



Stefan Erlachner, BSc

Einsatz von  
Entwicklungs- und Testdaten  
zur Bewertung automotiver Software

Masterarbeit  
zur Erlangung des akademischen Grades  
Diplom-Ingenieur

Fakultät für Informatik und Biomedizinische Technik

Technische Universität Graz

Beurteiler:

Univ.-Prof. Dipl.-Ing. Dr.techn. Franz Wotawa  
Institut für Softwaretechnologie

Betreuer:

Dipl.-Ing. Markus Ernst MLBT  
Dipl.-Ing. Dr.techn. Jürgen Fabian  
Institut für Fahrzeugtechnik

Betreuerin MAGNA Powertrain GmbH & Co KG:

Dipl.-Ing. Irenka Mandic

Graz, Dezember 2015

# Danksagung

Die Durchführung und Verfassung einer Masterarbeit ist ein langer und anstrengender Prozess, den man selten alleine bewältigen kann. Ich möchte deshalb an dieser Stelle die Gelegenheit nutzen um mich bei den Menschen zu bedanken, die zum Gelingen dieser Arbeit beigetragen haben.

Allen voran bei Herrn Andreas Münzer für die Ermöglichung dieser Arbeit in Kooperation mit MAGNA Powertrain. Bei Frau Irenka Mandic für die Projektleitung, Betreuung und Koordination, sowie bei Herrn Christian Sinn und Herrn Amir Tojaga für die wertvollen Anmerkungen und Beiträge.

Von der TU Graz bedanke ich mich recht herzlich bei Herrn Prof. Wotawa für die Begutachtung und fachliche Betreuung der Arbeit, sowie bei meinen beiden Betreuern Herrn Ernst und Herrn Fabian, die mich während des gesamten Prozesses begleitet haben. Besonderen Dank möchte ich hierbei nochmal an Herrn Ernst für die vielen produktiven Diskussionen und die fruchtbringende Zusammenarbeit aussprechen.

Aus meinem privaten Umfeld bedanke ich mich recht herzlich bei meinen Studienkollegen, die über die Jahre zu echten Freunden geworden sind. Sowie bei meinen Bundesbrüdern von der K.Ö.St.V. Babenberg Graz für die vielen heiteren, und manchmal auch ernsten, Gespräche. Ich bedanke mich außerdem bei meiner Familie, die mir während meines Studiums immer vollste Unterstützung und bedingungsloses Vertrauen entgegengebracht hat. Meinen letzten Dank an dieser Stelle richte ich an meine Freundin Julia, die mich auch ertragen hat wenn es mal nicht so gut gelaufen ist. Vielen Dank für dein Verständnis und deine Unterstützung während dieser arbeitsintensiven Zeit.

„Danke an alle Beteiligten!“

*Stefan Erlachner*

# Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtliche und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am .....

(Unterschrift)

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from used sources.

.....

(Date)

(Signature)

# Abstract

During the development of automotive software a huge amount of data is generated to ensure the correct flow of the process. However, this data is often not investigated further or used for other intentions. Thus, the present thesis is devoted to the use of development and testing data for ongoing evaluation of automotive software in the development process. For this purpose, the theoretical foundations of the development process and the quality assurance, as well as the use of software metrics to quantify automotive software are presented from the literature. After that, the development process of an automotive multinational supplier will be analyzed and the relevant development and test data will be identified. These are used to define an ideal concept that focuses on the allocation of the development and testing data on levels of the V-model, the evaluation of chronological sequences and the use of software metrics to quantify automotive software. This ideal concept is then analyzed based on the available data to check feasibility. Therefore, an analysis of the development and testing data, from a sample project, is performed, which serves as a basis for reviewing the ideal concept. The main parts are individually examined and a feasible concept is worked out for them. For the development of evaluation methods, concrete issues from interest groups are collected to ensure the relevance of the methods. Based on that evaluation methods are specified for the assessment of automotive software. At the end a prototype is illustrated to show a practical implementation of the use of development and test data for evaluation purposes.

# Kurzfassung

Während der Entwicklung automotiver Software wird eine Vielzahl an Daten generiert, die den reibungslosen Ablauf des Prozesses sicherstellen. Jedoch werden diese Daten oft nicht weiter untersucht oder für andere Zwecke verwendet. Die vorliegende Diplomarbeit widmet sich deshalb dem Einsatz von Entwicklungs- und Testdaten zur laufenden Bewertung automotiver Software im Entwicklungsprozess. Zu diesem Zweck werden die theoretischen Grundlagen des Entwicklungsprozesses und der Qualitätssicherung, sowie der Einsatz von Software-Metriken zur Quantifizierung automotiver Software aus der Literatur angeführt. Danach wird der Entwicklungsprozess eines international tätigen Automobilzulieferers analysiert und die relevanten Entwicklungs- und Testdaten identifiziert. Diese werden zur Definition eines Idealkonzeptes herangezogen wo im Speziellen auf die Zuordnung der Entwicklungs- und Testdaten zu den Ebenen des V-Modells, die Auswertung zeitlicher Verläufe und den Einsatz von Software-Metriken zur Quantifizierung automotiver Software eingegangen wird. Dieses Idealkonzept wird daraufhin auf die Machbarkeit überprüft. Zu diesem Zweck wird zuerst eine Analyse der Entwicklungs- und Testdaten, anhand eines Beispielprojektes, durchgeführt, die als Grundlage für die weitere Überprüfung dient. Daraufhin werden die Teile des Idealkonzeptes einzeln überprüft und ein umsetzbares Konzept für diese ausgearbeitet. Um daraus Analysemethoden zu entwickeln werden konkrete Fragestellungen einzelner Interessensgruppen gesammelt um die Relevanz der Methoden zu gewährleisten. Auf dieser Grundlage werden im Folgenden Analysemethoden zur Bewertung der automotiven Software spezifiziert. Eine praktische Umsetzung der entwickelten Analysemethoden wird am Ende anhand eines entwickelten Prototyps veranschaulicht.

# Abkürzungen

ALM	Application Lifecycle Management
AUTOSAR	Automotive Open System Architecture
CCB	Change Control Board
CI	Change Issue
CLI	Command Line Interface
CLS	Central Locking System
ECU	Electronic Control Unit
E/E	Electrical/Electrical
GQM	Goal-Question-Metrics
HiL	Hardware-in-the-Loop
ISO	International Standard Organisation
LOC	Lines of Code
MiL	Model-in-the-Loop
MPT	MAGNA Powertrain
MR	Module Requirement
NIST	National Institute of Standards and Technology
OEM	Original Equipment Manufacturer
OS	Operating System
PC	Personal Computer
RCP	Rapid Control Prototyping
RTE	Runtime Environment
SCR	System Component Requirement
SiL	Software-in-the-Loop
TC	Test Case
XML	Extended Markup Language

# Inhaltsverzeichnis

Danksagung .....	i
Eidesstattliche Erklärung .....	ii
Abstract .....	iii
Kurzfassung .....	iv
Abkürzungen .....	v
Inhaltsverzeichnis .....	vi
1 Einleitung.....	1
1.1 Zielsetzung der Arbeit .....	1
1.2 Gliederung der Arbeit.....	2
2 Theoretische Grundlagen zur automotiven Software.....	3
2.1 Begriffsbestimmung und Klassifizierung automotiver Software.....	3
2.1.1 Klassifizierung von automotiver Software .....	4
2.2 Softwareentwicklung im Kraftfahrzeug.....	5
2.2.1 Grundsätzliche Entwicklungskonzepte.....	6
2.2.2 Das V-Modell .....	9
2.2.3 Testprozess des V-Modells.....	12
2.2.4 Qualitätssicherung in der Softwareentwicklung .....	16
2.3 Software-Metriken .....	17
2.3.1 Dimensionen von Software .....	19
2.3.2 Code-Metriken.....	22
2.3.3 Prozess-Metriken.....	27
2.3.4 Methoden zur Auswahl geeigneter Metriken .....	32
2.3.5 Aggregation und Repräsentation von Software-Metriken.....	33
3 Analyse des Entwicklungsprozesses .....	37
3.1 Entwicklungsprozess für automotive Software.....	37
3.1.1 Anforderungen (SCR/MR).....	38
3.1.2 Testfälle (TC).....	39
3.1.3 Arbeitspakete, Änderungen und Fehler (CI).....	40
4 Analysemethoden und deren Machbarkeit auf Basis von Entwicklungs- und Testdaten.....	42
4.1 Idealkonzept .....	42

4.1.1	Zuordnung der Entwicklungs- und Testdaten zu Ebenen des V-Modells.....	42
4.1.2	Auswertung zeitlicher Verläufe der Entwicklungs- und Testdaten .....	42
4.1.3	Einsatz von Software-Metriken zur Quantifizierung automotiver Software.....	43
4.2	Überprüfung der Machbarkeit des Idealkonzeptes aufgrund vorhandener Entwicklungs- und Testdaten .....	44
4.2.1	Entwicklungs- und Testdatenanalyse .....	44
4.2.2	Machbarkeitsanalyse der Zuordnung zu den Entwicklungs-Ebenen.....	45
4.2.3	Machbarkeitsanalyse der Auswertung zeitlicher Verläufe .....	46
4.2.4	Machbarkeitsanalyse zum Einsatz von Software-Metriken .....	47
4.3	Detaillierte Spezifizierung der Analysemethoden aufgrund relevanter Fragestellungen im Entwicklungsprozess automotiver Software .....	50
4.3.1	Analysemethoden für das Projektmanagement .....	51
4.3.2	Analysemethoden für die Qualitätssicherung.....	53
4.3.3	Analysemethoden für das Management.....	58
5	Realisierung und Implementierung des Prototypen .....	62
5.1	Grundlegender Aufbau des Prototypen .....	62
5.2	Analyse-Tool .....	64
5.3	Datenexport aus dem MKS in die Analysedatenbank .....	69
5.4	Programmablauf zum Erstellen der Auswertungen .....	71
6	Zusammenfassung.....	73
6.1	Ausblick.....	73
	Abbildungsverzeichnis.....	I
	Tabellenverzeichnis .....	III
	Literaturverzeichnis.....	IV



# 1 Einleitung

Mechatronische Komponenten und Systeme durchdringen heutzutage das gesamte Fahrzeug. Mechatronik steht dabei für die gemeinsame Nutzung von Mechanik und Elektronik, wobei die Elektronik weiter in Soft- und Hardware unterteilt werden kann. Angefangen bei der Motorsteuerung über Fahrwerk und Getriebe bis hin zu Kommunikations- und Informationssystemen, spielen diese Komponenten eine immer entscheidendere Rolle, [1]. Insbesondere die automotive Software bietet neue Lösungswege. Mithilfe von Software können bekannte und neue Funktionen billiger, leichter und in höherer Qualität umgesetzt werden. Sei es nun die Optimierung des Kraftstoffverbrauchs oder das Verbessern des Fahrgefühls durch Fahrwerks- und Getriebesteuerungen, [2]. Automotive Software spielt in allen Bereichen eine immer größere Rolle. Dementsprechend entfallen heutzutage bereits 40% der Produktionskosten auf die Hard- und Software im Fahrzeug, [3]. Dieser steigende Einsatz führt aber auch zu einer immer weiter steigenden Komplexität der Programme und Steuerungen. Waren Software-Lösungen früher voneinander unabhängig und auf ein Steuergerät beschränkt, sind diese heute mit einer Vielzahl von Sensoren, Aktoren und anderen Programmen vernetzt, [3]. Die steigende Komplexität erhöht dabei auch die Wahrscheinlichkeit von Softwarefehlern, was zu hohen Kosten, etwa durch Rückholaktionen, führen kann, [4]. Aus diesem Grund ist es wichtig die Qualität und den Zustand der automotiven Software während des gesamten Entwicklungsprozesses im Auge behalten zu können. Eine mögliche Datenquelle für die Kontrolle und Analyse der Softwareeigenschaften stellen die Entwicklungs- und Testdaten dar, die während des gesamten Entwicklungsprozesses mechatronischer Komponenten generiert werden. Durch eine Analyse des Informationssystems können relevante Daten identifiziert und zu Analysemethoden kombiniert werden um eine Evaluierung des Entwicklungs- und Testprozesses automotiver Software zu ermöglichen.

## 1.1 Zielsetzung der Arbeit

Ziel dieser Masterarbeit ist die Analyse von Entwicklungs- und Testdaten automotiver Software um deren Potentiale zur Bewertung automotiver Software zu ermitteln. Als Grundlage dienen die Daten aus einem „Application lifecycle management“-System eines weltweit tätigen Autozulieferers. Diese Daten sollen auf ihre Eignung zur Verwendung in erweiterten Analysemethoden überprüft werden. Aufgrund der verschiedenen Interessensgruppen bei der Entwicklung automotiver Software sollen unterschiedliche Perspektiven und Fragestellungen spezifiziert werden und mithilfe der ausgearbeiteten Analysemethoden beantwortete werden. Dabei sollen folgende Ziele verfolgt werden:

- Evaluierung des Anstiegs von Fehlern um so früh wie möglich im Entwicklungsprozess eine Detektion und Fehlerbehebung zu ermöglichen.
- Erfüllungsgrad der Anforderungen um die funktionale Vollständigkeit des Produktes zu gewährleisten.
- Anzahl der ausgeführten Testfälle um das korrekte Testen sicherzustellen.
- Vergleich von Software-Versionen in Bezug auf Größe, Komplexität und Qualität um Trends im Entwicklungsverlauf zu erkennen.

Auf Grundlage dieser Anforderungen sind Analysemethoden für die einzelnen Interessensgruppen zu entwickeln. Die Ergebnisse sollen in einem lauffähigen Prototyp für ein Beispielprojekt realisiert und zur Verfügung gestellt werden.

### **1.2 Gliederung der Arbeit**

Diese Masterarbeit gliedert sich in einen theoretischen und einen praktischen Teil. Im theoretischen Teil werden die Grundlagen über automotive Software und den Entwicklungsprozess, sowie die Qualitätssicherung und der Einsatz von Software-Metriken zur Quantifizierung behandelt. Der praktische Teil wird mit einer Analyse des Entwicklungsprozesses und der Identifikation der relevanten Daten eingeleitet. Auf dieser Grundlage wird ein Idealkonzept für den Einsatz der Entwicklungs- und Testdaten definiert und die Haupteinsatzmöglichkeiten spezifiziert:

- Zuordnung der Entwicklungs- und Testdaten zu Ebenen des V-Modells.
- Auswertung zeitlicher Verläufe der Entwicklungs- und Testdaten.
- Einsatz von Software-Metriken zur Quantifizierung der automotiven Software.

Auf Grundlage einer Analyse der vorhandenen Entwicklungs- und Testdaten wird dieses Idealkonzept daraufhin auf die Machbarkeit überprüft und umsetzbare Konzepte ausgearbeitet. Grundlage dafür ist die Analyse der Entwicklungs- und Testdaten anhand eines Beispielprojektes. Aus diesen Erkenntnissen werden danach die konkreten Analysemethoden spezifiziert. Um die Relevanz und die Einsetzbarkeit der Methoden zu gewährleisten wurden Fragenstellungen von drei unterschiedlichen Interessensgruppen gesammelt und auf dieser Grundlage die Analysemethoden für:

- das Projektmanagement,
- die Qualitätssicherung und
- das Management entwickelt.

Eine praktische Umsetzung dieser Analysemethoden in Form eines voll funktionsfähigen Prototyps im letzten Kapitel veranschaulicht. Den Abschluss bilden eine Zusammenfassung der Arbeit und ein Ausblick auf die mögliche Erweiterung des Prototyps und der Analysemethoden.

## 2 Theoretische Grundlagen zur automotiven Software

Software in heutigen Fahrzeugen beinhaltet mehrere Millionen Zeilen an Code und okkupiert mehrere hundert Megabytes an Speicher. Es wird erwartet, dass die nächsten Generationen von Fahrzeugen den Speicherbedarf von 1 GB, in Bezug auf Software, überschreiten werden. Dieser zunehmende Speicherverbrauch ist jedoch ein verhältnismäßig geringes Problem verglichen mit der ständig steigenden Komplexität. Dieser Anstieg kann auf die Heterogenität der Software im Kraftfahrzeugbereich und der zunehmenden Notwendigkeit der Vernetzung von Komponenten zurückgeführt werden. Durch die Weiterentwicklung der Verkabelung und der Entwicklung von verschiedenen Bussystemen (z.B. CAN, FlexRay, LIN) können Steuergeräte in einem Kraftfahrzeug untereinander kommunizieren. Ein modernes Kraftfahrzeug, der oberen Mittelklasse, enthält über 80 ECUs (Electronic Control Unit), die über ein komplexes Bussystem verbunden sind. Durch diesen enormen Anstieg der Interkonnektivität und Komplexität entstehen hohe Anforderungen an den Entwicklungs- und Testprozess von OEMs („Original Equipment Manufacturer“) und Zulieferern in der Automobilindustrie, [3], [5], [6].

Die zunehmende Bedeutung von automotiver Software bewirkt, dass bereits 80% der Innovationen im Automobilbereich auf die Software zurückzuführen sind. Diese können in die Bereiche: Komfort (z.B. Infotainment), Sicherheit (z.B. ABS), Antriebsstrang (z.B. Motorsteuerung) oder Umwelt (z.B. Treibstoffeinsparung) eingeteilt werden, [2], [5], [6]. Durch dieses breite Einsatzgebiet entwickelten sich verschiedene Arten von Softwaresystemen, die sich, je nach Anwendung, in Merkmalen und Anforderungen unterscheiden.

### 2.1 Begriffsbestimmung und Klassifizierung automotiver Software

Der Begriff der „automotiven Software“ ist ein Sammelbegriff für die verschiedenen Ausprägungen von Software, die aufgrund der vielschichtigen Aufgaben in einem Kraftfahrzeug zum Einsatz kommen. Obwohl sich diese Softwaresysteme aufgrund ihres Einsatzgebietes sehr voneinander unterscheiden, handelt es sich bei allen um Steuergerätesoftware. Diese unterscheidet sich in vielen Bereichen von herkömmlicher PC-Software. Der Ausdruck PC-Software bezeichnet Programme, die auf einem Computer mit Betriebssystem installiert und ausgeführt werden. Das Betriebssystem steuert und verwaltet hierbei die Ressourcen sowie die Interaktionen mit den einzelnen Hardwarekomponenten. Die PC- Software selbst erhält nur über Schnittstellen und Abstraktionsebenen indirekten Zugriff auf diese Komponenten. Bei dem Programmieren einer PC-Software muss deshalb die Hardware nicht direkt angesprochen werden, was durchaus Vorteile bietet. So können dadurch Fehlerquellen minimiert und die große Anzahl an verschiedenen Hardwareherstellern besser berücksichtigt werden. Ein negativer Aspekt ist die höhere Laufzeit und der höhere Speicherverbrauch, der durch das Betriebssystem entsteht, [7]. Der Begriff Steuergerätesoftware wird dagegen für Programme verwendet, die direkt auf einem Steuergerät (z.B. einem Mikrocontroller) ausgeführt werden. Diese Programme haben direkten Zugriff auf die Hardware des Steuergerätes und benötigen deshalb auch kein Betriebssystem. Die Aufgabe einer Steuergerätesoftware besteht, in der Regel, im Auslesen von Sensordaten, auf deren Basis neue Steuerungswerte berechnet werden. Mit diesen Steuerungswerten werden wiederum Aktoren (z.B. Ventile, Gebläse,

Kompressoren) angesteuert. Tabelle 1 bietet eine Zusammenfassung über die wichtigsten Unterschiede zwischen PC- und Steuergeräte-Software, [7].

Tabelle 1: Vergleich zwischen PC-Software und Steuergeräte-Software, [7].

PC- Software	Steuergeräte- Software
Keine Echtzeitanforderungen (Aufgaben werden in undefinierter Zeit beendet.)	Echtzeitanforderung (Aufgaben müssen in definierter Zeit beendet werden.)
Mehrere Aufgaben werden gleichzeitig ausgeführt (Multitasking).	Aufgaben werden nacheinander in Endlosschleife ständig wiederholt (statisches Scheduling).
Ressourcen stehen in großen Mengen zur Verfügung.	Stark limitierte Ressourcen der Recheneinheit (ECU)
Wenige externe Schnittstellen.	Viele externe Schnittstellen, Reaktion auf äußere Ereignisse (Sensoren, Aktoren).
Interaktion mit BenutzerInnen über Peripheriegeräte (z.B. Maus und Tastatur).	Kein direkter Zugriff des Benutzers, wenn dann nur über Hilfsmittel.

### 2.1.1 Klassifizierung von automotiver Software

Komponenten in Kraftfahrzeuge müssen sehr unterschiedliche Aufgaben erfüllen und stellen somit stark abweichende Anforderungen an die Steuerungssoftware. Zum Beispiel muss die Software für die Getriebesteuerung höhere Echtzeit-Anforderungen erfüllen als die Software des eingebauten Multimediasystems. Durch diese hohe Heterogenität haben sich verschiedene Kategorien von automotiver Software entwickelt, [2], [4].

Die grundlegendste Unterscheidung kann aufgrund der E/E Systeme getroffen werden, [8]:

1. Kontroll- und Steuerungssystem für die grundlegenden Fahrzeugfunktionen (z.B. Motorsteuerung)
2. Komfortsysteme für die Fahrzeuginsassen (z.B. Klimaanlage)
3. Informationssysteme, inklusive Navigation und Infotainment (z.B. GPRS, Multimedia)

Diese erste grobe Klassifizierung kann noch weiter verfeinert werden, [2], [5]:

Tabelle 2: Klassifizierung von automotiver Software, [2], [5]

Kategorie	Merkmale
Infotainment und Mensch-Maschine-Kommunikation	<ul style="list-style-type: none"> <li>• niedrige Echtzeitanforderungen</li> <li>• Schnittstellen zu externen IT-Systemen (Smartphone, Bluetooth, etc.) und Fahrzeuginsassen</li> <li>• ereignisorientierte Funktionsabläufe</li> </ul>
Komfortsysteme	<ul style="list-style-type: none"> <li>• niedrige Echtzeitanforderungen</li> <li>• ereignisorientierte Funktionsabläufe</li> </ul>
Sicherheitssysteme	<ul style="list-style-type: none"> <li>• hohe Echtzeitanforderungen</li> <li>• ereignisorientierte Funktionsabläufe</li> <li>• hohe Sicherheitsanforderungen</li> </ul>
Getriebe- und Karosseriesteuerung	<ul style="list-style-type: none"> <li>• hohe Echtzeitanforderungen</li> <li>• Kontrollalgorithmen überwiegen gegenüber ereignisorientierten Abläufen</li> </ul>

	<ul style="list-style-type: none"> <li>• hohe Verfügbarkeitsanforderungen</li> </ul>
Wartungssysteme (z.B. Diagnose, Software Updates)	<ul style="list-style-type: none"> <li>• je nach Anwendungsgebiet hohe oder niedrige Echtzeitanforderungen</li> <li>• ereignisorientierte Abläufe</li> </ul>

Wie in Tabelle 2 ersichtlich sind in Bereichen, in denen eine Interaktion mit einem menschlichen Benutzer stattfindet, die Echtzeitanforderungen irrelevant. Bei Sicherheitssystemen beziehungsweise Getriebe- und Karosseriesteuerungen muss hingegen schnell auf sich ändernde Einflussgrößen reagiert werden können (z.B. Öffnen des Airbags bei einem Aufprall). Ein weiteres Unterscheidungsmerkmal kann aufgrund der Arbeitsweise getroffen werden. Eine ereignisorientierte Software reagiert auf das Eintreten festgelegter Ereignisse, wie zum Beispiel das Drücken des „Play“-Buttons des Radios. Wohingegen die Getriebe- Steuerungssoftware die Werte eines Sensors ständig überprüfen und die entsprechenden Signale an die Aktoren weiterleiten muss. Deshalb werden hier Kontrollalgorithmen mit hohen Echtzeitanforderungen verwendet um Veränderungen, der Sensorwerte, schnell registrieren zu können, [2].

## 2.2 Softwareentwicklung im Kraftfahrzeug

Die Automobilindustrie ist stark vertikal organisiert und in den letzten 100 Jahren wurde viel Zeit investiert um einzelne Sub-Systeme so unabhängig wie möglich entwickeln zu können. In der Software-Entwicklung wird diese Vorgehensweise modulare Entwicklung genannt. Diese Bemühungen führten zu einer höchst erfolgreichen Arbeitsteilung bei der Entwicklung eines Kraftfahrzeuges. Geschätzt 25% des Gesamtwertes des Fahrzeuges wird vom OEM selbst erbracht. Dabei konzentriert sich dessen Arbeit oft auf die Motorenentwicklung, die Integration der einzelnen Komponenten und das Marketing der Marke. Das bedeutet, dass bis zu 75% des restlichen Wertes von Zulieferern generiert werden. Dies betrifft dementsprechend auch die nötige Software zur Steuerung der Komponenten. Durch die angesprochene Konzentration auf die Unabhängigkeit der einzelnen Sub-Komponenten können Zulieferer große Teile des Fahrzeuges eigenständig entwickeln und fertigen. Für die Zulieferer können sich daraus Synergieeffekte ergeben indem für mehrere OEMs ähnliche Systeme entwickelt werden, [5]. Ausgehend von dieser Überlegung haben sich verschiedene Entwicklungskonzepte etabliert um diese Synergieeffekte effizient nutzen zu können (siehe Kapitel 2.2.1). Diese unabhängige Entwicklung der Zulieferer setzt allerdings auch große Anforderungen an den Entwicklungs- und Testprozess des Sub-Systems voraus. Um die Vorgehensweise der einzelnen Entwicklungsprojekte zu standardisieren wurde ein spezielles Vorgehensmodell entwickelt. Das V-Modell standardisiert die Vorgehensweisen, die dazugehörigen Ergebnisse und die Transparenz des Projektes zwischen OEM und Zulieferer. Durch diese Standardisierung des Entwicklungsprozesses wird eine einheitliche Zusammenarbeit des OEMs mit den einzelnen Zulieferern möglich (siehe Kapitel 2.2.2), [9]. Gleichzeitig definiert das V-Modell ein Mindestmaß an Verifizierungs- und Validierungsmaßnahmen und bestimmt somit zu großen Teilen den Testprozess von allen beteiligten Zulieferern (siehe Kapitel 2.2.3). Abhängig von der Ebene im V-Modell wird zwischen Komponententests, Integrationstests, Systemtests und Akzeptanztests unterschieden. Diese vorgegebene Teststrategie hilft dabei eine einheitliche Qualität und Korrektheit der Sub-Systeme über alle Zulieferer hinweg zu gewährleisten, [10].

### 2.2.1 Grundsätzliche Entwicklungskonzepte

Seit 1996 haben sich verschiedene Konzepte zur Entwicklung von automotiver Software entwickelt. Diese unterscheiden sich in erster Linie durch den Betrachtungsbereich, der in die Entwicklung miteinbezogen wird, [8].

Ein wichtiger Bereich ist dabei die Wiederverwendbarkeit von Software. Hierzu wird diese in einzelne, voneinander unabhängige, Komponenten zerlegt und diese danach getrennt voneinander entwickelt. Nach der Implementation und dem Testen können diese in anderen Projekten wiederverwendet werden. Dementsprechend wird dieses Konzept als „komponentenbasierter Entwicklung“ bezeichnet. Durch diese Herangehensweise ergeben sich zwei Phasen in der Entwicklung. Zuerst werden alle Komponenten einzeln entwickelt und getestet und in der zweiten Phase zu einer Gesamtsoftware kombiniert. Dieser zweite Schritt wird als Integration bezeichnet und setzt dabei eigene Integrationstests voraus, um das korrekte Zusammenspiel der einzelnen Komponenten zu gewährleisten, [8].

Ein weiteres Konzept ist die „modellbasierte Entwicklung“. Dabei wird die Software auf einer höheren Abstraktionsebene erstellt und der Code danach automatisch generiert, [8]. Eine Möglichkeit die Software nach diesem Konzept zu entwickeln, ist die Modellierung mithilfe eines Signalflussplans. Entwicklungswerkzeuge, die eine solche Modellierung zur Verfügung stellen, sind beispielsweise „Simulink“ [11] „Easy5“ [12] oder „ASCET“ [13]. Die Modellierung der Software mithilfe eines Signalflussplans bietet den Vorteil das entwickelte Modell gleichzeitig als Dokumentation der Software zu verwenden. Einige Werkzeuge bieten des Weiteren die Möglichkeit aus der Modellierung direkt Programmcode für Mikrocontroller zu generieren. Diese automatische Codegenerierung reduziert einerseits den Aufwand für die zeit- und kostenintensive Programmierung, und bietet andererseits auch die Möglichkeit die Software sehr früh in der Implementierungsphase zu testen. Diese Teststrategie wird als „Rapid Control Prototyping“ (RCP) bezeichnet. Als Nachteil der automatischen Codegenerierung können die schlechte Lesbarkeit des generierten Codes und die Abhängigkeit von bestimmten Werkzeughersteller, aufgrund fehlender Standards, erwähnt werden, [7].

Zusammenfassend können folgende Vorteile der modellbasierten Entwicklung festgehalten werden, [10]:

- Die abstrakte Modellierung erleichtert das Entwickeln eines gemeinsamen Problem- und Lösungsverständnisses.
- Ist das spezifizierte Modell eindeutig, so kann das System in einer Simulation ausgeführt werden und das bereits zu einem frühen Zeitpunkt in der Entwicklung. Diese Methode des frühzeitigen Testens wird als „Rapid Prototyping“ bezeichnet.
- Mithilfe von „Rapid Prototyping“ und sogenannten Laborfahrzeugen kann die Umgebung des Steuergerätes frühzeitig im Labor getestet werden. Vorteil von Laborfahrzeuge, gegenüber Testfahrzeugen und Prüfständen, ist die höhere Flexibilität und Reproduzierbarkeit der Testfälle.
- Mithilfe geeigneter Werkzeuge kann der Programmcode automatisch aus der Modellierung erstellt werden. Dies setzt verschiedene nicht funktionale Designinformationen, wie etwa Optimierungsoptionen, voraus. Im Gegenzug reduziert sich der zeitaufwändige und fehleranfällige Prozess, den Programmcode manuell zu schreiben.

Die „plattformbasierte Entwicklung“ reagiert auf die immer stärkere Vernetzung von Fahrzeugkomponenten. Das Grundprinzip dieses Konzeptes ist, dass die gesamte Software so entwickelt wird, als würde sie auf einem einzelnen Steuergerät laufen. Auf diese Art und Weise wird ein Bewusstsein für die Interkonnektivität und die Vernetzung der einzelnen Fahrzeugkomponenten geschaffen. Was wiederum zu einer Vereinfachung der Schnittstellen und Kommunikation zwischen den Komponenten führen soll, [8].

Alle diese Entwicklungskonzepte dienen zu einem großen Teil dazu das Entwicklungsrisiko zu reduzieren. Ein Entwicklungsrisiko ist ein Ereignis, das den Verlauf des Projektes beziehungsweise der Entwicklung maßgeblich negativ beeinflussen kann. Dementsprechend ist eine Minimierung der Auswirkungen und das generelle Reduzieren der Wahrscheinlichkeit des Entwicklungsrisikos ein wichtiger Punkt für OEMs und Zulieferer gleichermaßen. Grundsätzlich gibt es zwei Möglichkeiten das Entwicklungsrisiko zu minimieren, [10]:

- Durch frühzeitiges Testen der Software-Funktionen und
- durch die Wiederverwendung von bereits getesteten Software-Funktionen.

Durch das stetige Ansteigen der Kosten für das Finden und Korrigieren von Fehlern, im Laufe des Entwicklungsprozesses, ist es wichtig Fehler so früh wie möglich zu finden und zu beheben, [14]. Die modellbasierte Entwicklung unterstützt die frühzeitige Validierung, indem sie Simulationen und Rapid-Prototyping ermöglicht. Das frühzeitige Testen beugt außerdem kostenintensive Iterationen in der Implementation vor, indem grundsätzliche Designfehler vorab gefunden werden können, [10].

Eine andere Möglichkeit zur Reduzierung des Entwicklungsrisikos ist die Wiederverwendung von bereits fertig entwickelten Software-Funktionen. Dazu müssen Komponenten und Module so unabhängig wie möglich voneinander entwickelt werden und nur über zuvor festgelegte Schnittstellen miteinander kommunizieren. Aufgrund der Vielzahl an unterschiedlichen Steuergeräten ist auch eine Standardisierung der Schnittstellen, die beispielsweise konform zu AUTOSAR [15] ist, sinnvoll. AUTOSAR ist eine Entwicklungspartnerschaft von Automobilherstellern, um auf die steigende Komplexität von E/E-Systemen durch die Entwicklung von Standards und Konzepten zu reagieren. Nur wenn Schnittstellen standardisiert sind kann eine effiziente Wiederverwendung von Komponenten in einem anderen Steuergerät stattfinden. Die Wiederverwendung von bereits getesteten Komponenten und Modulen führt zur Reduktion von Fehlern und beschleunigt die Entwicklung entscheidend, [10].

### **Architektur-Standardisierungen von automotiver Software**

Die Wiederverwendung von Softwarekomponenten ist ein geeigneter Weg, um Zeit und Geld in der Entwicklung zu sparen und gleichzeitig das Entwicklungsrisiko zu minimieren. Jedoch ist es durch die Vielzahl an verschiedenen Steuergeräten nicht immer möglich Komponenten zu übernehmen. Stattdessen müssen diese mit viel Aufwand an das neue Zielsteuergerät angepasst werden, was wiederum viele potentielle Fehlerquellen beinhaltet. Deshalb versucht man mithilfe von Standardisierungen eine Abhilfe zu schaffen, indem die Grundstruktur aller Steuergeräte vereinheitlicht wird. Die Einführung solcher Standards erleichtert die Wiederverwendung von Softwarekomponenten erheblich und ist die Voraussetzung für den flächendeckend erfolgreichen Einsatz der komponentenbasierten Entwicklung, [10].

Beispiele für solche Architektur Standardisierungen sind AUTOSAR [15] oder JASPAR [16]. Beiden gemein ist, dass sie die Software in zwei grundsätzliche Kategorien unterteilen, in die Anwendungs-Software und die Basis-Software. Die Basis-Software enthält Funktionen zur Steuerung, Regelung und Überwachung der Hardwarekomponenten und ist demnach auf jeden Mikrocontroller zugeschnitten. In der Anwendungs-Software werden die Funktionen implementiert, die die speziellen Anforderungen des Kunden abdecken. Des Weiteren sollte die Anwendungs-Software keine Abhängigkeiten zur Hardware enthalten, dies erleichtert später die Portierung auf ein anderes Steuergerät. Die einzelnen Funktionen der Basis- und Anwendungs-Software kommunizieren lediglich über festgelegte standardisierte Schnittstellen. AUTOSAR beinhaltet zudem eine eigene Architekturschicht für die Kommunikation zwischen den beiden Softwarekategorien. Diese Schicht wird als „AUTOSAR Runtime Environment“ (RTE) bezeichnet und übernimmt die gesamte Kommunikation zwischen der Basis- und der Anwendungs-Software. Dadurch muss bei der Entwicklung der verschiedenen Schichten nur auf eine konforme Implementierung zu der RTE geachtet werden. Das führt dazu, dass Basis- und Anwendungs-Software vollkommen unabhängig voneinander entwickelt werden können, [10].

## AUTOSAR

AUTOSAR ist die Abkürzung für „Automotive Open System Architecture“ und stammt aus einer Entwicklungspartnerschaft von Automobilherstellern aus 2003 um auf die steigende Komplexität von E/E-Systemen zu reagieren, [6]. Die grundlegende Idee von AUTOSAR ist die logische Aufteilung der verschiedenen Softwarekomponenten in die steuengerätespezifische Basis-Software (Basic Software, BSW) und die steuengeräteunabhängige Anwendungs-Software (AWS), die über eine Kommunikationsschicht, die sogenannte „Runtime Environment (RTE)“, kommunizieren (siehe Abbildung 1), [1].

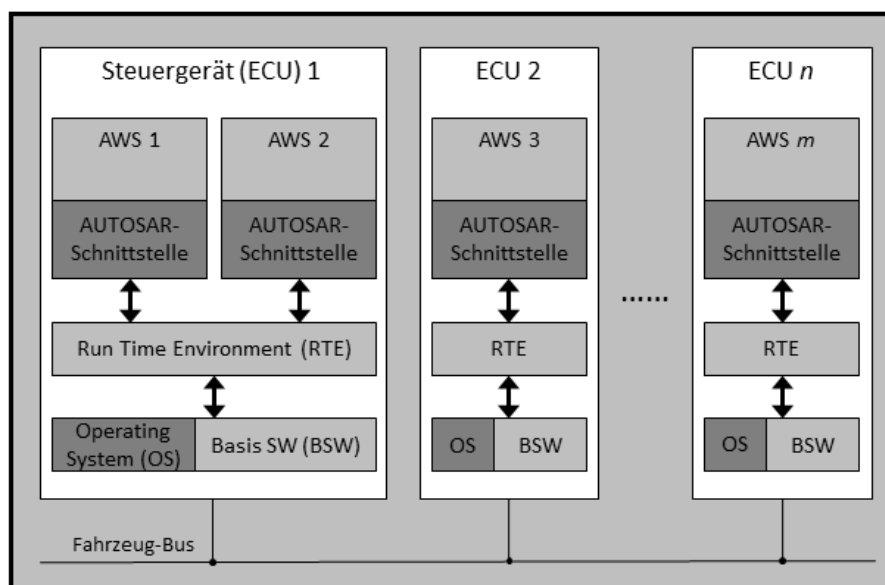


Abbildung 1: AUTOSAR-Architektur, vgl. [1]

Die **Anwendungs-Software (ASW)** enthält die funktionalen Softwarekomponenten, die zur Realisierung der Anforderungen nötig sind. Dabei ist die Anwendungs-Software strikt von der Basis-Software getrennt und kommuniziert nur über eine AUTOSAR-Schnittstelle mit der RTE. Diese Schnittstellen bestehen aus einer Beschreibung im XML (Extended Markup Language) Format, die



Informationen über die Art des Inhalts der Nachrichten enthält. Dadurch, dass jede einzelne Softwarekomponente nur direkt mit der RTE kommuniziert, ist es möglich, die einzelnen Komponenten vollkommen unabhängig von der anderen Anwendungs-Software und der hardwareabhängigen Basis-Software zu entwickeln, [1].

Die „**Runtime Environment**“ (**RTE**) ist der einzige Kommunikationskanal zwischen Anwendungs- und Basis-Software. Sie bildet somit eine Abstraktionsebene und erlaubt die unabhängige Entwicklung beider Teile. Dadurch, dass die RTE mit beiden Softwareebenen kommuniziert, ist sie gleichzeitig auf das verwendete Steuergerät und die spezifische Anwendungs-Software abgestimmt. Im Falle einer Portierung der Anwendungs-Software auf ein anderes Steuergerät muss nur die RTE angepasst werden. Meistens kann die RTE dabei aus den Schnittstellenanforderungen automatisch generiert werden, [1].

Unter der **Basis-Software (BSW)** versteht man die steuergerätespezifische, hardwarenahe, Software zum Betrieb und zur Steuerung des eigentlichen Steuergerätes. In ihr befinden sich Softwarekomponenten zur Ansteuerung der jeweiligen Hardware. Die Aufgaben umfassen das Speichermanagement, die Kommunikationsschnittstellen und die Diagnose. Die Basis-Software ist auch eng mit dem Betriebssystem („Operating System“ OS) des Steuergerätes verbunden. Das Betriebssystem steuert die grundlegenden Abläufe des Steuergerätes und sorgt unter anderem für eine optimale Ressourcenverteilung, [1].

### 2.2.2 Das V-Modell

Das V-Modell ist ein Vorgehensmodell zum Planen und Durchführen von IT-Projekten. Die erste Fassung wurde 1992 von der deutschen Bundeswehr veröffentlicht und 1997 zu dem „V-Modell 97“ überarbeitet. In Zusammenarbeit des Bundesministeriums für Verteidigung (BMVg), des Bundesamtes für Informationsmanagement und Informationstechnik der Bundeswehr (IT-AmtBw) und des Bundesministeriums für Inneres, Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (BMI-KBSt), wurde das „V-Modell 97“ als Entwicklungsstandard für IT-Systeme des Bundes im zivilen und militärischen Bereich festgelegt. Als Reaktion auf neue Methoden und Technologien wurde das V-Modell 97 später überarbeitet und 2005 als „V-Modell XT“ veröffentlicht. „XT“ steht hierbei für „eXtreme Tailoring“ und soll die flexible Anpassbarkeit an verschiedene Projektumfelder unterstreichen. Das V-Modell XT ist auch heutzutage aktuell und wird ständig weiterentwickelt. Hauptziel ist durch standardisierte Vorgehensprozesse die Projekttransparenz und das Management von Projekten zu verbessern und die Erfolgswahrscheinlichkeit zu erhöhen. Dementsprechend sollen V-Modell-konforme Projekte folgende Zielsetzungen verfolgen, [9], [17]:

- Minimierung der Projektrisiken,
- Gewährleistung und Verbesserung der Qualität,
- Eindämmung der Gesamtkosten und
- Verbesserung der Kommunikation.

Die Minimierung der Projektrisiken entsteht durch die Verwendung von standardisierten Vorgehensweisen. Dadurch werden die Transparenz und die Planbarkeit von Projekten erhöht. Zudem beschreibt das V-Modell zu jeder Vorgehensweise die zugehörigen Ergebnisse. Durch diese vordefinierten Zwischenergebnisse wird ebenfalls sichergestellt, dass die zu liefernden Ergebnisse vollständig und in der gewünschten Qualität vorliegen. Auf diese Weise ist es möglich Abweichungen

in der Qualität bereits früh zu erkennen was zur Gewährleistung und Verbesserung der Qualität beiträgt. Aufgrund der standardisierten Vorgehensmodelle lässt sich außerdem der Aufwand für das Gesamtprojekt leichter abschätzen. Die einheitlich erzeugten Ergebnisse verringern des Weiteren die Abhängigkeit zwischen Auftraggeber und Auftragnehmer und erleichtern so Folgeprojekte. Was in weiterer Folge zur Eindämmung der Gesamtkosten dient. Durch eine standardisierte, einheitliche Beschreibung aller im Projekt relevanten Bestandteile kann zudem die Kommunikation zwischen allen Beteiligten erleichtert und Missverständnisse vermieden werden, [9].

Im Allgemeinen regelt das V-Modell „Wer“, „Wann“, „Was“ zu tun hat. Weil aber nicht alle Projekte nach dem gleichen Schema ablaufen, ist es notwendig eine Klassifizierung der Projekte nach charakteristischen Merkmalen vorzunehmen. Dabei wird das Projekt anhand der Projektrolle der ausführenden Organisation und des angestrebten Projektgegenstandes klassifiziert (siehe Abbildung 2). Als Projektrolle kann die Organisation als **Auftraggeber (AG)**, **Auftragnehmer (AN)** oder gleichzeitig als **Auftraggeber und Auftragnehmer (AG/AN)** auftreten. Dementsprechend ergibt sich dann als Projekttyp eine **Systementwicklung (AG)**, bei welcher die Organisation einen Systementwicklungsauftrag an einen oder mehrere Auftragnehmer erteilt oder eine **Systementwicklung (AN)**, bei dem die Organisation einen Systementwicklungsauftrag einer anderen Organisation annimmt. Eine **Systementwicklung (AG/AN)** liegt dann vor, wenn das Projekt innerhalb der eigenen Organisation durchgeführt wird und die ausführende Organisation somit gleichzeitig als AG und AN auftritt. Mit der Zuteilung des Projektes zu einem Projekttyp wird eine spezifizierte Sichtweise auf das Projekt festgelegt. Um den Handlungsrahmen des Projektes noch genauer eingrenzen zu können, unterscheidet das V-Modell auch noch zwischen verschiedenen Arten des Projektgegenstandes. Projektgegenstände können die Entwicklung eines Hardware-Systems (**HW-System**), eines Software-Systems (**SW-System**), eines eingebetteten beziehungsweise komplexen Hardware und SW Systems (**HW- und SW-System / Eingebettetes System**) oder die Durchführung einer **Systemintegration** sein, [9].

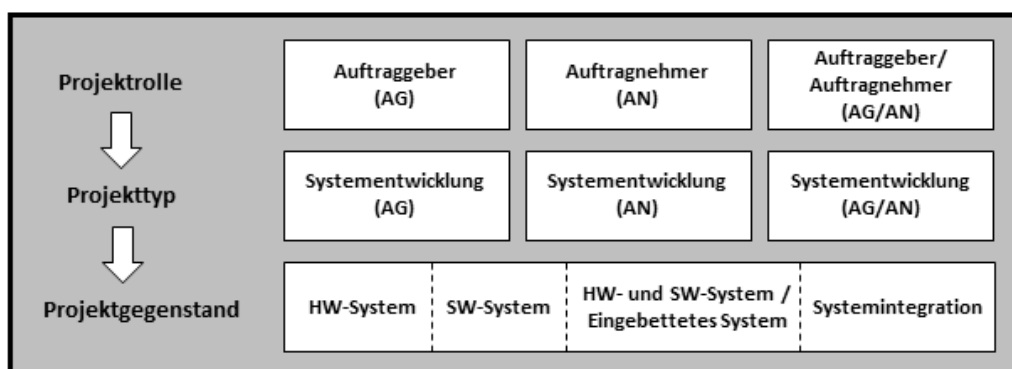


Abbildung 2: Klassifizierung von Projekten in Projekttypen, [9]

Für jeden Projekttypen existiert mindestens eine Projekttypvariante, mit der das Projekt näher charakterisiert werden kann. Mitunter können damit die Rahmenbedingungen und die möglichen Abläufe des Projektes bestimmt werden. Darüber hinaus bestimmt die Projekttypvariante genauer die zu verwendenden Vorgehensbausteine für das Projekt. **Vorgehensbausteine** beschreiben konkrete Aufgabenstellungen, die im Laufe eines V-Modell-Projektes erledigt werden müssen. Dabei legt ein Vorgehensbaustein alle Produkte, Aktivitäten und Rollen, die zur Erfüllung der Aufgabenstellung notwendig sind fest. Die Vorgehensbausteine lassen sich dabei grob in vier Kategorien unterteilen, [9]:

- V-Modell-Kern,
- Systementwicklung,
- Auftraggeber-/Auftragnehmer-Schnittstelle und
- Einführung und Pflege eines organisationspezifischen Vorgehensmodells.

Im **V-Modell-Kern** befinden sich alle Vorgehensbausteine, die für jeden Projekttyp und jede Projekttypvariante Anwendung finden. Darunter fallen zum Beispiel der Vorgehensbaustein für Projektmanagement oder der Vorgehensbaustein für Qualitätssicherung. Die Kategorie **Systementwicklung** beinhaltet Vorgehensbausteine, die für die Entwicklung eines Systems benötigt werden. Darunter fallen etwa Anforderungsfestlegung, Hardware-Entwicklung, Software-Entwicklung oder Sicherheit. Die **Auftraggeber-/Auftragnehmer-Schnittstelle** beschäftigt sich mit der Kommunikation zwischen Auftraggeber und Auftragnehmer und die letzte Kategorie listet die nötigen Vorgehensmodelle für die **Einführung und Pflege eines organisationspezifischen Vorgehensmodells** auf. Im weiteren Verlauf dieser Arbeit wird das Hauptaugenmerk auf die Aufgabenstellungen der Systementwicklung gelegt. Diese basiert auf der Idee, ein komplexes Problem solange zu zerlegen, bis die einzelnen Teilprobleme einzeln lösbar werden. Die aus dieser Zerlegung resultierenden Teilprobleme werden daraufhin einzeln gelöst und Schritt für Schritt zu einer Gesamtlösung zusammengeführt. Dieser Technik, des Zerlegens des Problems auf der einen Seite (linker Ast) und des anschließenden Zusammensetzens der Lösung auf der anderen Seite (rechter Ast), verdankt das V-Modell seinen Namen (siehe Abbildung 3). Das wichtigste Merkmal der Systementwicklung im V-Modell ist, dass die Verifizierungs- und Validierungsmethoden für die Realisierung und Integration bereits bei der Spezifikation festgelegt werden. Dadurch werden diese aufgrund der Anforderungen und nicht anhand der Realisierungen erstellt, was ein zielorientierteres Vorgehen in der Entwicklung sicherstellt. Zu diesem Zweck ist jede Spezifikationsebene über die Verifizierungs- und Validierungslinie mit einer Realisierungs- bzw. Integrationsebene verbunden. So kann das integrierte System gegen das entworfene System getestet werden, [9]. Da das V-Modell speziell für eingebettete Systeme mit hohen Zuverlässigkeits- und Sicherheitsanforderungen entwickelt wurde, wird es sehr oft in der Automobilindustrie eingesetzt. Neben den eingebauten Qualitätsprüfungen, durch vorgeschriebene Prüfschritte, ist auch die Berücksichtigung der verteilten Entwicklung von Komponenten eine wichtige Anforderung in der Automobilentwicklung. Eine Schwäche des V-Modells ist jedoch die fehlende Rückkopplung zu vorangegangenen Schritten auf der Spezifikations- und Realisierungsseite. Das bedeutet, dass das V-Modell implizit davon ausgeht, dass alle Benutzeranforderungen bereits zu Beginn vollständig zur Verfügung stehen und somit alle Systemteile hinreichend spezifiziert werden können. Weil dies aber in der Realität selten der Fall ist und um den Aufwand und die Kosten für spätere Änderungen gering zu halten, wird das V-Modell in der Praxis mehrmals durchlaufen. Dabei wird in jeder Iteration ein Prototyp entwickelt, getestet und in einer realen Umgebung erprobt. Dieser erprobte Prototyp dient daraufhin als Ausgangsbasis für den nächsten Iterationsschritt. Dies wird solange wiederholt bis in einer finalen Iteration alle Benutzeranforderungen erfüllt wurden. Abhängig von der Reife und der Qualität der Prototypen werden diese in Simulationen, Experimentalfahrzeugen, Fahrzeugprototypen, Vorserien- oder Serienfahrzeugen getestet und erprobt, [10].

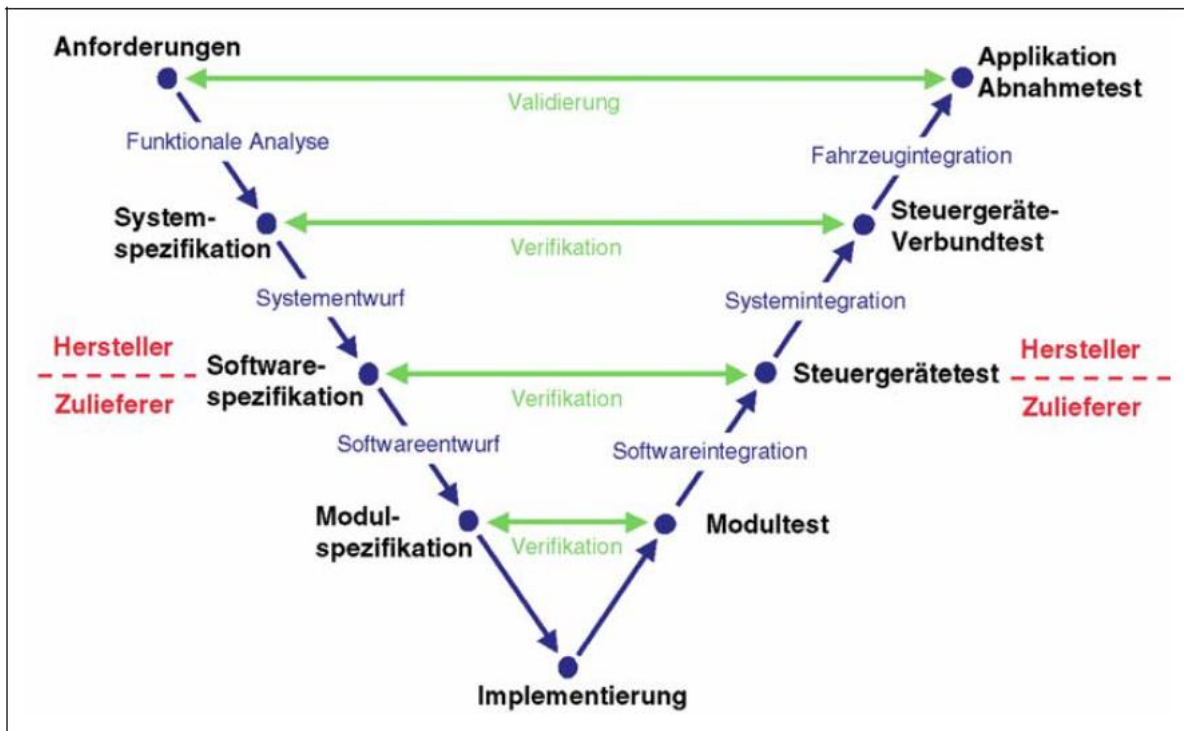


Abbildung 3: V-Modell des Software-Entwicklungsprozesses für Automobil-Anwendungen, [18]

Abbildung 3 zeigt die adaptierte Systementwicklung eines V-Modells für die Entwicklung elektronischer Systeme. Es wird dabei zwischen der Ebene der Systementwicklung (Hersteller) und der Ebene der Software-Entwicklung (Zulieferer) unterschieden. Während sich die Systementwicklungsebene (Hersteller) auf die Architektur und das korrekte Zusammenspiel von Steuergeräten und anderen E/E Komponenten konzentriert, steht auf der Software-Entwicklungsebene (Zulieferer) rein die Software im Vordergrund. Als Schnittstelle zwischen System- und Software-Ebene, auf dem linken Ast, dient die Softwarespezifikation. Diese legt die Aufgaben und Operationen der Software fest. Ausgehend von dieser Information erfolgen zu Beginn der Software-Entwicklungsebene die Analyse der Software-Anforderungen und die Spezifikation der Software-Architektur. Im nächsten Schritt wird, gemäß dem V-Modell, die Software-Architektur in immer kleinere Software-Komponenten zerlegt, bis diese unabhängig voneinander implementiert werden können. Wie beim originalen V-Modell werden auch hier, für jede Zerlegungsstufe, die dazugehörigen Validierungs- und Verifizierungsmethoden spezifiziert. Diese werden später, beim Erreichen der dazugehörigen Integrationsebene, ausgeführt. Dadurch wird die Implementierung beziehungsweise die Integration gegen die dazugehörige Spezifikation validiert. Die Schnittstelle zu der Systemebene, auf der Integrationsseite, bilden die korrekten Integrationstests (Steuergerätestests) des gesamten Software-Systems. Auf Basis dieser Software-Tests kann mit den Integrationstests der E/E-Komponenten und des Gesamtsystems fortgefahren werden. Am Ende des V-Modells, beziehungsweise einer Iteration, steht die Kalibrierung und das Ausführen der System- und Akzeptanztests, [10].

### 2.2.3 Testprozess des V-Modells

Nach dem V-Modell besteht der Testprozess aus der Verifikation und der Validierung. In der Verifikation wird überprüft ob die realisierte Implementierung der dazugehörigen Spezifikation entspricht. Während die Validierung überprüft, ob die ausgearbeitete Spezifikation die realen

Benutzeranforderungen erfüllt. Idealerweise sollte dieser Schritt dementsprechend unmittelbar nach der Spezifikationserstellung und noch vor der Implementierung erfolgen. Die Validierung beschäftigt sich also mit der Korrektheit der Anforderungen und die Verifikation mit der Korrektheit der Implementierung. Abhängig von der Position innerhalb des V-Modells, werden verschiedene Testarten unterschieden, [7]:

- Modultest,
- Integrationstest,
- Systemtest,
- Akzeptanztest.

Modul-, Integrations- und Systemtests testen die Software gegen die Spezifikation und gehören deshalb zu den Verifizierungsmethoden. Die Akzeptanztests testen die Software direkt gegen die Benutzeranforderungen und werden deshalb zu den Validierungsmethoden gezählt, [7], [10], [14]. Bevor die gesamte Software getestet werden kann, muss sichergestellt werden, dass alle einzelnen Software-Module die Spezifikation erfüllen. Die dazu notwendigen Tests werden als **Modultests** bezeichnet. Dabei gestaltet sich der Test eines Moduls einer Steuerungssoftware komplizierter, als der einer PC-Software. Grund dafür ist, dass die Module auf den Original-Steuergeräten getestet werden müssen und diese meistens keine direkten Anzeige- und Eingabemöglichkeiten bieten. Deshalb werden hier meist sogenannte „Monitore“ eingesetzt, um den Programmablauf und Variablenbelegungen abzufragen. Ein „Monitor“ ist im einfachsten Fall eine Software, die zusätzlich im Steuergerät vorhanden ist und über eine serielle Schnittstelle das Debuggen des Codes ermöglicht. Modultests werden in den meisten Fällen als Whitebox-Tests durchgeführt. Das bedeutet, dass dem Tester der dokumentierte Quellcode während des Testens zur Verfügung steht, [14]. Aufgabe des Testers ist es, einen möglichst großen Bereich des Quellcodes durch Testfälle abzudecken. Der Grad der Abdeckung des Quellcodes durch Testfälle wird als Testabdeckung bezeichnet. Eine vollständige Testabdeckung verlangt allerdings nicht nur die Einbindung jeder einzelnen Codezeile in die Tests, sondern auch den Test aller möglichen Kombinationsvarianten von Verzweigungen (wie z.B. „if-Anweisungen“, Schleifen). Das Ziel einer vollständigen Testabdeckung erzeugt deshalb sehr schnell einen großen Testaufwand. Aus diesem Grund wird durch erprobte Testmethoden, zum Beispiel das Testen von Grenzwerten und kritischen Pfaden, versucht, einen Kompromiss aus Testabdeckung und Testaufwand zu erreichen. Für einen vollständigen Test eines Moduls werden auch oft Daten benötigt, die von anderen Modulen geliefert werden. Stehen diese Module während des Testens nicht zur Verfügung, werden diese oft durch sogenannte Testtreiber simuliert. Ein Testtreiber ist ein Programm das dieselben Schnittstellen wie das zu simulierende Modul besitzt und sinnvolle Testdaten für die Tests liefert. Dadurch können Module unabhängig von anderen Modulen getestet werden, [7].

Wurden alle Module getestet und erfüllen die Spezifikation, ist der nächste Schritt die verschiedenen Module zu einer gesamten Software zusammenzuführen. Dieser Schritt, bei dem mehrere Software-Module, die oft auch von verschiedenen Herstellern entwickelt wurden, zusammengeführt werden, wird Integration genannt. Dementsprechend spricht man bei den mit der Integration einhergehenden Tests von **Integrationstests**. Die Integration kann entweder inkrementell erfolgen, indem immer mehr Testtreiber durch reale Module ersetzt werden, oder in einem einzigen großen Schritt, bei dem alle Module gleichzeitig integriert werden. Ziel der Integrationstests ist es, die korrekte Funktionsweise der Schnittstellen und modulübergreifenden Funktionen zu überprüfen.

Dabei werden die Tests, je nachdem, ob der Quellcode zur Verfügung steht, als Whitebox- oder Blackbox-Test durchgeführt. Bei einem Blackbox-Test wird nur die Funktionsweise des zu testenden Moduls überprüft, ohne den Quellcode heranzuziehen. Bei Modulen, die von anderen Herstellern stammen, ist dies die gängige Methode, weil der Quellcode in den meisten Fällen nicht zur Verfügung steht. In der Praxis treten während der Integration besonders viele Probleme auf, bei den meisten handelt es sich allerdings lediglich um Fehler, die auf eine abweichende Realisierung der Schnittstelle zwischen zwei Modulen zurückzuführen sind. Seltener sind Fehler, bei denen sich zwei Module in ihrer Ausführung direkt beeinflussen. Etwa wenn ein Modul einen Speicherbereich eines anderen Moduls überschreibt, ohne, dass dies erkannt wird. Solche Fehler sind auf einen fehlerhaften Umgang mit Zeigern zurückzuführen und sehr schwer zu lokalisieren,[7], [10].

Nach der Integration, wird das gesamte Software-System gegen die Spezifikation überprüft und die dazugehörigen Tests werden als **Systemtests** bezeichnet. Während bei Modul- und Integrationstests noch der Quellcode in Form von Whitebox-Tests miteinbezogen werden kann, werden die Systemtests als reine Blackbox-Tests ausgeführt. Das bedeutet, dass die Testenden keinen Einblick mehr auf den Quellcode nehmen können. Vielmehr ist für die Testenden hier entscheidend, ob das System das spezifizierte Verhalten aufweist. Dementsprechend muss überprüft werden, ob das System auf eine bestimmte Eingabe das erwartete Ergebnis liefert. Unter dem Begriff Systemtests ist eine Vielzahl an Teiltests mit unterschiedlichen Zielsetzungen zusammengefasst. Den größten Teil davon bilden die **funktionalen Tests**, die das Verhalten des Systems aufgrund unterschiedlicher Eingabeparameter überprüfen und die Ergebnisse mit der Spezifikation vergleichen. Weitere Teiltests sind etwa **Robustheitstests**, bei denen die Auslastung der Ressourcen und die Echtzeitanforderungen überprüft werden oder **Recovery-Tests**, die das korrekte Verhalten des Systems nach einer Störung testen. Der **Dauertest** überprüft ob bei längerer Laufzeit keine Fehlerspeicher oder andere Arten von Speicher überlaufen. Daneben existieren noch Konfigurations-, Kompatibilitäts-, Usability- und Sicherheitstests sowie Benchmarks, die alle anderen Bereiche der Spezifikation abdecken und prüfen, [7].

Alle bis jetzt erwähnten Tests befassen sich mit der Überprüfung der Software in Bezug auf die Spezifikation und zählen somit zu den Verifizierungsmethoden. Der letzte Testschritt einer Software bilden die **Akzeptanztests**. Dabei wird die gesamte Software auf die Benutzeranforderungen überprüft. In den meisten Fällen finden diese in enger Zusammenarbeit mit den Kunden statt und ähneln den bereits zuvor durchgeführten Systemtests. Eine besondere Ausprägung des Akzeptanztests ist der gefürchtete „Management Drive“, in dem ranghohe Vertreter/innen eines Automobilkonzerns, zum Beispiel Vorstandsmitglieder, noch während der Entwicklungsphase das Fahrzeug Probe fahren. Obwohl den meisten Beteiligten dabei bewusst ist, dass es sich bei dem gefahrenen Fahrzeug noch nicht um den Serienstand handelt, kann ein fehlerhaftes Verhalten eines Steuergerätes den guten Ruf eines Unternehmens nachhaltig schädigen, [7].

### Simulation

Um die Qualität und die Sicherheit der Software gewährleisten zu können, ist es wichtig die einzelnen Module und die Integration so bald wie möglich zu testen. Da aber oft nicht alle Module genau zur selben Zeit fertig gestellt sind, oder vielleicht sogar von anderen Unternehmen entwickelt werden, können noch nicht realisierten Komponenten, mithilfe von Modellierung und Simulation, als virtuelle Komponenten im Computer nachgebildet werden. Der Einsatz solcher virtueller Komponenten ist vor allem in der Automobilindustrie, in der viele verschiedene Hersteller Komponenten liefern,

notwendig. Neben dem Ermöglichen von frühen Tests bieten virtuelle Komponenten aber noch weitere positive Vorteile. So können auftretende Fehler mithilfe von virtuellen Komponenten leichter reproduziert, und Extremsituationen ohne eine Gefährdung von Mensch und Material simuliert werden. Ein weiterer positiver Aspekt ist, dass die Tests vollautomatisiert ablaufen können und günstiger in der Durchführung sind als ein Versuchsfahrzeug. Es ist jedoch immer zu berücksichtigen, dass Modellierungen und Simulationen immer unvollständig sind. Diese Tatsache beruht darauf, dass nur bestimmte Szenarien einer Komponente nachgebildet werden können. Es besteht deshalb immer ein Restrisiko durch Situationen und Szenarien die nicht berücksichtigt wurden. Daraus können sich, in der realen Betriebsumgebung, Komplikationen ergeben an die zuvor nicht gedacht wurde. Aus diesem Grund sind umfassende System- und Akzeptanztests mit allen realen Komponenten in einer realen Betriebsumgebung unerlässlich, [1], [7], [10].

Werden Tests in einer simulierten Umgebung ausgeführt, spricht man von sogenannten In-the-Loop-Tests. Dabei wird, abhängig davon, welche Komponenten in welchem Umfang real in den Test eingebunden werden, zwischen verschiedenen In-the-Loop-Testsystemen unterschieden (siehe Abbildung 4), [1]:

- a. Model-in-the-Loop (MiL),
- b. Software-in-the-Loop (SiL),
- c. Function-in-the-Loop (FiL) und
- d. Hardware-in-the-Loop (HiL).

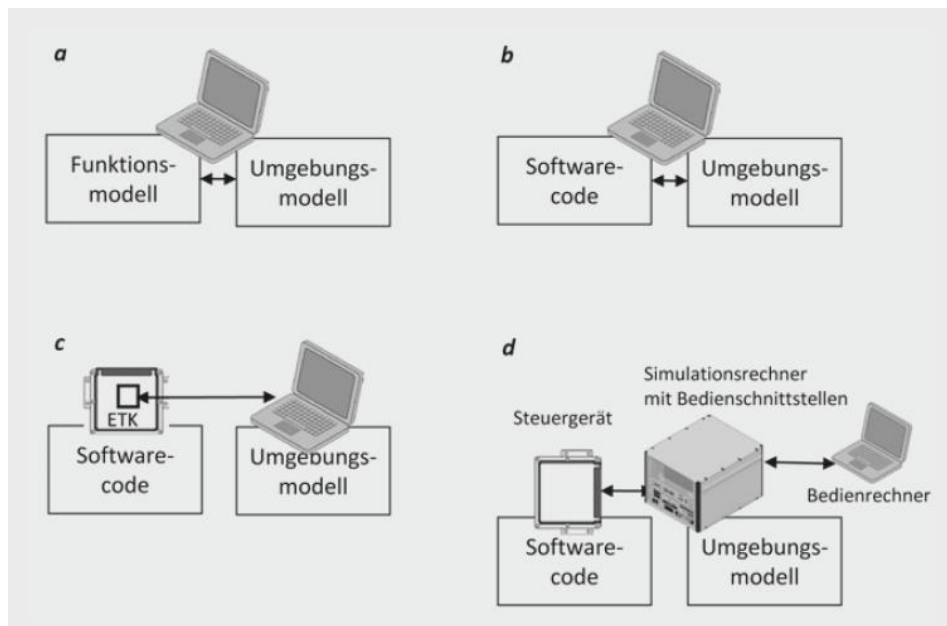


Abbildung 4: Klassifizierung von In-the-Loop-Testsystemen, [1]

**Model-in-the-Loop (MiL)** kommt in der modellbasierten Entwicklung zum Einsatz, indem ein mit einem Entwicklungswerkzeug (zum Beispiel Simulink [11]) erstelltes Modell zusammen mit einer simulierten Umgebung getestet wird. Durch diese Methode können Fehler in einem frühen Stadium des Entwicklungsprozesses lokalisiert und behoben werden, noch bevor die eigentliche Implementierung stattgefunden hat. Sobald das Modell vollständig und verifiziert ist, kann mit der Implementierung der Funktionen und Methoden begonnen werden. Viele modellbasierte Entwicklungswerkzeuge bieten hier die Möglichkeit den Quellcode direkt aus dem Model

automatisch generieren zu lassen. Diese Implementierungen, egal ob automatisch generiert oder von Hand geschrieben, werden danach wieder mit einer simulierten Testumgebung getestet. Diese Phase wird als **Software-in-the-Loop (SiL)** bezeichnet. Der einzige große Unterschied zwischen MiL- und SiL-Tests liegt darin, dass das Modul beim SiL-Test bereits in einer Programmiersprache vorliegt, zum Beispiel „C“. Nachdem die einzelnen SW-Module zu einer Gesamtsoftware integriert wurden, kann diese auf dem dazugehörigen Steuergerät getestet werden. Weil bei diesen Tests zum ersten Mal die Zielhardware miteinbezogen wird, wird diese Phase als **Hardware-in-the-Loop (HiL)** bezeichnet. Dabei wird das Steuergerät mit der Software an einen Simulationsrechner mit Bedienschnittstelle angeschlossen, der wiederum über einen Bedienrechner gesteuert wird. Mit diesen In-the-Loop-Testsystemen kann zu jedem Zeitpunkt im Entwicklungsprozess eine bestmögliche Testbarkeit der einzelnen Komponenten erzielt werden, bis am Ende das Gesamtsystem unter Realbedingungen validiert werden kann, [1], [10], [19], [20].

#### 2.2.4 Qualitätssicherung in der Softwareentwicklung

Im Vergleich zu der Mechanik und Elektrik spielt die Qualitätssicherung der Software in der Entwicklung mechatronischer Komponenten oftmals noch eine untergeordnete Rolle. Grund hierfür ist vermutlich, dass die Software in der Fertigung von E/E-Komponenten keine Kosten verursacht, weil sie sich relativ einfach reproduzieren lässt. Deshalb beschränkt man sich häufig einfach auf die richtige Funktionsweise des Gesamtsystems und die Komplexitätsbeherrschung während der Entwicklung, [1]. Es existieren aber auch Ansätze, die aussagen, dass es effizienter ist von Anfang an Wert auf fehlerfreie, qualitativ hochwertige Software zu legen als erst später auf Fehler zu reagieren, wenn es vielleicht schon zu spät ist. Demnach ist die heutige Herausforderung in der Softwareentwicklung dem Kunden Software in hoher Qualität pünktlich zu liefern. Dabei existieren viele verschiedene Definitionen was unter der Qualität eines Software-Produktes verstanden wird. Eine Definition misst beispielsweise die Qualität daran, inwieweit die Anforderungen an die Software umgesetzt wurden. Dabei wird allerdings vernachlässigt, ob die Anforderungen für das Produkt überhaupt geeignet sind und in welcher Güte diese Anforderungen umgesetzt wurden. Nach dieser Definition wären beispielsweise ein teures Luxusfahrzeug und ein Mittelklassewagen von derselben Qualität, vorausgesetzt beide würden im selben Umfang die eigenen Anforderungen umsetzen. Deshalb betrachtet eine weitere Definition die Software anhand der Eigenschaften, die für den Gebrauch notwendig sind. Obwohl diese Betrachtungsweise unpassende Anforderungen außer Acht lässt, gibt es wiederum keine Möglichkeit zwei Produkte zu vergleichen, die über dieselben Eigenschaften für den Gebrauch verfügen. Anhand dieser zwei Definitionen wird das Problem veranschaulicht, dass Qualität nicht nur anhand der umgesetzten Anforderungen bewertet werden kann. Die ISO 9126 [21] definiert deshalb sechs Qualitätskriterien, die für die Qualität der Software ausschlaggebend sind (siehe Tabelle 3). Je besser eine Software diese Kriterien erfüllt, desto höher ist auch die Qualität der Software aus der Sicht des Kunden, [21].

Tabelle 3: ISO 9126 – Software Qualitätskriterien, vgl. [21]

Kriterium	Beschreibung
Funktionalität	Inwieweit erfüllt die Software die definierten Anforderungen?
Zuverlässigkeit	Kann die Software die benötigten Leistungen, unter bestimmten Bedingungen, liefern?
Benutzbarkeit	Wie leicht fällt es einem Benutzer mit dem Programm zu arbeiten und welchen Aufwand erfordert es bis ein Benutzer mit dem Programm arbeiten kann?
Effizienz	Das Verhältnis zwischen dem Leistungsniveau und den eingesetzten



	Betriebsmitteln?
Wartbarkeit	Welchen Aufwand verursachen Änderungen (Korrekturen, Verbesserungen) an der Software?
Portierbarkeit	Welcher Aufwand ist notwendig um die Software auf eine andere Plattform zu übertragen?

Die Qualitätskriterien der ISO 9126 zeigen sehr gut die natürliche Vielschichtigkeit der Qualität. So ist es aus Kosten- und Zeitgründen so gut wie nie möglich alle Kriterien gleichermaßen zu erfüllen. Das magische Dreieck der Qualität stellt die Wechselwirkung zwischen Qualität, Kosten und Termin (Zeit) grafisch dar (siehe Abbildung 5). Wird zum Beispiel eine höhere Qualität angestrebt, müssen sich gleichzeitig entweder die Kosten (z.B. mehr Mitarbeiter/innen) erhöhen oder der Termin verschieben. Genauso verhält es sich, wenn die Kosten reduziert werden müssen. Dies kann entweder erreicht werden, indem die Qualität vernachlässigt wird (z.B. weniger Testen) oder aber wiederum durch die Verschiebung des Endtermins. Analog kann diese Wechselwirkung auch für den Termin veranschaulicht werden, [7].

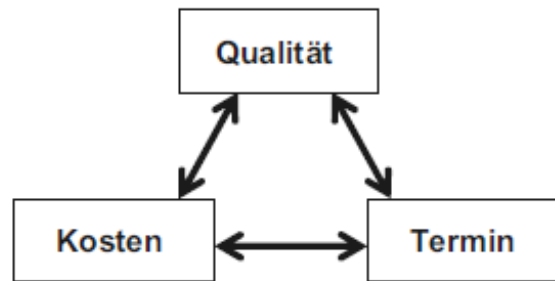


Abbildung 5: Magisches Dreieck der Qualität, [7]

Um auf den Kosten- und Zeitdruck in der Entwicklung zu reagieren, ist es wichtig für jedes Produkt ein eigenes Qualitätsmodell zu definieren, das die Kriterien hierarchisch nach Wichtigkeit für den Anwendungsfall ordnet. So kann beispielsweise bei einer Steuergerätesoftware die Benutzbarkeit größtenteils vernachlässigt werden, während der Zuverlässigkeit eine besonders große Bedeutung zukommt. Mit diesem definierten Qualitätsmodell sollte daraufhin die Qualität des Produkts während des gesamten Entwicklungsprozesses kontrolliert werden, [22], [23].

### 2.3 Software-Metriken

Die zuvor diskutierten Sicherheits- und Qualitätsanforderungen, die an Produkt und Prozess gestellt werden, müssen quantitativ messbar gemacht werden. Ein Ansatz dies zu erreichen besteht in der Softwaremessung und dem Einführen von verschiedenen Metriken. Im Allgemeinen bezeichnet der Begriff „Messung“ den Prozess Eigenschaften einer Entität durch Zahlen oder Symbole nach definierten Regeln zu beschreiben, [21], [24]. Dementsprechend ist eine Softwaremessung der andauernde Prozess Daten über den Softwareentwicklungsprozess oder des Produkts zu definieren, zu sammeln und zu analysieren. Dies dient dem übergeordneten Ziel, den Prozess oder das Produkt besser zu verstehen und kontinuierlich zu verbessern, [25], [26]. Weitere Vorteile bringt die Softwaremessung, indem es Prozesse und Produkte messbar und somit vergleichbar macht. Dadurch ist es möglich Vergangenheitswerte für die Abschätzung zukünftiger Projekte zu ermitteln und die Kommunikation zwischen allen Beteiligten durch konkrete Messwerte zu verbessern.

Zusammenfassend kann der Zweck der Softwaremessung durch fünf Punkte beschrieben werden, [27]:

- Verbesserung des Verständnisses für Prozesse und Produkte,
- Möglichkeit Prozesse und Produkte zu vergleichen,
- Sammeln von Daten zur Abschätzung zukünftiger Prozesse und Produkte,
- Grundlage zur Steuerung und Verbesserung von Prozessen und Produkten,
- Verbesserung der zwischenmenschlichen Verständigung mithilfe von konkreten Werten.

Die Werte, die nach der Analyse der gesammelten Daten zur Verfügung stehen, werden als Software-Metriken bezeichnet. Je nachdem, welchen Zweck eine Metrik erfüllen soll, werden die Daten aus der Anforderungs-, Design oder Implementierungsphase gewonnen. Zum Beispiel können Metriken aus der Designphase Aufschluss über den späteren Implementierungs- und Testaufwand geben oder es werden Metriken aus dem Code für die Abschätzung der Anzahl der Fehler herangezogen, [28]. Es gibt eine Vielzahl von Ansätzen, Software-Metriken zu kategorisieren. Ein paar beziehen sich auf die Eigenschaften der Metrik selbst, andere darauf wie sie berechnet werden. Beispiele für Metrik-Typen sind, [25]:

- Projekt- versus Prozess- versus Produkt-Metriken
- Objektive versus Subjektive Metriken
- Direkte versus Indirekte Metriken
- Basis versus Abgeleitete Metriken
- Absolute versus Relative Metriken
- Dynamische versus Statische Metriken
- Vorhersagende versus Erläuternde Metriken

### **Projekt- versus Prozess- versus Produkt-Metriken**

Projekt-Metriken bestehen aus den Eigenschaften des Projektes, wie die Anzahl der Entwickler oder die Kosten und Terminplanung und dienen dazu, das Projekt als Ganzes messbar zu machen, [25], [29], [30].

Prozess-Metriken messen Eigenschaften betreffend des Softwareentwicklungsprozesses. Mögliche Metriken sind etwa die Anzahl der gefundenen Fehler in einer Entwicklungsphase, die Effizienz beim Entfernen von Fehlern oder die Anzahl der umgesetzten Anforderungen in einem Release, [25], [29], [30].

Produkt-Metriken befassen sich mit dem Zwischen- beziehungsweise Endprodukt. Dementsprechend enthalten diese Metriken Informationen über den Aufbau und die Eigenschaften der Software selbst, wie die Größe, Komplexität oder Qualität, [25], [29], [30].

### **Objektive versus Subjektive Metriken**

Objektive Metriken sind absolute Werte, die direkt gemessen werden können und einen grundlegenden Startpunkt (natürlichen Nullpunkt) aufweisen. Sie können dadurch objektiv aus dem Prozess oder Produkt gewonnen werden. Vertreter dieser Metriken sind die Anzahl der Codezeilen oder die Anzahl der gefundenen Fehler, [25].

Subjektive Metriken besitzen keinen grundlegenden Startpunkt und benötigen deshalb eine subjektive Einschätzung durch einen Menschen. Beispiele für solche Metriken sind die Qualität oder Komplexität einer Software, [25].

### Direkte versus Indirekte Metriken

Direkte Metriken beziehen sich auf Eigenschaften, die unabhängig von anderen Eigenschaften sind. Deshalb können diese direkt aus dem Prozess oder dem Produkt gewonnen und verwendet werden. Die Anzahl der Fehler in einem Produkt oder die Anzahl der Module sind zum Beispiel direkte Metriken, [25].

Indirekte Metriken setzen sich aus mehreren abhängigen Werten zusammen, die alleine zu wenig Aussagekraft besitzen. Demnach sind indirekte Metriken immer eine Berechnung aus zwei oder mehreren Eigenschaften. Die Fehlerdichte besteht etwa aus der Anzahl der Fehler / Anzahl der Codezeilen, [25].

### 2.3.1 Dimensionen von Software

Eine weitere Möglichkeit Metriken zu klassifizieren besteht unter Zuhilfenahme der Dimensionen von Software. Eine Software ist ein multidimensionales Produkt und besteht aus vielen verschiedenen Dimensionen, von denen allerdings nur drei messbar sind (siehe Abbildung 6). Diese sind die Quantität (Größe), die Komplexität und die Qualität der Software. Dementsprechend können auch die Metriken in diese drei Dimensionen unterteilt werden, [27]:

- Quantitäts-Metriken,
- Komplexitäts-Metriken,
- Qualitäts-Metriken.

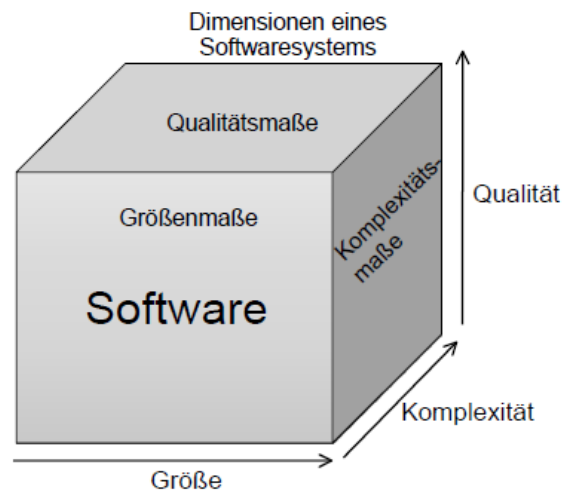


Abbildung 6: Drei Dimensionen von Software, [27]

**Quantitäts-Metriken** sind Mengenzahlen, die eine Aussage über den Umfang beziehungsweise die Größe der Software ermöglichen, zum Beispiel die Menge der Anforderungen oder die Menge aller Anweisungen in einer Code-Datei. Die Software besteht aus einer Ansammlung aufeinander gestapelter Sprachschichten, die gemessen werden können. Diese können in die Anforderungs-, Entwurfs- und Implementationsschicht unterteilt werden (siehe Abbildung 7). Zu diesen Schichten

befinden sich parallel dazu passende Tests zur Verifizierung und Validierung. Diese werden in Unit-Tests, Integrations-Tests und System-Tests unterteilt. Vervollständigt wird alles durch die Benutzerdokumentations-Säule, die dem Benutzer das Verhalten der Software erklärt. Jede einzelne Schicht, in den verschiedenen Säulen, beinhaltet andere Mengenzahlen, die gemessen und interpretiert werden können. So kann in der Benutzerdokumentations-Säule die Anzahl der Seiten des Handbuchs gemessen werden, während auf der Implementationsschicht der Test-Säule die Anzahl der Testfälle wichtige Informationen enthält. Zusammenfassend kann festgehalten werden, dass es notwendig ist aufgrund der Vielzahl an möglichen Mengenzahlen eine Auswahl zu treffen. Auch sind nicht in jedem Projekt alle Größenmaße ausreichend vorhanden. Bei der Messung eines Altsystems kann es etwa vorkommen, dass nur die Quantitäts-Metriken des Source-Codes zur Verfügung stehen, weil alle anderen Schichten entweder nicht erstellt oder über die Jahre verworfen wurden, [27].

Grundsätzlich können folgende Größenmaße gemessen werden, [27]:

- Größe der Anforderungsspezifikation,
- Größe des Systementwurfs,
- Größe des Source-Codes,
- Größe des Test-Codes,
- Größe der Benutzerdokumentation.

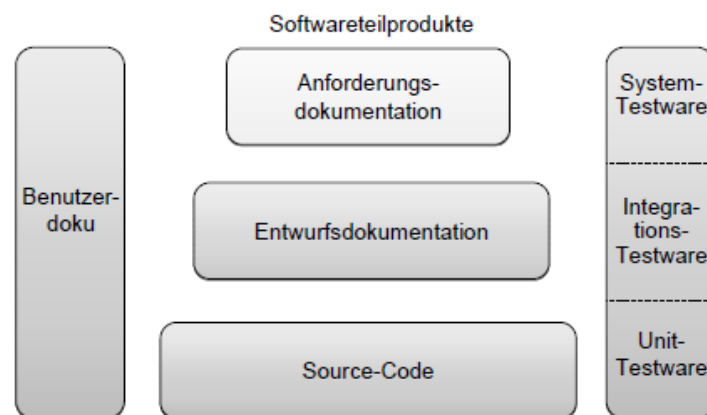


Abbildung 7: Quellen der Softwaregrößenmessung, [27]

**Komplexitäts-Metriken** geben Aufschluss darüber wie komplex beziehungsweise kompliziert eine Software ist. Es ist hierbei wichtig zu erwähnen, dass es keine einheitliche Definition gibt welche Eigenschaften zur Berechnung der Komplexität heranzuziehen sind. Komplexitäts-Metriken sind daher von Natur aus subjektive Metriken, die einer genaueren Interpretation durch den Menschen bedürfen. Durch die verschiedenen Ansätze und Definitionen ist auch bei Vergleichen von Software-Produkten Vorsicht geboten, [27]. Eine der ersten Definitionen für Komplexität liefert der IEEE-Standard für Terminologie: Complexity = „Degree of Complication of a system or system component determined by such factors as the number and intricacy of interfaces, the number and intricacy of conditional branches, the degree of nesting, the types of data structures and other system characteristics“, [31]. Nach dieser Definition ist die Komplexität einer Software abhängig von der Anzahl der Schnittstellen, der Anzahl der Bedingungen, der Anzahl der Verschachtelungsstufen und der Anzahl unterschiedlicher Datentypen. Abstrahiert man diese Aussage weiter, kommt man zu dem Schluss, dass Komplexität grundsätzlich von der Anzahl der Beziehungen zwischen Elementen sowie

der Anzahl verschiedener Elementtypen abhängig ist, [32]. Diese grundlegende Definition dient den meisten Komplexitäts-Berechnungen heute als Grundlage, wobei aber immer andere Aspekte in den Vordergrund gerückt werden. Halsteads Komplexitätsmaß basiert etwa auf der semantischen Komplexität des Codes, das bedeutet eine Software ist umso komplexer, je mehr unterschiedliche Befehls- und Datentypen im Code enthalten sind, [33]. Gilb verweist dagegen auf die strukturelle Komplexität und damit auf die Verknüpfungen zwischen den Elementen, [34]. Wiederum McCabe sieht Software als gerichteten Graph, dessen Verhältnis von Kanten zu Knoten einen Aufschluss über die Komplexität gibt, [35]. Der Grund für die Entwicklung so vieler Maße im Laufe der Zeit liegt laut Fenton daran, dass Komplexitätsmaße nicht additiv sind, da es sich um ungleiche Maße handelt. Deshalb sei es besser, sich je nach Anwendungsfall auf bestimmte Attribute zu konzentrieren, [36].

Softwarekomplexität kann auf mindestens vier Ebenen betrachtet werden, [27]:

- Problemebene,
- Anforderungsebene,
- Entwurfsebene,
- Codeebene.

Die oberste Ebene spiegelt die Komplexität der fachlichen Aufgaben an sich wieder und wird als **Problemebene** bezeichnet. Manche Fachprobleme sind komplexer als andere, was auf die Anzahl der gegenseitig abhängigen Elemente (Geschäftsobjekte und Geschäftsprozesse) zurückzuführen ist, [27].

Die **Anforderungsebene** beinhaltet die fachliche Lösung, beziehungsweise dessen Beschreibung. Dementsprechend erfolgt die Berechnung der Komplexität anhand der Anzahl der Beziehungen zwischen den fachlichen Elementen. Zu diesen gehören fachliche und technische Anforderungen, fachliche Objekte, fachliche Schnittstellen und Anwendungsfälle, [27].

In der **Entwurfsebene** befindet sich die technische Lösung in Form eines Gebildes aus softwaretechnischen Entitäten, beziehungsweise Modelltypen und deren Beziehungen. Modelltypen sind in der Regel Komponenten, Module, Klassen, Schnittstellen, Oberflächen und Datenbanktabellen. Die Komplexität steigt mit der Anzahl der Entitäten, der Anzahl der verschiedenen Modelltypen und der Anzahl der Beziehungen, [27].

Auf der **Codeebene** liegt das Hauptaugenmerk auf der Komplexität der Implementierung selbst. Es gibt eine Reihe von Ansätzen, um die Komplexität einer Implementierung zu messen, so spielen etwa die Anzahl der verwendeten Datentypen oder die Anzahl der Ausführungspfade eine wichtige Rolle, [27].

**Qualitäts-Metriken** beschäftigen sich mit der Qualität eines Produkts, eines Prozesses oder des gesamten Projekts. Dabei spielen Qualitäts-Metriken in Verbindung mit dem Produkt oder dem Prozess eine wichtigere Rolle, als Qualitäts-Metriken des Projekts. Jedoch gibt es natürlich auch für das Projekt Qualitäts-Metriken, wie etwa die Anzahl der Entwickler, Termine, Größe und die Organisationsstruktur, die indirekt die Qualität des Produkts und des Prozesses beeinflussen, [29]. Wie Komplexitäts-Metriken sind auch Qualitäts-Metriken stark subjektiv zu bewerten. Dabei gehen aber die Ansichten über gute Software-Qualität noch viel weiter auseinander als bei der Software-Komplexität. Ein Kunde misst die Qualität der Software an der Verarbeitungsgeschwindigkeit von Aufgaben, einem anderen ist die Verlässlichkeit besonders wichtig und wieder ein anderer legt nur Wert auf das Design und die Benutzerfreundlichkeit, egal wie lange die Software zum Ausführen von

Befehlen braucht. Das bedeutet, um Software-Qualität sinnvoll messen zu können, ist ein Regelwerk notwendig, welche Eigenschaften eine Software unbedingt, vielleicht oder gar nicht erfüllen muss. Der Kunde ist in diesem Fall in die Pflicht genommen, zusammen mit dem Lieferanten eine Qualitätsnorm für das geforderte Produkt zu definieren. Je genauer diese Qualitätsnorm im Vorfeld definiert wird, desto genauer kann am Ende und auch im Laufe der Entwicklung die Qualität überprüft werden. Demnach kann Qualität als Erfüllung von Anforderungen [37], als Eignung für den Zweck [38] oder als Grad der Erfüllung von Erwartungen des Anwenders [39] definiert werden. Es kann zum Abschluss festgehalten werden, dass ein enger Zusammenhang zwischen Qualität und Anforderungen besteht. Qualität ist das, was der Benutzer anfordert, [27].

Das wurde auch im IEEE-Standard 610 mit zwei Definitionen spezifiziert, [40]:

1. Softwarequalität ist der Grad, in dem eine Software seinen spezifizierten **Anforderungen** entspricht.
2. Softwarequalität ist der Grad, in dem eine Software die **Bedürfnisse und Erwartungen** der Benutzer/innen erfüllt.

Die erste Definition kann leicht mithilfe eines Soll-Ist-Vergleiches auf Basis der Anforderungen überprüft werden. Die Zweite hingegen ist subjektiver und geht auf die Bedürfnisse und Erwartungen einzelner Benutzer/innen ein, die nicht so einfach überprüft werden können. Der Frage, wie die Erwartungen der Benutzer/innen in Zahlen ausgedrückt werden können, widmete sich Gilb in seinem Buch zum Thema Softwaremetrik, [34]. Seiner Meinung nach können Erwartungen folgendermaßen klassifiziert werden, [34]:

- **Funktionalität** = Der Benutzer erwartet, dass das System seine Anforderungen erfüllt.
- **Leistung** = Der Benutzer erwartet, dass das System schnell auf seine Eingaben reagiert.
- **Zuverlässigkeit** = Der Benutzer erwartet, dass das System keine fehlerhaften Ergebnisse liefert.
- **Sicherheit** = Der Benutzer erwartet, dass das System seine Daten sichert und nicht verliert.
- **Effizienz** = Der Benutzer erwartet, dass das System sparsam mit den Rechnerressourcen umgeht.
- **Verfügbarkeit** = Der Benutzer erwartet, dass das System immer verfügbar ist.
- **Wartbarkeit** = Der Benutzer erwartet, dass sich das System leicht erweitern lässt.
- **Portierbarkeit** = Der Benutzer erwartet, dass sich das System ohne viel Aufwand auf eine andere Plattform portieren lässt.

Damit eine der oben erwähnten Qualitätseigenschaften einsetzbar ist, muss sie laut Gilb mit einer Zahl messbar oder mit einer Ja/Nein-Antwort zu beantworten sein. Damit stellte Gilb bereits im Jahre 1976 eine enge Verbindung zwischen Qualität und Metrik her. Eine Eigenschaft ist demnach nur einsetzbar, wenn sie auch messbar ist. Ähnliche Ansätze wurden auch von Boehm [41], McCall [42] oder auch in dem ISO-Standard 9126 [43] entwickelt, [27].

### 2.3.2 Code-Metriken

Code- oder Produkt-Metriken befassen sich mit der Messung der Implementation beziehungsweise des Produktes selbst. Programmcode weist dabei von allen Softwareartefakten den höchsten Grad an Formalismus auf und eignet sich deshalb sehr gut zur Messung und Berechnung von Metriken. Das hat zur Folge, dass bereits über 200 Metriken für die Messung von Programmcode bekannt sind und

es kommen ständig neue hinzu. Die Entwicklung von Code-Metriken wird durch die wohldefinierte Syntax von Programmiersprachen mit einem begrenzten Vokabular begünstigt. Grundsätzlich besteht eine Programmiersprache aus Operatoren und Operanden. Operatoren sind Befehlswörter (if, switch, while, ...) oder Befehlszeichen (+, -, =, &, ...) und spezifizieren die auszuführende Operation oder Aktion. Operanden spezifizieren die Daten, mit denen die Operationen ausgeführt werden. Operanden unterteilen sich weiter in Variablen und Konstanten und besitzen Eigenschaften, die den Typ und das Format des Operanden festlegen (z.B.: integer, char, string). Konstanten stellen einen fixen Wert dar und werden direkt in den Programmcode eingebettet, während Variablen einen Verweis auf eine Speicheradresse darstellen, der überschrieben werden kann. Eine sinnvolle Aneinanderreihung von Operatoren und Operanden wird als Anweisung bezeichnet, [27]. In der Anweisung „c = a + b“ spezifizieren „c“, „a“ und „b“ die Daten, sind also Operanden und „=“ beziehungsweise „+“ bestimmen die Art der Operationen. Die Form der Anweisung, der sogenannte Syntax, kann von Programmiersprache zu Programmiersprache sehr unterschiedlich ausfallen. Gemeinsam ist nur, dass Anweisungen in verschiedene Kategorien unterteilt werden können, [44]:

- Deklarative Anweisungen,
- Operative Anweisungen,
- Übersetzungsanweisungen und
- Kommentare.

Deklarative Anweisungen definieren Variablen, Konstanten oder andere Anweisungen. Operative Anweisungen verarbeiten Daten oder verbinden Anweisungen untereinander, zum Beispiel GOTO-Anweisung. Mit Übersetzungsanweisungen kann dem Compiler mitgeteilt werden wie der Code übersetzt werden soll. Kommentare besitzen keine Syntaxregeln und dienen rein dazu, den Code besser zu verstehen, [27].

### **Quantität**

Die Größe einer Software zu messen ist eine einfache und objektive Metrik und dient als Basis zur Berechnung der Produktivität und Komplexität. Jedoch gibt es eine Vielzahl an Quantitäts-Metriken und nicht alle eignen sich gleichermaßen für die verschiedenen Anforderungen. Zum Beispiel ist die Anzahl der Codezeilen mitsamt der Kommentare geeignet zur Messung der Produktivität, während bei der Ermittlung der Komplexität lediglich die Anweisungen gezählt werden sollten, [45]. Des Weiteren können über Quantitäts-Metriken Prognosen über Wartungsaufwände und Testaufwände getroffen werden. Es gibt eine Vielzahl an Größen, die auf der Codeebene ermittelt werden können; [27]:

- Codedateien,
- Codezeilen,
- Anweisungen,
- Prozeduren beziehungsweise Methoden,
- Module beziehungsweise Klassen,
- Entscheidungen,
- Logikzweige,
- Aufrufe,
- vereinbarte Datenelemente,
- benutzte Datenelemente beziehungsweise Operanden,

- Datenobjekte,
- Datenzugriffe,
- Benutzeroberflächen und
- Systemnachrichten.

**Codedateien** beinhalten den Source-Code von Methoden, Modulen, Klassen, Bibliotheken, und so weiter und existieren in sämtlichen Programmiersprachen. Diese Größe ist für die Codeanalyse wichtig, wie viele Codeeinheiten analysiert werden müssen, [27].

Eine **Codezeile** entspricht einem Satz in einer Codedatei. Früher war die Länge einer Codezeile auf 80 Zeichen begrenzt, weil nicht mehr Zeichen auf den Lochkarten Platz fanden. Durch diese Einschränkung haben Programmierer damals so gut wie immer eine Anweisung pro Zeile geschrieben. Heutzutage existiert diese Einschränkung nicht mehr und es können mehrere Anweisungen in einer Zeile stehen, sowie eine Anweisung über mehrere Zeilen aufgeteilt werden. Deshalb hat die Codezeile als Größenmaß heute weitgehend ihrer Bedeutung verloren und ist durch die Anzahl der Anweisungen ersetzt worden, [27].

Eine **Anweisung** stellt einen Befehl oder ein Kommando dar und besteht aus Operatoren und Operanden. Anweisungen kommen in allen Programmiersprachen vor und werden meistens mit einem Sonderzeichen, dem sogenannten „Delimiter“ abgeschlossen. Es gibt aber auch Sprachen, die auf einen Delimiter verzichten und stattdessen jede Zeile als Anweisung interpretieren. Die Anzahl der Anweisungen ist heutzutage das Wichtigste und am besten vergleichbare Maß für die Codegröße, [27].

**Prozeduren**, in objekt-orientierten Sprachen als Methoden bezeichnet, sind eine Ansammlung von Anweisungen mit dem Zweck, aus mehreren Argumenten ein Ergebnis zu erzeugen. Obwohl es viele Empfehlungen zur Anzahl der Anweisungen innerhalb einer Prozedur gibt, obliegt es den Programmierenden selbst, wie viele sie in eine Prozedur schreiben. Durch diese Unzuverlässigkeit ist die Anzahl der Prozeduren kein sehr verlässliches Größenmaß, [27].

Als **Module** werden Teile des Source-Codes bezeichnet, die in einem Durchlauf durch den Compiler in eine Bytecodedatei übersetzt werden. Ein Modul kapselt dabei mehrere Prozeduren in eine Bytecodedatei. Ähnlich wie bei Prozeduren, gibt es keine Einschränkungen wie lange ein Modul sein darf oder wie viele Prozeduren darin enthalten sind. Deshalb ist auch dieses Größenmaß alleine ein unzuverlässiger Wert für die Codegröße, [27].

**Entscheidungen** entstehen, indem der Programmablauf durch Bedingungen gesteuert wird. Typische Entscheidungen sind Alternativ-, Auswahl und Schleifenbedingungen. Wird der Programmablauf als gerichteter Graph dargestellt, entsprechen die Entscheidungen den Knoten in dem Graphen, deshalb ist die Zahl der Entscheidungen ein wichtiges Maß zur Ermittlung der Codekomplexität, [27].

Ein **Logikzweig** ist ein Ausgang aus einer Entscheidung. In einem gerichteten Graphen entspricht er also einer Kante. Eine IF-Anweisung kann etwa zwei Zweige enthalten (THEN-Zweig und ELSE-Zweig), während eine CASE- oder SWITCH-Anweisung beliebig viele Zweige aufweisen kann. Ziel beim Testen ist es, so viele Logikzweige wie möglich abzudecken, [27].

### Komplexität



Im Jahr 1985 haben Lehman und Belady festgestellt, dass Software kontinuierlich verändert und weiterentwickelt werden muss, um den neuesten Anforderungen zu entsprechen. Dabei wird aber in jedem Änderungsschritt zwangsweise die Komplexität der Software erhöht, wenn keine aktiven Gegenschritte eingeleitet werden (Lehman's laws), [46]. Laird und Brennan bezeichnen Komplexität sogar als Typhus, die sich wie eine Epidemie ausbreitet, [47]. Zusammen mit der Aussage von Katherine Gerould: „Simplicity must be enforced. When man is left free, he inevitably complicates matters.“, [48] wird augenscheinlich, wie wichtig die Messung und Kontrolle der Komplexität in einem Software-Projekt ist. Dabei ist es die Komplexität der Lösung, also der Implementation, die den Großteil der Fehler ausmacht und somit den Wartungsaufwand in die Höhe treibt.

Die Software-Komplexität kann grundsätzlich in drei Arten unterteilt werden, [27]:

- Strukturelle Komplexität,
- Sprachliche Komplexität und
- Algorithmische Komplexität.

**Strukturelle Komplexität** beschäftigt sich mit der Art der Systemzusammensetzung. Ein Softwaresystem besteht aus verschiedenen Elementen (Modulen, Operationen, Datenobjekten, usw.), die miteinander interagieren und somit in Beziehung zueinander stehen. Die Strukturkomplexität besteht demnach aus dem Verhältnis der Anzahl der Elemente zu der Anzahl der Beziehungen, [27]:

$$\text{Strukturkomplexität} = 1 - \frac{\text{Strukturelemente}}{\text{Strukturbeziehungen}} \quad (1)$$

Zusammenfassend kann festgehalten werden, dass, je dichter das Netz von Beziehungen zwischen den Elementen ist, desto höher wird auch die strukturelle Komplexität. Eine der bekanntesten Metriken ist hierzu McCabe's zyklomatische Komplexität, [35]. McCabe bediente sich der Graphentheorie und stellte den Programmcode als gerichteten Graphen mit den Entscheidungsanweisungen als Knoten und den Kanten als Programmzweige dar. Er formulierte folgende Gleichung für die zyklomatische Komplexität, [27]:

$$V(s) = e - n + 2p$$

$$\begin{array}{l} e \dots \text{Anzahl der Kanten} \\ n \dots \text{Anzahl der Knoten} \\ p \dots \text{Anzahl der Knoten ohne ausgehende Kanten im Graphen} \end{array} \quad (2)$$

Laut McCabe darf die zyklomatische Komplexität den Wert 10 nicht überschreiten. Die folgende Sortierroutine hat 3 Knoten und 6 Kanten, was eine zyklomatische Komplexität von 5 ergibt, [27].

```

do I to N;                                1 Knoten, 2 Kanten
  do J to I;                               1 Knoten, 2 Kanten
    if X(I) < X(J)                         1 Knoten, 2 Kanten
    then
      SAVE = X(I);
      X(I) = X(J);
      X(J) = SAVE;
    else
      continue;
  end;
end;

```

$$6 - 3 + 2 = 5$$

Kritik an McCabe's zyklomatischer Komplexität gibt es dadurch, dass Verschachtelungen oft als komplexer angesehen werden, als sie tatsächlich sind. So dienen „switch“-Anweisungen zum besseren Verständnis des Programmcodes, erhöhen dabei aber gleichzeitig die zyklomatische Komplexität, [49].

Die **sprachliche Komplexität** legt das Augenmerk auf die Verständlichkeit der Programmiersprache. Eine Sprache mit vielen verschiedenen Wörtern und Symbolen ist demnach komplexer als eine Sprache mit wenigen Wörtern und Symbolen. Die Sprachkomplexität für ein bestimmtes Programm kann ermittelt werden, indem das Verhältnis der benutzten Sprachelemente zu allen verfügbaren Sprachelementen in der jeweiligen Programmiersprache gebildet wird, [27]:

$$\text{Sprachkomplexität} = \frac{\text{benutzte Sprachelemente}}{\text{verfügbare Sprachelemente}} \quad (3)$$

Halstead stellte in seinem Buch „Software Science“ [33] eine Metrik zum Berechnen der Sprachkomplexität vor. Der Gedanke dabei ist, dass ein Programm mit wenigen Operatoren und Operanden, so viele Informationen wie möglich verarbeiten soll. Demnach setzt Halstead die Anzahl der verwendeten Operatoren und Operanden mit der jeweiligen Anzahl der Verwendungen ins Verhältnis, [27]:

$$\text{Komplexität (Halstead)} = \frac{\text{Operatoren}}{\text{Operatorenverwendungen}} * \frac{\text{Operanden}}{\text{Operandenverwendungen}} \quad (4)$$

Unter **algorithmischer Komplexität** wird die Anzahl der erforderlichen Schritte zur Auflösung eines Algorithmus verstanden. Je schwerer also ein Algorithmus zu verstehen ist, desto komplexer ist er. So wird beispielsweise ein rekursiver Algorithmus als komplexer empfunden als eine Schleife, weil er einen höheren Abstraktionsgrad verlangt, um ihn zu verstehen. Je allgemeiner, beziehungsweise abstrakter, also ein Algorithmus ist, desto schwieriger ist er zu verstehen. Eine Möglichkeit, die algorithmische Komplexität quantitativ darzustellen besteht in dem Verhältnis der Anwendungsschritte zu den Anwendungsfällen zusammen mit dem Verhältnis der Ein- und Ausgaben zu den Verarbeitungsschritten. Ein Algorithmus ist also umso komplexer, je mehr Schritte die Anwendungsfälle besitzen und je mehr Ein- und Ausgabeoperationen pro Schritt ausgeführt werden. Die algorithmische Komplexität kann auf folgende Weise berechnet werden, [27]:

$$\text{Algorithmische Komplexität} = 1 - \left( \frac{\text{Anwendungsfälle}}{\text{Anwendungsfallschritte}} * \frac{\text{Verarbeitungsschritte}}{\text{Ein- und Ausgaben}} \right) \quad (5)$$

## Qualität

Durch die subjektive Natur der Qualität, ist es schwierig eine allgemeingültige Metrik zu ermitteln. Trotz allem gibt es Versuche, einen allgemeinen Index für die Qualität einer Software zu entwickeln. Frank Simon hat zum Beispiel einen Codequalitätsindex entwickelt, indem er misst, zu welchem Grad die Software bestimmte Qualitätseigenschaften einhält, [50]. Diese Herangehensweise setzt allerdings wiederum ein intensives Auseinandersetzen mit den subjektiven Qualitätseigenschaften voraus, um eine sinnvolle Reihung der Eigenschaften vornehmen zu können. Einen weiteren Ansatz stellt der Maintainability-Index von Oman dar, [51]. Dabei wurden aus über 60 Metriken nach einer Reihe von Tests und mithilfe der subjektiven Meinung von Wartungsingenieuren 4 Metriken ausgewählt, aus denen ein Index berechnet werden kann, der den Wartungsaufwand einer Software am ehesten abbildet, [27]:

- Halsteads Aufwandsmaß,
- McCabes zyklomatische Komplexität,
- Kommentierungsgrad,
- Modulgröße.

Trotz der hohen Übereinstimmung des errechneten Indexes zu der subjektiven Einschätzung von Experten von bis zu 93%, wird er in der Praxis eher selten eingesetzt. Grund dafür ist, dass die Codequalität selbst, einen eher geringen Anteil an den Kosten für die Wartung hat. Viel größer ist der Einfluss, den die Wartungsumgebung und das Wartungspersonal auf die Kosten haben (zusammen zirka 60 %). Es lohnt sich also eher, in Verbesserung der Umgebung und des Personals zu investieren, als in die Messung und die Verbesserung der Codequalität. Daher wird in der Praxis eher die Frage gestellt, welche Qualitätseigenschaften für das Produkt gefordert sind und diese danach gezielt überprüft und bewertet, [27].

### 2.3.3 Prozess-Metriken

Um den gesamten Softwareentwicklungszyklus messen und im Endeffekt verbessern zu können, ist es notwendig, auch einen Blick auf die Prozesse der Entwicklung zu werfen. Die Aufgabe der Prozesse ist es sicherzustellen, dass ein Produkt mit der richtigen Funktionalität und der richtigen Qualität entwickelt wird. Demnach liegt das Hauptaugenmerk der Prozess-Metriken auf der Anforderungs-, Fehler- und Testmessung, [21].

## Quantität

Im Gegensatz zu Code-Metriken, bei denen die Messung der Quantität meistens einfach und objektiv erfolgen kann, ist es bei Prozess-Metriken schwieriger objektive Quantitäts-Messungen durchzuführen. Grund dafür ist, dass Anforderungen vorwiegend in natürlicher Sprache verfasst werden und dadurch das objektive Zählen erschweren. So können in einem Projekt mehrere Anforderungen in einem Textdokument zusammengefasst sein, während sie in einem anderen einzeln beschrieben werden. Ein Fortschritt ist, dass die Anforderungsdokumente heutzutage strukturiert und teilweise auch formalisiert sind. Dies ermöglicht erst die sinnvolle Messung der Quantität über mehrere Projekte hinweg. Trotz alledem gibt es jedoch noch immer

Anforderungsdokumente, die nicht messbar sind. Deshalb sind insbesondere Anforderungsgrößen mit besonderer Vorsicht zu betrachten. Folgende Anforderungsgrößen können gezählt werden, [27]:

- die Anzahl der Anforderungen,
- die Anzahl der Abnahmekriterien,
- die Anzahl der Anwendungsfälle,
- die Anzahl der Systemschnittstellen,
- die Anzahl der Geschäftsobjekte,
- die Anzahl der Benutzeroberflächen,
- die Anzahl der Akteure,
- die Anzahl der Verarbeitungsschritte.

**Anforderungen** sind die Menge aller Texteinträge, die das Verhalten und die Funktionalität des Produktes spezifizieren. Dabei können Anforderungen in funktionale und nicht-funktionale Anforderungen unterteilt werden. Ausschlaggebend für die Interpretation der Zählung ist hierbei der Detaillierungsgrad mit dem die Anforderungen beschrieben werden. Die Formulierungen können hierbei sehr allgemein („Das System muss dasselbe können wie das Altsystem.“) aber auch sehr genau sein („Nach Drücken der Enter-Taste wird das Datum automatisch befüllt.“). Als sinnvoll haben sich Anforderungen zwischen ein bis drei Sätzen bewährt, die eine konkrete Funktion oder eine bestimmte Systemeigenschaft definieren, [27].

Im optimalen Fall gibt es zu jeder Anforderung ein **Abnahmekriterium** an dem die Erfüllung der Anforderung gemessen wird. Abnahmekriterien können entweder direkt in den Anforderungen oder in einem eigenen Dokument aufgeführt werden. Aus der Anzahl der Abnahmekriterien kann in etwa eine Mindestanzahl von Prüffällen abgelesen werden, [27].

Anwendungsfälle entsprechen Geschäftsvorgängen und entsprechen einer einmaligen Nutzung des Systems, an einem Ort, zu einer Zeit, durch einen bestimmten Akteur. Wichtig ist zu beachten, dass gleiche Anwendungsfälle, die an verschiedenen Stellen beschrieben sind, nur einmal gezählt werden, [27].

**Systemschnittstellen** sollten aus dem Anforderungstext zu erkennen sein und spezifizieren Kommunikationskanäle zwischen verschiedenen Systemen. Die Anzahl der Systemschnittstellen ist vor allem zur Ermittlung der Function-Points von Bedeutung, [27].

Ein weiteres Größenmaß, welches auf Prozess-Ebene gewonnen werden kann, ist die Testgröße. Der Source-Code selbst liefert meistens eine recht ungenaue Abbildung der Systemanforderungen, deshalb wird die Testgröße oft auf Prozess-Ebene gewonnen. Ziel des Testens ist es das System durch das Finden und Beheben von Fehlern in einen funktionstüchtigen und korrekten Zustand zu bringen. Das Testen ist ein unerlässlicher Bestandteil des Software-Entwicklungsprozesses. Es können folgende Testgrößen ermittelt werden, [27]:

- die Anzahl der Testfälle,
- die Anzahl der Testfallattribute,
- die Anzahl der Testläufe,
- die Anzahl der Testskripte beziehungsweise Testprozeduren,
- die Anzahl der Testskriptzeilen,

- die Anzahl der Testskriptdurchführungen.

**Testfälle** sind die Beschreibung der Ausführung eines Testobjektes. Ein Testobjekt kann hierbei ein Softwarebaustein, ein Teilsystem oder das ganze System sein. Es gilt, jeden Objektzustand, jede Bedingung und jede Aktion eines Systems zu testen. Demzufolge muss es für jeden Zustand, jede Bedingung und jede Aktion einen Testfall geben. Die Anzahl der Testfälle ist die Basis für die Schätzung des Testaufwandes, [27].

**Testfallattribute** bezeichnen Vorbedingungen für die Testfallausführung, Nachbedingungen für die Erfolgsprüfung, Umgebungen und den Status eines Testfalles. Die Anzahl der vorhandenen Attribute im Vergleich zu den geplanten Attributen ist ein Indiz für die Vollständigkeit des Testfalls, [27].

Ein **Testlauf** besteht aus einer Sequenz mehrerer Testfälle, die in einem Durchgang ausgeführt werden. Die Anzahl der Testläufe relativ zu der Anzahl der Testfälle ermöglicht eine Aussage über die Komplexität des Tests. Die Anzahl der Testläufe selbst ermöglicht eine Einschätzung, wie lange der Test dauern wird, [27].

Das Testen erfüllt den Zweck, Fehler in der Software zu finden und zu beheben, bevor die Software in den produktiven Einsatz geht. Deshalb ist auch die **Anzahl der gefundenen Fehler** während des Testens eine wichtige Kennzahl. Dabei ist aber nicht jeder Fehler gleichwertig, sondern anhand seiner Auswirkungen zu bewerten. Dementsprechend ergeben sich Fehlerklassen, z.B. kritische, schwere, mittlere und leichte Fehler, die verwendet werden, um die Fehler zu gewichten und so zu einer Einschätzung aller gefundenen Fehler zu kommen. Eine Möglichkeit die Fehler zu gewichten, ist zum Beispiel kritische Fehler mit 8, schwere Fehler mit 4, mittlere Fehler mit 2 und geringe Fehler mit 1 zu multiplizieren. Es ergeben sich also drei Größen:

- Anzahl aller gefundenen Fehler,
- Anzahl aller Fehler je Fehlerklasse und
- eine gewichtete Summe aus allen gefundenen Fehler.

Die gewichtete Summe ist ein besserer Indikator für die Testeffektivität, weil das Finden von ein paar kritischen oder schweren Fehlern mehr wiegt als das Finden vieler geringer Fehler. Die reine Anzahl der gefundenen Fehler ist dagegen für die Prognose der Restfehlerwahrscheinlichkeit notwendig. Die Testeffektivität selbst ergibt sich aus dem Verhältnis der Anzahl der gefundenen Fehler zur Anzahl der ausgeführten Testfälle.

### **Komplexität**

Die Komplexität auf Prozess-Ebene vermischt sich sehr stark mit der Quantität, weil die Komplexität eines Systems aus Prozesssicht sehr stark von der Anzahl der verschiedenen Anforderungen und Testfälle abhängt. Zusätzlich ist auch die Anzahl der Beziehungen zwischen den einzelnen Elementen entscheidend. So zum Beispiel mit wie vielen anderen Anforderungen eine spezifische Anforderung in Beziehung steht oder wie viele Testfälle zu einem einzelnen Testlauf gehören, [27].

### **Qualität**

Mehrere Studien haben nachgewiesen, dass über 40 % der gemeldeten Softwarefehler auf Mängel in den Anforderungen zurückzuführen sind, [47], [52]. Zudem sind Probleme mit den Anforderungen der Hauptgrund für den Abbruch von Projekten, [53]. Umso wichtiger erscheint es, die Qualität der Anforderungen genauso wie die Qualität des Codes zu messen und zu überprüfen. Dabei muss

grundsätzlich zwischen funktionalen und nicht-funktionalen Anforderungen unterschieden werden. Funktionale Anforderungen definieren die Funktionen und den Umfang eines Produktes, während nicht-funktionale Anforderungen Qualitätsziele beschreiben, die das Produkt erreichen soll. Zur Definition der nicht-funktionalen beziehungsweise Qualitätsanforderungen schreibt der ISO-Standard 25030 [54] folgendes Schema, bestehend aus folgenden Attributen, vor. Diese Attribute sind, [27]:

- Qualitätsmerkmal,
- Messeinheit,
- Messmethode und
- Zielwert.

Ein Beispiel für die Anwendung wäre zum Beispiel die Definition der Antwortzeit eines Systems:

- Qualitätsmerkmal = Antwortzeit,
- Messeinheit = Sekunden,
- Messmethode = Erfassung der vergangenen Zeit zwischen dem Drücken der Enter-Taste und dem Erscheinen der Antwort,
- Zielwert = 2 Sekunden.

Laut dem Standard soll für jede nicht-funktionale Anforderung eine Definition nach gezeigtem Muster existieren. Dadurch wäre bereits am Ende der Anforderungserstellung klar wie die Qualitätsziele überprüft und welcher Wert zu erreichen ist. In weiterer Folge könnte diese Form von Definition auf die branchenspezifischen, funktionalen Anforderungen ausgeweitet werden, [27].

Ein weiterer Ansatz für die Kontrolle der Anforderungen stammt von Christof Ebert, [55]. Ihm zufolge genügen nur wenige Metriken, um die Qualität des Anforderungsmanagements zu überprüfen, [55]:

- Anzahl aller Anforderungen in einem Projekt,
- Fertigstellungsgrad der Anforderungen,
- Änderungsrate pro Entwicklungsphase,
- Anzahl der Ursachen für die Anforderungsänderung,
- Anzahl der Mängel in den Anforderungen,
- Nutzwert der einzelnen Anforderungen.

Die Anzahl aller Anforderungen ist ein Volumenmaß für den zu erwartenden Projektaufwand. Der Fertigstellungsgrad bestimmt, inwieweit die vereinbarten Anforderungen bereits umgesetzt wurden. Aufgrund der Änderungsrate kann bestimmt werden, wie stabil das Projekt ist und wie vielen Änderungen die Anforderungen ausgesetzt sind. Die Änderungsrate ist der Prozentsatz der geänderten oder neu hinzugefügten Anforderungen zu allen Anforderungen. Die Anzahl der Änderungsursachen hilft auszuwerten, welche Ursachen am öftesten zu Änderungen führen. Änderungsursachen können zum Beispiel Versäumnisse bei der Anforderungserstellung ebenso wie veränderte Projektziele darstellen. Die Ermittlung der Anzahl der Mängel erfolgt aufgrund einer Checkliste, die Regeln zur Erstellung von Anforderungen vorgibt. Regeln können etwa die maximale Anzahl an Sätzen oder die Spezifikation von Testmethoden sein. Zu guter Letzt empfiehlt Ebert jeder Anforderung einen Nutzwert zuzuordnen, der relativ zum Gesamtprojekt steht, um Anforderungen gewichten und priorisieren zu können, [27].

Ein weiteres wichtiges Messverfahren auf Prozess-Ebene ist die Verfolgung des Testfortschritts. Da das System erst nach der Ausführung aller Tests ausgeliefert werden kann und weil auch alle Fehler erst nach dem Testen feststehen, ist der Testfortschritt eine wichtige Kennzahl für den Zustand des Projekts. Eine Möglichkeit den Testfortschritt zu verfolgen, ist mithilfe einer S-Kurve über die Zeit. Dabei werden auf der x-Achse die Zeitintervalle aufgetragen (Tage, Wochen, Monate, etc...) und auf der y-Achse die Anzahl der Testfälle (siehe Abbildung 8). Die namensgebende S-Form ergibt sich aus der kumulierten Darstellung der Testfälle, die in intensiven Testphasen zu einem raschen Anstieg der geplanten Testfälle führt.

Um möglichst viel Information zu erhalten, sollte das Diagramm, pro Zeitintervall, folgende Werte enthalten, [29]:

- Geplante Testfälle: die Anzahl an Testfällen, die in einem speziellen Intervall ausgeführt werden sollten
- Ausgeführte Testfälle: die Anzahl an Testfällen, die in einem speziellen Intervall tatsächlich ausgeführt wurden
- Erfolgreich ausgeführte Testfälle: die Anzahl an Testfällen, die in einem speziellen Intervall ausgeführt wurden und keine Fehler meldeten

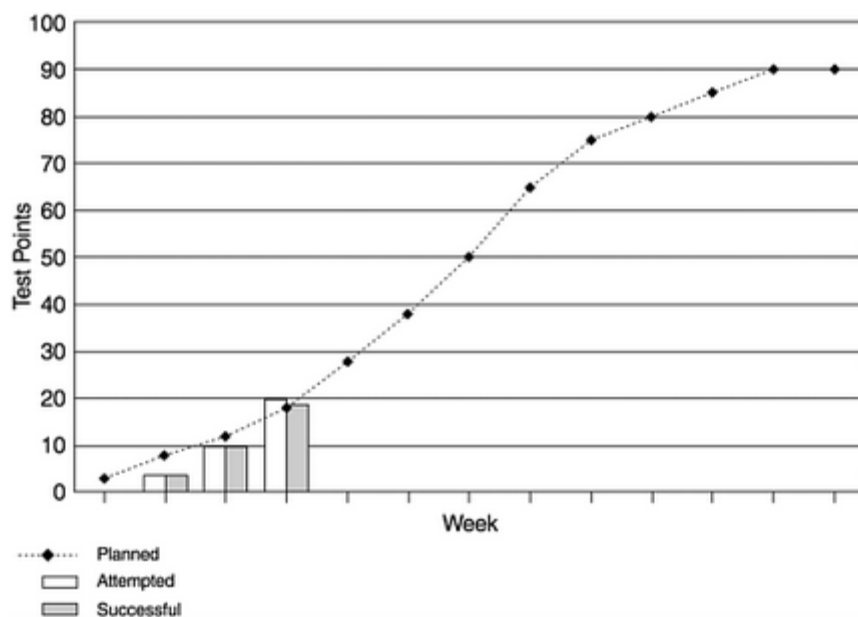


Abbildung 8: Testfortschritt S-Kurven-Diagramm, [29]

Mithilfe dieser Technik kann der geplante Testfortschritt mit dem tatsächlichen Testfortschritt verglichen werden. Dementsprechend können dann Maßnahmen gesetzt werden. Es ist zum Beispiel bekannt, dass das Testen unter einem engen Zeitplan oft in das Hintertreffen gerät und vernachlässigt wird. Mit einem funktionierenden S-Kurven-Diagramm fällt es dem Entwicklungsteam sehr viel schwerer das Problem des mangelnden Testens zu ignorieren, [29].

Es gibt weitere Metriken, die geplante Elemente mit tatsächlich umgesetzten Elementen vergleichen. So ist die Fehlerbehebungsrate das Verhältnis zwischen den behobenen Fehlern und allen gefundenen Fehlern. Die Fehlerbehebungsrate ist ein guter Indikator für die Effizienz, mit der gefundene Fehler im System behoben werden und somit auch eine Aussage über die Qualität. Es

bietet sich an, die Fehlerbehebungsrate für jede einzelne Entwicklungsphase zu berechnen, um zu sehen, in welcher Phase die Behebung der Fehler verbessert werden kann, [30].

### 2.3.4 Methoden zur Auswahl geeigneter Metriken

Durch die große Anzahl an verfügbaren Metriken ist es unmöglich, jede Metrik in einem Unternehmen zu messen und zu berechnen. Gepaart mit dem Umstand, dass die Messung, Berechnung und Interpretation der Metriken oft einen beträchtlichen Aufwand im Entwicklungsprozess einnehmen kann, stellt sich die Frage, auf welche Metriken man zurückgreifen soll. Das „National Institute of Standards and Technology (NIST)“ hat hierzu vier kritische Faktoren identifiziert, die für einen sinnvollen Einsatz einer Metrik notwendig sind. Diese vier Faktoren sind die Qualität des Datenmaterials, die Relevanz der Metrik für die Unternehmensziele, die Überschaubarkeit der Metrik und die Stakeholder, die Interesse an der Metrik haben, [56].

Eine Metrik ist nur so gut wie die Daten, die ihr zugrunde liegen. Deshalb ist die **Datenqualität** ein entscheidender Faktor für die Einsetzbarkeit einer Metrik. Dabei spielt nicht nur die Art der Datenerhebung (automatisiert oder manuell) eine Rolle, sondern auch die Datenintegrität und die Möglichkeit Messungen zu validieren. Eine Standardisierung und Periodisierung der Datenerhebung ist ein wichtiger Schritt, um die Zuverlässigkeit und das Vertrauen in die Daten zu vergrößern. Weil das Erstellen von Metriken in den meisten Fällen einen hohen Aufwand bedeutet, ist die **Relevanz** der Metrik entscheidend. Dementsprechend muss eine Metrik gezielt einen Beitrag zur Erfüllung von bestimmten Unternehmenszielen leisten. Deshalb ist eine genaue Analyse der Unternehmensziele und der verfügbaren Metriken erforderlich. Ein ebenso wichtiger Faktor beim Erstellen eines Metriken-Pools ist die **Überschaubarkeit**. Diese hängt sehr stark mit der Relevanz der Metriken zusammen. Wenn eine jede Metrik einem bestimmten Ziel dient, ist es unwahrscheinlicher, dass unwichtige oder veraltete Metriken im Pool verbleiben. Dazu ist es aber notwendig die Metriken einer laufenden Kontrolle auf Aktualität und Richtigkeit zu unterziehen. Zu guter Letzt ist zu beachten, dass eine Metrik nur ihren Sinn erfüllen kann, wenn sie auch von möglichst vielen **Stakeholdern** genutzt wird. Demzufolge ist es ratsam, Metriken zu generieren, die in möglichst vielen Bereichen eingesetzt werden können. Damit können Synergieeffekte, wie beispielsweise eine effektive Generierung der Metriken und eine verbesserte Kommunikation der Bereiche, genutzt werden, [56].

Aufgrund dieser Faktoren lässt sich eine Strategie zur Entwicklung geeigneter Metriken für Unternehmen formulieren, [57]:

1. Spezifizieren der Unternehmens- und Projektziele, die mithilfe der Metrik verbessert werden sollen,
2. die Ziele zu den relevanten operativen Daten zurückverfolgen,
3. eine übersichtliche Plattform zur Präsentation und Interpretation der Daten, in Bezug auf die Ziele, zur Verfügung stellen.

Eine sehr verbreitete Methodik zur Entwicklung von Metriken ist der „Goal-Question-Metrics-Ansatz (GQM-Ansatz)“ entwickelt von Victor Basili, [58]. Die Methodik verfolgt einen strengen „top-down Ansatz“, indem von Zielen ausgehend, zuerst Fragen formuliert und dann Metriken abgeleitet werden. Diese Metriken dienen danach dazu die Fragen zu beantworten. Der GQM-Ansatz besteht demnach aus drei Stufen (siehe Abbildung 9), [58]:



1. **Ziele (Goals)** definieren, die für das Unternehmen oder Projekt relevant sind,
2. **Fragen (Questions)** aus den Zielen ableiten,
3. **Metriken (Metrics)** und die dazu nötigen Daten ermitteln, die zur Beantwortung der Fragen notwendig sind.

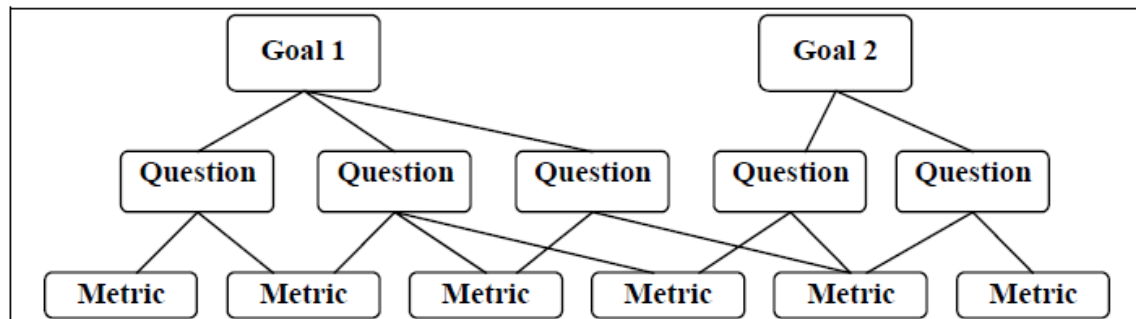


Abbildung 9: Die drei Stufen der GQM-Methode, [58]

Die erste Stufe, die Definition der Ziele, ist eine der wichtigsten, denn es gilt zu entscheiden, welche Bereiche eines Unternehmens oder Projektes genauer betrachtet und verbessert werden sollen. Demnach sollten die definierten Ziele eng verbunden mit den Unternehmenszielen sein, um einen Interessenskonflikt so weit wie möglich auszuschließen. Ebenso ist es entscheidend, Ziele zu definieren, die in der aktuellen Situation relevant sind. Es macht keinen Sinn, Ziele für einen Bereich zu formulieren, der nicht verbessert werden soll oder einen minimalen Einfluss auf den Erfolg des Unternehmens hat. In der nächsten Stufe müssen Fragen formuliert werden, die es erlauben, das Erreichen der Ziele zu messen. Zu diesem Zweck sind zusätzlich Fragen sinnvoll, die die Beantwortung des aktuellen Status zum Ziel haben. Den letzten Schritt bildet die Auswahl von Messungen und Metriken, die dazu beitragen, die Fragen quantitativ zu beantworten. Die ausgewählten Metriken bieten danach einen objektiven Blick auf den Fortschritt zur Erfüllung des Zieles. Demnach sind die Metriken durch den GQM-Ansatz eng mit den Zielen verbunden, was die Relevanz der Metriken sicherstellt, [21].

### 2.3.5 Aggregation und Repräsentation von Software-Metriken

Metriken, insbesondere Code-Metriken, werden für gewöhnlich auf der Mikroebene (Methoden, Klassen, Module) gewonnen. Mit den steigenden Anforderungen der Qualitätssicherung auf der Gesamtsoftwareebene ist es daher notwendig, die verfügbaren Metriken aus der Mikroebene zu kombinieren, um eine Vorstellung über den Zustand der Gesamtsoftware zu erhalten. Beliebte Aggregationstechniken sind statistische Messungen wie Mittelwert, Median oder Summe. Der Hauptvorteil von diesen Techniken ist die universelle Einsetzbarkeit. Der Mittelwert wird immer gleich berechnet, egal welche Metriken verwendet werden. Jedoch liefert dabei nicht jede Metrik verwertbare Ergebnisse. Wenn zum Beispiel eine Metrik keiner Standardverteilung folgt, können Extremwerte abgeschwächt und dadurch übersehen werden. Tabelle 4 zeigt die Anzahl der Codezeilen von vier Methoden (A bis D) in zwei unterschiedlichen Projekten. Wenn man davon ausgeht, dass eine geringere Anzahl von Codezeilen pro Methode bevorzugt wird, schneidet Projekt 2 besser ab als Projekt 1. Jedoch ist nicht ersichtlich, dass es sich bei Methode A in Projekt 2 um eine Ausnahme handelt. Das kann wiederum negative Auswirkungen auf die Qualität des Projektes haben, obwohl der arithmetische Mittelwert ein besseres Ergebnis liefert. Um auf dieses Problem einzugehen zu können, kann auf einen gewichteten Mittelwert zurückgegriffen werden, [59], [60].

Tabelle 4: Anzahl der Codezeilen von vier Methoden in zwei unterschiedlichen Projekten, vgl. [59]

Methoden	Projekt 1	Projekt 2
A	24	71
B	25	9
C	27	10
D	24	8
<b>Mittelwert</b>	<b>25</b>	<b>24,5</b>

Um kritischen Methoden oder Metriken mehr Aussagekraft und Gewicht im Mittelwert einzuräumen, können diese mit Gewichtungsfaktoren versehen werden. Das schafft jedoch ein neues Problem das in Tabelle 5 veranschaulicht wird. Es zeigt die Anzahl der Codezeilen von vier Methoden (A bis D) in zwei Projekten, auf die ein gewichteter Mittelwert angewandt wird. Die Gewichte sind:  $[0, 35] \rightarrow *1$ ;  $]35, 70] \rightarrow *3$ ;  $]70, 160] \rightarrow *9$ ;  $]160, \infty[ \rightarrow *27$ . Der gewichtete Mittelwert (gm) von Projekt 1 beträgt 222,75. Obwohl in Projekt 2 die Anzahl der Codezeilen in den Methoden A, B und C verringert wurde, ist der gewichtete Mittelwert mit 259,53 höher als der von Projekt 1. Der höhere gewichtete Mittelwert deutet daraufhin, dass die Qualität der Software abgenommen hat, obwohl sie in Wahrheit durch eine Reduzierung der Codezeilen verbessert wurde, [59], [60].

Tabelle 5: Zwei Projekte mit gewichteten Mittelwert der Anzahl der Codezeilen, vgl. [59]

Methoden	Projekt 1			Projekt 2		
	LOC	Gewicht	gew. LOC	LOC	Gewicht	gew. LOC
A	30	1	30	25	1	25
B	50	3	150	30	1	30
C	70	9	630	50	3	150
D	300	27	8100	300	27	8100
		sum=40	gm= <b>222,75</b>		sum=32	gm= <b>259,53</b>

Neben der Verwendung von Mittelwert, Median, und so weiter gibt es noch die Möglichkeit, die Verteilung für jede Metrik manuell festzulegen. So ist zum Beispiel die Softwaregröße normal logarithmisch verteilt. Obwohl dies eine genauere und differenziertere Bearbeitung der Metriken erlaubt, ist der Nachteil, dass die Auswahl der Verteilung mit jeder neu hinzukommenden Metrik neu evaluiert werden muss. Ebenfalls ist die Auswahl der richtigen Verteilung nicht eindeutig, so kann die Softwaregröße entweder durch eine normale logarithmische Verteilung oder durch eine Pareto-Verteilung dargestellt werden. Als Reaktion auf die Herausforderung, verlässliche Resultate unter schiefen Verteilungen zu ermitteln, werden immer öfter Aggregationstechniken aus der Ökonometrie (Ungleichheits-Index) entlehnt. Der Grund hierfür ist, dass sowohl in der Ökonometrie als auch in der Softwareentwicklung sehr schiefe Verteilungen auftreten. Ein Problem der Verwendung von ökonomischen Aggregationsverfahren ist, dass diese stark auf Ungleichheit ausgerichtet sind, also würde eine Menge aus gleich schlechten Werten ein positives Ergebnis liefern. Bekannte ökonomische Aggregationstechniken sind Gini, Theil und Hoover. Zusammenfassend kann festgehalten werden, dass die Art der Aggregation immer auf den Anwendungsfall abgestimmt werden muss und, dass es zurzeit keine Aggregationstechnik gibt, die allen Anforderungen entsprechen würde, [59], [60].

Ebenso wichtig wie die Aggregation der Metriken ist die Repräsentation der Ergebnisse. Damit Metriken verwendet und interpretiert werden können, müssen sie übersichtlich und leicht

verständlich dargestellt werden. Dabei sind komplexe Dashboards, die jede Information auf einem einzelnen Schirm enthalten, oft ungeeignet. Erstens hat nicht jeder Stakeholder Interesse an jeder einzelnen Information und zweitens ergibt ein komplexes Dashboard einen hohen Implementierungs- und Konfigurationsaufwand. Die Prämisse bei der Entwicklung eines Repräsentationssystems sollte deshalb lauten: „Start small and then scale out.“. Mehrere kleine, übersichtliche Reports sind meistens hilfreicher als ein riesiges Dashboard. Die Tendenz geht des Weiteren in die Richtung neben einfachen Darstellungsformen auch wenige eindeutige Farben zu verwenden. Oftmals kommt die sogenannte Ampellogik zum Einsatz. Zum Beispiel Rot = hohes Risiko, Gelb = mittleres Risiko, Grün = geringes Risiko. Oder es werden nur zwei Farben (rot und grün) verwendet, um einen akzeptablen und einen inakzeptablen Zustand zu unterscheiden. Die Verwendung von wenigen eindeutigen Farben hilft den Stakeholdern, sich schnell einen Überblick über ihre, für sie interessanten, Kennzahlen zu machen. Für die Repräsentation der Ergebnisse stehen des Weiteren mehrere Arten von Diagrammen zur Auswahl. Die Wahl des passenden Diagramms richtet sich stark danach, welche Vergleiche angestrebt werden. Abbildung 10 zeigt eine Auswahl von Diagrammartentypen zusammen mit den möglichen Vergleichen, [56].



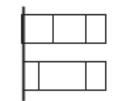
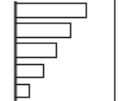
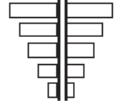

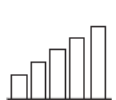
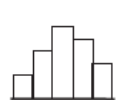
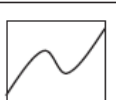
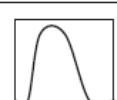
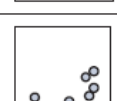

	Struktur	Rangfolge	Zeitreihe	Häufigkeit	Vergleich
Kreisdiagramm					
Balkendiagramm					
Säulendiagramm					
Histogramm					
Streuungs-/Punkte- diagramm					

Abbildung 10: Diagrammartentypen und ihre Anwendungsmöglichkeiten, [56]

**Kreisdiagramme** bestehen aus einem Kreis, in dem einzelne Kategorien als „Stücke“ des Kreises dargestellt werden. Die Größe des Stücks bestimmt die relative Häufigkeit der Kategorie zur Gesamtmenge. Kreisdiagramme können gut eingesetzt werden, um relative Beziehungen darzustellen. Durch die Aufteilung des Kreises in Stücke wird das Kreisdiagramm auch oft als Tortendiagramm bezeichnet. **Balkendiagramme** bilden die einfachste Darstellungsform. Die Länge der Balken ist entweder proportional zur absoluten (100%) oder relativen Häufigkeit. Durch die Einfachheit sind Balkendiagramme vielseitig einsetzbar. Sehr gut eignen sie sich zur Veranschaulichung von Rangfolgen. **Säulendiagramme** sind den Balkendiagrammen sehr ähnlich. Der einzige wirkliche Unterschied ist, dass Balkendiagramme horizontal anwachsen, während Säulendiagramme die Häufigkeiten vertikal darstellen. Säulendiagramme eignen sich gut, um wenige Ausprägungen darzustellen, da ansonsten die Übersichtlichkeit schnell verloren gehen kann. Die

einzelnen Säulen stellen, wie beim Balkendiagramm, entweder die absolute oder die relative Häufigkeit dar. Je nach Anwendung können auch gestapelte, gruppierte oder überlappende Säulendiagramme eingesetzt werden. **Histogramme** sind an die Balkendiagramme angelehnt und werden verwendet, um viele Daten darzustellen. Dabei müssen die Daten zuerst Klassen zugeteilt werden. Danach bieten Histogramme eine gute Darstellung über die Verteilung der Daten in den jeweiligen Klassen. Insbesondere zeitliche Verläufe, bei denen die Daten in Wochen oder ähnliches eingeteilt werden, können so gut dargestellt werden. **Streuungs- oder Punktdiagramme** werden zur Darstellung von zweidimensionalen Ergebnissen verwendet. Insbesondere lassen sich damit Abhängigkeiten und Korrelationen zwischen zwei verschiedenen Wertepaaren darstellen. Es werden dabei zwei verschiedene Merkmale jeweils auf der X- und Y-Achse aufgetragen und die Datenpunkte entsprechend eingetragen. Streuungsdiagramme können auch eingesetzt werden, um sogenannte „Cluster“ zu identifizieren. „Cluster“ sind Ballungen von oft auftretenden Korrelationen. Als letzte Diagrammart soll hier noch das **Netzdiagramm** erwähnt werden. Dieses Diagramm eignet sich sehr gut zur Darstellung der Entwicklung von Serien anhand zuvor festgelegter Kategorien. Jede Kategorie stellt dabei eine Achse dar und für jede Achse gilt die gleiche Orientierung, das bedeutet, die besseren Werte liegen entweder einheitlich im Zentrum oder am äußeren Rand. Zur richtigen Darstellung des Netzdiagrammes müssen zumindest drei Kategorien existieren, weil ansonsten alle Linien übereinander liegen würden. Die Kategorien werden danach gleichmäßig im Kreis angeordnet und die Werte jeder Serie werden mit Linien verbunden. Zur besseren Übersichtlichkeit sollte für jede Serie eine eigene Farbe verwendet werden (siehe Abbildung 11), [21], [29], [56].

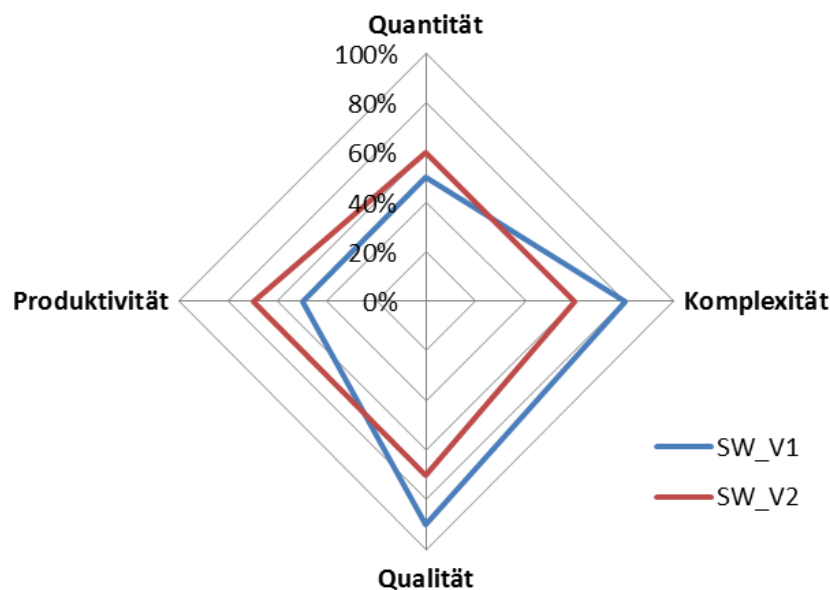


Abbildung 11: Beispiel eines Netzdiagramms mit vier Kategorien und zwei Serien, vgl. [61], [62]

### 3 Analyse des Entwicklungsprozesses

Um potentielle Entwicklungs- und Testdaten zur Evaluierung automotiver Software zu identifizieren wurde eine Analyse des Entwicklungsprozesses bei Magna Powertrain durchgeführt. Anhand eines Beispielprojekts wurde die Software-Architektur der Software bestimmt und das Entwicklungsmodell ermittelt. Die grundlegende Software-Architektur des Beispielprojektes folgt der AUTOSAR-Spezifikation, [15]. Dementsprechend kann das betrachtete Software-System in eine Funktions-Software (FSW), eine Run-Time-Environment (RTE) und eine Basis-Software (BSW) unterteilt werden. In der weiteren Betrachtung wurden die BSW und die RTE vernachlässigt. Die Funktions-Software setzt sich aus einzelnen SW-Komponenten zusammen, die wiederum aus einzelnen Sub-Komponenten bestehen. Diese Sub-Komponenten beinhalten mehrere SW-Module, welche die kleinste logische Einheit in der Software-Architektur darstellen (siehe Abbildung 12).

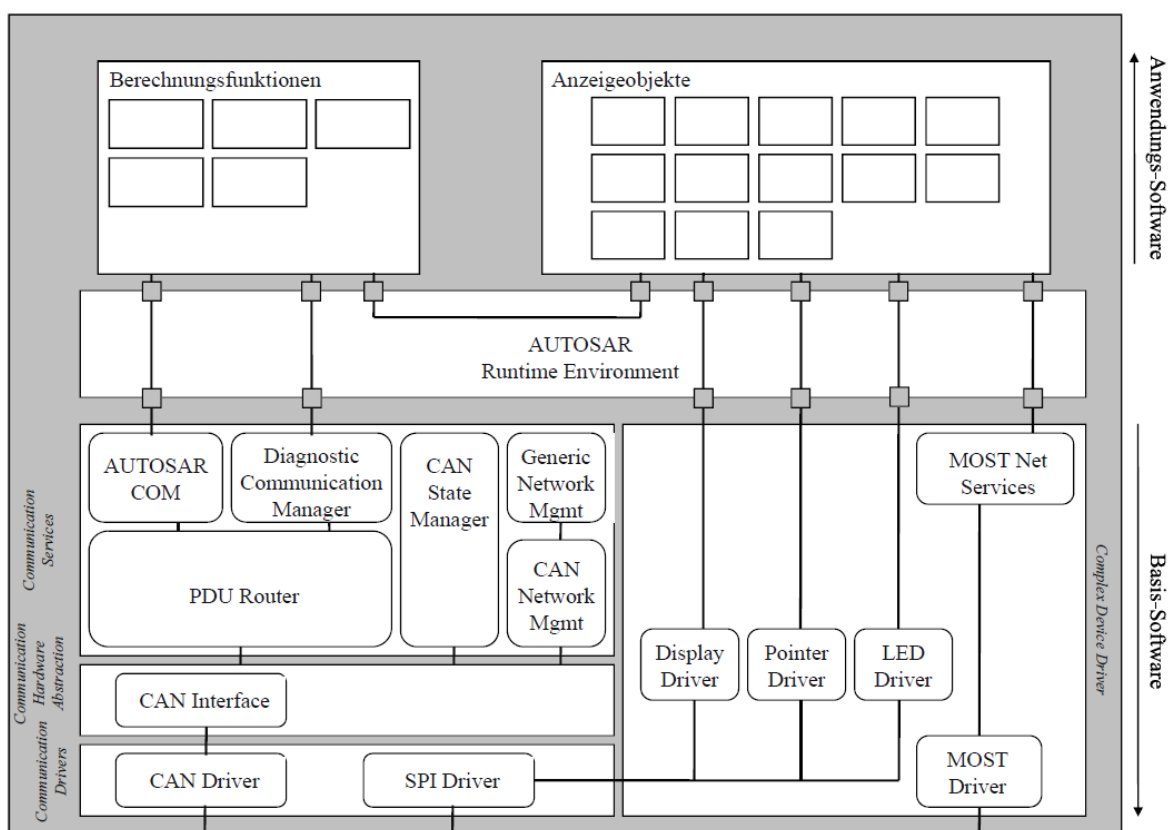


Abbildung 12: Grundlegende Software-Architektur des Projekts, [10]

#### 3.1 Entwicklungsprozess für automotive Software

Für die Entwicklung der automotiven Software kommt ein modifiziertes V-Modell zum Einsatz (siehe Abbildung 13). Wie beim originalen V-Modell (siehe Kapitel 2.2.2) kann der Entwicklungsprozess in eine Anforderungs- und eine Testseite unterteilt werden. Des Weiteren ist das modifizierte V-Modell in drei Ebenen unterteilt, die Modulebene, die Integrationsebene und die Gesamtsoftwareebene. Die Art der Anforderungen und Tests unterscheidet sich je nach Ebene. Auf der Gesamtsoftwareebene werden die Anforderungen auf Systemebene in Form von „System Component Requirements (SCRs)“

spezifiziert und diese durch Gesamtsoftwaretests auf der Testseite verifiziert. Die Anforderungen auf Integrationsebene spezifizieren die Kommunikation zwischen einzelnen Modulen und sind durch die Software-Architektur vorgegeben. Zur Überprüfung der Integration kommen Integrationstests zum Einsatz, die mehrere SW-Module gemeinsam testen. Die Modulebene besitzt den höchsten Detaillierungsgrad und spezifiziert die Anforderungen der einzelnen Software-Module in Form von „Module Requirements (MR)“. Diese werden auf der Testseite unter Zuhilfenahme von Modultests verifiziert.

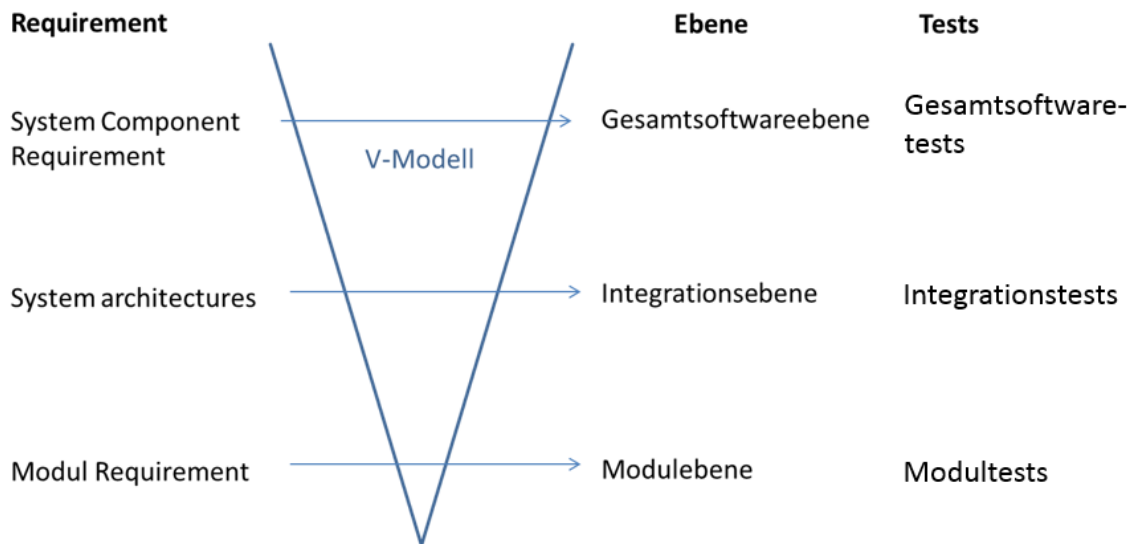


Abbildung 13: Modifiziertes V-Modell zur Entwicklung automotiver Software

Zur Verwaltung der Entwicklungsdaten wird das Informationssystem „MKS Integrity“ eingesetzt, das seit dem Erwerb von MKS durch PTC [63] im Jahr 2011 in dem Produkt „PTC Integrity“ [64] weitergeführt wird. „MKS Integrity“ ist ein „Application lifecycle management“-System, das Werkzeuge für das Anforderungsmanagement, Testmanagement, Änderungsmanagement und die Versionskontrolle zur Verfügung stellt. Das Entwicklungsteam hat somit die Möglichkeit, Arbeitspakete, Anforderungen, Testfälle und Quellcode in einem einzigen System zu verwalten. Für den Entwicklungsprozess automotiver Software sind insbesondere folgende Element-Typen des Systems von Interesse, [65]:

- Anforderungen (System Component Requirements SCRs, Modul Requirements MRs),
- Testfälle (Test Cases TCs) und
- Arbeitspakete, Änderungen und Fehler (Change Issues CI).

Jedes dieser Elemente besitzt einen Status, der den aktuellen Bearbeitungsstand des Elements widerspiegelt. Während des Entwicklungsprozesses kann jeder Element-Typ verschiedene Status durchlaufen, die in Form eines Workflow im System definiert sind, [65].

### 3.1.1 Anforderungen (SCR/MR)

Die Anforderungen eines Projekts teilen sich entsprechend dem V-Modell in SCRs und MRs. Während die SCRs die Anforderungen und Funktionen des Systems auf Systemebene spezifizieren, bestimmen die MRs die Realisierungen auf Modulebene. Ein SCR enthält dabei alle MRs, die zur Realisierung der Funktionalität notwendig sind. Innerhalb des Entwicklungsprozesses können SCRs und MRs folgende Status durchlaufen (siehe Abbildung 14), [66]:

- Requirement New,
- Requirement Specified,
- Requirement Implemented,
- Requirement Closed.

Wird eine neue Anforderung (SCR/MR) erstellt befindet sie sich im Status „Requirement New“. In diesem reicht die Spezifikation der Anforderung nicht aus um weitere Entwicklungsschritte einzuleiten. Ist diese ausreichend spezifiziert und einer ASIL-Stufe zugeteilt, geht sie in den Status „Requirement Specified“ über. In diesem kann mit der Implementierung und dem Erstellen der dazugehörigen Test Cases begonnen werden. Ist die Anforderung vollständig implementiert erhält sie den Status „Requirement Implemented“. Dadurch wird signalisiert, dass diese vollständig implementiert wurde. Demnach kann ab diesem Zeitpunkt auch der Test der Anforderung erfolgen. Der Status „Requirement Closed“ ist für Anforderungen vorbehalten, die für das Projekt nicht mehr relevant sind und deshalb nicht implementiert werden. Die Pfeile in Abbildung 14 zeigen von welchem Status eine Anforderung in einen anderen wechseln kann. So kann beispielsweise eine Anforderung von jedem möglichen Status in „Requirement Closed“ wechseln, während das bei „Requirement Implemented“ nur von „Requirement Specified“ möglich ist, [66].

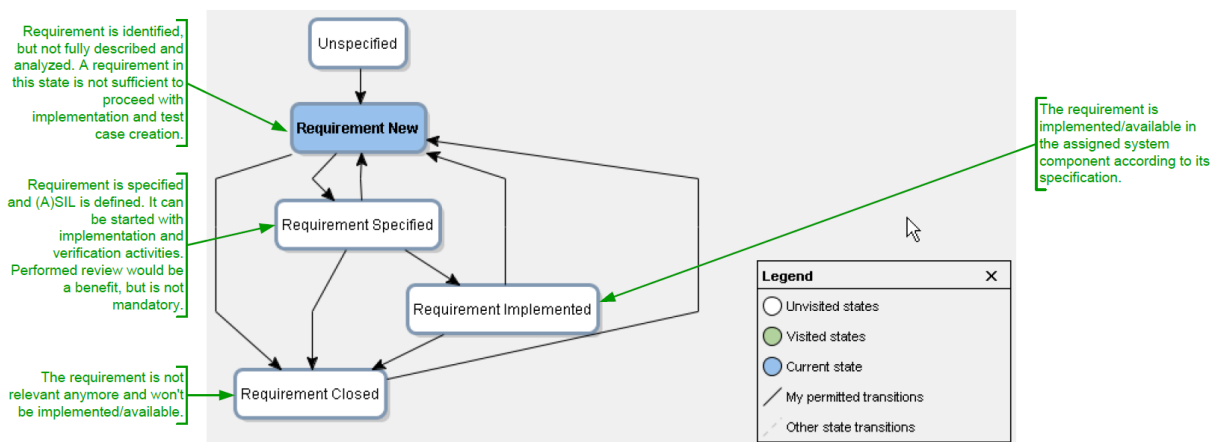


Abbildung 14: Anforderungs-Workflow, [66]

### 3.1.2 Testfälle (TC)

Jede Anforderung muss auf Richtigkeit und Funktionalität überprüft werden. Gemäß dem V-Modell steht deshalb jeder Anforderung auf der Spezifikationsseite zumindest ein Testfall (TC) auf der Testseite gegenüber. Die Implementierung der TCs ist er ab einem Anforderungsstatus von „Requirement Specified“ möglich. Ein Testfall kann während des Entwicklungsprozesses folgende Status durchlaufen (siehe Abbildung 15), [66]:

- TC New,
- TC Specified,
- TC In Work,
- TC Completed,
- TC Retest,
- TC Failed,
- TC Completed with restriction,
- TC Closed.

Der initiale Status eines TC nach dem Erstellen ist „TC New“. Erst nachdem alle notwendigen Informationen im System ergänzt wurden kann mit dem Status „TC Specified“ fortgefahren werden. In diesem verbleibt der TC solange bis die dazugehörige Anforderung auf „Requirement Implemented“ gesetzt wurde. Danach kann mit der Implementierung begonnen werden was durch den Status „TC In Work“ abgebildet wird. Nach der Implementierung ist der TC bereit ausgeführt zu werden und wird aufgrund des Ergebnisses auf „TC Completed“ oder „TC Failed“ gesetzt. „TC Completed“ steht dafür, dass keine Fehler gefunden wurden. Bei „TC Failed“ wurde ein Fehler entdeckt und es wird automatisch ein Arbeitspaket (CI) erstellt. Stellt sich bei einer genaueren Inspektion des Fehlers heraus, dass dieser aufgrund eines anderen Umstandes und nicht direkt durch den TC aufgetreten ist, kann dieser auf den Status „TC Completed with restriction“ gesetzt werden. Von dem Status „TC Completed“, „TC Failed“ und „TC Completed with restriction“ kann der TC jederzeit erneut ausgeführt werden, was durch „TC Retest“ abgebildet wird. Von dort aus kann wiederum, je nach Ergebnis, der Status „TC Completed“ oder „TC Failed“ folgen. „TC Closed“ ist für nicht relevante TC die im Projekt nicht ausgeführt werden, [66].

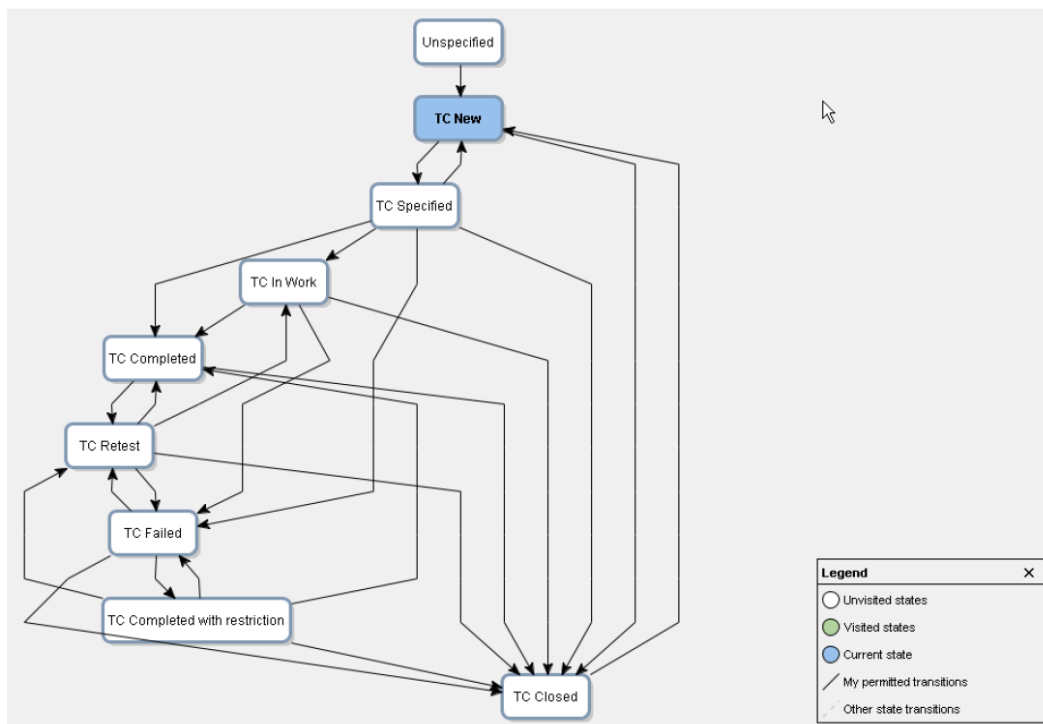


Abbildung 15: Testfall-Workflow, [66]

### 3.1.3 Arbeitspakete, Änderungen und Fehler (CI)

Arbeitspakete, Änderungen und Fehler werden im Informationssystem als ein Element-Typ (Change Issue CI) abgebildet. Um welchen konkreten Typ es sich handelt ist über eine Klassifizierung innerhalb des Elements ersichtlich. Ein CI stellt ein konkretes Arbeitspaket für das Projekt dar und dient zur Planung innerhalb des Projektes. Ein TC im Status „TC Failed“ erzeugt beispielsweise automatisch ein CI mit der Klassifizierung „Fehler“ um die Behebung des Fehlers in den Projektplan zu integrieren. Auf dieselbe Weise wird die Umsetzung von SCRs und MRs über CIs geplant. Folgende Status kann ein CI durchlaufen (siehe Abbildung 16), [66]:

- Change New,
- Change Accepted,



- Change CCB OK,
- Change CCB NOK,
- Change Rejected,
- Change Accepted,
- Change Planned,
- Change Failed,
- Change Implemented,
- Change Tested,
- Change Completed,
- Change Closed.

Wird ein neues CI erstellt befindet es sich zuerst im Status „Change New“. Nach einer ersten Analyse wird es auf den Status „Change Analyzed“ gesetzt. In diesem Status muss eine Expertengruppe über das weitere Vorgehen entscheiden. Diese Gruppe wird als „Change Control Board“ (CCB) bezeichnet. Je nachdem ob von dem CCB ein weiteres Handeln beschlossen wird geht das CI in den Status „Change CCB OK“, für eine weitere Bearbeitung, oder „Change CCB NOK“, wenn keine weitere Bearbeitung des Cis beschlossen wird, über. „Change CCB NOK“ bedeutet dabei, dass das CI vernachlässigt werden kann oder sich um keinen wirklichen Fehler handelt, sondern dieser auf andere Einflüsse zurückzuführen ist. In diesem Fall wird das CI im nächsten Schritt abgelehnt (=„Change Rejected“). Falls laut CCB jedoch ein weiteres Handeln notwendig ist, es akzeptiert und in das Projekt eingeplant („Change Accepted“ und „Change Planned“). Tritt während der Umsetzung des CI ein Fehler auf oder stellt sich heraus, dass es nicht umsetzbar ist, wird es auf den Status „Change Failed“ gesetzt. Bei einer erfolgreichen Implementierung lautet der Status „Change Implemented“ und das CI ist bereit getestet zu werden. Ist der Test erfolgreich wird das CI auf „Change Tested“ und in weiterer Folge auf „Change Completed“ gesetzt. Der Status „Change Closed“ ist wiederum für nicht mehr relevante Elemente vorgesehen, [66].

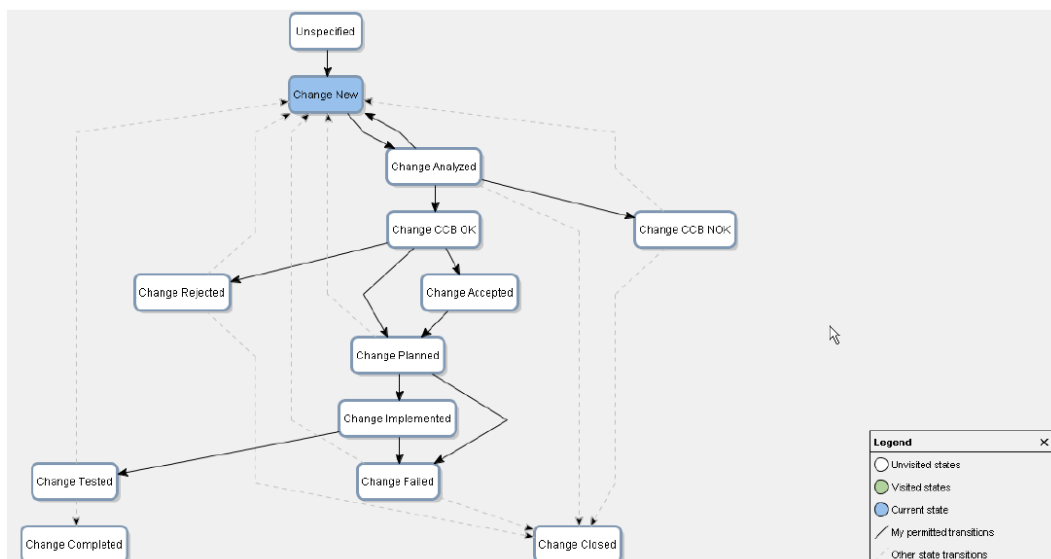


Abbildung 16: Arbeitspaket-Workflow, [66]

## 4 Analysemethoden und deren Machbarkeit auf Basis von Entwicklungs- und Testdaten

Auf Grundlage der Analyse des Entwicklungsprozesses und der relevanten Entwicklungs- und Testdaten wurde ein Idealkonzept für Analysemethoden entwickelt und dessen Machbarkeit überprüft. Aus diesen Erkenntnissen wurde daraufhin eine detaillierte Spezifikation der Analysemethoden für verschiedene Interessensgruppen erstellt.

### 4.1 Idealkonzept

Das Idealkonzept geht bewusst von einer 100% Verfügbarkeit aller Daten aus um alle möglichen Analysemethoden aufzuzeigen. Besonderes Augenmerk wurde dabei auf folgende Anforderungen gelegt:

- Zuordnung der Entwicklungs- und Testdaten zu Ebenen des V-Modells.
- Auswertung zeitlicher Verläufe der Entwicklungs- und Testdaten.
- Einsatz von Software-Metriken zur Quantifizierung der automotiven Software.

#### 4.1.1 Zuordnung der Entwicklungs- und Testdaten zu Ebenen des V-Modells

In Kapitel 3.1 wurde der Entwicklungsprozess für die automotive Software untersucht und drei verschiedene Ebenen innerhalb des V-Modells identifiziert. Jede dieser Ebenen repräsentiert einen speziellen Bereich in der Entwicklung und besitzt unterschiedliche Arten von Anforderungen und Testfällen (siehe Abbildung 13). Während der Entwicklung ist es wichtig die Entwicklungs- und Testdaten einer Ebene isoliert betrachten zu können. Zum Beispiel um die Fehlerlokalisierung und Behebung zu unterstützen. Zugleich ermöglicht dies Verbesserungspotentiale in den einzelnen Ebenen zu identifizieren. Das Idealkonzept geht davon aus, dass die Daten eindeutig auf die Ebenen zugeordnet werden können.

#### 4.1.2 Auswertung zeitlicher Verläufe der Entwicklungs- und Testdaten

Für eine genaue Evaluierung der automotiven Software ist die Betrachtung zeitlicher Verläufe der Entwicklungs- und Testdaten unerlässlich. Auf diese Weise kann die Entwicklung genau nachvollzogen und gesteuert werden. Die zeitliche Entwicklung der Status von Requirements ist beispielsweise ein Indikator für die funktionale Vollständigkeit der Software. Während durch Betrachtung von Test Cases der Testprozess evaluiert werden kann. Mithilfe des Verlaufs der gefundenen Fehler können wiederum Fehlerhäufigkeiten analysiert werden. Abbildung 17 zeigt eine beispielhafte Auswertung von Verläufen aufgrund von synthetischen Daten.

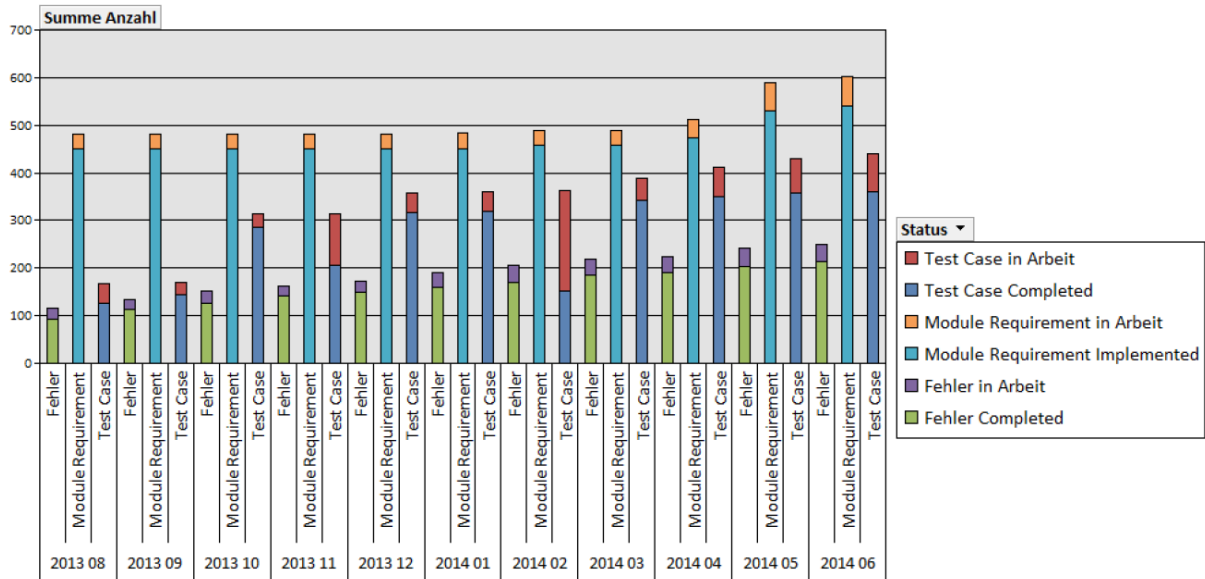


Abbildung 17: Beispielhafter Verlauf von Entwicklungs- und Testdaten

#### 4.1.3 Einsatz von Software-Metriken zur Quantifizierung automotiver Software

Eine weitere Möglichkeit zur Bewertung der automotiven Software bietet der Einsatz von Software-Metriken. Diese können eingesetzt werden um die Eigenschaften der automotiven Software quantifizierbar darzustellen. Hierzu werden Software-Metriken, die während des Entwicklungsprozesses generiert werden, gesammelt und einer von vier Kategorien zugeordnet. Durch die Anwendung eines Gewichtungsschemas, innerhalb der einzelnen Kategorien, kann ein quantifizierbarer Wert ermittelt werden. Legt man die Werte unterschiedlicher Software-Versionen übereinander können Trends in der Entwicklung veranschaulicht werden (siehe Abbildung 18). Die Kategorien umfassen im Idealkonzept:

- Quantität,
- Komplexität,
- Qualität,
- Produktivität.

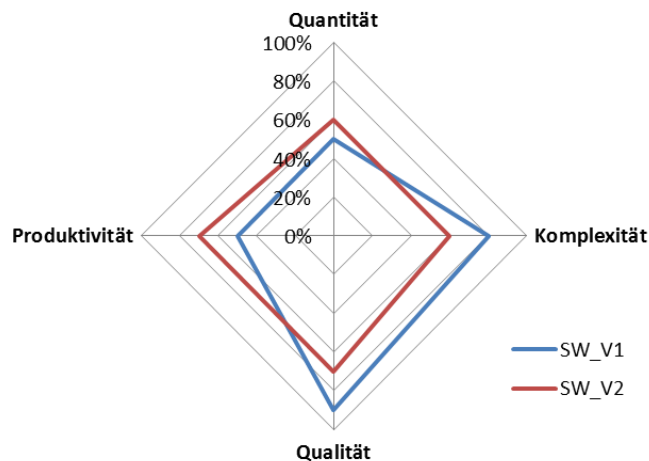


Abbildung 18: Bewertung von Software-Versionen mithilfe von Software-Metriken, vgl. [61], [62]

## 4.2 Überprüfung der Machbarkeit des Idealkonzeptes aufgrund vorhandener Entwicklungs- und Testdaten

Um die umsetzbaren Methoden zur erweiterten Analyse der Entwicklungs- und Testdaten identifizieren zu können, wurde eine umfassende Machbarkeitsstudie durchgeführt. Ziel dieser ist es die Durchführbarkeit des Idealkonzeptes zu untersuchen. Dafür wurden folgende Teilaspekte betrachtet:

1. Wie können die Entwicklungsdaten (MR, SCR, TC, CI) auf die einzelnen Entwicklungs-Ebenen zugeordnet werden?
2. Wie können Verläufe der Entwicklungsdaten mithilfe von MKS ermittelt werden?
3. Welche Metriken stehen zur Analyse und Bewertung der Software zur Verfügung?

### 4.2.1 Entwicklungs- und Testdatenanalyse

Um die Machbarkeit des Idealkonzeptes bewerten zu können wurde zuerst eine Analyse der vorhandenen Entwicklungs- und Testdaten anhand eines Beispielprojektes aus dem MKS durchgeführt. Besonderes Augenmerk wurde auf die Anforderungen (MR, SCR), Testfälle (TC) und Arbeitspakete/Fehler (CI) gelegt. Als Ergebnis wurde eine Auswertung über den Hinterlegungsgrad relevanter Attribute erstellt, die als Grundlage für die weitere Machbarkeitsstudie dient. Die betrachteten Attribute umfassen, [65]:

- SW Module = Das zugeteilte SW-Modul in der Software-Architektur.
- Sub Component = Die zugeteilte Sub-Komponente in der Software-Architektur.
- SW Release = Der SW-Release, in dem sich das Element gerade befindet.
- Planned SW Release = Der geplante SW-Release, zu dem das Element fertiggestellt sein soll.
- Completion SW Release = Der SW-Release, zu dem das Element fertiggestellt wurde.
- Change Severity = Die Schwere, die dem Element zugeteilt wurde, z.B. Fehlerschwere.

In Abbildung 19 ist der Hinterlegungsgrad der Attribute in den Daten ersichtlich. Zu beachten ist hierbei, dass die Hinterlegung der SW-Releases nur in den Change Issues erfolgt. Aus diesem Grund wurden in weiterer Folge auch die Beziehung der Daten untereinander untersucht.

Attribut	MR	SCR	TC	CI
SW Module	99,9 %	-	32,6 %	39,5 %
Sub Component	100 %	45,8 %	48,9 %	92 %
SW Release	-	-	-	81,3 %
Planned SW Release	-	-	-	96,6 %
Completion SW Release	-	-	-	93,4 %
Change Severity	-	-	-	80,3 %

Abbildung 19: Hinterlegungsgrad der Attribute

Abbildung 20 zeigt die Beziehungen der Elemente untereinander. Daraus ist ersichtlich, dass Requirements (MR, SCR) mit den dazugehörigen Test Cases verbunden sind. Des Weiteren besitzen die Change Issues eine Verknüpfung zu den Requirements, sowie zu den Test Cases. Durch die

Auswertung der Beziehungen stellten sich die Arbeitspakete als zentrales Planungselement heraus, mit deren Hilfe die indirekte Zuteilung der Requirements und Test Cases zu Software-Releases erfolgen kann.

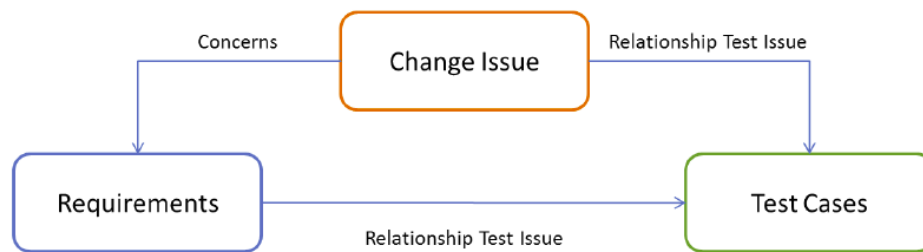


Abbildung 20: Beziehungen zwischen den Daten

#### 4.2.2 Machbarkeitsanalyse der Zuordnung zu den Entwicklungs-Ebenen

Das Idealkonzept sieht eine Zuordnung der Entwicklungs- und Testdaten zu den einzelnen Entwicklungs-Ebenen im V-Modell vor. Ausgehend von der Entwicklungs- und Testdatenanalyse wurde folgendes Konzept zur Zuordnung entwickelt. Die MRs enthalten nur Anforderungen für die Modul-Ebene und können deshalb vollständig dieser Ebene zugeordnet werden. Auf Gesamtsoftware-Ebene werden die Anforderungen in Form von SCRs verwaltet. Es gibt jedoch auch SCRs die eine Schnittstelle definieren und deshalb der Integrations-Ebene zugeordnet werden müssen. Dadurch teilen sich diese in SCRs mit Schnittstellendefinition, diese werden der Integrations-Ebene zugeordnet, und alle restlichen SCRs ohne Schnittstellendefinition, diese werden der Gesamtsoftware-Ebene zugeordnet. Bei den Change Issues werden für die Zuordnung die hinterlegten Sub-Komponenten herangezogen. Ist nur eine einzige Sub-Komponente hinterlegt ist das dazugehörige CI der Modul-Ebene zuzuordnen. Sind mehrere Sub-Komponenten vorhanden der Integrations-Ebene und sind alle Sub-Komponenten der Architektur in einem CI hinterlegt wird dieses der Gesamtsoftware-Ebene zugeordnet. Die Zuordnung der Test Cases erfolgt über die hinterlegte Test Plattform in den Daten. Gemäß dem V-Modell werden TCs mit der Test Plattform „MiL“ oder „SiL“ der Modul-Ebene zugeordnet, während jene mit der Test Plattform „HiL“ der Integrations- und Gesamtsoftware-Ebene angehören (siehe Tabelle 6).

Tabelle 6: Zuordnung der Elemente zu Entwicklungs-Ebenen

Entwicklungs-Ebene	Anforderungen	Testfälle	Arbeitspakete / Fehler
Modul-Ebene	Modul-Anforderungen	Testplattform: <b>MiL, SiL</b>	Bezug auf <b>eine</b> einzige Sub-Komponente
Integrations-Ebene	System-Anforderungen (Schnittstellen-Definition)	Testplattform: <b>HiL</b>	Bezug auf <b>mehrere</b> Sub-Komponenten
Gesamtsoftware-Ebene	System-Anforderungen (ohne Schnittstellen-Definition)	Testplattform: <b>HiL</b>	Bezug auf <b>alle</b> Sub-Komponenten

Bei genauerer Betrachtung fällt auf, dass bei TCs nicht zwischen Integrations- und Gesamtsoftware-Ebene unterschieden werden kann. Aus diesem Grund wurden die drei Ebenen des V-Modells für die Analysemethoden auf zwei reduziert. Hierzu wurde die Integrations-Ebene zu der Gesamtsoftware-Ebene hinzugefügt. Somit können alle TCs mit der Test Plattform „HiL“, alle SCRs und alle Cis mit

mehr als einer Sub-Komponente dieser Ebene zugeordnet werden. Die Modul-Ebenen bleibt unverändert erhalten (siehe Tabelle 7).

Tabelle 7: Zuordnung der Elemente zu Modul- und Gesamtsoftware-Ebene

Ebene	Anforderungen	Testfälle	Arbeitspakete / Fehler
Modul-Ebene	Modul-Anforderungen	Testplattform: <b>MiL, SiL</b>	Bezug auf <b>eine</b> einzige Sub-Komponente
Gesamtsoftware-Ebene	System-Anforderungen	Testplattform: <b>HiL</b>	Bezug auf <b>mehrere/alle</b> Sub-Komponenten

#### 4.2.3 Machbarkeitsanalyse der Auswertung zeitlicher Verläufe

Zur Auswertung von Verläufen ist es notwendig auf historische Entwicklungs- und Testdaten zugreifen zu können. Das Informationssystem MKS verfügt über mehrere Möglichkeiten Daten zu exportieren und auszuwerten. Diese Möglichkeiten wurden in Bezug auf die Verwendbarkeit zur Erstellung von Verläufen überprüft. MKS bietet folgende Abfragemöglichkeiten, [67]:

- „Queries“,
- „Reports“,
- „Charts“,
- „Dashboards“.

Die einfachste Möglichkeit Daten flexibel abzufragen besteht über eine „Query“. Mithilfe dieser können Attribute von Elementen abgefragt werden. Das Ergebnis kann durch Filter und Bedingungen eingeschränkt und angepasst werden. Eine vordefinierte „Query“ kann zudem als Daten-Grundlage für erweiterte Auswertungsmethoden wie „Reports“, „Charts“ und „Dashboards“ dienen. Jedoch können mithilfe von „Queries“ nur die aktuellen Daten und keine historischen abgefragt werden. Hierzu ist die Verwendung eines „Reports“ notwendig. Mittels einem „Report“ können entweder selbst Filter und Bedingungen definiert oder eine bereits vordefinierte Query als Daten-Grundlage verwendet werden. Danach können die ermittelten Daten in verschiedenen Formaten ausgegeben oder exportiert werden. Eine Möglichkeit ist beispielsweise die Ausgabe aufgrund eines definierten XML-Schemas. Dadurch kann ein XML-Dokument erstellt werden und durch eine andere Anwendung verwendet werden. Ein „Report“ kann zudem auch historische Daten der Elemente zu einer angegebenen Zeit ermitteln. Damit kann in weiterer Folge ein Verlauf über den Zustand eines Elements erstellt werden. Charts im MKS können unter gewissen Bedingungen historische Daten erstellen, verfügen aber über ein paar Limitierungen. Grundsätzlich gibt es drei verschiedene Chart-Typen die verschiedene Eigenschaften besitzen. Der erste Chart-Typ erlaubt eine Darstellung von Elementen nur zum aktuellen Zeitpunkt, deshalb kann dieser Chart nicht verwendet werden. Bei dem zweiten Chart-Typ ist es möglich den Verlauf von Elementen über die Zeit darzustellen, jedoch erlaubt dieser keine Filter. Die Verwendung von Filtern ist jedoch für die Verwendung in Verläufen notwendig, zum Beispiel um die Elemente anhand des Status zu filtern. Der letzte Chart-Typ erlaubt das Auswerten der Elemente über die Zeit und unterstützt auch das Zuweisen von Filtern, jedoch sind die auswertbaren Felder nur auf numerische Typen beschränkt. Deshalb ist es zum Beispiel nicht möglich einen Verlauf über die einzelnen Status (=Textfeld) von Elementen zu erstellen. Ein Dashboard ist eine Zusammensetzung aus mehreren „Reports“ und „Charts“. Nachdem die Verwendung von Charts aufgrund der Limitierungen ausgeschlossen werden musste, wurde der Fokus auf die Reports gelegt. Reports bieten, wie bereits zuvor erwähnt, die Möglichkeit Daten eines

Elements zu einem spezifischen Zeitpunkt auszuwerten. Die Auswahl der auszuwertenden Elemente kann direkt über Filter im Report oder vordefinierte Queries erfolgen und bietet somit eine flexible Möglichkeit zur Ermittlung der Daten. Der Nachteil ist, dass die Reports nur einzeln ausgewertet werden können (jeweils zu einem bestimmten Zeitpunkt) und es keine Möglichkeit gibt diese direkt im MKS darzustellen. Aufgrund dieser Einschränkungen wurde entschieden, dass sich die Auswertungsmethoden im MKS nicht für die Darstellung von Verläufen eignen. Aus diesem Grund wurde eine eigene Analyse-Datenbank entwickelt. Auf den Aufbau und die Speisung der Datenbank wird in Kapitel 5.3 genauer eingegangen, [64], [67].

#### 4.2.4 Machbarkeitsanalyse zum Einsatz von Software-Metriken

Zur Bewertung der automotiven Software wurde im Idealkonzept die Verwendung von Software-Metriken angedacht. Aus diesem Grund wurde überprüft welche Metriken während des Entwicklungsprozesses gewonnen und für die weitere Analyse verwendet werden können. Die Software-Metriken werden durch die Programme Polyspace [68] und M-XRAY [69] gewonnen. M-XRAY ermittelt die Metriken direkt aus den Simulink-Modellen, während Polyspace den generierten C-Code analysiert. Die verfügbaren Metriken wurden gesammelt und auf die reduzierten Entwicklungsebenen Modul und Gesamtsoftware (siehe Kapitel 4.2.2) zugeordnet. Danach wurden die einzelnen Metriken einer bestimmten Dimension der Software zugeteilt. Abbildung 21 zeigt die gesammelten Metriken auf Modul-Ebene und Abbildung 22 die Metriken auf Gesamtsoftware-Ebene. Der im Idealkonzept geplanten Dimension der Produktivität konnten keine Metriken zugeordnet werden. Eine Erfassung der Produktivität setzt Zugriff auf genaue Arbeitsstunden und Daten einzelner Mitarbeiter voraus, die in den Entwicklungs- und Testdaten nicht zur Verfügung stehen.

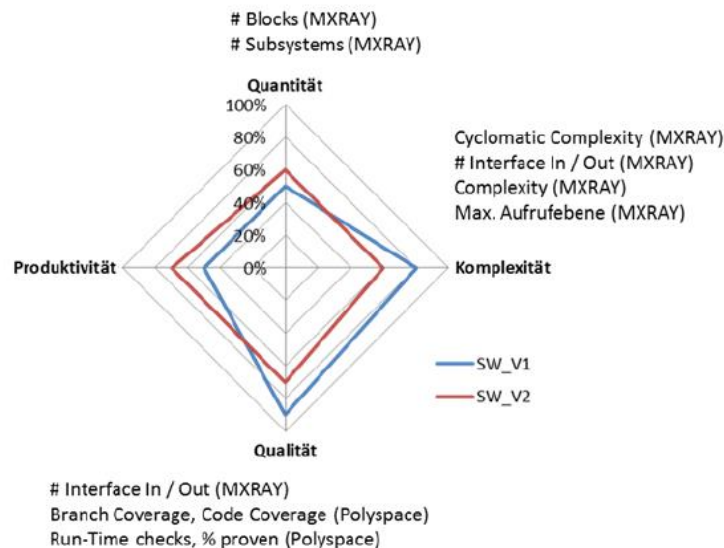


Abbildung 21: Verfügbare Code-Metriken auf Modul-Ebene

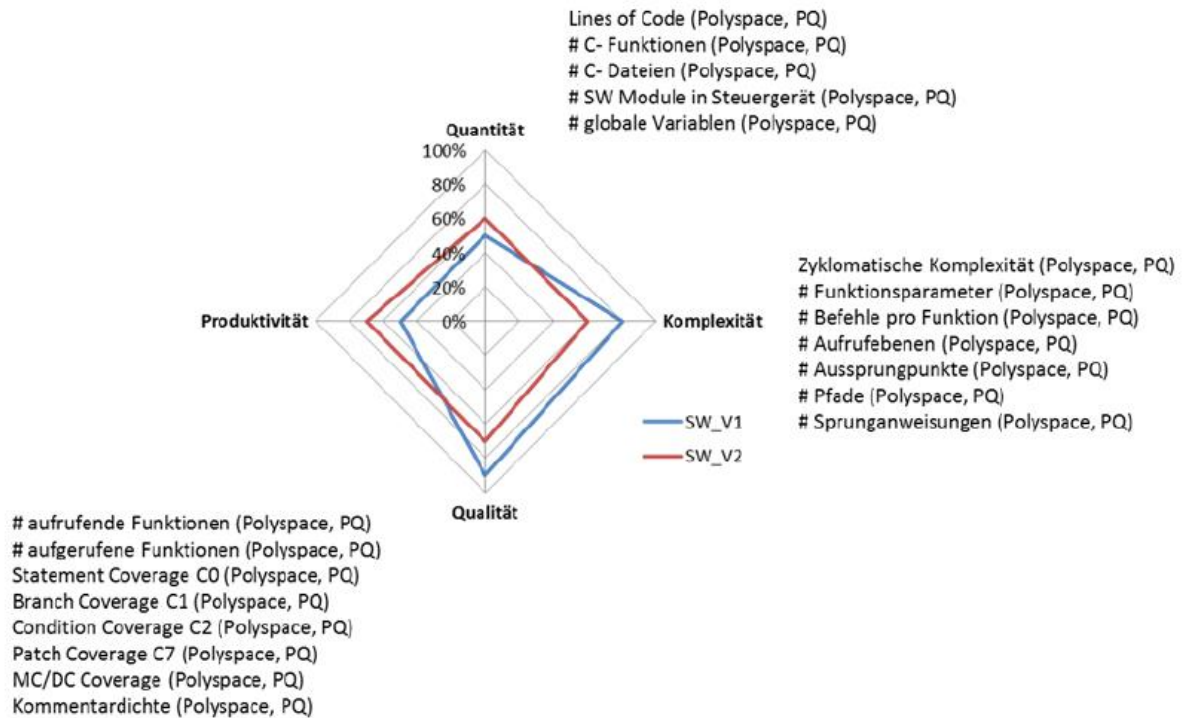


Abbildung 22: Verfügbare Code-Metriken auf Gesamtsoftware-Ebene

Ein Vergleich der verfügbaren Metriken auf Modul- und Gesamtsoftware-Ebene zeigte, dass die M-XRAY Metriken nur für einzelne Module gewonnen werden und diese durch ein paar Code-Coverage Metriken aus Polyspace ergänzt werden. Dahingegen stehen auf Gesamtsoftware-Ebene deutlich mehr Polyspace-Metriken zur Verfügung, jedoch keine Metriken aus M-XRAY. Aufbauend auf diesen Erkenntnissen wurden drei mögliche Strategien zur Einbindung von Software-Metriken für die Analyse entwickelt (siehe Abbildung 23). Die erste Strategie schlägt eine getrennte Betrachtung von den zwei Ebenen vor. Dadurch, dass die Metriken für die Ebenen unterschiedlich generiert werden ist jedoch auch kein direkter Vergleich zwischen ihnen möglich. Strategie 2 geht davon aus, dass die Metriken auf der Modul-Ebene gewonnen werden und dann bis zu der Gesamtsoftware aggregiert werden. Um diese Strategie einsetzen zu können müssen mehr Metriken auf Modulebene ermittelt werden, die daraufhin aggregiert werden können. Die dritte Strategie sieht eine reine Polyspace-Lösung vor. Diese geht davon aus dieselben Metriken auf beiden Ebenen mit demselben Werkzeug, zum Beispiel Polyspace, zu ermitteln. Die Verfügbarkeit von gleichen Metriken auf beiden Ebenen ermöglicht gleichzeitig den Einsatz erweiterter Analysemethoden. Zum Beispiel das Identifizieren der komplexesten Module relativ zur Gesamtsoftware.



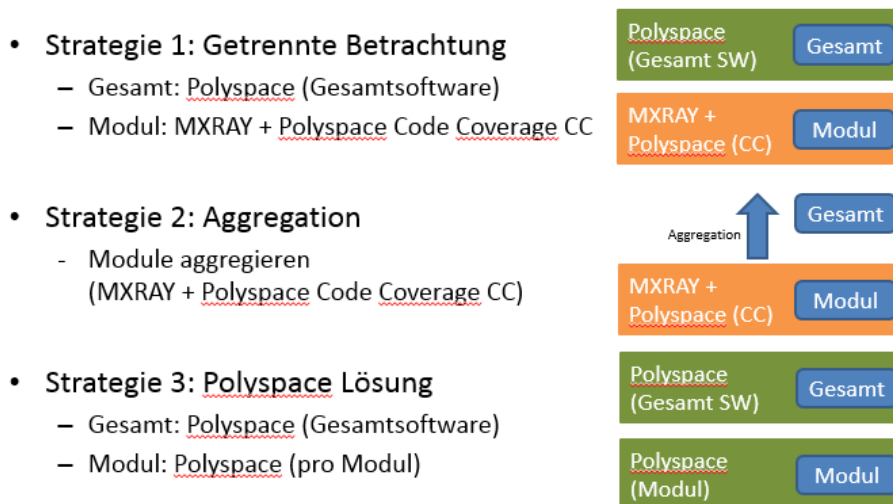


Abbildung 23: Strategien für die Metriken-Nutzung

Eine umsetzbare Anwendung der Metriken wurde auf der Gesamtsoftware-Ebene durch die Kombination von Code-Metriken aus Polyspace und Prozess-Metriken, die aus dem Informationssystem MKS gewonnen werden konnten, realisiert (siehe Abbildung 24). Es werden ausgewählte Code-Metriken aus Polyspace zusammen mit Prozess-Metriken aus MKS auf die drei Dimensionen der Software aufgeteilt. Daraufhin wird eine manuelle Gewichtung der einzelnen Metriken nach Aussagekraft vorgenommen und für jede Dimension ein Wert berechnet. Diese Werte können dann in einem Netzdiagramm dargestellt werden. Werden nun mehrere Software-Releases betrachtet kann auf diese Weise der Verlauf der Software aufgrund von den drei Dimensionen veranschaulicht werden. Die ausgewählten Metriken sowie deren Zuordnung werden in Abbildung 24 genauer dargestellt, [61].

### Quantität

- KLOC (Kilo lines of code): Die Anzahl der Codezeilen in Tausende.
- # C- Funktionen: Die Anzahl aller vorhandenen C-Funktionen.
- # SW Module im Steuergerät: Die Anzahl aller Software-Module im Steuergerät.
- # Anforderungen umgesetzt: Alle Anforderungen, die implementiert wurden.
- # TC ausgeführt: Alle Testfälle, die ausgeführt wurden.

### Qualität

- MC/DC Coverage: Testabdeckung der Software.
- Kommentardichte: Anteil der Kommentare im Verhältnis zu allen Codezeilen.
- # Anforderungen umgesetzt: Die Anzahl der implementierten Anforderungen (funktionale Qualität)
- # Fehler behoben: Die Anzahl der behobenen Fehler.

### Komplexität

- Zyklomatische Komplexität: Maß für die Strukturkomplexität.
- # Funktionsparameter: Anzahl aller Funktionsparameter (Interkonnektivität).
- # Aufrufebenen: Anzahl der Aufrufebenen (Verschachtelung).

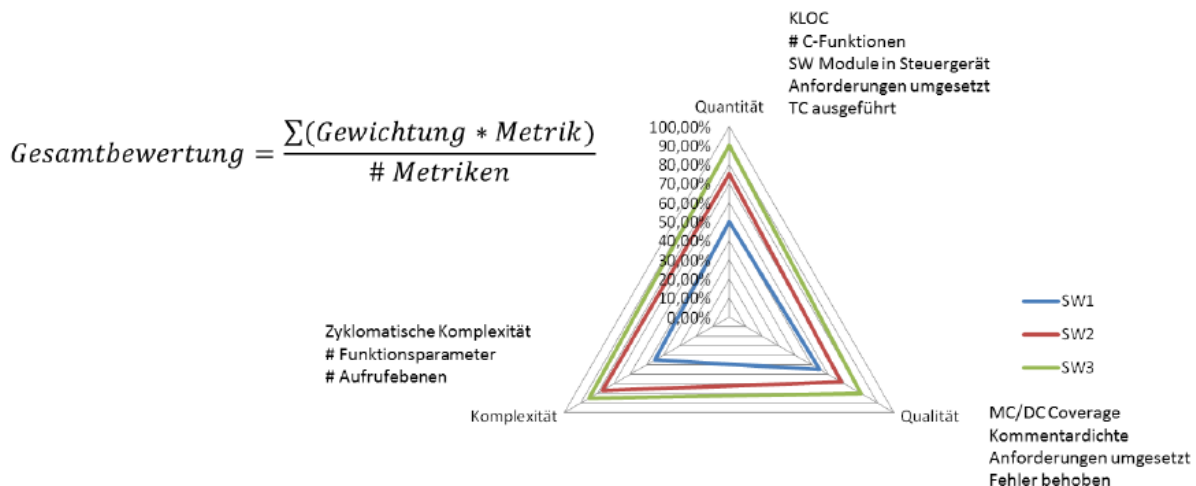


Abbildung 24: Kombination von Code- und Prozess-Metriken auf Gesamtsoftware-Ebene, vgl. [61]

### 4.3 Detaillierte Spezifizierung der Analysemethoden aufgrund relevanter Fragestellungen im Entwicklungsprozess automotiver Software

Nachdem durch die Machbarkeitsstudie die Realisierung des Idealkonzeptes überprüft werden konnte wurden davon ausgehend Analysemethoden entwickelt. Zu diesem Zweck wurden drei Interessengruppen ausgewählt und Fragen in Bezug auf den Entwicklungsprozess identifiziert. Mithilfe dieser Fragen konnten daraufhin maßgeschneiderte Analysemethoden für die einzelnen Interessensgruppen spezifiziert werden. Insgesamt wurden drei Interessensgruppen ausgewählt, [62]:

- Qualitätssicherung (Quality assurance),
- Management,
- Projektmanagement.

Jede dieser Interessensgruppen hat andere Prioritäten und Fragen, welche durch die Analyse von Entwicklungs- und Testdaten beantwortet werden sollen. Deshalb wurden für die Entwicklung der Analysemethoden die Anforderungen und Fragen gesammelt und als Basis für die Entwicklung der Methoden herangezogen. Ein weiterer Vorteil, den die Analyse und Auswertung der Entwicklungs- und Testdaten mit sich bringt, ist, dass die Ergebnisse als Basis für eine objektive Bewertung und Kommunikation über den Zustand der automotiven Software dienen können. Abbildung 25 stellt die drei Interessensgruppen mitsamt ihren dazugehörigen Fragen graphisch dar. Die Qualitätssicherung hat beispielsweise Interesse an dem Verlauf und den Abweichungen, die während der Entwicklung auftreten. Dementsprechend soll die entwickelte Analysemethode Aufschluss über diese Fragen bieten. Ein weiterer interessanter Aspekt für die Qualitätssicherung ist die Identifizierung von Planungs- und Implementierungsspitzen, weil für gewöhnlich solche Spitzen viele Fehler produzieren und dementsprechend das Qualitätsmanagement besonders gefordert ist. Neben der Fehlerreduktion ist auch die Funktionalität ein wichtiges Qualitätsmerkmal und muss daher von der Qualitätssicherung im Auge behalten werden. Die umgesetzten Analysemethoden für die Qualitätssicherung werden im Kapitel 4.3.2 im Detail erläutert. Das Management hat die Qualität der Systemfreigaben und den Gesamtfortschritt des Projektes zu überprüfen. Dementsprechend ist ein Gesamtüberblick über den Erfüllungsgrad aller Anforderungen, der Testabdeckung bei den

Systemfreigaben und die Fehlerbehebungsrate in Bezug auf das Gesamtprojekt von Interesse. Auf die Analysemethoden für das Management wird in Kapitel 4.3.3 genauer eingegangen. Das Projektmanagement ist für den funktionierenden Ablauf des Projektes verantwortlich und ist deswegen an dem Status der einzelnen Arbeitspakete in Bezug auf Verzug und Abweichungen interessiert. Eine übersichtliche Auswertung der Arbeitspakete gibt dem Projektmanagement die Möglichkeit Abweichungen in der Planung früh genug zu erkennen und entsprechende Gegenmaßnahmen einzuleiten. Die dazu entwickelten Analysemethoden werden in Kapitel 4.3.1 beschrieben. Es wird an dieser Stelle darauf hingewiesen, dass alle Abbildungen und Beispiele dieser Masterarbeit anhand synthetischer Daten erstellt wurden und auch keinen zusammenhängenden Bezug zueinander aufweisen, [62].

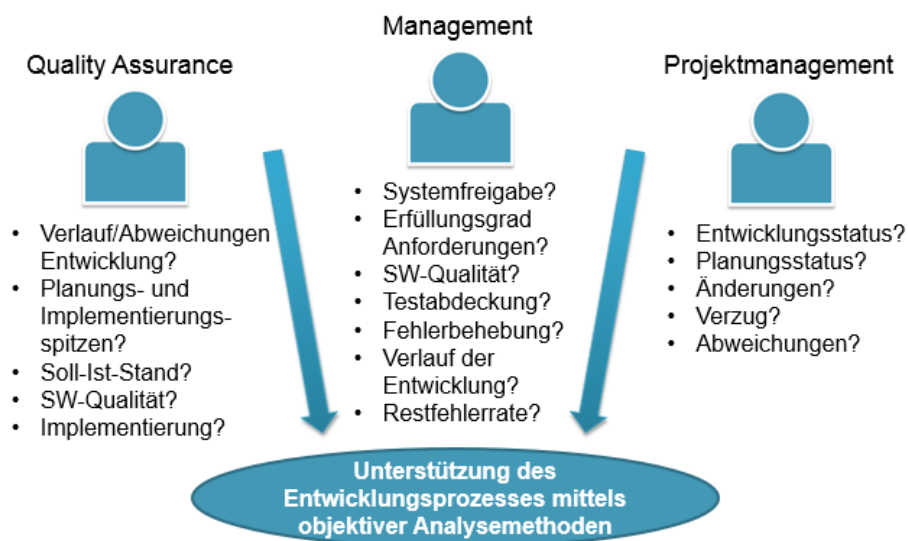


Abbildung 25: Fragen und Ansichten auf die Entwicklungs- und Testdaten, [62]

#### 4.3.1 Analysemethoden für das Projektmanagement

Für das Projektmanagement ist die Übersicht und die Kontrolle der Arbeitspakete entscheidend um Abweichungen im Projekt rechtzeitig erkennen zu können. Die wichtigsten Fragen sind somit die nach dem Entwicklungs- und Planungsstatus des Projektes und welche Arbeitspakete gefährdet sind den geplanten Fertigstellungstermin nicht erreichen zu können. Dementsprechend liegt das Hauptaugenmerk auf der Analyse und Auswertung der Arbeitspakete. Die spezifizierte Analysemethode soll das Projektmanagement bei der Beantwortung folgender Fragen unterstützen, [62]:

1. Was ist der aktuelle Status des Arbeitspaketes?
2. Welcher Ebene (Modul oder Gesamtsoftware) kann das Arbeitspaket zugeordnet werden?
3. Ist das Arbeitspaket aktuell in Verzug, beziehungsweise wie viele Tage verbleiben bis zum Fertigstellungstermin?
4. Welche Status hat das Arbeitspaket zuvor durchlaufen?
5. Wie viele Tage hat das Arbeitspaket in den Status verbracht?
6. Wie oft wurde das Arbeitspaket schon einmal verschoben beziehungsweise um geplant?
7. Wie viele Iterationen hat das Arbeitspaket bereits durchlaufen?

Der aktuelle Status der Arbeitspakete ist im MKS hinterlegt und kann einfach abgerufen werden. Die Zuteilung in eine der zwei Ebenen Modul oder Gesamtsoftware kann, wie bereits in Kapitel 4.2.2

erklärt, anhand der hinterlegten Sub-Komponenten getroffen werden. Ist bei einem Arbeitspaket nur eine einzige Sub-Komponente zugeteilt, wird sie der Modul-Ebene zugordnet. Sind dagegen mehrere Sub-Komponenten hinterlegt, wird das Arbeitspaket auf die Gesamtsoftware-Ebene zugeordnet. Um das geplante Fertigstellungsdatum beziehungsweise den Verzug zu berechnen, kann der geplante Software-Release im Arbeitspaket verwendet werden. Voraussetzung dafür ist, dass eine Zuteilung der Software-Releases auf ein konkretes Datum existiert. Dann kann der Verzug einfach als die Differenz zwischen dem geplanten und dem tatsächlichen Datum berechnet werden. Um den zeitlichen Verlauf der Arbeitspakete darstellen zu können wird auf die im Kapitel 0 beschriebene Analyse-Datenbank zurückgegriffen. 4.1.2 Dieser Verlauf ermöglicht dem Projektmanagement den Werdegang eines Arbeitspakets schnell nachvollziehen zu können. In diesem Zusammenhang ist auch die Anzahl der Tage interessant, die das Arbeitspaket in den verschiedenen Status verbracht hat. Diese ergibt sich aus dem Zeitraum zwischen dem Wechsel von einem Status in einen anderen. Um schnell eine erhöhte Anzahl von Tagen in einem Status sichtbar zu machen, wird für jeden Status ein Mittelwert gebildet, der angibt wie viele Tage die Arbeitspakete durchschnittlich in diesem verbringen. Übersteigt ein Arbeitspaket den Mittelwert eines Status wird die Anzahl der Tage farblich hervorgehoben. Damit ist für das Projektmanagement schnell ersichtlich welche Arbeitspakete ungewöhnlich lange in einem Status verbringen. Eine weitere wichtige Information ist wie oft das Arbeitspaket bereits um geplant, beziehungsweise auf einen anderen Software-Release verschoben wurde. Die vergangenen geplanten Software-Releases können wieder aus der Analyse-Datenbank ermittelt werden. Danach kann gezählt werden wie oft der geplante Software-Release in dem Arbeitspaket geändert wurde. Die konkreten Umplanungen von einem Software-Release auf einen anderen werden in Form einer Zeitleiste visualisiert (siehe Abbildung 28). Dabei macht es für das Projektmanagement einen Unterschied ob ein Arbeitspaket nicht bearbeitet wurde oder während der Umsetzung ein Problem aufgetreten ist. Um Informationen diesbezüglich zu erhalten wird ermittelt wie oft das Arbeitspaket den Workflow durchlaufen hat (siehe Kapitel 3.1.3). Als Indikator für eine volle Iteration dient dabei der Zustand „Change New“. Tritt bei der Umsetzung eines Arbeitspaketes ein Problem auf, wird über den Zustand „Change New“ eine neue Iteration eingeleitet. Die Differenz der Anzahl der Umplanungen zu der Anzahl der Iterationen gibt einen Aufschluss darüber wie oft das Arbeitspaket, ohne ersichtlichen Grund, verschoben wurde. Alle diese Fragen können für das Projektmanagement in einer Auflistung der Arbeitspakete, sowie über das Visualisieren der Statusänderungen und Umplanungen auf einer Zeitleiste beantwortet werden. Abbildung 26 zeigt eine beispielhafte Auswertung der Arbeitspakete anhand von synthetischen Daten, [61].

30.09.2014												
Nr.	Ebene	SIL	Erstellungsdatum	Classification	Iterationen	Umplanungen	Fertigstellungsdatum	Verzug	State	Tage	Iteration	SW-Release
1	Modul	No	18.06.2013 10:55	Change	7	2	04.06.2014		Change Analyzed	30	2	4.020.040
2	Modul	QM	19.09.2013 15:22	Finding	3	1	03.06.2014	207	Change Implemented	309		1 4.000.020
3	Gesamt	A	25.09.2013 10:02	Finding	2	2	01.09.2014		Change New	196		2 4.010.040
4	Modul		14.10.2013 12:21	Finding	1	1	10.02.2014	0	Change New	50		1 3.030.040
5	Modul	A	25.10.2013 16:28	Change	1	1	21.11.2013		Change Implemented	326		1 3.020.060
6	Modul	No	20.11.2013 07:52	Change	6	1	03.06.2014	113	Change Implemented	112		1 4.000.020
7	Modul	A	27.11.2013 15:27	Finding	8	2	14.08.2014	210	Change New	188		2 4.010.020
8	Modul	B	02.12.2013 08:45	Finding	4	1	01.01.2014		Change Implemented	288		1 3.030.000

Abbildung 26: Beispielhafte Auswertung der Arbeitspakete

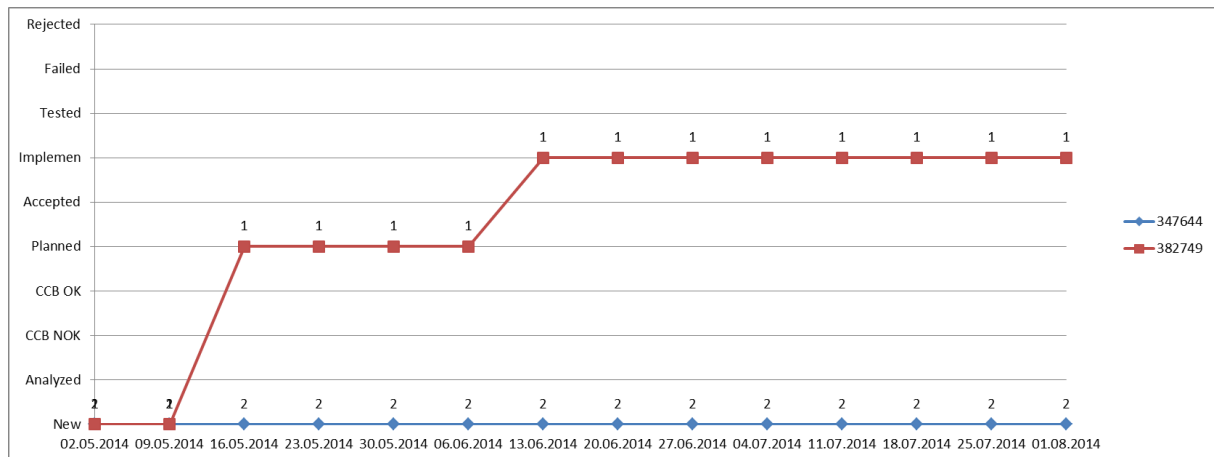


Abbildung 27: Status-Verlauf von Arbeitspaketen, vgl. [61]

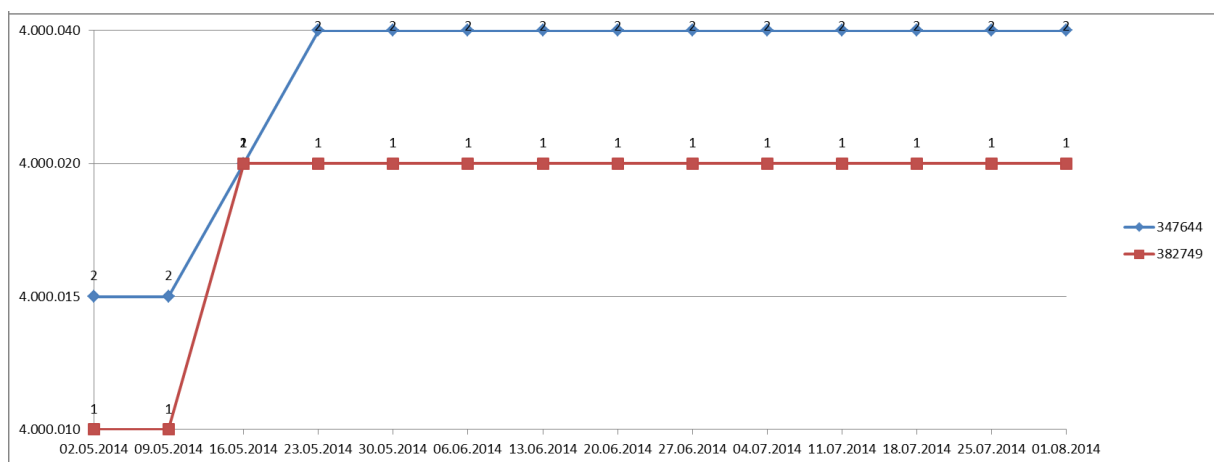


Abbildung 28: Geplante Software-Releases von Arbeitspaketen, vgl. [61]

### 4.3.2 Analysemethoden für die Qualitätssicherung

Die Aufgabe der Qualitätssicherung wird in der ISO 15504 [70] folgendermaßen beschrieben: „Der Zweck des Qualitätssicherungs-Prozesses besteht darin, durch eine unabhängige Instanz zu gewährleisten, dass die Arbeitsprodukte und Prozesse die vordefinierten Vorschriften und Pläne erfüllen.“, [70]. Aus diesem Grund hat die Qualitätssicherung ein Interesse an der Gesamtpformance des Projektes. Insbesondere in Hinsicht auf die Umsetzung von Anforderungen, dem Testen der Software und dem Beheben von Fehlern. Konkrete Fragen die für das Qualitätssicherung definiert wurden sind, [62]:

1. Wie ist der Verlauf beziehungsweise die Performance des Projektes?
2. Gibt es Planungs- und Implementierungsspitzen?
3. Was ist der Implementierungsstand (funktionale Vollständigkeit) des Projektes?
4. Wurden alle definierten Testfälle durchgeführt?
5. Wie viele Fehler konnten behoben werden?

Zur Beantwortung dieser Fragen wurden zwei eigenständige Analysemethoden entwickelt:

- Performance-Auswertung,
- Soll-Ist-Vergleich.

Mithilfe der Performance-Auswertung sollen die Fragen nach der Produktivität des Projektes und der Identifikation von Planungs- und Implementierungsspitzen beantwortet werden. Während mithilfe eines Soll-Ist-Vergleichs Aufschluss über den Stand der Implementierung, des Testprozesses und der gefundenen und behobenen Fehler gegeben werden soll. Die Performance-Auswertung bietet dabei bewusst einen abstrahierten Blick auf die Daten um einen Gesamtüberblick über das Projekt zu ermöglichen. Wohingegen beim Soll-Ist-Vergleich die Beziehung der Elemente zueinander und zu den Software-Releases im Vordergrund steht. Für beide Analysemethoden wurden die Anforderungen, Testfälle und Arbeitspakete/Fehler aufgrund des Zustandes in die Kategorien „done“ und „in work“ eingeteilt. Alle Elemente im Zustand „x Closed“ sind für das Projekt nicht mehr relevant und werden deshalb nicht betrachtet. Die Einteilung der Elemente in „done“ und „in work“ ist in Tabelle 8 ersichtlich, [61].

Tabelle 8: Einteilung der Elemente in „done“ und „in work“

Element	„done“	„in work“
Anforderungen	Requirement Implemented	Requirement New Requirement Specified
Testfälle	TC Completed	TC New TC Specified TC In Work TC Retest TC Failed TC Completed with restriction
Arbeitspakete/Fehler	Change Completed	Change New Change Accepted Change CCB OK Change CCB NOK Change Rejected Change Accepted Change Planned Change Failed Change Implemented, Change Tested

### Performance-Auswertung

Die Performance-Auswertung stellt einen zeitlichen Verlauf der Daten über die Projektlaufzeit dar. Die historischen Daten werden aus der im Kapitel 0 erwähnten Analyse-Datenbank gewonnen. Abbildung 30 zeigt einen Ausschnitt aus einer beispielhaften Performance-Auswertung, basierend auf synthetischen Daten. Im oberen Bereich wird eine Zeitleiste mit dem Datum aufgetragen. Darunter befinden sich die Anforderungen, Testfälle, Arbeitspakete und Fehler aufgeschlüsselt in „done“ und „in work“. Zur besseren Übersichtlichkeit sind die einzelnen Werte mit einem Farbschema versehen. Dabei wird für „done“ und „in work“ jeweils ein eigenes Farbschema angewendet. Die erledigten Elemente („done“) sollten im Laufe des Projektes ansteigen, deshalb bezieht sich das Farbschema immer auf den vorhergehenden Auswertungspunkt. Steigt die Anzahl der Elemente von einem Auswertungspunkt zum Nächsten an, wird das als positive Entwicklung angesehen und deshalb grün hinterlegt. Bleibt der Wert dagegen gleich, stagniert die Anzahl der erledigten Elemente und wird orange gekennzeichnet. Ein Rückgang der erledigten Elemente deutet auf ein besonderes Ereignis während der Entwicklung hin und sollte genauer untersucht werden, dieser Wert wird gelb markiert.

Bei der Anzahl der Elemente in Arbeit („in work“) ist dagegen das Erkennen von Extremwerten wichtig. Um Extremwerte auch in asymmetrischen Verteilungen richtig erkennen zu können werden Quantile verwendet. Abbildung 29 zeigt die Position von Quantilen, Median und Mittelwert in einer asymmetrischen Verteilung. Die Berechnung des unteren beziehungsweise oberen Quantils erfolgt folgendermaßen, [61], [71]:

$$\frac{|\{i \mid x_i \leq x_p\}|}{n} \geq p$$

$$\frac{|\{i \mid x_i \geq x_p\}|}{n} \geq 1 - p$$

(6)

*i ... Anzahl der beobachteten Werte*

*p ... Quantil*

*x<sub>p</sub> ... Wert des Quantils*

*x<sub>i</sub> ... Aktueller Wert der sortierten Verteilung*

*n ... Anzahl von Werten in der sortierten Verteilung*

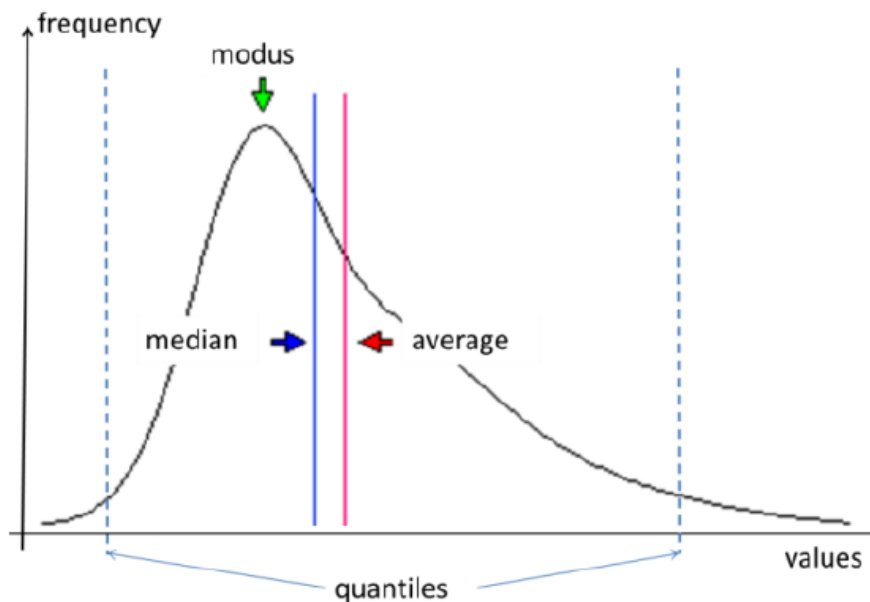


Abbildung 29: Quantile, Median und Mittelwert in einer asymmetrischen Verteilung , vgl. [61]

In einer sortierten Verteilung ist das untere p-Quantil der Wert bei dem p-Prozent der Werte der Verteilung kleiner oder gleich dem Quantil sind. Sinngemäß ist das obere p-Quantil der Wert bei dem p-Prozent der Werte in der sortierten Verteilung größer oder gleich dem Quantil sind. Die Verwendung von Quantilen ermöglicht auch in schiefen Verteilungen Extremwerte zu identifizieren. Diese Methode wird eingesetzt um die Elemente in Arbeit („in work“) mit einem Farbschema zu versehen. Liegt ein Wert über dem Quantil wird davon ausgegangen, dass in diesem Zeitraum mehr Elemente als gewöhnlich bearbeitet wurden. Bei einem Wert unter dem Quantil dagegen wurden unterdurchschnittlich wenig Elemente bearbeitet. Liegt der Wert zwischen den Quantilen wird von einer normalen Anzahl der Elemente in Bearbeitung ausgegangen. Dementsprechend werden die Werte farblich hervorgehoben. Bei den Fehlern wird ebenfalls die hinterlegte Fehlerschwere („faults severity“) ausgewertet. Dazu werden die drei möglichen Werte (schwer, mittel, leicht) prozentuell gewichtet und daraufhin ein Durchschnittswert über alle Fehler dieses Auswertungszeitpunkts

berechnet. Dieser Durchschnittswert ermöglicht eine grobe Einschätzung über die Schwere der Fehler. Als letzter Punkt in der Performance-Auswertung werden noch ausgewählte Verhältnisse („ratios“) berechnet, die interessante Werte zueinander ins Verhältnis setzen. Dadurch können die Auswertungszeitpunkte auch aufgrund dieser Verhältnisse verglichen werden. Die Performance-Auswertung bietet einen komfortablen Überblick über den Verlauf und die Performance des Gesamtprojektes. Durch das integrierte Farbschema können Spitzen und Extremwerte während der Entwicklung auf einen Blick identifiziert werden. Die kumulierte Darstellung der abgeschlossenen Elemente offenbart zudem einen schnellen Überblick über die Geschwindigkeit mit dem das Projekt voranschreitet. Zusätzlich können die Werte in Form eines Diagramms graphisch dargestellt werden. In Abbildung 31 wird das Farbschema der Performance-Auswertung anhand von synthetischen Daten veranschaulicht, [61].

date	01.01.2014	02.01.2014	03.01.2014	04.01.2014
<b>requirements</b>				
requirements done	703	713	714	714
requirements in work	131	128	128	129
<b>tests</b>				
tests done	354	349	351	353
tests in work	69	75	79	79
<b>planning elements</b>				
planning elements done	295	296	296	297
planning elements in work	71	71	73	76
<b>faults</b>				
faults done	220	220	220	221
faults in work	44	44	46	47
faults severity	0,31	0,31	0,31	0,31
<b>ratios</b>				
faults/planning elements	0,15	0,15	0,16	0,16
faults/tests	0,64	0,59	0,58	0,59
planning elements/ tests	0,7	0,7	0,69	0,69
requirements/tests	1,97	1,98	1,96	1,95

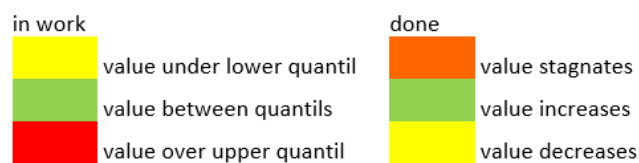


Abbildung 30: Beispielhafte Performance-Auswertung, [61]



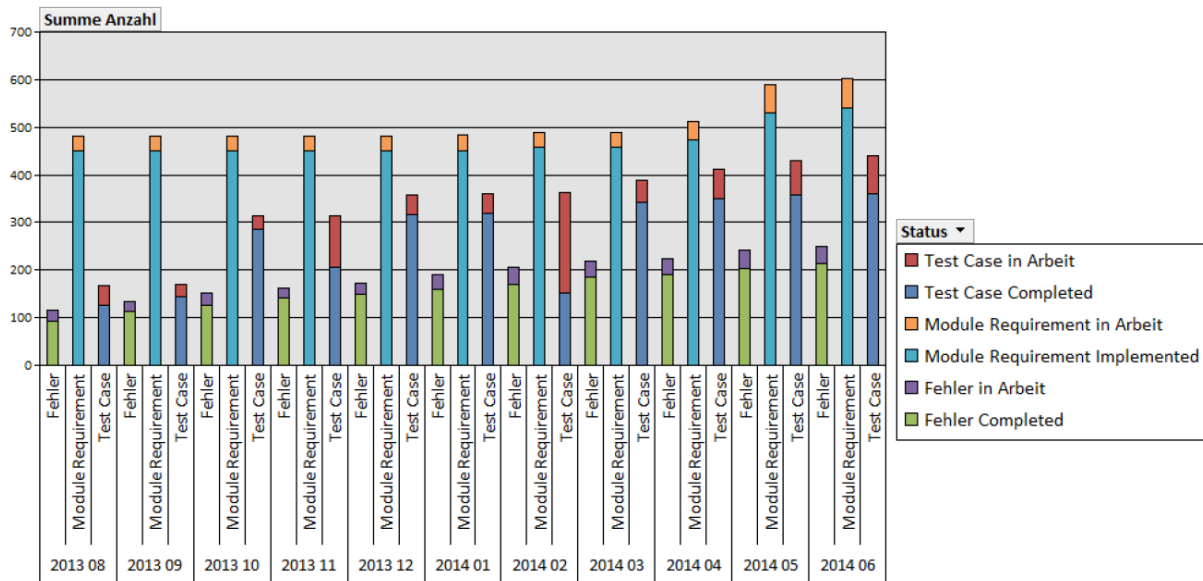
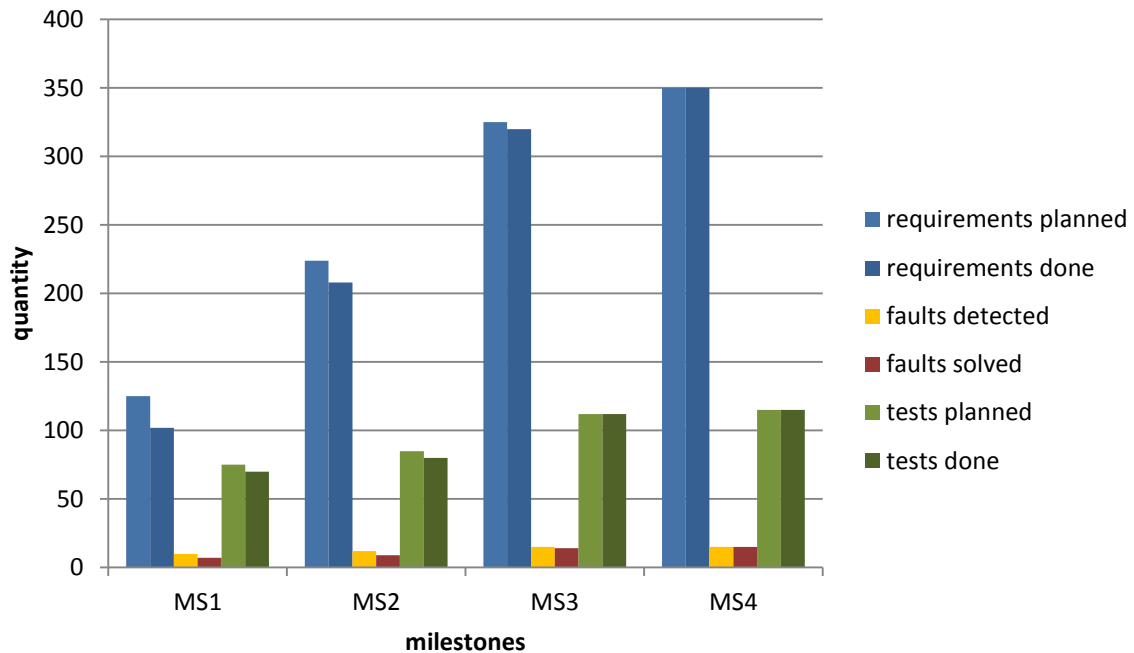


Abbildung 31: Diagramm der Performance-Auswertung

### Soll-Ist-Vergleich

Neben dem groben Überblick, den die Performance-Auswertung bietet, hat die Qualitätssicherung auch Interesse an detaillierteren Informationen bezüglich des Projektfortschritts. Zu diesem Zweck wurde ein Soll-Ist-Vergleich entwickelt um die Kontrolle des Projekts zu erleichtern. Voraussetzung für diesen Soll-Ist-Vergleich ist, dass das geplante und das tatsächliche Fertigstellungsdatum der Elemente (Anforderungen, Testfälle, Arbeitspakete) bekannt sind. In der Entwicklungs- und Testdatenanalyse (Kapitel 4.2.1) wurde bereits darauf hingewiesen, dass zur Planung Arbeitspakete verwendet werden. Dementsprechend enthalten diese Informationen über den geplanten und tatsächlichen Software-Release der Fertigstellung. Die dazugehörigen Anforderungen und Testfälle sind mit den Arbeitspaketen verbunden, wodurch eine indirekte Zuordnung zu dem Software-Release über die Arbeitspakete möglich ist. Voraussetzung ist dabei, dass die Anforderungen und Testfälle vollständig mit den Arbeitspaketen verknüpft sind. Aus diesem Grund ermöglicht der Soll-Ist-Vergleich die Ermittlung der Verknüpfungsgrade der Elemente untereinander. Auf diese Weise erhält die Qualitätssicherung auch einen Einblick in die Qualität der Verknüpfungen innerhalb des Projekts. Zur Erstellung der Auswertung werden, wie schon in den vorherigen Analysemethoden, historische Daten aus der Analyse-Datenbank verwendet (siehe Kapitel 0). Die Zuordnung der Meilensteine, Software-Releases und System-Freigaben zu konkreten Kalenderdaten steht über eine eigene Liste in Tabellenform zur Verfügung. Durch diese Zuteilung ist eine chronologische Anordnung anhand dieser Daten möglich. Die Elemente werden, wie schon bei der Performance-Auswertung, dem Status entsprechend in „done“ und „in work“ eingeteilt (siehe Tabelle 8). Ausgangspunkt für den Soll-Ist-Vergleich sind die Arbeitspakete, die zu dem betrachteten Software-Release bereits abgeschlossen sein sollten. Grundlage für diese Entscheidung ist der geplante Software-Release für Elemente „in work“ und der abgeschlossene Software-Release für Elemente „done“. Liegt der hinterlegte Software-Release vor dem betrachteten Software-Release ist das Arbeitspaket in Verzug und muss genauer inspiziert werden. Hierzu kann die Auswertung der Arbeitspakete des Projektmanagements als Hilfe dienen (siehe Kapitel 4.3.1). Die Anforderungen und Testfälle werden über die Verknüpfung zu den Arbeitspaketen ermittelt. Abbildung 32 zeigt eine beispielhafte Darstellung des Soll-Ist-Vergleichs anhand von synthetischen Daten. Der Anteil der bereits abgeschlossenen Elemente zu allen vorhandenen Elementen ergibt den prozentuellen

Projektfortschritt zu dem betrachteten Software-Release. Diese kumulierte Darstellung des Projektfortschritts kann auch genauer pro Software-Release aufgelöst werden, indem nur die Arbeitspakete betrachtet werden die dem ausgewählten Software-Release zugeordnet sind. Dadurch kann ein Blick auf die Performance der einzelnen Software-Releases geworfen werden. Durch diesen Soll-Ist-Vergleich können die Fragen der Qualitätssicherung in Bezug auf Implementierungs- und Teststatus sowie zu der Anzahl der behobenen Fehler beantwortet werden, [61].



milestone	MS1	MS2	MS3	MS4
system release	SR1	SR2	SR3	SR4
software release	SW1	SW2	SW3	SW4

requirements don	82%	93%	98%	100%
faults solved	70%	75%	93%	100%
tests done	93%	94%	100%	100%

Abbildung 32: Beispielhafter Soll-Ist-Vergleich, [61]

### 4.3.3 Analysemethoden für das Management

Im Gegensatz zum Projektmanagement und der Qualitätssicherung benötigt das Management primär keine detaillierten Informationen über einzelne Arbeitspakete oder Software-Releases. Eine der wichtigsten Aufgaben des Managements ist es zu entscheiden ob und wann ein System für die Auslieferung an den Kunden freigegeben werden kann. Zu diesem Zweck wird eine Analysemethode benötigt, die eine schnelle Einschätzung über den Zustand des Systems bietet und somit die Entscheidung über die System-Freigabe unterstützt. Das Management muss sich vor jeder System-Freigabe folgende Fragen stellen, die in weiterer Folge mithilfe von Analyse-Methoden beantwortet werden sollen, [62]:

1. Wie viele Arbeitspakete wurden umgesetzt?
2. Wie viele Fehler wurden behoben?
3. Wie viele Testfälle wurden ausgeführt?
4. In welchem Zustand befindet sich das System relativ zu vorherigen Freigaben?

Die Informationen über die Arbeitspakete, Fehler und Testfälle werden wie bei den anderen Analysemethoden wiederum aus der Analyse-Datenbank gewonnen (siehe Kapitel 0). Um dem Management einen schnellen Überblick über die verschiedenen System-Freigaben zu ermöglichen wurde ein Management-Cockpit als Analysemethode entwickelt. Ein beispielhaftes Cockpit ist anhand von synthetischen Daten in Abbildung 33 veranschaulicht. Das Herzstück dieses Cockpits bildet die Auswertung der Entwicklungs- und Testdaten aus dem MKS. Der prozentuelle Fertigstellungsgrad wird hierbei bezogen auf die System-Freigabe (relativ) und über den gesamten Entwicklungsverlauf (kumuliert) berechnet. Auf diese Weise sind die Performance der System-Freigabe und der Stand des Gesamtprojektes schnell ersichtlich. Die Daten-Auswertung aus dem MKS gliedert sich wiederum in die drei Kategorien „Arbeitspakete umgesetzt“, „Fehler behoben“ und „Testfälle ausgeführt“. Zur besseren Übersicht und schnelleren Interpretation der Werte wurde ein Bewertungsschema entwickelt, aufgrund dessen die Werte eingefärbt werden. Den Abschluss bildet die Bewertung der System-Freigabe, aufgrund von Metriken anhand der drei Software-Dimensionen: Quantität, Komplexität und Qualität.

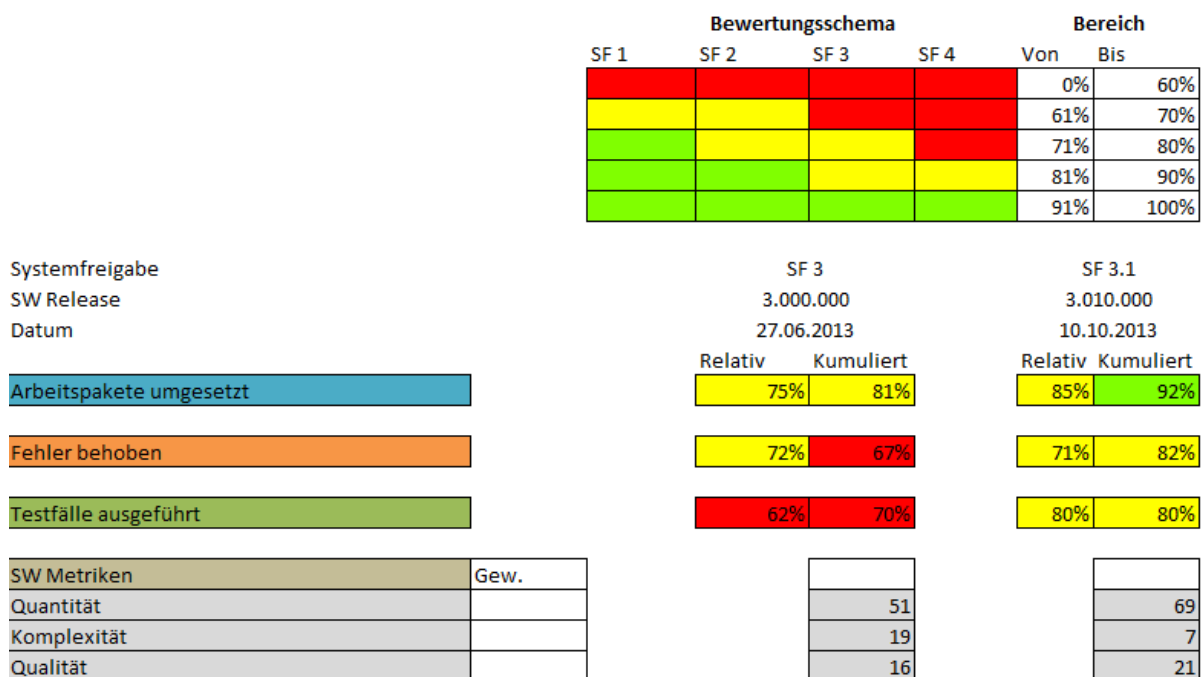


Abbildung 33: Beispielhaftes Management-Cockpit, vgl. [62]

Bei der Auswertung der Entwicklungs- und Testdaten werden die einzelnen System-Freigaben in chronologischer Reihenfolge aufgelistet. Der Prozentsatz der Arbeitspakete gibt an wie viele der geplanten Anforderungen und Änderungen in der System-Freigabe, beziehungsweise insgesamt, umgesetzt wurden. Ein geringer Wert deutet auf Probleme bei der Implementierung von Anforderungen oder der Realisierung von Änderungen hin. Durch die Kategorie „Fehler behoben“ ist ersichtlich wie viele der bekannten Fehler behoben wurden. Der relative Wert bezieht sich dabei auf Fehler deren Behebung in der System-Freigabe geplant war und die kumulierte Darstellung auf alle

bekanntem Fehler im Projekt. Der relative Wert ist somit ein Indikator für die Effizienz der Fehlerbehebung in der betrachteten System-Freigabe und der kumulierte Wert gibt Aufschluss über das Verhältnis aller behobenen zu allen bekannten Fehlern im Projekt. Der Prozentsatz der Testfälle steht für alle erfolgreich ausgeführten Testfälle zu allen vorhandenen Testfällen. Bei dem relativen Wert werden nur jene Testfälle berücksichtigt die einem Arbeitspaket für diese System-Freigabe zugeordnet sind. Beim kumulierten Wert werden alle Testfälle der vergangenen System-Freigaben mit berücksichtigt. Um eine schnelle und einfache Beurteilung der einzelnen System-Freigaben zu unterstützen, wurde ein Bewertungsschema für die System-Freigaben entwickelt (siehe Abbildung 34). Das Bewertungsschema beinhaltet drei verschiedene Farbkodierungen:

- rot = keine Freigabe möglich,
- gelb = Freigabe unter Umständen möglich (muss begründet werden),
- grün = Freigabe möglich.

Die Anforderungen an die System-Freigabe steigen aufgrund der Freigabe-Stufe immer weiter an. Während bei einer System-Freigabe auf Stufe 1 Werte ab 70% als in Ordnung angesehen werden, sind auf Stufe 4 Fertigstellungsgrade ab 91% erforderlich. Dieses Bewertungsschema ist frei konfigurierbar und kann somit an die Bedürfnisse eines jeden Projektes angepasst werden. In Abbildung 33 wurde beispielsweise das Bewertungsschema aus Abbildung 34 angewendet. Da sich die zwei ausgewerteten System-Freigaben auf Stufe 3 befinden sind Werte ab 91% in Ordnung, Werte zwischen 71%-90% unter Umständen vertretbar und Werte unter 70% nicht in Ordnung. Mithilfe dieses Bewertungsschemas kann ein schneller Überblick über die System-Freigaben gegeben werden.

Bewertungsschema				Bereich	
SF 1	SF 2	SF 3	SF 4	Von	Bis
				0%	60%
				61%	70%
				71%	80%
				81%	90%
				91%	100%

Abbildung 34: Bewertungsschema für Systemfreigaben, vgl. [62]

Das letzte Instrument zur Bewertung der System-Freigaben bezieht Software-Metriken aus dem Entwicklungsprozess mit ein. Dabei werden sowohl Code-Metriken als auch Prozess-Metriken herangezogen. Diese Metriken werden kombiniert und zu den drei Dimensionen der Software: Quantität, Komplexität und Qualität aggregiert. Da manche Metriken einen größeren Einfluss auf die Aggregation haben als andere werden die einzelnen Metriken gewichtet. Abbildung 35 zeigt die Zusammensetzung der einzelnen Dimensionen für die einzelnen Metriken. Die Quantität der Software setzt sich aus den Code-Metriken Kilo Lines of Code (KLOC) und Anzahl der C-Funktionen sowie den Prozess-Metriken Anzahl der umgesetzten Arbeitspaketen und Anzahl der ausgeführten Testfälle zusammen. Die Code-Metriken beziehen sich auf die physische Größe des Quellcodes, wobei die Anzahl der Codezeilen in Tausend (KLOC) den größten Aussagewert besitzt. Dementsprechend wird KLOC mit 20% und die Anzahl der C-Funktionen mit 10% gewichtet. Die Prozess-Metriken messen die Quantität der Software aus Sicht der Funktionalität. Je mehr Arbeitspakete und Testfälle umgesetzt wurden, desto mehr vorhandene Funktionalität kann in der Software angenommen werden. Die Komplexität der Software wird nur aufgrund von Code-Metriken

ermittelt. Hierzu wird die zyklomatische Komplexität als Wert für die Ablauflogik, die Anzahl der Funktionsparameter als Indikator für die Interkonnektivität und die Anzahl der Aufrufebenen als Grad der Verschachtelung herangezogen. Als wichtigste Metrik wurde die zyklomatische Komplexität mit 50% gewichtet. Die anderen 50% werden auf die zwei verbleibenden Metriken zu je 25% aufgeteilt. Die Qualität wird größtenteils aus Prozess-Metriken berechnet. Der Grund wieso nicht mehr Code-Metriken berücksichtigt wurden ist darauf zurückzuführen, dass diese nicht im MKS abgelegt werden, welches als Basis zur Erstellung der Analysemethoden dient. Deshalb wurden vermehrt Prozess-Metriken eingesetzt. Zur Berechnung der Qualität wurden Kennzahlen der Testüberdeckung, der Kommentardichte, der funktionalen Vollständigkeit und der Effizienz der Fehlerbehebung herangezogen. Die Testüberdeckung berechnet sich aus dem Verhältnis der Anzahl der Arbeitspakete, die zumindest einen Testfall zugeordnet haben, zu allen Arbeitspaketen der System-Freigabe. Die Kommentardichte wird für jede System-Freigabe ermittelt und kann deshalb miteinbezogen werden. Die umgesetzten Arbeitspakete sind ein Maß für die funktionale Vollständigkeit der System-Freigabe. Es werden dabei die umgesetzten Arbeitspakete zu allen Arbeitspaketen der System-Freigabe ins Verhältnis gesetzt. Zu guter Letzt wird die Effizienz der Fehlerbehebung miteinbezogen. Diese setzt sich aus der Anzahl der behobenen Fehler zu der Anzahl aller bekannten Fehler der System-Freigabe zusammen.

SW Metriken	Gew.
Quantität	
KLOC	20%
# C - Funktionen	10%
# Arbeitspakete umgesetzt	35%
# TC ausgeführt	35%
Komplexität	
Ø Zyklomatische Komplexität	50%
Ø Funktionsparameter	25%
Ø Aufrufebenen	25%
Qualität	
% Testüberdeckung (Arbeitspakete)	30%
% Kommentardichte	5%
% Arbeitspakete umgesetzt	30%
% Fehler behoben	35%

Abbildung 35: Auswahl und Gewichtung der Metriken

Die aggregierten Metriken der einzelnen System-Freigaben werden in einem Netzdiagramm dargestellt. Dadurch ist ein rascher Vergleich der einzelnen System-Freigaben aufgrund der Eigenschaften Quantität, Komplexität und Qualität möglich. Mithilfe des Netzdiagramms können Trends frühzeitig erkannt und geeignete Gegenmaßnahmen eingeleitet werden. So kann ein starker Anstieg der Komplexität oder ein Sinken der Qualität schnell erkannt werden (vgl. Abbildung 24).

## 5 Realisierung und Implementierung des Prototypen

Die im vorherigen Kapitel spezifizierten Analysemethoden wurden in Form eines Prototyps implementiert. Dieser dient als Ansatz wie eine zukünftige Umsetzung der Analysemethoden stattfinden könnte. Aus diesem Grund wurde die Implementierung auf Grundlage eines Beispielprojektes entwickelt und getestet. Grundlage für die Analysemethoden sind die Daten aus dem Informationssystem MKS. Um den Verlauf des Entwicklungsprozesses analysieren zu können sind die Daten zu mehreren Zeitpunkten im Projekt notwendig. Diese zeitpunktbezogenen Daten stehen zwar im MKS-System zur Verfügung, jedoch ist die Generierung dieser historischen Daten mit einem hohen Zeitaufwand verbunden. Um diesen Aufwand nicht bei jeder Verwendung der Analysemethoden erneut betreiben zu müssen, werden die historischen Daten automatisiert exportiert und in einer eigenen Analyse-Datenbank gespeichert. Die unterschiedlichen Analysemethoden können dann auf die historischen Daten direkt über die Analyse-Datenbank zugreifen ohne, dass diese erneut vom MKS-System generiert werden müssen. Der automatisierte Export sowie die weitere Verarbeitung der Daten erfolgt durch eine eigens dafür erstellte Applikation. Die Visualisierung und Ausgabe der Daten wird mithilfe von Microsoft Excel erstellt.

### 5.1 Grundlegender Aufbau des Prototypen

Abbildung 37 zeigt den grundlegenden Funktionsaufbau des Prototyps. Die Daten aus dem MKS-System werden über eine vb.net Anwendung exportiert und in eine Analyse-Datenbank aufgebaut. Als Analyse-Datenbank wurde Microsoft Access [72] verwendet. Der Hauptgrund für die Wahl von MS Access liegt in der einfachen Übertragbarkeit und Wartung der Datenbankdatei, die während der Entwicklung des Prototyps notwendig war. Durch die Limitierung von MS Access mit einer Datenbankdateigröße von höchstens 2 Gigabyte musste die Datenbank in eine Front-End und mehrere Back-End Datenbanken aufgeteilt werden [73] (siehe Abbildung 36). Der Zugriff auf die Daten kann dabei einheitlich über die Front-End-Datenbankdatei erfolgen, die mit allen Back-End-Datenbankdateien verbunden ist. Die Front-End-Datenbankdatei dient somit als Schnittstelle zu den Daten die auf mehrere Back-End-Datenbankdateien aufgeteilt sind. Durch das Aufteilen der Datenbank kann die Limitierung der Dateigröße umgangen werden und dennoch ein einheitlicher Datenbankzugriff gewährleistet werden, [74].

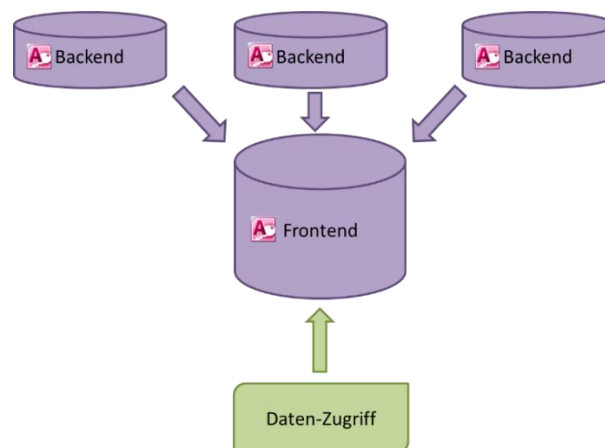
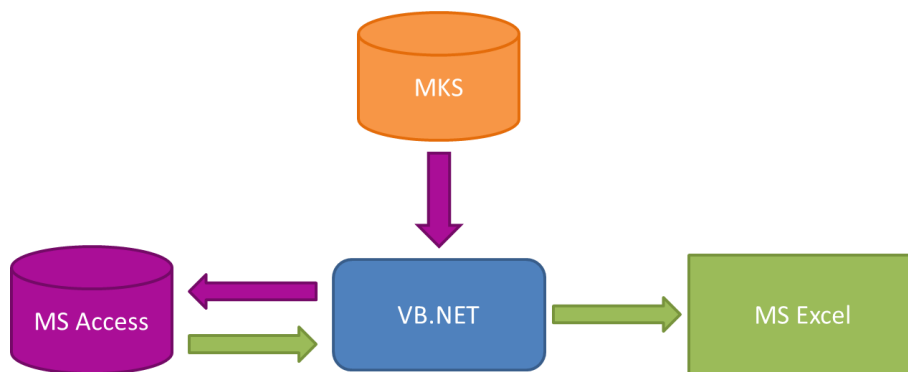


Abbildung 36: Aufteilung der Access-Datenbank

Auf Basis der Daten aus der Analyse-Datenbank kann der Benutzer danach, mithilfe des entwickelten Analyse-Tools, verschiedene Auswertungen erstellen. Hierzu stehen verschiedene Einstellungs- und Auswahlmöglichkeiten im Analyse-Tool zur Verfügung. Die Auswertungen selbst werden in Form einer MS Excel Arbeitsmappe erstellt. Die Kommunikation zwischen vb.net und der Arbeitsmappe findet dabei über die Office Interop Schnittstelle statt. Mithilfe dieser Schnittstelle kann die Anwendung direkt auf Office-Objekte zugreifen. So ist es möglich die Arbeitsmappe direkt über das Analyse-Tool zu erstellen, [75]. Manche Funktionalitäten, wie das Erstellen der Charts oder das Ausdrucken der Auswertungen, wurden direkt in MS Excel implementiert. Hierzu wurde die integrierte VBA-Entwicklungsumgebung verwendet, [76]. Als Grundlage für die Auswertungen dienen MS Excel-Vorlagen, die bereits alle benötigten VBA-Methoden enthalten. Dies hat den Vorteil, dass Steuerelemente innerhalb der Arbeitsmappe und die VBA-Methoden nicht jedes Mal beim Erstellen der Auswertung neu hinzugefügt werden müssen.



**Abbildung 37: Grundlegendes Konzept des Prototyps**

Der hier vorgestellte Prototyp besteht aus mehreren Dateien und Anwendungen, die in Abbildung 38 in Form einer empfohlenen Ordnerstruktur dargestellt werden. Die Pfade und Ordner auf die das Analyse-Tool zugreift sind jedoch nicht fest vorgegeben und können in den Einstellungen frei definiert werden. Das Analyse-Tool stellt das Herzstück des Prototyps dar, in dem alle Einstellungen vorgenommen und Auswertungen erstellt werden können. Auch der MKS-Export wird über das Analyse-Tool gestartet. Die Software-Releases und System-Freigaben werden mithilfe einer Liste auf die konkreten Kalenderdaten zugeordnet. Diese Liste muss in einer Excel-Datei mit vorgegebenen Spaltennamen vorhanden sein. Die Back-End-Dateien der Analyse-Datenbank müssen sich in einem Ordner „Backend“ innerhalb des Front-End-Datenbankordners befinden. Der Speicherort der Front-End-Datenbank kann wiederum frei festgelegt werden. Der Vorlagen-Ordner enthält die Vorlagen, die als Grundlage für die Erstellung der verschiedenen Auswertungen dienen. In diesen Vorlagen ist das Layout der späteren Auswertung bereits vorab grob festgelegt. Des Weiteren beinhalten sie auch bereits die Buttons und VBA-Makros zur Steuerung der Arbeitsmappe. Zuletzt ist noch ein Ordner notwendig in dem die Software-Metriken aus Polyspace abgelegt werden. Der Ablageort ist frei wählbar, jedoch muss die Datei nach einem vorgegebenen Schema benannt werden damit eine Zuordnung der XML-Datei zu bestimmten System-Freigaben möglich ist.

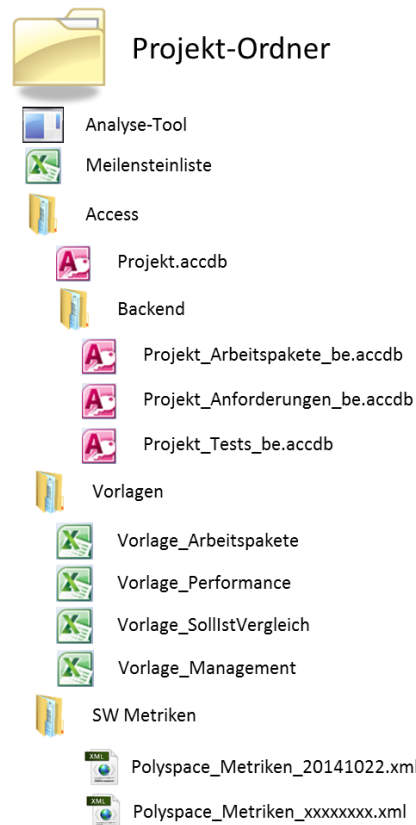


Abbildung 38: Empfohlene Ordnerstruktur

## 5.2 Analyse-Tool

In diesem Kapitel wird ein genauer Blick auf das Analyse-Tool geworfen, das alle notwendigen Einstellungen beinhaltet und die Erstellung von Auswertungen ermöglicht. Die Benutzeroberfläche besteht aus fünf Tabs und einem Menü-Button für die Einstellungen. Die ersten vier Tabs repräsentieren jeweils eine Auswertungsmethode und der fünfte Tab dient zur Steuerung des MKS-Exports. Abbildung 39 zeigt das Einstellungsfenster, welches über den Menü-Button „Einstellungen“ erreicht werden kann. Der obere Bereich der Einstellungen dient zur Spezifikation der Speicherorte der Front-End Datenbankdatei, der Meilensteinliste und dem Ordner für die Software-Metriken. Das Feld „RScript.exe“ ist Teil eines statistischen Projektes und hat für diese Arbeit keine Bedeutung. Die zweite Kategorie verweist auf die einzelnen Speicherorte der Excel-Vorlagen, die für die Erstellung der Auswertungen notwendig sind. Es ist zu beachten, dass für jede Auswertung die richtige Vorlage ausgewählt sein muss, weil ansonsten die korrekte Erstellung der Auswertungen nicht möglich ist. Die letzten vier Pfade dienen zur Auswahl des Ordners in dem die erstellten Auswertungen gespeichert werden. Dabei kann für jeden Auswertungstyp ein eigener Ausgabeordner spezifiziert werden um die erstellten Auswertungen besser zu organisieren. Die letzten drei Einstellungen beziehen sich auf den MKS-Export. Der MKS Projektname muss den Namen des betrachteten Projektes im MKS Informationssystem enthalten. Der Projektname wird benötigt um nur Elemente zu exportieren die dem Projekt zugeordnet sind. Das Feld „Variant“ dient in dem Beispielprojekt dazu Arbeitspakete aus dem Modulbaukasten in das Projekt miteinzubeziehen. Dabei wird in den Arbeitspaketen des Modulbaukastens der Name des Projektes in das Datenbankfeld „Variant“ eingetragen. Das Datum des Projektstarts ist sogleich das Datum von dem aus historische Daten für dieses Projekt exportiert werden. Die Einstellungen werden in einer eigenen Einstellungsdatei im XML-Format abgespeichert und bei jedem Programmstart eingelesen.



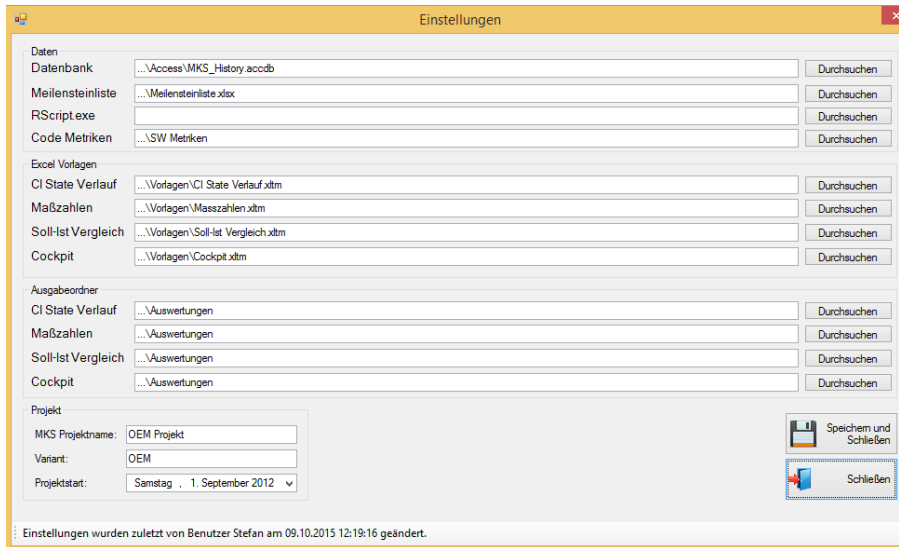


Abbildung 39: Einstellungen

Der erste Tab enthält die Einstellungsmöglichkeiten für die Auswertung für das Projektmanagement über die Verläufe der Arbeitspakete (siehe Abbildung 40). Die einzige Einstellungsmöglichkeit, welche die Auswertung beeinflusst, ist der Zeitraum für den die Auswertung erstellt werden soll. Wird zum Beispiel der 01. September 2014 und 30. September 2014 als Zeitraum ausgewählt, werden alle offenen Arbeitspakete zum 30. September aufgelistet und deren Verlauf rückwirkend bis zum 01. September dargestellt. Als Orientierungshilfe zur Auswahl des Zeitraums befindet sich im unteren Bereich eine Zeitlinie, die wahlweise alle Software-Releases oder System-Freigaben in dem ausgewählten Bereich anzeigt. So ist direkt im Analyse-Tool die Auswahl eines Zeitraums anhand interessanter Software-Meilensteine möglich. Es handelt sich bei der Zeitlinie um eine eindimensionale Darstellung, die unterschiedliche Höhe der Software-Meilensteine dient lediglich zur besseren Lesbarkeit bei vielen Datenpunkten. Der Button neben der Eingabe des Zeitraums startet die Auswertung und speichert die fertige Excel-Arbeitsmappe in den Ausgabeordner. Dieser Ausgabeordner kann mithilfe eines Buttons direkt im Explorer geöffnet werden. Die fertige Auswertung ist beispielhaft in Abbildung 26 dargestellt.

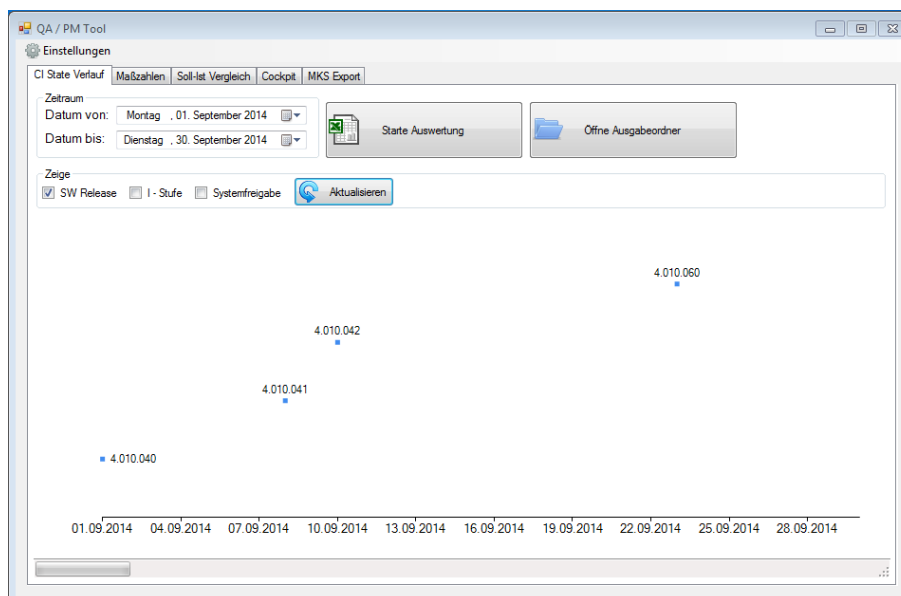


Abbildung 40: Verlauf der Arbeitspakete

Im zweiten Tab finden sich die Einstellungen für die Performance-Auswertung, die vor allem für die Qualitätssicherung von Interesse ist (siehe Abbildung 41). Wie bereits bei den Verläufen der Arbeitspakete, ist auch hier der Zeitraum für die Auswertung zu wählen. Zur besseren Orientierung wird wiederum eine Zeitleiste mit den Software-Meilensteinen in dem ausgewählten Zeitraum angezeigt. Neben der Auswahl des Zeitraums kann das Intervall zwischen den Auswertungspunkten bestimmt werden. Es steht eine wöchentliche und eine monatliche Auswertung zur Verfügung. Die Auswertung startet beim späteren Datum des Zeitraums und stellt im angegebenen Intervall Auswertungspunkte dar, bis das frühere Datum des Zeitraums erreicht ist. Die nächste Einstellungsmöglichkeit bezieht sich auf die unteren und oberen Schranken, die für die farbliche Markierung von Extremwerten verwendet werden. Für die untere Schranke steht ein 5%, 10% oder 20% Quantil zur Auswahl und für die obere Schranke kann zwischen einem 95%, 90% oder 80% Quantil gewählt werden. Im letzten Punkt kann die Fehlergewichtung für die einzelnen Fehlerschweren Low, Medium und High gewählt werden. Diese werden zur Berechnung der Gesamtfehlerschwere der Auswertungspunkte herangezogen. Ein Beispiel der Performance-Auswertung wird in Abbildung 30 dargestellt.

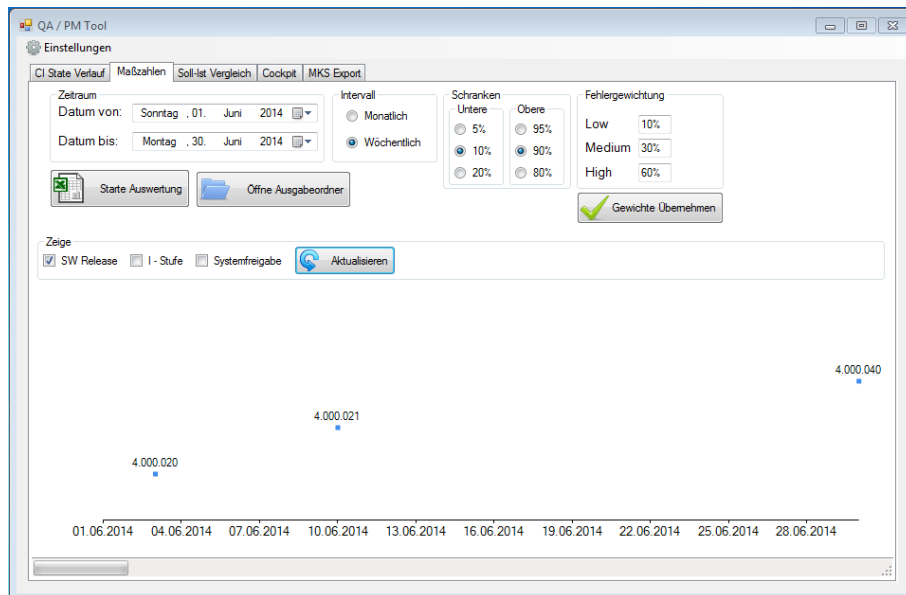


Abbildung 41: Performance-Auswertung

Im Tab des Soll-Ist Vergleichs erfolgt die Auswahl der auszuwertenden Daten über Software-Releases (siehe Abbildung 42). Hierzu wird ein hierarchischer Baum aus der Meilensteinliste generiert mit dem die verschiedenen Software-Releases für die Auswertung ausgewählt werden können. Die Werte in der Auswertung werden jeweils relativ zu der Software-Version und kumuliert auf die nächsthöhere Versions-Stufe angezeigt. Die Software-Version „3.030.060“ kumuliert zum Beispiel ausgehend von der Unterversion „3.030“. Ebenso kumulieren bei der Auswahl der Unterversion „3.030“ die Werte auf die Hauptversion 3. Die Hauptversionen selbst kumulieren immer ausgehend von der ersten verfügbaren Software-Version des gesamten Projektes. Die beispielhafte Auswertung mehrerer Software-Releases ist in Abbildung 32 ersichtlich.

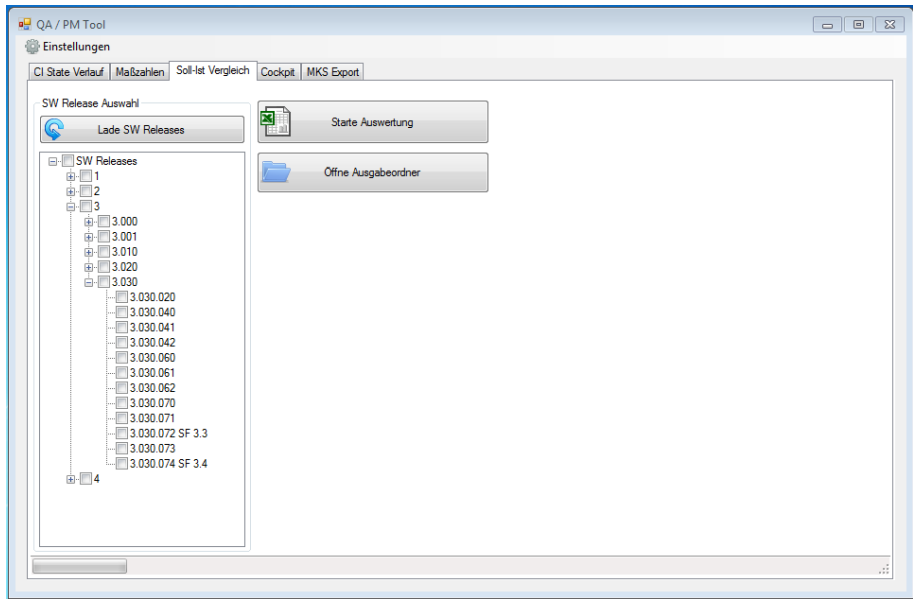


Abbildung 42: Soll-Ist-Vergleich

Der Tab für das Management-Cockpit enthält keine Auswahl eines Zeitraums oder von Software-Releases, weil die Auswertung automatisch alle verfügbaren System-Freigaben mit einbezieht (siehe Abbildung 43). Dafür ist es möglich das Bewertungsschema für die Auswertung frei zu definieren. Für jede System-Freigabe von 1 bis 4 kann eine eigene Farbe definiert werden, die für einen spezifizierten Bereich angewendet werden soll. Beispielsweise ist in Abbildung 43 für den Bereich von 0% - 60% in allen System-Freigabe-Stufen Rot als Hintergrundfarbe definiert. Die Einteilung des Bewertungsschemas kann durch das Hinzufügen von neuen Zeilen weiter verfeinert werden. Das hinterlegte Bewertungsschema wird in die Excel-Arbeitsmappe auch als Legende eingefügt. Das Eingabefeld „Restfehler Schranke“ bezieht sich auf ein weiteres Projekt das sich mit statistischen Methoden zur Bestimmung der Restfehlerwahrscheinlichkeit beschäftigt und ist für diese Arbeit nicht relevant. Ein beispielhaftes Management-Cockpit ist in Abbildung 33 abgebildet.

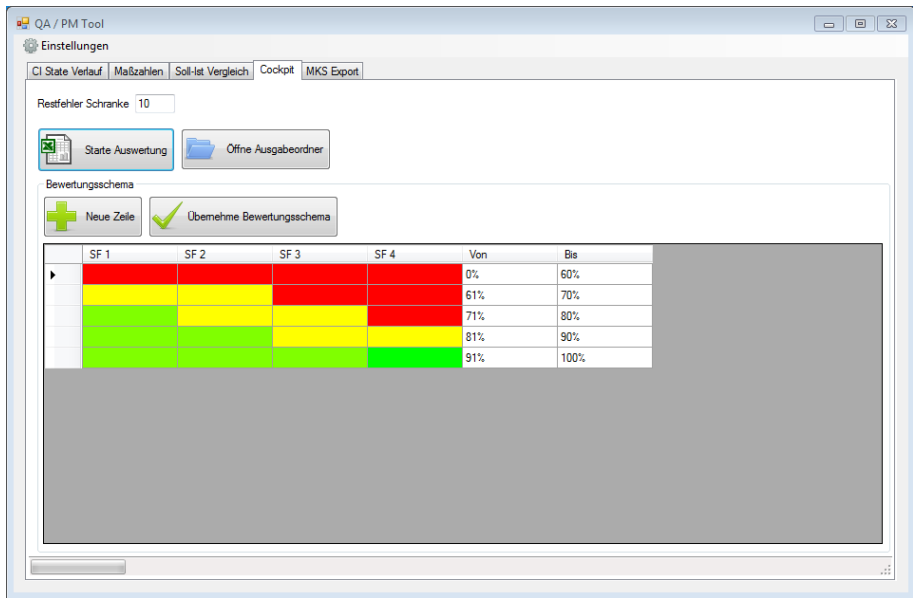


Abbildung 43: Management-Cockpit

Der letzte Tab enthält die Einstellungen für den MKS-Export (siehe Abbildung 44). Im oberen Bereich werden aktuelle Informationen zur aktuell ausgewählten Analyse-Datenbank angezeigt. Ersichtlich ist dabei der Pfad der Front-End-Datei der Datenbank, das Projekt auf das sich die Datenbank bezieht und der aktuelle Zeitraum der historischen Daten die bereits in der Datenbank vorhanden sind. Darunter befinden sich zwei Eingabefelder für die MKS-Anmeldeinformationen. Für den Zugriff auf das MKS-Informationssystem ist ein gültiger MKS-Benutzer samt Passwort notwendig um auf die Daten für den Export zugreifen zu können. Die Tabelle im unteren Bereich definiert die Export-Einstellungen für die einzelnen MKS-Elemente. Für jedes Element (Anforderungen, Testfälle, Arbeitspakete), das exportiert werden soll, müssen Informationen betreffend MKS und die Analyse-Datenbank angegeben werden. Die Bedeutung der einzelnen Spalten wird nachfolgend genauer erklärt:

- MKS Type: Die Bezeichnung des Elements im MKS Informationssystem.
- MKS Closed Item State: Jedes Element in MKS besitzt einen Zustand, der für nicht mehr relevante Elemente reserviert ist. Diese Elemente haben keinen Einfluss auf die Auswertungen und müssen nicht exportiert werden. Weil die Bezeichnung des Zustandes von Element zu Element unterschiedlich ist, muss dieser in den Einstellungen deklariert werden.
- MKS Report: Der im MKS-System gespeicherte Report, der zur Ermittlung der historischen Daten verwendet werden soll. Der Report bestimmt das XML-Schema der Export-Datei und kann deshalb von Element zu Element abweichen.
- Export Product Platform: Spezifiziert ob Elemente des Modulbaukastens mit exportiert werden.
- Access Table: Der Name der Tabelle in der Analyse-Datenbank wo die Elemente eingefügt werden.
- Relationship Field: Die MKS-Bezeichnung des Datenbankfeldes das die Verknüpfungen zu anderen Elementen enthält. Ist kein Feld angegeben werden keine Verknüpfungsinformationen exportiert.
- Access Relationship Table: Der Name der Tabelle in der Analyse-Datenbank in, die die Verknüpfungsinformationen gespeichert werden.

Nach der Eingabe des Benutzernamen und des Passworts kann mit dem Button „Starte MKS Export“ der Export gestartet werden. Der Export endet wenn das aktuelle Datum erreicht wurde oder vom Benutzer abgebrochen wird. Beim Abbrechen wird vom Programm immer der aktuell laufende Tag des Exports fertiggestellt bevor der Export beendet wird. Auf diese Weise sollen unvollständige Export-Tage in der Analyse-Datenbank vermieden werden. Auf den genauen Ablauf des Exports wird im nächsten Kapitel genauer eingegangen.

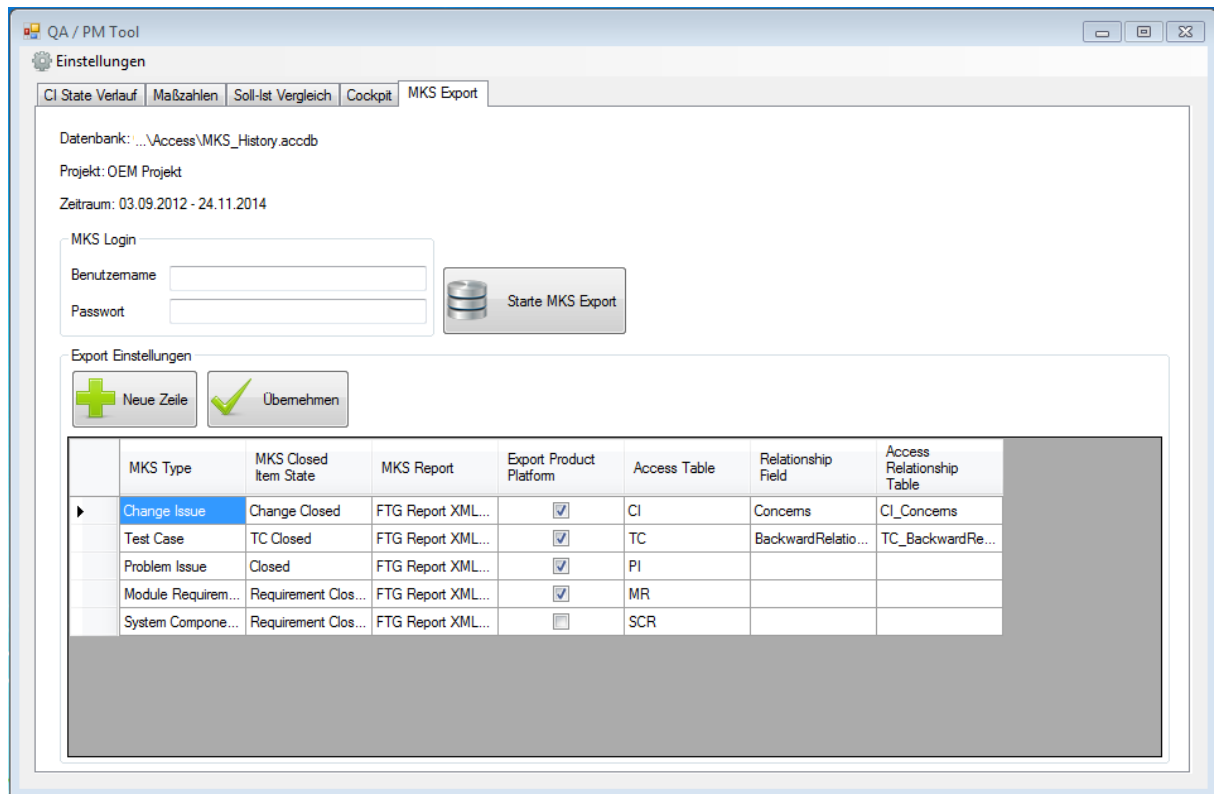


Abbildung 44: MKS-Export

### 5.3 Datenexport aus dem MKS in die Analysedatenbank

Der Export der Daten aus dem Informationssystem MKS und der Import in die Analyse-Datenbank erfolgt automatisch. Die Funktionsweise wird in Abbildung 47 schematisch dargestellt. Das Export-Programm wurde im Laufe der Entwicklung des Prototyps in das Analyse-Programm integriert. Somit ist nur mehr eine einzige Anwendung für den Export und die darauffolgenden Auswertungen notwendig. Das Export-Programm greift über die Kommandozeilen-Schnittstelle (Command-Line-Interface, CLI), die in MKS verfügbar ist, auf verschiedene Operationen von MKS zu, [67]. Ziel ist es über die Kommandozeilen-Schnittstelle historische Daten zu einem bestimmten Zeitpunkt im Projekt zu exportieren. Wie in Kapitel 0 besprochen ist die einzige durchführbare Methode die Verwendung von MKS Reports. Dazu ist es notwendig dem Report die gewünschten Elemente zu übergeben, die exportiert werden sollen. Über die MKS-Benutzeroberfläche kann diese Auswahl entweder über das Auswählen von Elementen in einer Liste oder durch die Verwendung einer Query erfolgen. Dadurch, dass nur Elemente aufgrund bestimmter Eigenschaften exportiert werden sollen (Projekt, Typ, und so weiter) ist der Einsatz von Queries notwendig. Durch eine Einschränkung in der Kommandozeilen-Schnittstelle ist es allerdings nicht möglich Queries als Auswahlmöglichkeit für Reports zu verwenden, die historische Daten ermitteln. Die einzige Möglichkeit besteht deshalb darin die Elemente direkt im Report-Befehl mitzugeben. Deshalb müssen die Daten zuvor mithilfe eines eigenen „Query-Befehls“ über die CLI ermittelt werden und die IDs der Elemente gespeichert werden. Diese IDs können danach dem „Report-Befehl“ als Parameter mitgegeben werden. Dadurch, dass die IDs der Elemente direkt im Befehl mitgegeben werden müssen, muss auf die maximale Länge eines Kommandozeilen-Befehls geachtet werden. Dieser beträgt 2.047 Zeichen auf Windows 2000 und NT 4.0 Systemen, beziehungsweise 8.191 Zeichen auf Windows XP und höheren Systemen. Um die maximale Länge des Kommandozeilen-Befehls nicht zu überschreiten, ist es notwendig den Export der Elemente auf mehrere Befehle aufzuteilen. Um den Export kompatibel zu älteren

Windows-Systemen zu halten wurde von einer maximalen Länge von 2047 Zeichen ausgegangen. Die ID eines Elements besteht aus 6 Zeichen plus einem Zeichen als Abtrennung zwischen den IDs. Unter Berücksichtigung der restlichen Zeichen des Befehls wurde die Anzahl der exportierten Elemente auf 250 festgelegt. Damit ergibt sich eine maximale Länge von mitgegebenen IDs von 1750 ( $= 7 * 250$ ) Zeichen. Die restlichen 297 Zeichen sind für die restlichen Teile des Befehls reserviert. Die ermittelten Datensätze werden in Form einer XML-Datei von dem Report exportiert. Die Datensätze in den gesammelten XML-Dateien werden nach dem Export automatisiert in die Analyse-Datenbank importiert.

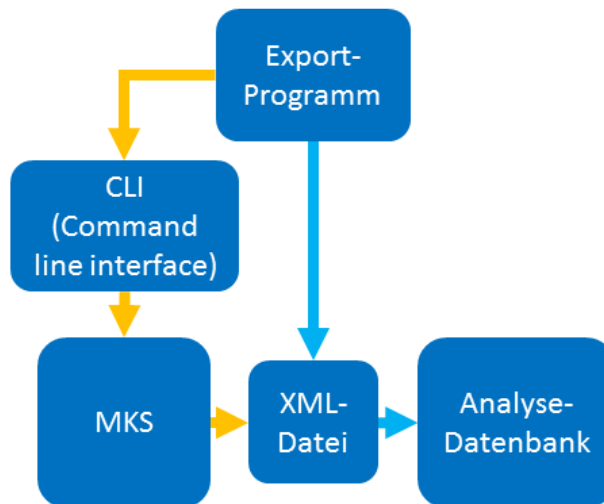


Abbildung 45: Funktionsweise des automatischen MKS-Exports

Um später die historischen Datensätze wieder dem dazugehörigen Tag zuordnen zu können wird ein eigenes Datumsfeld in der Datenbank mitgeführt. Auf diese Weise können durch einen einfachen SQL-Befehl alle historischen Daten eines bestimmten Tages ermittelt werden. Die Analyse-Datenbank selbst besteht aus mehreren Tabellen, die wiederum auf verschiedene Back-End-Dateien aufgeteilt sind (siehe Abbildung 46). Die Anforderungen, Testfälle und Arbeitspakete/Fehler werden aufgrund der Größe in eigenen Tabellen gespeichert. Die vierte Back-End-Datei „Verknüpfungen“ enthält Tabellen, die viele zu viele Beziehungen zwischen den Daten abbilden.

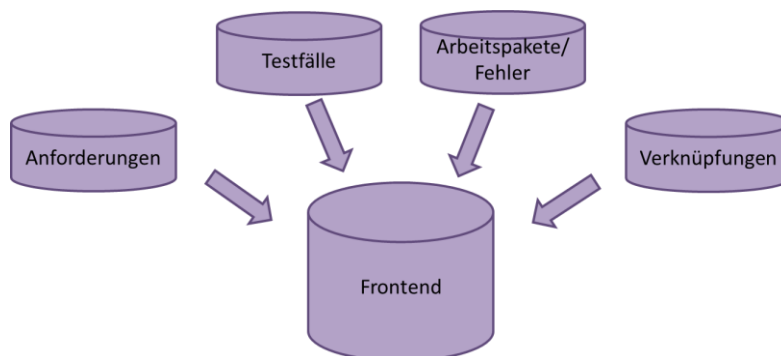


Abbildung 46: Tabellen und die Aufteilung auf die Back-End-Dateien

Um die Laufzeit des Exports zu optimieren werden Anforderungen, Testfälle und Arbeitspakete für einen einzelnen Tag parallel exportiert. Der Ablauf des Exports der einzelnen Elemente und des Imports wird von einem Export-Manager-Thread gesteuert. Zuerst wird ermittelt ob bereits Daten in der Datenbank vorhanden sind und bei welchem Tag der Export fortgesetzt werden muss. Danach

erstellt der Export-Manager eine Verbindung zum MKS-Server und startet die einzelnen Export-Threads. Die Export-Threads ermitteln über die Kommandozeilen-Schnittstelle von MKS die IDs der Elemente und führen die Report-Befehle aus. Die XML-Dateien, die durch die Report-Befehle erstellt werden, werden in einem Ordner gesammelt und später weiterverarbeitet. Der Export-Manager-Thread wartet solange bis alle Export-Threads ihre Arbeit beendet haben und beendet danach die Verbindung zum MKS-Server. Die abgelegten XML-Dateien werden danach vom Export-Manager eingelesen und in die einzelnen Tabellen der Datenbank eingefügt. Nach dem Import wählt der Export-Manager den nächsten Tag zum Exportieren aus und startet den Ablauf von neuem, solange bis das aktuelle Datum erreicht ist. Ein schematischer Ablauf des Export-Prozesses ist in Abbildung 47 dargestellt.

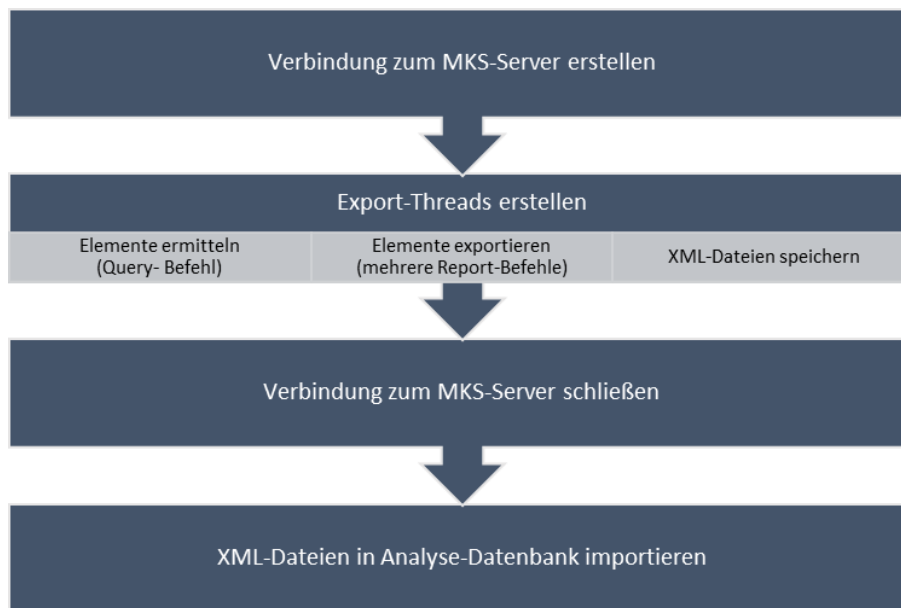


Abbildung 47: Ablauf des automatisierten Exports

#### 5.4 Programmablauf zum Erstellen der Auswertungen

Nachdem der Benutzer alle Einstellungen vorgenommen und die Auswertung gestartet hat verläuft der weitere Ablauf immer nach dem gleichen Schema. Zuerst wird eine Verbindung zu der Analyse-Datenbank aufgebaut und überprüft ob die Daten für den ausgewählten Auswertungszeitraum vorhanden sind. Nach der Überprüfung kann mit dem Abrufen der Daten aus der Datenbank begonnen werden. Um die Daten abzurufen werden gespeicherte Abfragen verwendet, die in Access vordefiniert wurden. Ein Vorteil dieser Methode ist, dass die Daten bereits in der Datenbank vorselektiert werden und dieser Schritt dadurch nicht von der Anwendung durchgeführt werden muss. Zudem war vor allem während der Entwicklung die zentrale Speicherung, die leichte Überprüfbarkeit und die leichte Änderbarkeit der Abfragen eine große Erleichterung. Auch die Zuteilung der Software-Releases und System-Freigaben zu den Kalenderdaten, die in einer Excel-Datei vorliegt, wird direkt über Access eingebunden und in die gespeicherten Abfragen mit einbezogen. Die Daten werden demnach von der Datenbank bereits richtig aufbereitet der Anwendung übermittelt. Die verbleibenden Schritte beziehen sich deshalb hauptsächlich auf die Darstellung und Anordnung der Daten in der MS Excel Arbeitsmappe. Die Vorlagen für die einzelnen Auswertungen müssen über den Pfad erreichbar, sein der in den Einstellungen angegeben wurde. Ebenso müssen die Software-Metriken in dem Metriken-Ordner abgelegt sein, der in den Einstellungen angegeben wurde. Die Daten aus der Datenbank werden nach einem definierten

Schema in die MS Excel Arbeitsmappe eingefügt. Danach werden noch die Chart-Objekte und die bedingten Formatierungen in die Excel-Arbeitsmappe hinzugefügt. Die fertige Arbeitsmappe wird in dem angegebenen Speicherort für die Auswertung gespeichert.

Zum Schluss wird noch genauer auf den Aufbau des Analyse-Tools eingegangen indem der Zusammenhang der einzelnen Komponenten und Klassen beschrieben und dargestellt wird (siehe Abbildung 48). Das Analyse-Tool besteht aus drei Benutzeroberflächen, zwei Handlern für den Zugriff auf die Datenbank und die Einstellungsdatei, sowie aus vier Klassen für die Erstellung der Auswertungen. Die MainForm bildet die Hauptbenutzeroberfläche und dient zur Steuerung des Analyse-Tools. Grundsätzlich besteht die Oberfläche aus fünf Tabs die jeweils die Einstellungen für die vier Auswertungsmethoden beinhalten, sowie aus einem Tab für den MKS-Export. Während des MKS-Exports wird dem Benutzer ein Fenster angezeigt, das Informationen über den Fortschritt des Exports und die Möglichkeit zum Abbrechen des Exports bietet. Die globalen Einstellungen für das Analyse-Tool, wie die Pfade zu der Datenbank, können über die ConfigForm vorgenommen werden. Die ConfigForm wird über den Menüpunkt „Einstellungen“ in der MainForm aufgerufen. Die genaue Funktionsweise des Analyse-Tools wurde bereits im Kapitel 5.2 diskutiert. Die beiden Handler-Klassen stellen Funktionen zur Kommunikation mit der Datenbank beziehungsweise mit der Config-Datei zur Verfügung. Demnach erfolgt die Kommunikation zwischen dem Tool und den externen Ressourcen ausschließlich über Handler. Mit dem DBHandler können die vordefinierten gespeicherten Abfragen in der Analyse-Datenbank ausgeführt werden. Der direkte Zugriff über selbst-definierte SQL-Befehle ist nicht möglich. Ähnlich wie der DBHandler enthält auch der Config-Handler Methoden um auf die Einstellungs-Datei zuzugreifen. Dabei besteht die Hauptaufgabe aus dem Serialisieren und Deserialisieren der Einstellungsdatei im XML-Format. Die Überprüfung der angegebenen Pfade und Einstellungen auf Richtigkeit wird ebenfalls bei jedem Zugriff automatisch vom Config-Handler durchgeführt. Die vier verbleibenden Klassen sind für die Erstellung der Auswertungen zuständig und werden durch die MainForm in einem eigenen Thread gestartet. Dadurch ist es möglich mehrere Auswertungen gleichzeitig erstellen zu lassen.

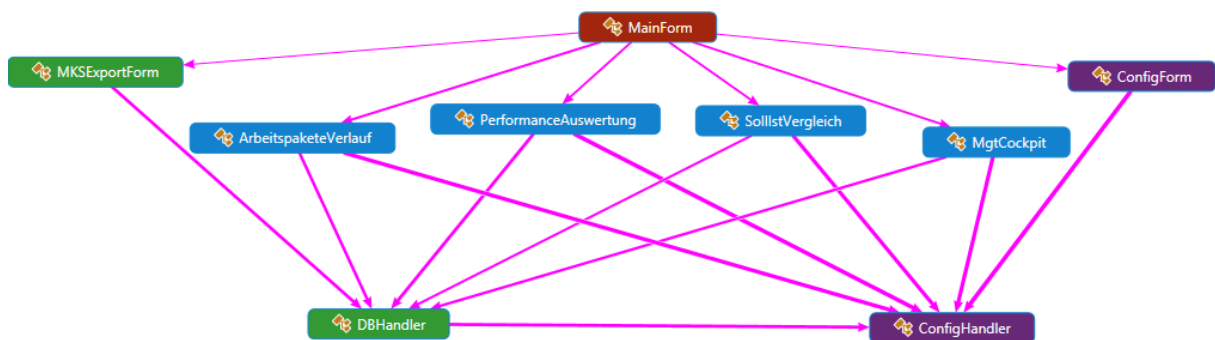


Abbildung 48: Vereinfachtes Klassendiagramm des Prototyps



## 6 Zusammenfassung

In dieser Masterarbeit wurde auf theoretische Grundlagen der automotiven Software und des Entwicklungsprozesses, sowie der Qualitätssicherung und dem Einsatz von Software-Metriken eingegangen. In einem nächsten Schritt wurde die Verwendung von Entwicklungs- und Testdaten zur Bewertung automotiver Software, anhand eines Beispielprojektes, bei einem weltweit tätigen Automobilzulieferer veranschaulicht. Zu diesem Zweck wurde eine Analyse des Entwicklungsprozesses und die Identifikation relevanter Daten aus dem Informationssystem vorgenommen und ein Idealkonzept daraus erstellt. Dieses wurde danach auf Basis der Analyse der vorhandenen Entwicklungs- und Testdaten auf die Machbarkeit überprüft. Zugleich wurden auch umsetzbare Vorgehensweisen aufbauend auf dem Idealkonzept entwickelt. Zur Spezifizierung der Analysemethoden wurden Fragestellungen des Projektmanagements, der Qualitätssicherung und des Managements gesammelt und anhand dieser die Methoden spezifiziert. Dadurch konnte eine hohe Akzeptanz und Relevanz der Analysemethoden sichergestellt werden. Eine praktische Umsetzung wurde mithilfe eines lauffähigen Prototyps für das Beispielprojekt veranschaulicht.

### 6.1 Ausblick

Die in dieser Masterarbeit vorliegenden Analysemethoden auf Basis der Entwicklungs- und Testdaten wurden in Form eines Prototyps für ein Beispielprojekt umgesetzt. Um diese auf mehrere Projekte auszuweiten ist eine erweiterte Analyse der Entwicklungs- und Testdaten aller Projekte notwendig. Insbesondere müssen alle entscheidenden Attribute, wie anhand des Beispielprojektes veranschaulicht, ermittelt und geprüft werden. Der Ansatz Interessensgruppen zu identifizieren und Fragestellungen zu formulieren hat sich in dieser Masterarbeit bewährt und sollte für die Weiterentwicklung wieder verwendet werden. Zugleich ist eine technische Umsetzung auf Basis einer Client-Server Architektur zu prüfen. Durch die logische Trennung von Analyse- und Export-Tool im Prototyp wird dieser mögliche Schritt unterstützt. Für eine zukünftige Implementierung im operativen Einsatz sollte, aufgrund der Einschränkungen von MS Access, ein eigener Datenbankserver für die Analyse-Datenbank verwendet werden.

# Abbildungsverzeichnis

Abbildung 1: AUTOSAR-Architektur, vgl. [1] .....	8
Abbildung 2: Klassifizierung von Projekten in Projekttypen, [9] .....	10
Abbildung 3: V-Modell des Software-Entwicklungsprozesses für Automobil-Anwendungen, [18].....	12
Abbildung 4: Klassifizierung von In-the-Loop-Testsystemen, [1] .....	15
Abbildung 5: Magisches Dreieck der Qualität, [7].....	17
Abbildung 6: Drei Dimensionen von Software, [27].....	19
Abbildung 7: Quellen der Softwaregrößenmessung, [27].....	20
Abbildung 8: Testfortschritt S-Kurven-Diagramm, [29].....	31
Abbildung 9: Die drei Stufen der GQM-Methode, [58] .....	33
Abbildung 10: Diagrammarten und ihre Anwendungsmöglichkeiten, [56] .....	35
Abbildung 11: Beispiel eines Netzdiagramms mit vier Kategorien und zwei Serien, vgl. [61], [62] .....	36
Abbildung 12: Grundlegende Software-Architektur des Projekts, [10] .....	37
Abbildung 13: Modifiziertes V-Modell zur Entwicklung automotiver Software .....	38
Abbildung 14: Anforderungs-Workflow, [66].....	39
Abbildung 15: Testfall-Workflow, [66] .....	40
Abbildung 16: Arbeitspaket-Workflow, [66] .....	41
Abbildung 17: Beispielhafter Verlauf von Entwicklungs- und Testdaten.....	43
Abbildung 18: Bewertung von Software-Versionen mithilfe von Software-Metriken, vgl. [61], [62] ..	43
Abbildung 19: Hinterlegungsgrad der Attribute.....	44
Abbildung 20: Beziehungen zwischen den Daten .....	45
Abbildung 21: Verfügbare Code-Metriken auf Modul-Ebene .....	47
Abbildung 22: Verfügbare Code-Metriken auf Gesamtsoftware-Ebene.....	48
Abbildung 23: Strategien für die Metriken-Nutzung.....	49
Abbildung 24: Kombination von Code- und Prozess-Metriken auf Gesamtsoftware-Ebene, vgl. [61].	50
Abbildung 25: Fragen und Ansichten auf die Entwicklungs- und Testdaten, [62] .....	51
Abbildung 26: Auswertung der Arbeitspakete .....	52
Abbildung 27: Status-Verlauf von Arbeitspaketen, vgl. [61] .....	53
Abbildung 28: Geplante Software-Releases von Arbeitspaketen, vgl. [61] .....	53
Abbildung 29: Quantile, Median und Mittelwert in einer asymmetrischen Verteilung , vgl. [61] .....	55
Abbildung 30: Performance-Auswertung mithilfe von deskriptiver Statistik, [61] .....	56
Abbildung 31: Diagramm der Performance-Auswertung.....	57
Abbildung 32: Soll-Ist-Vergleich, [61] .....	58
Abbildung 33: Management-Cockpit, vgl. [62].....	59
Abbildung 34: Bewertungsschema für Systemfreigaben, vgl. [62] .....	60
Abbildung 35: Auswahl und Gewichtung der Metriken .....	61
Abbildung 36: Aufteilung der Access-Datenbank.....	62
Abbildung 37: Grundlegendes Konzept des Prototyps .....	63
Abbildung 38: Empfohlene Ordnerstruktur .....	64
Abbildung 39: Einstellungen.....	65
Abbildung 40: Verlauf der Arbeitspakete.....	65
Abbildung 41: Performance-Auswertung.....	66

Abbildung 42: Soll-Ist-Vergleich .....	67
Abbildung 43: Management-Cockpit .....	67
Abbildung 44: MKS-Export .....	69
Abbildung 45: Funktionsweise des automatischen MKS-Exports .....	70
Abbildung 46: Tabellen und die Aufteilung auf die Back-End-Dateien .....	70
Abbildung 47: Ablauf des automatisierten Exports .....	71
Abbildung 48: Vereinfachtes Klassendiagramm des Prototyps .....	72

# Tabellenverzeichnis

Tabelle 1: Vergleich zwischen PC-Software und Steuergeräte-Software, [7]. .....	4
Tabelle 2: Klassifizierung von automotiver Software, [2], [5] .....	4
Tabelle 3: ISO 9126 – Software Qualitätskriterien, vgl. [21] .....	16
Tabelle 4: Anzahl der Codezeilen von vier Methoden in zwei unterschiedlichen Projekten, vgl. [59].	34
Tabelle 5: Zwei Projekte mit gewichteten Mittelwert der Anzahl der Codezeilen, vgl. [59] .....	34
Tabelle 6: Zuordnung der Elemente zu Entwicklungs-Ebenen.....	45
Tabelle 7: Zuordnung der Elemente zu Modul- und Gesamtsoftware-Ebene .....	46
Tabelle 8: Einteilung der Elemente in „done“ und „in work“ .....	54

# Literaturverzeichnis

- [1] Reif, K.: "Bosch Autoelektrik und Autoelektronik - Bordnetze, Sensoren und elektronische Systeme; mit 43 Tabellen", 6th ed, ISBN 978-3-8348-9902-6, Vieweg + Teubner, Wiesbaden, Deutschland, 2011.
- [2] Pretschner, A.; Broy, M.; Kruger, I. H. und Stauner, T.: "Software Engineering for Automotive Systems: A Roadmap" in Future of Software Engineering, pp. 55–71, ISBN 0-7695-2829-5, IEEE Computer Society, Los Alamitos, CA, USA, 2007.
- [3] Broy, M.: "Challenges in automotive software engineering" in Proceedings of the 28th international conference on Software engineering, pp. 33–42, ISBN 1-59593-375-1, ACM, New York, NY, USA, 2006.
- [4] Shaout, A.; Arora, M. und Awad, S.: "Automotive software development and management" in International Computer Engineering Conference, pp. 9–15, ISBN 978-1-61284-185-4, IEEE, Piscataway, NJ, USA, 2010.
- [5] Broy, M.; Kruger, I. H.; Pretschner, A. und Salzmann, C.: "Engineering Automotive Software", Proceedings of the IEEE, vol. 95, no. 2, pp. 356–373, 2007.
- [6] Mössinger, J.: "Software in Automotive Systems", IEEE Software, vol. 27, no. 2, pp. 92–94, 2010.
- [7] Borgeest, K.: "Elektronik in der Fahrzeugtechnik - Hardware, Software, Systeme und Projektmanagement", 3rd ed, ISBN 978-3-8348-2145-4, Springer Vieweg, Wiesbaden, Deutschland, 2014.
- [8] Goto, M.: "Innovation of automotive software development" in Proceedings of the 17th International Software Product Line Conference, pp. 5–6, ISBN 978-1-4503-1968-3, 2013.
- [9] IABG - Industrieanlagen-Betriebsgesellschaft mbH: "Das V-Modell", Online: <http://v-modell.iabg.de/index.php> [Zugriff am 03.08.2015].
- [10] Schäuffele, J. und Zurawka, T.: "Automotive Software Engineering - Grundlagen, Prozesse, Methoden und Werkzeuge effizient einsetzen", 5th ed, ISBN 3834824704, Springer Vieweg, Wiesbaden, Deutschland, 2013.
- [11] MathWorks Deutschland: "Simulink - Simulation und Model-Based Design", Online: <http://de.mathworks.com/products/simulink/> [Zugriff am 30.07.2015].
- [12] MSC Software: "Easy5", Online: [www.mscsoftware.com/de/product/easy5](http://www.mscsoftware.com/de/product/easy5) [Zugriff am 30.07.2015].
- [13] ETAS: "ASCET", Online: [www.etas.com/de/products/ascet\\_software\\_products.php](http://www.etas.com/de/products/ascet_software_products.php) [Zugriff am 30.07.2015].
- [14] Thaller, G. E.: "Software-Test - Verifikation und Validation", 2nd ed, ISBN 3-88229-183-4, Heise, Hannover, Deutschland, 2000.
- [15] AUTOSAR development partnership: "AUTOSAR: Automotive Open System Architecture", Online: [www.autosar.org](http://www.autosar.org) [Zugriff am 30.07.2015].
- [16] JASPAR: "JASPAR: Japan Automotive Software Platform and Architecture", Online: [www.jaspar.jp](http://www.jaspar.jp) [Zugriff am 30.07.2015].
- [17] Bundesverwaltungsamt: "V-Modell XT", Online: [http://www.bva.bund.de/DE/Organisation/Abteilungen/Abteilung\\_BIT/Leistungen/IT\\_Standards/VMoellXT/node.html](http://www.bva.bund.de/DE/Organisation/Abteilungen/Abteilung_BIT/Leistungen/IT_Standards/VMoellXT/node.html) [Zugriff am 05.08.2015].

- [18] H. Wallentowitz, Ed. "Handbuch Kraftfahrzeugelektronik - Grundlagen, Komponenten, Systeme, Anwendungen ; mit zahlreichen Tabellen", 1st ed, Vieweg, Wiesbaden, Deutschland, 2006.
- [19] Plummer, A. R.: "Model-in-the-Loop Testing", Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering, vol. 220, no. 3, pp. 183–199, [http://www.researchgate.net/publication/245389264\\_Model-in-the-Loop\\_Testing](http://www.researchgate.net/publication/245389264_Model-in-the-Loop_Testing), 2006.
- [20] Mennenga, M.; Dziobek, C. und Bahous, I.: "Modell- und Software-Verifikation vereinfacht - Zeitkritische Software-Anforderungen effizient testen", Elektronik automotive, no. 4, pp. 45–49, 2009.
- [21] O'Regan, G.: "Introduction to software quality", 2014th ed, ISBN 978-3-319-06105-4, Springer International Publishing, Cham, Deutschland, 2014.
- [22] Wieczorek, M.; Vos, D. und Bons, H.: "Systems and Software Quality - The next step for industrialisation", ISBN 978-3-642-39970-1, Springer Berlin Heidelberg, Berlin, Heidelberg, Deutschland, 2014.
- [23] van der Aalst, W.; Mylopoulos, J.; Rosemann, M.; Shaw, M. J.; Szyperski, C.; Winkler, D.; Biffli, S. und Bergsmann, J.: "Software Quality. Model-Based Approaches for Advanced Software and Systems Engineering", ISBN 978-3-319-03601-4, Springer International Publishing, Cham, Deutschland, 2014.
- [24] Fenton, N. E. und Pfleeger, S. L.: "Software metrics - A rigorous and practical approach", 2nd ed, ISBN 0534954251, PWS Pub, Boston, USA, 1997.
- [25] van Solingen, R. und Berghout, E.: "The goal/question/metric method - A practical guide for quality improvement of software development", ISBN 9780077095536, McGraw-Hill, London, UK; Chicago, USA, 1999.
- [26] Goodman, P.: "Practical Implementation of software metrics", ISBN 0077076656, McGraw-Hill, London, UK; New York, USA, 1993.
- [27] Sneed, H. M.; Seidl, R. und Baumgartner, M.: "Software in Zahlen - Die Vermessung von Applikationen", ISBN 978-3-446-42175-2, Hanser, München, Deutschland, 2010.
- [28] Perlis, A. J.; Sayward, F. und Shaw, M.: "Software metrics - An analysis and evaluation", ISBN 0262160838, MIT Press, Cambridge, Mass, USA, 1981.
- [29] Kan, S. H.: "Metrics and models in software quality engineering", 2nd ed, ISBN 9780201729153, Addison-Wesley, Boston, USA, 2003.
- [30] Gwak, T. und Jang, Y.: "An Empirical Study on SW Metrics for Embedded System" in Lecture Notes in Computer Science, vol. 3966, "Software Process Change", pp. 302–313, Hutchison, D.; Kanade, T.; Kittler, J.; Kleinberg, J. M.; Mattern, F.; Mitchell, J. C.; Naor, M.; Nierstrasz, O.; Pandu Rangan, C.; Steffen, B.; Sudan, M.; Terzopoulos, D.; Tygar, D.; Vardi, M. Y.; Weikum, G.; Wang, Q.; Pfahl, D.; Raffo, D. M. und Wernick, P, Eds. ISBN 978-3-540-34201-4, Springer Berlin Heidelberg, Berlin, Heidelberg, Deutschland, 2006.
- [31] IEEE: "Standard Glossary of Software Engineering Terminology - ANSI/IEEE Std 729-1983", Institute of Electrical and Electronics Engineers, New York, USA, 1983.
- [32] Moore, J. W.: "Software engineering standards - A user's road map", ISBN 0-8186-8008-3, IEEE Computer Society, Los Alamitos, CA, USA, 1998.
- [33] Halstead, M. H.: "Elements of software science", ISBN 0444002057, North-Holland Publishing, New York, USA, 1977.
- [34] Gilb, T.: "Software Metrics", ISBN 914412631X, Studentlitteratur, Lund, Sweden, 1976.
- [35] McCabe, T. J.: "A Complexity Measure", IEEE Transactions on Software Engineering, vol. SE-2, no. 4, pp. 308–320, 1976.

- [36] Fenton, N. E.: "Software metrics - A rigorous approach", ISBN 0442313551, Chapman and Hall, London, UK, 1991.
- [37] Crosby, P. B.: "Quality is free - The art of making quality certain", ISBN 9780070145122, McGraw-Hill, New York, NY, USA, 1979.
- [38] Juran, J. M. und Gryna, F. M.: "Quality planning and analysis - From product development through use", McGraw-Hill, New York, NY, USA, 1970.
- [39] Deming, W. E.: "Out of the crisis", ISBN 0911379010, M.I.T. Press, Cambridge, Mass, USA, 1982.
- [40] IEEE: "Standard Glossary of Software Engineering Terminology", ISBN 9780738103914, Institute of Electrical and Electronics Engineers, New York, NY, USA, 1990.
- [41] Boehm, B.; Brown, J.; Kaspar, H. und Lipow, M.: "Characteristics of Software Quality", ISBN 0-444-85105-4, North-Holland Publishing, Amsterdam, Niederlande, 1978.
- [42] McCall, J.; Richards, P. und Walters, G.: "Factors in Software Quality" in vol. 1, "Concepts and Definitions of Software Quality", General Electric NTIS Publishing, Springfield, Va, USA, 1977.
- [43] International Standards Organisation: "ISO/IEC: Software Product Evaluation - Quality Characteristics and Guidelines for their Use - ISO/IEC Standard 9126", International Standards Organisation, Genf, Schweiz, 1994.
- [44] Gries, D.: "Compiler construction for digital computers", ISBN 0-471-32776-X, Wiley, New York, NY, USA, 1971.
- [45] Basili, V. R.: "Tutorial on models and metrics for software management and engineering", Computer Society Press, Los Alamitos, CA, USA, 1980.
- [46] M.M. Lehman and L.A. Belady, Eds. "Program evolution - Processes of software change", Academic Press, London, UK, 1985.
- [47] Laird, L. M. und Brennan, M. C.: "Software measurement and estimation - A practical approach", ISBN 9781280468445, John Wiley & Sons, Hoboken, NJ, USA, 2006.
- [48] Gerould, K. F.: "The Virtue of Simplicity", Encyclopedia of Science, Chicago, USA, 1935.
- [49] Zuse, H.: "Software complexity - Measures and methods", ISBN 311012226X, W. de Gruyter, Berlin, Deutschland; New York, NY, USA, 1991.
- [50] Simon, F.; Mohaupt, T. und Seng, O.: "Code-quality-Management - Technische Qualität industrieller Softwaresysteme transparent und vergleichbar gemacht", 1st ed, ISBN 9783898643887, Dpunkt-Verl, Heidelberg, Deutschland, 2006.
- [51] Coleman, D.; Ash, D.; Lowther, B. und Oman, P.: "Using metrics to evaluate software system maintainability", Computer, vol. 27, no. 8, pp. 44-49, 1994.
- [52] Sneed, H. M.; Hasitschka, M. und Teichmann, M.-T.: "Software-Produktmanagement - Wartung und Weiterentwicklung bestehender Anwendungssysteme", 1st ed, ISBN 3-89864-274-7, Dpunkt Verlag, Heidelberg, Deutschland, 2005.
- [53] Ebert, C.: "Systematisches Requirements-Engineering und Management - Anforderungen ermitteln, spezifizieren, analysieren und verwalten", 2nd ed, ISBN 3898645460, Dpunkt-Verl, Heidelberg, 2008.
- [54] International Standards Organisation: "ISO/IEC: Software Product Quality Requirements and Evaluation (SQUARE) - Quality Requirements - ISO/IEC Standard 25030-2007", International Standards Organisation, Genf, Schweiz, 2007.
- [55] Ebert, C.: "Systematisches Requirements Management - Anforderungen ermitteln, spezifizieren, analysieren und verfolgen", 1st ed, ISBN 3-89864-336-0, dpunkt-Verl, Heidelberg, Deutschland, 2005.

- [56] Sowa, A. und Fedtke, S.: "Metriken - der Schlüssel zum erfolgreichen Security und Compliance Monitoring - Design, Implementierung und Validierung in der Praxis", ISBN 978-3-8348-1480-7, Vieweg + Teubner, Wiesbaden, Deutschland, 2011.
- [57] Basili, V. R.: "Software modeling and measurement - The goal/question/metric paradigm", University of Maryland, College Park, Md, USA, 1992.
- [58] van Solingen, R.; Basili, V. R.; Caldiera, G. und Rombach, H. D.: "Goal Question Metric (GQM) Approach" in "Encyclopedia of Software Engineering", Marciniak, J. J, Ed. ISBN 0471028959, John Wiley & Sons, Inc, Hoboken, NJ, USA, 2002.
- [59] Mordal, K.; Anquetil, N.; Laval, J.; Serebrenik, A.; Vasilescu, B. und Ducasse, S.: "Software quality metrics aggregation in industry", Journal of Software: Evolution and Process, vol. 25, no. 10, pp. 1117–1135, 2013.
- [60] Vasilescu, B.; Serebrenik, A. und van den Brand, M.: "By no means: a study on aggregating software metrics" in Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics, p. 23, ISBN 978-1-4503-0593-8, Association for Computing Machinery, New York, NY, USA, 2011.
- [61] Ernst, M.; Erlachner, S.; Hirz, M.; Fabian, J. und Wotawa, F.: "Analysis Methods in the Development Process of Mechatronic Drivetrain Systems with Special Focus on Automotive Software" in The 19th World Multi-Conference on Systemics, Cybernetics and Informatics WMSCI, pp. 63–68, ISBN 978-1-941763-25-4, International Institute of Informatics and Systemics, Orlando, Florida, USA, 2015.
- [62] Ernst, M.; Erlachner, S.; Hirz, M. und Fabian, J.: "Optimisation of Automotive Software Quality Management by Use of Analysis Methods in the Development Process of Mechatronic Systems" in International Conference on Advances in Software, Control and Mechanical Engineering, pp. 21–25, ISBN 978-93-84422-37-0, Universal Researchers in Science and Technology, Antalya, Türkei, 2015.
- [63] PTC Inc.: "PTC - Parametric Technology Corporation", Online: [www.ptc.com](http://www.ptc.com) [Zugriff am 25.09.2015].
- [64] PTC Inc.: "PTC Integrity", Online: <http://de.ptc.com/application-lifecycle-management/integrity> [Zugriff am 25.09.2015].
- [65] Klostermann, M.: "Schematical MKS Document Structure: Feature View, focused on SW", 2014.
- [66] Klostermann, M.: "MKS Items: Annotated Masks and Workflows", 2013.
- [67] PTC Inc.: "MKS Integrity 2009 (SP6) CLI Reference for Workflows and Documents"
- [68] MathWorks Deutschland: "Polyspace - Static Analysis Tool for C/C++ and Ada", Online: <http://de.mathworks.com/products/polyspace/> [Zugriff am 27.09.2015].
- [69] MES - Model Engineering Solutions: "M-XRAY - Komplexität von Simulink-Modellen messen und bewerten", Online: <http://www.model-engineers.com/de/m-xray.html> [Zugriff am 27.09.2015].
- [70] Automotive SIG: "ISO/IEC 15504 Automotive SPICE Process Reference Model - ISO/IEC 15504 v4.5", 2010.
- [71] H. Lohninger: "Fundamentals of Statistics", Online: <http://statistics4u.info/> [Zugriff am 01.10.2015].
- [72] Microsoft: "Datenbanksoftware- und anwendungen | Microsoft Access", Online: <https://products.office.com/de-at/access> [Zugriff am 15.11.2015].



- [73] Microsoft: "MS Access - Spezifikationen", Online: <https://support.office.com/de-de/article/Access-2010-Spezifikationen-1e521481-7f9a-46f7-8ed9-ea9dff1fa854?ui=de-DE&rs=de-DE&ad=DE> [Zugriff am 09.10.2015].
- [74] Microsoft: "MS Access - Aufteilen einer Datenbank", Online: <https://support.office.com/de-de/article/Aufteilen-einer-Datenbank-3015ad18-a3a1-4e9c-a7f3-51b1d73498cc> [Zugriff am 09.10.2015].
- [75] Microsoft: "Zugreifen auf Office-Interop-Objekte", Online: <https://msdn.microsoft.com/de-de/library/dd264733.aspx> [Zugriff am 14.10.2015].
- [76] M. Asal: "Visual Basic für Applikationen - Das VBA-Tutorial", Online: <http://www.vba-tutorial.de/> [Zugriff am 14.10.2015].