Institut für Softwaretechnologie

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3551

Entwicklungsprozess für qualifizierbare Softwarewerkzeuge nach ISO 26262

Marius Blascheck

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Stefan Wagner
Betreuer/in:	Dipl.-Ing. Jan-Peter Ostberg Dr. Roberto Kretschmer (TWT)
Beginn am:	1. August 2013
Beendet am:	31. August 2015
CR-Nummer:	D.2.9, K.6.3

Kurzfassung

Um die hohen Qualitätsanforderungen an Softwarewerkzeuge für die Entwicklung eingebetteter Systeme im Automobilumfeld zu gewährleisten, wurde in dieser Arbeit in Zusammenarbeit mit dem Unternehmen TWT ein Qualitätsprozess definiert, der die Nachweisbarkeit von Anforderungen sowie eine Qualifizierung nach dem Sicherheitsstandard ISO 26262 ermöglicht. Hierfür wurden zunächst die Vorgaben des Sicherheitsstandards zur Softwareentwicklung und zur Qualifizierung von Softwarewerkzeugen analysiert. Danach wurden die bestehenden Softwareentwicklungsprozesse bei der TWT untersucht. Aus beidem wurden anschließend Anforderungen an den Qualitätsprozess abgeleitet, so dass dieser sowohl die Anforderungen des Sicherheitsstandards erfüllt als auch sich in die bestehenden Entwicklungsprozesse bei TWT einfügt. Das Konzept des Qualitätsprozesses basiert auf dem im Sicherheitsstandard verwendeten V-Modell, erweitert dieses jedoch um den Einsatz eines kontinuierlichen, testorientierten Requirements Engineerings, einer kontinuierlichen Integration und um Quality Gates, die die Phasen des V-Modells voneinander trennen. Durch das Requirements Engineering und die Quality Gates werden sowohl die Softwareanforderungen validiert als auch die Anforderungen und Vorgaben des Sicherheitsstandards auf ihre Einhaltung überprüft. Durch den Einsatz einer kontinuierlichen Integration, fügt sich der Qualitätsprozess in die bestehenden Entwicklungsprozesse bei TWT ein. Abschließend wurde der ausgearbeitete Qualitätsprozess in einem Expertenreview evaluiert. Die Evaluation ergab, dass der Qualitätsprozess die Anforderungen erfüllt. Des Weiteren gab es Anregungen für eine Erweiterung des Qualitätsprozesses.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Dieses Kapitel umfasst eine Einleitung in den thematischen Hintergrund der Diplomarbeit sowie eine Beschreibung der Problemstellung und der damit verbundenen Aufgaben. Anschließend folgt eine Gliederung der Arbeit sowie die Definition der in dieser Arbeit verwendeten Notation zur Zitierung des Sicherheitsstandards ISO 26262 (2011).

1.1. Hintergrund, Problemstellung und Aufgaben

Die TWT GmbH bietet Dienstleistungen bei der Systementwicklung in den Branchen Automotive, Aerospace, Healthcare und Energy an. Dabei stellt TWT Expertise im IT-, Engineering- und Beratungsbereich zur Verfügung und unterstützt die Entwicklung eingebetteter Systeme unter Einhaltung aktueller Sicherheitsstandards wie ISO 26262. TWT ist darüber hinaus in den internationalen Forschungsprojekten VeTeSS und openETCS involviert und setzt sich dabei aktiv für die Entwicklung standardisierter Werkzeuge, Prozesse und Methoden für sicherheitsrelevante, eingebettete Systeme in Fahrzeugen ein. Die vorliegende Arbeit stellt in diesem Zusammenhang einen wichtigen Beitrag dar, da untersucht wird, wie agile Softwareentwicklungsmethoden mit den Qualitätsanforderungen vereinbar sind, die sich unter anderem aus dem Sicherheitsstandard ISO 26262 ergeben.

Das Hauptaugenmerk der vorliegenden Arbeit liegt dabei auf der Definition eines Entwicklungsprozesses für Softwarewerkzeuge, die für die Entwicklung sicherheitsrelevanter, eingebetteter Systeme in Fahrzeugen eingesetzt werden. TWT möchte bei der Entwicklung eines solchen Softwarewerkzeugs sicherstellen, dass eine Nachweisbarkeit der Anforderungen sowie eine Qualifizierung nach ISO 26262 möglich ist. Dafür soll der Entwicklungsprozess den Anforderungen und Vorgaben des Sicherheitsstandards ISO 26262 zur Softwareentwicklung und Qualitätssicherung genügen. Außerdem soll er sich in die bestehenden agilen Softwareentwicklungsprozesse bei TWT einfügen. Nach Möglichkeit sollen auch die sich bereits bei TWT in Benutzung befindenden Softwarewerkzeuge zur Qualitätssicherung eingesetzt werden.

Hierbei stellt sich die Frage, inwieweit sich agile Softwareentwicklungsprozesse, das im Sicherheitsstandard ISO 26262 definierte V-Modell sowie die strengen Sicherheitsanforderungen des Standards miteinander vereinbaren lassen. Außerdem gilt es zu untersuchen, wie durch einen geeigneten Softwareentwicklungsprozess die spätere Qualifizierung gemäß dem Sicherheitsstandard ISO 26262 unterstützt werden kann.

Daraus ergeben sich folgende Aufgaben für die vorliegende Arbeit. Die bei TWT eingesetzten Softwareentwicklungsprozesse und Softwarewerkzeuge sollen mit dem Ziel untersucht werden, die Vereinbarkeit mit dem im Sicherheitsstandard ISO 26262 definierten Softwareentwicklungsprozess bewerten zu können. Dies soll vor allem in Hinsicht auf die Qualifizierung damit entwickelter Softwarewerkzeuge gemäß den Vorgaben des Sicherheitsstandards ISO 26262 geschehen. Basierend auf den Ergebnissen der Untersuchung soll ein Softwareentwicklungsprozess definiert werden, der die Qualitätssicherung für die Neuentwicklung qualifizierbarer Softwarewerkzeuge beschreibt. Wo nötig, sollen auch Empfehlungen für neue Softwarewerkzeuge zur Qualitätssicherung ausgesprochen werden. Der Qualitätsprozess soll nach seiner Fertigstellung in einem Expertenreview evaluiert werden.

1.2. Aufbau der Arbeit

Die Arbeit ist in folgender Weise gegliedert:

- Beschreibung der für das Thema der Arbeit relevanten Grundlagen (Kapitel 2).
- Analyse und Bewertung der von TWT eingesetzten Softwarequalitätssicherungsprozesse und Softwarewerkzeuge (Kapitel 3).
- Konzeption eines Qualitätsprozesses für die Neuentwicklung qualifizierbarer Softwarewerkzeuge (Kapitel 4).
- Spezifizierung eines Qualitätsprozesses für die Neuentwicklung qualifizierbarer Softwarewerkzeuge (Kapitel 5).
- Evaluation des Qualitätsprozesses durch ein Expertenreview (Kapitel 6).
- Zusammenfassung der Ergebnisse der Arbeit und ein Ausblick auf die weiteren Schritte (Kapitel 7).

1.3. Notation

Der Sicherheitsstandard ISO 26262 (2011) ist in zehn Teilen organisiert. Jeder Teil besteht wiederum aus mehreren Klauseln, die die einzelnen Anforderungen des Standards definieren. Diese Arbeit nutzt für die Zitierung des Standards folgende drei Stile:

Teil – z. B. ISO 26262-8 repräsentiert den achten Teil der ISO 26262.

Teil, Seite – z. B. ISO 26262-6, S. 13 repräsentiert Seite 13 des sechsten Teils der ISO 26262.

Teil, Klausel – z. B. ISO 26262-8, Klausel 11.2 repräsentiert Klausel 11.2 des achten Teils der ISO 26262.

2. Grundlagen

In diesem Kapitel werden die wichtigen Themen dieser Arbeit vorgestellt. Zunächst erfolgt eine Übersicht des Sicherheitsstandard ISO 26262 (Abschnitt 2.1), mit Schwerpunkt auf die Softwareentwicklung nach ISO 26262 (Abschnitt 2.1.2) und der Qualifizierung von Softwarewerkzeugen nach ISO 26262 (Abschnitt 2.1.3). Anschließend folgt eine Übersicht aller für die Arbeit relevanten Methoden und Prozesse der Softwaretechnik. Es werden die Softwareentwicklungsprozesse (Abschnitt 2.2), der Einsatz von Softwarewerkzeugen (Abschnitt 2.3) in der Softwareentwicklung und die verschiedenen Methoden der Softwarequalitätssicherung (Abschnitt 2.4) vorgestellt. Bei den Methoden handelt es sich um die konstruktive Softwarequalitätssicherung (Abschnitt 2.4.1), die analytische Softwarequalitätssicherung (Abschnitt 2.4.2), das Requirements Engineering (Abschnitt 2.4.3) und die Continuous Integration (Abschnitt 2.4.4).

2.1. Sicherheitsstandard ISO 26262

Der Standard ISO 26262 (2011) beachtet die speziellen Anforderungen an elektrische und/oder elektronische (E/E) Systeme im Automobilumfeld und ist damit eine Anpassung des IEC 61508 (2010) Standards. Der Standard gilt für alle Aktivitäten des Sicherheitslebenszyklus sicherheitsrelevanter E/E-Systeme inklusive deren Softwarekomponenten. Die Systemsicherheit wird durch eine Reihe von Sicherheitsmaßnahmen erreicht, die auf verschiedenen Ebenen des Entwicklungsprozesses eingesetzt werden. Der Rahmen für die funktionale Sicherheit von E/E-Systemen, den der Standard bereitstellt, beinhaltet folgende Aspekte:

- Einen Sicherheitslebenszyklus im Automobilumfeld, dessen Aktivitäten in den verschiedenen Phasen (Management, Entwicklung, Produktion, Einsatz, Wartung und Stilllegung) angepasst werden können.
- Einen automobilspezifischen risikobasierten Ansatz zur Bestimmung der Automotive Safety Integrity Levels (ASIL), die dazu verwendet werden, passende Anforderungen des Standards zu spezifizieren, um ein unangemessenes Restrisiko zu vermeiden.
- Anforderungen an die Validierung der Sicherheit, um sicherzustellen dass diese ausreichend und akzeptabel ist.
- Anforderungen an die Zusammenarbeit mit Zulieferern.

Die funktionale Sicherheit wird durch den Entwicklungsprozess, den Produktions- und Wartungsprozess sowie den Managementprozess beeinflusst. Der Entwicklungsprozess basiert auf dem V-Modell als Referenzprozessmodell für die verschiedenen Phasen der Produktentwicklung. Bei den einzelnen Phasen handelt es sich um die Anforderungsspezifikation, den Entwurf, die Implementierung, die Integration, die Verifikation, die Validierung und die Konfiguration. Die Sicherheitsaspekte sind dabei mit den normalen funktions- und qualitätsorientierten Entwicklungsaktivitäten und Arbeitsergebnissen verknüpft, wobei sich der Standard ausschließlich mit den sicherheitsrelevanten Aspekten der Entwicklungsaktivitäten und Arbeitsergebnisse befasst.

2.1.1. Sicherheitslebenszyklus nach ISO 26262

Der ISO 26262 Sicherheitslebenszyklus umfasst die grundsätzlichen Sicherheitsaktivitäten während der Konzeptphase (vgl. ISO 26262-3), der System-, Hardware- und Softwareentwicklung (vgl. ISO 26262-4, ISO 26262-5 und ISO 26262-6), der Produktion und des Einsatzes (vgl. ISO 26262-7), der Wartung sowie der Stilllegung. Eine Anpassungen des Sicherheitslebenszyklus, inklusive der Iteration einzelner Phasen, ist möglich. Der Standard spezifiziert sowohl Anforderungen in Bezug auf bestimmte Phasen des Sicherheitslebenszyklus als auch Anforderungen, die für mehrere oder alle Phasen des Sicherheitslebenszyklus gelten. Die Managementaufgaben Planung, Koordination und Dokumentation der Sicherheitsaktivitäten sind von entscheidender Bedeutung und müssen in allen Phasen des Sicherheitslebenszyklus durchgeführt werden. Der Sicherheitslebenszyklus setzt sich aus folgenden Phasen zusammen (vgl. ISO 26262-2, Klausel 5.2.1f):

- Definition der Komponenten (vgl. ISO 26262-3, Klausel 5)
- Initialisierung des Sicherheitslebenszyklus (vgl. ISO 26262-3, Klausel 6)
- Gefahrenanalyse und Risikoeinschätzung (vgl. ISO 26262-3, Klausel 7)
- Produktentwicklung auf der Systemebene (vgl. ISO 26262-4)
- Produktentwicklung auf der Hardwareebene (vgl. ISO 26262-5)
- Produktentwicklung auf der Softwareebene (vgl. ISO 26262-6)
- Validierung der Sicherheit (vgl. ISO 26262-4, Klausel 9)
- Einschätzung der funktionalen Sicherheit (vgl. ISO 26262-4, Klausel 10)
- Freigabe für die Produktion (vgl. ISO 26262-4, Klausel 11)
- Produktion (vgl. ISO 26262-7, Klausel 5)
- Einsatz, Wartung und Außerbetriebnahme (vgl. ISO 26262-7, Klausel 6)

Für diese Arbeit ist vor allem die Phase Produktentwicklung auf Softwareebene von Interesse. Deswegen wird ISO 26262-6 im nächsten Abschnitt genauer betrachtet.

2.1.2. Softwareentwicklung nach ISO 26262

Die Softwareentwicklung des Sicherheitsstandards wird in ISO 26262-6 behandelt. Für die Softwareentwicklung wird das V-Modell als grundlegende Struktur verwendet (vgl. ISO 26262-6, S. 5). Das V-Modell ist dabei in folgende Phasen eingeteilt, die jeweils einer der Klauseln des Standards entsprechen:

- Initialisierung der Produktentwicklung auf Softwareebene (vgl. ISO 26262-6, Klausel 5)
- Spezifizierung der Softwaresicherheitsanforderungen (vgl. ISO 26262-6, Klausel 6)
- Entwurf der Softwarearchitektur (vgl. ISO 26262-6, Klausel 7)
- Entwurf und Implementierung der Softwaremodule (vgl. ISO 26262-6, Klausel 8)
- Test der Softwaremodule (vgl. ISO 26262-6, Klausel 9)
- Integration und Test der Software (vgl. ISO 26262-6, Klausel 10)
- Verifikation der Softwaresicherheitsanforderungen(vgl. ISO 26262-6, Klausel 11)

In jeder Phase bzw. Klausel (ISO 26262-6, Klausel X) sind in den entsprechenden Unterklauseln das Ziel (ISO 26262-6, Klausel X.1), eine allgemeine Beschreibung (ISO 26262-6, Klausel X.2), die Eingaben (ISO 26262-6, Klausel X.3), die Anforderungen und Empfehlungen (ISO 26262-6, Klausel X.4) sowie die Arbeitsergebnisse (ISO 26262-6, Klausel X.5) beschrieben.

Bei den Eingaben handelt es sich um Dokumente oder Softwareartefakte, die für die jeweilige Phase vorausgesetzt werden. Häufig stammen sie aus den vorherigen Phasen des Entwicklungsprozesses. Außerdem werden zusätzliche Informationen in Form von Dokumenten, Richtlinien oder Softwareartefakten aufgelistet, die für die jeweilige Phase hilfreich sein können.

Die Anforderungen und Empfehlungen beinhalten neben den Aufgaben der Phase häufig auch Methoden und Techniken der Softwarequalitätssicherung, die in der jeweiligen Phase eingesetzt werden sollen. Diese Methoden und Techniken sind in der Form von Tabellen gegeben, die die Methoden auflisten und je nach ASIL Level der zu entwickelnden Software mehr oder weniger stark empfohlen werden (vgl. Tabelle 2.1). Die Einträge sind dabei als Alternativen zu verstehen, d. h. nicht alle Methoden müssen eingesetzt werden, sondern nur eine geeignete Auswahl davon.

Die Arbeitsergebnisse jeder Phase werden in Form von Dokumenten und Softwareartefakten aufgelistet. Dabei kann es sich entweder um neue Dokumente handeln oder um eine Weiterentwicklung von Dokumenten aus den vorherigen Phasen des Entwicklungsprozesses.

Tabelle 2.1.: Prinzipien für den Softwarearchitekturentwurf (ISO 26262-6, Klausel 7.4.3).

	Methods	ASIL			
		A	**B**	**C**	**D**
1a	Hierarchical structure of software components	++	++	++	++
1b	Restricted size of software components[a]	++	++	++	++
1c	Restricted size of interfaces[a]	+	+	+	+
1d	High cohesion within each software component[b]	+	++	++	++
1e	Restricted coupling between software components[a, b, c]	+	++	++	++
1f	Appropriate scheduling properties	++	++	++	++
1g	Restricted use of interrupts[a, d]	+	+	+	+

[a] In methods 1b, 1c, 1e and 1g „restricted“ means to minimize in balance with other design considerations.
[b] Methods 1d and 1e can, for example, be achieved by separation of concerns which refers to the ability to identifiy, encapsulate, and manipulate those parts of software that are relevant to a particular concept, goal, task, or purpose.
[c] Method 1e addresses the limitation of the external coupling of software components.
[d] Any interrupts used have to be priority-based.

2.1.3. Qualifizierung von Softwarewerkzeugen nach ISO 26262

An Softwarewerkzeuge, die bei der Entwicklung kritischer Systeme zum Einsatz kommen, werden spezielle Anforderungen gestellt. Durch die Softwarewerkzeuge dürfen keine Fehler in das zu entwickelnde System eingeführt werden oder Fehler unentdeckt bleiben. Diese Anforderungen sind in ISO 26262 (2011) im Detail definiert. Der Standard beschreibt auch Methoden, um Softwarewerkzeuge für den Einsatz zur Entwicklung kritischer Systeme zu qualifizieren. Solche Softwarewerkzeuge heißen deshalb auch qualifizierbare Softwarewerkzeuge.

Softwarewerkzeuge werden in der Regel für einen bestimmten Use-Case in einem bestimmten Projekt qualifiziert. Die Qualifizierung erfolgt dadurch meistens im Auftrag des Unternehmens, das das Softwarewerkzeug einsetzen möchte, und nicht durch den Softwarewerkzeughersteller. Die Qualifizierung muss von einer dritten unabhängigen Stelle durchgeführt werden.

Der ISO 26262 (2011) Standard beschreibt die Methoden zur Qualifizierung eines Softwarewerkzeugs in ISO 26262-8, Klausel 11. Für die Qualifizierung muss das Tool Confidence Level (TCL 1-3) des Softwarewerkzeugs ermittelt werden. Dafür muss zuerst der Tool Impact (TI 1-2) bestimmt werden (vgl. ISO 26262-8, Klausel 11.4.5.2), d.h. die Möglichkeit, dass das Softwarewerkzeug durch ein Fehlverhalten einen Fehler in das zu entwickelnde System

Tabelle 2.2.: Basierend auf den ermittelten Werten für den Tool Impact (TI) und die Tool Error Detection (TD) wird das Tool Confidence Level (TCL) des zu qualifizierenden Softwarewerkzeugs bestimmt (vgl. ISO 26262-8, Klausel 11.4.5.5).

		Tool Error Detection		
		TD1	TD2	TD3
Tool Impact	TI1		TCL1	
	TI2	TCL1	TCL2	TCL3

Tabelle 2.3.: Für die Qualifizierung von Softwarewerkzeugen mit TCL3 sollen folgende Methoden verwendet werden. Die Wichtigkeit der Methoden ist dabei abhängig vom Automotive Safety Integrity Level (ASIL) der zu entwickelnden Software (vgl. ISO 26262-8, Klausel 11.4.6.1).

	Methods	**ASIL**			
		A	**B**	**C**	**D**
1a	Increased confidence from use in accordance with ISO 26262-8, Klausel 11.4.7	++	++	+	+
1b	Evaluation of the tool development process in accordance with ISO 26262-8, Klausel 11.4.8	++	++	+	+
1c	Validation of the software tool in accordance with ISO 26262-8, Klausel 11.4.9	+	+	++	++
1d	Development in accordance with a safety standard[a]	+	+	++	++

[a] No safety standard is fully applicable to the development of software tools. Instead, a relevant subset of requirments of the safety standard can be selected.

einführt oder dabei versagt einen Fehler aufzudecken. Anschließend muss die Tool Error Detection (TD 1-3) bestimmt werden (vgl. ISO 26262-8, Klausel 11.4.5.2). Die TD beschreibt den Grad an Vertrauen in die Maßnahmen, die getroffen wurden, um entweder zu verhindern, dass das Softwarewerkzeug ein Fehlverhalten zeigt oder um ein Fehlverhalten und die damit zusammenhängenden fehlerhaften Ausgaben nachträglich zu erkennen. Aus der Kombination aus TI und TD ergibt sich das TCL (vgl. Tabelle 2.2).

In Kombination mit dem ASIL der zu entwickelnden Software ergeben sich die Tool Qualification Methods, die für die Qualifizierung eines Softwarewerkzeugs angewandt werden sollen (vgl. Tabelle 2.3 und Tabelle 2.4). Für ein Softwarewerkzeug mit TCL1 muss keine Qualifizierung durchgeführt werden (vgl. ISO 26262-8, Klausel 11.4.6.1).

Tabelle 2.4.: Für die Qualifizierung von Softwarewerkzeugen mit TCL2 sollen folgende Methoden verwendet werden. Die Wichtigkeit der Methoden ist dabei abhängig vom Automotive Safety Integrity Level (ASIL) der zu entwickelnden Software (vgl. ISO 26262-8, Klausel 11.4.6.1).

	Methods	ASIL			
		A	**B**	**C**	**D**
1a	Increased confidence from use in accordance with ISO 26262-8, Klausel 11.4.7	++	++	++	+
1b	Evaluation of the tool development process in accordance with ISO 26262-8, Klausel 11.4.8	++	++	++	+
1c	Validation of the software tool in accordance with ISO 26262-8, Klausel 11.4.9	+	+	+	++
1d	Development in accordance with a safety standard[a]	+	+	+	++

[a] No safety standard is fully applicable to the development of software tools. Instead, a relevant subset of requirments of the safety standard can be selected.

Tool Qualification Methods

Die verschiedenen Methoden zur Qualifizierung von Softwarewerkzeugen müssen nicht alle angewendet werden, sondern es können eine oder mehrere davon ausgewählt werden. Das ASIL der zu entwickelnden Software gibt einen Hinweis darauf, welche Methoden für die Qualifizierung eines jeweiligen Softwarewerkzeugs sinnvoll sind (vgl. Tabelle 2.3 und Tabelle 2.4). Bei den Methoden handelt es sich im Einzelnen um:

- Erhöhtes Vertrauen in das Softwarewerkzeug aus den Erfahrungen früherer Nutzungen
- Evaluierung des Softwareentwicklungsprozesses
- Validierung des entwickelten Softwarewerkzeugs
- Übereinstimmung des Softwareentwicklungsprozesses mit einem Sicherheitsstandard (z. B. ISO 26262, IEC 61508 oder RTCA DO-178)

Erhöhtes Vertrauen in das Softwarewerkzeug aus den Erfahrungen früherer Nutzungen. Mit einem erhöhten Vertrauen in das Softwarewerkzeug aus den Erfahrungen früherer Nutzungen kann nur argumentiert werden, wenn die vorherige Nutzung des Softwarewerkzeugs für denselben Zweck, mit vergleichbaren Use-Cases, vergleichbarer Einsatzumgebung und ähnlichen funktionalen Einschränkungen erfolgte. Außerdem müssen die Daten über die Nutzung des Softwarewerkzeugs ausreichend und geeignet sein, um eine begründete Aussage treffen

zu können. Dazu gehört auch, dass das Auftreten von Fehlfunktionen und den zugehörigen fehlerhaften Ausgaben während der früheren Nutzung des Softwarewerkzeugs systematisch erfasst wurden. Des Weiteren muss die Spezifikation des Softwarewerkzeugs unverändert sein.

Die Erfahrungen, die während früherer Nutzung des Softwarewerkzeugs gemacht wurden, sollen anhand folgender Informationen analysiert und evaluiert werden:

- der eindeutigen Identifizierung und Versionsnummer des Softwarewerkzeugs,
- der Konfiguration des Softwarewerkzeugs,
- den Details über die Einsatzdauer und den relevanten Daten der Benutzung,
- der Dokumentation der Fehlfunktionen und den zugehörigen fehlerhaften Ausgaben, mit Details der Bedingungen die dazu geführt haben,
- der Liste der früheren kontrollierten Versionen und den Fehlern, die in jeder Version behoben wurden sowie
- der Schutzmaßnahmen, Vermeidungsstrategien oder Work-arounds für bekannte Fehlfunktionen oder Maßnahmen zur Erkennung der zugehörigen fehlerhaften Ausgaben, falls verfügbar.

Die Argumentation eines erhöhten Vertrauens aufgrund früheren Erfahrungen ist allerdings nur für die betrachtete Version des Softwarewerkzeugs gültig (vgl. ISO 26262-8, Klausel 11.4.7).

Evaluierung des Softwareentwicklungsprozesses. Der Softwareentwicklungsprozess, der für die Entwicklung des Softwarewerkzeugs angewendet wurde, soll mit einem geeigneten Softwareentwicklungsstandard übereinstimmen (z. B. Wasserfallmodell, V-Modell). Die Evaluierung des Softwareentwicklungsprozesses soll durch eine Begutachtung erfolgen, die auf einem geeigneten Standard basiert (z. B. Automotive SPICE, CMMI oder ISO 15504). Außerdem soll die korrekte Anwendung des Softwareentwicklungsprozesses demonstriert werden (vgl. ISO 26262-8, Klausel 11.4.8).

Validierung des entwickelten Softwarewerkzeugs. Die Validierung des Softwarewerkzeugs umfasst verschiedene Aspekte. Es soll gezeigt werden, dass das Softwarewerkzeug mit den spezifizierten Anforderungen übereinstimmt. Dies kann z. B. durch Tests, die funktionale Anforderungen und Qualitätsanforderungen des Softwarewerkzeugs prüfen, erreicht werden. Die Fehlfunktionen und die zugehörigen fehlerhaften Ausgaben des Softwarewerkzeugs, die während der Validierung auftreten, sollen, zusammen mit Informationen über ihre möglichen Konsequenzen und mit Maßnahmen, diese zu erkennen oder zu entdecken, analysiert werden. Außerdem soll die Reaktion des Softwarewerkzeugs auf anormale Einsatzbedingungen untersucht werden. Unter anormale Einsatzbedingungen fallen z. B. vorhersehbarer falscher

Gebrauch, unvollständige Eingaben, unvollständige Updates des Softwarewerkzeugs oder unerlaubte Kombinationen von Konfigurationseinstellungen (vgl. ISO 26262-8, Klausel 11.4.9).

Übereinstimmung des Softwareentwicklungsprozesses mit einem Sicherheitsstandard. Die Entwicklung des Softwarewerkzeugs soll in Übereinstimmung mit einem Sicherheitsstandard erfolgen (z. B. ISO 26262, IEC 61508 oder RTCA DO-178). Da kein Sicherheitsstandard für die Entwicklung von Softwarewerkzeugen vollständig zutreffend ist, kann eine relevante Untermenge der Anforderungen des Standards ausgewählt werden.

2.2. Softwareentwicklungsprozesse

Softwareentwicklungsprozesse dienen dazu, eine geregelte und kontrollierte Durchführung eines Softwareprojekts zu gewährleisten. Sie schaffen dadurch einen Rahmen für die Qualitätssicherung innerhalb eines Softwareprojekts. Ludewig u. Lichter (2007) sowie Hoffmann (2013) zählen deswegen alle Methoden und Verfahren, die eine geregelte und kontrollierte Durchführung eines Softwareprojekts zum Ziel haben, zu den Softwareentwicklungsprozessen. Allgemein beschreibt ein Prozess die Durchführung eines Projekts, indem das Projekt in einzelne Aktivitäten unterteilt wird. Aus Eingangsgrößen werden, unter der Zuhilfenahme von Ressourcen, schrittweise Ausgangsgrößen – das Ergebnis – abgeleitet (vgl. Abbildung 2.1).

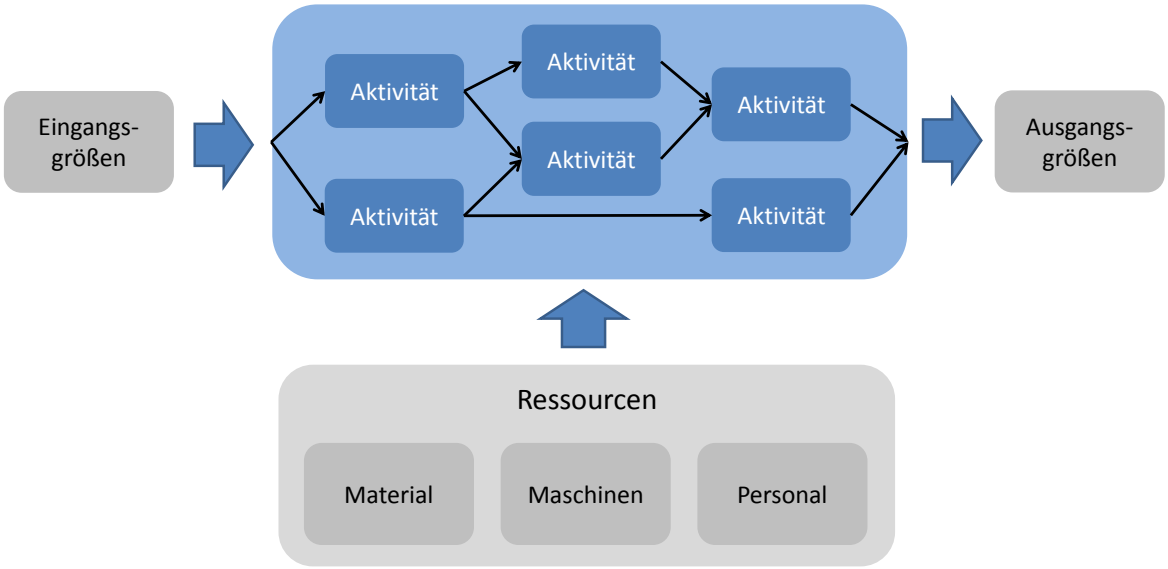

Abbildung 2.1.: Übersicht eines Prozess in Anlehnung an Hoffmann (2013). In einem Prozess werden aus Eingangsgrößen, unter Zuhilfenahme von Ressourcen, schrittweise Ausgangsgrößen abgeleitet.

Die dafür notwendigen Vorgehensweisen und Arbeitsabläufe werden in einem Prozessmodell beschrieben. Häufig werden die Begriffe Prozessmodell und Vorgehensmodell synonym verwendet. Ludewig u. Lichter (2007) jedoch unterscheiden die beiden Begriffe, da ein Prozess mehr beinhaltet als das reine Vorgehen. Prozessmodelle enthalten neben einem Vorgehensmodell, ein Qualitätsmodell, eine Organisationsstruktur sowie Vorgaben für das Projektmanagement, die Dokumentation und die Konfigurationsverwaltung. Diese Arbeit folgt der Unterscheidung von Ludewig u. Lichter (2007), da dadurch eine saubere Abgrenzung des Themas der Arbeit möglich ist.

Außer den Prozessmodellen gibt es auch noch Reifegradmodelle, die nicht die Durchführung eines Projekts, sondern die Prozessverbesserung zum Ziel haben. Abbildung 2.2 zeigt die Zusammenhänge der Softwareentwicklungsprozesse. Die Prozessmodelle, mit den untergeordneten Vorgehens- und Qualitätsmodellen, unterliegen einer ständigen Prozessverbesserung. Die Prozessverbesserung bedient sich dabei den Reifegradmodellen.

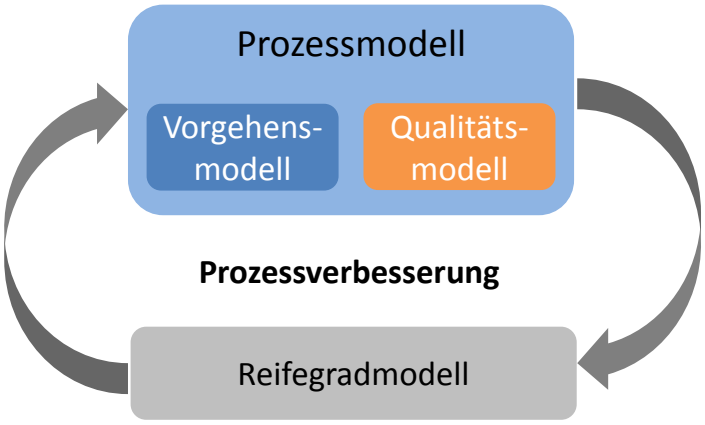

Abbildung 2.2.: Übersicht der Softwareentwicklungsprozesse inklusive der Prozessverbesserung durch Reifegradmodelle.

Zunächst werden Prozessmodelle im Allgemeinen erläutert, ehe mit dem Phasenmodell ein grundlegendes Prozessmodell beschrieben wird. In diesem Zusammenhang werden auch Meilensteine und deren Weiterentwicklung, die Quality Gates, beschrieben. Dann werden die Vorgehensmodelle allgemein betrachtet und mit dem Wasserfallmodell eines ausgeführt, das Aktivitäten und Dokumente in den Vordergrund stellt. Mit dem V-Modell wird dann eine Erweiterung des Wasserfallmodells erläutert. Anschließend werden die Qualitätsmodelle vorgestellt. Zum Abschluss wird noch auf die Reifegradmodelle und die Prozessverbesserung eingegangen. Die Abschnitte beziehen sich dabei auf die Bücher von Ludewig u. Lichter (2007) und Hoffmann (2013).

2.2.1. Prozessmodelle

Prozessmodelle regeln den grundlegenden Ablauf eines Softwareprojekts. Typische Bestandteile eines Prozessmodells sind die Projektplanung auf Grundlage der Schätzung des Aufwands und der Zeit, die Kontrolle des Projektfortschritts, Fragen des Personalmanagements sowie die Erstellung von Abnahmeplänen. Außerdem beinhaltet ein Prozessmodell ein Vorgehensmodell, das die einzelnen Entwicklungsschritte definiert, sowie ein Qualitätsmodell, in dem die Anforderungen an das Qualitätsmanagement definiert sind (vgl. Abbildung 2.2).

Nicht immer sind solche Abläufe explizit dokumentiert, aber auch bei einer informellen kontinuierlichen Wiederholung von Arbeitsabläufen handelt es sich um ein – wenn auch implizites – Prozessmodell. Auch bei einem expliziten Prozessmodell sind nicht alle Details vorgegeben, sondern sie müssen für jedes Projekt als konkreter Prozess ausgestaltet werden. Dieser Vorgang wird als Prozessausprägung (engl. *Tailoring*) bezeichnet. Die verschiedenen Prozessmodelle lassen dabei unterschiedlich viel Raum zur Ausprägung.

Hoffmann (2013) nennt das V-Modell XT, den Rational Unified Process und das Extreme Programming als die heute wichtigsten Prozessmodelle. Weitere gängige Prozessmodelle sind Cleanroom Development, Scrum und Crystal. Ludewig u. Lichter (2007) beschreiben als grundlegendes Prozessmodell das Phasenmodell. In diesem wird ein Projekt in Phasen unterteilt, die durch Meilensteine getrennt sind. Dadurch wird ein Rahmen geschaffen, welcher eine wichtige Voraussetzung dafür ist, dass ein Projekt erfolgreich geplant und durchgeführt werden kann. Ludewig u. Lichter (2007) geben folgende Definition für das Phasenmodell:

> **Phasenmodell** – Die Softwareentwicklung wird vor Beginn in Phasen gegliedert, die streng sequentiell durchlaufen werden. Für jede Phase gibt es ein Budget; dieses Budget wird erst freigegeben, wenn der vorangehende Meilenstein erreicht ist. Dann kann die nächste Phase beginnen.
>
> Ludewig u. Lichter (2007)

Zu Beginn eines Projekts werden Zwischenziele – die Meilensteine – definiert. Anschließend wird für jeden Abschnitt zwischen zwei Meilensteinen – die Phasen – der Aufwand und die benötigten Ressourcen – das Budget – abgeschätzt. Während der Durchführung des Projekts wird überprüft, ob ein Meilenstein planmäßig erreicht wurde oder ob nachgesteuert werden muss. Bei erfolgreichem Erreichen eines Meilensteins kann in die nächste Phase gewechselt werden. Wallin u. a. (2002) definieren den Begriff Meilenstein wie folgt:

> **milestone** – A milestone is defined as a scheduled event that marks the completion of one or more important tasks, and it is used to measure achievements and development progress. At a milestone, a predefined set of deliverables should have reached a predefined state to enable a review.
>
> Wallin u. a. (2002)

Dabei ist entscheidend, dass ein Meilenstein nicht durch den Termin, sondern durch das zu erreichende Zwischenziel definiert ist. Deshalb müssen die Kriterien für das Erreichen eines Meilensteins klar definiert und überprüfbar sein. Die Prüfungen sind dabei keine Folge, sondern eine zwingende Voraussetzung für das Erreichen eines Meilensteins. Die Ergebnisse einer Phase müssen anhand der definierten Kriterien überprüft werden, so dass am Ende einer Phase eine objektive Entscheidung über das Erreichen des Meilensteins getroffen werden kann. Die Prüfung der Ergebnisse muss dabei nicht am Ende der Phase durchgeführt werden, sondern kann bereits direkt nach Fertigstellung erfolgen. Wichtig ist nur, dass spätestens am Ende einer Phase alle Ergebnisse geprüft wurden.

Sondermann (2007), Prefi (2007) und Schneider (2012) beschreiben mit den Quality Gates eine Weiterentwicklung und Verfeinerung der Meilensteine, die vor allem die Qualitätsaspekte der Softwareentwicklung in den Vordergrund stellen. Quality Gates verfügen über im Voraus definierte Qualitätskriterien, die erfüllt sein müssen, bevor in die nächste Phase eines Prozesses gewechselt werden kann. Dadurch rückt statt dem Termin, wie häufig bei den Meilensteinen, die gemessene Qualität in der Vordergrund. Die Messung und Prüfung der Qualität kann z. B. durch Metriken oder Checklisten erfolgen.

2.2.2. Vorgehensmodelle

Vorgehensmodelle beschreiben das Vorgehen bei der Softwareentwicklung, also der Umsetzung von Benutzerbedürfnissen in ein Softwareprodukt. Dies wird auch bei folgender Definition von Vorgehensmodellen deutlich, die im IEEE Standard 610.12 (1990) zu finden ist:

> **software development process** – The process by which user needs are translated into a software product. The process involves translating user needs into software requirements, transforming the software requirements into design, implementing the design in code, testing the code, and sometimes, installing and checking out the software for operational use.
>
> Note: These activities may overlap or be performed iteratively.
>
> IEEE Standard 610.12 (1990)

Die Umsetzung von Benutzerbedürfnissen in ein Softwareprodukt findet dabei in mehreren Schritten statt. Zuerst werden aus den Benutzerbedürfnissen Softwareanforderungen abgeleitet. Die so erhobenen Softwareanforderungen werden als nächstes in einem Softwareentwurf umgesetzt und anschließend implementiert. Nach dem Test der Implementierung folgt die Installation und die Abnahme der fertigen Software.

Das Wasserfallmodell, als grundlegendes Vorgehensmodell, stellt dabei genau diese Entwicklungsschritte – auch Aktivitäten genannt – und die dabei entstehenden Dokumente in den Vordergrund. Dies wird auch in der Definition von Ludewig u. Lichter (2007) deutlich:

> **Wasserfall- oder Dokumentenmodell** – Die Software-Entwicklung wird als Folge von Aktivitäten betrachtet, die durch Teilergebnisse (Dokumente) gekoppelt sind. Diese Aktivitäten können auch gleichzeitig oder iterativ ausgeführt werden. Davon abgesehen ist die Reihenfolge der Aktivitäten fest definiert, nämlich (sinngemäß) Analysieren, Spezifizieren, Entwerfen, Codieren, Testen, Installieren und Warten.
>
> Ludewig u. Lichter (2007)

Bei den Aktivitäten handelt es sich um die Analyse, die Spezifikation, den Softwareentwurf, die Implementierung, der Softwaretest, die Installation und Abnahme sowie die Inbetriebnahme und Wartung (vgl. Abbildung 2.3), die jeweils Dokumente bzw. Softwareartefakte zum Ergebnis haben. Es entstehen z. B. bei der Implementierung der Programmcode und beim Softwaretest die Testprotokolle. Deswegen wird das Wasserfallmodell auch als Dokumentenmodell bezeichnet.

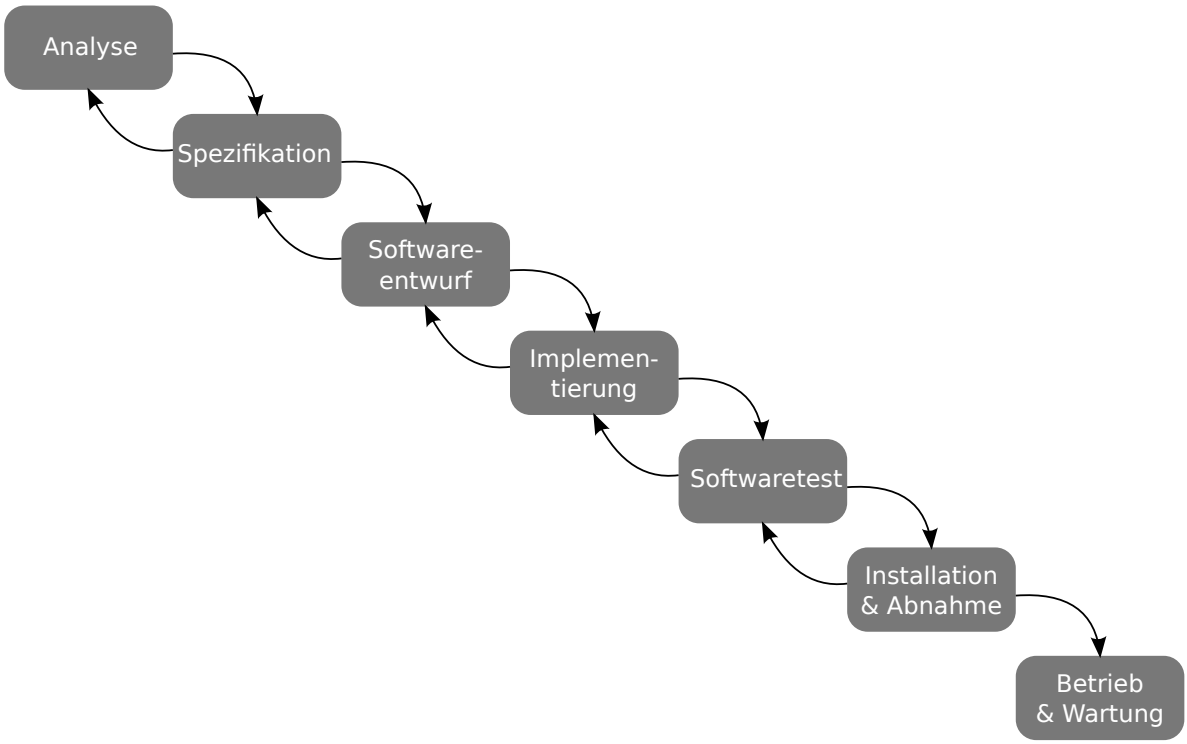

Abbildung 2.3.: Übersicht des Wasserfallmodells mit den Aktivitäten Analyse, Spezifikation, Softwareentwurf, Implementierung, Softwaretest, Installation und Abnahme sowie Betrieb und Wartung.

Das allgemeine V-Modell ist eine Erweiterung des Wasserfallmodells. Die Aktivität Softwaretest wird im V-Modell aufgeteilt, so dass jeder Entwicklungsaktivität eine Testaktivität zugeordnet werden kann. So wird z. B. die Spezifikation durch den Systemtest, die Softwarearchitektur durch den Integrationstest und der Modulentwurf durch Modultests abgedeckt. Des Weiteren

werden die Aktivitäten V-förmig angeordnet, so dass die Entwicklungsaktivitäten sich im linken Ast des V befinden und die Testaktivitäten im rechten Ast (vgl. Abbildung 2.4).

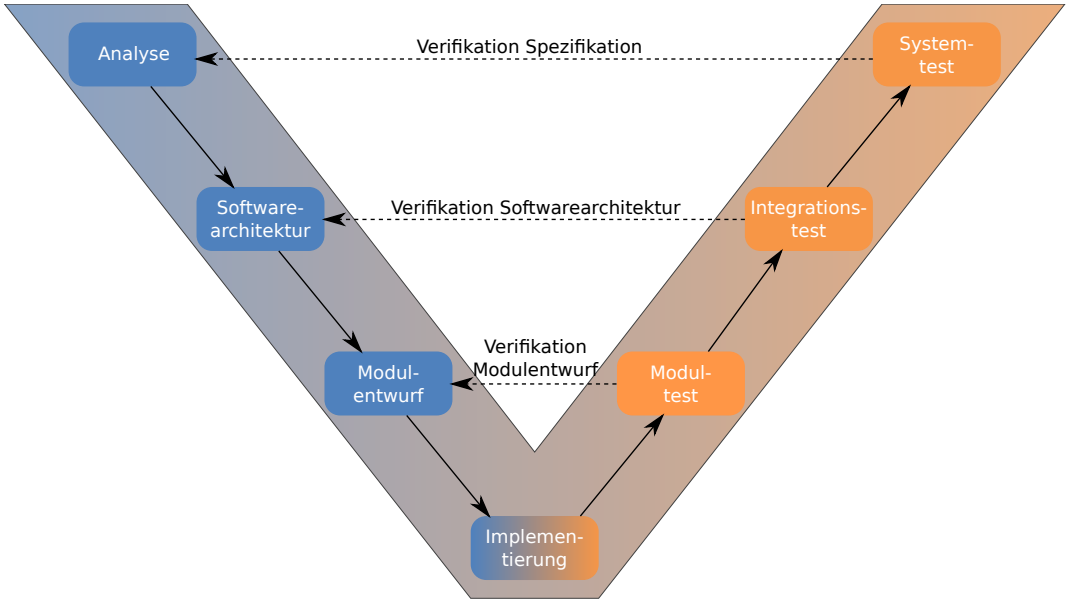

Abbildung 2.4.: Übersicht des V-Modells mit den Entwicklungsaktivitäten Analyse, Softwarearchitektur, Modulentwurf und Implementierung sowie den Testaktivitäten Modultest, Integrationstest und Systemtest.

Das allgemeine V-Modell bildet die Grundlage für den gleichnamigen Entwicklungsstandard, der über das Vorgehensmodell hinaus noch die Aspekte eines Prozessmodells abdeckt (vgl. Abschnitt 2.2.1). Die aktuelle Version des Entwicklungsstandards wird als V-Modell XT bezeichnet und bietet zahlreiche Möglichkeiten für die Prozessausprägung.

2.2.3. Reifegradmodelle & Prozessverbesserung

Reifegradmodelle werden für die Prozessanalyse und Prozessverbesserung eingesetzt. Das Ziel ist, bewährte Praktiken herauszustellen, Stärken und Schwächen zu identifizieren sowie Verbesserungsmaßnahmen zu definieren, um Strukturen und Prozesse einer Organisation zu optimieren. Die wichtigsten Reifegradmodelle sind CMMI (CMMI Product Team, 2010) und SPICE (ISO/IEC 15504-5, 2012). Mit Automotive SPICE (VDA QMC Working Group 13 / Automotive SIG, 2015) gibt es auch eine domänenspezifische Variante für das Automobilumfeld. Sowohl CMMI als auch SPICE sind mit Zertifizierungen verbunden, bei denen verschiedene Reifegrade bzw. Stufen erreicht werden können. CMMI definiert die fünf Reifegrade Initial, Managed, Defined, Quantitatively Managed und Optimizing, während SPICE die sechs ähnlichen Reifegrade Incomplete, Performed, Managed, Established, Predictable und Optimizing definiert.

2.3. Softwarewerkzeuge

Softwarewerkzeuge spielen in der Softwareentwicklung eine wichtige Rolle. Sie kommen in allen Bereichen und Phasen eines Softwareentwicklungsprozesses zum Einsatz. Dies wird auch im IEEE Standard 610.12 (1990) deutlich, in dem Softwarewerkzeuge als Programme, die zur Entwicklung, Analyse, Prüfung oder Wartung von Software verwendet werden, definiert sind:

> **software tool** – A computer program used in the development, testing, analysis, or maintenance of a program or its documentation. Examples include comparator, cross-reference generator, decompiler, driver, editor, flowcharter, monitor, test case generator, timing analyzer.
>
> IEEE Standard 610.12 (1990)

Ludewig u. Lichter (2007) haben eine andere, ergänzende Sicht auf Softwarewerkzeuge:

> **Werkzeug** – Ein Werkzeug dient zur Ausführung einer Arbeit, die – wenigstens prinzipiell – auch ohne das Werkzeug geleistet werden könnte. Im Produkt ist das Werkzeug nicht mehr enthalten, es sei denn als Ausrüstung für die Wartung.
>
> Ludewig u. Lichter (2007)

Demnach sind Softwarewerkzeuge nicht notwendig, aber nützlich um Software effizient zu entwickeln. Vor allem die Kombination aus geeigneten Entwicklungsprozessen und den dazu passenden Softwarewerkzeugen zur Unterstützung sorgt für eine entscheidende Steigerung der Effizienz (vgl. Abbildung 2.5). Wichtig ist außerdem, dass Softwarewerkzeuge, die bei der Entwicklung von Software zum Einsatz kommen, nicht Teil der entwickelten Software sind.

Wie bereits aus der Definition des IEEE Standard 610.12 (1990) deutlich wird, spielen Softwarewerkzeuge bei der Softwareentwicklung eine wichtige Rolle. In jeder Phase eines Softwareentwicklungsprozesses kommen Softwarewerkzeuge zum Einsatz. In der Analysephase wird vor allem ein Softwarewerkzeug für die Anforderungsverwaltung benötigt. Ein Anforderungsverwaltungssystem findet jedoch auch über die Analysephase hinaus Verwendung, da es auch für das Änderungsmanagement und der Verfolgung von Anforderungen essentiell ist. Im Entwurf werden vor allem Modellierungswerkzeuge eingesetzt, mit den z. B. UML- oder ER-Diagramme erstellt und verwaltet werden können. Während der Implementierung kommt eine IDE zum Einsatz. Außerdem werden Softwarewerkzeuge für die statische Codeanalyse eingesetzt. Für den Systemtest sind Softwarewerkzeuge zur Testfallverwaltung und Testüberdeckung sowie Testtreiber hilfreich. Außerdem kann der GUI-Test durch Softwarewerkzeuge unterstützt werden.

Neben den Entwicklungswerkzeugen wird auch eine Softwareinfrastruktur benötigt, auf die ein Projekt aufgesetzt werden kann. Laut Hoffmann (2013) lässt sich ohne eine geeignete

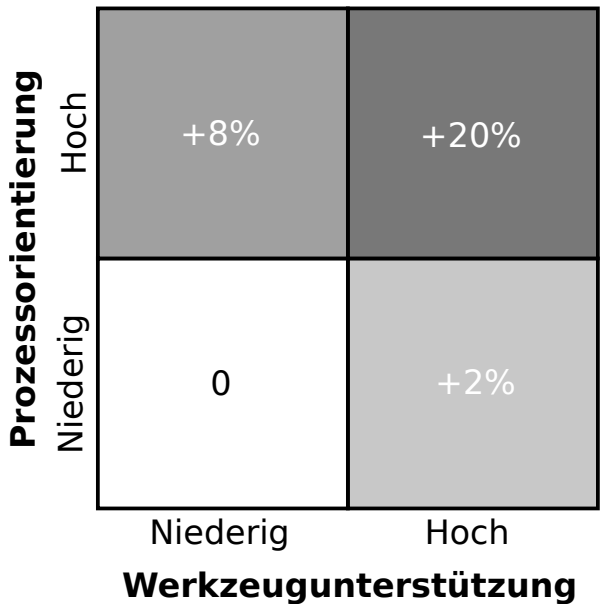

Abbildung 2.5.: Effinzienzverbesserung durch richtige Prozesse und Werkzeuge (vgl. Ebert (2014) Abb.9-1 S.278).

Softwareinfrastruktur ein Softwareentwicklungsprozess und eine wirkungsvolle Softwarequalitätssicherung nur schwer umsetzen. Unter Softwareinfrastruktur versteht man alle technischen Einrichtungen und Maßnahmen, die einen Softwareentwickler bei seiner täglichen Arbeit unterstützen. Hoffmann (2013) identifiziert als die vier Kernbereiche der Softwareinfrastruktur das Konfigurationsmanagement, die Build-Automatisierung, die Testautomatisierung und das Defektmanagement.

Konfigurationsmanagement. Beim Konfigurationsmanagement handelt es sich um einen der wichtigsten Kernbereiche der Softwareinfrastruktur. Es hat die technische und organisatorische Verwaltung aller während einer Softwareentwicklung entstehender Artefakte zur Aufgabe. Dafür wird vor allem ein Versionsverwaltungssystem benötigt.

Build-Automatisierung. Bei der Build-Automatisierung geht es um die automatische Übersetzung einzelner Programmkomponenten sowie der automatischen Integration dieser übersetzten Komponenten in ein komplexes Gesamtsystem. Dafür wird ein Build-Server benötigt. Durch die kontinuierliche Übersetzung und Integration der Softwarekomponenten soll sicher gestellt werden, dass jederzeit ein lauffähiges Softwaresystem vorliegt. Dadurch können Überraschungen bei der Auslieferung einer neuen Version vermieden werden.

Testautomatisierung. Die Testautomatisierung ist meistens an die Build-Automatisierung gekoppelt und führt nach einer erfolgreichen Übersetzung und Integration automatische Tests durch. Dabei kann es sich um einzelne Modultests für geänderte Softwaremodule oder sogar um aufwändige Regressionstests handeln. Durch die Testautomatisierung soll sicher gestellt werden, dass frühzeitig und kontinuierlich Tests durchgeführt und dadurch Fehler frühzeitig erkannt werden. Wichtig dafür ist, dass Modultests oder sogar ganze Test-Frameworks vor Beginn der Implementierung erstellt werden, da nur dadurch neu implementierte Funktionalitäten und Module direkt getestet werden können.

Defektmanagement. Beim Defektmanagement geht es um die zentrale Erfassung und Verwaltung von Softwaredefekten. Hierbei kommen Request- und Bug-Tracker zum Einsatz.

Über die Softwareentwicklung hinaus kommen auch im Projektmanagement Softwarewerkzeuge zur Anwendung. So gibt es Softwarewerkzeuge, die einen Projektleiter bei der Kostenschätzung, der Risikoanalyse sowie der Personal- und Ressourcenplanung unterstützen.

2.4. Softwarequalitätssicherung

Ein wichtiger Aspekt der Softwareentwicklung ist die Softwarequalitätssicherung, d. h. die Sicherstellung der geforderten Qualität der zu entwickelnden Software. Dies wird auch in der Definition des IEEE Standard 610.12 (1990) deutlich:

> **software quality assurance** – See: quality assurance.
>
> **quality assurance** – (1) A planned and systemative pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements.
> (2) A set of activities designed to evaluate the process by which products are developed or manufactured.
>
> IEEE Standard 610.12 (1990)

Bei der Softwarequalitätssicherung handelt es sich also um einen geplanten und systematischen Ansatz, um sicherzustellen, dass die entwickelte Software den Anforderungen entspricht. Außerdem befasst sich die Softwarequalitätssicherung auch mit der Qualität der eingesetzten Softwareentwicklungsprozesse. Dies wird auch als Prozessverbesserung bezeichnet (vgl. Abschnitt 2.2.3). Die diversen Methoden und Techniken der Softwarequalitätssicherung können laut Ludewig u. Lichter (2007) und Hoffmann (2013) in die drei Bereiche konstruktive, analytische und organisatorische Qualitätssicherung eingeteilt werden.

Die Methoden und Techniken der konstruktiven und der analytischen Softwarequalitätssicherung haben die direkte Verbesserung des Softwareprodukts zum Ziel und werden deswegen

auch unter der Produktqualitätssicherung zusammengefasst. Die Methoden und Techniken der organisatorischen Softwarequalitätssicherung, die in die Bereiche Softwareinfrastruktur und Softwareentwicklungsprozesse unterteilt ist, beschäftigen sich ausschließlich mit der geregelten Entwicklung eines Softwareprodukts in Form eines definierten Prozesses. Die organisatorische Softwarequalitätssicherung wird deswegen auch als Prozessqualitätssicherung bezeichnet. Die Softwareentwicklungsprozesse wurden bereits in Abschnitt 2.2 ausführlich besprochen. Auf die Softwareinfrastruktur wurde bereits im Rahmen der Softwarewerkzeuge in Abschnitt 2.3 eingegangen.

Hoffmann (2013) beschreibt den Umstand, dass es bei der Softwarequalitätssicherung eine Korrelation im Projektmanagement zwischen Qualität, Kosten und Zeit gibt, die die erfolgreiche Softwarequalitätssicherung erschwert. Wenn einer der Aspekte optimiert werden soll, müssen bei den beiden anderen Aspekten Abstriche gemacht werden. Eine hohe Qualität bedeutet höhere Kosten und/oder eine längere Entwicklungsdauer, während eine Kostenoptimierung zu Lasten der Qualität geht und eine Optimierung der Entwicklungszeit eine verminderte Qualität und evtl. höhere Kosten bedeutet.

Im Folgenden werden zuerst die Methoden und Techniken der konstruktiven Softwarequalitätssicherung vorgestellt. Anschließend folgt eine Beschreibung der Methoden und Techniken der analytischen Softwarequalitätssicherung. Diese Abschnitte beziehen sich dabei auf die Bücher von Hoffmann (2013), Ludewig u. Lichter (2007), Liggesmeyer (2009) und Sommerville (2012). Zum Schluss werden mit Requirements Engineering (Ebert, 2014; Pohl, 2008) und Continuous Integration (Fowler, 2006) wichtige Konzepte der Softwarequalitätssicherung im Detail besprochen.

2.4.1. Konstruktive Softwarequalitätssicherung

Die konstruktive Qualitätssicherung hat zum Ziel, die geforderten Qualitätskriterien während der Entwicklung sicherzustellen. Dadurch sollen Fehler frühzeitig erkannt oder gar vermieden werden. Man spricht deswegen auch von dem Ansatz der Fehlervermeidung. In Anlehnung an Hoffmann (2013) fallen darunter Softwarerichtlinien, Entwurfsmuster, Typisierung, vertragsbasierte Programmierung, fehlertolerante Programmierung und Dokumentation.

Softwarerichtlinien

Unter Softwarerichtlinien versteht man Konventionen, die den Gebrauch einer bestimmten Programmiersprache syntaktisch und semantisch regeln. Syntaktische Softwarerichtlinien sollen die Verwendung einer bestimmten Programmiersprache vereinheitlichen, d. h. den Effizienzverlust bei großen Softwareprojekten, der durch individuelle Programmierstile entsteht, reduzieren. Semantische Softwarerichtlinien sollen die Anzahl der Softwaredefekte durch die Verwendung einheitlicher Lösungsmuster verringern.

Entwurfsmuster

Gamma u. a. (1995) beschreiben Entwurfsmuster als „einfache und elegante Lösungen für spezifische Probleme beim objektorientierten Softwareentwurf“. Für die Anwendung eines Entwurfsmusters ist es wichtig, dass man versteht welches Problem es löst und wie man diese generische Lösung auf den konkreten Fall überträgt. Deswegen beinhalten Entwurfsmuster typischerweise, neben einem Namen, eine Beschreibung des Problems, eine Lösung für dieses Problem und die Konsequenzen, die sich aus der Verwendung des Entwurfsmuster ergeben. Für die Umsetzung eines Entwurfsmusters werden normalerweise keine ausgefallene Sprachkonstrukte oder außergewöhnliche Programmierkenntnisse benötigt, sondern sie können in gewöhnlichen objektorientierten Programmiersprachen implementiert werden. Gamma u. a. (1995) teilen Entwurfsmuster in die drei Gruppen Erzeugungs-, Struktur- und Verhaltensmuster ein und bieten eine ausführliche Übersicht der gängigsten Entwurfsmuster.

Typisierung

Die Typisierung gehört heutzutage zu den mächtigsten Hilfsmitteln der konstruktiven Qualitätssicherung und wird dementsprechend von vielen modernen Programmiersprachen umgesetzt. Die Typsysteme der heute gängigen Programmiersprachen lassen sich in statische und dynamische Typisierung einteilen. Bei der statischen Typisierung werden Variablen zusammen mit ihrem Datentyp deklariert, d. h. der Compiler kennt zu jedem Zeitpunkt den Datentyp einer Variable. Dadurch können die meisten Typinkonsistenzen durch einen einfachen Vergleich erkannt werden. Bei dynamisch typisierten Programmiersprachen sind die verwendeten Variablen nicht an Objekte eines bestimmten Datentyps gebunden. Die dadurch erhöhte Flexibilität und evtl. Steigerung der Produktivität werden mit der schlechteren Erkennung von Inkonsistenzen zur Übersetzungszeit bezahlt. Des Weiteren können die Programmiersprachen nach schwacher und starker Typisierung unterschieden werden. Schwach typisierte Programmiersprachen erlauben, den Datentyp einer Variable nach Belieben unterschiedlich zu interpretieren. Stark typisierte Programmiersprachen stellen dagegen sicher, dass der Zugriff auf alle Variablen stets typkonform erfolgt.

Vertragsbasierte Programmierung

Bei der vertragsbasierten Programmierung (engl. *design by contract*) wird die Implementierung um einen Vertrag erweitert. Dazu werden Funktionen und Prozeduren um Vor- und Nachbedingungen, Invarianten sowie Zusicherungen ergänzt.

Fehlertolerante Programmierung

Das Ziel der fehlertoleranten Programmierung ist die Verbesserung der Fehlertoleranz eines Softwaresystems im Fall des Auftretens eines Fehlers. Eine verbreitete Technik zur Verbesserung der Fehlertoleranz ist die Erzeugung von funktionaler, informationeller, temporaler oder struktureller Redundanz. Das Erzeugen von struktureller Redundanz bei Systemkomponenten kann in homogene und heterogene Redundanz unterschieden werden. Bei der homogenen Redundanz werden n Komponenten gleicher Bauart parallel Betrieben. Da Software im Gegensatz zur Hardware keiner Alterung unterliegt, spielt die homogene Redundanz bei Softwaresystemen keine Rolle. Bei der heterogenen Redundanz werden deswegen n Komponenten unterschiedlicher Bauart parallel betrieben. Dadurch wird die Wahrscheinlichkeit von gemeinsam vorhandenen Fehlern deutlich reduziert. Des Weiteren können die Techniken für strukturelle Redundanz in statische und dynamische Redundanz unterschieden werden. Bei der statischen Redundanz werden die redundanten Systemkomponenten parallel betrieben und die Ergebnisse verglichen. Dies kann einen erheblichen Einfluss auf die Laufzeit ausüben. Bei der dynamischen Redundanz werden die redundanten Systemkomponenten als Ersatzkomponenten vorgehalten und nur bei Bedarf aktiviert. Die Schonung der Ressourcen wird durch eine deutlich kompliziertere automatische Fehlererkennung erkauft. Eine weitere Technik zur Erhöhung der Fehlertoleranz ist die Ausnahmebehandlung, die einen geregelten Umgang mit spontan auftretenden Fehlersituationen ermöglicht. Dabei kann zwischen geplanten und ungeplanten Ausnahmen unterschieden werden. Die Ausnahmebehandlung kann entweder an das Betriebssystem delegiert oder vom Programmierer umgesetzt werden. Im letzten Fall werden die Ausnahmen bei Bedarf abgefangen und können flexibel behandelt werden.

Dokumentation

Ludewig u. Lichter (2007) beschreiben Dokumentation als Daueraufgabe in der Softwareentwicklung, die „wie Denken und Diskutieren zu allen konstruktiven Tätigkeiten gehört.“ Das Ergebnis der Softwareentwicklung ist die Dokumentation. Deswegen führt eine nachträglich durchgeführte Dokumentation einer entwickelten Software auch zwangsläufig zu schlechter Softwarequalität. Bei der Dokumentation kann zwischen der integrierten und der separaten Dokumentation unterschieden werden. Unter integrierter Dokumentation versteht man dabei Kommentare im Programmcode oder die Bezeichnung von Variablen. Bei der separaten Dokumentation handelt es sich folglich um die Teile einer Software, die nicht im Programmcode enthalten sind. Diese Teile werden auch als Dokumente bezeichnet. Beispiele für Dokumente sind das Begriffslexikon, die Anforderungsspezifikation oder die Softwarearchitektur.

Für eine erfolgreiche (separate) Dokumentation ist eine Prüfung aller Dokumente nach der Fertigstellung notwendig. Ohne eine erfolgreiche Prüfung dürfen die Dokumente nicht freigegeben werden. Dafür ist es wichtig, dass die Anforderungen an die Dokumente klar sind und dass die Verantwortung für die Erstellung, Prüfung und Freigabe der Dokumente klar

geregelt sind. Außerdem sind Vorlagen, eine Werkzeugunterstützung und eine konsequente Versionierung mit der Konfigurationsverwaltung hilfreich.

Allerdings ist die Dokumentation kein Selbstzweck, sondern muss Ziele und einen Nutzen haben. Dokumente erlauben den Wissenstransfer innerhalb eines Entwicklerteams. Des Weiteren bewahren sie das Wissen über die entwickelte Software und erleichtern so deren Weiterentwicklung und Wartung. Ebenfalls ermöglichen Dokumente systematische Prüfungen im Rahmen der Softwareentwicklung. Außerdem machen Dokumente den Projektfortschritt messbar und erlauben dadurch eine bessere Projektkontrolle. Letztlich kann durch Dokumente der Ablauf der Entwicklung nachvollzogen werden, also ob systematisch und sorgfältig entwickelt wurde. Dies ermöglicht z. B. den Nachweis, dass Standards eingehalten wurden.

In Anlehnung an Frühauf u. a. (2002) und Ludewig u. Lichter (2007) können Dokumente den vier Kategorien Entwicklungs-, Qualitätssicherungs-, Projekt- und Prozessdokumente zugeordnet werden. Durch die Einteilung können auch Fragen nach Zweck, Leserkreis, Notwendigkeit der Nachführung / Aktualisierung und Aufbewahrungsdauer eines Dokuments direkt beantwortet werden.

Entwicklungsdokumente. Zu den Entwicklungsdokumenten zählen allen Dokumente, die für die Entwicklung und Wartung der Software benötigt werden. Darunter fallen z. B. die Anforderungsspezifikation, die Softwarearchitektur oder die Systemtestfälle.

Qualitätssicherungsdokumente. Zu den Qualitätssicherungsdokumenten gehören alle Dokumente, in denen die durchgeführten analytischen Qualitätssicherungsmaßnahmen dokumentiert sind. Darunter fallen z. B. die Test- und Review-Berichte.

Projektdokumente. Zu den Projektdokumenten gehören alle Dokumente, die für Planung, Leitung und Abschluss des Entwicklungsprojekts notwendig sind. Darunter fallen z. B. der Projektplan, der Projektstatusbericht und der Projektabschlussbericht.

Prozessdokumente. Zu den Prozessdokumenten gehören alle Dokumente, die den Entwicklungsprozess und seine konkrete Umsetzung im Projekt beschreiben. Darunter fallen z. B. Richtlinien, Standards und Musterdokumente.

2.4.2. Analytische Softwarequalitätssicherung

Der Ansatz der analytischen Qualitätssicherung ist es, die Qualitätseigenschaften eines Softwaresystems nach der Implementierung zu analysieren und zu bewerten. Dabei können das gesamte Softwaresystem oder einzelne Teile der Software (z. B. Dokumente oder Softwarekomponenten) einer Prüfung unterzogen werden. Der zu prüfende Teil der Software wird auch Prüfling genannt. Man spricht auch von dem Ansatz der Fehlerentdeckung. Hoffmann (2013) nennt und beschreibt den Softwaretest, die statische Codeanalyse und die formale Softwareverifikation als Teilgebiete der analytischen Qualitätssicherung. Die formale Softwareverifikation ist allerdings für diese Arbeit nicht von Interesse.

Softwaretest

Der Softwaretest ist die am weitesten verbreitete Technik, um Softwaredefekte zu identifizieren. Beim Softwaretest handelt es sich um ein dynamisches Verfahren, d. h. der Quellcode muss für die Prüfung ausgeführt werden. Neben der Testausführung kommt der Testplanung eine zentrale Rolle zu. Dem Softwaretest sind allerdings auch Grenzen gesetzt. So kann nur auf die Funktionen getestet werden, die überhaupt spezifiziert wurden. Unklare oder fehlende Anforderungen können durch den Softwaretest nicht aufgedeckt werden. Häufige Änderungen der Anforderungen haben ebenfalls negative Einflüsse auf den Testprozess, da dieser jedes Mal an die neuen Anforderungen angepasst werden muss. Auch die Programmkomplexität hat eine Auswirkung auf den Softwaretest. Durch die große Anzahl an Eingabekombinationen ist ein erschöpfender Softwaretest unmöglich. Entscheidend für die Qualität der auszuliefernden Software ist also die Auswahl der richtigen Testfälle. Hierbei macht sich die mangelnde Werkzeugunterstützung für die Konstruktion von Softwaretests bemerkbar. Die meisten Werkzeuge unterstützen den Softwareentwickler nur bei der Durchführung von Softwaretests. Des Weiteren werden Softwaretests häufig als weiche Entwicklungstätigkeit angesehen, was diese Aufgabe nicht leichter. Häufig kommt es deswegen zu Zeitmangel bei der Durchführung von Softwaretests oder sie werden sogar ganz gestrichen.

Laut Hoffmann (2013) können die Methoden des Softwaretests anhand der Prüfebene, des Prüfkriteriums, der Prüftechnik und der Testmetriken unterschieden werden.

Prüfebene. Als erstes ist die Ebene, auf der geprüft wird, entscheidend. Wird der Softwaretest auf Modulebene durchgeführt, d. h. werden einzelne Module auf ihre Korrektheit getestet, spricht man von Modul- bzw Unit-Tests. Werden die Schnittstellen von Modulen sowie das Zusammenspiel und die Kommunikation mehrerer Module getestet, spricht man von Integrationstests. Auf der höchsten Ebene, der Systemebene, wird die Funktionalität des Gesamtsystems getestet. Beim Systemtest wird die entwickelte Software gegen die spezifizierten Anforderungen geprüft. Eine vergleichbare Rolle nimmt der Abnahmetest ein, nur dass hier der Kunde das System testet.

Prüfkriterium. Ein weiterer Aspekt ist das Prüfkriterium, d. h. ob es sich um einen funktionalen, operationalen oder temporalen Softwaretest handelt. Beim funktionalen Softwaretest wird überprüft, ob aus den Eingangsgrößen die Ausgangsgrößen korrekt berechnet werden. Da die meisten, der in der Praxis durchgeführten Tests, unter den funktionalen Softwaretest fallen, wird dieser häufig mit dem Softwaretest gleichgesetzt. Unter den operationalen Softwaretest fallen der Installationstest, der Ergonomietest und der Sicherheitstest. Der Installationstest soll die Inbetriebnahme einer Software sicherstellen. Bei den Ergonomietests wird die Benutzerschnittstelle und die Benutzerdokumentation überprüft. Die Sicherheitstests schließlich sollen sowohl die funktionale Sicherheit (engl. *safety*) als auch die Informationssicherheit (engl. *security*) sicherstellen. Bei den temporalen Softwaretests handelt es sich um Komplexitäts-, Laufzeit-, Last- und Stresstests.

Prüftechniken. Außerdem sind die eingesetzten Prüftechniken wichtig. Diese werden in Black-Box- und White-Box-Tests unterschieden. Die Black-Box-Techniken haben gemeinsam, dass der geprüfte Programmcode nicht angeschaut und berücksichtigt wird. Er befindet sich sozusagen in einer undurchsichtigen Box. Bei den White-Box-Techniken dagegen wird der Programmcode angeschaut und berücksichtigt, er befindet sich also in einer transparenten Box. Zu den Black-Box-Techniken zählen z. B. der Äquivalenzklassentest und die Grenzwertbetrachtung. Ein bekannter Vertreter der White-Box-Techniken sind z. B. die Unit-Tests.

Testmetriken. Als letztes können noch die eingesetzten Testmetriken betrachtet werden. Durch die Testmetriken können verschiedene Aspekte des Softwaretests quantitativ erfasst werden. Mit Hilfe der Überdeckungsmetriken, z. B. Anweisung-, Zweig-, Pfad- und Bedingungsüberdeckung, und des Mutationstests kann untersucht werden, ob ein Softwaremodul ausreichend getestet wurde, wie viele unentdeckte Fehler ein Programm voraussichtlich enthält und wie effektiv ein eingesetztes Testverfahren ist.

Statische Codeanalyse

Im Gegensatz zu den dynamischen Testtechniken, wie z. B. den Softwaretests, verzichten statische Analyseverfahren auf die reale Ausführung eines Programms. Die untersuchten syntaktischen und semantischen Programmeigenschaften werden direkt aus den Quelltexten abgeleitet. Für die Untersuchung des Prüflings mit Hilfe statischer Codeanalyseverfahren muss dieser nicht verändert werden. Durch die statische Codeanalyse können deswegen auch keine neuen Fehler in den Prüfling hineingebracht werden. Die statische Codeanalyse besteht aus den Teilgebieten Softwaremetriken, Konformitätsanalyse, Exploit-Analyse, Anomalienanalyse und manuelle Softwareprüfung.

Softwaremetriken. Mit Hilfe von Softwaremetriken können verschiedene Aspekte von Softwaresystemen systematisch und quantitativ erfasst werden. Außerdem können auch Bereiche des Softwaremanagements, z. B. Projektfortschritt und Mitarbeiterzufriedenheit, durch Softwaremetriken erfasst werden. Durch die quantitative Erfassung vorher unsichtbarer Aspekte, werden diese sichtbar sowie objektiv vergleichbar und dadurch beherrschbar. Diesen Zusammenhang beschreibt DeMarco (1986) in folgendem Zitat:

> You can't manage what you can't control, and you can't control what you don't measure. To be effective software engineers or software managers, we must be able to control software development practice. If we don't measure it, however, we will never have that control.
>
> DeMarco (1986)

Doch ist nicht jede Metrik, die erhoben werden kann, auch automatisch eine sinnvolle Metrik, die einen Mehrwert bringt. Softwaremetriken müssen deswegen gewissen Anforderungen genügen. Hoffmann (2013) fordert z. B., dass sie objektiv, robust, vergleichbar, verwertbar und ökonomisch sein müssen und dass sie korrelieren müssen. Bekannte Beispiele für Softwaremetriken sind Lines of Code (LOC), Non Commented Source Statements (NCSS), die Halstead-Metriken, die McCabe-Metrik sowie objektorientierte Metriken, wie z. B. Response for a Class (RFC), Weighted Methods for Class (WMC), Lack of Cohesion in Methods (LCOM), Number of Children (NOC) und Coupling between Objects (CBO). Kan (2003) beschreibt die Rolle von Softwaremetriken in der Qualitätssicherung im Allgemeinen, während Lanza u. Marinescu (2006) auf objektorientierte Softwaremetriken im Speziellen eingehen.

Konformitätsanalyse. Bei der Konformitätsanalyse wird der Quellcode auf die Einhaltung vorgegebener Konformitätsregeln überprüft. Dabei kann sowohl eine Syntax- als auch eine Semantikanalyse durchgeführt werden.

Exploit-Analyse. Bei der Exploit-Analyse wird der Quellcode auf Schwachstellen überprüft, die von Angreifern missbraucht werden können. Ein Beispiel für solche Schwachstellen sind Buffer Overflows.

Anomalienanalyse. Mit Hilfe der Anomalienanalyse wird nach ungewöhnlichen oder auffälligen Anweisungssequenzen, die statistisch auf einen Fehler hindeuten, gesucht. Allerdings ist dabei wichtig, dass nicht jede Softwareanomalie auch automatisch ein Softwarefehler ist. Bei der Anomalienanalyse können sowohl Anomalien im Kontrollfluss als auch im Datenfluss untersucht werden.

Manuelle Softwareprüfung. Bei der manuellen Softwareprüfung wird der Prüfling entweder durch die Entwickler selbst oder durch Gutachter einer manuellen und formellen Prüfung unterzogen. Es gibt verschiedene Formen der manuellen Softwareprüfung, die auch als Review bezeichnet wird. So gibt es z. B. Codereviews, Technische Reviews, Inspektionen und Walkthroughs. Für diese Arbeit sind vor allem Inspektionen und Walk-throughs von Interesse.

Bei einer Inspektion prüfen Gutachter den Prüfling auf Konsistenz, Korrektheit und Vollständigkeit und auf die Einhaltung von Richtlinien und Normen. Während einer Inspektion dürfen nur Fehler oder Beanstandungen aufgezeigt, aber keine Lösungsvorschläge erarbeitet werden. Die Korrektur aller Befunde erfolgt durch die für den Prüfling zuständigen Entwickler. Unter Umständen muss der Prüfling nach der Korrektur einer erneuten Inspektion unterzogen werden. Die Ergebnisse einer Inspektion werden in einem Protokoll festgehalten. Darin sind sämtliche Befunde inklusive einer Gewichtung (z. B. kritischer Fehler, Hauptfehler, Nebenfehler) und die daraus resultierende abschließende Bewertung des Prüflings festgehalten, z. B. akzeptiert, akzeptiert mit Änderungen oder nicht akzeptiert. Außer den Gutachtern gibt es häufig noch die Rollen des Moderators und des Protokollanten. Der Moderator ist für die Organisation der Inspektion zuständig, inklusive Vor- und Nachbereitung. Der Protokollant verfasst das Protokoll. Optional kann ein Vertreter der Entwickler des Prüflings als Autor an der Inspektion teilnehmen. Dabei beschränkt sich seine Rolle jedoch auf das Beantworten von Fragen zum Prüfling.

Ein Walk-through ist im Vergleich dazu weniger formal. Der Autor des zu prüfenden Dokuments führt die Teilnehmer durch den Prüfling. Diese stellen Fragen und weisen auf Fehler, Verletzungen von Standards oder andere mögliche Probleme hin. Dabei dürfen, im Gegensatz zur Inspektion, auch direkt Lösungen für die angesprochenen Probleme aufgezeigt werden.

2.4.3. Requirements Engineering

Korrekte Anforderungen und klare Ziele, die dadurch erreicht werden sollen, entscheiden über den Erfolg eines Softwareprojekts. Laut dem CHAOS Report der Standish Group (1995) sind unzureichende Anforderungen einer der Hauptgründe für das Scheitern von Softwareprojekten. Ebert (2014) benennt deswegen fehlende, falsche und sich ändernde Anforderungen als Risiken, die es zu erkennen und zu beachten gilt.

Das Konzept des systematischen Requirements Engineerings (RE) soll deswegen das Auftreten fehlender und falscher Anforderungen reduzieren sowie das Risiko sich ändernder Anforderungen senken bzw. beherrschbar machen. Ebert (2014) definiert dazu passend das Ziel systematischen Requirements Engineerings:

> Das Ziel von RE ist es, qualitativ gute - nicht perfekte - Anforderungen zu entwickeln und sie in der Umsetzung risiko- und qualitätsorientiert zu verwalten.
>
> Ebert (2014)

Ebert (2014) liefert auch folgende Definition des Requirements Engineerings, in der auch die einzelnen Aktivitäten (Ermittlung, Dokumentation, Analyse, Prüfung, Abstimmung und Verwaltung), mit denen das oben genannte Ziel erreicht werden soll, aufgezählt werden:

> **Requirements Engineering** – Requirements Engineering (RE) ist das disziplinierte und systematische Vorgehen zur Ermittlung, Dokumentation, Analyse, Prüfung, Abstimmung und Verwaltung von Anforderungen unter kundenorientierten, technischen und wirtschaftlichen Vorgaben.
>
> Ebert (2014)

Pohl (2008) identifiziert die zwei Aktivitäten Prüfung (Validierung) und Verwaltung (Management) als Querschnittaktivitäten, die über den gesamten Prozess hinweg durchgeführt werden müssen. Die kontinuierliche Prüfung und Verwaltung von Anforderungen ist wichtig, um eine hohe Qualität der entwickelten Software sicherzustellen.

Im Folgendem wird zuerst der Begriff Anforderungen definiert und beschrieben. Anschließend wird das kontinuierliche Requirements Engineering eingeführt und zum phasenbezogenen Requirements Engineering abgegrenzt, das testorientierte Requirements Engineering erläutert sowie die Verwaltung und Änderung von Anforderungen ausgeführt. Zum Schluss wird die Ermittlung, die Dokumentation, die Modellierung und Analyse, die Prüfung und die Abstimmung von Anforderungen erläutert.

Anforderungen

Im IEEE Standard 610.12 (1990) wird der Begriff Anforderung wie folgt definiert:

> **requirement** – (1) A condition or capability needed by a user to solve a problem or achieve an objective.
> (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
> (3) A documented representation of a condition or capability as in (1) or (2).
>
> IEEE Standard 610.12 (1990)

Die Definition macht eine Unterscheidung bezüglich des Ursprungs einer Anforderung. Anforderungen können zum einen von einem Benutzer formuliert werden und drücken dann dessen Wünsche oder Erwartungen an eine Software aus oder sie können sich aus Verträgen oder Normen ableiten, die die Software erfüllen muss. Ebert (2014) und Pohl (2008) unterscheiden dazu passende funktionale Anforderungen, Qualitätsanforderungen und Randbedingungen.

Funktionale Anforderungen und Qualitätsanforderungen haben ihren Ursprung in den Erwartungen und Wünschen der Benutzer, während die Randbedingungen sich entweder aus Verträgen und Normen ableiten oder sich aus technischen Anforderungen ergeben.

Funktionale Anforderungen beschreiben laut Ebert (2014) Funktionen, die von der Software bereitzustellen sind. Sie lassen sich während der Entwicklung leicht verfolgen und können leicht verifiziert sowie validiert werden. Pohl (2008) liefert dazu passend folgende Definition:

> **Funktionale Anforderung** – Eine funktionale Anforderung definiert eine vom System bzw. von einer Systemkomponente bereitzustellende Funktion oder einen bereitzustellenden Service. Als Benutzeranforderung kann eine funktionale Anforderung sehr allgemein beschrieben sein. Als Bestandteil einer Spezifikation beschreibt eine funktionale Anforderung detailliert die Eingaben und Ausgaben sowie bekannte Ausnahmen.
>
> Pohl (2008)

Ebert (2014) beschreibt Qualitätsanforderungen, die auch als nicht-funktionale Anforderungen bezeichnet werden, als qualitative Eigenschaften, die die Software oder einzelne Funktionen der Software aufweisen müssen. Sie ergänzen die funktionalen Anforderungen und sind nicht so einfach zu verfolgen, verifizieren und validieren. Wieder liefert Pohl (2008) eine passende Definition:

> **Qualitätsanforderung** – Eine Qualitätsanforderung definiert eine qualitative Eigenschaft des gesamten Systems, einer Systemkomponente oder einer Funktion.
>
> Pohl (2008)

Nach Ebert (2014) sind Randbedingungen organisatorische und technische Anforderungen, die die Realisierung der Software einschränken. Sie ergänzen funktionale Anforderungen und Qualitätsanforderungen. Robertson u. Robertson (2012) definiert Randbedingungen wie folgt (von Pohl (2008) übersetzt):

> **Randbedingungen** – Eine Rahmenbedingung ist eine organisatorische oder technologische Anforderung, die die Art und Weise einschränkt, wie ein Produkt entwickelt wird.
>
> Robertson u. Robertson (2012)

Laut Ebert (2014) können Anforderungen auch anhand der verschiedenen Sichten auf sie in Markt-, Produkt-, und Komponentenanforderungen unterschieden werden (vgl. Abbildung 2.6). Marktanforderungen beschreiben dabei Anforderungen aus Sicht des Kunden. Sie werden auch als Kunden-, Benutzer-, Geschäftsanforderungen oder Bedürfnisse bezeichnet. Sie beschreiben den Nutzen eines Produkts und werden im Lastenheft dokumentiert. Produktanforderungen beschreiben dagegen Anforderungen aus Sicht einer späteren Lösung. Sie beschreiben wie

Bedürfnisse in ein Produkt umgesetzt werden können und werden im Pflichtenheft dokumentiert. Komponentenanforderungen beschreiben schließlich Anforderungen an eine Komponente eines Produkts. Sie beschreiben aus Sicht der Lösung, wie Produktanforderungen durch eine Komponente des Produkts adressiert werden. Eine klare Trennung zwischen Anforderungen und Lösung ist entscheidend. Dies gilt auch, wenn beide sich im selben Dokument befinden.

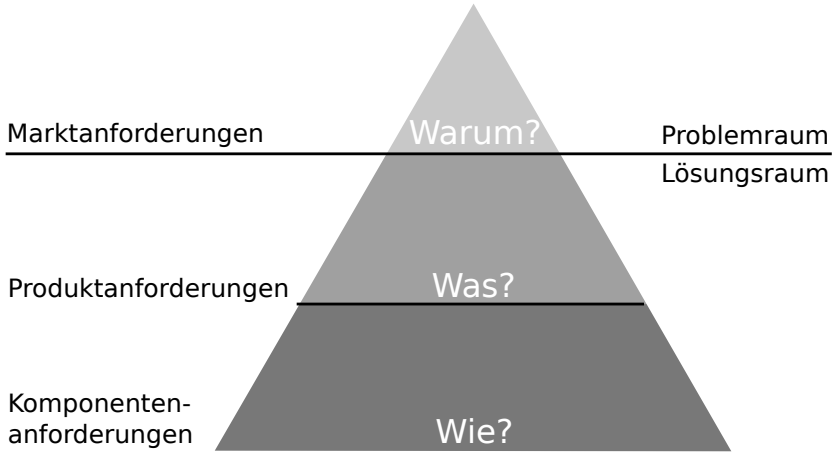

Abbildung 2.6.: Anforderungen und Lösungen (vgl. Ebert (2014) Abb. 2-2 S. 23).

Kontinuierliches Requirements Engineering

Pohl (2008) beschreibt das phasenbezogene Requirements Engineering als auf eine einzelne Phase beschränkt, in der Anforderungen entwickelt und in einer Spezifikation dokumentiert werden. Diese Spezifikation dient dann als Grundlage für alle weiteren Phasen. Dadurch ist die Änderung von Anforderungen nur schwer möglich, was zu Inkonsistenzen im Laufe der Entwicklung führt. Das kontinuierliche Requirements Egineering dagegen ist als phasen- und projektübergreifende Aktivität konzipiert. Phasenübergreifendes Requirements Engineering ist eine Querschnitttätigkeit, d.h. sie kommt im gesamten Entwicklungsprozess zum Einsatz. Durch sie kann eine konsistente Erfassung und Verwaltung von Anforderungen über den gesamten Prozess gewährleistet werden. Es entsteht eine kontinuierliche Wechselwirkung zwischen Requirements Engineering und Projekt, wodurch ein wirksames Änderungsmanagement möglich wird. Projektübergreifendes Requirements Engineering erweitert den Fokus des phasenübergreifenden Requirements Engineering. Durch ein projektübergreifendes Requirements Engineering ist es möglich, Anforderungen ohne die Bindung an konkrete Projekte zu erheben und zu verwalten, wodurch eine Wiederverwendung von Anforderungen möglich wird.

Testorientiertes Requirements Engineering

Laut Ebert (2014) gehören erfolgreiches Requirements Engineering und Tests zusammen. Durch die parallele Entwicklung von Anforderungen und Testfällen (vgl. Abbildung 2.7) verbessern sich die Anforderungen. Unvollständige Anforderungen sowie ungenaue und unklare Anforderungen werden frühzeitig erkannt und können verbessert werden. Ebenfalls wird durch das Schreiben von Testfällen die Testbarkeit der Anforderungen sicher gestellt. Auch die Umsetzbarkeit der Anforderungen und der Integrationsaufwand werden frühzeitig betrachtet, was zu einer höheren Planungssicherheit führt. Zusammengefasst sind die Vorteile des testorientierten Requirements Engineering:

- Vollständigkeit der Anforderungen
- Genauigkeit und Klarheit der Anforderungen
- Testbarkeit der Anforderungen
- Abschwächung der Randbedingungen

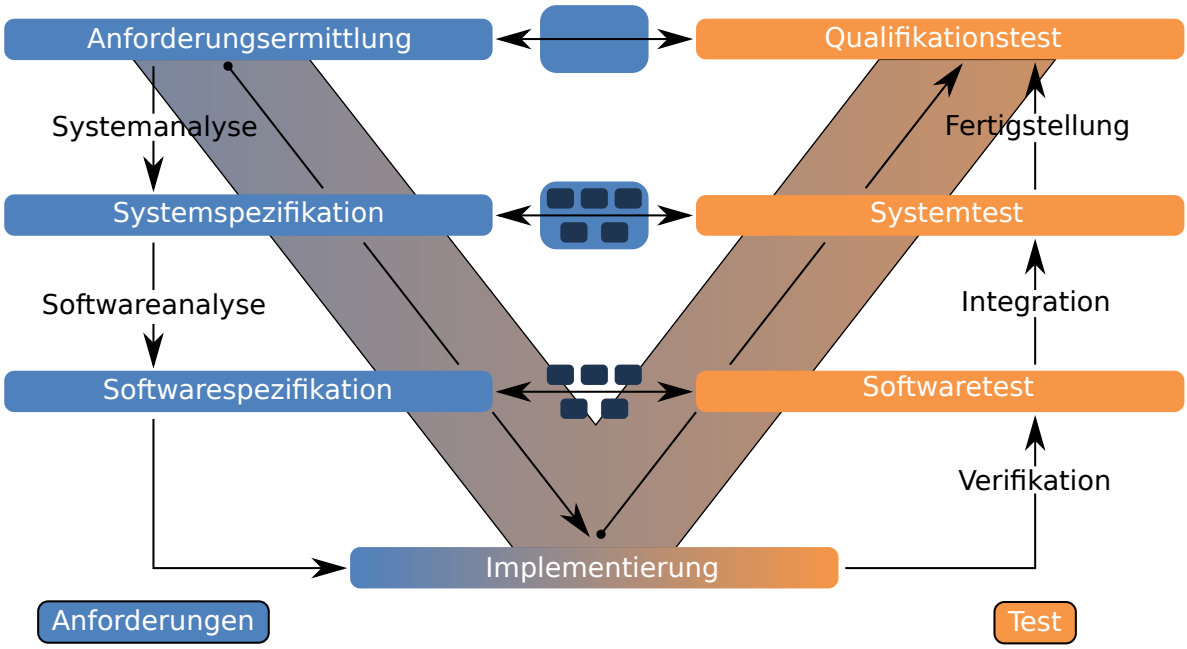

Abbildung 2.7.: Testorientiertes Requirements Engineering (vgl. Ebert (2014) Abb.6-3 S.193).

Änderungsmanagement und Verfolgbarkeit

Da Anforderungsänderungen in der Softwareentwicklung Alltag sind, muss ein Entwicklungsprozess über ein wirksames Änderungsmanagement verfügen. Da Änderungen zu jedem Zeitpunkt der Lebensdauer einer Anforderung auftreten können, ist ein Konfigurationsmanagement, in dem die Anforderungen über ihre gesamte Lebensdauer verwaltet werden können, von entscheidender Bedeutung (vgl. Abbildung 2.8).

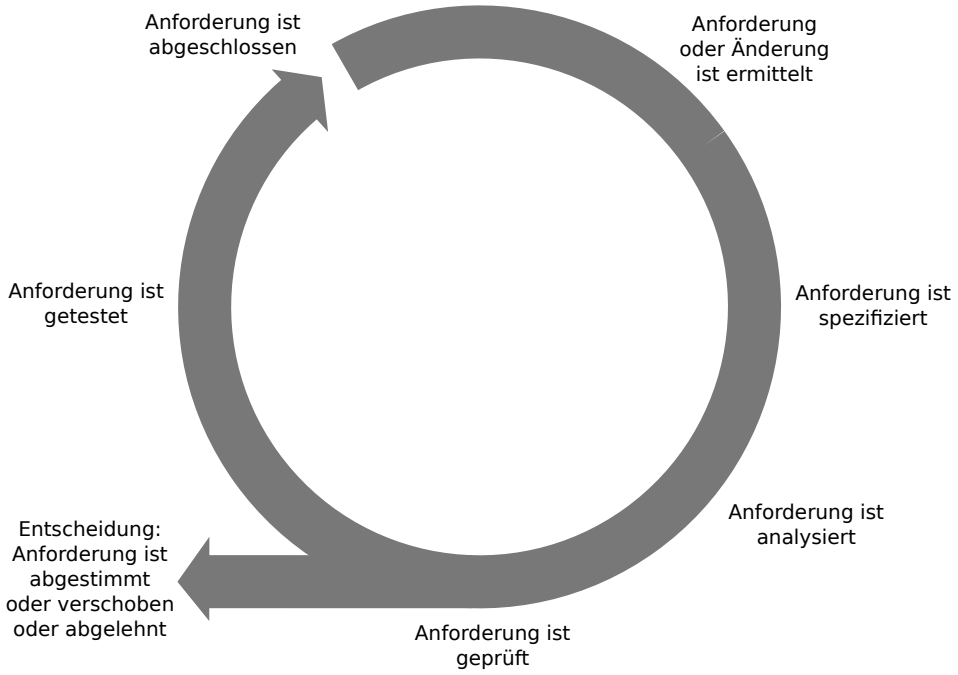

Abbildung 2.8.: Der Lebenszyklus einer Anforderung (vgl. Ebert (2014) Abb.8-2 S.246).

Außerdem muss eine Änderung nicht nur an der Stelle durchgeführt werden, an der sie auftritt, sondern auch an allen anderen Stellen, zu denen Abhängigkeiten bestehen. Dafür müssen die Abhängigkeiten sichtbar sein. Für ein wirksames Änderungsmanagement ist deswegen eine Verfolgbarkeit (engl. *traceability*) der Anforderungen über den gesamten Lebenszyklus hinweg von entscheidender Bedeutung. Dabei muss zwischen vertikaler und horizontaler Verfolgbarkeit von Anforderungen unterschieden werden. Bei der horizontalen Verfolgbarkeit geht es um eine Verfolgung einer Anforderung von ihrer Quelle, typischerweise der Anforderungsanalyse, bis zu ihrer Umsetzung, typischerweise der Implementierung, über alle dazwischenliegenden Schritte im Entwicklungsprozess. Kann die Anforderung auch umgekehrt von der Umsetzung zur Quelle zurückverfolgt werden, spricht man von bidirektionaler Verfolgbarkeit. Bei der horizontalen Verfolgbarkeit von Anforderungen geht es dagegen darum, die Abhängigkeiten zwischen Anforderungen verfolgbar zu machen machen. Abbildung 2.9 zeigt welche direkten und indirekten Abhängigkeiten bei der Änderung einer Anforderung in der Anforderungsspezifikation bestehen.

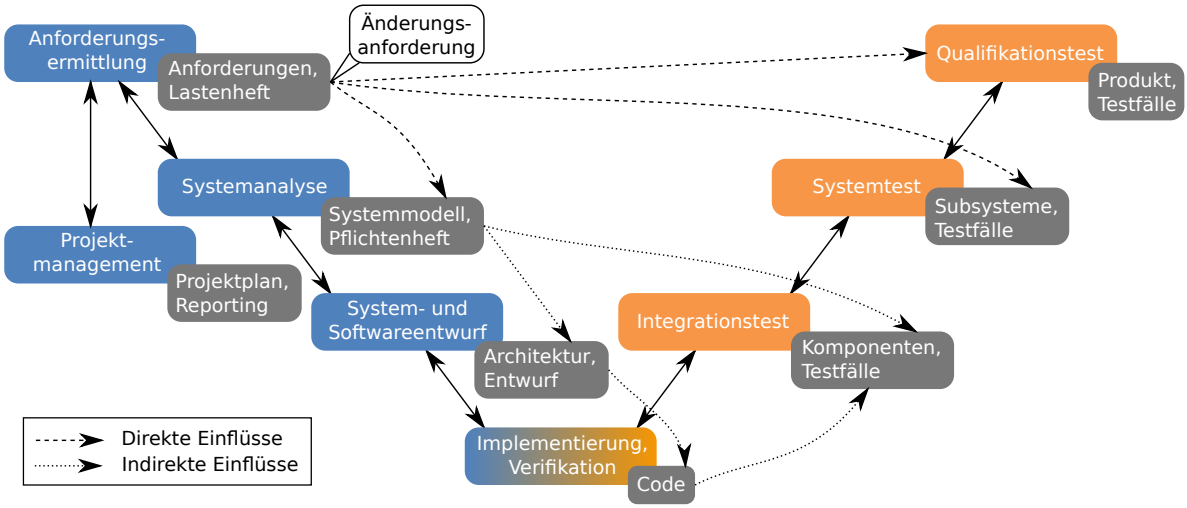

Abbildung 2.9.: Verfolgbarkeit erleichtert das Änderungsmanagement (vgl. Ebert (2014) Abb.8-3 S.249).

Anforderungen ermitteln, dokumentieren, prüfen, modellieren, analysieren und abstimmen

Die Ermittlung der Anforderungen erfolgt in Gesprächen mit dem Kunden. Dabei ist es wichtig, alle Anforderungen zu erheben, auch die, die der Kunde implizit an die neue Software stellt. Um dies zu erreichen ist es hilfreich, nicht nur den Soll-Zustand sondern auch den Ist-Zustand in den Gesprächen zu erfragen. Außerdem helfen Checklisten und Fragebögen bei der Ermittlung verborgener Anforderungen. Außer den funktionalen Anforderungen müssen auch die Qualitätsanforderungen sowie evtl. Randbedingungen ermittelt werden. Es können auch negative Anforderungen ermittelt werden. Diese können genutzt werden, um Szenarien, Fälle und Verhalten zu beschreiben, die nicht eintreten dürfen. Zu diesem Zeitpunkt darf noch keine Wertung der Anforderungen durch die Entwickler vorgenommen werden. Auch widersprüchliche oder nicht umsetzbare Anforderungen müssen ermittelt werden.

> Anforderungen sollen so strukturiert und formal wie nötig ermittelt, spezifiziert und analysiert werden – und so leichtgewichtig wie möglich.
>
> Ebert (2014)

Sind alle Anforderungen ermittelt, werden sie in einem nächsten Schritt in eine Spezifikation für die neue Software umgesetzt. Dabei auftretende Widersprüche müssen dem Kunden aufgezeigt werden. Dabei können auch potentielle Lösungsvorschläge präsentiert werden. Der Kunde muss anschließend entscheiden, welche Anforderungen beibehalten, geändert oder gar verworfen werden, um die Widersprüche aufzulösen. Es ist wichtig, dass nicht die Entwickler die abschließende Entscheidung treffen. Bei der Dokumentation der Anforderungen ist eine

sprachlich exakte Beschreibung wichtig. Die Beschreibung muss testbar und entscheidbar sein. Vorlagen helfen, um eine einheitliche Struktur und die Lesbarkeit des Spezifikation zu gewährleisten. Außerdem ist auf eine klare Trennung zwischen Aufgaben- und Lösungsbeschreibung zu achten. Ebert (2014) nennt folgende Ergebnisse der Anforderungsermittlung:

- Vision der Software
- Bedürfnisse und Erwartungen an die Software
- Kontext und Randbedingungen
- Dokumentierte Anforderungen als kontrollierte Basis für die weitere Entwicklung

Die dokumentierten Anforderungen müssen anschließend einem formellen Review unterzogen werden. In diesem Review prüfen die Gutachter die Spezifikation auf Qualitätskriterien wie Konsistenz, Korrektheit und Vollständigkeit sowie auf die Übereinstimmung mit den relevanten Standards, Normen und Richtlinien. Ebert (2014) nennt in Anlehnung an IEEE Standard 830 (1998) folgende Qualitätskriterien für die Prüfung der Anforderungen:

- Wertorientierung
- Verständlichkeit
- Relevanz
- Realisierbarkeit
- Korrektheit
- Eindeutigkeit
- Vollständigkeit
- Konsistenz
- Bewertbarkeit
- Prüfbarkeit
- Modifizierbarkeit
- Verfolgbarkeit

Auf Grundlage der in der Spezifikation formulierten Anforderungen werden alle Testfälle für den Systemtest erstellt.

Nachdem die Anforderungsspezifikation geprüft wurde, müssen im Sinne des testorientierten Requirements Engineering die Testfälle für den Systemtest und die Abnahme erstellt werden. Dadurch soll sichergestellt werden, dass auf die spezifizierten Anforderungen getestet wird und nicht auf die, die im Laufe der Entwicklung implementiert wurden. Dabei ist es wichtig, die Testfälle systematisch anhand von Prüftechniken zu erstellen. Wichtige Prüftechniken sind z. B. Äquivalenzklassen und Grenzwertbetrachtung (vgl. Abschnitt 2.4.2). Außerdem müssen die Testfälle alle Anforderungen abdecken. Für die Verfolgbarkeit der Anforderungen ist es wichtig, dass die Testfälle den Anforderungen, die sie abdecken, zugeteilt werden.

Als letztes folgt die Modellierung der Anforderungen in einem Entwurf. Dafür müssen zuerst die Anforderungen in einer geeigneten Softwarearchitektur umgesetzt werden. Dabei sind vor allem die Qualitätsanforderungen entscheidend, da diese in einer ungeeigneten Softwarearchitektur nur schwer erfüllt werden können. Danach müssen die funktionalen Anforderungen für die einzelnen Modulen in einem Modulentwurf umgesetzt werden. Sowohl in der Softwarearchitektur als auch im Modulentwurf müssen die Anforderungen verfolgbar sein, d. h. es muss klar sein, welche Anforderungen wo im Entwurf umgesetzt wird.

2.4.4. Continuous Integration

In einem herkömmlichen Softwareprojekt findet die Integration erst am Ende der Softwareentwicklung statt. Dadurch werden potentielle Probleme erst spät entdeckt, sind somit nicht kalkulierbar und nur mit großem Aufwand zu beheben. Um diese Probleme bei der Integration zu lösen, wurde die Methode Continuous Integration entwickelt. Ursprünglich wurde sie von Booch (1991) vorgeschlagen und später als Teil des eXtreme Programming (XP) aufgenommen. Fowler (2006) beschreibt in seinem Essay seine Erfahrungen und wie die Methode heutzutage eingesetzt werden kann. Duvall u. a. (2007) liefern in ihrem Buch eine ausführliche Beschreibung der Methode.

Das Grundprinzip der Methode ist die regelmäßige Integration der Software. Die Entwickler werden angehalten, mindestens einmal pro Tag, am besten nach jeder implementierten Funktionalität, eine Integration durchzuführen. Dafür wird der Programmcode der Versionsverwaltung unterstellt und ein automatischer Build inklusive Tests ausgeführt. Dies wird von Fowler als Commit Build bezeichnet. Anschließend werden alle gefundenen Fehler direkt behoben. Dabei ist es wichtig, dass keine langen Wartezeiten entstehen, sondern die Entwickler direkt eine Rückmeldung bekommen. Fowler nennt deswegen eine Dauer von 10 Minuten für den Commit Build als Maximum. Dies hat zur Folge, dass beim Commit Build nicht alle Tests durchgeführt werden können, sondern nur eine sinnvolle Untermenge. Dabei muss eine Balance zwischen der Anforderung, möglichst viele Fehler zu finden, und der Anforderung an die Geschwindigkeit gefunden werden. Zusätzlich kann ein weiterer Build durchgeführt werden, in dem alle Test, auch die zeitintensiven, ausgeführt werden. Häufig wird dies als Nightly-Build durchgeführt.

Der Vorteil dieses Vorgehens ist, dass Fehler und Probleme der Integration direkt während der Implementierung gefunden werden und dadurch zeitnah behoben werden können. Dies führt zu einer kohäsiveren Software und einer besseren Planbarkeit der Softwareentwicklung.

Durch den automatisierten Charakter der Methode, ist eine ausreichende Werkzeugunterstützung wichtig. Neben einer Konfigurationsverwaltung ist dabei ein Continuous Integration Server von zentraler Bedeutung. Außerdem ist ein geeignetes Framework für die Durchführung der Tests notwendig (vgl. Abschnitt 2.3).

3. TWT Softwareentwicklungsprozesse

In diesem Kapitel werden die Softwareentwicklungsprozesse und die eingesetzten Softwarewerkzeuge bei TWT beschrieben und die sich daraus ergebenden Anforderungen an den Qualitätsprozess diskutiert. Die Ausprägungen der Softwareentwicklungsprozesse bei TWT unterscheiden sich je nach Projekt stark voneinander. Dies ist unter anderem durch Vorgaben der Kunden bedingt, die in den jeweiligen Projekten involviert sind. Deswegen werden in diesem Kapitel ausschließlich die Kernaktivitäten der Softwareentwicklungsprozesse identifiziert und die ihnen zugrunde liegenden Konzepte herausgearbeitet.

3.1. Konzepte und Kernaktivitäten

Abbildung 3.1 zeigt eine Übersicht der Kernaktivitäten der Softwareentwicklungsprozesse bei TWT, bestehend aus Planungs-, Entwicklungs- und Qualitätssicherungsaktivitäten, sowie die Abschnitte bzw. Konzepte des Softwareentwicklungsprozesses zu denen sie gehören. Des Weiteren sind den Abschnitten die verwendeten Qualitätssicherungskonzepte und die wichtigsten Softwarewerkzeuge zugeordnet, die bei TWT zum Einsatz kommen. Bei den Abschnitten, die im Folgenden näher erläutert werden, handelt es sich um das Management, die Anforderungsanalyse, die Continuous Integration und das Release.

3.1.1. Management

Das Management besteht aus den Planungsaktivitäten Projektinitialisierung und Projektende. In der Projektinitialisierung werden die klassischen Projektmanagementaufgaben Aufwand- und Kostenschätzung, Personalplanung, Risikomanagement usw. durchgeführt. Am Projektende wird ein Projektbericht verfasst und das Projekt abgeschlossen.

3.1.2. Anforderungsanalyse

Die Anforderungsanalyse besteht aus den Entwicklungsaktivitäten Erhebung der Anforderungen und Dokumentation der Anforderungen. Die Erhebung der Anforderungen unterscheidet sich in den einzelnen Projekten, abhängig vom Kunden. Die Anforderungen können sowohl über ein Lasten- und Pflichtenheft erhoben werden als auch in protokollierten Gesprächen mit dem

Kunden. Unabhängig von der Form der Erhebung werden sie mit dem Issue-Tracking-System FogBugz dokumentiert und über die Dauer des Projektes verwaltet und verfolgt.

3.1.3. Continuous Integration

Das Konzept der Continuous Integration (vgl. Abschnitt 2.4.4) wird durch die Entwicklungs- und Qualitätssicherungsaktivitäten Implementierung / Korrektur, Statische Codeanalyse, Modultests, Integrationstest und Überdeckungsmetriken umgesetzt. Von zentraler Bedeutung für die Umsetzung sind die eingesetzten Softwarewerkzeuge. Die Konfigurationsverwaltung wird durch die Versionsverwaltungssysteme Mercurial und Kiln realisiert. Als Continuous Integration-Server wird Jenkins eingesetzt. Abhängig von der verwendeten Programmiersprache werden mit Sonar oder Resharper die statische Codeanalyse durchgeführt sowie die Überdeckungsmetriken erhoben.

3.1.4. Release

Das Release schließt eine Iteration der Entwicklung ab und besteht aus den Qualitätssicherungsaktivitäten Systemtest und Abnahme. Im Systemtest wird die neue Version der Software vollständig anhand vorher erstellter Testfälle geprüft. Abschließend erfolgt die Abnahme der neuen Version durch den Kunden.

3.2. Anforderungen an den Qualitätsprozess

Da sich die Softwareentwicklungsprozesse von Projekt zu Projekt stark unterscheiden, bleibt als zentrales Qualitätssicherungskonzept nur die Continuous Integration übrig, die in allen Projekten eingesetzt wird. Deswegen muss der zu definierende Softwareentwicklungsprozess für die Entwicklung qualifizierbarer Softwarewerkzeuge die kontinuierliche Integration umsetzen und die eingesetzten Softwarewerkzeuge wiederverwenden.

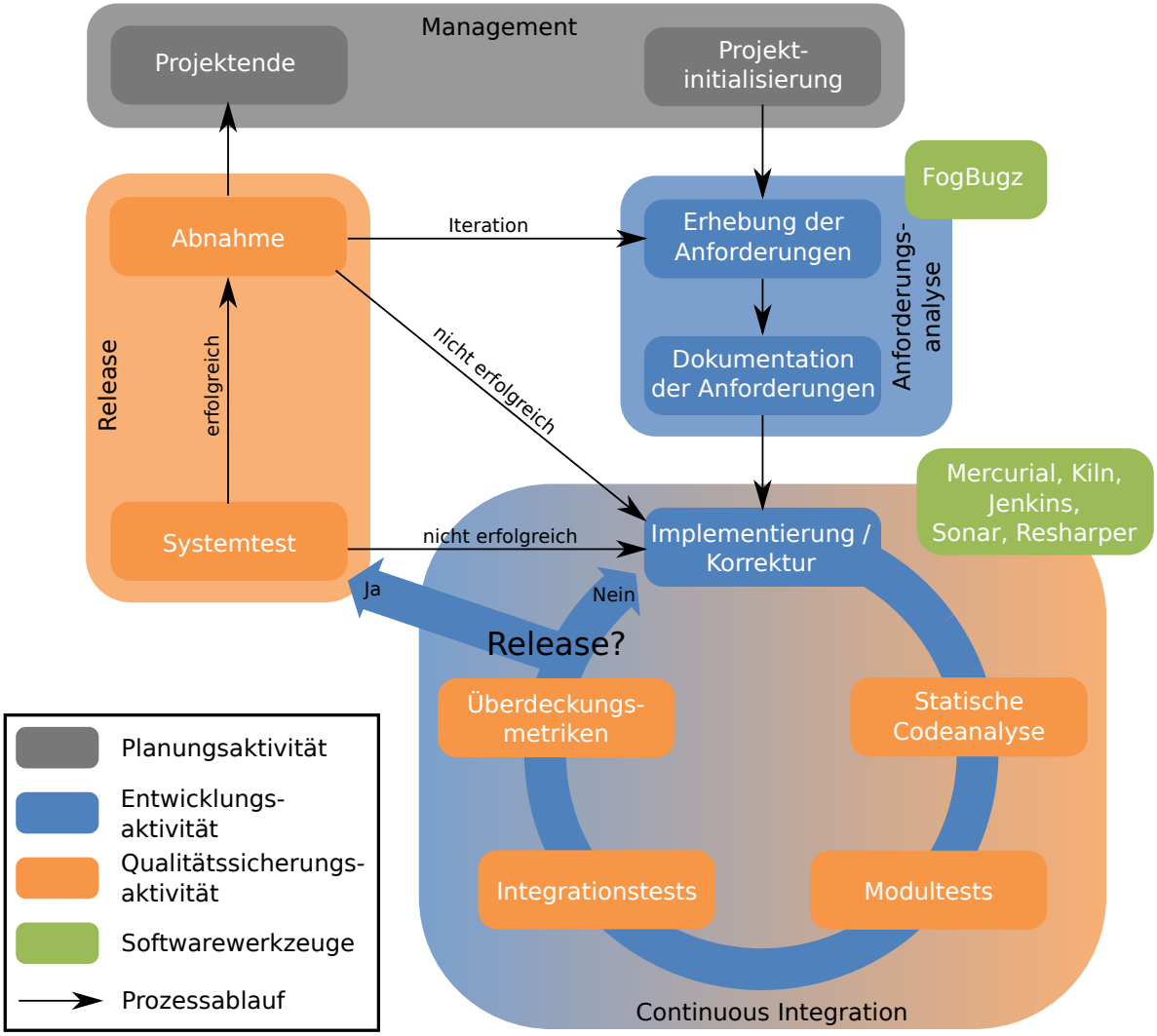

Abbildung 3.1.: Übersicht der Kernaktivitäten der Softwareentwicklungsprozesse bei TWT, die Abschnitten bzw. Konzepte, denen sie zugeordnet sind sowie die eingesetzten Softwarewerkzeugen: das Management mit den Planungsaktivitäten Projektinitialisierung und Projektende, die Anforderungsanalyse mit den Entwicklungsaktivitäten Erhebung der Anforderungen und Dokumentation der Anforderungen sowie FogBugz als Issue-Tracker, das Konzept der Continuous Integration mit den Entwicklungs- und Qualitätssicherungsaktivitäten Implementierung / Korrektur, Statische Codeanalyse, Modultests, Integrationstests und Überdeckungsmetriken sowie den Konfigurationsverwaltungswerkzeugen Mercurial und Kiln, dem Continuous Integration-Server Jenkins sowie den statische Codeanalyse-Werkzeugen Sonar und Resharper sowie das Release mit den Qualitätssicherungsaktivitäten Systemtest und Abnahme.

4. Konzept

Den Kern der Arbeit bildet die Definition eines Qualitätsprozesses, der die speziellen Anforderungen an die Entwicklung qualifizierbarer Softwarewerkzeuge berücksichtigt, die bei der Entwicklung eingebetteter Systeme im Automobilumfeld eingesetzt werden. In diesem Kapitel wird das Konzept des Qualitätsprozesses vorgestellt. Dafür werden zunächst die Anforderungen an den Qualitätsprozess hergeleitet. Anschließend wird das Konzept des Qualitätsprozesses im Detail vorgestellt.

4.1. Anforderungen an den Qualitätsprozess

Abbildung 4.1 zeigt den Kontext des Qualitätsprozesses und den sich daraus ergebenden Anforderungen und Abhängigkeiten. Der Qualitätsprozess soll für die Entwicklung qualifizierbarer Softwarewerkzeuge eingesetzt werden. Die Softwarewerkzeuge werden wiederum für die Entwicklung eingebetteter Systeme im Automobilumfeld eingesetzt. Aus den Anforderungen an die funktionale Sicherheit solcher eingebetteter Systeme ergeben sich Anforderungen an die eingesetzten Softwarewerkzeuge, die diese für eine erfolgreiche Qualifizierung erfüllen müssen (vgl. ISO 26262-8, Klausel 11). Dies wird auch als Tool Qualification bezeichnet (vgl. Abschnitt 2.1.3). Außerdem soll sich der Qualitätsprozess in die bestehenden Entwicklungsprozesse bei TWT einfügen. Auch die Softwarewerkzeuge, die für die Unterstützung des Qualitätsprozesses eingesetzt werden, sollen sich an den bisher eingesetzten Softwarewerkzeugen bei TWT orientieren (vgl. Kapitel 3).

Aus dem Umstand, dass sich der Qualitätsprozess in die bestehenden Softwareentwicklungsprozesse bei TWT einfügen soll und die bereits eingesetzten Softwarewerkzeuge soweit wie möglich wiederverwendet werden sollen, ergeben sich Abhängigkeiten. Diese Abhängigkeiten können wiederum als Anforderungen an den Qualitätsprozess formuliert werden. Die bestehenden Softwareentwicklungsprozesse verwenden als gemeinsamen Kern das Konzept der Continuous Integration, das der Qualitätsprozess deswegen ebenfalls umsetzen muss (vgl. Abschnitt 2.4.4).

Die Anforderungen an die Qualifizierung von Softwarewerkzeugen gemäß des Sicherheitsstandards ISO 26262 werden in Abschnitt 2.1.3 behandelt. Zusammengefasst lauten sie wie folgt:

- Erhöhtes Vertrauen in das Softwarewerkzeug aus den Erfahrungen früherer Nutzungen

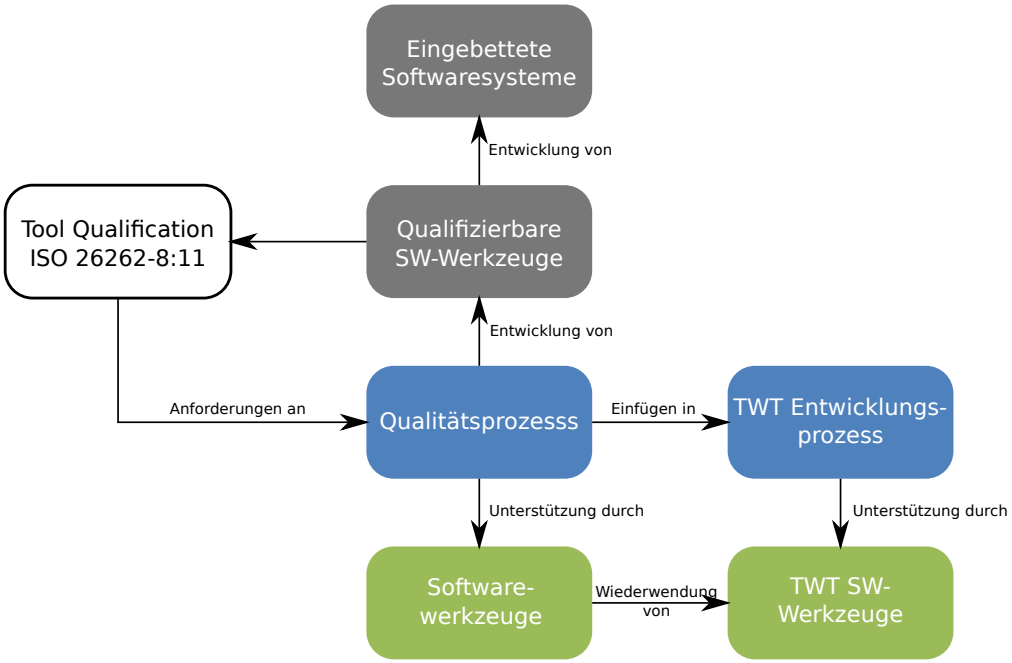

Abbildung 4.1.: Der Qualitätsprozess soll zur Entwicklung qualifizierbarer Softwarewerkzeuge eingesetzt werden, die wiederum bei der Entwicklung eingebetteter Softwaresysteme zum Einsatz kommen. Die Anforderungen an solche Softwarewerkzeuge sind in ISO 26262-8, Klausel 11 definiert. Der Qualitätsprozess soll sich in die bestehenden Entwicklungsprozesse einfügen und zur Unterstützung die bestehenden Softwarewerkzeuge wiederverwenden.

- Evaluierbarkeit des Softwareentwicklungsprozesses
  - Übereinstimmung des Softwareentwicklungsprozesses mit einem Standard zur Softwareentwicklung (z. B. Wasserfallmodell, V-Modell)
  - Evaluierung der Übereinstimmung durch einen Standard zur Begutachtung von Softwareentwicklungsprozessen (z. B. Automotive SPICE, CMMI oder ISO 15504)
- Validierbarkeit des entwickelten Softwarewerkzeugs
  - Nachträgliche Validierung der Übereinstimmung des Softwarewerkzeugs mit den spezifizierten Anforderungen (z. B. durch entsprechende Tests)
  - Analyse der während der Validierung auftretenden Fehlfunktionen und ihrer Konsequenzen
  - Analyse der Reaktion des Softwarewerkzeugs auf anormale Einsatzbedingungen
- Übereinstimmung des Softwareentwicklungsprozesses mit einem Sicherheitsstandard (z. B. ISO 26262, IEC 61508 oder RTCA DO-178)

Aus den Anforderungen an die Qualifizierung von Softwarewerkzeugen lassen sich Anforderungen an den Qualitätsprozess ableiten, um so bereits bei der Entwicklung der Softwarewerkzeuge die spätere Qualifizierung zu unterstützen. Allerdings sind nicht alle der Anforderungen direkt auf die Definition eines Qualitätsprozesses übertragbar, der die Qualifizierung neu entwickelter Softwarewerkzeuge unterstützen soll. So lässt sich offensichtlich bei einem neu entwickelten Softwarewerkzeug kein erhöhtes Vertrauen aus den Erfahrungen früherer Nutzung ableiten. Die nachträgliche Validierung des Softwarewerkzeugs für die Qualifizierung lässt sich dagegen bereits bei der Entwicklung des Softwarewerkzeugs sicherstellen bzw. unterstützen. Dafür ist es notwendig, die entsprechenden Test bereits während der Entwicklung durchzuführen und ihre Ergebnisse zu dokumentieren. Der Umgang mit Fehlfunktionen sowie die Reaktion auf anormale Einsatzbedingungen lassen sich ebenfalls bereits während der Entwicklung unterstützen. Dafür ist es notwendig auf eine hohe Fehlertoleranz des Softwarewerkzeugs zu achten. Für die Übereinstimmung des Qualitätsprozesses mit einem Sicherheitsstandard muss eine, für die Entwicklung von Softwarewerkzeugen relevante, Untermenge der Anforderungen des Sicherheitsstandards ausgewählt werden. Die Anforderung an die Evaluierbarkeit des Qualitätsprozesses kann direkt übernommen werden.

Die aus der Qualifizierung von Softwarewerkzeugen abgeleiteten Anforderungen und die aus den Abhängigkeiten von bestehenden Softwareentwicklungsprozessen entstehenden Anforderungen ergeben insgesamt folgende Anforderungen an den Qualitätsprozess:

- Der Qualitätsprozess muss evaluierbar sein, d. h.
  - er muss mit einem Softwareentwicklungsstandard übereinstimmen (z. B. V-Modell) und
  - die Übereinstimmung muss durch eine Begutachtung gemäß Automotive SPICE oder CMMI überprüft werden,
- der Qualitätsprozess muss eine Validierung des entwickelten Softwarewerkzeugs ermöglichen, d. h.
  - die Übereinstimmung des Softwarewerkzeugs mit den spezifizierten Anforderungen muss z. B. durch entsprechende Tests nachgewiesen werden und
  - das Softwarewerkzeug muss eine hohe Fehlertoleranz gegenüber Fehlfunktionen und anormalen Einsatzbedingungen aufweisen,
- der Qualitätsprozess muss mit den Teilen des Sicherheitsstandards ISO 26262 übereinstimmen, die für die Entwicklung von Softwarewerkzeugen relevant sind, und
- der Qualitätsprozess muss eine kontinuierliche Integration anwenden.

4.2. Konzept des Qualitätsprozesses

Abbildung 4.2 zeigt das Konzept des Qualitätsprozesses, in dem alle oben genannten Anforderungen umgesetzt sind. Durch die Wahl des V-Modells (vgl. Abschnitt 2.2) als grundlegende Struktur des Qualitätsprozesses, wird ein Standard zur Softwareentwicklung verwendet und dadurch auch eine erfolgreiche Evaluierung durch Automotive SPICE oder CMMI sichergestellt. Der Qualitätsprozess orientiert sich durch die Wahl des V-Modells außerdem am Sicherheitsstandard ISO 26262, der ebenfalls ein V-Modell für die Softwareentwicklung verwendet (vgl. ISO 26262-6). Das V-Modell ist in die Phasen Analyse, Softwarearchitektur, Modulentwurf, Implementierung & Qualitätsanalyse und Systemtest unterteilt. Um Software iterativ zu entwickeln, kann außerdem über alle Phasen iteriert werden. Darüber hinaus sind in den Klauseln des Sicherheitsstandards ISO 26262-6 für jede Phase des V-Modells verschiedene Entwicklungs- und Qualitätssicherungsmethoden definiert, die bei der Softwareentwicklung eingesetzt werden sollen (vgl. Abschnitt 2.1.2). Da sich der Softwareentwicklungsprozess des Sicherheitsstandards ISO 26262 auf die Entwicklung eingebetteter Systeme bezieht, muss aus den definierten Methoden eine relevante Auswahl für die Entwicklung von qualifizierbaren Softwarewerkzeugen getroffen werden.

Durch den Einsatz eines kontinuierlichen, testorientierten Requirements Engineering wird die Validierbarkeit des Softwarewerkzeugs unterstützt. Die Anforderungen können so kontinuierlich von der Analyse bis zur Abnahme verfolgt werden. Durch die Testorientierung lässt sich die Übereinstimmung der spezifizierten mit den implementierten Anforderungen sicherstellen (vgl. Abschnitt 2.4.3).

Durch die Umsetzung einer kontinuierlichen Integration (vgl. Abschnitt 2.4.4) ist eine Anpassung des V-Modells, und damit des Sicherheitsstandards ISO 26262, notwendig. Dies ist grundsätzlich mit dem Sicherheitsstandard ISO 26262 vereinbar, da eine Anpassung des Standards vorgesehen ist (vgl. ISO 26262-6, Klausel 4.1). Durch eine kontinuierliche Integration werden die Entwicklungs- und Qualitätssicherungsaktivitäten der Phasen des V-Modells Implementierung, Modultest und Integrationstest zwar mehrfach, aber trotzdem noch vollständig durchlaufen. Dafür werden sie in der Phase Implementierung & Qualitätsanalyse zusammengefasst. Der Sicherheitsstandard ISO 26262 wird so noch immer eingehalten.

Abbildung 4.3 zeigt das Konzept der einzelnen Phasen des Qualitätsprozesses. Jeder Phase ist ein Quality Gate zugeordnet, das diese Phase von der darauf folgenden Phase trennt. In jeder Phase gibt es eine Abfolge von zueinander gehörenden Entwicklungs- und Qualitätssicherungsaktivitäten. In den Qualitätssicherungsaktivitäten werden die in den dazugehörigen Entwicklungsaktivitäten entwickelten Dokumente und Softwareartefakte überprüft. Am Ende einer Phase werden im Quality Gate die Anforderungen validiert, d. h. es wird überprüft, ob alle Anforderungen aus der vorherigen Phase auch in dieser Phase vollständig umgesetzt wurden. Außerdem werden die in der Phase erstellten Dokumente und Softwareartefakte verifiziert. Dabei dienen die in den Qualitätssicherungsaktivitäten entstandenen Qualitätssicherungsdokumente als Grundlage für die Verifizierung.

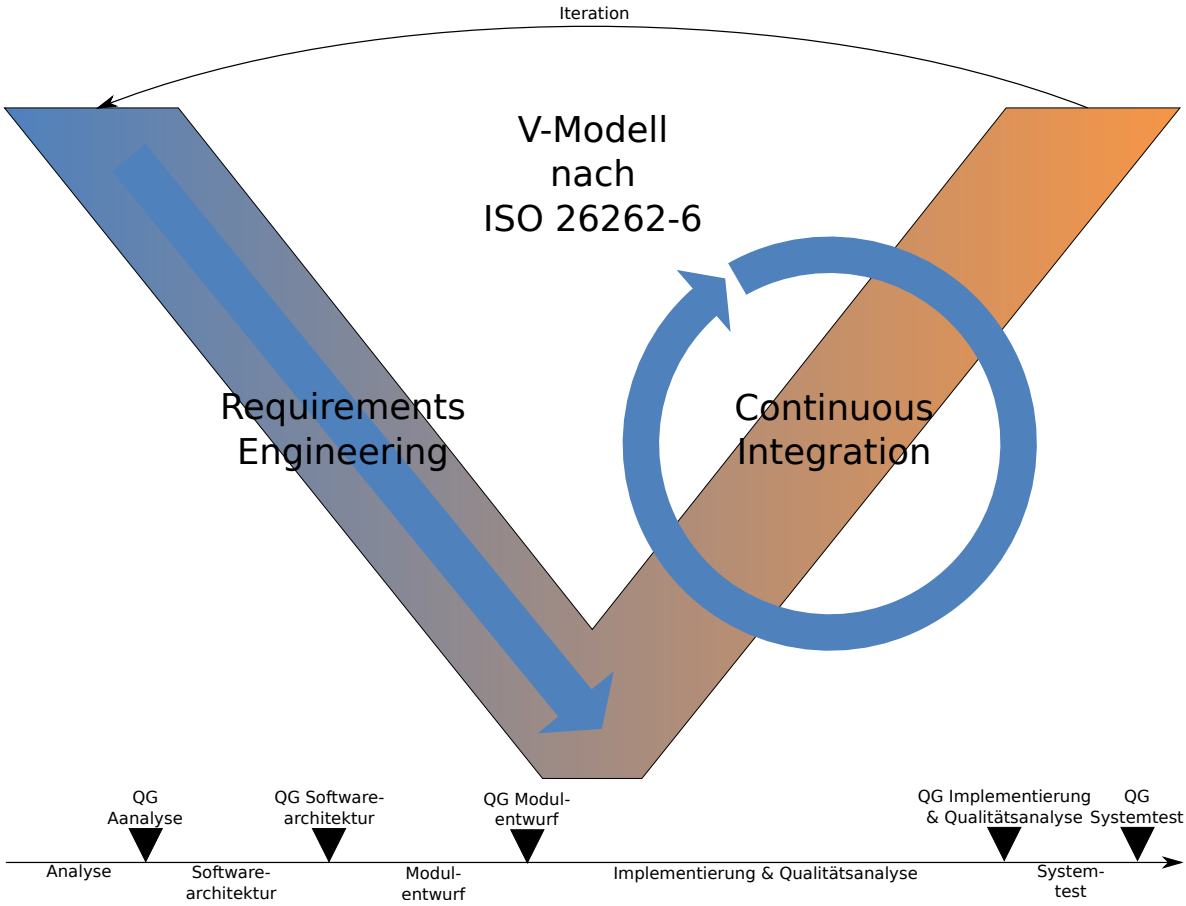

Abbildung 4.2.: Übersicht der verwendeten Konzepte und Prozesse des Qualitätsprozesses. Die grundlegende Struktur ist, in Anlehnung an ISO 26262-6, ein V-Modell mit den Phasen Analyse, Softwarearchitektur, Modulentwurf, Implementierung & Qualitätsanalyse und Systemtest. Zu jeder Phase gehört ein Quality Gate, das die Phase abschließt. Durch ein kontinuierliches, testorientiertes Requirements Engineering wird die Validierung der entwickelten Softwarewerkzeuge ermöglicht. Durch den Einsatz einer kontinuierlichen Integration fügt sich der Qualitätsprozess in die bestehenden Softwareentwicklungsprozesse ein. Um Software iterativ zu entwickeln, kann außerdem über alle Phasen iteriert werden.

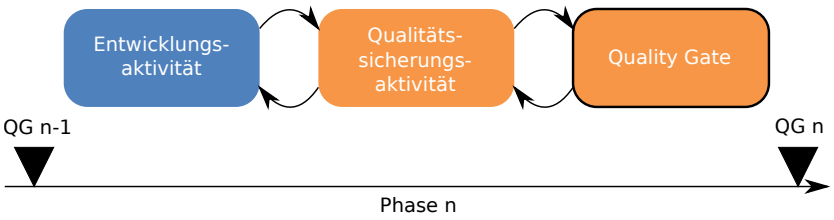

Abbildung 4.3.: In jeder Phase des Qualitätsprozesses finden Entwicklungs- und Qualitätssicherungsaktivitäten statt. Zu jeder Phase gehört ein Quality Gate, in dem die Ergebnisse der Phase validiert und verifiziert werden.

5. Qualitätsprozess

In diesem Kapitel wird das in Kapitel 4 vorgestellte Konzept des Qualitätsprozesses ausgearbeitet. Abbildung 5.1 zeigt Umfang und Grenzen des Qualitätsprozesses. So handelt es sich bei dem Qualitätsprozess nicht um ein vollständiges Prozessmodell, sondern er beschränkt sich auf das Vorgehens- und das Qualitätsmodell, d. h. dass Aspekte wie Personalmanagement oder Kostenschätzung nicht abgedeckt werden (vgl. Abschnitt 2.2). Des Weiteren werden im Qualitätsprozess zwar Empfehlungen für die wichtigsten Softwarewerkzeuge gegeben, die den Qualitätsprozess unterstützen, die komplette Softwareinfrastruktur wird jedoch nicht betrachtet (vgl. Abschnitt 2.3). Deswegen muss der Qualitätsprozess auf einen bestehenden Prozess sowie die vorhandene Infrastruktur aufgesetzt und angepasst werden. Aus diesem Grund werden an möglichst vielen Stellen im Prozess nur die verschiedenen Möglichkeiten dargelegt, z. B. bei den empfohlenen Entwicklungs- und Qualitätssicherungsmethoden, deren Auswahl dann bei der Umsetzung des Prozesses in einem konkreten Projekt getroffen werden muss.

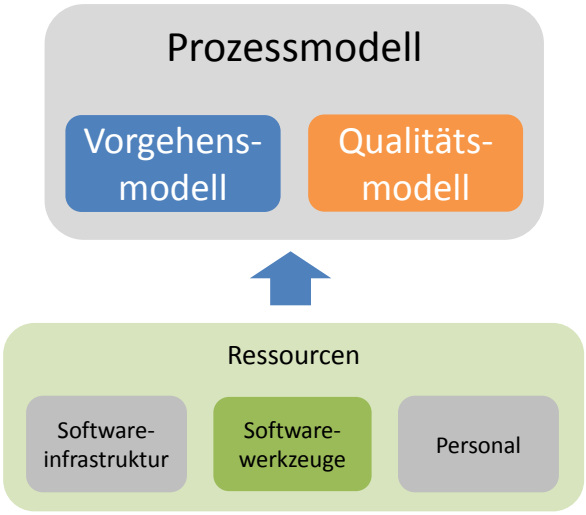

Abbildung 5.1.: Der Qualitätsprozess besteht aus einem Vorgehens- und einem Qualitätsmodell. Die anderen Aspekte eines vollständigen Prozessmodells, wie Personalplanung oder Aufwands- und Kostenschätzung, werden jedoch nicht abdeckt. Bei den Ressourcen, auf die der Prozess zurückgreift, werden ausschließlich einzelne Softwarewerkzeuge betrachtet, die komplette Softwareinfrastruktur und das Personal bleiben ebenfalls außen vor.

Im Folgenden wird zuerst eine Übersicht über den Qualitätsprozess gegeben. Anschließend wird das Prinzip der Quality Gates erläutert, inklusive der Eingaben und Ausgaben. Zum Schluss werden die einzelnen Phasen mit den zur Auswahl stehenden Entwicklungs- und Qualitätssicherungsmethoden, den zu entwickelnden Dokumenten und Softwareartefakten und den Anforderungen an die unterstützende Softwarewerkzeuge im Detail besprochen. Abbildung 5.2 zeigt die Legende für alle folgenden Abbildungen des Qualitätsprozesses.

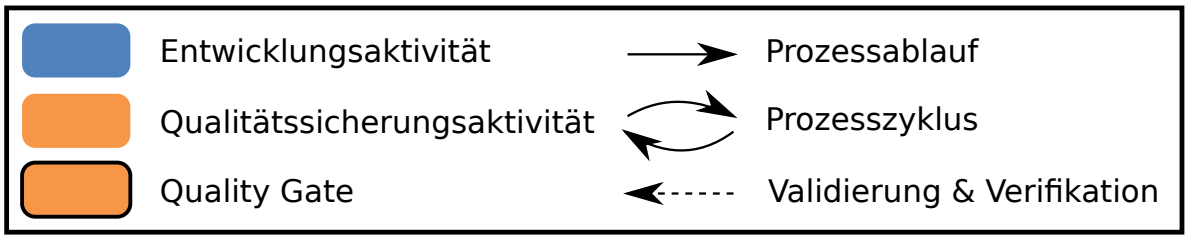

Abbildung 5.2.: Die Legende zu den Abbildungen des Qualitätsprozesses mit Entwicklungs- und Qualitätssicherungsaktivitäten, Quality Gates, dem Prozessablauf, Zyklen im Prozessablauf und der Validierung & Verifikation.

Die Tabellen mit den Entwicklungs- und Qualitätssicherungsmethoden (z. B. Tabelle 5.1) orientieren sich an den Tabellen des Sicherheitsstandards ISO 26262-6 (vgl. Abschnitt 2.1.2). Die Einträge wurde für den Qualitätsprozess übersetzt. Dabei sind die **fett ausgezeichneten Einträge** die für den Qualitätsprozess verwendeten Entwicklungs- und Qualitätssicherungsmethoden. Die Einträge der Tabellen sind als Alternativen zu verstehen, d. h. sie müssen nicht alle erfüllt werden, sondern es kann für jedes konkrete Projekt eine passende Untermenge ausgewählt werden.

5.1. Übersicht

Wie Abbildung 5.3 zeigt, orientiert sich der Qualitätsprozess in seiner Struktur am V-Modell (vgl. Abschnitt 2.2). Er besteht aus mehreren Phasen, die dem absteigenden Ast bzw. dem aufsteigenden Ast zugeordnet sind. Es gibt eine Analysephase, eine Softwarearchitekturphase, eine Modulentwurfsphase, eine Implementierungs- & Qualitätsanalysephase sowie eine Systemtestphase. Zu jeder Phase gehört ein Quality Gate, das die Phase abschließt. In den Phasen des absteigenden Astes werden die Anforderungen validiert, d. h. es wird sichergestellt, dass alle Anforderungen der vorherigen Phase erfüllt werden. Die Ergebnisse einer Phase im absteigenden Ast, werden in der entsprechenden Phase im aufsteigenden Ast verifiziert. Die Aktivitäten der Implementierungs- & Qualitätsanalysephase können mehrfach durchlaufen werden. Nach der Systemtestphase kann entweder das Projekt abgeschlossen werden oder es können in einer weiteren Iteration neue Anforderungen an die Software erhoben und umgesetzt werden.

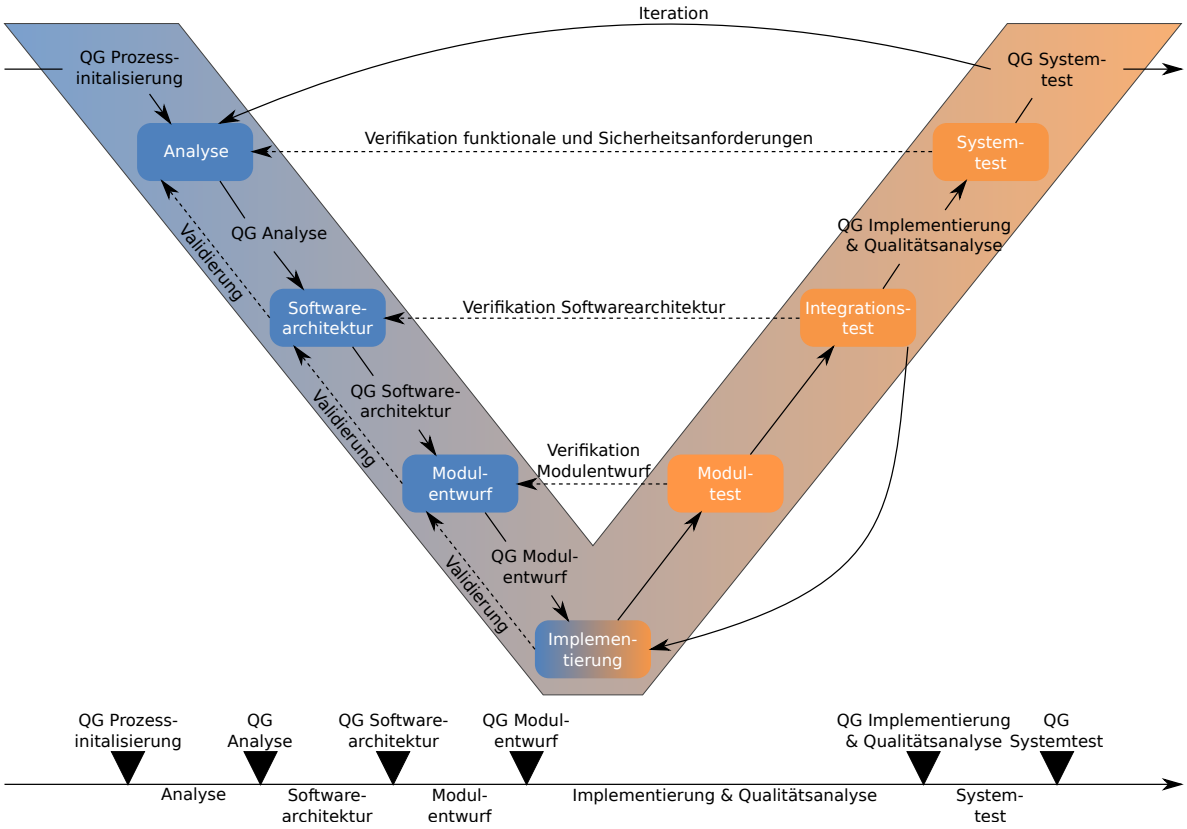

Abbildung 5.3.: Übersicht aller Phasen und Quality Gates des Qualitätsprozesses und den in den Phasen stattfindenden Entwicklungs- und Qualitätssicherungsaktivitäten. In den Entwicklungsaktivitäten werden die Anforderungen validiert. In den Qualitätssicherungsaktivitäten werden die Ergebnisse der jeweiligen Entwicklungsaktivitäten verifiziert.

Den einzelnen Phasen sind verschiedene Entwicklungs- und Qualitätssicherungsaktivitäten sowie das zur Phase gehörende Quality Gate zugeordnet. Die Entwicklungsaktivitäten beschreiben die einzelnen Entwicklungsschritte, die während einer Phase durchgeführt werden müssen sowie die dabei entstehenden Entwicklungsdokumente und Softwareartefakte. Die Qualitätssicherungsaktivitäten beschreiben die einzelnen Qualitätssicherungsmaßnahmen, die in der jeweiligen Phase zum Einsatz kommen sowie die dabei entstehenden Qualitätssicherungsdokumente. Zwischen Entwicklungsaktivitäten und den dazugehörigen Qualitätssicherungsaktivitäten findet eine Wechselwirkung statt, d. h. dass die Entwicklungsdokumente und Softwareartefakte solange überarbeitet und geprüft werden, bis es keine Befunde mehr gibt. Den einzelnen Entwicklungs- und Qualitätssicherungsaktivitäten sind Methoden zugeordnet, durch deren Anwendung die geforderte Qualität der zu entwickelnden Software sichergestellt bzw. unterstützt werden soll. Der Einsatz dieser Methoden wird durch geeignete Softwarewerkzeuge unterstützt.

5.1.1. Quality Gates

Abbildung 5.4 zeigt das Prinzip eines Quality Gates. In jedem Quality Gate ist definiert, welche Dokumente und Softwareartefakte die verifizierte Ausgabe bilden. Die Dokumente setzen sich dabei aus Entwicklungs- und Qualitätssicherungsdokumenten zusammen. Außerdem gehören die validierten Anforderungen der Phase zur Ausgabe. Die Dokumente und Softwareartefakte werden dabei nicht im Quality Gate erstellt, sondern sind das Ergebnis der Entwicklungs- und Qualitätssicherungsaktivitäten der aktuellen Phase und kommen als Eingabe in das Quality Gate. Außerdem gehören die validierten Anforderungen der vorherigen Phase und die umgesetzten Anforderungen der aktuellen Phase zur Eingabe. Die Entwicklungsdokumente und Softwareartefakte werden mit Hilfe der Qualitätssicherungsdokumente verifiziert. Die Validierung der Anforderungen erfolgt durch einen Vergleich der validierten Anforderungen der vorherigen Phase mit den umgesetzten Anforderungen der aktuellen Phase. Dadurch soll eine Verfolgung der Anforderungen erreicht werden.

In jeder Phase ist definiert, welche Dokumente und Softwareartefakte als Voraussetzung für die jeweilige Phase vorhanden sein müssen. Die Dokumente und Softwareartefakte setzen sich dabei aus den Ausgaben der Quality Gates der vorherigen Phasen zusammen. Bei jedem Dokument und Softwareartefakt wird aufgelistet in welcher Phase es erstellt wurde und ggf. in welchen Phasen es angepasst wurde.

Abbildung 5.4.: Übersicht des Prinzips eines Quality Gates. Die Entwicklungs- und Qualitätssicherungsdokumente, die Softwareartefakte, die umgesetzten Anforderungen der aktuellen Phase sowie die validierten Anforderungen der vorherigen Phase bilden die Eingabe. Die verifizierten Entwicklungsdokumente, die Softwareartefakte und die validierten Anforderungen der aktuellen Phase bilden die Ausgabe.

5.2. Prozessinitialisierung

Die Prozessinitialisierung ist eine Planungsaktivität, in der die Ausprägung des Qualitätsprozesses für das jeweilige konkrete Projekt durchgeführt wird. Für jede Phase des Qualitätsprozesses müssen aus den möglichen Entwicklungs- und Qualitätssicherungsmethoden, die für das Projekt geeigneten ausgewählt werden. Außerdem müssen die unterstützenden Softwarewerkzeuge ausgewählt werden, die die Anforderungen erfüllen. Die möglichen Methoden und die Anforderungen an die Softwarewerkzeuge wird in den jeweiligen Phasen besprochen, in denen sie hauptsächlich zum Einsatz kommen. Zu Beginn der Prozessinitialisierung müssen folgende

Dokumente und Informationen vorliegen (vgl. ISO 26262-6, Klausel 5.3), auf deren Grundlage dann die Ergebnisse erzeugt werden:

- Projektplan
- verfügbare qualifizierte Softwarewerkzeuge
- verfügbare qualifizierte Softwarekomponenten
- Richtlinien für Modellierungs- und Programmiersprachen
- Richtlinien für die Anwendung möglicher Methoden
- Richtlinien für den Einsatz möglicher Softwarewerkzeuge

Auf Grundlage der ausgewählten Methoden und Softwarewerkzeuge sowie den dazugehörigen Richtlinien muss ein Validierungs- & Verifikationsplan erstellt werden. Außerdem müssen passende Modellierungs- und Programmiersprachen sowie dazugehörige Richtlinien ausgewählt werden (vgl. ISO 26262-6, Klausel 5.4.6). Für die Modellierungs- und Programmierrichtlinien muss dabei eine Auswahl der in Tabelle 5.1 aufgezählten Themenbereiche getroffen werden, die für die jeweilige Entwicklung relevant ist. Bei der Auswahl gilt es verschiedene Aspekte zu berücksichtigen: Bei der Durchsetzung geringer Komplexität muss eventuell ein geeigneter Kompromiss mit den restlichen Anforderungen des ISO 26262-6 Standards gefunden werden (1a). Die Auswahl einer Sprachuntermenge einer Programmiersprache hat zum Ziel, uneindeutige Sprachkonstrukte, Sprachkonstrukte, die aus Erfahrung leicht zu Fehlern führen, sowie Sprachkonstrukte, die zu unbehandelten Laufzeitfehlern führen können, auszuschließen (1b). Eine starke Typisierung muss vor allem dort mit Hilfe von Richtlinien durchgesetzt werden, wo diese nicht bereits durch die Programmiersprache selbst umgesetzt ist (1c). Durch defensive Programmierung und die Verwendung etablierter Entwurfsmuster soll die Qualität und die Lesbarkeit des Programmcodes verbessert werden (1d, 1e). Eindeutige grafische Darstellungen sowie die Verwendung von Style Guides und Namenskonventionen sorgen für ein einheitliches Erscheinungsbild der zu entwickelnden Dokumente und Softwareartefakte, was ebenfalls die Lesbarkeit verbessert (1f, 1g, 1h).

5.2.1. Quality Gate Prozessinitialisierung

Die Ausgabe des Quality Gates Prozessinitialisierung sind folgende Dokumente (vgl. ISO 26262-6, Klausel 5.5):

- Validierungs- & Verifikationsplan (vgl. 5.2)
- Richtlinien für die verwendeten Modellierungs- und Programmiersprachen (vgl. 5.2)
- Richtlinien für die Anwendung der ausgewählten Methoden (vgl. 5.2)
- Richtlinien für die eingesetzten Softwarewerkzeuge (vgl. 5.2)

Tabelle 5.1.: Übersicht der Themenbereiche, die von den Modellierungs- und Programmierrichtlinien berücksichtigt werden müssen, um die Korrektheit des Entwurfs und der Implementierung zu unterstützen (vgl. ISO 26262-6, Klausel 5.4.7).

	Themenbereiche
1a	**Durchsetzung geringer Komplexität**
1b	**Verwendung von Sprachuntermengen**
1c	**Durchsetzung starker Typisierung**
1d	**Verwendung defensiver Programmierung**
1e	**Verwendung etablierter Entwurfsmuster**
1f	**Verwendung eindeutiger grafischer Darstellungen**
1g	**Verwendung von Style Guides**
1h	**Verwendung von Namenskonventionen**

5.2.2. Werkzeugunterstützung

Für die Erstellung des Validierungs- & Verifikationsplans sowie der verschiedenen Richtlinien sind keine speziellen Softwarewerkzeuge notwendig. Allerdings werden Softwarewerkzeuge für das Konfigurationsmanagement benötigt, die über die gesamte Projektdauer zum Einsatz kommen. Am wichtigsten ist hierbei ein Softwarewerkzeug für die Versionsverwaltung.

5.3. Analyse

Die Analysephase ist eine Entwicklungsphase, in der die korrekten und vollständigen Anforderungen an das neue Softwarewerkzeug erhoben, spezifiziert und geprüft werden. Außer den funktionalen Anforderungen, den Qualitätsanforderungen und den Randbedingungen müssen auch die Sicherheitsanforderungen erhoben, spezifiziert und geprüft werden. Des Weiteren müssen im Sinne des testorientierten Requirements Engineerings (vgl. Abschnitt 2.4.3) die Testfälle für den Funktions- und Sicherheitstest erstellt werden. Zu Beginn der Analysephase müssen folgende Dokumente und Informationen vorliegen (vgl. ISO 26262-6, Klausel 6.3), auf deren Grundlage dann die Ergebnisse erzeugt werden:

- Validierungs- & Verifikationsplan (vgl. 5.2)
- Richtlinien für die Anwendung der ausgewählten Methoden (vgl. 5.2)
- Richtlinien für die eingesetzten Softwarewerkzeuge (vgl. 5.2)

Abbildung 5.5 zeigt die verschiedenen Entwicklungs- und Qualitätssicherungsaktivitäten der Analysephase. Im Einzelnen sind dies die Erhebung der Anforderungen, die Spezifikation der Anforderungen, das Review der Anforderungen, die Erstellung der Testfälle für den Funktionstest, die Spezifikation der Sicherheitsanforderungen, das Review der Sicherheitsanforderungen und die Erstellung der Testfälle für den Sicherheitstest. Die Ergebnisse werden im Quality Gate der Phase verifiziert. Die Ergebnisse sind die Anforderungs- und die Sicherheitsanforderungsspezifikation sowie die Testfälle für den Funktions- und den Sicherheitstest. Die Spezifikationsdokumente müssen in einer Inspektion überprüft werden. Die Ergebnisse der Inspektion müssen dokumentiert werden und bilden den Bericht der Validierung & Verifikation. Auf Grundlage der spezifizierten Anforderungen und der erstellten Testfälle muss evtl. der Validierungs- & Verifikationsplan für die nächsten Phasen angepasst werden.

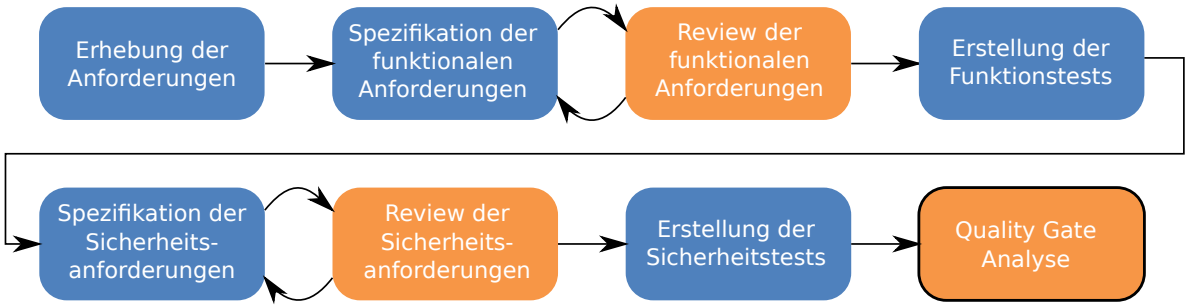

Abbildung 5.5.: Detailansicht der Analysephase mit den Entwicklungs- und Qualitätssicherungsaktivitäten Erhebung der Anforderungen, Spezifikation der Anforderungen, Review der Anforderungen, Erstellung der Testfälle für den Funktionstest, Spezifikation der Sicherheitsanforderungen, Review der Sicherheitsanforderungen und Erstellung der Testfälle für den Sicherheitstest. Abgeschlossen wird die Phase durch ein Quality Gate.

5.3.1. Quality Gate Analyse

Die Ausgabe des Quality Gates Analyse sind folgende Dokumente (vgl. ISO 26262-6, Klausel 6.5, 11.5) sowie die validierten Anforderungen:

- Anforderungsspezifikation (vgl. 5.3)
- Sicherheitsanforderungsspezifikation (vgl. 5.3)
- Testfälle für den Funktionstest (vgl. 5.3)
- Testfälle für den Sicherheitstest (vgl. 5.3)
- Validierungs- & Verifikationsplan (angepasst) (vgl. 5.3, 5.2)
- Bericht der Validierung & Verifikation (vgl. 5.3)

5.3.2. Werkzeugunterstützung

Das wichtigste Softwarewerkzeug in der Analysephase ist ein System zur Anforderungsverwaltung. Es muss die Ermittlung, Dokumentation, Analyse, Prüfung, Abstimmung und Verwaltung von Anforderungen unterstützen. Außerdem muss die horizontale und vertikale Verknüpfung von Anforderungen unterstützt werden, um die Verfolgbarkeit zu gewährleisten. Des Weiteren müssen Anforderungen sowohl über einen Status im Lebenszyklus verfügen als auch über eine Versionsnummer, um Änderungen der Anforderungen zu dokumentieren (vgl. Abschnitt 2.4.3). Für die Erstellung und Dokumentation der Testfälle für den Funktions- und den Sicherheitstest kann ein Softwarewerkzeug hilfreich sein, das die Verknüpfung der Anforderungen mit den dazugehörigen Testfällen unterstützt.

5.4. Softwarearchitektur

Die Softwarearchitekturphase ist eine Entwicklungsphase, in der aus den spezifizierten Anforderungen eine geeignete Softwarearchitektur abgeleitet werden muss. Vor allem die Qualitätsanforderungen haben einen entscheidenden Einfluss auf die Wahl der Softwarearchitektur. Ohne eine geeignete Softwarearchitektur lassen sich Qualitätsanforderungen wie Fehlertoleranz nur schwer realisieren. Für die Verfolgung der Anforderungen ist es entscheidend, dass jede Anforderung mit den Teilen der Softwarearchitektur verknüpft wird, die diese Anforderung umsetzen (vgl. Abschnitt 2.4.3). Zu Beginn der Softwarearchitekturphase müssen folgende Dokumente und Informationen vorliegen (vgl. ISO 26262-6, Klausel 7.3), auf deren Grundlage dann die Ergebnisse erzeugt werden:

- Anforderungsspezifikation (vgl. 5.3)
- Sicherheitsanforderungsspezifikation (vgl. 5.3)
- Richtlinien für Modellierungs- und Programmiersprachen (vgl. 5.2)
- Validierungs- & Verifikationsplan (angepasst) (vgl. 5.3, 5.2)
- Bericht der Validierung & Verifikation (vgl. 5.2)
- verfügbare qualifizierte Softwarekomponenten
- Richtlinien für die eingesetzten Softwarewerkzeuge (vgl. 5.2)
- Richtlinien für die Anwendung der ausgewählten Methoden (vgl. 5.2)

Abbildung 5.6 zeigt die verschiedenen Entwicklungs- und Qualitätssicherungsaktivitäten der Softwarearchitekturphase. Im Einzelnen sind dies die Erstellung der Softwarearchitektur, das Review der Softwarearchitektur und die Erstellung der Integrationstests. Die Ergebnisse werden im Quality Gate der Phase verifiziert. Die Ergebnisse sind die Softwarearchitektur und die Testfälle für den Integrationstest.

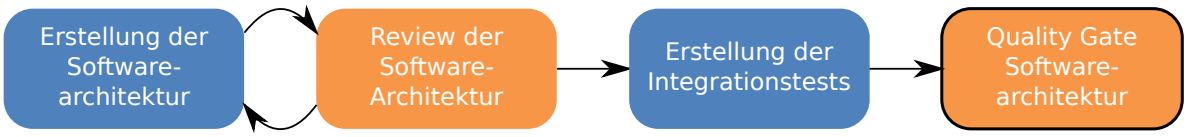

Abbildung 5.6.: Detailansicht der Softwarearchitekturphase mit den Entwicklungs- und Qualitätssicherungsaktivitäten Erstellung der Softwarearchitektur, Review der Softwarearchitektur und Erstellung der Integrationstests. Abgeschlossen wird die Phase durch ein Quality Gate.

Tabelle 5.2.: Mögliche Notationen für die Dokumentation der Softwarearchitektur (vgl. ISO 26262-6, Klausel 7.4.1).

	Methoden
2a	**Informelle Notation**
2b	**Semi-formale Notation**
2c	**Formale Notation**

Für die Dokumentation der Softwarearchitektur gibt es verschiedene Notationen (vgl. Tabelle 5.2). Dabei kann keine der Notationen pauschal empfohlen werden, sondern es ist wichtig, dass die jeweils passende Notation für die Dokumentation der jeweiligen Aspekte der Softwarearchitektur verwendet wird. Als Richtlinie kann dabei gelten:

> So formal wie nötig und so informell wie möglich.

Bei der Erstellung der Softwarearchitektur gilt es auf verschiedene Prinzipien zu achten, um eine hohe Qualität der Software zu erreichen (vgl. Tabelle 5.3). Es ist wichtig hierarchische Strukturen zu schaffen, d. h. die Software in einzelne Module aufzuteilen und deren Beziehungen zueinander klar zu strukturieren (3a). Die Module dürfen dabei nicht zu groß werden und nicht über zu große Schnittstellen verfügen (3b, 3c). Um dies zu erreichen, muss auf einen hohen Zusammenhalt innerhalb jedes Moduls geachtet werden (3d). Außerdem muss die Koppelung der Module untereinander so gering wie möglich sein (3e). Bei Softwarewerkzeugen spielen Scheduling-Eigenschaften und Interrupts, im Gegensatz zu eingebetteter Software, keine große Rolle (3f, 3g).

Um das Auftreten von Fehlern und ihre Auswirkungen im Falle eines Auftretens zu reduzieren, können verschiedene Maßnahmen getroffen werden. Die Fehlerentdeckung kann bereits auf Softwarearchitekturebene durch verschiedene Methoden unterstützt werden (vgl. Tabelle 5.4). Der Bereich der Ein- und Ausgabedaten kann überprüft werden, um Fehler abzufangen (4a), die Plausibilität von Daten kann durch Assertions überprüft werden, um mögliche Fehler aufzudecken (4b) und es kann versucht werden Datenfehler zu erkennen, indem entweder Fehlererkennungscodes verwendet oder Daten mehrfach gespeichert werden (4c). Externe

Tabelle 5.3.: Prinzipien für den Softwarearchitekturentwurf (vgl. ISO 26262-6, Klausel 7.4.3).

	Methoden
3a	**Hierarchische Struktur der Softwaremodule**
3b	**Eingeschränkte Größe der Softwaremodule**
3c	**Eingeschränkte Größe der Schnittstellen**
3d	**Hoher Zusammenhalt innerhalb jedes Softwaremoduls**
3e	**Eingeschränkte Koppelung zwischen Softwaremodulen**
3f	Angemessene Scheduling-Eigenschaften
3g	Eingeschränkte Verwendung von Interrupts

Tabelle 5.4.: Mechanismen für die Fehlerentdeckung auf Softwarearchitekturebene (vgl. ISO 26262-6, Klausel 7.4.14).

	Methoden
4a	**Bereichsprüfung von Eingabe- und Ausgabedaten**
4b	**Plausibilitätsprüfung**
4c	**Erkennung von Datenfehlern**
4d	Externe Überwachungsmöglichkeiten
4e	Überwachung des Kontrollflusses
4f	Diverser Softwareentwurf

Überwachungsmöglichkeiten spielen bei der Entwicklung von Softwarewerkzeugen keine wichtige Rolle. Es reicht, wenn bei der Wahl der Entwicklungsumgebung auf eine leistungsstarke Debugging-Umgebung geachtet wird (4d). Die Überwachung bzw. Steuerung des Kontrollflusses ist bereits ausreichend durch das Einhalten der Konzepte der geringen Koppelung und des hohen Zusammenhalts gewährleistet (4e). Ein diverser Softwareentwurf ist wiederum bei nicht eingebetteten Softwaresystemen nicht notwendig (4f).

Außerdem kann die Fehlerbehandlung durch verschiedene Methoden verbessert werden (vgl. Tabelle 5.5). Es können statische Wiederherstellungsmechanismen eingebaut werden, die im Fehlerfall einen definierten Zustand des Systems wieder herstellen (5a). Außerdem kann die Fehlertoleranz des Systems erhöht werden, um die Auswirkungen von Fehlern einzuschränken (5b). Unabhängige parallele Redundanz ist bei nicht eingebetteten Softwaresystem nicht

Tabelle 5.5.: Mechanismen für die Fehlerbehandlung auf Softwarearchitekturebene (vgl. ISO 26262-6, Klausel 7.4.15).

	Methoden
5a	**Statische Wiederherstellungsmechanismen**
5b	**Fehlertoleranz**
5c	Unabhängige parallele Redundanz
5d	Korrekturcodes für Daten

notwendig und Korrekturcodes für Daten höchstens, wenn Daten auch über ein Netzwerk übertragen werden (5c, 5d).

Die Softwarearchitektur muss anschließend mit passenden Methoden geprüft werden (vgl. Tabelle 5.6). Die Softwarearchitektur sollte entweder durch ein Walk-through oder durch eine Inspektion geprüft werden (6a, 6b). Unter Umständen kann auch die Generierung eines Prototypen Sinn ergeben, um die Umsetzbarkeit zu untersuchen und mögliche Schwierigkeiten frühzeitig zu identifizieren (6d). Die Simulation dynamischer Komponenten der Softwarearchitektur setzt die Verwendung von ausführbaren Modellen voraus. Wird also der Ansatz des Model-Driven Developments verfolgt, ist auch dies eine mögliche Methode. In allen anderen Fällen lohnt sich der extra Aufwand für das Erstellen eines solchen Modells nicht (6c). Auch die formale Verfikation setzt eine Softwarearchitektur voraus, die formal verifiziert werden kann. Dies ist mit einem erheblichen Mehraufwand verbunden, der sich für die Entwicklung von Softwarewerkzeugen im Allgemeinen nicht lohnt (6e). Die Kontrollfluss- und Datenflussanalyse ist bei nicht eingebetteten Systemen zur Verifikation auf Softwarearchitekturebene nicht notwendig (6f, 6g).

Die Ergebnisse der Prüfung der Softwarearchitektur müssen im Bericht zur Validierung & Verifikation dokumentiert werden.

Auf Basis der geprüften Softwarearchitektur müssen abschließend die Testfälle für den Integrationstest erstellt werden, um auch hier das testorientierte Requirements Engineering umzusetzen. Dabei müssen zuerst die verschiedenen Arten von Tests betrachtet werden (vgl. Tabelle 5.7). Der Integrationstest kann sowohl als funktionsbasierter Test als auch als Schnittstellentest durchgeführt werden (7a, 7b). Der Ressourcentest wird auf der Systemtestebene durchgeführt (7d). Ein fault-injection-Test ist bei nicht eingebetteter Software nicht sinnvoll (7c). Der Vergleich zwischen Modell und Code setzt wieder ein Modell der Softwarearchitektur voraus, kann also eingesetzt werden, wenn der Ansatz des Model-driven Developments verfolgt wird (7e).

Auch für die Erstellung der konkreten Testfälle gibt es verschiedene Ansätze, um möglichst gute Testfälle zu generieren und eine gute Testabdeckung zu erreichen (vgl. Tabelle 5.8). Neben

Tabelle 5.6.: Methoden für die Prüfung der Softwarearchitektur (vgl. ISO 26262-6, Klausel 7.4.18).

	Methoden
6a	**Walk-through der Softwarearchitektur**
6b	**Inspektion der Softwarearchitektur**
6c	Simulation dynamischer Komponenten der Softwarearchitektur
6d	**Generierung eines Prototypen**
6e	Formale Verifikation
6f	Kontrollflussanalyse
6g	Datenflussanalyse

Tabelle 5.7.: Methoden für den Integrationstest (vgl. ISO 26262-6, Klausel 10.4.3).

	Methoden
7a	**Funktionsbasierter Test**
7b	**Schnittstellentest**
7c	Fault-injection-Test
7d	Ressourcentest
7e	Back-to-back Vergleich zwischen Modell und Code, falls anwendbar

der Analyse der Anforderungen, um sämtliche Anforderungen abzudecken (8a), können auch Äquivalenzklassen erstellt werden (8b) und eine Grenzwertbetrachtung durchgeführt werden (8c). Die Methode der Fehlererwartung findet auf der Systemtestebene Anwendung (8d).

Auf Grundlage der spezifizierten Softwarearchitektur und der erstellten Testfälle für den Integrationstest muss evtl. der Validierungs- & Verifikationsplan angepasst werden. Auch muss evtl. die Sicherheitsanforderungsspezifikation an die Softwarearchitektur angepasst werden.

Tabelle 5.8.: Methoden für die Erstellung von Testfällen für den Integrationstest (vgl. ISO 26262-6, Klausel 10.4.4).

	Methoden
8a	**Analyse der Anforderungen**
8b	**Erstellung und Analyse von Äquivalenzklassen**
8c	**Grenzwertbetrachtung**
8d	Fehlererwartung

5.4.1. Quality Gate Softwarearchitektur

Die Ausgabe des Quality Gates Softwarearchitektur sind folgende Dokumente (vgl. ISO 26262-6, Klausel 7.5, 10.5) sowie die validierten Anforderungen:

- Softwarearchitekturspezifikation (vgl. 5.4)
- Testfälle für den Integrationstest (vgl. 5.4)
- Sicherheitsanforderungsspezifikation (angepasst) (vgl. 5.4, 5.3)
- Validierungs- & Verifikationsplan (angepasst) (vgl. 5.4, 5.3, 5.2)
- Bericht der Validierung & Verifikation (angepasst) (vgl. 5.4, 5.3)

5.4.2. Werkzeugunterstützung

Für die Erstellung der Softwarearchitektur sind Softwarewerkzeuge für die Erstellung von z. B. UML-Diagrammen hilfreich. Wird der Ansatz des Model-driven Developments verfolgt, ist eine Unterstützung für die Generierung von Programmcode aus den Modellen sinnvoll. Für die Erstellung und Dokumentation der Testfälle für den Integrationstest kann ein Softwarewerkzeug hilfreich sein, das die Verknüpfung der Anforderungen mit den dazugehörigen Testfällen unterstützt.

5.5. Modulentwurf

Die Modulentwurfsphase ist eine Entwicklungsphase, in der auf Grundlage der Softwarearchitektur die einzelnen Module entworfen werden müssen. Dabei werden die funktionalen Anforderungen und die Sicherheitsanforderungen in den einzelnen Modulen umgesetzt. Für die Verfolgung der Anforderungen ist es entscheidend, dass jede Anforderung mit dem Modul

verknüpft wird, das diese Anforderung umsetzt (vgl. Abschnitt 2.4.3). Zu Beginn der Modulentwurfsphase müssen folgende Dokumente und Informationen vorliegen (vgl. ISO 26262-6, Klausel 8.3), auf deren Grundlage dann die Ergebnisse erzeugt werden:

- Softwarearchitekturspezifikation (vgl. 5.4)
- Anforderungsspezifikation (vgl. 5.3)
- Sicherheitsanforderungsspezifikation (angepasst) (vgl. 5.4, 5.3)
- Richtlinien für Modellierungs- und Programmiersprachen (vgl. 5.2)
- Validierungs- & Verifikationsplan (angepasst) (vgl. 5.4, 5.3, 5.2)
- Bericht der Validierung & Verifikation (angepasst) (vgl. 5.4, 5.3)
- Richtlinien für die eingesetzten Softwarewerkzeuge (vgl. 5.2)
- Richtlinien für die Anwendung der ausgewählten Methoden (vgl. 5.2)

Abbildung 5.7 zeigt die verschiedenen Entwicklungs- und Qualitätssicherungsaktivitäten der Modulentwurfsphase. Im Einzelnen sind dies die Erstellung des Modulentwurfs, das Review des Modulentwurfs und die Erstellung der Modultests. Die Ergebnisse werden im Quality Gate der Phase verifiziert. Die Ergebnisse der Modulentwurfsphase sind der Modulentwurf und die Testfälle für den Modultest.

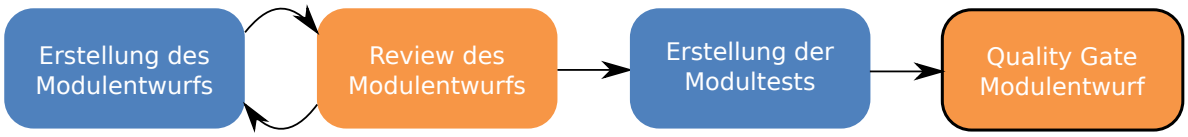

Abbildung 5.7.: Detailansicht der Modulentwurfsphase mit den Entwicklungs- und Qualitätssicherungsaktivitäten Erstellung des Modulentwurfs, Review der Modulentwurfs und Erstellung der Modultests. Abgeschlossen wird die Phase durch ein Quality Gate.

Für die Dokumentation des Modulentwurfs gibt es verschiedene Notationen (vgl. Tabelle 5.9). Dabei kann keine der Notationen pauschal empfohlen werden, sondern es ist wichtig, dass die jeweils passende Notation für die Dokumentation der jeweiligen Aspekte des Modulentwurfs verwendet wird. Als Richtlinie kann dabei wieder gelten:

> So formal wie nötig und so informell wie möglich.

Der Modulentwurf muss anschließend mit passenden Methoden geprüft werden (vgl. Tabelle 5.10). Der Modulentwurf sollte entweder durch ein Walk-through oder durch eine Inspektion geprüft werden (10a, 10b). Eine semi-formale oder gar formale Verifikation des Modulentwurfs

Tabelle 5.9.: Notationen für den Modulentwurf (vgl. ISO 26262-6, Klausel 8.4.2).

	Methoden
9a	**Natürliche Sprache**
9b	**Informelle Notation**
9c	**Semi-formale Notation**
9d	**Formale Notation**

Tabelle 5.10.: Methoden für die Prüfung des Modulentwurfs (vgl. ISO 26262-6, Klausel 8.4.5).

	Methoden
10a	**Walk-through**
10b	**Inspektion**
10c	Semi-formale Verifikation
10d	Formale Verifikation

setzt wieder einen formal verifizierbaren Modulentwurf voraus, was mit erheblichem Mehraufwand verbunden ist, der für die Entwicklung von Softwarewerkzeugen nicht gerechtfertigt ist (10c, 10d).

Die Ergebnisse der Prüfung des Modulentwurfs müssen im Bericht zur Validierung & Verifikation dokumentiert werden.

Auf Basis des geprüften Modulentwurfs müssen abschließend die Testfälle für den Modultest erstellt werden, um auch hier das Konzept des testorientierten Requirements Engineerings umzusetzen. Dabei müssen zuerst die verschiedenen Arten von Tests betrachtet werden (vgl. Tabelle 5.7). Der Modultest kann sowohl als funktionsbasierter Test als auch als Schnittstellentest durchgeführt werden (11a, 11b). Der Ressourcentest wird auf der Systemtestebene durchgeführt (11d). Ein fault-injection-Test ist zwar sinnvoll, aber mit einem erheblichen Mehraufwand verbunden, der sich bei nicht eingebetteten Systemen im Allgemeinen nicht lohnt (11c). Der Vergleich zwischen Modell und Programmcode setzt ein Modell der Softwarearchitektur voraus, kann also eingesetzt werden, wenn der Ansatz des Model-driven Developments verfolgt wird (11e).

Auch für die Erstellung der konkreten Testfälle gibt es verschiedene Ansätze, um möglichst gute Testfälle zu generieren und eine gute Testabdeckung zu erreichen (vgl. Tabelle 5.12). Neben der Analyse der Anforderungen, um sämtliche Anforderungen abzudecken (12a), können auch

Tabelle 5.11.: Methoden für den Modultest (vgl. ISO 26262-6, Klausel 9.4.3).

	Methoden
11a	**Funktionsbasierter Test**
11b	**Schnittstellentest**
11c	Fault-injection-Test
11d	Ressourcentest
11e	Back-to-back Vergleich zwischen Modell und Code, falls anwendbar

Tabelle 5.12.: Methoden für die Erstellung von Testfällen für den Modultest (vgl. ISO 26262-6, Klausel 9.4.4).

	Methoden
12a	**Analyse der Anforderungen**
12b	**Erstellung und Analyse von Äquivalenzklassen**
12c	**Grenzwertbetrachtung**
12d	Fehlererwartung

Äquivalenzklassen erstellt werden (12b) und eine Grenzwertbetrachtung durchgeführt werden (12c). Die Methode der Fehlererwartung findet auf der Systemtestebene Anwendung (12d).

Auf Grundlage des spezifizierten Modulentwurfs und der erstellten Testfälle für den Modultest muss evtl. der Validierungs- & Verifikationsplan angepasst werden.

5.5.1. Quality Gate Modulentwurf

Die Ausgabe des Quality Gates Modulentwurf sind folgende Dokumente (vgl. ISO 26262-6, Klausel 8.5, 9.5) sowie die validierten Anforderungen:

- Modulentwurfsspezifikation (vgl. 5.5)
- Testfälle für den Modultest (vgl. 5.5)
- Validierungs- & Verifikationsplan (angepasst) (vgl. 5.5, 5.4, 5.3, 5.2)
- Bericht der Validierung & Verifikation (angepasst) (vgl. 5.5, 5.4, 5.3)

5.5.2. Werkzeugunterstützung

Für den Modulentwurf sind Softwarewerkzeuge für die Erstellung von z. B. UML-Diagrammen hilfreich. Wird der Ansatz des Model-driven Developments verfolgt, ist eine Unterstützung für die Generierung von Programmcode aus den Modellen sinnvoll. Für die Erstellung und Dokumentation der Testfälle für den Modultest kann ein Softwarewerkzeug hilfreich sein, das die Verknüpfung der Anforderungen mit den dazugehörigen Testfällen unterstützt.

5.6. Implementierung & Qualitätsanalyse

Die Implementierungs- & Qualitätsanalysephase ist eine Entwicklungs- und Qualitätssicherungsphase, in der iterativ die in der Anforderungs- und Sicherheitsanforderungsspezifikation spezifizierten und im Modulentwurf modellierten Anforderungen in Programmcode umgesetzt und getestet werden. Die Softwarearchitektur dient dabei als Rahmen für die Implementierung der einzelnen Module. Durch den Implementierungs- & Qualitätsanalysezyklus soll eine kontinuierliche Implementierung, Prüfung und Integration der Module sichergestellt werden. Dadurch sollen Probleme und Fehler frühzeitig erkannt und behoben werden. Zu Beginn der Implementierungs- & Qualitätsanalysephase müssen folgende Dokumente und Informationen vorliegen (vgl. ISO 26262-6, Klausel 8.3, 9.3, 10.3), auf deren Grundlage dann die Ergebnisse erzeugt werden:

- Anforderungsspezifikation (vgl. 5.3)
- Sicherheitsanforderungsspezifikation (angepasst) (vgl. 5.3, 5.4)
- Softwarearchitekturspezifikation (vgl. 5.4)
- Modulentwurfsspezifikation (vgl. 5.5)
- Testfälle für den Modultest (vgl. 5.5)
- Testfälle für den Integrationstest (vgl. 5.4)
- Richtlinien für Modellierungs- und Programmiersprachen (vgl. 5.2)
- Validierungs- & Verifikationsplan (angepasst) (vgl. 5.5, 5.4, 5.3, 5.2)
- Bericht der Validierung & Verifikation (angepasst) (vgl. 5.5, 5.4, 5.3)
- verfügbare qualifizierte Softwarekomponenten
- Bericht der Softwarewerkzeugqualifizierung
- Richtlinien für die eingesetzten Softwarewerkzeuge (vgl. 5.2)
- Richtlinien für die Anwendung der ausgewählten Methoden (vgl. 5.2)

Wie Abbildung 5.8 zeigt, besteht die Implementierungs- & Qualitätsanalysephase aus einem Implementierungs- & Qualitätsanalysezyklus sowie dem zur Phase gehörenden Quality Gate. Abbildung 5.9 zeigt die einzelnen Entwicklungs- und Qualitätssicherungsaktivitäten des Zyklus. Im Einzelnen sind dies Implementierung / Korrektur, Statische Codeanalyse, Modultests, Integrationstests und Überdeckungsmetriken. Die Ergebnisse des Zyklus werden im Quality Gate der Phase verifiziert. Die Ergebnisse der Implementierungs- & Qualitätsanalysephase sind die einzelnen implementierten und getesteten Module sowie die getestete Integration aller Module.

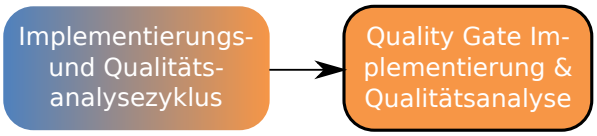

Abbildung 5.8.: Detailansicht der Implementierungs- & Qualitätsanalysephase mit dem Implementierungs- & Qualitätsanalysezyklus. Abgeschlossen wird die Phase durch ein Quality Gate.

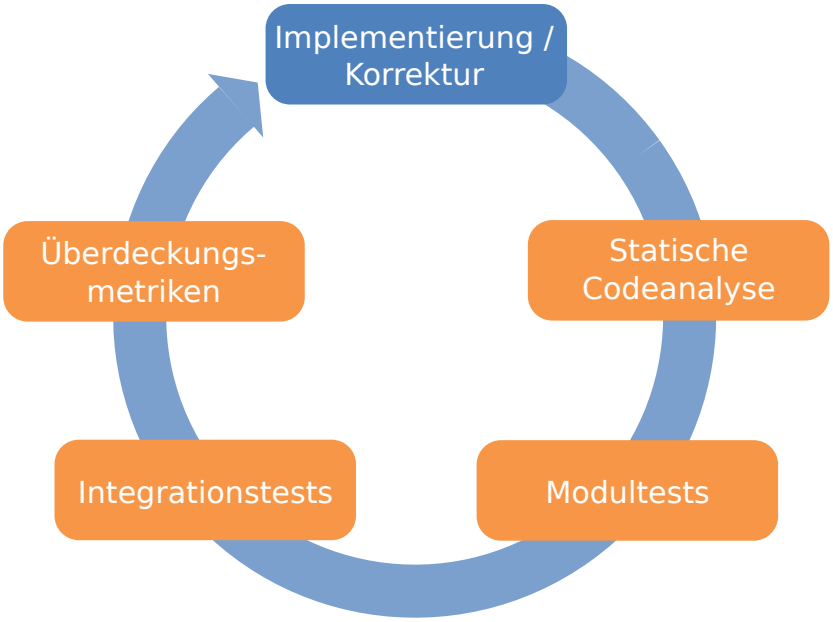

Abbildung 5.9.: Detailansicht des Implementierungs- & Qualitätsanalysezyklus mit den Entwicklungs- und Qualitätssicherungsaktivitäten Implementierung / Korrektur, Statische Codeanalyse, Modultests, Integrationstests und Überdeckungsmetriken.

Zu Beginn des Zyklus wird eine funktionale Anforderung eines Moduls in der Entwicklungsaktivität Implementierung in Programmcode umgesetzt und der Versionsverwaltung unterstellt. Alternativ wird eine Korrektur des bestehenden Programmcodes durchgeführt. Als erstes wird

Tabelle 5.13.: Prinzipien für die Implementierung (vgl. ISO 26262-6, Klausel 8.4.8).

	Methoden
13a	**Ein Eintritts- und Austrittspunkt in Unterprogrammen und Funktionen**
13b	**Keine dynamischen Objekte und Variablen oder Test während ihrer Erzeugung**
13c	**Initialisierung von Variablen**
13d	**Keine Mehrfachverwendung von Variablennamen**
13e	**Vermeidung globaler Variablen oder Begründung ihrer Verwendung**
13f	**Eingeschränkte Verwendung von Zeigern**
13g	**Keine impliziten Typumwandlungen**
13h	**Kein versteckter Daten- oder Kontrollfluss**
13i	**Keine unbedingten Sprünge**
13j	**Keine Rekursion**

der Programmcode kompiliert. Ist die Kompilierung erfolgreich, folgt eine Reihe von Qualitätssicherungsaktivitäten. Zuerst wird eine statische Codeanalyse durchgeführt. Als nächstes wird durch den Modultest die Funktionalität des Moduls geprüft. Dann werden die Module integriert und durch den Integrationstest die Schnittstellen der Module getestet. Als letztes werden die Überdeckungsmetriken erhoben, um die Testabdeckung zu erfassen. Nach der Qualitätsanalyse werden die gefundenen Fehler in der Entwicklungsaktivität Korrektur behoben und die Qualitätsanalyse erneut durchlaufen. Wenn keine neuen Fehler mehr gefunden werden, kann eine neue funktionale Anforderung umgesetzt werden.

Um eine hohe Qualität des Programmcodes zu erreichen, müssen verschiedene Prinzipien bei der Implementierung der Module beachtet werden (vgl. Tabelle 5.13). Dabei müssen die Prinzipien nur für die Module befolgt werden, die sicherheitsrelevant sind (vgl. ISO 26262-6, Klausel 8.4.1). Sicherheitsrelevant bedeutet, dass die Module Sicherheitsanforderungen umsetzen. Allerdings sind diese Anforderungen auch für die Implementierung aller anderen Module zu empfehlen. Das Ziel aller aufgezählten Prinzipien ist es, einen leserlichen und verständlichen Programmcode zu erreichen.

Die statische Codeanalyse ist ein wichtiger Bestandteil der Qualitätsanalyse (vgl. Tabelle 5.14). Kontrollfluss- und Datenflussanalysen sind ein Teil der statischen Codeanalyse und müssen deswegen nicht gesondert betrachtet werden (14a, 14b). Für die statische Codeanalyse ist wichtig, dass ein zur verwendeten Programmiersprache passendes Softwarewerkzeug ausgewählt wird. Sinnvoll ist es außerdem, die verwendeten Richtlinien und Style Guides für die Programmiersprache durch die statische Codeanalyse auf ihre Einhaltung zu überprüfen (14c).

Tabelle 5.14.: Methoden für die Verifikation der Implementierung (vgl. ISO 26262-6, Klausel 8.4.5).

	Methoden
14a	**Kontrollflussanalyse**
14b	**Datenflussanalyse**
14c	**Statische Codeanalyse**
14d	Semantische Codeanalyse

Tabelle 5.15.: Strukturelle Überdeckungsmetriken auf Modulebene und Softwarearchitekturebene (vgl. ISO 26262-6, Klausel 9.4.5, 10.4.6).

	Methoden
15a	**Anweisungsüberdeckung**
15b	**Zweigüberdeckung**
15c	**MC/DC (Modified Condition/Decision Coverage)**
15d	**Funktionsüberdeckung**
15e	**Aufrufüberdeckung**

Die Überdeckungsmetriken helfen dabei, eine ausreichende Testabdeckung zu erreichen (vgl. Tabelle 5.15). Dabei sollten sowohl für den Modultest als auch für den Integrationstest ausreichend hohe Werte für die Abdeckung durch die einzelnen Metriken gewählt werden. Ist die Abdeckung zu gering, müssen die Testfälle überarbeitet bzw. durch neue Testfälle ergänzt werden.

Die Ergebnisse der Qualitätsanalyse müssen im Bericht zur Validierung & Verifikation dokumentiert werden.

5.6.1. Quality Gate Implementierung & Qualitätsanalyse

Die Ausgabe des Quality Gates Implementierung & Qualitätsanalyse sind folgende Dokumente und Softwareartefakte (vgl. ISO 26262-6, Klausel 8.5, 10.5) sowie die validierten Anforderungen:

- Getestete Implementierung der Module (vgl. 5.6)
- Integrierte Software (vgl. 5.6)
- Bericht der Validierung & Verifikation (angepasst) (vgl. 5.6, 5.5, 5.4, 5.3)

5.6.2. Werkzeugunterstützung

Die wichtigsten Softwarewerkzeuge für die Implementierung & Qualitätsanalyse sind eine geeignete Entwicklungsumgebung (IDE), ein Continuous Integration-Server, ein Softwarewerkzeug zur statischen Codeanalyse und ein Framework für Unit-Tests.

5.7. Systemtest

Die Systemtestphase ist eine Qualitätssicherungsphase, in der die integrierte Software anhand der vorher erstellten Testfälle für den Funktions- und Sicherheitstest getestet wird. Anschließend müssen alle Befunde korrigiert werden und noch einmal alle Testfälle durchlaufen werden. Dies muss solange wiederholt werden, bis keine Befunde mehr auftreten. Zu Beginn der Systemtestphase müssen folgende Dokumente, Softwareartefakte und Informationen vorliegen (vgl. ISO 26262-6, Klausel 11.3), auf deren Grundlage dann die Ergebnisse erzeugt werden:

- Integrierte Software (vgl. 5.6)
- Anforderungsspezifikation (vgl. 5.3)
- Sicherheitsanforderungsspezifikation (angepasst) (vgl. 5.4, 5.3)
- Softwarearchitekturspezifikation (vgl. 5.4)
- Testfälle für den Funktionstest (vgl. 5.3)
- Testfälle für den Sicherheitstest (vgl. 5.3)
- Validierungs- & Verifikationsplan (angepasst) (vgl. 5.5, 5.4, 5.3, 5.2)
- Bericht der Validierung & Verifikation (angepasst) (vgl. 5.6, 5.5, 5.4, 5.3)
- Richtlinien für die eingesetzten Softwarewerkzeuge (vgl. 5.2)
- Richtlinien für die Anwendung der ausgewählten Methoden (vgl. 5.2)

Abbildung 5.10 zeigt die verschiedenen Entwicklungs- und Qualitätssicherungsaktivitäten der Systemtestphase. Im Einzelnen sind dies der Funktionstest, die Korrektur der Befunde des Funktionstests, der Sicherheitstest und die Korrektur der Befunde des Sicherheitstests. Die Ergebnisse werden im Quality Gate der Phase verifiziert. Das Ergebnis der Systemtestphase ist das auf funktionale und Sicherheitsanforderungen getestete Softwarewerkzeug.

Abbildung 5.10.: Detailansicht der Systemtestphase mit den Entwicklungs- und Qualitätssicherungsaktivitäten Funktionstest, Korrektur der Befunde des Funktionstest, Sicherheitstest und Korrektur der Befunde des Sicherheitstests. Abgeschlossen wird die Phase durch ein Quality Gate.

Die Ergebnisse des Systemtests müssen im Bericht zur Validierung & Verifikation dokumentiert werden.

5.7.1. Quality Gate Systemtest

Die Ausgabe des Quality Gates Systemtest sind folgende Dokumente und Softwareartefakte (vgl. ISO 26262-6, Klausel 11.5) sowie die validierten Anforderungen:

- Getestete Software (vgl. 5.7, 5.6)
- Bericht der Validierung & Verifikation (angepasst) (vgl. 5.7, 5.6, 5.5, 5.4, 5.3)

5.7.2. Werkzeugunterstützung

Bei der Durchführung des Systemtests gibt es keine besonderen Anforderungen an eingesetzte Softwarewerkzeuge.

6. Expertenreview

In diesem Kapitel wird der in Kapitel 4 konzipierte und in Kapitel 5 ausgearbeitete Qualitätsprozess evaluiert. Die Evaluation erfolgte durch ein Expertenreview. Bei einem Expertenreview handelt es sich um eine Evaluation eines Prüflings durch Gutachter, die auf den für den Prüfling relevanten Gebieten über gute Kenntnisse verfügen. Der Prüfling wird dabei anhand vorgegebener Kriterien begutachtet und bewertet sowie evtl. vorhandene Mängel und mögliche Verbesserungen benannt. Im Folgenden wird zuerst die Vorbereitung des Expertenreviews ausgeführt, anschließend folgt die Beschreibung der Durchführung und zum Schluss die Auswertung.

6.1. Vorbereitung

Die Vorbereitung bestand aus der Überlegung, welche Aspekte bei der Prüfung wichtig sind. Darüber hinaus wurde die Form und der Inhalt der zu erstellenden Gutachten festgelegt. Sowohl die Prüfaspekte als auch die Form des Gutachtens wurden in einem Informationsdokument für die Gutachter niedergeschrieben (vgl. Anhang A). Schließlich wurden anhand der Prüfaspekte passende Gutachter ausgewählt.

6.1.1. Prüfaspekte

Der Qualitätsprozess sollte unter folgenden Aspekten geprüft werden:

- Übereinstimmung des Qualitätsprozesses mit dem Sicherheitsstandard ISO 26262
- Evaluierbarkeit des Qualitätsprozesses durch Automotive SPICE/CMMI
- Konsistenz und Vollständigkeit des Qualitätsprozesses
- Umsetzbarkeit des Qualitätsprozesses in der Praxis

Außerdem sollten noch folgende Aspekte bei der Prüfung berücksichtigt werden:

- Verständlichkeit der Beschreibung und Abbildungen des Qualitätsprozesses
- Korrektheit der Beschreibung und Abbildungen des Qualitätsprozesses

Übereinstimmung mit ISO 26262. Die Gutachter sollten den Qualitätsprozess auf Umsetzung und Einhaltung des Sicherheitsstandards ISO 26262 überprüfen und bewerten. Dabei war vor allem der Abschnitt des Sicherheitsstandards zur Softwareentwicklung von Bedeutung (vgl. ISO 26262-6).

Evaluierbarkeit durch Automotive SPICE/CMMI. Die Gutachter sollten die mögliche Evaluierbarkeit des Qualitätsprozesses durch Automotive SPICE/CMMI prüfen und bewerten.

Konsistenz und Vollständigkeit. Die Gutachter sollten den Qualitätsprozess auf seine Konsistenz und Vollständigkeit überprüfen und bewerten. Vollständigkeit bedeutet in diesem Kontext, dass der Qualitätsprozess den kompletten Softwareentwicklungszyklus abdeckt.

Umsetzbarkeit in der Praxis. Die Gutachter sollten die Umsetzbarkeit des Qualitätsprozesses in der Praxis prüfen und bewerten. Dies bedeutet, dass mit dem Prozess im industriellen Kontext realistische Softwarewerkzeuge entwickelt werden können.

6.1.2. Form des Gutachtens

Das Gutachten wurde in schriftlicher Form durchgeführt, d. h. jeder Gutachter hielt seine Befunde und Einschätzungen in einem Dokument fest. Das Dokument sollte dabei folgende Informationen enthalten:

- den Namen des Gutachters,
- eine Einschätzung der eigenen Expertise in den relevanten Themen,
- eine Liste aller Befunde und
- eine allgemeine Einschätzung des Qualitätsprozesses.

Expertise des Gutachters. Jeder Gutachter sollte seine Expertise in den Themen „Sicherheitsstandard ISO 26262“, „Automotive SPICE/CMMI“ und „Softwareentwicklungsprozesse“ auf einer Skala von 1 bis 5 einschätzen, bei der 1 gering und 5 hoch bedeutet. Dadurch sollten die Befunde und die allgemeine Einschätzung des Qualitätsprozesses der einzelnen Gutachter besser eingeordnet werden können.

Liste der Befunde. Jeder Gutachter sollte seine Befunde auflisten. Jeder Befund sollte dabei eine kurze Beschreibung des Befunds(z. B. in Stichworten), eine Referenz auf die betroffenen Bereiche des Qualitätsprozesses (z. B. Abschnitt, Abbildung oder Tabelle) und eine Bewertung der Schwere des Befunds beinhalten (kritischer Fehler, Hauptfehler oder Nebenfehler). Um einen kritischen Fehler handelt es sich, wenn der Qualitätsprozess dadurch nicht eingesetzt werden kann. Ein Hauptfehler beschreibt ein Problem, dass den Einsatz des Qualitätsprozesses erheblich erschwert. Ein Nebenfehler betrifft zwar nicht den Einsatz des Qualitätsprozesses direkt, aber der Qualitätsprozess wird dadurch zumindest schwerer zu verstehen.

Allgemeine Einschätzung des Qualitätsprozesses. Jeder Gutachter sollte eine allgemeine Einschätzung des Qualitätsprozesses, unter Berücksichtigung der Prüfaspekte, abgeben. Dabei sollte es sich um einen über die bloßen Befunde hinausgehenden Gesamteindruck des Qualitätsprozesses handeln.

6.1.3. Auswahl der Gutachter

Die Auswahl der Gutachter sollte anhand der Prüfaspekte erfolgen. Dabei ist es nicht notwendig, dass sich jeder Gutachter in allen Prüfaspekten auskennt. Außerdem sollte darauf geachtet werden, dass sowohl Gutachter aus dem akademischen Bereich als auch aus der Industrie ausgewählt werden. Um eine ausreichende Relevanz der Ergebnisse des Expertenreviews zu gewährleisten, sollten drei bis fünf Gutachter ausgewählt werden.

6.2. Durchführung

Es wurden fünf Gutachter für das Expertenreview ausgewählt, zwei aus dem akademischen Bereich und drei aus der Industrie. Die Gutachter erhielten per E-Mail das Informationsdokument. Außerdem erhielten sie Kapitel 4 und Kapitel 5 der Diplomarbeit als zu begutachtende Unterlagen (Prüfling). Im Prüfling sind sowohl das Konzept als auch die Ausarbeitung des Qualitätsprozesses beschrieben. Die Gutachter hatten anschließend eine Woche Zeit die Unterlagen zu prüfen und ihr Gutachten zu erstellen und per E-Mail zurückzuschicken. Nach einer Woche wurden sie erneut an die Anfertigung des Gutachtens erinnert.

6.3. Auswertung

Bei der Auswertung der Gutachten wurde zunächst die Expertise der Gutachter ausgewertet. Anschließend erfolgte eine Auswertung der allgemeinen Einschätzung der Gutachter. Zum Schluss wurden die einzelnen Befunde der Gutachter ausgewertet und wenn nötig korrigiert.

Tabelle 6.1.: Übersicht der Expertise der Gutachter in den Themengebieten „Sicherheitsstandard ISO 26262“, „Automotive SPICE/CMMI“ und „Softwareentwicklungsprozesse“ auf einer Skala von 1 bis 5.

Gutachter	ISO 26262	Automotive SPICE/CMMI	Softwareentwicklungsprozesse
A	4	4	5
B	1	1	4
C	4-5	2	3-4
D	4	4	3

6.3.1. Gutachter

Vier der fünf Gutachter fertigten das Gutachten innerhalb der Frist an. Bei den vier Gutachtern handelt es sich um zwei aus dem akademischen Bereich und um zwei aus der Industrie. Als erstes wurde die Expertise der Gutachter ausgewertet, um ihre allgemeine Einschätzung des Prüflings und ihre Befunde bewerten zu können. Tabelle 6.1 zeigt die Expertise der Gutachter.

6.3.2. Allgemeine Einschätzung des Qualitätsprozesses

Die allgemeine Einschätzung der Gutachter anhand der Prüfaspekte war insgesamt positiv. So erfüllt der Qualitätsprozess nach Einschätzung der Gutachter den Sicherheitsstandard ISO 26262. Auch die erfolgreiche Evaluierung durch Automotive SPICE/CMMI wurde positiv bewertet. Ebenfalls wurde die Umsetzbarkeit des Qualitätsprozesses in der Praxis als realistisch eingeschätzt. Die Vollständigkeit des Qualitätsprozesses, im Sinne dass er alle Aspekte eines Softwareentwicklungsprozesses abdeckt, wurde ebenfalls als gegeben gesehen. Dennoch gab es einige kritische Anmerkungen zur Konzeption und Umsetzung des Qualitätsprozesses. Es handelte sich dabei um folgende Themen:

- Inkonsistente Verwendung des Test-driven Devlopments (Gutachter A und B)
- Konflikt der Continuous Integration mit dem V-Modell (Gutachter B)
- Stärkere agile Ausprägung des Qualitätsprozesses (Gutachter A und D)
- Quality Gates innerhalb des Implementierungs- & Qualitätsanalysezyklus (Gutachter B)
- Beschreibung des Inhalts der geforderten Dokumente (Gutachter D)
- Fehlende Richtlinien für die Auswahl der Entwicklungs- und Qualitätssicherungsmethoden (Gutachter D)
- Keine Berücksichtigung des Tool Confidence Levels (TCL) (Gutachter D)

Inkonsistente Verwendung des Test-driven Developments. Im Konzept und in der Umsetzung des Qualitätsprozesses wurde zwar immer wieder die Verwendung des Test-driven Developments betont, aber in der Implementierungs- & Qualitätssicherungsphase dann doch nicht umgesetzt. Es wurde im Zuge dessen auch angemerkt, ob evtl. das Acceptance Test-driven Development gemeint ist und dass es andere Möglichkeiten gibt, eine Verfolgung der Anforderungen und deren Validierung umzusetzen. Nach eingehender Prüfung wurde deswegen das Test-driven Development verworfen und zur Verfolgung und Validierung der Anforderungen das testorientierte Requirements Engineering verwendet.

Konflikt der Continuous Integration mit dem V-Modell. Im V-Modell ist vorgesehen, dass z. B. beim Aufdecken eines Fehlers im Integrationstest in die Softwarearchitekturphase gewechselt wird, um dort den Fehler zu beheben und anschließend alle Phasen noch einmal durchlaufen werden. Dies ist bei der Continuous Integration durch die Häufigkeit der Durchführung des Integrationstests nicht praktikabel und steht deswegen im Konflikt mit dem V-Modell. Die Bedenken sind berechtigt und bedürfen auf jeden Fall der weiteren Untersuchung.

Stärkere agile Ausprägung des Qualitätsprozesses. Eine stärkere agile Ausprägung des Qualitätsprozesses ist vermutlich wünschenswert. Als eine Möglichkeit dafür wurde SafeScrum genannt. Dies ist sicher richtig und kann in der zukünftigen Forschung zu diesem Thema weiter verfolgt werden.

Quality Gates innerhalb des Implementierungs- & Qualitätsanalysezyklus. Das Fehlen von Quality Gates nach den einzelnen Aktivitäten des Implementierungs- & Qualitätsanalysezyklus wurde angemahnt. Dabei würde es sich um eine Inkonsistenz handelt, durch die die Mechanik aufgeweicht würde und der Standard evtl. unterwandert werden könnte. Die für die Quality Gates notwendigen Dokumente und Ergebnisse könnten werkzeugbasiert und damit automatisch erzeugt und ausgewertet werden. Dies ist sicher richtig und kann in der zukünftigen Forschung zu diesem Thema weiter verfolgt werden.

Beschreibung des Inhalts der geforderten Dokumente. Die Dokumente, die in jeder Phase erstellt werden müssen, werden zwar benannt, aber ihr Inhalt nicht näher beschrieben. Außerdem gibt es auch keinen Verweis auf weiterführende Literatur zu den Dokumenten. Dies ist sicher richtig. Allerdings wurde der Qualitätsprozess für einen Leserkreis beschrieben, der über grundlegende Kenntnisse der Softwareentwicklung verfügt und dem somit die Dokumente bekannt sein sollten.

Fehlende Richtlinien für die Auswahl der Entwicklungs- und Qualitätssicherungsmethoden. Die Kriterien für die Auswahl der Qualitätssicherungs- und Entwicklungsmethoden für die einzelnen Phasen in der Prozessinitialisierung werden nicht näher beschrieben. Dies ist sicher richtig, allerdings ist es auch nicht trivial, solche Richtlinien aufzustellen. Grundsätzlich sind alle Methoden sinnvoll, das handhabt auch der Sicherheitsstandard ISO 26262 so. In diesem wird lediglich eine unterschiedlich starke Empfehlung, abhängig vom Automotive Safety Integrity Level (ASIL) der zu entwickelnden Softwarekomponente, abgegeben. Um etwas Vergleichbares auch für die Entwicklung von Softwarewerkzeugen aufzustellen, müssten diese kategorisiert und dann den Kategorien die Methoden zugeordnet werden. Dies ist sicher ein interessanter Ansatz, sprengt aber den Rahmen dieser Arbeit.

Keine Berücksichtigung des Tool Confidence Levels (TCL). Die unterschiedliche Einstufung der Softwarewerkzeuge bezüglich des Tool Confidence Levels (TCL) bei der späteren Qualifizierung werden im Qualitätsprozess nicht erwähnt und berücksichtigt. Die Berücksichtigung des TCL bei der Entwicklung eines Softwarewerkzeugs ist allerdings nur schwer möglich, da es von der konkreten Aufgabe abhängt, für die das Softwarewerkzeug später eingesetzt wird. So kann sich das TCL von Aufgabe zu Aufgabe unterscheiden. Dies ist aber nicht weiter problematisch, da auch bei einem unterschiedlichen TCL dieselben Methoden für die Qualifizierung empfohlen werden. Alleine die Gewichtung der Empfehlung ist unterschiedlich. Eine Ausnahme hiervon bildet TCL1, bei dem überhaupt keine Qualifizierung erfolgen muss (vgl. Abschnitt 2.1.3).

6.3.3. Befunde der Gutachter

Alle Befunde der Gutachter, bei denen es sich um eine Präzisierung von Formulierungen, dem Beheben von Inkonsistenzen in der Wortwahl und dem Korrigieren von Fehlern in Abbildungen handelt, wurden in das Konzept und die Ausarbeitung des Qualitätsprozesses eingearbeitet. Auch die Nummerierung der Abschnitte in Kapitel 5 wurde überarbeitet, um den Aufbau der Beschreibung des Qualitätsprozesses deutlicher zu machen. Alle Befunde, die inhaltliche Änderungen des Konzeptes oder der Ausarbeitung anregten, wurden daraufhin geprüft, ob es sich um inhaltliche Fehler oder um Erweiterungen des Qualitätsprozesses handelt. Alle inhaltlichen Fehler wurden ebenfalls korrigiert. Die Anregungen für inhaltliche Erweiterungen wurden dagegen nicht umgesetzt, sondern im Rahmen der allgemeinen Einschätzung des Qualitätsprozesses diskutiert (vgl. Abschnitt 6.3.2).

7. Zusammenfassung und Ausblick

Dieses Kapitel fasst die in dieser Arbeit erarbeiteten Konzepte, Erkenntnisse und Ergebnisse zusammen. Außerdem wird ein Ausblick gegeben wie der Qualitätsprozess weiterentwickelt werden kann.

7.1. Zusammenfassung

Um die hohen Qualitätsanforderungen an Softwarewerkzeuge für die Entwicklung eingebetteter Systeme im Automobilumfeld zu gewährleisten, wurde in dieser Arbeit in Zusammenarbeit mit dem Unternehmen TWT ein Qualitätsprozess definiert, der die Nachweisbarkeit von Anforderungen sowie eine Qualifizierung nach dem Sicherheitsstandard ISO 26262 ermöglicht. Dafür muss der Qualitätsprozess den Anforderungen und Vorgaben des Sicherheitsstandards genügen und sich in die bestehenden Softwareentwicklungsprozesse bei TWT einfügen.

Hierfür wurden zunächst die Vorgaben des Sicherheitsstandards zur Softwareentwicklung und zur Qualifizierung von Softwarewerkzeugen analysiert. Danach wurden die bestehenden Softwareentwicklungsprozesse bei TWT untersucht und die Gemeinsamkeiten herausgearbeitet. Aus beidem wurden anschließend Anforderungen an den Qualitätsprozess abgeleitet. Der Qualitätsprozess muss evaluierbar sein, er muss eine Validierung des entwickelten Softwarewerkzeugs ermöglichen, er muss mit den relevanten Teilen des Sicherheitsstandards übereinstimmen sowie die Methode Continuous Integration umsetzen.

Als nächstes wurde das Konzept des Qualitätsprozesses entwickelt. Das Konzept basiert auf dem im Sicherheitsstandard verwendeten V-Modell, erweitert dieses jedoch um den Einsatz eines kontinuierlichen, testorientierten Requirements Engineerings und der Continuous Integration. Des Weiteren wurden Quality Gates eingeführt, die die Phasen des V-Modells voneinander trennen. In ihnen werden sowohl die Softwareanforderungen validiert als auch die Anforderungen und Vorgaben des Sicherheitsstandards auf ihre Einhaltung überprüft.

Abschließend wurde der ausgearbeitete Qualitätsprozess in einem Expertenreview evaluiert. Dabei wurde insbesondere die Übereinstimmung des Qualitätsprozesses mit dem Sicherheitsstandard, die Evaluierbarkeit des Qualitätsprozesses durch Automotive SPICE bzw. CMMI, die Konsistenz und Vollständigkeit des Qualitätsprozesses sowie die Umsetzbarkeit des Qualitätsprozesses in der Praxis bewertet. Die Evaluation ergab, dass der Qualitätsprozess die genannten Prüfaspekte erfüllt. Des Weiteren gab es einige Anregungen für die Erweiterung.

7.2. Ausblick

Dieser Abschnitt soll, basierend auf den Ergebnissen des Expertenreviews, einen Ausblick über mögliche Weiterentwicklungen des Qualitätsprozesses geben.

Der Konflikt zwischen der Continuous Integration und dem V-Modell sollte weiter untersucht werden. Eine mögliche Lösung dieses Konfliktes könnte sein, dass nicht bei jedem Fehler sofort in die entsprechende Entwicklungsphase gewechselt wird. Stattdessen wird die Schwere des Fehlers bewertet. Bei kritischen Fehlern wird direkt in die entsprechende Entwicklungsphase gewechselt, bei nicht kritischen werden diese zunächst nur gesammelt. Wird entweder eine bestimmte Anzahl von Fehlern oder ein passender Zeitpunkt erreicht, wird in die entsprechende Entwicklungsphase gewechselt und alle Fehler auf einmal bearbeitet. Ein passender Zeitpunkt könnte z. B. das Auftreten eines kritischen Fehlers, das Erreichen eines Zwischenziels oder nach Ablauf eines vorher definierten Zeitraums (z. B. einmal pro Woche) sein.

Bisher gibt es innerhalb der Continuous Integration keine Quality Gates, d. h. erst am Ende der Implementierungs- & Qualitätsanalysephase wird die Umsetzung und Einhaltung der Anforderungen an den Qualitätsprozess überprüft. Dadurch kann die Einhaltung des Sicherheitsstandards ISO 26262 gefährdet sein. Eine mögliche Lösung könnte sein, nach jeder Aktivität des Implementierungs- & Qualitätsanalysezyklus ein Quality Gate einzufügen. Um den automatisierten Charakter der Continuous Integration nicht auszuhebeln, muss die Validierung und Prüfung in den Quality Gates ebenfalls automatisiert werden. Dies könnte durch eine werkzeugunterstützte Auswertung der Testprotokolle und dem Erfassen von geeigneten Softwaremetriken ermöglicht werden.

Es gibt bisher keine Richtlinien mit Kriterien für die Auswahl der Qualitätssicherungs- und Entwicklungsmethoden für die einzelnen Phasen in der Prozessinitialisierung. Eine mögliche Lösung könnte sein, solche Richtlinien in Anlehnung an den Sicherheitsstandard ISO 26262 aufzustellen. Dieser spricht Empfehlungen abhängig vom Automotive Safety Integrity Level (ASIL) der zu entwickelnden Softwarekomponente aus. Dementsprechend müssten Kategorien für Softwarewerkzeuge definiert, den Kategorien Kriterien zugeordnet und abhängig davon Empfehlungen abgeleitet werden.

Außerdem könnte eine stärkere agile Ausprägung des Qualitätsprozesses ein sinnvoller Ansatz sein. Eine interessante Möglichkeit wäre es, den Qualitätsprozess an SafeScrum anzulehnen. Dies würde bedeuten, dass das V-Modell, die bisherige Grundstruktur des Qualitätsprozesses, aufgegeben werden müsste und eine komplett neue Struktur entwickelt werden muss.

Unabhängig von den vorgeschlagenen Weiterentwicklungen, sollte die Umsetzbarkeit des Qualitätsprozesses in der Praxis, durch eine Anwendung bei TWT weiter untersucht werden.

A. Expertenreview für die Evaluation des Qualitätsprozesses

In diesem Expertenreview soll der Qualitätsprozess für die Entwicklung qualifizierbarer Softwarewerkzeuge evaluiert werden, der in der Diplomarbeit „Entwicklungsprozess für qualifizierbare Softwarewerkzeuge nach ISO 26262“ definiert wurde. Die Gutachter erhalten dafür Kapitel 4 und 5 der Diplomarbeit als zu begutachtende Unterlagen.

Prüfaspekte

Die Gutachter sollen den Qualitätsprozess unter folgenden Aspekten prüfen:

- Übereinstimmung des Qualitätsprozesses mit dem Sicherheitsstandard ISO 26262
- Evaluierbarkeit des Qualitätsprozesses nach Automotive SPICE/CMMI
- Konsistenz und Vollständigkeit des Qualitätsprozesses
- Umsetzbarkeit des Qualitätsprozesses in der Praxis

Außerdem können noch folgende Aspekte bei der Prüfung berücksichtigt werden:

- Verständlichkeit der Beschreibung und Abbildungen des Qualitätsprozesses
- Korrektheit der Beschreibung und Abbildungen des Qualitätsprozesses

Form des Gutachtens

Das Gutachten wird in schriftlicher Form durchgeführt, d. h. jeder Gutachter hält seine Befunde und Einschätzungen in einem Gutachten fest. Das Gutachten soll dabei folgende Informationen enthalten:

- den Namen des Gutachters,
- eine Einschätzung der eigenen Expertise in den relevanten Themen,
- eine Liste aller Befunde und
- eine allgemeine Einschätzung des Qualitätsprozesses.

Expertise des Gutachters. Jeder Gutachter soll seine Expertise in den Themen „Sicherheitsstandard ISO 26262“, „Automotive SPICE/CMMI“ und „Softwareentwicklungsprozesse“ auf einer Skala von 1 bis 5 einschätzen, bei der 1 gering und 5 hoch bedeutet.

Liste der Befunde. Jeder Gutachter soll seine Befunde auflisten. Jeder Befund soll dabei eine kurze Beschreibung des Befunds (z. B. in Stichworten), eine Referenz auf die betroffenen Bereiche des Qualitätsprozesses (z. B. Abschnitt, Abbildung oder Tabelle) und eine Gewichtung des Befunds beinhalten (kritischer Fehler, Hauptfehler oder Nebenfehler). Um einen kritischen Fehler handelt es sich, wenn der Qualitätsprozess dadurch nicht eingesetzt werden kann. Ein Hauptfehler beschreibt ein Problem, das den Einsatz des Qualitätsprozesses erheblich erschwert. Ein Nebenfehler betrifft zwar nicht den Einsatz des Qualitätsprozesses direkt, aber der Qualitätsprozess wird dadurch zumindest schwerer zu verstehen.

Allgemeine Einschätzung des Qualitätsprozesses. Jeder Gutachter soll eine allgemeine Einschätzung des Qualitätsprozesses, unter Berücksichtigung der Prüfaspekte, abgeben. Dabei soll es sich um einen über die bloßen Befunde hinausgehenden Gesamteindruck des Qualitätsprozesses handeln.

Beispiel für ein Gutachten

Name	Max Mustermann	
Expertise	ISO 26262	4
	Automotive SPICE/CMMI	2
	Softwareentwicklungsprozesse	5
Einschätzung	Eine allgemeine Einschätzung des Qualitätsprozesses, unter Berücksichtigung der Prüfaspekte, die über die bloßen Befunden hinausgeht.	
Befund 1	Tabelle 5.1	Nebenfehler
	Ein Befund in Tabelle 5.1 des Qualitätsprozesses, der als Nebenfehler gewichtet ist.	
Befund 2	Abschnitt 5.1.6	Kritischer Fehler
	Ein Befund im Modulentwurf des Qualitätsprozesses, der als kritischer Fehler gewichtet ist.	
Befund 3	Abbildung 5.8	Hauptfehler
	Ein Befund in Abbildung 5.8 des Qualitätsprozesses, der als Hauptfehler gewichtet ist.	

Literaturverzeichnis

[Booch 1991] BOOCH, G.: *Object Oriented Analysis and Design with Applications*. Benjamin-Cummings Publishing Company, 1991 (Zitiert auf Seite 42)

[CMMI Product Team 2010] CMMI PRODUCT TEAM: CMMI for Development, Version 1.3 / Software Engineering Institute. 2010. – Technical Report (Zitiert auf Seite 23)

[DeMarco 1986] DEMARCO, T.: *Controlling Software Projects: Management, Measurement, and Estimates*. Prentice Hall PTR, 1986. – ISBN 0131717111 (Zitiert auf Seite 33)

[Duvall u. a. 2007] DUVALL, P. M. ; MATYAS, S. ; GLOVER, A.: *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Professional, 2007 (Zitiert auf Seite 42)

[Ebert 2014] EBERT, C.: *Systematisches Requirements Engineering*. 5. dpunkt.verlag, 2014. – ISBN 978–3–86490–139–3 (Zitiert auf den Seiten 25, 27, 34, 35, 36, 37, 38, 39, 40 und 41)

[Fowler 2006] FOWLER, M.: *Continuous Integration*. Mai 2006. – http://www.martinfowler.com/articles/continuousIntegration.html (Zitiert auf den Seiten 27 und 42)

[Frühauf u. a. 2002] FRÜHAUF, K. ; LUDEWIG, J. ; SANDMAYR, H.: *Software-Projektmanagement und -Qualitätssicherung*. 4. vdf, Hochschul-Verlag an der ETH, 2002. – ISBN 978–3–7281–2822–5 (Zitiert auf Seite 30)

[Gamma u. a. 1995] GAMMA, E. ; HELM, R. ; JOHNSON, R. ; VLISSIDES, J.: *Design Patterns*. 1. Addison-Wesley, 1995 (Zitiert auf Seite 28)

[Hoffmann 2013] HOFFMANN, D.W.: *Software-Qualität*. 2. Springer Vieweg, 2013 (eXamen.press). – ISBN 978–3–643–35699–5 (Zitiert auf den Seiten 18, 19, 20, 24, 25, 26, 27, 31 und 33)

[IEC 61508 2010] IEC 61508: Functional safety of electrical/electronical/programmable electronic sefety-related systems / International Electronical Commission. 2010. – Norm (Zitiert auf Seite 11)

[IEEE Standard 610.12 1990] IEEE STANDARD 610.12: Standard Glossary of Software Engineering Terminology / Institute of Electrical and Electronics Engineers. 1990. – Glossar (Zitiert auf den Seiten 21, 24, 26 und 35)

[IEEE Standard 830 1998] IEEE STANDARD 830: Recommended Practice for Software Requirements Specifications / Institute of Electrical and Electronics Engineers. 1998. – Norm (Zitiert auf Seite 41)

[ISO 26262 2011] ISO 26262: Road Vehicles - Functional Safety. / International Organization for Standardization. 2011. – Norm (Zitiert auf den Seiten 9, 10, 11 und 14)

[ISO/IEC 15504-5 2012] ISO/IEC 15504-5: Information technology — Process Assessment — Part 5: An exemplar Process Assessment Model / International Organization for Standardization. 2012. – Norm (Zitiert auf Seite 23)

[Kan 2003] KAN, S.H.: *Metrics and Models in Software Quality Engineering*. 2. Addison-Wesley, 2003. – ISBN 978–0–201–72915–3 (Zitiert auf Seite 33)

[Lanza u. Marinescu 2006] LANZA, M. ; MARINESCU, R.: *Object-Oriented Metrics in Practice*. 1. Springer-Verlag, 2006. – ISBN 978–3–540–24429–5 (Zitiert auf Seite 33)

[Liggesmeyer 2009] LIGGESMEYER, P.: *Software-Qualität: Testen, Analysieren und Verifizieren von Software*. 2. Spektrum Akademischer Verlag, 2009. – ISBN 978–3–8274–2056–5 (Zitiert auf Seite 27)

[Ludewig u. Lichter 2007] LUDEWIG, J. ; LICHTER, H.: *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*. 1. dpunkt.verlag, 2007. – ISBN 978–3–89864–268–2 (Zitiert auf den Seiten 18, 19, 20, 21, 22, 24, 26, 27, 29 und 30)

[Pohl 2008] POHL, K.: *Requirements Engineering: Grundlagen, Prinzipien, Techniken*. 2. dpunkt.verlag, 2008. – ISBN 978–3–89864–550–8 (Zitiert auf den Seiten 27, 35, 36 und 37)

[Prefi 2007] PREFI, T.: Qualitätsmanagement in der Produktentwicklung. In: PFEIFER, T. (Hrsg.) ; SCHMITT, R. (Hrsg.): *Masing Handbuch Qualitätsmanagement*. 5. Carl Hanser Verlag, 2007, Kapitel 19, S. 405–440 (Zitiert auf Seite 21)

[Robertson u. Robertson 2012] ROBERTSON, S. ; ROBERTSON, J.: *Mastering the Requirements Process: Getting Requirements Right*. 3. Addison-Wesley Professional, 2012. – ISBN 978–0321815743 (Zitiert auf Seite 36)

[Schneider 2012] SCHNEIDER, K.: *Abenteuer Software Qualität: Grundlagen und Verfahren für Qualitätssicherung und Qualitätsmanagement*. 2. dpunkt.verlag, 2012. – ISBN 978–3–89864–784–7 (Zitiert auf Seite 21)

[Sommerville 2012] SOMMERVILLE, I.: *Software Engineering*. 9. Pearson Studium, 2012. – ISBN 978–3–86894–099–2 (Zitiert auf Seite 27)

[Sondermann 2007] SONDERMANN, J.P.: Interne Qualitätsanforderungen und Anforderungsbewertung. In: PFEIFER, T. (Hrsg.) ; SCHMITT, R. (Hrsg.): *Masing Handbuch*

Qualitätsmanagement. 5. Carl Hanser Verlag, 2007, Kapitel 18, S. 387–404 (Zitiert auf Seite 21)

[Standish Group 1995] STANDISH GROUP: *The CHAOS Report*. 1995 (Zitiert auf Seite 34)

[VDA QMC Working Group 13 / Automotive SIG 2015] VDA QMC WORKING GROUP 13 / AUTOMOTIVE SIG: Automtoive SPICE Process Assessment / Reference Model 3.0 / VDA QMC Working Group 13 / Automotive SIG. 2015. – Technical Report (Zitiert auf Seite 23)

[Wallin u. a. 2002] WALLIN, C. ; LARSSON, S. ; EKDAHL, F. ; CRNKOVIC, I.: Combining Models for Business Decisions and Software Development. In: *Proceedings of the 28th Euromicro Conference* IEEE Computer Society, 2002 (EC 2002), S. 266–271 (Zitiert auf Seite 20)

Alle URLs wurden zuletzt am 31. 07. 2015 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift