist der Idealwert für die Produktivität $70\frac{€}{h}$. In diesem Fall zeigt die Produktivität an, dass sämtliche Aufwände in der Softwareentwicklung für die Umsetzung neuer Softwarefunktionen aufgebracht werden. Werte unter $70\frac{€}{h}$ könnten ein Indikator dafür sein, dass die Softwareteams einen Teil ihrer Arbeitszeit nicht wertschöpfend einsetzen.

6.4 Bewertung der semantischen Äquivalenz

Ergänzend zu den Berechnungsgrundlagen der SW-Produktionskennzahlen erfolgt eine Prüfung der semantischen Äquivalenz der beiden Ausprägungen einer Produktionskennzahl. Gemäß den Ausführungen in Abschnitt 2.1.1 bilden der Name, die Maßeinheit, der Wertebereich, der Idealwert, die Möglichkeit der Festlegung von Soll-Werten, die Frage, das Ziel und die Interpretation die Semantik einer Kennzahl.

Um die semantische Äquivalenz der beiden Ausprägungen einer Produktionskennzahl zu prüfen, werden diese beschreibenden Informationen in Tabelle 6.1 gegenübergestellt. Da die jeweiligen Interpretationen in den vorhergehenden Abschnitten dargestellt wurden, wird auf den Abschnitt verwiesen, in dem die Interpretation der jeweiligen HW-Produktionskennzahl bzw. SW-Produktionskennzahl aufgeführt wurde.

	Produktion	**Softwareentwicklung**
First Pass Rate		
Maßeinheit	%	%
Wertebereich	0...100 %	0...100 %
Idealwert	100 %	100 %
Festlegung Soll-Werte	möglich	möglich
Ziel	Hohe Fertigungsqualität	Hohe Softwareentwicklungsqualität
Frage	Wie ist das Verhältnis von gefertigten Produkten, die fehlerfrei getestet wurden, zu allen gefertigten Produkten?	Wie ist das Verhältnis von Quelltextänderungen, die der Implementierung von Softwarefunktionen zugeordnet werden können, zu allen Quelltextänderungen?
Interpretation	Abschnitt 3.2.2.2.1	Abschnitt 6.3.1
	Die Interpretationen sind im Grundsatz identisch. Durch einen Vergleich von Soll-Werten und Ist-Werten entscheidet das Management über die Umsetzung von Maßnahmen zur Verbesserung der Ist-Werte. Die Soll-Werte können allerdings unterschiedlich sein.	
Technische Rückläuferrate		
Maßeinheit	%	%
Wertebereich	0...100 %	0...100 %
Idealwert	0 %	0 %
Festlegung Soll-Werte	möglich	möglich
Ziel	Hohe Kundenzufriedenheit	Hohe Kundenzufriedenheit
Frage	Wie ist das Verhältnis von ausgelieferten Produkten, die aufgrund eines technischen Defekts reklamiert werden, zu allen ausgelieferten Produkten?	Wie ist das Verhältnis von Quelltextänderungen, die der Implementierung extern entdeckter Fehler zugeordnet werden können, zu allen Quelltextänderungen?
Interpretation	Abschnitt 3.2.2.2.2	Abschnitt 6.3.2
	Die Interpretationen sind im Grundsatz identisch. Durch einen Vergleich von Soll-Werten und Ist-Werten entscheidet das Management über die Umsetzung von Maßnahmen zur Verbesserung der Ist-Werte. Die Soll-Werte können allerdings unterschiedlich sein.	

Fortsetzung der Tabelle auf der nächsten Seite...

Servicegrad

Maßeinheit	%	%
Wertebereich	0...100 %	0...100 %
Idealwert	100 %	100 %
Festlegung Soll-Werte	möglich	möglich
Ziel	Hohe Lieferqualität	Hohe Lieferqualität
Frage	Wie ist das Verhältnis von Auftragspositionen, die zum Bestätigungstermin geliefert wurden, zu allen Auftragspositionen?	Wie ist das Verhältnis von Softwarefunktionen, die zum zugesagten Termin geliefert wurden, zu allen gelieferten Softwarefunktionen?
Interpretation	Abschnitt 3.2.2.2.3	Abschnitt 6.3.3
	Die Interpretationen sind im Grundsatz identisch. Durch einen Vergleich von Soll-Werten und Ist-Werten entscheidet das Management über die Umsetzung von Maßnahmen zur Verbesserung der Ist-Werte. Die Soll-Werte können allerdings unterschiedlich sein.	

Wertschöpfung

Maßeinheit	€	€
Wertebereich	spezifisch	spezifisch
Idealwert	spezifisch	spezifisch
Festlegung Soll-Werte	möglich	möglich
Ziel	Hohe Fertigungsrentabilität	Hohe Softwareentwicklungsrentabilität
Frage	Wie hoch sind die Fertigungsplankosten (ohne Berücksichtigung der Materialkosten)?	Wie hoch sind die geplanten Kosten für die Entwicklung neuer Softwarefunktionen?
Interpretation	Abschnitt 3.2.2.2.4	Abschnitt 6.3.4
	Die Interpretation ist im Grundsatz identisch. Das Management kann erkennen, wie hoch die wertschöpfend eingesetzten Kosten sind. Die Wertschöpfung kann und sollte in der Preisbildung berücksichtigt werden.	

Fortsetzung der Tabelle auf der nächsten Seite...

Produktivität

Maßeinheit	€/h	€/h
Wertebereich	spezifisch	spezifisch
Idealwert	spezifisch	spezifisch
Festlegung Soll-Werte	möglich	möglich
Ziel	Hohe Fertigungsrentabilität	Hohe Softwareentwicklungsrentabilität
Frage	Wie ist das Verhältnis von der Wertschöpfung zu den Ist-Stunden, die der Fertigung des Produktes direkt zugeordnet werden können?	Wie ist das Verhältnis von der Wertschöpfung zu den geleisteten Ist-Stunden, die der Entwicklung der Softwareversion direkt zugeordnet werden können?
Interpretation	Abschnitt 3.2.2.2.5	Abschnitt 6.3.5
	Die Interpretation ist im Grundsatz identisch. Das Management kann die Trends der Produktivität beobachten und erkennen, ob die überwiegende Arbeitszeit der Mitarbeiter wertschöpfend eingesetzt wird. Die konkreten Ist-Werte der Produktivität sind allerdings in den beiden Domänen unterschiedlich.	

Tabelle 6.1: Semantische Äquivalenz der HW- und SW-Produktionskennzahlen

Bei der Prüfung der semantischen Äquivalenz der jeweiligen Ausprägung einer Produktionskennzahl kann zusammenfassend Folgendes festgestellt werden:

1. Die Maßeinheiten sind identisch.
2. Bei den Produktionskennzahlen mit einem vorgegebenen Wertebereich sind sowohl der Wertebereich als auch der Idealwert innerhalb des Wertebereichs identisch.
3. Die Soll-Werte können in den beiden Domänen unterschiedlich sein.
4. Die Ziele sind identisch, enthalten jedoch zum Teil domänenspezifische Begriffe.
5. Die Fragen sind im Satzbau identisch. Die Frageninhalte unterscheiden sich.
6. Die SW-Produktionskennzahl erlaubt die im Grundsatz gleiche Interpretation wie die dazugehörende HW-Produktionskennzahl.

Die unter Punkt 1, 2 und 4 aufgeführten Erkenntnisse zeigen, dass die jeweiligen Semantikmerkmale gleich sind. Punkt 3 besagt, dass die Soll-Werte in der Produktionsdomäne und die Soll-Werte in der Softwaredomäne unterschiedlich sein können. Dies spricht einer

semantischen Äquivalenz nicht entgegen: Das Management legt bereits heute unterschiedliche Soll-Werte für eine bestimmte HW-Produktionskennzahl für unterschiedliche Produktionsstätten fest. Jede Produktionsstätte zeichnet sich durch spezifische Eigenschaften aus, daher können in der Regel nicht die gleichen Soll-Wertvorgaben gelten. Die Steuerung der Produktionsstätten erfolgt durch den Vergleich von Ist- und Soll-Werten. Aus diesem Grund kann das Management beliebige Soll-Werte für die SW-Produktionskennzahlen festlegen: Um die Zielerreichung zu überprüfen, ist auch hier nicht der Vergleich des Ist-Wertes mit dem Idealwert von 100 % relevant, sondern der Vergleich des Ist-Wertes mit dem vom Management vorgegebenen Soll-Wert.

Die Erkenntnis unter Punkt 5 ließe die Schlussfolgerung zu, dass die beiden Ausprägungen einer Produktionskennzahl semantisch nicht äquivalent sind. Es wird jedoch argumentiert, dass die Semantik der Frage im Grundsatz identisch und damit semantisch äquivalent ist, da der Satzbau identisch ist. Zwar sind die Frageninhalte unterschiedlich, dennoch unterstützt der Satzbau eine im Grundsatz identische Interpretation. Dies soll am Beispiel der First Pass Rate dargelegt werden: Die Frage, die beantwortet wird, ist eine Frage zu einem Verhältnis von etwas „Gutem“ zu etwas „Schlechtem“. Während das Management diese Frageninhalte für die Produktionsdomäne kennt, ist es nicht zwingend notwendig, dass es auch die entsprechenden Frageninhalte der Frage für die SW-Produktionskennzahl kennt. Es muss lediglich verstehen, dass das in der Frage beschriebene Verhältnis der Frageninhalte zu dem Idealwert tendieren sollte.

Basierend auf den Ausführungen in diesem Abschnitt lässt sich schlussfolgern, dass für alle fünf Produktionskennzahlen die SW-Produktionskennzahl und die dazugehörende HW-Produktionskennzahl semantisch äquivalent sind. Die RQGM-Methode ist geeignet, um SW-Produktionskennzahlen zu bestimmen und wird folglich als eine Antwort auf die erste Detailfrage dieser Arbeit angesehen: *Wie können SW-Produktionskennzahlen, die die Semantik der äquivalenten HW-Produktionskennzahlen beibehalten, bestimmt werden?*

Als Antwort auf die zweite Detailfrage: *Wie sollte der Softwareentwicklungsprozess aufgebaut sein, damit die definierten SW-Produktionskennzahlen und Softwarekennzahlen erfasst werden können?* wird das Sliced V-Modell angesehen. In Abschnitt 5.2.3 wurde durch eine manuelle Prüfung eine erste Bestätigung geliefert, dass das Sliced V-Modell die in den Kapiteln 3 und 4 formulierten Anforderungen grundsätzlich erfüllt. Durch die Ermittlung der Berechnungsgrundlagen aller Kennzahlen in diesem Kapitel erfolgte eine weitere Bestätigung. Die abschließende Bestätigung erfolgt durch die Implementierung eines Informationsverarbeitungssystems, mit der die dritte Detailfrage beantwortet werden soll: *Wie sollte ein Informationsverarbeitungssystem aufgebaut sein, das SW-Produktionskennzahlen und Softwarekennzahlen erfassen und verarbeiten kann?* Der Implementierung eines Informationsverarbeitungssystems widmet sich das folgende Kapitel.

Kapitel 7

Entwicklung des Informationsverarbeitungssystems

Dieses Kapitel widmet sich der dritten Detailfrage dieser Arbeit: *Wie sollte ein Informationsverarbeitungssystem aufgebaut sein, das Softwarekennzahlen und SW-Produktionskennzahlen erfassen und verarbeiten kann?* sowie der Evaluierung der Forschungsfrage. Dafür wird ein Prototyp eines Informationsverarbeitungssystems, genannt *SofProSys*, entwickelt, der sowohl praxisnah als auch praktisch angewendet wird. Außerdem wird die Einhaltung der empfohlenen Gestaltungsgrundsätze für Informationsverarbeitungssysteme (vgl. Abschnitt 2.1) bewertet. Die Inhalte dieses Kapitels entstanden iterativ in den Design Research-Entwurfsphasen *Erstellen* und *Evaluierung* (vgl. Abschnitt 1.5).

Abbildung 7.1 zeigt, welche Inhalte in diesem Kapitel behandelt werden und an welcher Stelle diese Inhalte zur Erreichung der Zielsituation dieser Arbeit beitragen. Das graue Rechteck markiert den Inhalt: die Erfassung der Kennzahlen für K_{SW} und K_S und deren Interpretation durch das Management bzw. durch die Softwareteams.

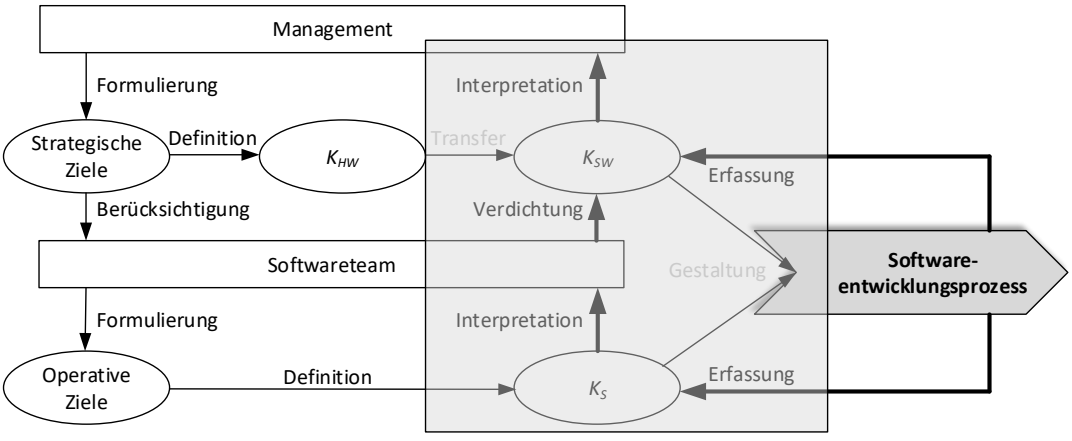


Abbildung 7.1: Inhalt des Kapitels 7 in Bezug auf die Zielsituation dieser Arbeit

7.1 Prototyp

Das Ziel der Entwicklung und Anwendung des Prototyps ist es, eine Lösung für die dritte Detailfrage zu erarbeiten. Der Prototyp muss folglich in der Lage sein, Softwarekennzahlen und SW-Produktionskennzahlen zu erfassen und zu verarbeiten. Das dieser Kennzahlenerfassung und -verarbeitung zugrunde liegende Datenmodell ist das Sliced V-Modell. Die Softwareartefakte eines Sliced V-Modells werden mit einem ALM-System und einem Versionsmanagementsystem verwaltet. Bei der Implementierung von *SofProSys* müssen folglich die technischen Eigenschaften dieser Collaboration Tools, wie zum Beispiel deren API, berücksichtigt werden. Es ist nicht Anspruch dieser Arbeit, ein allgemein anwendbares Informationsverarbeitungssystem zu entwerfen, das die Daten aus verschiedenen ALM-Systemen bzw. Versionsmanagementsystemen verarbeiten kann. Es ist vielmehr das Ziel, die Eigenschaften eines Informationsverarbeitungssystems als Lösung der dritten Detailfrage an einem konkreten Beispiel aufzuzeigen. *SofProSys* erfasst die Daten aus den beim Kooperationspartner eingesetzten Collaboration Tools. Wie bereits in Abschnitt 2.5.2.2 erläutert, sind dies *Polarion ALM* und *Subversion*.

Damit der Prototyp als eine Lösung für die dritte Detailfrage angesehen werden kann, muss er die folgenden Anforderungen erfüllen:

Anforderung 1 (Korrektheit der Kennzahlen) *Alle erfassten Kennzahlen müssen den im ALM-System und im Versionsmanagementsystem gespeicherten Daten entsprechen.*

Anforderung 2 (Erfassung aller Kennzahlen) *Alle in den vorherigen Kapiteln definierten Softwarekennzahlen und SW-Produktionskennzahlen müssen erfasst werden können und sie müssen plausibel sein, d.h. sie sind ein reales Abbild der Softwareentwicklung.*

Sofern die Anforderung 2 erfüllt wird, wird dies neben der manuellen Prüfung (vgl. Abschnitt 5.2.3) und der Erstellung der Berechnungsgrundlagen (vgl. Kapitel 6) als abschließende Bestätigung angesehen, dass das Sliced V-Modell alle Anforderungen an seine Gestaltung erfüllt.

SofProSys wurde mit der Programmierumgebung MS Visual Studio in der Programmiersprache C# erstellt. Es handelt sich dabei um eine sogenannte Konsolenanwendung. Damit ist gemeint, dass *SofProSys* keine grafische Benutzeroberfläche aufweist. Sämtliche Konfigurationsinformationen, wie zum Beispiel der Name des auszuwertenden Sliced V-Modell Storage, werden in der Konsole vorgenommen. Abbildung 7.2 vermittelt einen Eindruck über die Bedienung von *SofProSys*.

Die eingesetzten Collaboration Tools verfügen über folgende Schnittstellen: Auf die in *Polarion ALM* gespeicherten Daten wird mit Hilfe einer SQL-Datenbank und Webservices

```
Check: Excel is installed.

*** Welcome to SofProSys ***
Enter project name (x for Abort):
- 0: SofProSys
- 5: Test
0
Extract internal defect module from CCB document name automatically? (y/n):y

Analyze development tasks automatically? (y/n):y

Enter user name:up0-a9g
Enter password:*********

Connect to Polarion database
Succesfully connected to Polarion database...
Analyze CCB Document:
```

Abbildung 7.2: *SofProSys*-Screenshot

zugegriffen. Die in *Subversion* gespeicherten Daten werden mit Hilfe der *Subversion*-API erfasst. *SofProSys* erfasst schrittweise die Daten aus beiden Collaboration Tools unter Anwendung dieser Schnittstellen, verarbeitet sie und speichert die Ergebnisse in mehreren MS Excel-Dateien. Abbildung 7.3 zeigt den *SofProSys*-Systemkontext.

Eine grafische Aufbereitung der erfassten und verarbeiteten Daten ist nicht im Funktionsumfang von *SofProSys* enthalten. Sofern eine grafische Aufbereitung während der praxisnahen bzw. praktischen Anwendung benötigt wurde, erfolgte die Aufbereitung der Daten mit MS Excel-Bordmitteln.

Wie in Abschnitt 5.2.1 erwähnt, basiert das Sliced V-Modell auf internen Prozessbeschreibungen des Kooperationspartners. In den internen Prozessbeschreibungen werden betriebsspezifische Begriffe genutzt, die im Sliced V-Modell aus Gründen der Abstraktion nicht verwendet wurden. Während der Entwicklung von *SofProSys* mussten diese betriebsspezifischen Begriffe jedoch verwendet werden, da zum einen nur mit diesen die Datenbankabfragen korrekt funktionieren und zum anderen die Gesprächspartner mit den ihnen vertrauten Termini in der praktischen Anwendung arbeiten können.

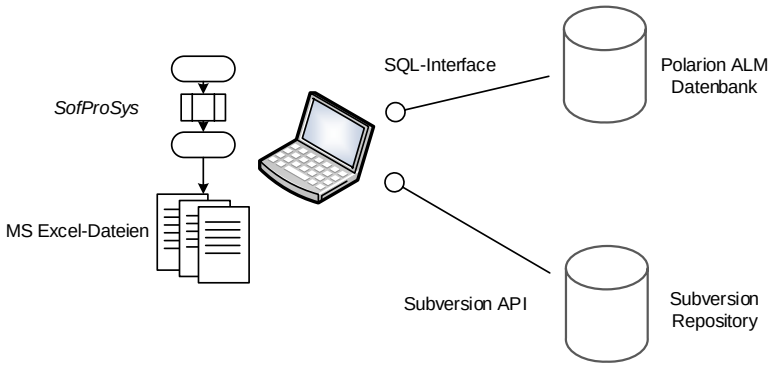


Abbildung 7.3: *SofProSys*-Systemkontext

Dem Datenmodell von *SofProSys* liegt das Sliced V-Modells zugrunde. In der Entwicklung wurden das beim Kooperationspartner eingesetzte ALM-System und das Versionsmanagementsystem verwendet. Im Sliced V-Modell Storage von *SofProSys* wurden die Termini des Kooperationspartner verwendet. *SofProSys* wurde im Laufe dieser Arbeit kontinuierlich über mehrere Jahre entwickelt. In dieser Zeit entstanden drei Softwareversionen.

7.2 Praxisnahe Anwendung

Die praxisnahe Anwendung wurde vom Verfasser dieser Arbeit durchgeführt. Sie erfolgte auf Basis der Entwicklungsdaten von *SofProSys*. Da *SofProSys* auf dem Sliced V-Modell basiert, können die Softwareartefakte des dazugehörigen Sliced V-Modell Storage durch *SofProSys* erfasst und verarbeitet werden. Diese Form der Anwendung ist praxisnah, da es sich bei dem untersuchten Softwareprodukt nicht um ein reales Softwareprodukt eines produzierenden Betriebes handelt. Um die Entwicklungsarbeiten zu unterstützen, wurde ein sogenanntes Dummy Sliced V-Modell genutzt. Ein Dummy Sliced V-Modell ist ein Sliced V-Modell Storage, das zwar alle Softwareartefakte eines Sliced V-Modells in einer ausreichenden Menge enthält, allerdings keinen Bezug zu einem tatsächlichen Softwareprodukt aufweist. So lauten zum Beispiel die Titel der *Requirements Work Items* lediglich „Test Requirement1“, „Test Requirement2“ usw. Diese Work Items wurden mit anderen exemplarischen Work Items gemäß den Regeln des Sliced V-Modells verlinkt. Die *Module Work Items* wurden nicht mit Änderungen an realen Quelltextdateien verlinkt, sondern lediglich mit Änderungen an Textdateien, die einen frei gewählten Text enthalten.

Um zu überprüfen, ob die Anforderungen 1 und 2 erfüllt werden, wurde *SofProSys* getestet. Dieser Test kann allerdings nicht als ein vollwertiger systematischer Test, wie er zum Beispiel in [SL03] beschrieben ist, angesehen werden. Da es sich bei *SofProSys* um einen Prototyp handelt, wird die nachfolgend dargestellte Testtiefe als ausreichend angesehen.

Überprüfung der Anforderung 1:
Es wurde stichprobenartig geprüft, ob die im ALM-System und im Versionsmanagementsystem gespeicherten Daten mit den von *SofProSys* erfassten und angezeigten Werten übereinstimmen. So wurde zum Beispiel mit Hilfe der *Polarion ALM*-Benutzerschnittstelle die Anzahl aller Work Items, die mit einem *Feature Work Item* verlinkt sind, manuell gezählt, um die Dokumentationsgröße einer Softwarefunktion Doc_{fs} zu ermitteln. Der auf diese Art und Weise manuell erfasste Wert wurde mit den Werten verglichen, die *SofProSys* für diese Softwarefunktion ermittelt hat. Des Weiteren wurden manuell die Aufwände für eine Softwareversion erfasst, indem die eingetragenen Stunden aller zu dieser Softwareversion gehörenden *Task Work Items* addiert wurden. Diese manuell ermittelten Werte wurden mit den von *SofProSys* erfassten Werten verglichen.

Um die erfassten Churngrößen für ein Work Item zu überprüfen, wurden die mit diesem Work Item verlinkten Revisionen wie folgt analysiert: Zunächst wurde mit *Polarion ALM*-Bordmitteln festgestellt, welche Revisionen mit dem Work Item verlinkt sind. Diese Revisionen wurden daraufhin mit der Subversion Client Software *TortoiseSVN* untersucht [TOR17]: Es wurde mit Bordmitteln von *TortoiseSVN* für jede geänderte Datei in einer Revision der *Unified Diff Patches* zur Vorgängerversion dieser Datei ermittelt. Abbildung 7.4 zeigt die Vorgehensweise in *TortoiseSVN*. Danach wurde die Textgröße in KB für je-

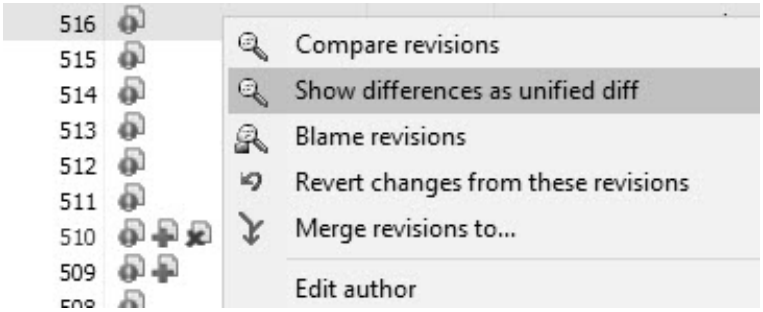


Abbildung 7.4: Anzeige eines *Unified Diff Patches* in *TortoiseSVN*

des einzelne *Unified Diff Patch* manuell ermittelt, um anschließend die einzelnen Werte zu summieren. Dieser manuell ermittelte Wert einer Churn-Kennzahl wurde mit dem von *SofProSys* automatisiert erfassten Wert verglichen.

Alle beschriebenen Tests waren erfolgreich. Dies wird als Bestätigung angesehen, dass die Anforderung 1 von *SofProSys* erfüllt wird. Wie erwähnt, erfolgte allerdings kein systematischer Test. Daher können Fehler in *SofProSys* nicht ausgeschlossen werden.

Überprüfung der Anforderung 2:
Es wurde überprüft, ob alle Softwarekennzahlen und SW-Produktionskennzahlen im Sliced V-Modell erfasst werden können. Dabei wurde berücksichtigt, dass einige der Daten im Sliced V-Modell Storage von *SofProSys* simuliert sind. Damit ist gemeint, dass zum Beispiel Daten für die zugesagten Termine einer Softwarefunktion willkürlich gesetzt wurden. Der Grund dafür liegt darin, dass es keinen realen Kunden für *SofProSys* gibt. Aus demselben Grund wurden auch extern entdeckte Fehler simuliert, das heißt, es wurden in das *External Defect Document* einige Fehler eingetragen und mit Revisionen verknüpft. Die Fehlerklassifizierung an *Defect Work Items* (vgl. Tabelle 5.4) erfolgte durch den Verfasser der Arbeit. Es handelte sich also nicht um die Einschätzung eines Kunden oder eines Produktverantwortlichen. Des Weiteren wurde der betriebsspezifische Stundensatz mit 70 € willkürlich festgelegt.

Mit der Messung des Sliced V-Modell Storage von *SofProSys* wurden die in den Tabellen 7.1, 7.2 und 7.3 aufgeführten Kennzahlen ermittelt. Die Softwarekennzahlen für die prozentuale Verteilung der Werte in den Attributen, die einen Fehler klassifizieren, sind in Anhang B aufgeführt.

Softwarekennzahl	**Version 1**	**Version 2**	**Version 3**
Ch_{f_v} $[KB]$	3.539	58	514
Ch_{di_v} $[KB]$	17	17	154
E_{dev_v} $[h]$	311	12	124
P_{ch_v} $[\frac{KB}{h}]$	11,14	6,21	5,39
D_v $[d]$	1.115	191	103
V_{ch_v} $[\frac{KB}{d}]$	3,19	0,39	6,49
Doc_{f_v} $[WI]$	152	41	98
E_{doc_v} $[h]$	61	12	31
P_{doc_v} $[\frac{WI}{h}]$	2,49	3,42	3,16
V_{doc_v} $[\frac{WI}{d}]$	0,14	0,21	0,95
DD_{ch_v} $[\frac{WI}{KB}]$	0,0014	0,0268	0,0150

Tabelle 7.1: Versionsbezogene *SofProSys*-Softwarekennzahlen

SW-Produktionskennzahl	**Version 1**	**Version 2**	**Version 3**
FPR_v [%]	99,51	77,36	76,88
SL_v [%]	100	95,18	91,23
AV_v [€]	18.270	1.190	15.260
P_v $[\frac{€}{h}]$	49,11	49,58	98,45

Tabelle 7.2: Versionsbezogene *SofProSys*-SW-Produktionskennzahlen

Diese Messung zeigte, dass *SofProSys* alle definierten SW-Produktionskennzahlen und Softwarekennzahlen automatisiert erfassen und verarbeiten kann.

Die Werte einiger Kennzahlen konnten nicht auf Plausibilität geprüft werden, da folgende Daten im Dummy Sliced-V-Modell Storage willkürlich gesetzt wurden:

- Die Fehlerbehebungsrate DFR_p und die Technische Rückläuferrate TRR_p, da es keine realen extern entdeckten Fehler gibt.
- Der Servicegrad SL_v, da es keine einem Kunden zugesagten Softwarefunktionen gibt.
- Die Wertschöpfung AV_v und die Produktivität P_v, da die Entwicklung nicht in einem produzierenden Betrieb stattfindet.

Kennzahl	**Wert**
DFR_p [%]	86,36
TRR_p [%]	0,02

Tabelle 7.3: Produktbezogene *SofProSys*-Kennzahlen

Alle weiteren Kennzahlen konnten mit kleineren Einschränkungen erfolgreich auf deren Plausibilität geprüft werden:

Die Entwicklung der ersten Version von *SofProSys* dauerte gut drei Jahre. In diese Zeit fielen die Programmierarbeiten, die zunächst darauf abzielten, die verschiedenen Schnittstellen der Collaboration Tools anzusprechen und die ersten Datenbankabfragen zu implementieren. Des Weiteren wurde die Speicherung der Kennzahlen implementier, zunächst in einer CSV-Datei, später in mehreren MS Excel-Dateien. Die erste Version von *SofProSys* beinhaltete also die für die Datenerfassung und -verarbeitung notwendigen technischen Voraussetzungen.

Während der Erarbeitung der ersten Version wurde das Sliced V-Modell kontinuierlich entwickelt. Durch die fortlaufenden Anpassungen wurde *SofProSys* immer wieder geändert. All dies erklärt den höheren Wert des *Version Feature Churns* in der Version 1 von *SofProSys* gegenüber den Werten der beiden folgenden Versionen.

Dies erklärt ebenfalls den höheren Wert der Dokumentationsgröße der Version 1 gegenüber den beiden folgenden Versionen. Der Unterschied in den Werten ist allerdings nicht so groß wie beim *Version Feature Churn*. Dies ist plausibel, da während der Entwicklung der ersten Version die Dokumentation weniger stark berücksichtigt wurde als in den anderen Versionen. Die Implementierung, und nicht die Dokumentation, stand im Fokus der ersten Version.

Der Softwareentwicklungsprozess von *SofProSys* enthielt keine systematischen qualitätssichernden Maßnahmen. Lediglich die oben aufgeführten Tests wurden durchgeführt. Die Version 1 wurde kaum getestet. Dies erklärt den hohen Wert der First Pass Rate: Ohne Test kann kein Fehler gefunden werden, und folglich gibt es keine Quelltextänderungen, die Fehlerbehebungen zugeordnet werden können. Während der Entwicklung der Versionen 2 und 3 wurde die praktische Anwendung beim Kooperationspartner berücksichtigt. Während der praktischen Anwendung (vgl. Abschnitt 7.4) wurden Fehler entdeckt und behoben. Daher sind die Werte der First Pass Rate in diesen Versionen geringer als in der Version 1, die Werte der *Version Internal Defect Churns* und der Churn-Fehlerdichte sind dagegen größer.

Während der Entwicklung der Version 1 wurden keine Aufwände geplant und gebucht. Die Planung und Buchung von Aufwänden wurde erst durch den späteren Entwurf der dazugehörigen Kennzahlen notwendig. Um dennoch die mit der Planung und Buchung zusammenhängenden Kennzahlen prüfen zu können, wurden Soll-Aufwände und Ist-Aufwände rückblickend für die Version 1 abgeschätzt. Daher kann die Plausibilität der Churn-Produktivität P_{ch_v} und der Dokumentationsproduktivität P_{doc_v} nicht vollständig bewertet werden. Die Werte für die Versionen 2 und 3 sind plausibel.

Es kann zusammengefasst werden, dass *SofProSys* die beiden Anforderungen an ein Informationsverarbeitungssystem erfüllt. *SofProSys* ist in der Lage, die Daten aus den Collaboration Tools korrekt zu erfassen und zu verarbeiten und alle bestimmten Kennzahlen zu berechnen. Zwar konnte die Plausibilität nur eingeschränkt geprüft werden: Wo es möglich war, ergab die Überprüfung jedoch plausible Werte, die den Realitäten der Softwareentwicklung von *SofProSys* entsprechen. Diese Zusammenfassung wird als dritte Bestätigung angesehen, dass das Sliced V-Modell die in den Kapiteln 3 und 4 formulierten Anforderungen erfüllt und folglich als eine Lösung für die zweite Detailfrage dieser Arbeit angesehen werden kann (vgl. letzter Absatz in Kapitel 6).

7.3 Bewertung der Gestaltungsgrundsätze

Nach der praxisnahen Anwendung und der Überprüfung der Anforderungen an *SofProSys* wird bewertet, ob *SofProSys* die in Abschnitt 2.1.1 aufgeführten empfohlenen Gestaltungsgrundsätze für den Aufbau von Informationsverarbeitungssystemen berücksichtigt und folglich als eine Lösung für die dritte Detailfrage angesehen werden kann. Es folgen die einzelnen Gestaltungsgrundsätze und die Bewertung:

Hohe Validität: Ein Kennzahlensystem muss korrekt messen, um eine hohe Validität der Ist-Werte sicherzustellen. *SofProSys* erfasst alle Kennzahlen prinzipiell automatisiert. Da es sich um einen Prototyp handelt, waren während der praxisnahen und praktischen Anwendung manuelle Nachbearbeitungen notwendig. Diese könnten fehlerhaft sein und folglich die Validität der Ist-Werte beeinträchtigen. Die manuellen Nachbereitungen würden entfallen, falls *SofProSys* weiterentwickelt und für den Produktivbetrieb vorbereitet wird. Die Validität müsste in jedem Fall durch einen systematischen Test sichergestellt werden. Bislang wurden nur stichprobenartige Tests durchgeführt.

Berücksichtigung der wesentlichen Kennzahlen: Die Softwarekennzahlen wurden zielorientiert mit der GQM-Methode bestimmt. Die Auswahl der SW-Produktionskennzahlen erfolgte durch die Bestimmung der äquivalenten HW-Produktionskennzahlen. Daher berücksichtigt der aktuelle Stand von *SofProSys* alle wesentlichen Kennzahlen. Die kennzahlenorientierte Prozessgestaltung ist jedoch eine kontinuierliche Aktivität in einem produzierenden Betrieb. Daher kann nicht ausgeschlossen werden, dass in Zukunft einige der momentan berücksichtigten Kennzahlen als unwesentlich eingestuft werden, andere bislang nicht berücksichtigte Kennzahlen dagegen in das Informationsverarbeitungssystem aufgenommen werden.

Einfluss der Entscheidungsträger: Dieser Gestaltungsgrundsatz besagt, dass die Werte der Kennzahlen unmittelbar durch die Entscheidungsträgern beeinflussbar sein soll-

ten. Dies gilt für *SofProSys*: Sowohl das Management als auch die Softwareteams wirken auf den Softwareentwicklungsprozess ein. Sie können damit die Ist-Werte der SW-Produktionskennzahlen und der Softwarekennzahlen durch Umsetzung von Maßnahmen beeinflussen.

Ausgewogene Ausgestaltung hinsichtlich Menge und Zeithorizont: Dieser Gestaltungsgrundsatz kann mit dem gegenwärtigen Stand von *SofProSys* nicht bewertet werden, da die Kennzahlen nicht grafisch aufbereitet werden. In der grafischen Aufbereitung ist zu berücksichtigen, dass die Informationsmenge von den Entscheidungsträgern aufgenommen und in einen angemessenen zeitlichen Kontext gesetzt werden kann.

Festlegung von Soll-Werten, um Ist-Werte interpretieren zu können: Für einige der Kennzahlen ist die Festlegung von Soll-Werten möglich, zum Beispiel für die First Pass Rate, die Technische Rückläuferrate oder die Churn-Fehlerdichte. Jedoch fehlen bislang Erfahrungen für realistische Soll-Werte. Für einige der Kennzahlen, zum Beispiel für die Wertschöpfung, können keine Soll-Werte festgelegt werden, da sie abhängig von der entwickelten Softwareversion sind. Solche Kennzahlen dienen zwar den Informationsbedürfnissen der Entscheidungsträger, können jedoch nicht für eine Soll-/Ist-Steuerung eingesetzt werden.

Ausgestaltung unter Kosten-Nutzen-Bedingungen: Dieser Gestaltungsgrundsatz besagt, dass der Nutzen von *SofProSys* die Kosten für dessen Entwicklung und dessen Betrieb rechtfertigt. Der angestrebte Nutzen ist die Motivation für diese Arbeit. Allerdings ist *SofProSys* bislang nicht im operativen Betrieb, so dass keine endgültige Aussage zum Nutzen getätigt werden kann. Da *SofProSys* konzeptionell alle Daten automatisiert erfasst und verarbeitet, sind die Kosten für die Nutzung gering. Da jedoch weitere IT-basierte Funktionen und qualitätssichernde Maßnahmen in der Ausgestaltung von *SofProSys* zu berücksichtigen sind, lassen sich die Kosten für die vollständige Entwicklung nicht quantifizieren. Lediglich die bisherigen Aufwände (551 h gemäß Tabelle 7.1) können berücksichtigt werden. Eine abschließende Bewertung dieses Gestaltungsgrundsatzes ist daher erst nach Fertigstellung des produktiven Informationsverarbeitungssystems möglich.

Eindeutige Darstellung der Kennzahlen: Dieser Gestaltungsgrundsatz besagt, dass die Darstellung der Kennzahlen eine grundsätzlich falsche Interpretation ausschließen soll. Da alle Softwarekennzahlen und SW-Produktionskennzahlen korrekt erfasst und in einem Excel-Dokument dargestellt werden, sollte eine grundsätzlich falsche Interpretation ausgeschlossen sein. Das Excel-Dokument ist allerdings sehr einfach gehalten, was eine intuitive Bedienung erschweren könnte.

Es lässt sich schlussfolgern, dass *SofProSys* die empfohlenen Gestaltungsgrundsätze für den Aufbau von Informationsverarbeitungssystemen weitestgehend erfüllt. Einige der

Grundsätze können jedoch erst abschließend bewertet werden, wenn *SofProSys* zu einem operativen Informationsverarbeitungssystem weiterentwickelt und über einen längeren Zeitraum in der betrieblichen Praxis verwendet wird.

Da *SofProSys* sowohl die an den Prototypen gestellten Anforderungen erfüllt und die Gestaltungsgrundsätze für den Aufbau von Informationsverarbeitungssystemen prinzipiell erfüllt, kann *SofProSys* als eine Lösung für die dritte Detailfrage angesehen werden.

Nachdem die Lösungen für die drei Detailfragen dieser Arbeit entwickelt wurden, widmet sich der nachfolgende Abschnitt der Evaluierung der Forschungsfrage.

7.4 Praktische Anwendung

Die praktische Anwendung fand beim Kooperationspartner statt. Es wurden die Sliced V-Modell Storages von zwei Softwareprodukten gemessen. Der Kooperationspartner schränkte die Veröffentlichung der Ergebnisse ein und erlaubt nur die Darstellung und Erläuterung der Softwarekennzahlen und SW-Produktionskennzahlen von einem der beiden Softwareprodukte. Außerdem muss auf die Nennung der Softwareprodukte und der betriebsspezifischen Stundensätze verzichtet werden.

Das untersuchte Softwareprodukt ist eine Windows-Desktop-Software für die Bedienung von intelligenten Produkten, beispielsweise für deren Parametrierung oder Diagnose. Um Rückwirkungen auf das operative ALM-System und das Versionsmanagementsystem zu vermeiden, wurden das Sliced V-Modell Storage des Softwareproduktes und die dazugehörenden Repositories in dem Versionsmanagementsystem auf eine Testumgebung kopiert, in der vorab *Polarion ALM* und *Subversion* installiert wurden.

Beim Kooperationspartner wurden zum Zeitpunkt der Messung nicht alle Eigenschaften des Sliced V-Modells in die betriebliche Praxis überführt. Daher kommt es zu folgenden Abweichungen von dem im Abschnitt 5.2.2 erläuterten Datenmodell bzw. in der Nutzung einiger im Datenmodell vorgesehenen Informationseinheiten:

- Die Work Item-Attribute *internal* und *external* an *Defect Work Items* sind nicht vorhanden, das Attribut *dueDate* an *Feature Work Items* wird nicht genutzt. Daher können die darauf aufbauenden Softwarekennzahlen bzw. SW-Produktionskennzahlen nicht ermittelt werden. Dies sind die prozentuale Verteilung der Werte der Attribute *internal* und *external* und der Servicegrad.
- Des Weiteren werden zwar End-Baselines, jedoch keine Start-Baselines gesetzt. Während der praktischen Anwendung wurden die Starttermine der einzelnen Softwareversionen aus der beim Kooperationspartner eingesetzten Projektdatenbank

manuell ermittelt (vgl. Abschnitt 1.3). Wie in Abschnitt 2.1.2.3 erläutert, gibt es keine allgemeingültige Definition für den Starttermin und den Endtermin der Entwicklung einer Softwareversion. Jeder produzierende Betrieb muss somit diese Termine betriebsspezifisch bestimmen. Beim Kooperationspartner sind dies der Tag der Freigabe der Anforderungen an eine Softwareversion bzw. der getesteten Softwareversion.

Durch die Messung des Sliced V-Modell Storage des Softwareproduktes wurden die in den Tabellen 7.4, 7.5 und 7.6 aufgeführten Kennzahlen ermittelt. Da die Nennung des betriebsspezifischen Stundensatzes nicht erlaubt ist, wird den SW-Produktionskennzahlen ein fiktiver Stundensatz von 10 $\frac{€}{h}$ zugrunde gelegt. Die prozentuale Verteilung der Werte des Attributs *severity*, die den Schweregrad eines Fehlers klassifizieren, werden nicht aufgeführt: Die Messungen zeigten, dass die meisten entdeckten Fehler nicht klassifiziert wurden, das Attribut *severity* war mit dem voreingestellten Standardwert *Neutral* belegt.

Diese Messungen zeigten, dass *SofProSys* in der Lage ist, Softwarekennzahlen und SW-Produktionskennzahlen in einem produzierenden Betrieb zu erfassen. Um die Plausibilität der Werte der einzelnen Kennzahlen zu überprüfen, wurden sie in mehreren Meetings einigen Stakeholdern aus der Softwareentwicklung vorgestellt.

Zunächst konnte die in den Kennzahlen erkennbare unterschiedliche Natur der drei Softwareversionen bestätigt werden: Version 1 beinhaltete die Umsetzung zahlreicher neuer Softwarefunktionen, Version 2 war im Wesentlichen geprägt von Fehlerkorrekturen an der Version 1, und Version 3 beinhaltete wiederum neue Softwarefunktionen, allerdings nicht in dem Umfang der Version 1.

Softwarekennzahl	**Version 1**	**Version 2**	**Version 3**
Ch_{f_v} $[KB]$	1.676	178	557
Ch_{di_v} $[KB]$	445	35	297
E_{dev_v} $[h]$	6.450	972	1.216
P_{ch_v} $[\frac{KB}{h}]$	0,41	0,28	1,12
D_v $[d]$	550	192	218
V_{ch_v} $[\frac{KB}{d}]$	3,86	1,11	3,92
Doc_{f_v} $[WI]$	941	40	370
E_{doc_v} $[h]$	1.215	200	454
P_{doc_v} $[\frac{WI}{h}]$	0,15	0,04	0,30
V_{doc_v} $[\frac{WI}{d}]$	1,71	0,21	1,70
DD_{ch_v} $[\frac{WI}{KB}]$	0,0476	0,0141	0,0293

Tabelle 7.4: Versionsbezogene Softwarekennzahlen des Softwareproduktes

SW-Produktionskennzahl	Version 1	Version 2	Version 3
FPR_v [%]	79,02	83,59	65,20
SL_v [%]	-	-	-
AV_v [€]	59.750	9.980	19.920
P_v [$\frac{€}{h}$]	9,26	10,27	16,38

Tabelle 7.5: Versionsbezogene SW-Produktionskennzahlen des Softwareproduktes

Kennzahl	Wert
DFR_p [%]	70,47
TRR_p [%]	10,28

Tabelle 7.6: Produktbezogene Kennzahlen des Softwareproduktes

Daher ist das Verhältnis der Softwarekennzahlen, die die Software- und Dokumentationsquantität anzeigen, plausibel: In Version 1 haben die Softwarekennzahlen Ch_{f_v} und Doc_{f_v} die größten Werte, in Version 2 sind sie am kleinsten und in Version 3 weisen sie Werte auf, die zwischen denen der Version 1 und Version 2 liegen.

Da die Softwareversionen unterschiedlicher Natur sind, sind die verschiedenen Werte der Softwarekennzahlen der drei Softwareversionen für die Aufwände E_{dev_v} und E_{doc_v} und für die Entwicklungsdauer D_v ebenfalls plausibel. Die Werte in der Version 1 sind am größten, die Werte der Version 2 am kleinsten und die Werte für die Version 3 liegen zwischen den Werten der Version 1 und der Version 2.

Etwas überraschend war die Tatsache, dass der Schweregrad der Fehler kaum bewertet wurde. Die Stakeholder sagten, zwar sei eine entsprechende Vorgabe in den vorhandenen Prozessbeschreibungen enthalten, aber offensichtlich werden diese Vorgaben nicht durchgängig eingehalten. Sie würden dies in den nächsten Teammeetings thematisieren.

Des Weiteren ist das durch die First Pass Rate ausgedrückte Verhältnis der Churn-Kennzahlen Ch_{f_v} und Ch_{di_v} plausibel. Ohne dafür jemals konkrete Kennzahlen genutzt zu haben, wurde von den befragten Stakeholdern abgeschätzt, dass ca. 20 % der Quelltextänderungen für eine Softwareversion wegen intern entdeckter Fehler erfolgen. Dies wurde durch die First Pass Rate für Version 1 und Version 2 bestätigt. Der geringere Wert in der Version 3 war allerdings auch plausibel, da in dieser Version die Entwicklungsumgebung, mit der das Softwareprodukt entwickelt wurde, auf eine höhere Softwareversion umgestellt wurde. Durch diese Umstellung kam es zu zahlreichen Quelltextänderungen, die intern entdeckten Fehlern zugeordnet wurden.

Es wurde nicht bezweifelt, dass die Werte in den Softwarekennzahlen für die Aufwände E_{dev_v} und E_{doc_v} von *SofProSys* korrekt gemessen werden. Dennoch konnten die Unterschiede in den Werten der Softwarekennzahlen nicht erklärt werden, in die E_{dev_v} und E_{doc_v}

einfließen, und zwar in die Churn-Produktivität P_{ch_v} und in die Dokumentationsproduktivität P_{doc_v}. Eine mögliche Ursache könne die nicht sachgemäße Buchung von Stunden durch die Mitarbeiter der Softwareteams sein. Es könne durchaus sein, dass Stunden, die tatsächlich mit Dokumentationsaktivitäten verbracht wurden, den Entwicklungsaktivitäten zugeordnet wurden und umgekehrt. Eine solche Fehlbuchung lasse sich in der täglichen Praxis nicht ganz vermeiden.

Da die Churn-Produktivität durch die Messung erstmalig für die befragten Stakeholder sichtbar wurde, konnte die deren Plausibilität nicht bestätigt bzw. bezweifelt werden. Die Werte für die Dokumentationsproduktivität, wonach 0,15, 0,04 bzw. 0,30 Work Items pro Stunde erstellt wurden, erschienen zu gering. Das Anlegen eines Work Items dauert wenige Sekunden. Zwar werden danach die Work Items immer wieder geändert, jedoch wurden höhere Werte erwartet. Ein Grund dafür könnte neben der inkorrekten Zuordnung der gebuchten Stunden die Erstellung des grafischen Softwaredesigns außerhalb des ALM-Systems sein. Die Daten des Softwaredesigns sind nicht in die Datenerfassung von *SofProSys* eingebunden.

Es wurde positiv anerkannt, dass die zugrundeliegende IT-gestützte Datenerfassung eine notwendige Voraussetzung für die Anwendung von *SofProSys* in der betrieblichen Praxis sei. Um alle Softwarekennzahlen vollständig zu erfassen, müssten jedoch alle benötigten Daten in den Softwareprojekten eingetragen werden. Das würde bedeuten, dass in der betrieblichen Praxis das Sliced V-Modell vollständig genutzt würde. Zwar könne *SofProSys* verwendet werden, um auf fehlende Daten hinzuweisen, jedoch würden in der täglichen Arbeit immer wieder Daten fehlen. Dies sei nicht ganz zu vermeiden. Es müsse ein längerfristiger Prozess gestartet werden, um in Schulungen oder mit anderen geeigneten Maßnahmen auf die Vollständigkeit des Sliced V-Modells hinzuwirken.

Die Anzeige von Detaildaten für einzelne Softwarefunktionen wurde begrüßt. Wie in Abschnitt 6.2 erläutert, werden die Menge des Churns und die Menge der Dokumentation jeweils pro einzelner Softwarefunktion erfasst und erst danach für die Softwareversion akkumuliert. Die Werte einzelner Softwarefunktionen werden von *SofProSys* angezeigt. Tabelle 7.7 zeigt exemplarisch die Softwarekennzahlen zweier gemessener Softwarefunktionen. Die Softwarekennzahlen *Rev* (Anzahl an Revisionen) und *Files* (Anzahl an geänderten Quelltextdateien) sind zusätzlich Softwarekennzahlen, die *SofProSys* erfassen kann. Diese Softwarekennzahlen wurden nicht von den operativen Zielen hergeleitet (vgl. Abschnitt 4.2.2), sondern entstanden während der Entwicklung von *SofProSys*.

Es ist unmittelbar erkennbar, welche der beiden Softwarefunktionen eine „große“ Softwarefunktion ist. „Groß“ bedeutet, dass die Dokumentationsgröße Doc_{fs} und die Churn-Menge Ch_{fs} vergleichsweise hohe Werte annehmen. Die Kenntnis der „Größe“ einer Softwarefunktion könne, so das Meinungsbild der befragten Stakeholder, in der Testplanung und

Softwarefunktion	$\boldsymbol{Doc_{fs}}$	$\boldsymbol{Ch_{fs}}$	***Rev***	***Files***
A	21 WI	356 KB	25	121
B	7 WI	50 KB	5	8

Tabelle 7.7: Softwarekennzahlen einzelner Softwarefunktionen

in der Testdurchführung berücksichtigt werden: Eine „große“ Softwarefunkton müsse eingehender getestet werden.

Die Plausibilität der Werte der aufgeführten Softwarekennzahlen und SW-Produktionskennzahlen konnte bestätigt werden. Jedoch wurde angemerkt, dass die grafische Darstellung der Kennzahlen für einen produktiven Einsatz von *SofProSys* nicht ausreichen würde. Dafür müssten die Softwarekennzahlen grafisch aufbereitet werden, zum Beispiel in Form von Diagrammen.

Nach den Gesprächen mit den Stakeholdern in der Softwareentwicklung zu den Softwarekennzahlen wurden die SW-Produktionskennzahlen dem Geschäftsbereichsleiter vorgestellt. Die SW-Produktionskennzahlen wurden vorab grafisch aufbereitet. Dies erfolgte manuell mit MS Excel, da *SofProSys* dazu aktuell nicht in der Lage ist. Abbildung 7.5 zeigt die First Pass Rate der drei gemessenen Versionen als ein Beispiel für die grafisch aufbereiteten SW-Produktionskennzahlen.

Der Geschäftsbereichsleiter zeigte sich vom Format der Präsentation der Ergebnisse einer Softwareentwicklung beeindruckt: „Diese Form der Ergebnisdarstellung und die Verwendung mir bekannter Produktionskennzahlen erleichtern deutlich die Bewertung der Softwareergebnisse.“ Ihm war klar, dass die Ist-Werte der einzelnen SW-Produktions-

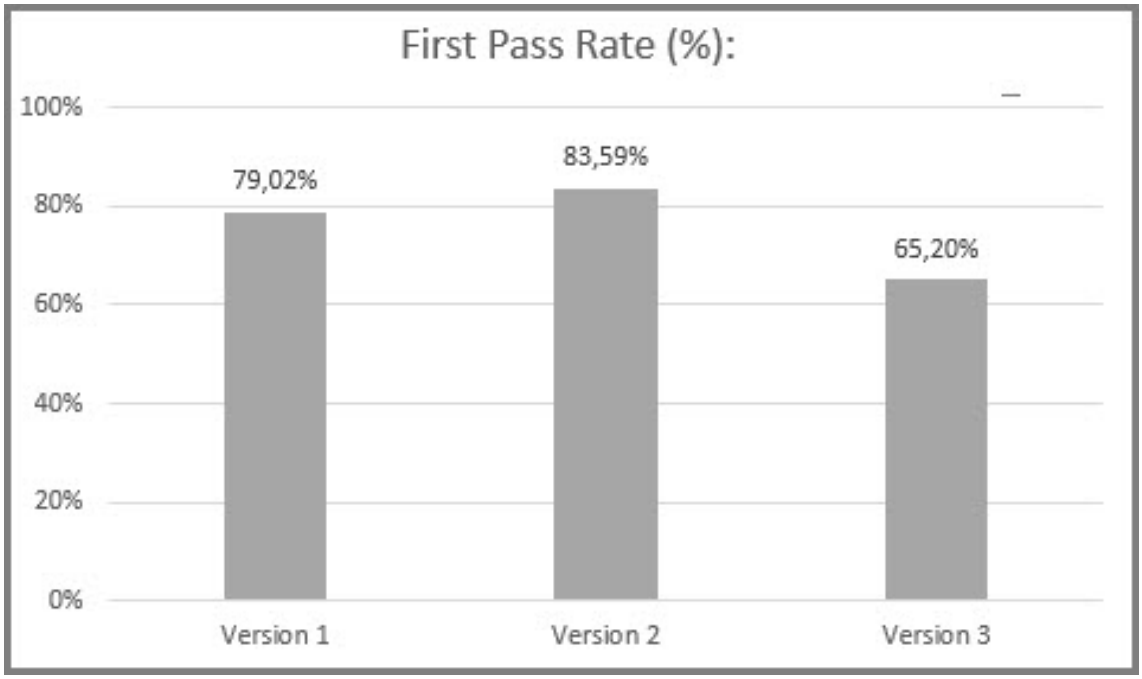


Abbildung 7.5: Grafische Darstellung der First Pass Rate in MS Excel

kennzahlen von den typischen Ist-Werten der HW-Produktionskennzahlen abweichen werden. Dies sei jedoch nicht kritisch, da bereits heute Ist-Werte von HW-Produktionskennzahlen in verschiedenen Produktionslinien unterschiedlich sein können. Wichtig sei es, bei der Einführung eines produktiven Informationsverarbeitungssystems sinnvolle Soll-Werte für die einzelnen SW-Produktionskennzahlen festzulegen, die mit Ist-Werten verglichen werden, um somit dieselben Steuerungs- und Überwachungsmechanismen zu etablieren wie in der Produktion. Auf Grundlage der ersten Messergebnisse formulierte er als erste Soll-Werte für die First Pass Rate 80 % und für die Technische Rückläuferrate 5 %.

Weiterhin äußerte er die Erwartungshaltung, dass die Softwareteams diejenigen Daten eintragen, die eine Ermittlung des Servicegrades ermöglichen. Wie schon in Abschnitt 3.2.2.1 erläutert, sei ihm der Servicegrad wichtig, weil er die Lieferqualität der vielen kleineren Softwarefunktionen nicht bewerten kann, da ihm diese kleineren Softwarefunktionen nicht bekannt seien.

Obwohl ihm die Personalkosten für die Softwareentwicklungsteams bekannt sind, äußerte er sich positiv über den Einblick in die Zuordnung der Kosten zu den Softwareentwicklungsprojekten, die in der *Wertschöpfung* angezeigt werden. Es müsse mittelfristig bewertet werden, wie die Wertschöpfung in der Preisbildung der Softwareprodukte oder der darauf aufbauenden Servicedienstleistungen berücksichtigt werden kann. Des Weiteren ermögliche die Anzeige der Produktivität eine kontinuierliche Beobachtung, ob die Personalkosten wertschöpfend eingesetzt werden.

Er fühlte sich darin bestätigt, dass die fünf mit ihm ausgewählten HW-Produktionskennzahlen, angewandt als SW-Produktionskennzahlen, eine wertvolle Unterstützung in der Beobachtung des Softwareentwicklungsprozesses sein werden. Die SW-Produktionskennzahlen würden ihm eine im Grundsatz identische Interpretation wie die äquivalenten HW-Produktionskennzahlen ermöglichen. Die Erreichung strategischer Ziele würde in diesen SW-Produktionskennzahlen angezeigt. Des Weiteren begrüßte er, dass durch die in dieser Arbeit entworfenen Lösungen die Softwareentwicklungsteams in die Lage versetzt werden, die Erreichung der operativen Ziele zu überprüfen. Er teilte die Bewertung der Stakeholder aus der Softwareentwicklung, dass die *SofProSys* zugrunde liegende IT-gestützte Datenerfassung eine notwendige Voraussetzung ist, um das Informationsverarbeitungssystem in die betriebliche Praxis zu überführen.

Alle Gesprächspartner des Kooperationspartners sehen in den gezeigten Ergebnissen eine Lösung für die Fragestellung dieser Arbeit: *Wie kann der Softwareentwicklungsprozess in produzierenden Betrieben kennzahlenorientiert gestaltet werden?* Zwar wurde sie in nur einem produzierenden Betrieb evaluiert. Allerdings wird es als realistisch angesehen, dass die bislang entwickelten Lösungen auch in andere produzierende Betriebe übertragen werden können. Dies wird im folgenden Kapitel begründet.

Kapitel 8

Verallgemeinerung der Ergebnisse

Die in dieser Arbeit entwickelte kennzahlenorientierte Gestaltung des Softwareentwicklungsprozesses erfolgte in Zusammenarbeit mit dem Kooperationspartner und umfasst folgende Ergebnisse: Mit der RGQM-Methode wurden betriebsspezifische SW-Produktionskennzahlen bestimmt. Die SW-Produktionskennzahlen dienen der Überprüfung strategischer Ziele des Kooperationspartners. Softwarekennzahlen wurden mit der GQM-Methode aus operativen Zielen hergeleitet. Der gestaltete Softwareentwicklungsprozess ist das Sliced V-Modell. Der Prototyp des Informationsverarbeitungssystems setzt die Verwendung der Collaboration Tools *Polarion ALM* und *Subversion* voraus. Folglich beziehen sich die bisherigen Lösungen auf die Fragestellungen dieser Arbeit auf die betriebsspezifische lokale Umgebung des Kooperationspartners. Wie in Abschnitt 1.5 dargelegt, erfordert das methodische Vorgehen nach dem Design Research-Paradigma eine Theoriebildung auf Basis der entwickelten lokalen Forschungsergebnisse. Diesem Themengebiet widmet sich dieses Kapitel, in dem dargestellt wird, wie die lokalen Forschungsergebnisse verallgemeinert und auf andere produzierende Betriebe übertragen werden können.

8.1 Bestimmung von SW-Produktionskennzahlen

Um die SW-Produktionskennzahlen zu bestimmen, die die Semantik der äquivalenten HW-Produktionskennzahlen beibehalten, wurde die RGQM-Methode entworfen und beim Kooperationspartner evaluiert. Ausgangspunkt für den Entwurf war die Fragestellung, ob Produktionskennzahlen in der Softwaredomäne angewendet werden können. Diese Fragestellung weckte ein hohes Interesse beim Management des Kooperationspartners, da es ein sehr gutes Verständnis von Produktionskennzahlen hat und diese in der kennzahlenorientierten Gestaltung des Produktionsprozesses bereits verwendet. Die Ergebnisse der im Abschnitt 1.4 erläuterten Umfrage der IHK Ostwestfalen zu Bielefeld werden als ein

Indikator angesehen, dass weitere produzierende Betriebe Interesse haben, Produktionskennzahlen in der Softwaredomäne zu nutzen. Die Rückläuferzahl der Fragebögen lässt jedoch keinen Rückschluss darüber zu, wie ausgeprägt dieses Interesse ist. Daher kann diese Verallgemeinerung lediglich eingeschränkt erfolgen.

Jeder produzierende Betrieb, der Produktionskennzahlen in der Softwaredomäne anwenden möchte, kann auf die RGQM-Methode zurückgreifen. Die RGQM-Methode ist nicht auf die Anwendung im betriebsspezifischen Umfeld des Kooperationspartners beschränkt. An dem folgenden Beispiel der Produktionskennzahl *Ausnutzungsgrad* soll dies theoretisch erläutert werden:

Der *Ausnutzungsgrad* ist eine Produktionskennzahl, die die Maschinenauslastung anzeigt. Sie ist eine Verhältnisgröße, die in % angegeben wird. Deren Wertebereich liegt zwischen 0 % und 100 % und der Idealwert ist 100 %. Es können Soll-Werte für den *Ausnutzungsgrad* festgelegt werden. Die Frage, das strategische Ziel und die Interpretation sind in Tabelle 8.1 aufgeführt. Die Frage, die mit dem *Ausnutzungsgrad* beantwortet wird, ist [Pre08] entnommen. Das strategische Ziel und die Interpretation dieser Produktionskennzahl wurden theoretisch festgelegt, das heißt es fand keine Befragung in einem produzierenden Betrieb statt.

Für diese theoretische Betrachtung wird davon ausgegangen, dass das strategische Ziel „Effiziente Nutzung aller vorhandenen Betriebsmittel“ auch in der Softwaredomäne gültig ist. Dann wäre es möglich, die in Tabelle 8.2 aufgeführte Frage für die Softwaredomäne zu formulieren und, wie in der Tabelle aufgeführt, zu interpretieren. Die tatsächlichen Berechnungsgrundlagen können für dieses theoretische Beispiel nicht ermittelt werden, da dies konkret von der Art und Weise der Erfassung der genutzten Lizenzen

Ausnutzungsgrad

Frage	Zu wie viel Prozent sind die vorhandenen Maschinen tatsächlich ausgelastet? [Pre08]
Ziel	Effiziente Nutzung aller vorhandenen Betriebsmittel
Interpretation	Der Ausnutzungsgrad ist das Verhältnis von Ist-Maschinenlaufstunden und möglichen Maschinenlaufstunden [Pre08]. Ein hoher Wert des Ausnutzungsgrades zeigt an, dass die Maschinen kontinuierlich im Einsatz sind. Stillstehende Maschinen tragen nicht zur Wertschöpfung bei. Sie verursachen dennoch Kosten, zum Beispiel für die Wartung oder durch die Flächennutzung. Das Management erwartet möglichst hohe Werte für den Ausnutzungsgrad. Können diese Werte nicht erreicht werden, müssen Maschinen aussortiert oder verkauft werden.

Tabelle 8.1: Frage, Ziel, Interpretation des *Ausnutzungsgrades* in der Produktion

Ausnutzungsgrad	
Ziel	Effiziente Nutzung aller vorhandenen Betriebsmittel
Frage	Zu wie viel Prozent sind die vorhandenen Lizenzen von IT-Systemen, die für die Softwareentwicklung benötigt werden, tatsächlich ausgelastet?

Tabelle 8.2: Frage und Ziel des *Ausnutzungsgrades* in der Softwareentwicklung

eines IT-Systems abhängig ist. Gleichung 8.1 zeigt daher lediglich eine grundsätzlich mögliche Berechnungsgrundlage.

$$LF = \frac{L_{av}}{L_{max}} \tag{8.1}$$

mit:

LF	Ausnutzungsgrad
L_{av}	Durchschnittliche Anzahl genutzter Lizenzen
L_{max}	Maximal verfügbare Lizenzen

Die Maßeinheit des Ausnutzungsgrades in der Softwaredomäne ist %. Der Wertbereich liegt zwischen 0 % und 100 %. Der Idealwert beträgt 100 %. Der Ausnutzungsgrad in der Softwareentwicklung wird wie folgt interpretiert:
Ein hoher Wert des Ausnutzungsgrades zeigt an, dass die beschafften Lizenzen tatsächlich im Einsatz sind. Beschaffte, aber nicht genutzte Lizenzen haben unnötige Kosten verursacht bzw. bewirken laufende Kosten, beispielsweise durch Wartungsgebühren. Das Management verlangt, dass die beschafften Lizenzen tatsächlich genutzt werden. Falls dies nicht möglich ist, sind die Wartungsverträge mit den Lieferanten anzupassen. Um Änderungen im Ausnutzungsgrad festzustellen, sollte dieser in regelmäßigen zeitlichen Abständen, zum Beispiel einmal pro Quartal, erfasst werden.

Die Bewertung der semantischen Äquivalenz der beiden Ausprägungen des *Ausnutzungsgrades* zeigt, dass der Name, der Wertebereich sowie der Idealwert identisch sind und dass in beiden Domänen Soll-Werte für den Ausnutzungsgrad festgelegt werden können. Das zum Ausnutzungsgrad gehörende strategische Ziel in der Produktion und in der Softwareentwicklung ist identisch. Der Satzbau der jeweiligen Fragen ist ebenfalls identisch. In beiden Domänen kann diese Kennzahl im Grundsatz in gleicher Weise interpretiert werden.

Dieses theoretische Beispiel zeigt folglich, dass die RGQM-Methode verallgemeinert und für andere als beim Kooperationspartner eingesetzte HW-Produktionskennzahlen angewendet werden kann.

8.2 Gestaltung des Softwareentwicklungsprozesses

Der in dieser Arbeit gestaltete Softwareentwicklungsprozess ist das Sliced V-Modell, in dem zum einen die durch die RGQM-Methode bestimmten SW-Produktionskennzahlen und zum anderen Softwarekennzahlen erfasst werden können.

Ausgangspunkt für die Bestimmung von Softwarekennzahlen sind die operativen Ziele der Softwareteams, die sich aus den strategischen Zielen des Managements ableiten. Es ist naheliegend, dass die beim Kooperationspartner angewandte pragmatische Vorgehensweise bei der Ableitung der operativen Ziele auf andere produzierende Betriebe übertragen werden kann.

Um Softwarekennzahlen aus den operativen Zielen abzuleiten, wurde die GQM-Methode angewendet. Die GQM-Methode ist seit langem verfügbar und ihre erfolgreiche Anwendung ist in zahlreichen Veröffentlichungen beschrieben. Es ist folglich naheliegend, dass die GQM-Methode von anderen produzierenden Betrieben genutzt werden kann.

Dem Sliced V-Modell liegt das Vorgehensmodell des Kooperationspartners zugrunde, das V-Modell der DIN EN 61508-3. In der Gestaltung des Sliced V-Modells wurde die Erfassung der beim Kooperationspartner bestimmten SW-Produktionskennzahlen und Softwarekennzahlen berücksichtigt. Sollte ein produzierender Betrieb ein anderes Vorgehensmodell als das V-Modell einsetzen, kann keine Aussage zur konkreten Vorgehensweise in dessen Gestaltung getroffen werden. Das in Abschnitt 5.1 erläuterte Vorgehen sollte jedoch für jedes beliebige Vorgehensmodell anwendbar sein. Ziel dieses Vorgehens ist es, ein Datenmodell zu erstellen und in UML zu beschreiben.

Jeder produzierende Betrieb kann das Sliced V-Modell nutzen und es in sein betriebliches Umfeld übertragen. In dieser Arbeit wurde ein Datenmodell entwickelt, das zwar die Verwendung eines ALM-Systems und eines Versionsmanagementsystems voraussetzt, jedoch nicht die beiden Collaboration Tools *Polarion ALM* und *Subversion*. Ein produzierender Betrieb kann entsprechend seinen Anforderungen aus einer Reihe am Markt verfügbarer alternativer Collaboration Tools auswählen und darin das Sliced V-Modell realisieren.

Wird das Sliced V-Modell nicht berücksichtigt, können dennoch einige der Anforderungen an das Sliced V-Modell durch ein anderes Datenmodell erfüllt werden. Dies soll exemplarisch an den Anforderungen *A1 Zuordnung Quelltextänderungen* und *A4 Erweiterte Zuordnung Quelltextänderungen* (vgl. Abschnitt 3.2.2.4 und 4.2.2.1) gezeigt werden:

Es ist eine Unterscheidung in *feature churn* und *defect churn* möglich, sofern Quelltextänderungen eindeutig der Implementierung einer Softwarefunktion und der Behebung eines Fehlers zugeordnet werden können. Es genügt ein einfaches Work Item-basiertes Datenmodell, in dem Work Items mit den Revisionen eines Versionsmanagementsystems

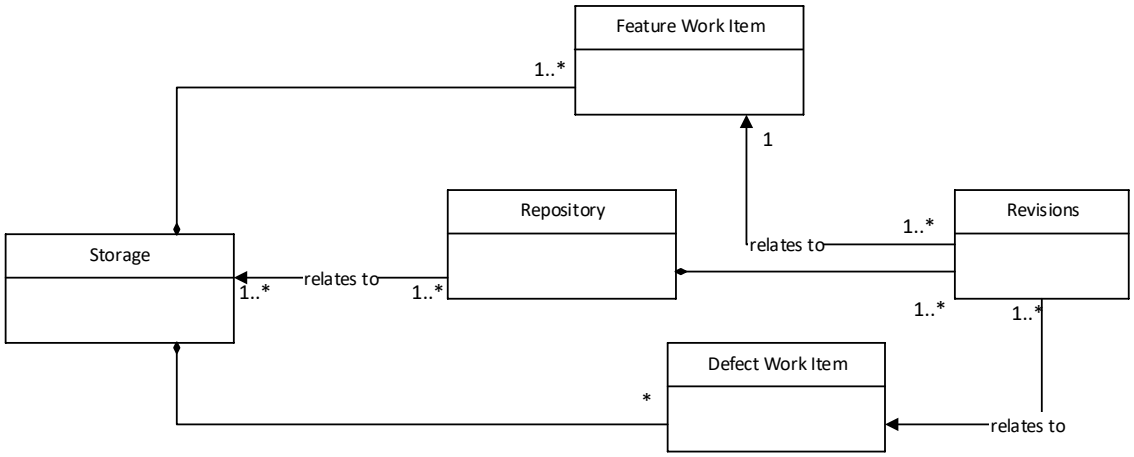


Abbildung 8.1: Vereinfachtes Work Item-basiertes Datenmodell

verknüpft werden können. Abbildung 8.1 zeigt eine mögliche Variante eines vereinfachten Work Item-basierten Datenmodells.

Sofern die dargestellten *Feature Work Items* und *Defect Work Items* die einzigen Dokumentationselemente sind, die während der Softwareentwicklung entstehen, könnte in diesem Fall ebenfalls die Anforderung *A7 Softwaredokumentation* (vgl. Abschnitt 4.2.2.2) erfüllt werden, falls diese in dem jeweiligen produzierenden Betrieb formuliert würde.

8.3 Aufbau eines Informationsverarbeitungssystems

Der in dieser Arbeit entwickelte Prototyp eines Informationsverarbeitungssystems (*SofProSys*) erfasst und verarbeitet Daten aus den Collaboration Tools *Polarion ALM* und *Subversion*, den beim Kooperationspartner eingesetzten Collaboration Tools. Sofern ein produzierender Betrieb genau diese Collaboration Tools verwendet, kann *SofProSys* als Ausgangspunkt für ein betriebsspezifisches Informationsverarbeitungssystem dienen. Werden andere Collaboration Tools eingesetzt, muss ein Informationsverarbeitungssystem neu entwickelt werden.

In der Ausgestaltung des Informationsverarbeitungssystems sind die Bedürfnisse des produzierenden Betriebes zu berücksichtigen. Dies betrifft die Art der Applikation (Desktop, Webanwendung etc.), die Art der grafischen Aufbereitung der Kennzahlen, ein Rechtemanagementsystem usw. Als Fazit wird es als naheliegend erachtet, dass ein produzierender Betrieb in der Lage ist, ein mit *SofProSys* vergleichbares Informationsverarbeitungssystem zu entwickeln und zu betreiben. Dabei sollten die Gestaltungsgrundsätze für Informationsverarbeitungssysteme berücksichtigt werden (vgl. Abschnitt 2.1.1).

Es liegt ebenfalls nahe, dass der in dieser Arbeit aufgeführte Prozess der Evaluierung von jedem produzierenden Betrieb nachvollzogen und durchgeführt werden kann. Das Informationsverarbeitungssystem ist anhand der Daten realer Softwareprojekte zu testen. Die erfassten und verarbeiteten Daten sind von Personen, die diese Softwareprojekte kennen, zu prüfen. Anschließend wird das Management in die Evaluierung einbezogen.

Die Möglichkeit der Verallgemeinerung der lokalen Forschungsergebnisse dieser Arbeit wurde in diesem Kapitel theoretisch dargelegt. Es erfolgte allerdings keine Evaluierung der Generalisierbarkeit, zum Beispiel durch Anwendung der Forschungsergebnisse in einem weiteren produzierenden Betrieb. Daher muss kritisch festgestellt werden, dass die Forschungsfrage dieser Arbeit lediglich für den lokalen Kontext eines produzierenden Betriebes beantwortet werden kann. Eine vollständige, tiefgehende und vor allem auf eine breite Datenbasis aufgestellte Evaluierung fehlt. Es werden folglich weitere, auf dieser Arbeit aufbauende Forschungsaktivitäten empfohlen. Dem Ausblick auf weiterführende Forschungsaufgaben und der Zusammenfassung dieser Arbeit widmet sich das folgende letzte Kapitel.

Kapitel 9

Zusammenfassung und Ausblick

Neben der Zusammenfassung (Abschnitt 9.1) geht dieses Kapitel auf die konkreten nächsten Schritte für die betriebliche Operationalisierung der kennzahlenorientierten Gestaltung des Softwareentwicklungsprozesses beim Kooperationspartner sowie auf einige wissenschaftliche Fragestellungen für zukünftige Forschungsaktivitäten ein (Abschnitt 9.2).

9.1 Zusammenfassung

Diese Arbeit widmete sich der kennzahlenorientierten Gestaltung des Softwareentwicklungsprozesses in produzierenden Betrieben. Durch die mit dem Begriff „Industrie 4.0“ beschriebene industrielle Digitalisierung wächst für produzierende Betriebe die Notwendigkeit, intelligente Produkte zu entwickeln und zu vertreiben. Intelligente Produkte beruhen auf einem Zusammenwirken von mehreren Domänen, zum Beispiel der Mechanik, der Elektronik und der Softwaretechnik. Sie werden folglich domänenübergreifend entwickelt. Diese Arbeit ging davon aus, dass der Softwaretechnik eine besondere Bedeutung eingeräumt werden kann, da ohne Software eine digitalisierte Industrie nicht möglich ist. Daher gewinnt der Softwareentwicklungsprozess an strategischer Bedeutung in produzierenden Betrieben.

Es ist die Aufgabe des Managements, strategisch wichtige Prozesse, folglich auch den Softwareentwicklungsprozess, zu überwachen und zu gestalten. Mit dem Management sind Führungskräfte in oberen Hierarchieebenen gemeint. Die Gestaltung von betrieblichen Prozessen dient der Umsetzung von strategischen Zielen, die vom Management formuliert werden. Für jedes der strategischen Ziele sind Kennzahlen zu definieren, die die Zielerreichung anzeigen. Um dem Management die Kennzahlen zuführen zu können, ist in der Prozessgestaltung deren Erfassung durch ein Informationsverarbeitungssystem zu berücksichtigen.

Eine Motivation für diese Untersuchung ergab sich aus der konkreten Situation beim Kooperationspartner dieser Arbeit, dem Geschäftsbereich Automatisierungstechnik des Unternehmens Phoenix Contact. Zwar gestaltet der Kooperationspartner bereits den Produktionsprozess mit der Vorgabe, Kennzahlen aus dem Produktionsprozess zu erfassen, allerdings fehlt bislang eine kennzahlenorientierte Gestaltung des Softwareentwicklungsprozesses, und somit ein dazugehörendes Informationsverarbeitungssystem. Dies ist ein IT-System, das Daten aus dem Prozess ermittelt, zu Kennzahlen verarbeitet und grafisch darstellt [Ben07]. Daher ist das Management nur unzureichend in der Lage, den Softwareentwicklungsprozess zu überwachen und zu steuern.

Eine weitere Motivation ergab sich aus einer Umfrage der IHK Ostwestfalen zu Bielefeld, die darauf hindeutet, dass in anderen produzierenden Betrieben eine kennzahlenorientierte Gestaltung des Softwareentwicklungsprozesses ebenfalls fehlt, aber auch gewünscht ist.

Aus diesen Motivationen leitete sich die Forschungsfrage dieser Arbeit ab:
Wie kann der Softwareentwicklungsprozess in produzierenden Betrieben kennzahlenorientiert gestaltet werden?

Aus der Forschungsfrage ergaben sich drei Detailfragestellungen, deren Herleitung in Abschnitt 1.4 begründet wurde.

1. Wie können SW-Produktionskennzahlen, die die Semantik der äquivalenten HW-Produktionskennzahlen beibehalten, bestimmt werden?
2. Wie sollte der Softwareentwicklungsprozess aufgebaut sein, damit die definierten SW-Produktionskennzahlen und Softwarekennzahlen erfasst werden können?
3. Wie sollte ein Informationsverarbeitungssystem aufgebaut sein, das SW-Produktionskennzahlen und Softwarekennzahlen erfassen und verarbeiten kann?

Die Lösungen für diese Fragen ermöglichen dem Management eine integrierte Prozesssteuerung des Produktions- und Softwareentwicklungsprozesses. Abbildung 9.1 zeigt schematisch diese Prozesssteuerung: Das Management überwacht und gestaltet den Produktionsprozess mit der Menge an HW-Produktionskennzahlen K_{HW}. Da es mit der Semantik dieser HW-Produktionskennzahlen vertraut ist, kann es ebenfalls den Softwareentwicklungsprozess mit K_{SW}, der Menge an semantisch äquivalenten SW-Produktionskennzahlen, überwachen und steuern.

Um die Fragestellung zur Bestimmung von SW-Produktionskennzahlen zu beantworten, wurde in dieser Arbeit eine neue Methode entwickelt: die Reversed Goal-Question-Metric-Methode (RGQM). Die RGQM-Methode besteht aus acht Bearbeitungsschritten und basiert auf der bekannten GQM-Methode. Ausgehend von einer bereits in der Produktionsdomäne eingesetzten HW-Produktionskennzahl werden die dazugehörende Frage, das

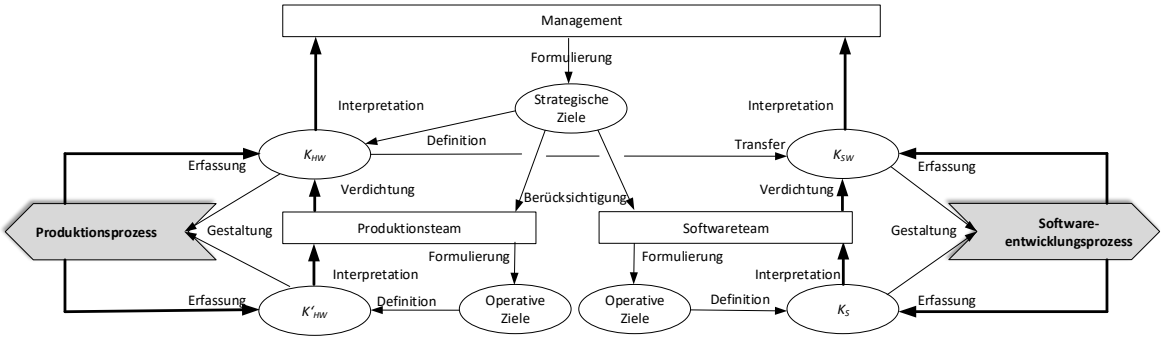


Abbildung 9.1: Integrierte Prozesssteuerung durch das Management

Ziel und die Interpretation ermittelt. Falls das Ziel für den Softwareentwicklungsprozess gültig ist, wird eine domänenspezifische Frage formuliert, die mit einer SW-Produktionskennzahl beantwortet wird. In dieser Arbeit wurden mit der RGQM-Methode fünf HW-Produktionskennzahlen in SW-Produktionskennzahlen transferiert: *First Pass Rate*, *Technische Rückläuferrate*, *Servicegrad*, *Wertschöpfung* und *Produktivität*.

Damit die SW-Produktionskennzahlen in dem Softwareentwicklungsprozess erfasst werden können, muss dieser mehrere Anforderungen erfüllen. Diese Anforderungen wurden während der schrittweisen Bearbeitung der RGQM-Methode formuliert.

Gemäß der Zielsituation in Abbildung 9.1 nutzen die Softwareteams Softwarekennzahlen, die die Zielerreichung operativer Ziele anzeigen. Zur Bestimmung der Softwarekennzahlen aus den operativen Zielen wurde die GQM-Methode angewendet. Es wurden fünf operative Ziele definiert, die dazugehörenden Fragen formuliert und Softwarekennzahlen bestimmt, die die Fragen beantworten. Damit die Softwarekennzahlen in dem Softwareentwicklungsprozess erfasst werden können, muss dieser wiederum bestimmte Anforderungen erfüllen. Diese wurden bei der Anwendung der GQM-Methode formuliert.

Zur Beantwortung der Fragestellung zum Aufbau des Softwareentwicklungsprozesses wurde das Vorgehensmodell des Kooperationspartners berücksichtigt. Dieses Vorgehensmodell basiert auf dem V-Modell der DIN EN 61508-3. In dieser Arbeit wurde ein Datenmodell für das V-Modell der DIN EN 61508-3 entwickelt und als UML-Klassendiagramm beschrieben. Es ist derart gestaltet, dass es die Anforderungen für die Erfassung der vorher definierten SW-Produktionskennzahlen und Softwarekennzahlen erfüllt. Das Ergebnis dieser Gestaltung ist das *Sliced V-Modell*. Eine wesentliche Eigenschaft des Sliced V-Modells ist die Anwendung von Work Items, die untereinander verlinkt werden.

Dem Entwurf des Sliced V-Modells und der Bestimmung von SW-Produktionskennzahlen und Softwarekennzahlen folgte die Definition der Berechnungsgrundlagen aller Kennzahlen. Die Semantik der HW-Produktionskennzahlen und der dazugehörenden SW-Produktionskennzahlen wurde verglichen. Der Vergleich zeigte, dass durch die An-

wendung der RGQM-Methode eine semantische Äquivalenz beider Ausprägungen einer Produktionskennzahl erreicht wird.

Um die Fragestellung zum Aufbau des Informationsverarbeitungssystems zu beantworten, wurde ein Prototyp, genannt *SofProSys*, entwickelt und angewendet. Bei der Entwicklung von *SofProSys* wurde das Sliced V-Modell genutzt. Eine erste Anwendung von *SofProSys* erfolgte auf Basis des Sliced V-Modells, das während der Entwicklung von *SofProSys* entstand. Diese praxisnahe Anwendung zeigte, dass alle definierten Kennzahlen erfasst werden können. Damit wurde eine grundsätzliche Eignung des entworfenen Informationsverarbeitungssystems für den Einsatz beim Kooperationspartner demonstriert.

Im Rahmen der praktischen Anwendung wurden zwei Softwareprodukte des Kooperationspartners mit *SofProSys* analysiert. *SofProSys* konnte nahezu alle SW-Produktionskennzahlen und alle Softwarekennzahlen erfassen. Da einige für den Servicegrad benötigte Daten bislang nicht in den operativen Softwareentwicklungsprozess überführt wurden, konnte der Servicegrad nicht erfasst werden. Um eine grafische Darstellung zu ermöglichen, mussten die Kennzahlen in den von *SofProSys* generierten MS Excel-Dateien manuell nachbearbeitet werden.

Die aufbereiteten MS Excel-Dateien wurden einigen Stakeholdern der Softwareteams vorgestellt. Diese bestätigten, dass die erfassten Kennzahlen plausibel sind und dass *SofProSys* die Softwareentwicklung unterstützen wird, sofern es in den operativen Betrieb überführt wird. Jedoch müsste durch Mitarbeiterschulungen darauf hingewirkt werden, die bisherigen Abweichungen in der Anwendung des Sliced V-Modells zu reduzieren.

Die grafisch aufbereiteten SW-Produktionskennzahlen der beiden Softwareprodukte wurden dem Geschäftsbereichsleiter vorgestellt. Die grafische Aufarbeitung erfolgte mit Bordmitteln von MS Excel. Der Geschäftsbereichsleiter war nach der Präsentation der Diagramme für die First Pass Rate und die Technischen Rückläuferrate sofort in der Lage, eine zielgerichtete Diskussion zu führen. Insbesondere wurden die gemessenen Ist-Werte erörtert. Ihm war sehr wohl bewusst, dass die Ist-Werte keinen Bezug zu den ihm bekannten Ist-Werten aus der Produktion haben können. Jedoch zeigte er sich begeistert von der Möglichkeit zukünftig das ihm vertraute Verfahren zur Überwachung und Steuerung des Produktionsprozesses, also das Festlegen von Soll-Werten und die regelmäßige Überprüfung der Ist-Werte, auch für den Softwareentwicklungsprozess anwenden zu können.

Es kann zusammengefasst werden, dass sowohl die praxisnahe als auch die praktische Anwendung erfolgreich waren. Der Prototyp des Informationsverarbeitungssystems ist in der Lage, Softwarekennzahlen und SW-Produktionskennzahlen, die die Semantik der dazugehörenden HW-Produktionskennzahlen beibehalten, IT-gestützt zu erfassen und anzuzeigen. Eine IT-gestützte Datenerfassung und -verarbeitung ist für eine erfolgreiche Über-

führung der kennzahlenorientierten Gestaltung des Softwareentwicklungsprozesses in die betriebliche Praxis beim Kooperationspartner notwendig.

Im Rahmen dieser Arbeit wurde beim Kooperationspartner nicht evaluiert, ob die kennzahlenorientierte Gestaltung des Softwareentwicklungsprozesses tatsächlich zur Erreichung der strategischen Ziele beiträgt. Eine derartige Evaluierung müsste über mehrere Jahre erfolgen. Während dieser Zeit müssten ausgehend von den strategischen und operativen Zielen Maßnahmen definiert und umgesetzt und die Zielerreichung müsste mithilfe der erfassten Kennzahlen überprüft werden.

Die Erkenntnisse dieser Arbeit wurden verallgemeinert. Es wurde theoretisch dargelegt, dass sowohl die RGQM-Methode als auch *SofProSys* in anderen produzierenden Betrieben eingesetzt werden könnten. So wurde die RGQM-Methode in einem theoretischen Beispiel angewendet. Damit *SofProSys* in andere Umgebungen übertragen werden kann, müsste entweder das Sliced V-Modell oder zumindest ein Work Item-basiertes Vorgehensmodell, in dem die Quelltextänderungen in einem Versionsmanagementsystem eindeutig einer Softwarefunktion oder einem Fehler zugeordnet werden, Anwendung finden.

9.2 Ausblick

Obwohl in dieser Arbeit eine kennzahlenorientierte Gestaltung des Softwareentwicklungsprozesses erfolgreich entwickelt und evaluiert wurde, bedarf es weiterführender Aktivitäten, um sie in die betriebliche Praxis des Kooperationspartners zu überführen. Darüber hinaus ergeben sich wissenschaftliche Fragestellungen, die beantwortet werden sollten.

9.2.1 Überführung in die betriebliche Praxis

Die Definition von strategischen und operativen Zielen sowie die Bestimmung von SW-Produktionskennzahlen bzw. Softwarekennzahlen ist eine kontinuierliche Aufgabe in der kennzahlenorientierten Prozessgestaltung. In dieser Arbeit wurde dieser Ablauf einmal durchgeführt, in der betrieblichen Praxis sollte dies allerdings in regelmäßigen Abständen erfolgen. Darüber hinaus sollte kontinuierlich bewertet werden, ob die bislang definierten Kennzahlen für die Anzeige der Zielerreichungen ausreichen und ob sie zur Verbesserung des Softwareentwicklungsprozesses beitragen. Beispielsweise wäre zu prüfen, ob Kennzahlen für die Anzeige der Qualität der Softwaredokumentation bestimmt werden sollten (vgl. Abschnitt 2.1.2.4.3).

Die Ergebnisse der Arbeit wurden mit Hilfe eines Prototyps, *SofProSys*, evaluiert. Damit der Prototyp im operativen Betrieb nutzbar ist, müsste er systematisch getestet und

funktional ergänzt werden. Insbesondere fehlen Funktionen für die grafische Aufbereitung der Softwarekennzahlen und der SW-Produktionskennzahlen. Bislang werden diese lediglich in Excel-Tabellen angezeigt, was die intuitive Bewertung und Interpretation der Kennzahlen erschwert. Für die grafische Aufbereitung wäre eine Integration in eine sich im Aufbau befindliche Webplattform denkbar. Ziel der Webplattform ist es, die HW-Produktionskennzahlen zu erfassen und grafisch aufzubereiten. Durch die Integration der SW-Produktionskennzahlen würde ein Informationsverarbeitungssystem entstehen, das dem Management einen homogenen Blick auf den Produktions- und Softwareentwicklungsprozess ermöglicht. Die in Abbildung 9.1 gezeigte integrierte Prozesssteuerung wäre in der betrieblichen Praxis erreicht. Da sich allerdings diese Webplattform im Aufbau befindet, war es bislang nicht möglich, die Ergebnisse dieser Arbeit darin zu ergänzen. Bei der grafischen Aufbereitung der Kennzahlen sollten Gestaltungskriterien für Dashboards berücksichtigt werden, wie sie zum Beispiel in [SNM15] aufgeführt sind.

Es kann davon ausgegangen werden, dass durch die Anwendung der Ergebnisse dieser Arbeit in der betrieblichen Praxis das Domänenwissen des Managements über den Softwareentwicklungsprozess wachsen wird. Folglich wird das Management in Zukunft befähigt, strategische Ziele zu formulieren, die nur für den Softwareentwicklungsprozess gelten. Zur Überprüfung dieser strategische Ziele wird die Menge K_{SW} durch Kennzahlen angereichert, die aus diesen strategischen Zielen hergeleitet werden. Für diese Kennzahlen ist kein Transferprozess nötig, sondern lediglich ein GQM-basierter Prozess zur Kennzahlenbestimmung. Das konkrete Vorgehen für die Anreicherung der Menge K_{SW} mit derartigen Kennzahlen ist das Thema weiterführender Aktivitäten.

9.2.2 Wissenschaftliche Fragestellungen

Eine regelmäßige Fragestellung in der Softwareentwicklung ist die Schätzung des Aufwands und der Entwicklungsdauer für die Umsetzung neuer Softwarefunktionen. Durch den in dieser Arbeit entwickelten Ansatz werden SW-Produktionskennzahlen und Softwarekennzahlen „rückblickend“ erfasst und ausgewertet. Daraus ergibt sich die Fragestellung, inwiefern diese historischen Daten zur Schätzung des Aufwands und der Entwicklungsdauer neuer Softwarefunktionen verwendet werden können.

Der in dieser Arbeit entwickelte Ansatz der kennzahlenorientierten Gestaltung des Softwareentwicklungsprozesses bewirkt ein präzises Datenmodell: das Sliced V-Modell. Durch Kenntnis dieses Datenmodells ist ein Informationsverarbeitungssystem in der Lage, die Softwarekennzahlen und SW-Produktionskennzahlen zu berechnen. Jedoch zeigt die Arbeit, dass das Eintragen aller benötigten Daten durch die Softwareteams in der betrieblichen Praxis problematisch sein kann. Die Stakeholder in der Softwareentwicklung sind

der Meinung, dass es immer Abweichungen vom Sliced V-Modell geben wird. Zwar könne darauf hingewirkt werden, diese Abweichungen so gering wie möglich zu halten, ganz vermeidbar seien sie jedoch nicht. Ausgehend von dieser Situation wäre Gegenstand weiterer Forschungsarbeiten die Fragestellung, wie eher unstrukturierte Daten für die Erfassung von Softwarekennzahlen oder SW-Produktionskennzahlen verarbeitet werden können. Mit Big Data-Technologien und Machine Learning-Algorithmen stehen mittlerweile Methoden zur Verfügung, um unstrukturierte Daten auswerten und analysieren zu können. Ihre Anwendung in der Domäne der Softwaremessung wurde zwar bereits als Themenkomplex erkannt, konkrete Forschungsergebnisse fehlen jedoch bislang [HSD16].

Diese Arbeit war motiviert durch die zunehmende Digitalisierung der Industrie, in der immer mehr intelligente Produkte benötigt werden. Intelligente Produkte beruhen auf einem Zusammenwirken verschiedener Domänen, zum Beispiel der Mechanik, der Elektrotechnik/Elektronik und der Softwaretechnik. Da der Softwaretechnik in dieser Arbeit eine besondere Rolle zugeordnet wurde, stand die kennzahlenorientierte Gestaltung des Softwareentwicklungsprozesses im Mittelpunkt dieser Arbeit. Durch eine Ausweitung des wissenschaftlichen Fokus auf die kennzahlenorientierte Gestaltung des domänenübergreifenden, auf Methoden des Systems Engineerings basierenden Produktentwicklungsprozesses würde sich die Frage stellen: *Wie kann der domänenübergreifende Produktentwicklungsprozess in produzierenden Betrieben kennzahlenorientiert gestaltet werden?*

Um diese Frage beantworten zu können, müsste der Produktentwicklungsprozess domänenübergreifend untersucht und gestaltet werden. Der Produktentwicklungsprozess umfasst neben der Softwareentwicklung unter anderem die mechanische und elektrische Konstruktion (M-CAD/E-CAD). Die dazugehörenden Produktentwicklungsdaten werden in PLM-Systemen verwaltet (vgl. Abschnitt 2.4.2). Alle in PLM-Systemen verwalteten Produktentwicklungsdaten können IT-basiert erfasst und verarbeitet werden.

Literaturverzeichnis

[Abr14] Abran, A.: Software Estimation: Transforming Dust into Pots of Gold? In: *Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)*, 2014, S. 64-65

[ADG+09] Adelt, P. ; Donoth, J. ; Gausemeier, J. ; Geisler, J. ; Henkler, S. ; Kahl, S. ; Klöpper, B. ; Krupp, A. ; Münch, E. ; Oberthür, S. ; Paiz, C. ; Porrmann, M. ; Radkowski, R. ; Romaus, C. ; Schmidt, A. ; Schulz, B. ; Voecking, H. ; Witkowski, U. ; Witting, K. ; Znamenshchykov, O.: *Selbstoptimierende Systeme des Maschinenbaus - Definitionen, Anwendungen, Konzepte*. Verlagsschriftenreihe des Heinz Nixdorf Instituts, Paderborn, 2009 (Bd. 234)

[AG12] Albers, A. ; Gausemeier, J.: Von der fachdisziplinorientierten Produktentwicklung zur Vorausschauenden und Systemorientierten Produktentstehung. In: *Smart Engineering*. Springer Berlin Heidelberg, 2012, S. S. 17–29

[Alb79] Albrecht, A. J.: Measuring Application Development Productivity. In: *Proceedings of the IBM Application Development Symposium* Bd. 83. IBM Cooperation, 1979, S. 83–92

[Apa17] *Apache Subversion*. https://subversion.apache.org. [31.10.2017]

[ASSH16] Antinyan, V. ; Staron, M. ; Sandberg, A. ; Hansson, J.: A Complexity Measure for Textual Requirements. In: *Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)*, 2016, S. 148–158

[Bal97] Balzert, H.: *Lehrbuch der Software-Technik, Bd. 2: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung* . 1. Aufl. Spektrum Akademischer Verlag, 1997

[Bal00] Balzert, H.: *Lehrbuch der Softwaretechnik, Bd.1: Software-Entwicklung.* 2. Aufl. Spektrum Akademischer Verlag GmbH, 2000

[BB10] Bailly, H. W. ; Bülow, F. von: *Die ISO 9001:2008: Interpretation der Anforderungen der DIN EN ISO 9001:2008-12 unter Berücksichtigung der ISO 9004:2009.* 6. Aufl. TÜV Media GmbH, 2010

[BB16] Blanchard, B. S. ; Blyler, J. E.: *System Engineering Management.* 5. Aufl. Wiley, 2016

[BBBC09] Baldassarre, M. ; Boffoli, N. ; Bruno, G. ; Caivano, D.: Statistically Based Process Monitoring: Lessons from the Trench. In: *Trustworthy Software Development Processes* Bd. 5543. Springer Berlin Heidelberg, 2009, S. 11–23

[Bec05] Becker, T.: *Prozesse in Produktion und Supply Chain optimieren.* Springer Berlin Heidelberg, 2005

[Ben07] Benson, A.: *Qualitätssteigerung in komplexen Entwicklungsprojekten durch prozessbegleitende Kennzahlensysteme: Vorgehen zur Herleitung, Einführung und Anwendung.* 1. Aufl. Cuvillier Verlag, 2007

[BGA14] Bajwa, S. S. ; Gencel, C. ; Abrahamsson, P.: Software Product Size Measurement Methods. In: *Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)*, 2014, S. 176–190

[BHL+07] Basili, V. ; Heidrich, J. ; Lindvall, M. ; Münch, J. ; Regardie, M. ; Rombach, D. ; Seaman, C. ; Trendowicz, A.: Bridging The Gap Between Business Strategy And Software Development. In: *International Conference on Information Systems (ICIS)*, 2007

[Bin97] Binder, R. V.: Can a manufacturing quality model work for software? In: *IEEE Software* 14 (1997), Sep, Nr. 5, S. 101–105

[BKK91] Banker, R. D. ; Kauffman, R. J. ; Kumar, R.: An empirical test of object-based output measurement metrics in a computer aided software engineering (CASE) environment. In: *Journal of Management Information Systems* 8 (1991), Dezember, Nr. 3, S. 127–150

[BMS02] Barry, E. J. ; Mukhopadhyay, T. ; Slaughter, S. A.: Software Project Duration and Effort: An Empirical Study. In: *Information Technology and Management* 3 (2002), Januar, Nr. 1-2, S. 113–136

[Boe79] Boehm, B. W.: Guidelines for Verifying and Validating Software Requirements and Design Specifications. In: *European Conference on Applied Information Technology of the International Federation for Information Processing (Euro IFIP)*, 1979, S. 711–719

[Bro97] Brown, M.: *Kennzahlen: harte und weiche Faktoren erkennen, messen und bewerten.* Carl Hanser Verlag, 1997

[Bru91] Bruns, M.: *Systemtechnik: Ingenieurwissenschaftliche Methodik zur interdisziplinären Systementwicklung.* Springer Berlin Heidelberg, 1991

[Bug17] *Bugzilla.* http://www.bugzilla.org. [31.10.2017]

[BW84] Basili, V. ; Weiss, D.: A Methodology for Collecting Valid Software Engineering Data. In: *IEEE Transactions on Software Engineering* SE-10 (1984), Nov, Nr. 6, S. 728–738

[CA01] Carmel, E. ; Agarwal, R.: Tactical Approaches for Alleviating Distance in Global Software Development. In: *IEEE Software* 18 (2001), März, Nr. 2, S. 22–29

[Car94] Card, D.: Statistical process control for software? In: *IEEE Software* 11 (1994), May, Nr. 3, S. 95–97

[Car06] Card, D.: The Challenge of Productivity Measurement. In: *Pacific Northwest Software Quality Conference (PNSQC)*, 2006

[Cry17] *CRYSTAL - Critical System Engineering Accelaration (2012-2016).* http://www.crystal-artemis.eu/, [31.10.2017]

[DA11] Dumke, R. ; Abran, A.: *COSMIC Function Points: Theory and Advanced Practices.* Taylor & Francis, 2011

[DCKV08] Deininger, W. ; Cottingham, C. ; Kanner, L. ; Verbeke, M. A.: Systems Engineering Data Book (SEDB) – A Product Baseline Definition and Tracking Tool. In: *International Conference on Systems Engineering (ICSENG)*, 2008, S. 19–24

[DD15] Deuter, A. ; Dreyer, J.: Reversed-GQM: Ein Ansatz zur Wiederverwendung von Kennzahlen. In: *Metrikon 2015 - Praxis der Software-Messung.* Shaker Verlag, Aachen, 2015, S. 3–14

[DE14] Deuter, A. ; Engels, G.: Measuring the Software Size of Sliced V-model Projects. In: *Joint Conference of the International Workshop on Software*

Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2014, S. 233–242

[DEHW13] Dumke, R. ; Ebert, C. ; Heidrich, J. ; Wille, C.: Messung und Bewertung von Software. In: *Informatik-Spektrum* 36 (2013), Nr. 6, S. 508–519

[Deu12] Deuter, A.: Messung der Software-Produktivität in einem Work Item-basierten V-Modell. In: *Metrikon 2012 - Praxis der Software-Messung*. Shaker Verlag, Aachen, 2012, S. 69–84

[Deu13] Deuter, A.: Slicing the V-model - Reduced effort, higher flexibility. In: *International Conference on Global Software Engineering (ICGSE)*, 2013, S. 1–10

[Deu14] Deuter, A.: Software wird auch im Maschinenbau zur Kernkompetenz. In: *IEE Elektrische Automatisierung + Antriebstechnik* 10 (2014), S. 16–18

[Deu16] Deuter, A.: *Software measurement in the context of Industry 4.0*. Workshop in Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2016

[DK15] Deuter, A. ; Koch, H.-J.: Applying Manufacturing Performance Figures to Measure Software Development Excellence. In: *Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)*, 2015, S. 62–77

[DKE10a] *DIN EN 61508-3: Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme-Teil 3: Anforderungen an Software*. 2010

[DKE10b] *DIN EN 61508-4: Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme-Teil 4: Begriffe und Abkürzungen*. 2010

[DOO17] *IBM Rational DOORS*. http://www-03.ibm.com/software/products/de/ ratidoor, [31.10.2017]

[DP12] Dondey, H. ; Peron, C.: Software Qualimetry at Schneider Electric - a field background. In: *Embedded Real Time Software and Systems ERTS²*. Toulouse, France, 2012

[ED07] Ebert, C. ; Dumke, R.: *Software Measurement: Establish - Extract - Evaluate - Execute*. Springer-Verlag Berlin Heidelberg, 2007

[EKM17] Eigner, M. (Hrsg.) ; Koch, W. (Hrsg.) ; Muggeo, C. (Hrsg.): *Modellbasierter Entwicklungsprozess cybertronischer Systeme: Der PLM-unterstützte Referenzentwicklungsprozess für Produkte und Produktionssysteme.* Springer Vieweg, 2017

[Elm08] Elm, J. P.: A Study of Systems Engineering Effectiveness - Initial Results. In: *2008 2nd Annual IEEE Systems Conference*, 2008, S. 1–7

[ERZ14] Eigner, M. (Hrsg.) ; Roubanov, D. (Hrsg.) ; Zafirov, R. (Hrsg.): *Modellbasierte virtuelle Produktentwicklung.* Springer Vieweg, Berlin Heidelberg, 2014

[ES09] Eigner, M. ; Stelzer, R.: *Product Lifecycle Management: Ein Leitfaden für Product Development und Life Cycle Management.* 2. Aufl. Springer Berlin Heidelberg, 2009

[FHZ+15] Fiegler, A. ; Herden, S. ; Zwanziger, A. ; Meissner, M. ; Dumke, R.: Qualitätsbemessung von automatisierten ITIL Prozessen in Cloud Systemen am Beispiel der bedingten Entropie. In: *Metrikon 2015 - Praxis der Software-Messung.* Shaker Verlag, Aachen, 2015, S. 133–146

[FLM+98] Fuggetta, A. ; Lavazza, L. ; Morasca, S. ; Cinti, S. ; Oldano, G. ; Orazi, E.: Applying GQM in an Industrial Software Factory. In: *ACM Transactions on Software Engineering and Methodology* 7 (1998), Oktober, Nr. 4, S. 411–448

[FP97] Fenton, N. ; Pfleeger, S. L.: *Software metrics (2nd ed.): a rigorous and practical approach.* Boston, MA, USA : PWS Publishing Co., 1997

[FPG+04] Faulk, S. ; Porter, A. ; Gustafson, J. ; Tichy, W. ; Johnson, P. ; Votta, L.: Measuring HPC productivity. In: *International Journal of High Performance Computing Applications* (2004), S. 459–473

[FS99] Friedag, H. R. ; Schmidt, W.: *Balanced Scorecard. Mehr als ein Kennzahlensystem.* Haufe Verlag, 1999

[Gau14] Gaulke, M.: *Praxiswissen COBIT.* 2. Aufl. dpunkt.verlag, 2014

[GCW+13] Gausemeier, J. ; Czaja, A. M. ; Wiederkehr, O. ; Dumitrescu, R. ; Tschirner, C. ; Steffen, D.: Studie: Systems Engineering in der industriellen Praxis. In: *9. Paderborner Workshop: Ëntwurf mechatronischer Systeme"* (2013)

[GF94] Gotel, O. C. Z. ; Finkelstein, C. W.: An analysis of the requirements traceability problem. In: *International Conference on Requirements Engineering (ICRE)*, 1994, S. 94–101

[GFL+13] Génova, G. ; Fuentes, J. M. ; Llorens, J. ; Hurtado, O. ; Moreno, V.: A framework to measure and improve the quality of textual requirements. In: *Requirements Engineering* 18 (2013), Nr. 1, S. 25–41

[Gla03] Gladen, W.: *Kennzahlen- und Berichtssysteme.* Gabler Verlag, 2003

[Gol13] Goldkuhl, G.: Action research vs. design research : using practice research as a lens for comparison and integration. In: *IT Artefact Design & Workpractice Improvement (ADWI)*, 2013

[Han10] Hanser, E.: *Agile Prozesse: Von XP über Scrum bis MAP.* 1. Aufl. Springer-Verlag Berlin Heidelberg, 2010

[HD06] Hartmann, D. ; Dymond, R.: Appropriate agile measurement: Using metrics and diagnostics to deliver business value. In: *Agile Conference (AGILE)*, 2006, S. 126–134

[HdFV15] Haberfellner, R. ; de Weck, O. ; Fricke, E. ; Vössner, S.: *Systems Engineering: Grundlagen und Anwendung.* 13., aktualisierte Aufl. Zürich : Orell Füssli, 2015

[Hir08] Hiranabe, K.: *Kanban Applied to Software Development: from Agile to Lean.* http://www.infoq.com/articles/hiranabe-lean-agile-kanban. 2008. – [31.10.2017]

[HJK14] Hinrichsen, S. ; Jungkind, W. ; Könneker, M.: Industrial Engineering - Begriff, Methodenauswahl und Lehrkonzept. In: *Betriebspraxis & Arbeitsforschung* (2014), September, Nr. 221, S. 28–35

[HK15] Hartenstein, S. ; Könnecke, H.: Metrics for Evaluation of Trustworthiness-By-Design Software Development Process. In: *Metrikon 2015 - Praxis der Software-Messung.* Shaker Verlag, Aachen., 2015, S. 95–106

[HKLR84] Hesse, W. ; Keutgen, H. ; Luft, A. ; Rombach, D.: Ein Begriffssystem für die Softwaretechnik - Vorschlag zur Terminologie. In: *Informatik-Spektrum* 7 (1984), S. 200–213

[HMT09] Heidrich, J. ; Münch, J. ; Trendowicz, A.: Messbasierte Ausrichtung von Softwarestrategien an Geschäftszielen. In: *Fachzeitschrift für Information Management & Consulting (IM)* (2009), Februar, S. 82–89

[Hor11] Horvath, P.: *Controlling.* Franz Vahlen Verlag, 2011 (11. Aufl)

[HP07] Hinrichsen, S. ; Peters, M.: Kennzahlensysteme. In: *Lexikon Arbeitsgestaltung: Best Practice im Arbeitsprozess.* Universum-Verlag, 2007, S. 708–710

[HSD16] Hentschel, J. ; Schmietendorf, A. ; Dumke, R.: Big Data benefits for the Software Measurement Community. In: *Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)*, 2016, S. 108–114

[IPPS02] Ioannou, G. ; Papalexandris, A. ; Prastacos, G. ; Soderquist, E.: Implementing a balanced scorecard at a software development company. In: *International Engineering Management Conference (IEMC)* Bd. 2, 2002, S. 743–748

[ISO01] *ISO/IEC 9126 - Software engineering - Product quality.* 2001

[ISO03] *ISO/IEC 19761:2003 - Software engineering - COSMIC-FFP - A functional size measurement method.* 2003

[ISO09] *ISO/IEC 20926 - IFPUG Functional Size Measurement Method.* 2009

[ISO10] *ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models.* 2010

[ISO11] *ISO/IEC/IEEE 42010:2011 - Systems and software engineering - Architecture description.* 2011

[ISO15] *ISO/IEC/IEEE 15288:2015 - Systems and software engineering – System life cycle processes.* 2015

[ISO16a] *ISO/IEC 25022 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Measurement of quality in use.* 2016

[ISO16b] *ISO/IEC 25023 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Measurement of system and software product quality.* 2016

[ISO17] *ISO/IEC/IEEE 15939 - Systems and Software Engineering - Measurement Process.* 2017

[IV09] Iivari, J. ; Venable, J.: Action research and design science research–seemingly similar but decisively dissimilar. In: *European Conference on Information Systems (ECIS)*, 2009, S. 1–13

[JAB12] Jang, J. ; Agrawal, A. ; Brumley, D.: ReDeBug: Finding Unpatched Code Clones in Entire OS Distributions. In: *IEEE Symposium on Security and Privacy* (2012), S. 48–62

[JLC12] Jones, C. ; Layman, B. ; Clark, E.: *Practical Software Measurement: Objective Information for Decision Makers*. Prentice Hall, 2012

[Jon17] Jones, C.: *A short history of the cost per defect metric*. http://www.ifpug.org/Documents/Jones-CostPerDefectMetricVersion4.pdf, 2013, [31.10.2017]

[Juk11] Jukka Kääriäinen: *Towards an Application Lifecycle Management Framework*. Bd. 179. VTT, 2011

[JYW+11] Jeng, B. ; Yeh, D. ; Wang, D. ; Chu, S.-L. ; Chen, C.-M.: A Specific Effort Estimation Method Using Function Point. In: *Journal of Information Science and Engineering* 27 (2011), Nr. 4, S. 1363–1376

[Kan03] Kan, S.: *Metrics and Models in Software Quality Engineering*. 2nd. Addison-Wesley, 2003

[Kap10] Kaplan, R. S.: Conceptual Foundations of the Balanced Scorecard. In: *Harvard Business School* (2010)

[Kar93] Karner, G.: Resource Estimation for Objectory Projects. In: *Objectory Systems SF AB* (1993)

[Kas08] Kasunic, M.: A Data Specification for Software Project Performance Measures: Results of a Collaboration on Performance Measurement / Software Engineering Institute. Carnegie Mellon University, Pittsburgh, Pennsylvania, 2008 (CMU/SEI-2008-TR-012)

[KBNAJ11] Khatibi Bardsiri, V. ; Norhayati Abang Jawawi, D.: Software Cost Estimation Methods: A Review. In: *Journal of Emerging Trends in Computing and Information Sciences* 2 (2011), S. 21–29

[Kil01] Kilpi, T.: Implementing a software metrics program at Nokia. In: *IEEE Software* 18 (2001), Nov, Nr. 6, S. 72–77

[KL14] Kuhrmann, M. ; Linssen, O.: Welche Vorgehensmodelle nutzt Deutschland? In: *Projektmanagement und Vorgehensmodelle 2014*, Gesellschaft für Informatik e.V. (GI), 2014 (Gemeinsame Tagung der Fachgruppen Projektmanagement (WI-PM) und Vorgehensmodelle (WI-VM) im Fachgebiet Wirtschaftsinformatik der Gesellschaft für Informatik e.V.)

[Klo17] *Klocwork*. http://www.klocwork.com, [31.10.2017]

[KN92] Kaplan, R. S. ; Norton, D. P.: The Balanced Scorecard - Measures that Drive Performance. In: *Harvard Business Review* (1992), Jan-Feb, S. 71–79

[Küt10] Kütz, M.: *Kennzahlen in der IT*. 4. Aufl. Heidelberg : dpunkt.verlag, 2010

[Kui14] Kuijpers, C.: Automated FPA (eFPA) in SAP Environment. In: *Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)*, 2014, S. 72–78

[KWH17] Kagermann, H. ; Wahlster, W. ; Helbig, J.: *Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0*. https://www.bmbf.de/files/ Umsetzungsempfehlungen_Industrie4_0.pdf, 2013. [31.10.2017]

[Leh94] Lehner, F.: *Software-Dokumentation und Messung der Dokumentationsqualität*. Carl Hanser Verlag, 1994

[LEPV10] Lanubile, F. ; Ebert, C. ; Prikladnicki, R. ; Vizcaíno, A.: Collaboration Tools for Global Software Engineering. In: *IEEE Software* 27 (2010), März, Nr. 2, S. 52–55

[LH11] Lind, K. ; Heldal, R.: A Model-based and Automated Approach to Size Estimation of Embedded Software Components. In: *International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2011, S. 334–348

[Lik04] Liker, J.: *The Toyota Way*. 1st. McGraw-Hill, 2004

[LMT+10] Lamersdorf, A. ; Munch, J. ; Torre, A. ; Sánchez, C. ; Rombach, D.: Estimating the Effort Overhead in Global Software Development. In: *International Conference on Global Software Engineering (ICGSE)*, 2010, S. 267–276

[Man17] *Manifest für Agile Softwareentwicklung*. http://agilemanifesto.org, [31.10.2017]

[MD06] Muranko, B. ; Drechsler, R.: Technical Documentation of Software and Hardware in Embedded Systems. In: *International Conference on Very Large Scale Integration (IFIP)*, 2006, S. 261–266

[ME98] Munson, J. C. ; Elbaum, S. G.: Code Churn: A Measure for Estimating the Impact of Code Change. In: *International Conference on Software Maintenance (ICSM)*, 1998, S. 24–31

[Moc03] Mockus, A.: Analogy Based Prediction of Work Item Flow in Software Projects: a Case Study. In: *International Symposium on Empirical Software Engineering (ISESE)*, IEEE Computer Society, 2003, S. 110–119

[Moh08] Mohagheghi, P.: Evaluating Software Development Methodologies Based on their Practices and Promises. In: *International Conference in Software Methodologies, Tools and Techniques (SOMET)*, 2008, S. 14–35

[MR93] Mayrhauser, A. von ; Roeseler, A.: Software process assessment and improvement using production models. In: *International Conference on Computer Software and Applications Conference (COMPSAC)*, 1993, S. 34–40

[MSH+12] Myklebust, T. ; Stålhane, T. ; Hanssen, G. K. ; Wien, T. ; Haugset, B.: Scrum, documentation and the IEC 61508-3:2010 software standard. In: *Probabilistic Safety Assessment & Management Conference (PSAM)*, 2012

[NC14] Nugroho, A. ; Chaudron, M.: The impact of UML modeling on defect density and defect resolution time in a proprietary system. In: *Empirical Software Engineering* 19 (2014), Nr. 4, S. 926–954

[NCK+15] Naedele, M. ; Chen, H.-M. ; Kazman, R. ; Cai, Y. ; Xiao, L. ; Silva, C. V.: Manufacturing Execution Systems. In: *Journal of Systems and Software* 101 (2015), März, Nr. C, S. 59–68

[OBB+14] Oriou, A. ; Bronca, E. ; Bouzid, B. ; Guetta, O. ; Guillard, K.: Manage the automotive embedded software development cost & productivity with the automation of a Functional Size Measurement Method (COSMIC). In: *Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)*, 2014, S. 1–4

[OMG17a] OMG: *Software & Systems Process Engineering Meta-Model Specification.* http://www.omg.org/spec/SPEM/2.0/PDF, [31.10.2017]

[OMG17b] OMG: *System Modeling Language.* https://www.omg.org/spec/SysML, [31.10.2017]

[OMG17c] OMG: *Unified Modeling Language.* https://www.omg.org/spec/UML/, [31.10.2017]

[OSL17] *OSLC-Open Services for Lifecycle Collaboration.* http://open-services.net/, [31.10.2017]

[PCL17] *PC-Lint.* http://www.gimpel.com/html/pcl.htm, [31.10.2017]

[Pet08] Peters, M.: *Methodik zur Entwicklung und Evaluation von Ziel- und Kennzahlensystemen für Produktentwicklungsprojekte in Virtuellen Unternehmen der Luftfahrtzulieferindustrie*, RWTH Aachen, Diss., 2008

[Pet11] Petersen, K.: Measuring and predicting software productivity: A systematic map and review. In: *Information and Software Technology* 53 (2011), Nr. 4, S. 317–343

[Pet13] Petre, M.: UML in Practice. In: *International Conference on Software Engineering (ICSE)*, 2013, S. 722–731

[PHBV08] Pries-Heje, J. ; Baskerville, R. ; Venable, J. R.: Strategies for Design Science Research Evaluation. In: *European Conference on Information Systems (ECIS)*, 2008, S. 1–12

[Pla17] Plattform Industrie 4.0: *Was ist Industrie 4.0?* http://www.plattform-i40.de/I40/Navigation/DE/Industrie40/WasIndustrie40/was-ist-industrie-40.html. [31.10.2017]

[Pol17] *Polyspace.* http://www.mathworks.de/products/polyspace, [31.10.2017]

[Pop13] Popp, G.: *Konfigurationsmanagement mit Subversion, Maven und Redmine Grundlagen für Softwarearchitekten und Entwickler.* 4. Aufl. Heidelberg : dpunkt Verlag, 2013

[PP11] Plewan, H.-J. ; Poensgen, B.: *Produktive Softwareentwicklung - Bewertung und Verbesserung von Produktivität und Qualität in der Praxis.* 1. Aufl. Heidelberg : dpunkt.verlag, 2011

[Pre08] Preissler, P.: *Betriebswirtschaftliche Kennzahlen: Formeln, Aussagekraft, Sollwerte, Ermittlungsintervalle.* Oldenbourg, 2008

[PW10] Petersen, K. ; Wohlin, C.: Software Process Improvement Through the Lean Measurement (SPI-LEAM) Method. In: *Journal of Systems and Software* 83 (2010), Juli, Nr. 7, S. 1275–1287

[RD13] Rahman, F. ; Devanbu, P.: How, and why, process metrics are better. In: *International Conference on Software Engineering (ICSE)*, 2013, S. 432–441

[REF92] *Methodenlehre des Arbeitsstudiums: Teil 2-Datenermittlung.* 7. Aufl. München : Carl Hanser Verlag, 1992

[RSRL08] Russ, R. ; Sperling, D. ; Rometsch, F. ; Louis, P.: Applying Six Sigma in the Field of Software Engineering. In: *Software Process and Product Measurement* Bd. 5338. Springer Berlin Heidelberg, 2008, S. 36–47

[SB99] Solingen, R. van ; Berghout, E.: *The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development.* McGraw-Hill, 1999

[SB12] Sonnenberg, C. ; Brocke, J. vom: Evaluation Patterns for Design Science Research Artefacts. In: *Practical Aspects of Design Science* Bd. 286. Springer Berlin Heidelberg, 2012, S. 71–83

[SB16] Schucht, C. ; Berger, N.: *Praktische Umsetzung der Maschinenrichtlinie.* Carl Hanser Verlag, 2016

[SBL10] Schlick, C. ; Bruder, R. ; Luczak, H.: *Arbeitswissenschaft.* Springer Berlin Heidelberg, 2010

[Sch95] Schwaber, K.: SCRUM Development Process. In: *Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA)*, 1995, S. 117–134

[Sch11] Schneidewind, N.: What can software engineers learn from manufacturing to improve software process and product? In: *Journal of Intelligent Manufacturing* 22 (2011), Nr. 4, S. 597–606

[See08] Seeger, K.: Zielorientierte Prozessgestaltung - Die Prozesse an der Strategie ausrichten. In: *Zielorientierte Unternehmensführung.* Wiesbaden : Gabler, 2008, S. 119–144

[Sen14] Sendler, U.: Industriegipfel Feldafing - System Leadership 2030. Ein Resümee erster Strategiegespräche zu Industrie 4.0. In: *Informatik-Spektrum* 37 (2014), Nr. 1, S. 54–72

[Sie17] Siemens Industry Software: *Polarion ALM.* https://polarion.plm.automation.siemens.com/. [31.10.2017]

[SJ12] Swaminathan, B. ; Jain, K.: Implementing the Lean Concepts of Continuous Improvement and Flow on an Agile Software Development Project: An Industrial Case Study. In: *AGILE India*, 2012, S. 10–19

[SJS12] Sjoberg, D. I. ; Johnsen, A. ; Solberg, J.: Quantifying the Effect of Using Kanban versus Scrum: A Case Study. In: *IEEE Software* 29 (2012), S. 47–53

[SL03] Spillner, A. ; Linz, T.: *Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester – Foundation Level nach ISTQB-Standard.* 1. Aufl. Heidelberg : dpunkt.verlag, 2003

[SMKN11] Staron, M. ; Meding, W. ; Karlsson, G. ; Nilsson, C.: Developing Measurement Systems: An Industrial Case Study. In: *Journal of Software Maintenance and Evolution* 23 (2011), März, Nr. 2, S. 89–107

[Sne87] Sneed, H. M.: *Softwaremanagement.* 1. Aufl. Köln : Verlagsgesellschaft Rudolf Möller, 1987

[Sne05] Sneed, H.: Reverse Engineering deutschsprachiger Fachkonzepte. In: *Workshop für Software Reengineering.* Bad Honnef, 2005

[Sne07] Sneed, H.: Testing against Natural Language Requirements. In: *International Conference on Quality Software (QSIC)*, 2007, S. 380–387

[Sne15] Sneed, H.: Measuring the Degree of Requirement Fulfilment. In: *Metrikon 2015 - Praxis der Software-Messung.* Shaker Verlag, Aachen., 2015, S. 41–50

[SNM15] Staron, M. ; Niesel, K. ; Meding, W.: Selecting the Right Visualization of Indicators and Measures - Dashboard Selection Model. In: *Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)*, 2015, S. 130–143

[Som12] Sommerville, I.: *Software Engineering.* 9. Aufl. Pearson Education, 2012

[SSB10] Sneed, H. M. ; Seidl, R. ; Baumgartner, M.: *Software in Zahlen die Vermessung von Applikationen.* Carl Hanser Verlag, 2010

[SSR+08] Seaman, C. B. ; Shull, F. ; Regardie, M. ; Elbert, D. ; Feldmann, R. L. ; Guo, Y. ; Godfrey, S.: Defect Categorization: Making Use of a Decade of Widely Varying Historical Data. In: *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2008, S. 149–157

[Ste06] Stepanek, G.: *Software Project Secrets: Why Software Projects Fail (Expert's Voice).* 1. Apress, 2006

[Sym88] Symons, C.: Function Point Analysis: Difficulties and Improvements. In: *IEEE Transactions on Software Engineering* 14 (1988), S. 2–11

[Sym10] Symons, C.: Software Industry Performance: What You Measure Is What You Get. In: *IEEE Software* 27 (2010), Nr. 6, S. 66–72

[TOR17] *TortoiseSVN*. http://tortoisesvn.tigris.org, [31.10.2017]

[TS12] Treude, C. ; Storey, M.-A.: Work Item Tagging: Communicating Concerns in Collaborative Software Development. In: *IEEE Transactions on Software Engineering* Bd. 38, 2012, S. 19–34

[Tsc16] Tschirner, C.: *Rahmenwerk zur Integration des modellbasierten Systems Engineering in die Produktentstehung mechatronischer Systeme*, Universität Paderborn, Dissertation, 2016

[VDI93] *VDI-Richtlinie 2221 Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte*. 1993

[VDI04] *VDI-Richtlinie 2206 Entwicklungsmethodik für mechatronische Systeme*. 2004

[Ven06] Venable, J.: A Framework for Design Science Research Activities. In: *Proceedings of the 2006 Emerging Trends and Challenges in Information Technology Management* Bd. 1 and 2. Washington DC, USA, 2006

[Vol17] Volkwein, G.: *Industrie 4.0 from both a user's and a vendor's perspective*. International Conference on System-integrated Intelligence (SysInt2016), http://www.sysint-conference.org/uploads/media/SysInt2016_Keynote_Volkwein.pdf. [31.10.2017]

[Wei14] Weilkiens, T.: *Systems Engineering mit SysML/UML: Anforderungen, Analyse, Architektur*. 3. Aufl. dpunkt.verlag, 2014

[Wes99] Westfall, L.: 12 Steps to Useful Software Metrics. In: *Pacific Northwest Software Quality Conference (PNSQC)*, 1999

[WP10] Winkler, S. ; Pilgrim, J.: A survey of traceability in requirements engineering and model-driven development. In: *Software & Systems Modeling* 9 (2010), September, Nr. 4, S. 529–565

[ZVE70] ZVEI: *ZVEI-Kennzahlensystem: Ein Instrument zur Unternehmenssteuerung*. Zentralverband der Elektrotechnischen Industrie, 1970

Anhang A

Bedeutung der Werte zur Fehlerklassifizierung

Die folgenden Tabellen zeigen die Wertemengen der Aufzählungstypen, die beschreibende Informationen eines Fehlers, d.h. eines *Defect Work Items*, enthalten. Sie werden für die Qualitätsbewertung von Softwareversionen bzw. Softwareprodukten verwendet (vgl. Abschnitt 5.2.2). Diese Wertemengen sind aus der Literatur entnommen bzw. bauen darauf auf. Die jeweiligen Quellen sind in den Tabellenunterschriften angegeben.

Wert	**Beschreibung**
Critical	Die Anwendung oder wesentliche Funktionen der Anwendung sind nicht mehr verfügbar bzw. nutzbar.
Major	Eine wesentliche Funktion ist nicht mehr verfügbar oder liefert nicht die richtigen Ergebnisse, aber es gibt einen Workaround.
Neutral	Voreingestellter Standardwert: Fehler wurde noch nicht klassifiziert.
Minor	Eine nicht wesentliche Funktion ist nicht mehr verfügbar oder liefert nicht die richtigen Ergebnisse.
Trivial	Kleinerer Fehler, der die Verwendung der Anwendung nicht wesentlich beeinträchtigt.

Tabelle A.1: Wertemenge des Aufzählungstyps *DefSeverity* angelehnt an [PP11]

Wert	Beschreibung
Algorithm, method	An error in the sequence or set of steps used to solve a particular problem or computation, including mistakes in computations, incorrect implementation of algorithms, or calls to an inappropriate function for the algorithm being implemented.
Assignment, initialization	A variable or data item that is assigned a value incorrectly or is not initialized properly or where the initialization scenario is mishandled (e.g., incorrect publish or subscribe, incorrect opening of file etc.).
Checking	Inadequate checking for potential error conditions, or an inappropriate response is specified for error conditions.
Data	Error in specifying or manipulating data items, incorrectly defined data structure, pointer or memory allocation errors, or incorrect type conversions.
External interface	Errors in the user interface (including usability problems) or the interfaces with other systems.
Internal interface	Errors in the interfaces between system components, including mismatched calling sequences and incorrect opening, reading, writing or closing of files and databases.
Logic	Incorrect logical conditions on if, case or loop blocks, including incorrect boundary conditions („off by one“ errors are an example) being applied, or incorrect expression (e.g., incorrect use of parentheses in a mathematical expression).
Non-functional	Includes non-compliance with standards, failure to meet non-functional requirements such as portability and performance constraints, and lack of clarity of the design or code to the reader - both in the comments and the code itself.
Timing, optimization	Errors that will cause timing (e.g., potential race conditions) or performance problems (e.g., unnecessarily slow implementation of an algorithm).
Other	Anything that does not fit any of the above categories that is logged during an inspection of a design artifact or source code.

Tabelle A.2: Wertemenge des Aufzählungstyps *DefInternal* [SSR+08]

Wert	**Beschreibung**
Documentation	Incomplete instructions, mismatch between function and documentation, missing documentation
Functionality	Expected functionality is not met, functionality only works partially
Handling	Not comfortable, easy to make an user error, unclear how to do, unnecessary information
Optic	Spelling errors, incomplete sentences in a dialog, general appearance, missing translation
Performance	Time behavior too slow, resource allocation too high, resources needed too high
Stability	Program crash, data loss, data corruption, exception messages

Tabelle A.3: Wertemenge des Aufzählungstyps *DefExternal* angelehnt an [ISO10]

Anhang B

Praxisnahe Anwendung

Die folgenden Tabellen zeigen die prozentuale Verteilung der Werte in den einen Fehler klassifizierden Attributen. Diese Werte wurden exemplarisch während der praxisnahen Anwendung eingetragen. Wie in Abschnitt 7.2 erläutert, erfolgte die Klassifizierung durch den Verfasser der Arbeit. Es handelt sich also nicht um die Einschätzung eines Kunden oder eines Produktverantwortlichen.

Wert	**Version 1**	**Version 2**	**Version 3**
Critical [%]	20	50	40
Major [%]	60	0	30
Minor [%]	20	0	30
Trivial [%]	0	50	0

Tabelle B.1: Prozentuale Teilwerte des Attributs *severity*

Wert	**Version 1**	**Version 2**	**Version 3**
Algorithm, method [%]	0	0	20
Assignment, initialization [%]	0	50	30
Checking [%]	0	0	30
Data [%]	0	0	0
External interface [%]	20	0	0
Internal interface [%]	40	0	0
Logic [%]	40	0	10
Non-functional [%]	0	50	10
Timing, optimization [%]	0	0	0
Other [%]	0	0	0

Tabelle B.2: Prozentuale Teilwerte des Attributs *internal*

Wert	**Version 1**	**Version 2**	**Version 3**
Documentation [%]	0	0	0
Functionality [%]	40	50	70
Handling [%]	0	0	0
Optic [%]	0	50	10
Performance [%]	0	0	10
Stability [%]	60	0	10

Tabelle B.3: Prozentuale Teilwerte des Attributs *external*