HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

DIPLOMARBEIT

Herr Ing.

Ernst Henry Reichert

Die spezielle Sicherheitsbetrachtung und ihre Ausfallraten im Software Design

Düsseldorf, 2021

Fakultät Angewandte Computer-und Biowissenschaft

DIPLOMARBEIT

Die spezielle Sicherheitsbetrachtung und ihre Ausfallraten im Software Design

Autor:
Herr Ing. Ernst Henry Reichert

Studiengang:
Technische Informatik

Seminargruppe:
TI16w1-F

Erstprüfer:
Prof. Dr. Dr.-Ing. Hartmut Luge

Zweitprüfer:
Dipl.-Ing. (FH) Hans Jürgen Schoenrock

Einreichung:
Mittweida, 12.04.2021

Verteidigung/Bewertung:
Mittweida, 2021

Fakultät Angewandte Computer-und Biowissenschaft

DIPLOM THESIS

The special safety consideration and its failure rates in software design

author:
Herr Ing. Ernst Henry Reichert

course of studies:
Technische Informatik

seminar group:
TI16w1-F

first examiner:
Prof. Dr. Dr.-Ing. Hartmut Luge

second examiner:
Dipl.-Ing. (FH) Hans Jürgen Schoenrock

submission:
Mittweida, 12.04.2021

defence/ evaluation:
Mittweida, 2021

1.1 Bibliografische Beschreibung:

Reichert, Ernst Henry:

Die spezielle Sicherheitsbetrachtung und ihre Ausfallraten im Software Design –2021. – v, 161, IV S.Mittweida, Hochschule Mittweida, Fakultät Computer- und Biowissenschaft, Diplomarbeit, 2021

1.2 Kurzreferat:

In meiner vorherigen Arbeit wurden mit Hilfe von Ansätzen in der Deterministik und Probabilistik nach den Regelwerken [DIN12010] und [DIN22010] Gegenmaßnamen aufgezeigt, was beim heutigen Komponentenausfall in Kraftwerken dazu führt, diese schnellstmöglich zu beheben. Wie dort beschrieben, hat seit den siebziger Jahren die exponierte Computertechnologie sowie der anzuwendenden Wechselwirkung mit der Prozessleittechnik und deren Simulation wie in meiner Projektarbeit diskutiert wird, eingesetzt. In dieser Arbeit wird das Regelwerk [DIN32010] erörtert. Es wird eine partielle Softwareentwicklung im eingebetteten System in den Programmiersprachen ‘C‘ bzw. ‘C++‘ gestellt, welche anhand der ‘Funktionalen Sicherheit‘ über die Bussysteme in der Sicherheitstechnik darstellt werden. Die einzelnen Kapitel werden in Anforderungen von Werkzeugen, Anforderung an die Softwaresicherheit, Anforderung aus der Hardware, Planung der Validation, Planung der Softwarearchitektur, Planung der Softwarekapselung, Planung von Systemdesign, Planung von Moduldesign, Planung von Hardware zu Software, Softwareintegration, Softwarevalidation und Softwaremodifikation diskutiert.

1.3 Danksagung:

An dieser Stelle möchte ich all jenen meinen Dank aussprechen, die zur Entstehung der vorliegenden Diplomarbeit beigetragen haben. Herrn Professor Dr. Dr.-Ing Luge danke ich für die Betreuung meiner Diplomarbeit. Ebenso gilt Herrn Dipl.-Ing. Schoenrock meinen Dank für die fachkundige Bewertung während der Diplomarbeit. Desweiteren bedanke ich mich bei meiner Familie für die Unterstützung während der Diplomarbeitsphase.

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

CPU	Cenrtral Process Unit
CRC	Cyclic redundancy check
FIT	The number of failures per billion hours for a piece of equipment
HTF	Hardware-Fehler-Toleranz
IOPS	Input/Output operations Per Second
IO	Input / Output
IP	Internet Protocol
IT	Informationstechnik
MISRA	Motor Industry Software Reliability Association
MMU	Memory Management Unit
MTTF, $MTTF_d$	Mean Time To Failure (dangerous)
PFD	Probability of dangerous failure per hour
PSD /PST	PROFIsafe- Driver (Treiber)
PV	Process Value
RAM	Read Access Memory
RIOforFA	Remote IO for Factory Automation
ROM	Read Only Memory
RTA	Real Time Automation
RTC	Real Time Cycle
SIL	Safety-Integrity-Level
SFF	Safe Failure Fraction
SRS	Safty Requirements Specifications
SV	Substitute Value
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UML	Unified Modeling Language
VuM	Verfahren und Maßnahme

2 Einleitung

2.1 Problemstellung

Wie schon vorher im Kurzreferat angesprochen, gelten für formale Methoden Sprachen zur Modellierung und oder Spezifikation von Systemen. Ein Design eines Softwaresystems mit Hilfe eines Hardwaresystems beschreibt auf einer gewissen Abstraktionsebene die Funktionalität des Gesamtsystems. Im Gegensatz zu (den meisten) Programmiersprachen besitzen formale Methoden eine genau festgelegte Semantik. Somit gilt eine mathematische Beschreibung zur Bedeutung einer Spezifikation. Diese Festlegung der Semantik erlaubt es, das Software Design bereits vor der eigentlichen Implementierung formal zu analysieren und mögliche Fehler frühzeitig zu finden. Für dezentrale Eingangsgeräte sowie deren Ausgangsgeräte wird eine hohe Kommunikationsleistung für die Automation gefordert. Es reicht für viele Anwendungen nicht aus nur die Qualitätsinformationen eines asynchronen Kanalwerts über den Diagnosemechanismus zu bestimmen. Dieses Profil kann mit Hilfe des ‘PROFIsafe‘ abgedeckt werden.

2.2 Zielsetzung

Ziel dieser Arbeit ist die Bestimmung der erweiterten Sicherheitsbetrachtungen von Bussystemen. In dieser Arbeit wird das Bussystem ‘PROFIBUS‘ mit ‘PROFINET‘ diskutiert. Neben der Systemanalyse der Ausfallmechanismen und der Ursachenanalyse für den Ausfall derer, steht die Abarbeitung bekannter Methoden sowie die Fragestellung “Wie sieht die Fehlerbeherrschung aus und wie hoch ist die Restfehlerrate in Bezug der Ausfallrate?" im Mittelpunkt der Arbeit. Das Resultat dieser Arbeit ist die Betrachtungsweise der Sicherheit im Design, bezogen auf die integrierte Sicherheitsbetrachtung der einzelnen Kanäle. Bei einem Kanalausfall wird das gesamte Fail-Safe-Modul in den Fail-Safe-Status versetzt. Darüber hinaus bietet ‘RIOforFA‘[1] für das ‘PROFIsafe‘ zusätzliche Vorteile für Anwendungen. Normalerweise bietet ein Fail-Safe-Modul mehr als einen Kanal für die Sicherheitssensoren sowie deren Sicherheitsaktoren, das heißt, jeder Kanal kann einer individuellen Sicherheitsfunktion zugeordnet werden.

[1] [RIO2017]

2.3 Aufbau der Arbeit

Nach der Bearbeitung der allgemeinen Anforderungen werden im zweiten Teil die produktspezifische Software, die Anforderung derer Werkzeuge, die Anforderungen des gesamten Sicherheitslebenszyklus, der Systemlebenszyklus der Software, der Entwicklungslebenszyklus nach dem V-Modell, die Sicherheitsanforderung, die Hardwareanforderung, die Validationsplanung, die Softwarearchitektur, die Softwarekapselung, das Systemdesign, das Moduldesign, das Design Hardware zu Software, der Modultest, der Integrationstest, die Softwarevalidation und die Softwaremodifikation diskutiert. Im dritten Teil gibt es einen Überblick der formalen Softwaremethoden, der Integration sowie der Validation und Modifikation. Im darauffolgenden vierten Teil wird das Bussystem mit “PROFIsafe“ vorgestellt, welche den Prozess bestimmen. Durch eine Zusammenführung der einzelnen Notationen werden die verschiedenen Typen untersucht. Im fünften Teil werden die einzelnen Phasen gegenübergestellt. Ausgehend von der Darstellung des grundsätzlichen Vorgehens wird zuletzt eine Nachweisführung der formalen Methoden in der Sicherheitsbetrachtung durch einen Programmauszug eingebracht.

3 Grundlagen und Stand der Technik

3.1 Präzisierung der Aufgabenstellung

Um eine Bussicherheitsintegrität der Software und dem Entwicklungslebenszyklus zu verifizieren bedient man sich dem V-Modell, was in meiner Arbeit ausgiebig erörtert wird. Bei dem Softwareentwicklungsprozess sind nach der Richtlinie und den Regelwerken[2] folgende Phasen zur Vermeidung von systematischen Fehlern in der Software zu betrachten: Softwarespezifikationen für die Sicherheitsfunktion; Entwurf der Softwarearchitektur; Auswahl der Tools und Programmiersprachen; Detaillierter Entwurf der Softwarestruktur und Module sowie der Test der Softwaremodule und Integration. Wie nachher noch erörtert wird, muss bei der Integration von Hardware und Software, die Software mit Eingangssignalen, erwartenden Ergebnissen und erwünschten Bedingungen, die eine Systemreaktion auslösen könnte, programmiert werden. Weitere Phasen sind Softwareverifikation und Validation der Software. Bei der Beurteilung der funktionalen Sicherheit ist es das Ziel den Nachweis, die Anforderungen der Bussicherheit[3] und den definierten Sicherheits-Integritätslevel zu erbringen. Wie bei einer projektierten Software können bei der Integration bzw. Verifikation als auch bei der Validation Modifikationen durchgeführt werden.

[2][DIN32010],[TUEV2018]

[3][Syst2007] Abb.2: Feldbus- und Sicherheitsnormen für Fabrikautomatisierung

3.1.2 Gesamtrahmen der Sicherheitstechnik

Der Gesamtrahmen der Sicherheitstechnik beschreibt den Austausch zwischen Teil 3 der Realisierungsphase für sicherheitsbezogene elektrische/elektronische/programmierbare elektronische Systeme mit Teil 2 der Realisierungsphase für ein sicherheitsbezogenes elektrisches/elektronisches/programmierbares elektronisches System. Die nachfolgenden Themen erörtern diesen Zusammenhang. Der Bezug zwischen Teil 1 und Teil 2 wurde in meiner erste Arbeit[4] ausgiebig diskutiert und ist hier ein Querverweis.

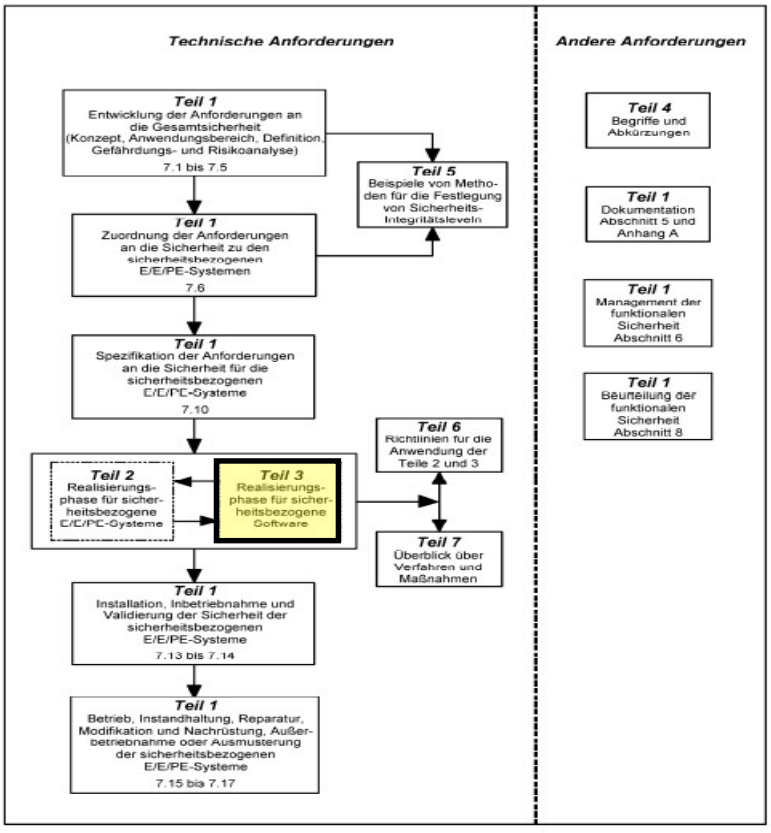

Abbildung 1 Gesamtrahmen der Normreihe 61508

3.1.2.1 Gesamtsicherheitslebenszyklus

[4][Reich2016]

In dieser Arbeit werden die Sicherheitsaspekte softwarenah analysiert und interpretiert. Die Software, die bis heute in der Sichertechnik benutzt wird, ist produktspezifisch und gilt zum Beispiel für ein Betriebssystem, eine Firmware und eine Anwendersoftware. Somit müssen sicherheitsrelevante und rückwirkungsfreie Softwares durch die Richtline[5] Kapitel 1 bewertet werden. Die dazugehörigen Werkzeuge werden zu einem späteren Zeitpunkt erörtert. Es handelt sich hierbei um: Compiler, Entwurfswerkzeuge, Testwerkzeuge, Konfigurationsmanagement, Codegeneratoren, Anforderungsmanagement, Bibliotheken und Andere. Unter Kapitel 9 und 10 des gesamten Sicherheitslebenszyklus werden der Hinweis zum Sicherheitslebenszyklus sowie dessen Realisierung der Software definiert.

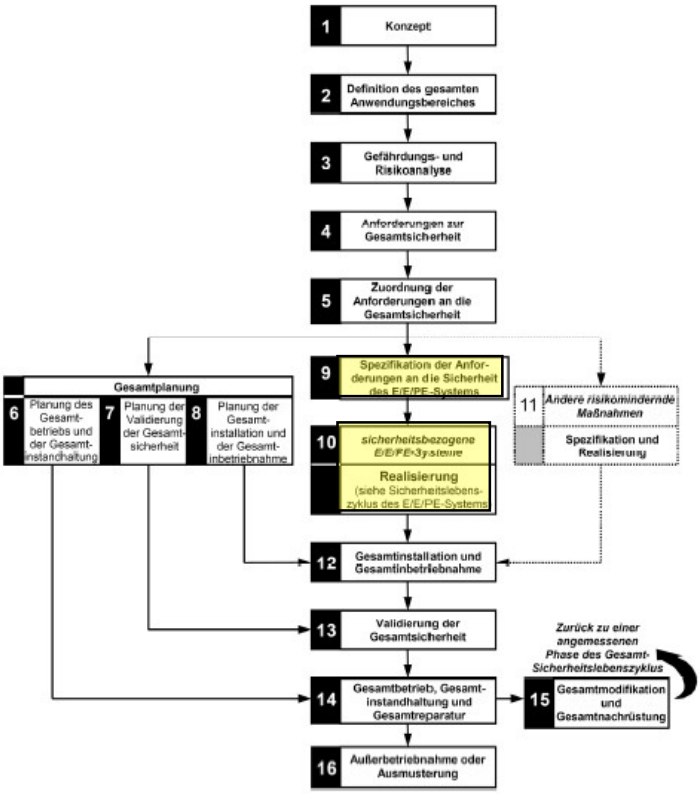

Abbildung 2 Gesamtsicherheitslebenszyklus

3.1.2.2 Systemlebenszyklus der Software

Während der Softwareentwicklung können Fehler vermieden werden, wie beispielsweise in der Planung, wobei dies durch Festlegung von Meilensteinen, das Einsetzen bewährter

[5][DIN32010]

Module, der sorgfältigen Austestung und der Verifikation von unabhängigen Personen und Durchführungen von internen Reviews betrachtet wird. Nachfolgend werden nur die wichtigen Methoden für Software und Design erörtert und im nachfolgenden Kapitel weiter diskutiert. Im Allgemeinen werden die Verfahren oder Maßnahmen nach dem Sicherheits-Integritätslevel SIL definiert. Die Methoden sind je nach SIL „besonders empfohlen", „empfohlen", „nicht empfohlen" und „ausdrücklich nicht empfohlen".

Nach Kastennummer 10.1 im Bild 3[6] – Ziel Spezifikation der Softwareanforderungen an die Sicherheit der Software – beschrieben in Kapitel 7.2.2 gilt: Die Eigenschaft des Entwurfs nach der Sicherheitsintegrität sind in Formale Methoden, Vorwärtsverfolgbarkeit von Systemsicherheit zu Softwaresicherheit und Rückwärtsverfolgbarkeit von der Anforderungssicherheit zu den Sicherheitsbedürfnissen aufgeteilt.

Nach Kastennummer 10.2 im Bild 3 – Ziel Validierungsplan für die Softwareaspekte der Systemsicherheit – beschrieben in Kapitel 7.2.3 gilt: Die Eigenschaft des Entwurfs nach der Sicherheitsintegrität sind in statistische Tests Simulationen, Vorwärtsverfolgbarkeit und Rückwärtsverfolgbarkeit aufgeteilt.

Nach Kastennummer 10.3 im Bild 3 – Ziel Softwareentwurf und Softwareentwicklung – beschrieben in Kapitel 7.4.3 gilt: Architektur und Eigenschaft des Entwurfs nach Sicherheitsintegrität sind in Fehlererkennung, fehlerkennenden Code, diversitäre Überwachungseinrichtungen, diversitäre Redundanz und Rückwärtsgeneration aufgeteilt.

Nach Kastennummer 10.3 im Bild 3 – Ziel Softwareentwurf und Softwareentwicklung – beschrieben in Kapitel 7.4.4 gilt: Werkzeuge und Programmiersprachen als auch die Eigenschaft des Entwurfs nach Sicherheitsintegrität sind in geeignete Programmiersprache, zertifizierte Werkzeuge und zertifizierte Übersetzer aufgeteilt.

Nach Kastennummer 10.3 im Bild 3 – Ziel Softwareentwurf und Softwareentwicklung – beschrieben in Kapitel 7.4.5 gilt: Detaillierter Entwurf und Entwicklung als auch die Eigenschaft des Entwurfs nach Sicherheitsintegrität sind in strukturierte Methoden, defensive Programmierung und Vorwärtsverfolgbarkeit aufgeteilt.

Nach Kastennummer 10.3 im Bild 3 – Softwareentwurf und Softwareentwicklung – beschrieben in Kapitel 7.4.6 gilt: Detaillierte Codeimplementierung und Eigenschaft des Entwurfs nach Sicherheitsintegrität sind in strukturierte Methoden, defensive Programmierung und Vorwärtsverfolgbarkeit aufgeteilt.

---

[6][DIN32010]

Nach Kastennummer 10.3 im Bild 3 – Ziel Softwareentwurf und Softwareentwicklung – beschrieben in Kapitel 7.4.7 und 7.4.8 gilt: Test der Softwaremodule und Entwicklung als auch die Eigenschaft des Entwurfs nach Sicherheitsintegrität sind in statistische Tests, Funktionstests und Vorwärtsverfolgbarkeit aufgeteilt.

Nach Kastennummer 10.3 im Bild 3 – Ziel Softwareentwurf und Softwareentwicklung – beschrieben in Kapitel 7.5.2 gilt: Integration der Software in die Zielhardware als auch die Eigenschaft des Entwurfs nach Sicherheitsintegrität sind in statistische Tests, Black-Box-Tests und Leistungstests aufgeteilt.

Nach Kastennummer 10.3 im Bild 3 – Ziel Softwareentwurf und Softwareentwicklung – beschrieben in Kapitel 7.6.2 gilt: Modifikation und die Eigenschaft des Entwurfs nach Sicherheitsintegrität sind in Einflussanalyse, Validierung durch Regressionstest, Vorwärtsverfolgbarkeit und Rückwärtsverfolgbarkeit aufgeteilt.

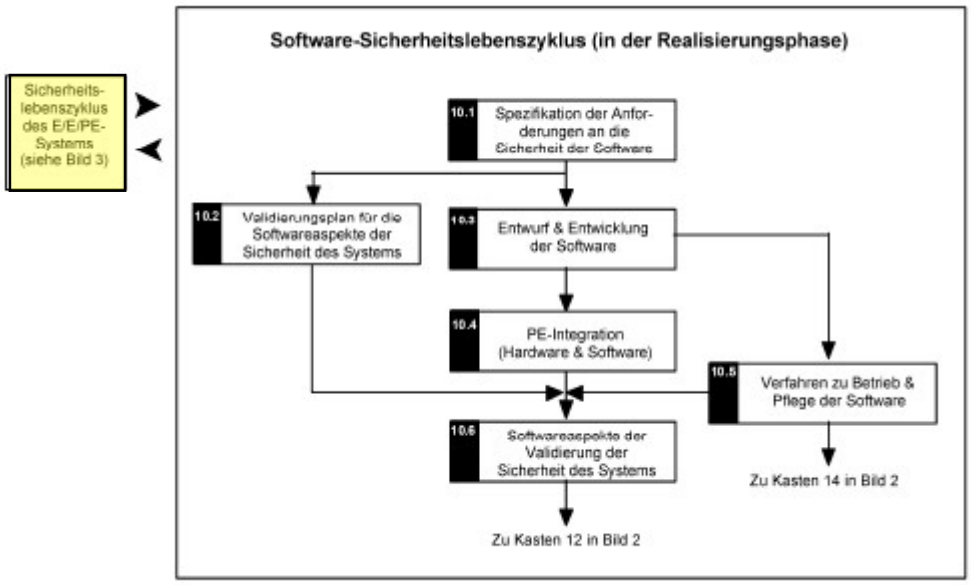

Abbildung 3 Sicherheitslebenszyklus des E/E/PE-Systems (in der Realisierungsphase)

3.1.3 Entwicklungslebenszyklus (nach V-Modell)

Die nachfolgende Abbildung zeigt die Vorgehensweise der nächsten Abschnitte und Kapiteln nach dem V-Modell wie im Systemlebenszyklus der Software erörtert wurde. Die Themen werden von links nach rechts in abfallender und ansteigender Reihenfolge erörtert. Auf der linken Seite handelt es sich um die Verfolgbarkeit von Spezifikationsanforderungen; Architektur und dessen Codier-Richtlinien, Softwaresystementwurf und Modultest. Auf der rechten Seite handelt es sich um die Verfolgbarkeit von Testanforderungen,

Modulprüfung, Integrationstest, die Verifikation, die Hardwareintegration, die Softwareintegration sowie die Validierung der Sicherheitsanforderungen. Die Implementierung über die Kodierung wird in beiden Seiten verwendet.

Was gegenüber der Hardware in der Softwaresicherheitsspezifikation zum Beispiel die Anforderungen der Sicherheitsintegrität ist, ist bei der Softwaresicherheitsanforderungsspezifikation, dass jede Funktion in den verschiedenen Betriebsarten den zugehörigen Sicherheitsintegrität besitzt, das heißt die Erkennung und Beherrschung von Softwarefehlern sowie der Hardwarefehler, der Systemumgebung speziell in Schnittstellen und jeder Funktion zu klassifizieren, ob diese Auswirkungen auf den sicheren Zustand haben oder für die Diagnose zuständig sind.

Bei der Softwarearchitektur handelt es sich um die Umsetzung nach der Sicherheitsintegrität der Software und des Entwicklungslebenszyklus. Heutzutage werden diese Architekturen meistens in der UML gezeigt, die für den Standard des objektorientierten Entwurfs informationsverarbeitende Systeme besitzen und die visuelle Modellierung der Zielsysteme durch verschiedene Diagrammtypen und Erweiterungsmechanismen wie in Kompositionsstrukturdiagrammen, Zustandsdiagrammen und Klassendiagrammen darstellen. Dieses Architekturwerkzeug dient für Techniken und Methoden wie Entwurf, Maßnahmen der Fehlervermeidung, Fehlertoleranzen, Systemumfelder über die Hardware in Eingangsdaten und Ausgangsdaten, Interne Datenstrukturen, Interne und externe Kommunikation, Schnittstellen und Methoden, um Daten aus der Architektur resultierender Testfälle zu sichern. Ein wichtiger Punkt ist, dass die Beschreibung ausgiebig formalisiert und eindeutig sein muss. Im Thema „Validierung“ wird näher auf ihre Bedeutung eingegangen.

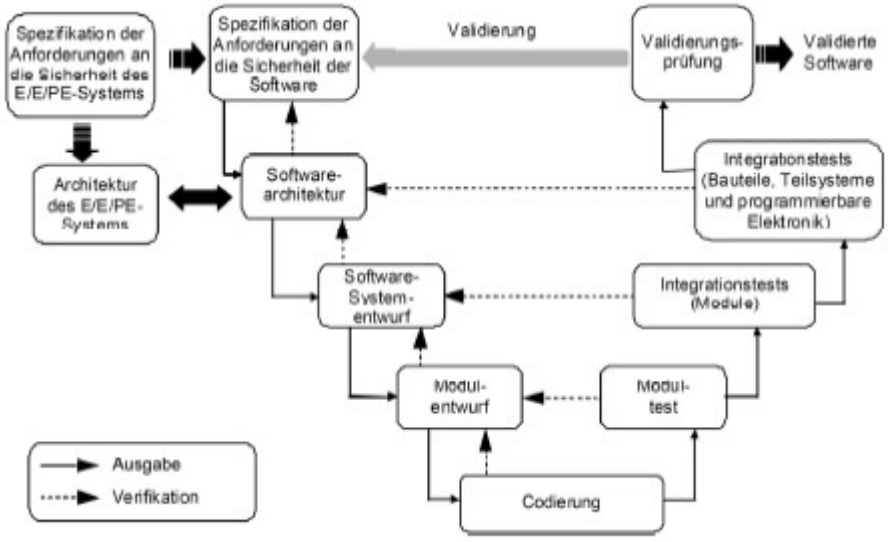

Abbildung 4 Systematische Eignung der Software und Entwicklungslebenszyklus (V-Modell)

3.2 Anforderungen von Werkzeugen

3.2.1 Werkzeugkategorien

Wie schon im vorherigen Kapitel erwähnt, ist eine sorgfältige und gut strukturierte Entwicklung wichtig, um jede Phase so gut zu dokumentieren, dass sie reproduzierbar ist. Die Anforderungen der Werkzeuge werden als Leitfaden in den Anhang[7] A von „Vermeiden von systematischen Fehlern in der Software während den Lebenszyklen“ und Anhang B als Detailtabelle für Verfahren und Maßnahmen zum Qualitätsmanagement der Software herangezogen. Für ein sicherheitsbezogenes System muss ein Online-Software-Werkzeug als Softwareelement angesehen werden. Es sollten Offline-Werkzeuge verwendet werden, welche die Entwicklung der Software mit dem Ziel, die Integrität der Software durch Verringerung der Wahrscheinlichkeit Fehler einzubringen zu unterstützen oder solche nicht zu bemerken, zu erhöhen. Für Transformations- oder Übersetzungs-Werkzeuge sind Compiler, Assembler, Linker, Ladeprogramme, Ladewerkzeuge und Code generierende Werkzeuge notwendig und für die Verifikation- und Validierungswerkzeuge werden statische Codeanalysatoren, Testabdeckungskontrollen und Simulatoren angewendet. Für Diagnosewerkzeuge gilt es die Software unter Betriebsbedingungen zu überwachen und zu erhalten, für Werkzeuge zur Infrastruktur werden Systeme zur Unterstützung der Entwicklung benutzt und für Konfigurationswerkzeuge beispielsweise Werkzeuge zur Versionsverwaltung. Werkzeuge für Anwendungsdaten beinhalten Funktionsparameter, Anzeigebereiche, Alarm – und Schaltschwellen. Ein Offline-Software-Werkzeug unterstützt eine Phase und kann ein sicherheitsrelevantes System nicht zur Laufzeit beeinflussen.

[7] [DIN32010]

3.2.2 Klassen von Hilfswerkzeugen

Es handelt sich hier um Hilfswerkzeuge, da die Klassen nur indirekt im Kapitel „Werkzeuge und Programmiersprache“ beschrieben werden und deren Klassen in Kapitel 3.1 definiert[8] und dokumentiert werden müssen. Diese Richtlinien sind rudimentär zu betrachten. Es handelt sich um die einzelnen Hilfswerkzeuge in den Klassendefinitionen T1, T2 und T3.
Die Klasse T1 erzeugt keine Ausgaben, die direkt oder indirekt zum ausführbaren Code des sicherheitsbetriebenen Systems beitragen können. Als Beispiele sind Texteditoren und Konfigurationsmanagement zu bezeichnen.
Die Klasse T2 unterstützt den Test, die Verifikation des Entwurfs oder die ausführbaren Codes, bei dem Fehler im Werkzeug zur Nichterkennung von Fehlern führen, jedoch in der ausführbaren Software keine Fehler direkt erzeugen können. Als Beispiele sind Messungen zur Textabdeckung und statische Analysen zu benennen.
Die Klasse T3 erzeugt Ausgaben, die direkt oder indirekt zum ausführbaren Code (einschließlich Daten) des Systems beitragen.

3.2.2.1 Anforderungen an Hilfswerkzeugen

Nach Kapitel 7.7.4.4 und 7.7.4.6[9] gilt es für die Beurteilung der Offline-Software-Werkzeuge den Grad des dem Hilfswerkzeug entgegengebrachten Vertrauens und die potenziellen Ausfallmechanismen, welche die ausführbare Software beeinflussen können, zu bestimmen. Weiterhin muss für das Hilfswerkzeug der Nachweis verfügbar sein, dass das Software-Werkzeug durch erfolgreiche Verwendung oder ähnliche Umgebungen oder Anwendungen und- oder Validation des Softwarewerkzeuges mit seiner Spezifikation oder seinem Handbuch übereinstimmt.

---

[8] [TUEV2018] Bahnanwendungen - Anwendungen für Schienenfahrzeuge

[9] [DIN32010]

3.2.2.2 Validation der Werkzeuge

Für Kapitel 7.4.4.7 gilt für die Definition und Dokumentation: Validierungsaktivitäten; Version des verwendeten Produkthandbuch; validierte Funktion des Werkzeugs; Testfälle und Ergebnisse. Unter Kapitel 7.4.4.18 findet man die Begründung einer neuen Version. Ein Software-Werkzeug kann auf dem Vertrauen der Vorgängerversion basieren, wenn ausreichende Nachweise vorliegen.

3.3 Anforderungen an die Softwaresicherheit

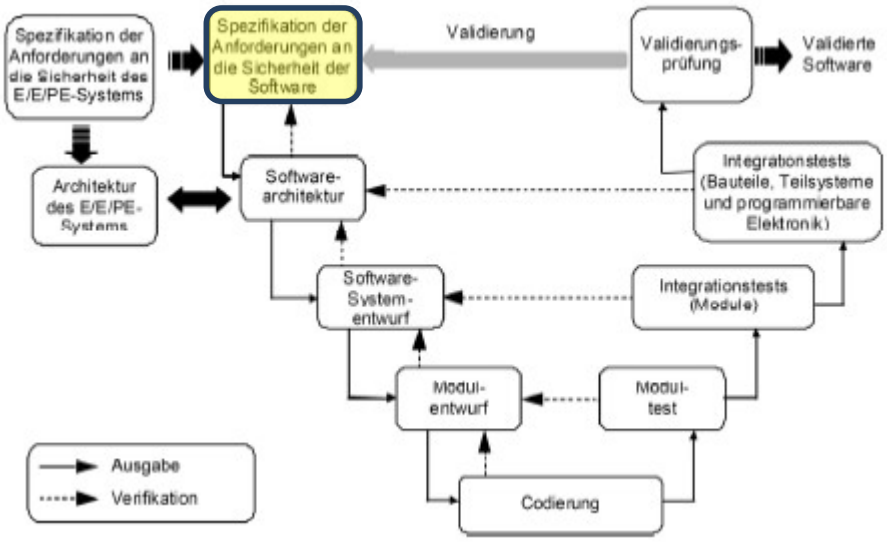

Abbildung 5 Software und Entwicklungslebenszyklus (Spezifikationsanforderung an die Softwaresicherheit)

3.3.1 Sicherheitsrelevante Funktionen

Wie in Tabelle 1[10] gilt der Verweis 7.2.2. Der Softwareentwickler muss die Information 7.2.2.2 bewerten, um sicherzustellen, dass die Anforderungen wie folgt aussehen: Alle

[10][DIN32010]

Softwarefunktionen sind dem Sicherheits-Integritätslevel angepasst; Sicherheitsfunktionen, Betriebsarten, Selbstüberwachung, Kapazitäten sowie Antwortzeiten und Bedienschnittstellen sind spezifiziert. Die Softwaresicherheitsanforderungen können auch schon in den Systemsicherheitsanforderungen implementiert sein. In diesem Falle müssen die Anforderungen hier nicht erneut genannt werden.

3.3.1.1 Nichtsicherheitsrelevante Funktionen

Wie in Tabelle 1 gilt der Verweis 7.4.2. Wenn die Software sowohl Sicherheitsfunktionen als auch Nichtsicherheitsfunktionen umsetzt, muss die gesamte Software als sicherheitsbezogen behandelt werden, es sei denn die Rückwirkungsfreiheit zwischen den Funktionen im Entwurf können nicht gezeigt werden. Ein weiterer Punkt betrifft den höchsten Sicherheits-Integritätslevel bei verschiedenen Sicherheits-Integritätslevel in den Sicherheitsfunktionen.

3.3.2 Verfahren und Maßnahmen

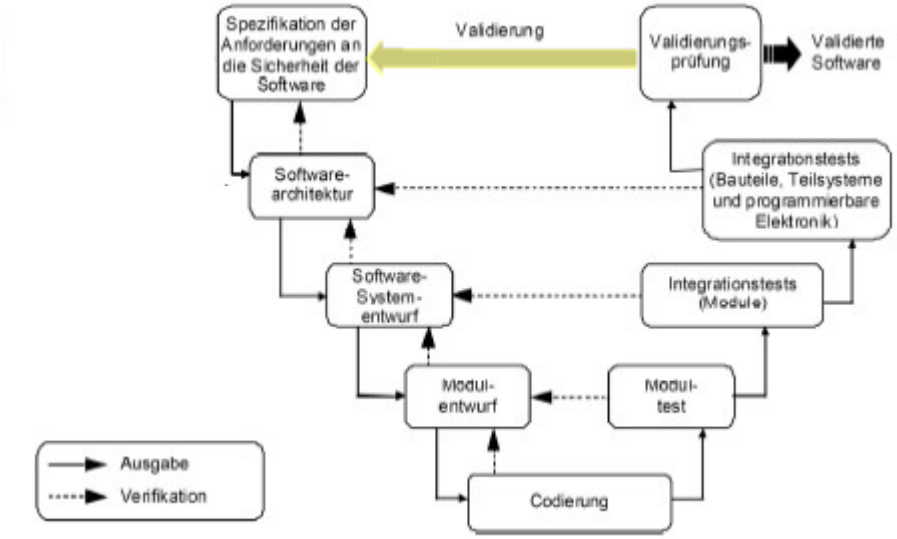

Abbildung 6 Software und Entwicklungslebenszyklus (VuM)

Wie in Tabelle A1 gilt der Verweis zur Tabelle B1 Softwaresicherheitsanforderungen bestehen immer aus einer Beschreibung und notwendigen mathematischen Darstellungen, welche die Anwendung beschreiben. Rechnergestützte Spezifikationswerkzeuge sollen unter anderem helfen, dass eine geforderte Spezifikation das Sicherheitsabbild widerspiegelt und nicht bei der Entwicklung vergessen wird. Auch die Vermeidung der Zweideutigkeit ist Aufgabe der Softwarespezifikationswerkzeuge. Formale Methoden haben das Ziel eine mathematische basierte Softwareentwicklung zu fordern. Dies beinhaltet

einen formalen Entwurf und formale Kodierungsverfahren. Wie vorher die Themen Vorwärts- und Rückwärtsverfolgbarkeit im Kapitel 2.1.1.2 schon einmal angesprochen, gilt für die Vorwärtsverfolgbarkeit, die weitgehende Überprüfung, das sich in späteren Lebenszyklusphasen angemessen mit einer Anforderung befasst wird. Für die Rückwärtsverfolgbarkeit gilt die weitgehende Überprüfung, dass jede Implementierungsentscheidung (…) durch irgendeine Anforderung klar gerechtfertigt wird.

3.4 Anforderungen aus der Hardware

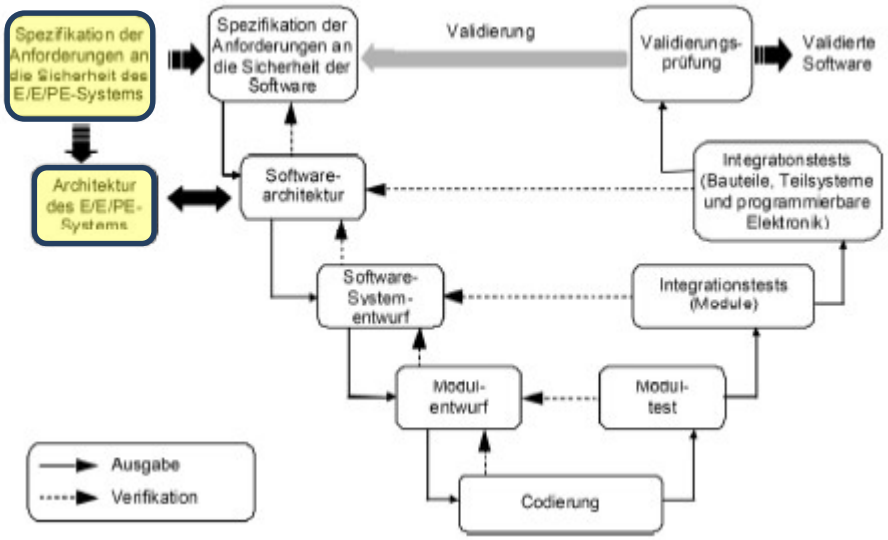

Abbildung 7 Software und Entwicklungslebenszyklus (Anforderung aus der Hardware)

Da in der vorherigen Arbeit das Regelwerk[11] erörtert wurde, wird nachfolgend nur die Hardware in ihrer Komplexibilität aus Sicht des sogenannten Regelwerkes angeschnitten. Bei der Komplexibilität handelt es sich um die Hardware der CPU. Im Allgemeinen verlangt der Einsatz von Hardware in der Sicherheitstechnik eine zuverlässige Hardwareumgebung, um eine sichere Steuerung oder Regelung sowie Überwachung zu gewährleisten. Die einzusetzenden Hardwarediagnosen dienen dem Auffinden je nach Hardwarearchitektur möglichst alle Fehler aufzudecken und Ausfälle während des Betriebes zu erkennen und zu beherrschen. Die implementierenden Maßnahmen werden über Anforderung des SIL bestimmt. Das Regelwerk fordert somit eine angemessene Qualität in RAM, ROM und in der zentralen Prozessorprüfung.

[11][DIN22010]

3.4.1 Fehlermodelle der CPU

Mit Hilfe einer eigenen Ablaufsteuerung sollen bei den Testungen interne Fehler gefunden werden. Bei diesen Fehlern handelt es sich um den internen Aufbau einer CPU im Speicher, Adressbus, Datenbus sowie dem Mikroprozessor in Puffer, die logische arithmetische Einheit mit Register, Befehlsdecoder und Adressregister. Da die CPU-Prüfungen von der Hardwarearchitektur abhängig sind, muss ein minimaler Zyklus der CPU in Lesen des Hauptspeicherbefehls, Decodieren dieser, Befehlsabarbeitung durch Abspeichern und wenn vorhanden durch einen Sprung auf eine bestimmte Adresse sowie den korrekten Befehl signalisiert geprüft werden. Bei der Abarbeitung der einzelnen Befehle werden alle Funktionseinheiten bis auf die Register benutzt. Dadurch ist eine separate Prüfung nicht möglich. Es ist möglich stationäre Fehler bei der CPU-Prüfung zu erkennen. Hingegen sind transitions-behaftete Fehler nur von zufälliger Natur diese aufzudecken.“Der Anteil SFF ist der Quotient der mittleren Ausfallrate ungefährlicher Ausfälle (λ_{Savg}) plus der gefahrbringenden erkannten Ausfälle (λ_{DDavg}) und der ungefährlichen Ausfälle (λ_{Savg}) plus der gefahrbringenden Ausfälle (λ_{Davg}).“[12] Durch eine implizite Gewährleistung basierend auf der CPU- Prüfung garantiert ein korrekter Durchlauf die Zuordnungskorrektheit des Operationscodes. Es muss gewährleistet werden, dass die Gestaltung der CPU-Prüfung stark abhängig der tatsächlichen vorhandenen CPU in der Simulation anzupassen ist. Es müssen mindestens einmal die in der Applikation verwendeten Maschinenbefehle geprüft werden. Über die Quellcodeanalyse können die Maschinenbefehle erodiert und die Prüfung muss maschinennah implementiert werden. Aufgrund der komplexen Arbeitsweise eines Compilers ist die Verwendung einer Hochsprache für die implementierte CPU-Prüfung ausgeschlossen. Wie vorher schon angemerkt, sollte der Anteil sicherer Ausfälle zuerst getestet werden. Der wahrscheinlichste Ausfall ist die Störung auf einer Datenleitung. Somit ist die Prämisse diese Prüfung zuerst auszuführen und den internen Aufbau der CPU über den Operationscode zu verifizieren. Der Ablauf der CPU-Prüfung sollte nachdem möglichst wenige Funktionsblöcke verwenden.

3.4.1.1 RAM-Test

Durch das Ansprechen der Register kann es zu Fehlern kommen und es sollte ein geeignetes Verfahren für die RAM-Prüfung gemäß dem Diagnosedeckungsgrad vorgesehen werden. Dabei wird mit dem Diagnosedeckungsgrad “ $DC = (\sum \lambda_{DD} \ / \sum \lambda_D)$ sowie mit der Tabelle A die mittlere Rate der Ausfälle (λ) mit der ungefährlichen Ausfallrate (λ_S)

[12] [Reich2016]

und der gefährlichen Ausfallrate (λ_D) bestimmt. Bei der Ausfallrate (λ_S) gilt es die Addition der entdeckten (λ_{SD})und unentdeckten (λ_{SU}) ungefährlichen Ausfällen zu bewerten. Bei der Ausfallrate (λ_D) ist die Addition der entdeckten (λ_{DD}) und unentdeckten (λ_{DU}) gefährlichen Ausfällen zu bewerten“. Nach der Reihenfolge sollte zuerst die unbedingten Sprünge, die Transferbefehle und danach die bedingten - also Sprungmarken, wenn vorhanden- getestet werden. Weitere Maschinenbefehle der Typen wie logische und arithmetische Operationen, Bitoperationen, Unterprogramme, und Unterbrechungsbehandlung können aufbauend getestet werden. Die Prüfung muss im Fehlerfall das Anhalten der CPU gewährleisten oder ein Halten über Endlosschleifen pseudomäßig erbringen. Es muss ausgeschlossen werden, dass kein Watchdog mehr für einen Task beziehungsweise eine Programmlaufkontrolle getriggert wird. Es muss sofort in das sichere Ereignisprogramm gesprungen werden. Für eine Reinitialisierung muss eine Quittierung vorgenommen werden. Bei erfolgreichem Test soll das Ergebnis in ein Bitformmuster und in den RAM-Speicher geschrieben werden. Der erfolgreiche Ausgang der Prüfung darf nur einer sein. Die Sprungbefehle sollten bei Fehlern reagieren und der Startpunkt der Routine im Programmspeicher liegen. Um das Thema der Prüfung noch einmal aufzugreifen, ist es nötigt die Speicherzellen zu lesen und zu beschreiben. Die Verfahren, die benutzt werden, überprüfen das Zielfeld, den Adressdekoder und die Schreib- sowie Leseeinrichtung. Die benutzen Bitmuster stehen dann im RAM-Speicher, um das Prüfungsverfahren zu evaluieren, damit der Inhalt des Speichers beim Prüfen nicht zerstört wird. Die Funktionsweise ist ein abwechselndes Beschreiben mit Nullen und Einsen, Auslesen und kontrolliert die einzelnen Bitinformationen. Danach wird ein zweiter Durchlauf mit dem invertierten Bitmuster im Einerkomplement durchgeführt. Auf die einzelnen Testverfahren bis auf Sicht der Speicherbereiche in Tabelle A6 wird in dieser Arbeit nicht eingegangen. Der Überblick dieses Testverfahrens ist das einfache Verfahren der Prüfsumme über die Antivalenz sowie zyklisch spaltenweise gegeneinander verschobene Speicherstellen. Die Bitinformation von gerade oder ungerade bildet ein einfaches Verfahren vom Parity-Bit zu jeder Speicherstelle. Die arithmetische Addition ist ein hochwertiges Verfahren und wird nur mit definierter Ausfallrichtung beim Speicher beschrieben, wobei die Speicherstellen saldiert und die Überläufe nicht verworfen werden. Beim mittleren Verfahren des modifizierten Hammingcodes wird über eine Polynomdivision über die einzelnen Speicherstellen beziehungsweise der verschachtelten Parity-Bits gewertet. Über das hochwertige Verfahren der Signaturbildung wird eine fortgesetzte Polynomdivision angewendet und im weiteren Kapitel noch einmal aufgegriffen. Fazit ist, dass die Diagnose des variablen Speicherbereiches neben Datenverfälschung auch eine falsche Adressierung aufdecken muss.

3.4.1.2 ROM-Test

Nach Tabelle A5 und Tabelle A 4.4[13] ist das Ziel die Erkennung aller Einbit- und Mehrbitfehler innerhalb eines Wortes. Bei großen Datenmengen wird eine 128bit Signatur verwendet. Der Überblick dieses Testverfahrens ist das einfache Einbitverfahren, welches ein Parity-Bit zu jeder Speicherzelle bildet, die arithmetische Addition als ein hochwertiges Verfahren benutzt und nur mit definierter Ausfallrichtung beim Speicher beschrieben wird, wobei die Speicherstellen saldiert und die Überläufe nicht verworfen werden. Beim Mehrbitverfahren ist ein leistungsfähigerer Code einzusetzen. Dieser Signaturcode wird später noch einmal aufgegriffen.

[13][DIN72010]

Planung der Validation

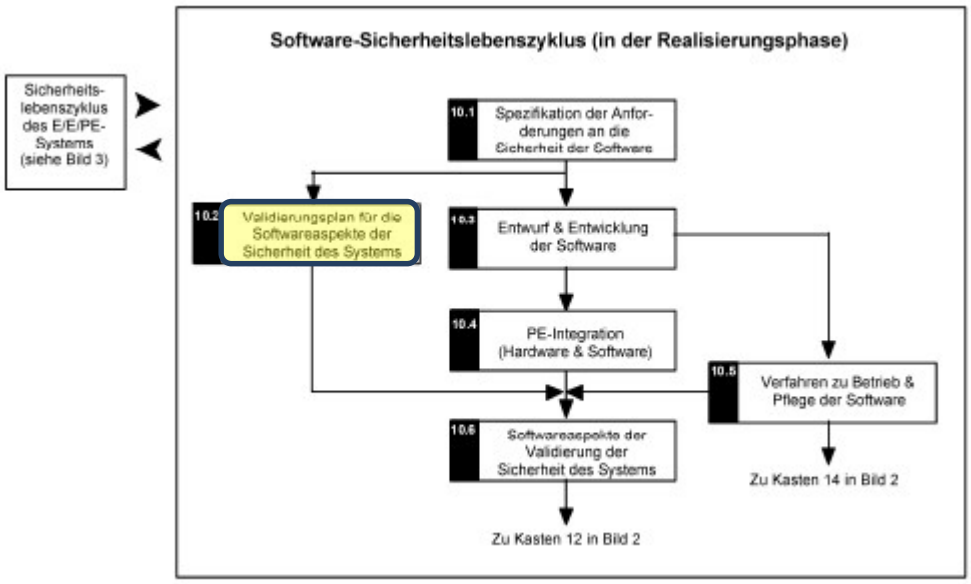

Abbildung 8 Systemlebenszyklus der Software

3.4.2 Validation und Verfolgbarkeit

Nach Tabelle A 7[14] gilt für die obligatorische Validation: Die Validation ist abgeschlossen, wenn alle Anforderungen aus der SRS getestet wurden.

	Verfahren/Maßnahme *)	Siehe	SIL 1	SIL 2	SIL 3	SIL 4
1	Statistische Tests	C.5.1	o	+	+	++
2	Simulation des Prozesses	C.5.18	+	+	++	++
3	Modellbildung	Tabelle B.5	+	+	++	++
4	Funktionstest und Black-Box-Test	B.5.1, B.5.2, Tabelle B.3	++	++	++	++

Abbildung 9 Validation und Verfolgbarkeit

[14][DIN32010]

Diese Anforderungen gelten für die obligatorische Validation: Die Validation ist abgeschlossen, wenn alle Anforderungen aus der SRS getestet wurden. Die Anforderungen an die Software-Sicherheitsvalidationsplanung gelten für die Kriterien „bestanden“ und „nicht bestanden“. Es werden Eingangs- / und Ausgangssignale mit Reihenfolge und Werten validiert sowie andere Annahmekriterien wie Stackgröße (Speicherbereich), Zeit und Wertetoleranz bewertet.

3.5 Planung der Softwarearchitektur

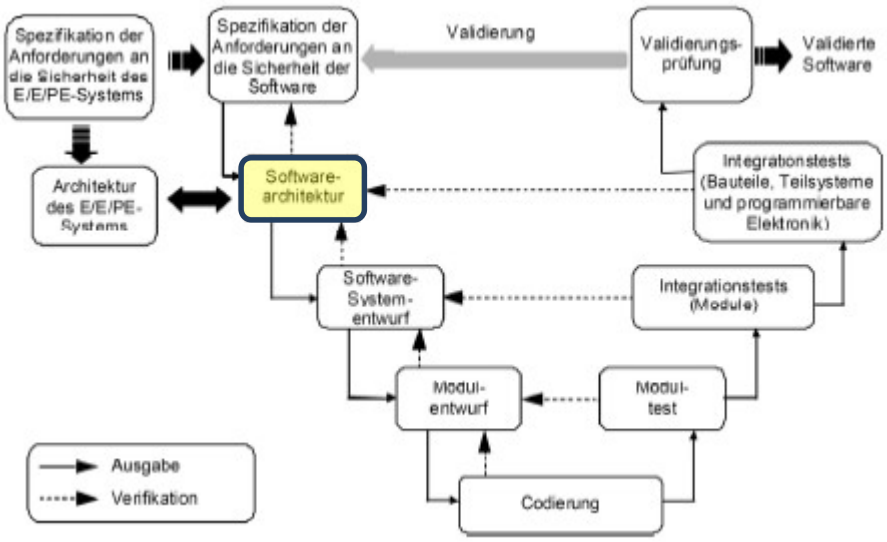

Abbildung 10 Software und Entwicklungslebenszyklus (Planung der Softwarearchitektur)

3.5.1 Besonderheiten in der Architektur

Die Anforderungen des Entwurfs sind im Kapitel 7.4.3[15] definiert. Die Themen sind: Das Ablegen der Daten, die Realisierung von interner und externer Kommunikation, die Realisierungen von Ein- und Ausgaben, die Erklärung von Einsatz fremder Software, die Anwenderdefinition von Schnittstellen und die Verhinderung von nicht sicheren Softwareteilen mit der Vermischung sicherer Softwareteile. Die wesentlichen Elemente und Teilsysteme der Software und deren Verbindung legt die Softwarearchitektur fest. Beim Thema Anforderungen bereits existierender Software -erörtert im Kapitel 7.4.2.12- ist auf die

[15][DIN32010]

Vermeidung und Beherrschung von systematischen Fehlern zu achten. Den Ansatz von ‘prove on use‘ und ‘Assessment‘ werden hier vernachlässigt, da sich die PROFIsafe-Schnittstelle auf Kapitel 7.4.12 stützt. Bei der Empfehlung der Programmiersprache in Tabelle C.1 wird der Verweis C 4.5[16] in den Programmiersprachen für SIL 2 und SIL 3, was ebenfalls für PROFIsafe gilt, definiert. Der Kodierungsstandard wird im nächsten Kapitel beschrieben.

	Programmiersprache	SIL1	SIL2	SIL3	SIL4
1	ADA	++	++	+	+
2	ADA mit Teilmenge	++	++	++	++
3	Java	--	--	--	--
4	Java mit Teilmenge (entweder ohne automatischer Speicherbereinigung (en: garbage collection) oder einschließlich automatischer Speicherbereinigung, die den Anwendungscode nicht für eine wesentliche Zeitspanne anhält). Siehe Anhang G zur Anleitung für die Anwendung von objektorientierten Möglichkeiten.	+	+	--	--
5	PASCAL (siehe Anmerkung 1)	++	++	+	+
6	PASCAL mit Teilmenge	++	++	++	++
7	FORTRAN 77	+	+	+	+
8	FORTRAN 77 mit Teilmenge	++	++	++	++
9	C	+	o	--	--
10	C mit Teilmenge und Programmierrichtlinie und Verwendung von statischen Analysewerkzeugen	++	++	++	++
11	C++ (siehe Anhang G zur Anleitung für die Anwendung von objektorientierten Möglichkeiten)	+	o	--	--
12	C++ mit Teilmenge und Programmierrichtlinie und Anwendung von statischen Analysewerkzeugen (siehe Anhang G zur Anleitung für die Anwendung von objektorientierten Möglichkeiten)	++	++	++	++
13	Assembler	+	+	o	o
14	Assembler mit Teilmenge und Programmierrichtlinie	+	+	+	+
15	Kontaktplan	+	+	+	+
16	Kontaktplan mit definierter Sprachenteilmenge	++	++	++	++
17	Funktionsbausteinsprache	+	+	+	+

Abbildung 11 Programmiersprache für SIL 2 und SIL 3

3.5.2 Kodier- und Programmierrichtlinien

Wie in Tabelle A4 gilt der Verweis C 2.6 sowie die Tabelle B1, um die Programmierrichtlienen für den Entwurf und für eingebettete Programme zu deuten. Dies wird hier durch MISRA C[17] vorgestellt. Es dient der Richtlinie für eingebettete C – Programme in kriti-

[16][DIN72010]

[17][MISR2013]

schen Systemen. Es gelten keine allgemeinen Programmierrichtlinien. Um MISRA C zu verstehen, sind noch einige Fragen zu klären. Jede Einteilung von MISRA C wird entweder als Regel oder als Richtlinie klassifiziert. Eine Richtlinie ist ein Leitfaden, die es nicht ermöglicht, die vollständige Beschreibung zur Überprüfung der Einhaltung von Vorschriften zu liefern. Zusätzliche Informationen wie zum Beispiel in Entwurfsspezifikationen oder Anforderungsspezifikationen angegeben, sind erforderlich, um die Prüfung durchführen zu können. Statische Analysewerkzeuge können in der Lage sein bei der Überprüfung der Einhaltung von Richtlinien zu helfen, aber falsche Interpretationen können zu unterschiedliche Auslegungen hinsichtlich der Unkompliziertheit vornehmen. Eine Regel ist eine Richtlinie, die eine vollständige Beschreibung der Anforderungen enthält. Es sollte möglich sein, zu überprüfen, ob der Quellcode einer Regel entspricht ohne dass weitere Informationen erforderlich sind. Statische Analysewerkzeuge sollten in der Lage sein, die Einhaltung von Regeln zu überprüfen, die der Einschränkungen unterliegen. Bei der Kategorie von „zwingend notwendig“ gilt, dass der C-Code dem ihn aufgeprägten Anspruch entspricht und alle zwingenden Richtlinienabweichungen der vorgeschriebenen Richtlinien enthalten muss. Bei der Kategorie von „erforderlich“ gilt, dass der C-Code dem ihn aufgeprägten Anspruch entspricht und alle vorgeschriebenen Richtlinien bei einer formalen Abweichung enthalten muss. Bei der Kategorie von „beratend“ handelt es sich um Empfehlungen. Der Status der Beratung bedeutet jedoch nicht, dass diese Punkte ignoriert werden können, sondern dass sie so weit wie möglich vernünftig sein sollten. Formelle Abweichungen sind für Beratungsrichtlinien nicht erforderlich. Aber wenn der formale Abweichungsprozess nicht eingehalten wird, sollten alternative Vorkehrungen getroffen werden, um Nichteinhaltungen zu dokumentieren. Über diese grundlegende Klassifikation hinaus gibt das Dokument keine Wichtigkeitsabstufung der einzelnen Richtlinien an oder impliziert dies. Alle erforderlichen Richtlinien, ob Vorschriften oder Regeln, sollten genauso berücksichtigt werden wie alle zwingend notwendigen, obligatorischen und beratenden Kategorien. Zusätzlich werden die Kennzeichner C90 und C99 genannt, die für das undefinierte Verhalten stehen und gelöst werden müssen. Diese Verhalten stehen in diesem Regelwerk explizit im Anhang G. Im Allgemeinen enthält dieser Anhang die Prüfliste für das implementierungsdefinierte Verhalten auf der in der Richtlinie 1.1 verwiesen wird. Als wichtiger Zusatz sollte noch das Verhindern von rekursiven Aufrufen, Schleifen kein definiertes Ende in der Bedingung, kein GOTO und nur verifiziertes Block-Copy im RAM Bereich vorhanden sein.

3.6 Planung der Softwarekapselung

3.6.1 Motivation zur Kapselung

Von einer Kapselung spricht man bei einer Kopplung zwischen sicherer und nicht sicherer Software, dass heißt, wenn die sichere Software und die nicht sichere Software auf die gleichen Ressourcen zugreifen. Als Ergebnis dürfen die sichere Software sowie deren

Daten und Abläufe keinen negativen Einfluss haben. In Angang F[18] stehen die nominativen Anforderungen. Elemente dürfen sich nicht gegenseitig durch ihr Verhalten bei der Ausführung nachteilig beeinflussen, sodass ein gefährlicher Ausfall auftreten könnte. Bei der räumlichen Trennung dürfen die verwendeten Daten eines Elements nicht durch ein anderes verändert werden. Bei der zeitlichen Trennung darf die korrekte Funktion eines Elementes nicht von einem anderen Element durch Verwendung von zu viel verfügbarer Prozessorleistung oder durch Verhinderung der Ausführung des anderen Elements durch Blockieren irgendwelcher, gemeinsam genutzter Ressourcen beeinflusst werden.

3.6.1.1 Kapselung von Daten

Mit Hilfe von Funktionen wird ein Zugriff auf das Modul und seine Daten sowie jede Änderung des Variablewertes erfolgen. Die nachfolgende Abbildung zeigt eine Datenkapselung und deren Austausch über eine public - Funktion zwischen einer sicheren und nicht sicheren Schreibweise sowie über eine private - Funktion zwischen einer sicheren und nicht sicheren Schreibweise.

[18][DIN32010]

```
// Die Funktion void access() erlaubt einen Zugriff über extern. Alle Variablen
// sowie die Funktion standardfile.c ist von aussen erreichbar.

// safetyfile.c
#include "standardfile.h"

//...

void acess() {
    extern lon getSub(); // Achtung Zugriff von extern
    long | = getSub();

    }

// standardfile.h
long getValue();

// standardfile.c
typedef struc{x,y}
object_a

object_a a = {4,2};

long getSub() {
    return (a.x -a.y);
    }

long getValue() {
    return getSub();
    }
```

```
// Die Funktion void access() verbietet einen Zugriff über extern. Alle Variablen
// sowie die Funktion standardfile.c ist nur noch private und die Schnittstelle bezieht
// sich auf den Header.

// safetyfile.c
#include "standardfile.h"

//...

void acess() {
    // Kann nur über getValue zugreifen
    }

// standardfile.h
long getValue();

// standardfile.c
typedef struc{x,y}
object_a

static object_a a = {4,2};

static long getSub() {
    return (a.x -a.y);
    }

long getValue() {
    return getSub();
    }
```

Abbildung 12 Software-Kapselung von Daten

Bei einer weiteren Definition der Kapselung bedient man sich der Hardware. Es wird bei Betrachtung des Arbeitsspeichers innerhalb eines Mikroprozessors ein sogenanntes MMU verwendet, welches verhindert, dass auf sichere Daten zugegriffen werden kann. Diagnose der MMU ist der zyklische Zugriff der sicheren Funktionen auf nicht sichere Daten. Ziel ist es den verweigerte Zugriff für einen erfolgreichen Test zu gewährleisten.

3.6.1.2 Beachtung beim Kontextwechsel

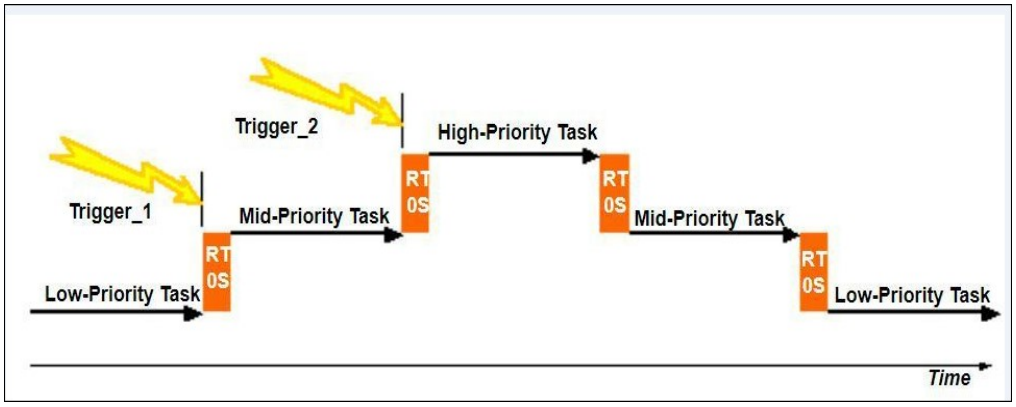

Abbildung 13 Beachtung beim Kontextwechsel

Als letzte Definition wird das Betriebssystem in der Kapselung erörtert. Es handelt sich hier um die Tasks und ihre Probleme beim Kontextwechsel. Dieses Thema wird nur kurz erörtert, da der CRC als Signaturcode gilt, wie oben diskutiert wird. Unter „Kontextwechsel“ [19] versteht man den Wechsel zwischen zwei Prozessen. Wie zu ersehen, müssen bei einem Wechsel alle relevanten Daten des Prozesses gesichert werden, bevor die CPU an den nächsten Prozess übergeben wird. Es handelt sich um ein preemptiv-Betriebssystem, wenn der Scheduler einen gerade laufenden Task an jeder Stelle anhalten kann, sobald er erkennt, dass ein anderer Task sofort ausgeführt werden muss. In der oberen Abbildung ist dies dargestellt als „Trigger_1“ und zeigt ein prioritätsbasierten preemptiven Scheduler, der in diesem Fall den Low-Priority Task und somit die Ausführung des aktuellen Assemblerbefehls bis zum nächsten Timer Tick gestattet. Danach hält der Scheduler sofort die Ausführung des Low-Priority Task an und lässt den Mid-Priority Task ablaufen. Wenn der Mid-Priority Task fertig abgelaufen ist, kann der Low-Priority Task weiter ausgeführt werden.Um die Sicherheitstechnik darzustellen, wird der Mid-Priority Task als Non-Safety-Task und der High-Priority Task als Safety-Task betitelt. Es kann nun vorkommen, dass der Non-Safety Task noch abläuft und dabei den Safety -Task bereit stellt. In der oberen Abbildung ist dies dargestellt als „Trigger_2. In diesem Fall handelt es sich um ein non preemptiv-Betriebssystem, welches den gerade laufenden Non-SafetyTask verdrängen würde und die Ausführung des Safety-Task ermöglicht. Es findet eine Überprüfung des Kontextes statt. Nach Beendigung der Safety-Task kann die Non-Safety Task weiter

[19][EZB52018]

ausgeführt werden und Sicherung der Variable kann über die CRC stattfinden. Der umgekehrte Fall bedeutet, dass wenn die Kontextsicherung im Scheduler läuft ein häufiges Unterbrechen der Safety-Task möglich ist. In diesem Fall ist die Sicherung der Variable über die CRC aus Zeitgründen nicht möglich. Jedoch können sich die Flags ändern und das Carry-Flag muss mitgesichert werden. Ebenso kann das Gleiche beim Hardwareinterrupt stattfinden.

3.7 Planung von Modul- und Systemdesign

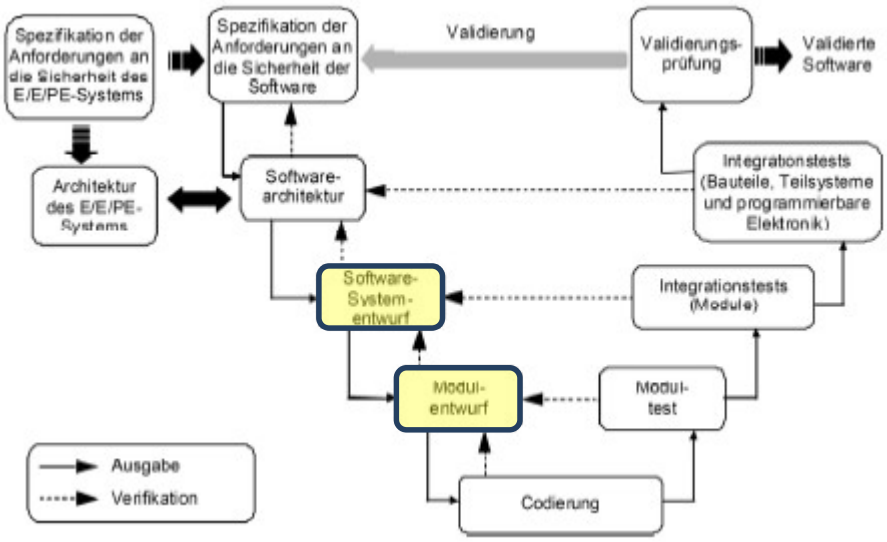

Abbildung 14 Software und Entwicklungslebenszyklus (Planung von Modul und Systemdesign)

3.7.1 Anforderung an die Feinplanung

In Kapitel 7.4.5[20] gilt der detaillierte Entwurf und wird hier als Softwareentwurf verstanden. Bei den Definitionen handelt es sich um die Aufteilung der Hauptelemente der Architektur, die ein System von Softwaremodulen, das Entwerfen von einzelnen Softwaremodulen und die Kodierung in Betracht ziehen. Chronologisch ist die Main in readInput(A), RAM-Check, ROM-Check und writeOutput() über die Top/Down Methode, Modularisieren, Testbarkeit, Spezifikation der Modultestfälle und Spezifikation der Intergrationstestfälle zu realisieren. Ein weiterer Faktor im Moduldesign ist die defensive Programmierung, die im nächsten Kapitel erörtert wird.

[20][DIN32010]

3.7.2 Defensive Programmierung

	Verfahren/Maßnahme *)	Siehe	SIL 1	SIL 2	SIL 3	SIL 4
1a	Strukturierte Methoden **)	C.2.1	++	++	++	++
1b	Semiformale Methoden **)	Tabelle B.7	+	++	++	++
1c	Formale Entwurfs- und Verfeinerungsmethoden **)	B.2.2, C.2.4	o	+	+	++
2	Rechnergestützte Entwurfswerkzeuge	B.3.5	+	+	++	++
3	Defensive Programmierung	C.2.5	o	+	++	++
4	Modularer Ansatz	Tabelle B.9	++	++	++	++
5	Entwurfs- und Programmierrichtlinien	C.2.6, Tabelle B.1	+	++	++	++
6	Strukturierte Programmierung	C.2.7	++	++	++	++
7	Verwendung bewährter/verifizierter Softwareelemente (wenn verfügbar)	C.2.10	+	++	++	++
8	Vorwärtsverfolgbarkeit von der Spezifikation der Anforderungen an die Sicherheit der Software zum Softwareentwurf	C.2.11	+	+	++	++

Abbildung 15 Tabelle Defensives Verfahren

Wie in Tabelle A4[21] gilt der Verweis C 2.5[22] für die defensive Programmierung. Die folgende Aufzählung zeigt einige der defensiven Verfahren: Bereich prüfen, Plausibilität prüfen und Prozedurparameter testen. Beim Bereich prüfen, sollten die Variablen auf ihren Bereich hin überprüfen. Bei der Plausibilität prüfen, sollten die Werte der Variablen geprüft werden. Bei Prozedurparameter testen, sollen die Prozedurparameter bei Einsprung in die Prozedur auf Typ, Dimension, und Bereich getestet werden. Zusätzlich sollten nur ROM- und RAM-Parameter getrennt und ihr Zugriff getestet werden.

[21][DIN32010]

[22][DIN72010]

3.8 Planung der Implementierung

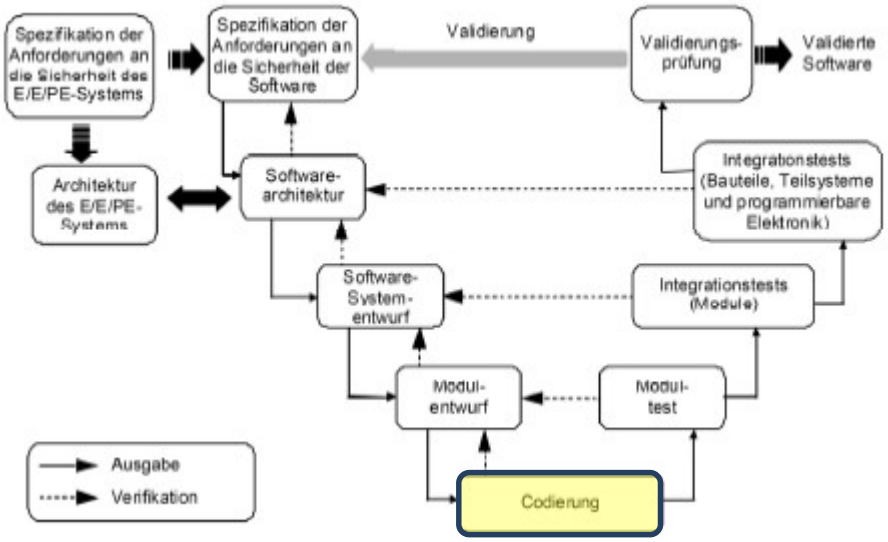

Abbildung 16 Software und Entwicklungslebenszyklus (Planung der Implementierung)

Wie in Kapitel 7.4.6[23] wird das Maß des erforderlichen SIL beschrieben. Dahingehend muss der Quellcode nachfolgende Eigenschaften besitzen: Der Quelltext soll lesbar, verständlich und prüfbar sein und die speziellen Anforderungen des Entwurfs der Softwaremodule, die speziellen Anforderungen der Programmierrichtlinie sowie alle wichtigen Anforderungen, die während der Sicherheitsplanung spezifiziert wurden, erfüllen.

3.8.1 Inspektion des Softwarecodes

Wie in Kapitel 7.4.6.1 sowie Tabelle B8 beschrieben, steht, dass jedes Modul des Softwarecodes inspiziert werden muss. Bei einer Inspektion durch eine Einzelperson handelt es sich um eine Bewertung von sicherheitsrelevanten Funktionen um die Konformität zur Spezifikation zu garantieren. Eine Code-Überprüfung kann durch das Code-Review mit

[23][DIN32010]

Hilfe individueller walk-through[24] (siehe Verweis C 5.15[25]) durch den Ersteller in Verbindung der Arbeitsgruppe

Verfahren / Maßnahme	Siehe	SIL 1	SIL 2	SIL 3	SIL 4	Strenge
Walk-Through	C 5.15	+	+	+	+	niedrig bis

Tabelle 1 Walk-Through

oder durch eine formale Überprüfung (siehe Verweis C 5.14) alleine durch die Arbeitsgruppe oder / und durch den strukturierten Prozess (Agenda, Start– und Abschlusskriterien und Klassifizierung der Mängel)

Verfahren / Maßnahme	Siehe	SIL 1	SIL 2	SIL 3	SIL 4	Strenge
Formale Inspektion	C 5.14	+	+	++	++	zu hoch

Tabelle 2 Formale Inspektion

in aufsteigender Reihenfolge der Strenge durchgeführt werden.

[24]Beim Walk-Through-Verfahren wird das Einhalten von Spezifikation und Ausführung aufgedeckt.

[25][DIN72010]

3.8.2 Referenzunterlagen und Fragenkatalog

Ein Review wird durch ein Team bestehend aus Moderator, Protokollant, Manager und Autor sowie Gutachter durchgeführt. Der Moderator leitet die Review-Sitzung, der Protokollant führt das Protokoll, der Manager ist der Projektverantwortliche und somit für die Freigabe verantwortlich, der Autor fungiert als Urheber der Software und der Gutachter fungiert als Sachverständiger der Prüfung des Design-Reviews in Hard- und Software. Ein Review muss immer hinsichtlich von konkreten Fragen, welche sich auf unterschiedliche Aspekte beziehen können, durchgeführt werden. Abhängig von den Aspekten, werden die Fragen von vorher definierten Personen im Rahmen einer Vorbereitung bearbeitet. Das Thema Treffsicherheit der Prüfverfahren sind Mängel in der Dokumentation, welche durch die Reviews gefunden werden können. Ein Abwägen zwischen Review und Test ist daher nur für das Prüfen von Programmen notwendig und können sowohl Code- als auch Entwurfsmängel enthalten. Die Mängel des Programms spiegeln entweder Mängel des Entwurfs wieder oder Fehler, die beim Umsetzen des Entwurfs in den Programmcode gemacht wurden. Die Mängel des eigentlichen Entwurfs werden höchstwahrscheinlich im Review nicht entdeckt und führen zu dem Kapitel Integrationstest. Bei den Arten von Codemängeln - siehe nächste Abbildung- gibt es eine gute Codeabdeckung im Code-Review. Dies ist zudem die einzige Möglichkeit Fehler in einem nicht dokumentierten Entwurf zu entdecken. Daraus ergibt sich folgende Aussage: Je weniger Entwurfsdokumente in einem Sicherheitsprojekt erstellt wurden, desto größer muss das Code-Review sein.

Arten von Codemängel	Code-Review	Black-Box	White-Box
fehlender Verarbeitungszweig	o	+	-
falsche Wahl des Verarbeitungszweigs	o	o	+
Falsche Berechnung	+	o	o
unvollständige Berechnung	+	o	-
falsche Schnittstelle	+	o	-
mangelnde Leistung	-	+	-
mangelnde Wartbarkeit	+	-	-

Legende: Entdeckung der Fehlerart	siehe[26]

Tabelle 3 Prüfverfahren von Codemängeln

3.9 Software Modul- Integrationstest

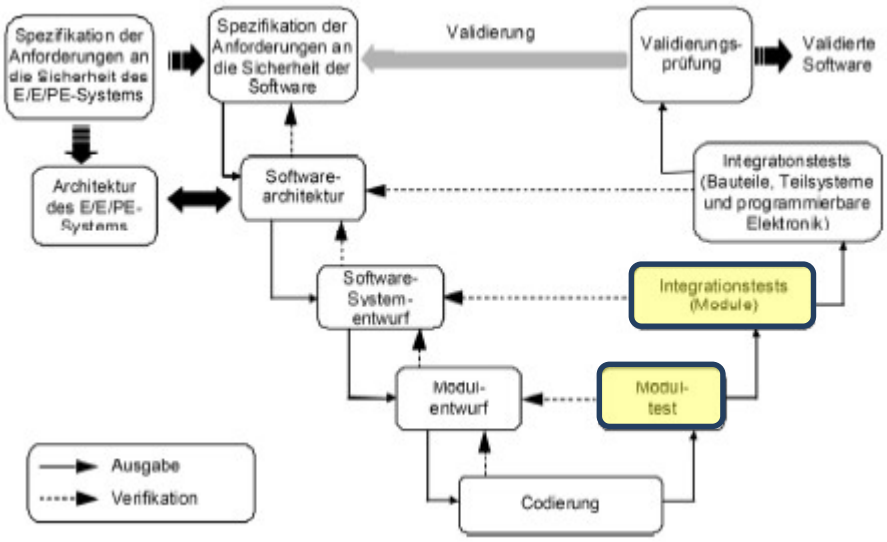

Abbildung 17 Software und Entwicklungslebenszyklus (Software Modul- Integrationstest)

3.9.1 Modultest

Im Allgemeinen gilt für das Modul, dass jedes getestet wird, die Ergebnisse dokumentiert werden müssen, das Verfahren für Korrekturen spezifiziert und angewendet werden muss und bei Änderungen immer eine Einflussanalyse durchgeführt werden muss. Das Resümee ist, dass jede Prüfung reproduzierbar sein muss. Wie in Kapitel 7.4.7[27] beschrieben,

[26][DIN32010] Anhang A

[27][DIN32010]

wird jede Anforderung der Modulspezifikation getestet. Der Code der Module wird in ihren Verzweigungen (Branch Coverage) verifiziert. Die Verzweigungsbedingungen (Condition Coverage) untersucht Testungen von Daten aus dem erlaubten Datenbereich, Testungen von Daten aus dem unerlaubten Datenbereich und Verifikation der Integrität der lokalen Variablen bei Interrupts sowie Aufrufe der Unterprogramme. Die angewendeten Methoden sind das Walk-Through, Testspezifikation entsprechend der Modulspezifikation, Ausführung der Testfälle und die Analyse der strukturellen Abdeckung. Es handelt sich um eine Whitebox[28]- Analyse, wenn die Abdeckung kleiner als 100 % beträgt. Als Anmerkung steht in der Richtlinie: Wenn ein Modul einfach genug ist, um einen vollständigen Test durchzuführen, dann kann es der wirksame Weg sein, eine Übereinstimmung nachzuweisen.

3.9.2 Software-Integrationstest

Wie in Kapitel 7.4.8 beschrieben, gilt für die Dokumentation der Software-Integrationstests, dass die Testergebnisse und deren Bewertung Gründe für Fehler sind. Die zugehörige Spezifikation beschreibt die Integrationsstrategie in ihren Prinzipien und Schritten, den Typ des Integrationstest, die Testdaten und ihre Testfälle, die Testendekriterien und die Verfahren, wenn Fehler gefunden werden. Da das Kapitel Softwaretest eine eigene Arbeit sein könnte, werden nur die wichtigsten Themen partiell diskutiert.

[28]Beim White-Box-Verfahren handelt es sich um die ermittelten Fehlersymptome beim dynamischen Verhalten in den inneren Funktionsbedingungen.

3.9.3 Test der Softwaremodule und Integrationstest

	Verfahren/Maßnahme *)	Siehe	SIL 1	SIL 2	SIL 3	SIL 4
1	Statistische Tests	C.5.1	o	+	+	+
2	Dynamische Analyse und Test	B.6.5, Tabelle B.2	+	++	++	++
3	Datenaufzeichnung und Analyse	C.5.2	++	++	++	++
4	Funktionstest und Black-Box-Test	B.5.1, B.5.2, Tabelle B.3	++	++	++	++
5	Leistungstest	Tabelle B.6	+	+	++	++
6	Modellbasiertes Testen	C.5.27	+	+	++	++
7	Schnittstellentest	C.5.3	+	+	++	++
8	Testmanagement und Automatisierungswerkzeuge	C.4.7	+	++	++	++
9	Vorwärtsverfolgbarkeit von der Spezifikation des Softwareentwurfs zu den Spezifikationen des Modul- und Integrationstests.	C.2.11	+	+	++	++
10	Formale Verifikation	C.5.12	o	o	+	+

Tabelle 4 Techniken der Softwaretests

Statistische Tests:

Ziel: Erreichen einer quantitativen Aussage über die Zuverlässigkeit der untersuchten

Software.

Funktionstest und Black-Box-Test[29]

Ziel: Unter realen Bedingungen zu testen und es wird überprüft wie sich die Software bei folgenden Eingangsparametern verhält:

• Daten aus zulässigen Bereichen

• Daten aus unzulässigen Bereichen

• Daten aus den Bereichsgrenzen

• Extremwerte

• Kombination dieser Klassen

[29] Beim Black-Box-Verfahren handelt es sich um das dynamische Verhalten unter realen Bedingungen und der Aufdeckung der Nichteinhaltung der Spezifikation.

	Verfahren/Maßnahme *)	Siehe	SIL 1	SIL 2	SIL 3	SIL 4
1	Durchführung von Testfällen nach einer Grenzwertanalyse	C.5.4	+	++	++	++
2	Durchführung von Testfällen aus der Fehlererwartung	C.5.5	+	+	+	+
3	Durchführung von Testfällen nach Fehlereinpflanzung	C.5.6	o	+	+	+
4	Testfallausführung durch modellbasierte Testfallgenerierung	C.5.27	+	+	++	++
5	Modellierung der Leistungsfähigkeit	C.5.20	+	+	+	++
6	Äquivalenzklassen und Test von Partitionen des Eingangsbereichs	C.5.7	+	+	+	++
7a	Strukturabhängige Tests mit einer Testabdeckung (Eingangspunkte) 100 % **)	C.5.8	++	++	++	++
7b	Strukturabhängige Tests mit einer Testabdeckung (Anweisungen) 100 % **)	C.5.8	+	++	++	++
7c	Strukturabhängige Tests mit einer Testabdeckung (Verzweigungen) 100 % **)	C.5.8	+	+	++	++
7d	Strukturabhängige Tests mit einer Testabdeckung (Bedingungen) 100 % **)	C.5.8	+	+	+	++

Tabelle 5 Testabdeckungen

Wie in der oberen Tabelle gezeigt, werden die einzelnen Maßnahmen über einen Kontrollflussgraph dargestellt. Ein Kontrollflussgraph beschreibt den Ablauf in einem Programm. Jede ausführbare Anweisung wird durch einen Knoten dargestellt. Ein Knoten ist hier z.B. ein Kreis. Der Kontrollflussgraph beginnt mit einem Startknoten und endet mit einem Endknoten. Diesen kompletten Kontrollflussgraph nennt man auch Pfad. Die einzelnen Knoten sind miteinander in Richtung des Programmablaufs verbunden. Diese Verbindungen nennt man Zweige.

Das Modul[30] soll in einem Intervall von 1 bis 5 die Anzahl der Primzahlen mitteln. Als Eingabe erhält sie den Startpunkt c im Intervall und die Variable (durchlaufAnz), womit die Durchlaufanzahl der Whileschleife kontrolliert werden kann.

[30][TUB2003]

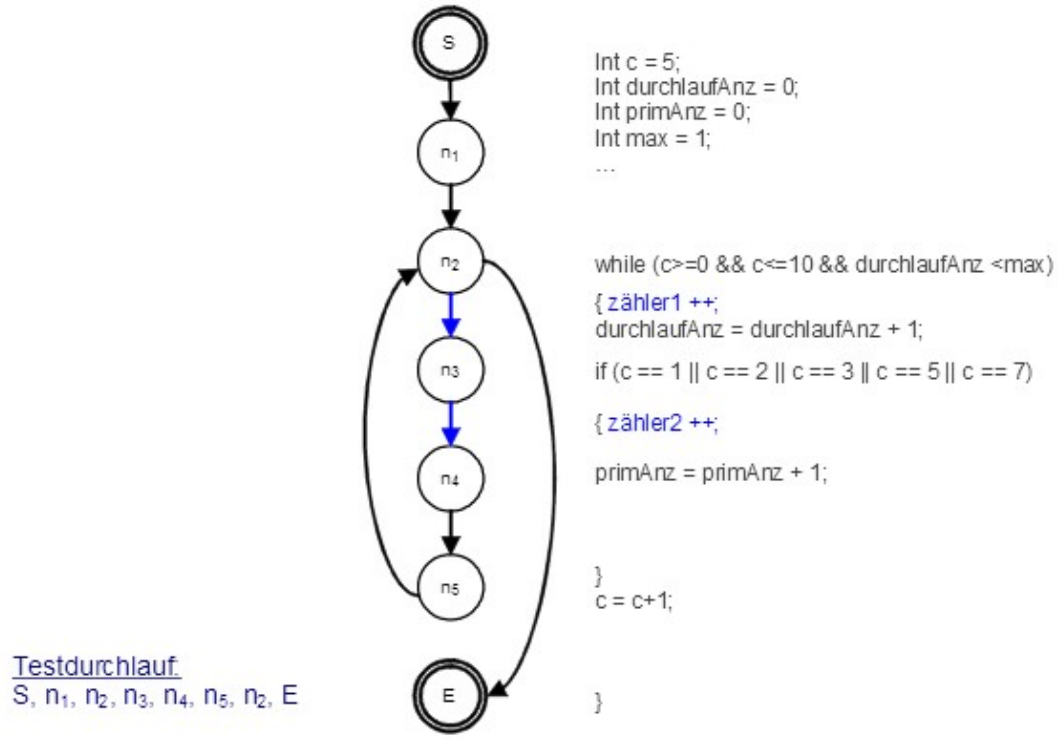

Abbildung 18 Anweisungsabdeckung

Bei der Anweisungsabdeckung werden alle Knoten eines Kontrollflussgraphen durchlaufen. Im oben dargestellten Beispiel wird die Anweisung n2, n3, n4 und n5 einmal durchlaufen. Danach ist die While-Schleife erfüllt. Der Testdurchlauf ist blau dargestellt. Dabei werden die Anweisungen mitgezählt, jedoch treten nicht alle möglichen Fehler in der Definition der Bedingungen auf.

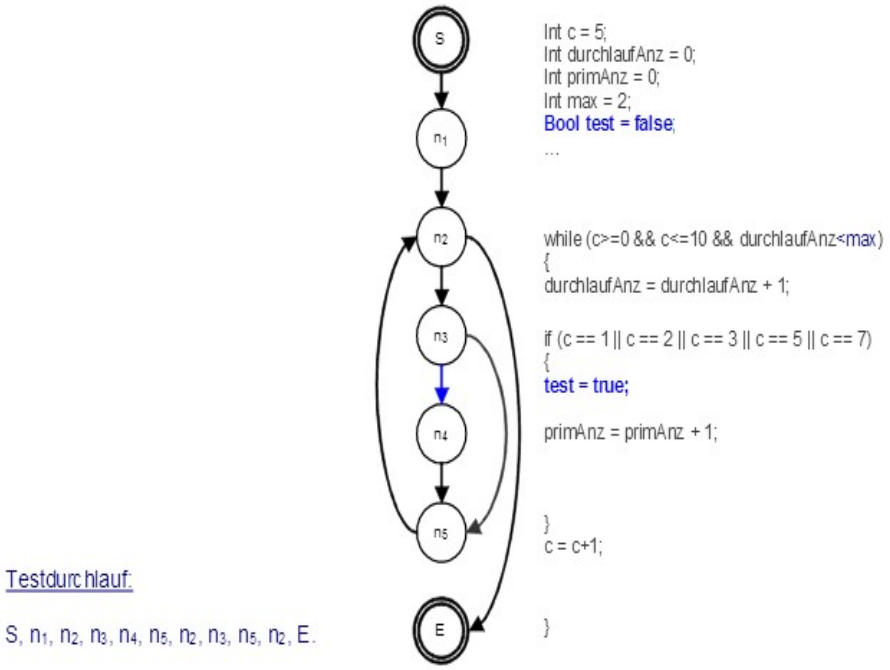

Abbildung 19 Zweigabdeckung

Bei der Zweigabdeckung werden alle Zweige eines Kontrollflussgraphen durchlaufen. Im oben dargestellten Beispiel wird die erste Anweisung n2, n3, n4 und n5 einmal durchlaufen. Danach wird die zweite Anweisung n2, n3, n5, und n2 durchlaufen bis die Bedingung erfüll ist. Der Testdurchlauf ist blau dargestellt. Dabei gibt es die Fehleraufdeckung bei test = true, da der zweite Durchlauf bei der Bedingung (c= 6) stoppt.

```
Int c = 1;
Int durchlaufAnz = 0;
Int primAnz = 0;
Int max = 8;

while (c>=0 && c<=10 && durchlaufAnz<max)

{
durchlaufAnz = durchlaufAnz + 1;

if (c == 1 || c == 2 || c == 3 || c == 5 || c == 7)

{

primAnz = primAnz + 1;

}

c = c+1;

}
```

Abbildung 20 Bedingungsabdeckung

Bei der Bedingungsabdeckung werden alle Bedingungen und ihre Kombinationen durchlaufen. Im oben dargestellten Beispiel wird die Bedingung (c==1,c==2,c==3,c==5,c==7)mit true durchlaufen, Dabei gibt es die Fehleraufdeckung bei der Bedingung (c==4,c==6), denn diese werden mit false durchlaufen.

Als Letztes werden die Methoden für den Funktionstest und Black-Box-Test nach der untenstehenden Tabelle erörtert.

	Verfahren/Maßnahme *)	Siehe	SIL 1	SIL 2	SIL 3	SIL 4
1	Funktionstest und Black-Box-Test	B.5.1, B.5.2, Tabelle B.3	++	++	++	++
2	Leistungstest	Tabelle B.6	+	+	++	++
3	Vorwärtsverfolgbarkeit von den Anforderungen an den System- und Softwareentwurf der Hardware-/Software-integration zu den Spezifikationen der Hardware-/Software-integrationstests	C.2.11	+	+	++	++

Tabelle 6 Testmethode Funktionstest

Bezogen auf die Tabelle 4 „Technik des Softwaretest“, werden die Methoden explizit mit Grenzwertanalyse und Äquivalenzklasse angesprochen. Für die Grenzwertanalyse[31] gilt: Das Programm, das mit Extremwerten gestresst wird, ist in Division durch Null, ASCII-Leerzeichen, leeres Stack oder leeres Listenelement, volle Matrix und leerer Tabelleneinsprung unterteilt. Für die Äquivalenzanalyse gilt: Eine minimale Auswahl mit dem Ziel von Eingabewerten, welche die Software komplett ausführt. Äquivalenzklassen sind ein Beispiel eines Kriteriums für die Festlegung von Black-Box-Testdaten. Der Eingabedatenraum wird anhand der Spezifikation in spezielle Eingabewertebereiche (Äquivalenzklassen) unterteilt.

[31][DIN72010]

4 Ergebnisse der Software

4.1 Softwareintegration

4.1.1 Hardware- und Softwareintegrationstest

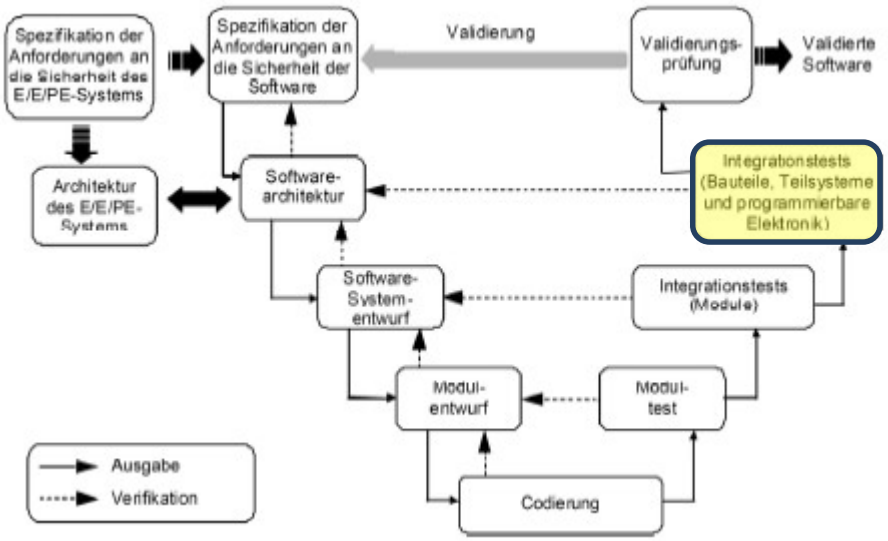

Abbildung 21 Software und Entwicklungslebenszyklus (Hardware- und Softwareintegrationstest)

4.1.2 Anforderung an die Sicherheitsfunktion

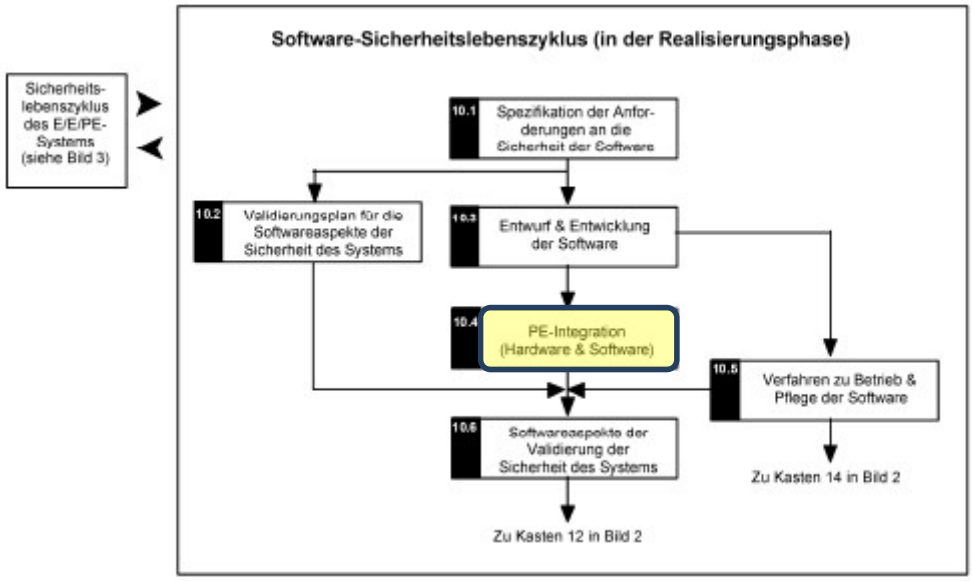

Abbildung 22 Integration im Software-Sicherheitslebenszyklus

Wie in Kapitel 7.5.2[32] beschrieben, gilt für die Spezifikation des Software-Integrationstests die zugehörige Hardware. Die Integrationsstrategie, die verfolgt wird, heißt: „Was wird integriert?“. Die Festlegung der Reihenfolge basiert auf die Bottom-Up-Methode oder Top-Down-Methode sowie der Sandwich-Methode. Weiter wird die Spezifikation mit Hilfe von Testarten durchgeführt, wie vorher im Kapitel beschrieben. Ferner sind diese in Testdaten und ihre Testfälle, Testumgebung, Werkzeuge, Konfiguration und Programme sowie dem Test-End-Kriterium unterteilt. Zusätzliche wichtige Aspekte sind die Verfahren von Korrekturen bei Nichtbestehen und das bei Änderung der Sicherheitsfunktion immer eine Einflussanalyse erbracht werden muss. Ein weiterer Punkt ist die Klärung wer verifiziert und wer validiert. Im Normalfall ist das der Hersteller und der Anwender. Ansonsten sind es die gleichen Bedingungen wie beim Modultest sowie im Integrationstest.

[32][DIN32010]

4.2 Softwarevalidation

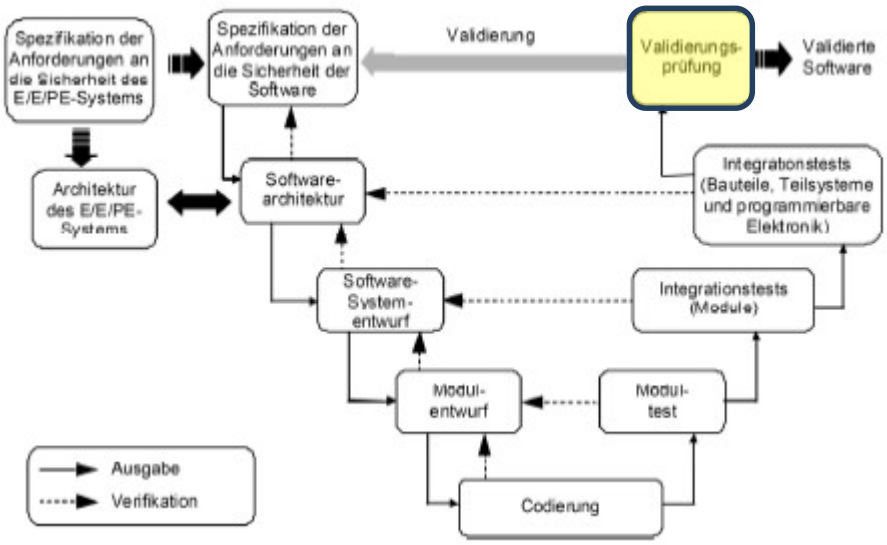

Abbildung 23 Software und Entwicklungslebenszyklus (Softwarevalidation)

4.2.1 Anmerkung an die Softwaresicherheitsvalidation

Ziel der Validation ist es nachzuweisen, dass das System der Software-Sicherheitsfunktion entspricht. Wie in Kapitel 7.2.2 beschrieben, gilt der Validierungsplan für die Softwareaspekte der Sicherheit des Systems. Dieser Validierungsplan diskutiert den Validierungstest unter realen Betriebsbedingungen, die Umsetzung der erwarteten Ereignisse und unter Fehlerbedingungen, welche in den sicheren Zustand führen. Weitere Diskussionsthemen sind im Thema Validation erörtert worden.

4.3 Softwaremodifikation

	Verfahren/Maßnahme	Siehe	SIL 1	SIL 2	SIL 3	SIL 4
1	Einflussanalyse	C.5.23	++	++	++	++
2	Neuverifikation geänderter Softwaremodule	C.5.23	++	++	++	++
3	Neuverifikation betroffener Softwaremodule	C.5.23	+	++	++	++
4a	Neuvalidierung des gesamten Systems	Tabelle A.7	o	+	++	++
4b	Validierung durch Regressionstest	C.5.25	+	++	++	++
5	Software-Konfigurationsmanagement	C.5.24	++	++	++	++
6	Datenaufzeichnung und Analyse	C.5.2	++	++	++	++
7	Vorwärtsverfolgbarkeit von der Spezifikation der Anforderungen an die Sicherheit der Software zum Plan der Softwaremodifikation (einschließlich Neuverifikation und Neuvalidierung)	C.2.11	+	+	++	++
8	Rückwärtsverfolgbarkeit vom Plan der Softwaremodifikation (einschließlich Neuverifikation und Neuvalidierung) zur Spezifikation der Anforderungen an die Sicherheit der Software	C.2.11	+	+	++	++

Tabelle 7 Softwaremodifikation

Wie die Tabelle A8[33] in Kapitel C. 5.23 beschrieben, gilt es als Ziel die Ermittlung der Auswirkungen, die eine Veränderung oder Verbesserung an einem Softwaresystem auf andere Softwaremodule in diesem Softwaresystem oder auch auf andere Systeme haben wird, zu erörtern.

4.3.1 Einflussanalyse

Daraus folgt aus der Softwaremodifikation: „Welchen Einfluss hat die Änderung auf die Sicherheit? Ist eine neue Risikobeurteilung von Nöten? Und welche Softwarelebenszyklusphase muss wiederholt werden?“. Die einzelnen Phasen sind in personelle Verantwortung, detaillierte Änderungsspezifikation, Verifikationsplanung und Umfang des Reengineerings der Validierung unterteilt. Die letzte Phase ist impliziert in den Tabellenpunkten 7 (Vorwärtsverfolgbarkeit) und 8 (Rückwärtsverfolgbarkeit) aufgeführt.

[33][DIN72010]

4.3.2 Dokumentation und Modifikation

Der letzte Teil des Kapitels betrifft die Dokumentation. Es ist eine detailgetreue Modifikationsbeschreibung zu tätigen, wobei die Fragen: „Welchen Einfluss hat die Programmteilmodifikation? Welchen Einfluss hat die Einflussanalyse auf das Konfigurationsmanagement? Gibt es eventuelle Abweichungen in verschiedenen Betriebsarten?“ auftreten. Alle diese betroffenen Themen erfordern eine Überarbeitung in ihren Dokumenten. Die auftretenden Fragen bleiben hier unbeantwortet, da es sich nicht um ein Projekt handelt, sondern nur um die technologische Struktur für eine Sicherheitstechnik in Bussystemen, welche schon im nachfolgenden Kapitel festgelegt sind.

5 Bussysteme in der Sicherheitstechnik

5.1 Bussysteme PROFIsafe

Der Anwendungsbereich[34] des PROFIsafe-Treibers wird im nachfolgenden Kapitel PROFIBUS und PROFINET mit sicherheitsgerichteten Feldgeräten zum Einsatz gebracht und erörtert. Der Begriff Feldgerät wird in den nachfolgenden Kapiteln F-Device bzw. F-Slave genannt. Damit das F-Device auch über den sicherheitsgerichteten Bus die F-Daten austauschen kann, braucht es einen Partner. Diesen findet es im IO-Partner, welcher mit Hilfe des Master- / Slave-Verfahrens die Kommunikation aufbaut. Obwohl es sich nicht um einen Rechner handelt, wird dieser Partner F-Host bzw. F-Master genannt. Damit der F-Datenaustausch funktioniert, wird ein F-Telegramm in Eingang und Ausgang angefordert, was später noch genauer beschrieben wird.

5.1.1 Sicherheitsgerichtete Einrichtung

In Anlage 1 wird die sicherheitsgerichtete Einrichtung ohne Bussystem sowie die hardwarenahen Auszüge, die als Ausfallwahrscheinlichkeit aus der vorherigen Arbeit[35] erörtert werden, beschrieben.

5.1.2 Sicherheitsgerichtete Busübersicht

In Anlage 2 wird die sicherheitsgerichtete Einrichtung mit Bussystem erörtert. Ein ganz wichtiger Punkt bei den Bussystemen in der Sicherheitstechnik ist das Logikschaltprinzip, das immer nach dem Ruhestromprinzip[36] arbeitet.

---

[34] [[Siem2010], [Siem2019]

[35] [Reich2016]

[36] Beim Ruhestromprinzip ist ein Kontakt als Öffner oder ein aktives Signal gemeint. Es verhindert beim Leitungsbruch, dass die Schutzmaßnahmen inaktiv werden.

5.1.3 Architekturübersicht

Allgemein ist das Protokoll weitgehend durch zwei Zustandsmaschinen (F-Host {A}) und (F-Device {B}) definiert, wobei die Kommunikation über die einzelnen Stacks zwischen dem Steuerrechner mit beliebigen Endgeräten stattfindet. Das definierte Nachrichtenformat wird zwischen den F-Geräten ausgetauscht.

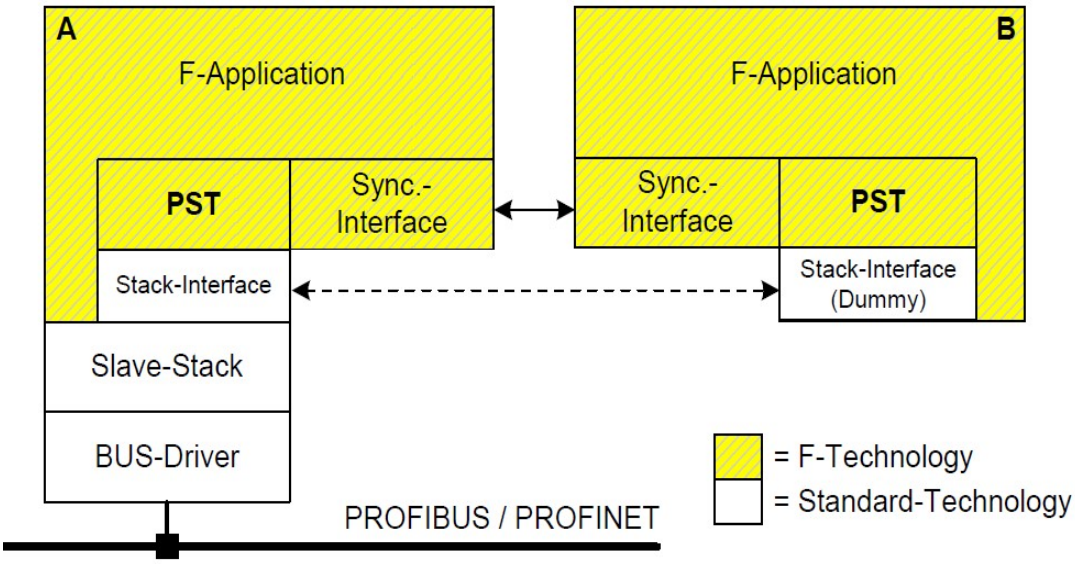

Abbildung 24 Architekturübersicht PROFIsafe

Die Aufgaben des BUS-Treibers ist die Steuerung der Kommunikation über PROFIBUS und PROFINET und wird in dem Kapitel über die OSI-Referenz definiert. Wenn man sich die obere Abbildung ansieht, fällt auf, dass es zwei Technologien gibt. Hier handelt sich um eine strenge Trennung zwischen der Standard-Technologie und der F-Technologie. Zunächst wird die F-Technologie definiert.

Die Aufgaben des Slave-Stack ist die Steuerung der Kommunikation von PROFIBUS und PROFINET. Ferner empfängt er die F-Parameter und optional die iParameter sowie die Ausgangstelegramme. Zusätzlich hat er die Aufgabe die Eingangstelegramme zu senden und die zyklischen Buseigenschaften wie Start, Stopp und Diagnose zu erkennen.

Die Aufgabe des Stack-Interface ist die Anpassung der PROFIsafe-Treiber -hier weiter genannt als PST- an die verschiedenen Slave-Stacks. Ferner löst er die F-Parameter und die Ausgangstelegramme aus den Slave-Stacks und reicht diese an den PST weiter. Des Weiteren steuert und empfängt er den PST, der jeweils auf die Anforderung eines Puffers zur Eingangstelegrammübernahme fungiert und beim Slave-Stack das Versenden dessen an den F-Host regelt.

Die Aufgaben der F-Applikation liegen außerhalb des PST und es hat die Nutzung über PROFIBUS und PROFINET des zyklischen F-Telegramms. Ferner steuert sie den Ge-

samtablauf, initialisiert den Slave-Stack, verteilt bei Zweikanaligkeit die F-Datenpakete auf beide Kanäle mit Hilfe des Synchronisations-Interface, übernimmt vom PST die Ausgangsdaten sowie das Controlbyte und generiert die Eingangsdaten sowie die Bits im Statusbyte und übergibt sie wieder.

Die Aufgaben des Synchronisations-Interfaces dienen zum Datenaustausch. Der Synchronisationsablauf findet im zyklischen Betrieb statt und wird wie der Funktionsweiser der asynchronen Synchronisation[37] dargestellt. Der Ablauf ist wie folgt zu sehen. Die Transferfunktionen werden im ersten Schritt über den PST mit den einzelnen Synchronisationsdaten zum jeweiligen anderen Kanal gesendet. Der Vergleich mit dem eigenen Kanal geschieht im zweiten Schritt, wobei der PST mit Hilfe der Synchronisation die empfangenen Daten liest. Bei unterschiedlichen Daten wird ein Hardwarefehler ausgegeben. Wichtig ist, dass Pufferdaten nur einmal gelesen werden dürfen und diese Daten somit nach der Synchronisationsübergabe an den PST zu löschen sind. Bevor die weiteren PST-Daten erläutert werden, muss eine Zuordnung zum Bus-Driver für PROFIBUS und PROFINET definiert werden.

5.1.4 Zuordnung zu PROFIBUS und PROFINET

	User program		Application ProfiBus		Application ProfiNet
7	Application Layer		ProfiBus DP, Protocol		ProfiNet, Protocol
6	Presentation Layer		not used		not used
5	Session Layer				
4	Transport Layer				TCP / UDP
3	Network Layer				IP
2	Data link Layer		Fielbus Data Link		Standard Fast Ethernet IEEE 802.3
1	physical Layer		Transmision Technology		
	OSI Referenz Model		OSI Implementation ProfiBus		OSI Implementation ProfiNet

Tabelle 8 OSI-Referenz Model

Die obere Tabelle beschreibt die OSI-Referenz mit seinen eigenen Schichten und die Implementierung von PROFIBUS und PROFINET. Die sieben definierten Schichten werden vorausgesetzt. Der PROFIBUS benutzt für sein Telegramm alle Schichten bis auf drei bis sechs und das PROFINET benutzt für sein Telegramm außer die Schichten fünf und sechs. Da der PROFIBUS und das PROFINET in Anlage 2 von seinen Regelwerken schon bestimmt worden sind, ist nur eine allgemeine Beschreibung der beiden Bussystemen von Bedeutung. Der PROFIBUS[38] hat eine unbegrenzte Teilnehmerzahl und Daten-

[37][Siem2010], [Siem2019]

[38][Kunb2019]

übertragungsraten zwischen 9,6 kbit/s und 500 kbit/s. Seine hierarchische Struktur ist in den Ebenen Sensoren und Aktoren sowie Feld- und Prozess unterteilt. Im Master-Slave-Betrieb wird mit dem Zugangsverfahren Token Passing gearbeitet, bei dem Slaves nur auf Anforderung des Masters auf den PROFIBUS zugreifen dürfen. Beim PROFINET werden TCP/IP und IT-Standards genutzt. Des Weiteren ist es RT-Ethernet-fähig und die Integration von Feldbussystemen wird ermöglicht. Aufgrund des modularen Aufbaus des PROFINET-Konzeptes kann der Anwender die Funktionalität entsprechend seiner Anforderungen wählen. Es gibt mehrere Möglichkeiten ein Gerät mit PROFINET direkt auszustatten. Die offensichtliche Variante ist den PROFINET-Stack selber zu entwickeln. Eine weitere Variante ist der Zukauf eines fertigen PROFINET-Stacks und eine dritte Möglichkeit ist der Zukauf eines Chips oder eines Mikroprozessors mit integriertem PROFINET-Stack. Nach dieser kleinen Exkursion ist nun das Thema PST-Daten aktuell. Diese Daten[39] werden für die Schnittstellenfunktion als „nicht veränderbar“ und „veränderbar“ beschrieben. Ferner sind die zertifizierten PST- Quellen mit ihren CRC-Werten ein wichtiger Bestandteil für die nachfolgenden Erörterungen und der PST ist nach dem ANSI-Standard C90[40] entwickelt. Da die weiterführenden Beschreibungen von PST auf dem englischen Standard basieren, wird dies ab hier PSD genannt.

Um den Übergang zum Qualifizierer[41] zu geben, wird ‘Table 3 Qualifier and IOPS of Input Submodule‘ erwähnt. Es werden 3 Eingangskanalsituationen dargestellt. Situation n.a.: Das Submodul ist nicht betriebsbereit (z. B. (Sub-) Modulfehler). Der E / A-Controller / Host setzt Q_Ch 0 auf Q_Ch n = 0 = auf „schlecht“. Situation good: Der Eingabewert im Prozessbildwert wurde aus dem tatsächlichen physikalischen Wert am Sensor generiert, siehe Abbildung 3 des Dokumentes. Situation bad: Der Eingabewert im Prozessbild entspricht nicht dem physischen Wert am Sensor. Weiteres ist in der Literatur nachzulesen.Ferner gilt für ‘Table 4 Qualifier and IOPS of Output Submodule ‘: Es werden 3 Ausgangskanalsituationen dargestellt. Situation n.a: Das Submodul ist nach dem Start nicht bereit. Der Host setzt Q_Ch 0 auf Q_Ch n = 0 = auf „schlecht“. Situation good: Der Wert als physikalisches Signal wurde aus dem tatsächlichen Prozessabbild generiert. Situation bad: Der Ausgabewert im Prozessabbild entspricht nicht dem physikalischen Wert des physikalischen Signals. Im nächsten Kapitel werden die einzelnen Fälle der Eingangs- und Ausgangskarten in ihren Qualifizierungen erörtert.

[39][Siem2010], [Siem2019]

[40][MISR2013]

[41][RIO2017]

5.1.5 Definition von Qualifizierer und Strukturen

Die Einbettung in das Submodul bezieht sich hier auf das Kapitel Architekturübersicht. Es werden anhand der Tabelle[39] ‘Requirements unabhängig von der Generiervariante‘ die einzelnen Qualifizierungsanforderungen aufgeführt und die Strukturrümpfe erörtert.

Im Speziellen gelten für eine 1 – Kanal F - Applikation die Anforderungen PSD_RQ_1_01 bis PSD_RQ_1_02. Für eine 2 – Kanal F - Applikation gilt die Anforderung PSD_RQ_2_01. Daraus folgt, dass die allgemeinen Anforderungen von PSD_RQ_0_01 bis PSD_RQ_0_20 gelten. Die Anforderung PSD_RQ_0_07 fällt aus dem Allgemeinteil raus, da die Qualifizierung der Entwicklungstools vom Hersteller des F-Slaves durchzuführen ist. Nachfolgend werden die wichtigsten Qualifizierungen erörtert: PSD_RQ_0_04:Es dürfen nur die PSD-Schnittstellenfunktionen verwendet werden, die in der Datei p_api.h deklariert sind; PSD_RQ_0_05: Es dürfen nur die PSD-Schnittstellenstrukturen verwendet werden, wenn diese in der Datei p_global.h deklariert sind; PSD_RQ_0_06: Für die Auswertung der PSD-Returns müssen die Defines aus p_global.h verwendet werden; PSD_RQ_0_10: Die F-Adresse muss in der F-Applikation hinterlegt sein. Ein Extrahieren der F-Adresse bzw. anderer Werte aus den F-Parametern ist unzulässig, PSD_RQ_0_11: Bei folgenden Returnwerten von psd_GetFOutData(...) muss die F-Applikation den sicheren Zustand für den Prozess mit F_OUTPUT_COMM_ERR, _OUTPUT_WD_TIMEOUT, F_OUTPUT_PASSIVATED und F_OUTPUT_NOT_OK sicherstellen; PSD_RQ_0_14: Die F-Applikation muss das Statusbyte.Bit_1 (Device_Fault) für mindestens 2 Wechsel der Monitoring-Number setzen usw.; PSD_RQ_0_16: Die beiden von der F-Applikation dem PSD übergebenen Zeitwerte müssen unabhängig voneinander generiert werden usw., PSD_RQ_0_18: Die F-Applikation muss geeignete Maßnahmen ergreifen, um unterschiedliche F-Ausgangsnutzdaten oder Steuerbytekennungen zwischen den beiden Kanälen (z. B. bedingt durch einen Fehler im Ausgabe-Speicherbereich) zu erkennen; PSD_RQ_0_19: Meldet der PST einen Kommunikationsfehler usw.; PSD_RQ_0_07: Nicht zulässige Returnwerte müssen in den Zustand HARD_FAIL der F-Applikation führen.

Um den Übergang zum Qualifizierer[40] zu geben, wird ‘Table 5 Cases for Qualifier = 1 (good)‘ erwähnt. Es werden 4 Kartenkanalsituationen dargestellt. Fall IO: Nur die Quelle der Qualifiziererinformationen darf den Qualifizierer auf “gut“ setzen. Fall in DI, AI: Der Prozessbildwert wurde aus dem tatsächlichen Wert als physikalisches Signal (PV) unter Berücksichtigung von Filter, Kommunikationszeit usw. generiert. Fall in DO, AO: Der Wert als physikalisches Signal wurde aus dem tatsächlichen Prozessbild unter Berücksichtigung der Signalverarbeitung, der Kommunikationszeit usw. erzeugt. Fall AI/AO: Die Genauigkeit des generierten Wertes entspricht der vom Hersteller definierten Genauigkeit.Ferner gilt für ‘Table 6 Cases for Qualifier = 0 (bad)‘. Es werden 3 Kartenkanalsituationen dargestellt. Fall All IO: Alle Komponenten im Kanalpfad dürfen den Qualifizierer auf

“schlecht“ setzen. Fall in DI, AI: Der Wert sollte nicht von der Host-Anwendung verwendet werden. Fall in DO, AO: Das physikalische Signal (SV) wird vom (Sub-) Modul angelegt.

5.1.6 Entwicklung

Bevor man die Entwicklung und ihre Aufgaben[39] erörtert, sind die Datentypen, Byte-Order und C-Compiler zu diskutieren. Bei den Datentypen dürfen nur unsigned char: 8 Bits, unsigned short: 16 Bits, unsigned long: 32 Bits und unsigned long long: 64 Bits verwendet werden. Bei dem Byte-Order können Microcontroller mit Little- oder Big-Endian Byte verwendet werden. Bei dem C-Compiler ist der PSD nach dem ANSI-Standard C90- wie vorher schon in der Beschreibung unter MISRA angegeben- entwickelt.

Wie in dem Kapitel Zuordnung zu PROFIBUS und PROFINET schon beschrieben, werden die nachfolgenden ‘*.c‘ und ‘*.h‘ Dateien mit CRC – Werten versehen und werden nur gegen die Lizenz vom Autor (Firma Siemens) freigeschaltet. Die benötigten Daten werden als Quell - und Inkludierdatei aufgeführt. Die c - Dateien sind in aufsteigender Reihenfolge aufgezählt: p_64bit.c, p_crc.c, p_c_fapplication.c, p_init.c, p_input.c, p_invers.c, p_output.c, p_param.c, p_state.c und p_time.c. Für die h – Dateien gelten:p_api.h, p_global.h, p_local.h, p_c_si.c, p_types.h, p_c_config.h, p_c_crc_m_h, p_c_fapplication.h und p_c_si.h.

Die einzelnen Schnittstellen werden anhand der Tabelle [42] „Nichtveränderbare Dateien“ berücksichtigt. Für die Datei p_crc.c gelten die Funktionen(psd_CRC16(…), psd_CRC24(…),psd_CRC32(…)); für die Datei psd_init gilt die Funktion psd_InitInstance(…); für die Datei p_state.c gelten die Funktionen (psd_GetState(…),psd_Stop(…),psd_Run(…)); für die Datei p_input.c gelten die Funktionen psd_SendFInTele(…),psd_SetFInData(); für die Datei p_output.c gelten die Funktionen psd_GetFOutData(…),psd_RecvFOutTele() und für die Datei p_param.c gelten die Funktionen (psd_GetFPar(…), psd_FParBuild(…), psd_Config(…)), die später im Kapitel Kompatibilität und Migration in ihren Aufrufen gezeigt werden. Die Dateien p_time.c, p_64bit.c und p_c_fapplication.c haben keine Funktionen und es gelten die Methoden in p_types.h, p_api.h, p_global.h und p_local.h. Für die, welche nach Tabelle ‘Veränderbare Dateien‘ arbeiten, gelten die Methoden p_c_config.h, p_c_crc_m.h, p_c_crc_b.h, p_c_fapplication.h und p_c_si.h

[42][Siem2010], [Siem2019]

5.1.7 RIOforFA mit PROFIsafe

In diesem Thema[43] werden die Definitionen für die Zustände „gut“ und „schlecht“ für das RIOforFA bestimmt. Für die Karten in DI, AI gilt: Der Wert an der PROFIsafe-Schnittstelle ist ungültig. Der Wert im Prozessabbild muss ein eigensicherer Wert sein. Für die Karten in DO, AO gilt: Der Wert als physikalisches Signal muss ein eigensicherer Wert sein. Für „einen Qualifizierer gleich schlecht“ gilt auch die Kanal- (Neu-) Aktivierung, wenn dieser nicht bestätigt" wird oder ein PROFIsafe-Fehler ansteht.

Ferner wird ein Kommunikationsfehler in PROFIsafe-Host und PSD behandelt. Durch eine Quittierung im Fehlerfall, wenn ein (Sub-) Modul aus- und wieder eingeschaltet wird, ist der PROFIsafe-Host für den Neustart des Sicherheitsbetriebs verantwortlich, indem er auf die Bestätigung wartet. Es gelten folgende Regeln: Die Bestätigung auf einem Kanal ohne Fehler darf den Kanalstatus “gut“ nicht ändern; die Bestätigung auf einem Kanal mit einem vorhandenen Fehler darf den Kanalstatus “schlecht“ nicht ändern; die Bestätigung auf einem Kanal mit einem vorherigen (z. B. reparierten) ändert den Kanalstatus von "schlecht" in "gut". Mit Hilfe von Zustandsautomaten lassen sich diese Kanalqualifizierer darstellen.

Im ersten Fall / Modus kann der Kanal jederzeit den Sicherheitsvorgang starten und auf “Kanal gut“ umschalten und muss nicht auf eine Benutzerbestätigung warten. Dieser Fall bezieht sich auf Anwendungen, für die kein ChF_Ack erforderlich ist. Dieser Fall wird auch als "Kanal mit automatischer Bestätigung" oder "Kanal ohne Bestätigung" bezeichnet. Dieser Fall ist eine Funktion des Geräts / Kanals und diese Funktion kann z.B. eine Parametrierung sein. Wenn die Kanaldiagnose einen Fehler feststellt, wechselt der Kanal in den Zustand “Kanal schlecht“. Wenn die Kanaldiagnose keinen Fehler feststellt, schaltet sie den Kanal ohne erforderliche Benutzerbestätigung in den Status “Kanal gut“.

[43][RIO2017]

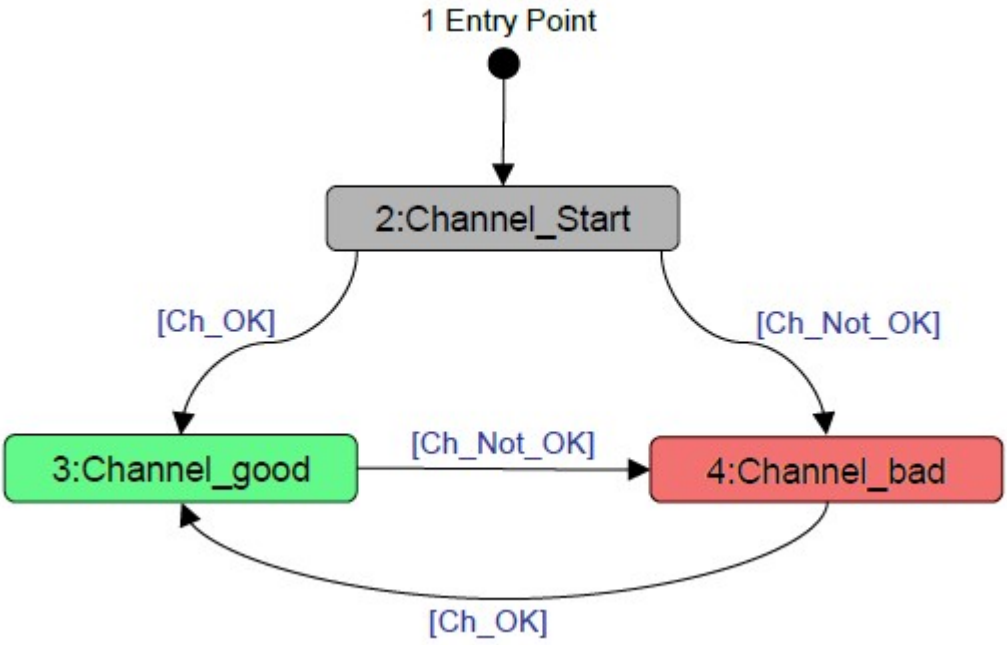

Tabelle 9 Zustandsautomat Keine Benutzerbestätigung erforderlich

In zweiten Fall / Modus ist der Neustart des Sicherheitsbetriebs nur durch Benutzerbestätigung zulässig. In diesem Diagramm wird ein zusätzlicher Status "Ch_unacknowledged" (Kanal nicht bestätigt) definiert. Befindet sich der Kanal im Zustand “Kanal schlecht“, prüft die Kanaldiagnose, ob ein Kanalfehler vorliegt (oder ein Kanalfehler aufgetreten ist). Dann schaltet der Übergang "Ch_OK" den Kanal in den Zustand "Ch_unacknowledged" und setzt das Statusbit ChF_Ack_Req auf 1. Danach kann der Benutzer den Kanal in den Zustand “Kanal gut“schalten, indem er das Steuerbit ChF_Ack von 0 auf 1 setzt.

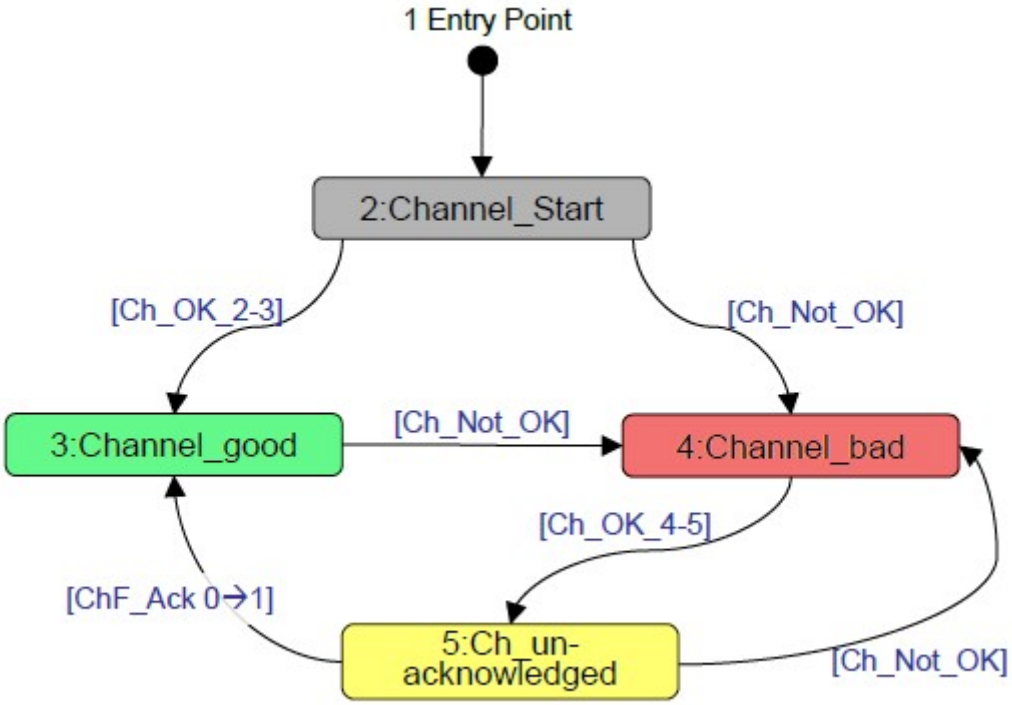

Tabelle 10 Zustandsautomat Benutzerbestätigung durch ChF_Ack_Req erforderlich

Der Unterschied zur vorherigen Zustandsmaschine ist ohne ChF_Ack_Req. Die Dauer im Status „5: Neustartdiagnose“ muss auf die Prozesssicherheitszeit begrenzt sein, um einen unerwarteten Neustart nach der Bestätigung durch den Benutzer zu vermeiden. Diese Anforderung wird durch das Timeout „CDTO“ (Channel DiagnosticsTimeOut) erfüllt (siehe Abbildung in [RIO2017]). CDTO muss so eingestellt sein, dass die Sicherheitszeit verarbeitet wird. Der Beispielwert eines typischen CDTO ist 2 × F_WD_Time.

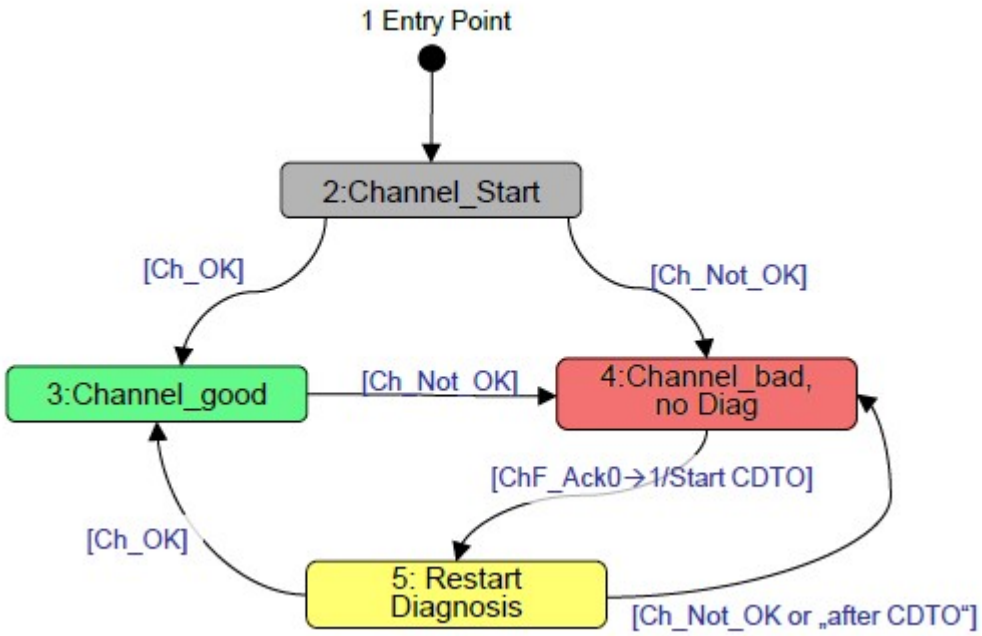

Tabelle 11 Zustandsautomat Benutzerbestätigung ohne ChF_Ack_Req erforderlich

Aus den Algorithmen der Zustandsautomaten können Zeitdiagrame abgeleitet werden. Diese Diagramme sind informativ und können verschiedene Fehler anzeigen. Die Parameter, die ein Diagramm aufweist, sind in der Gruppe Benutzerdaten und Kommunikationsdaten beschrieben. Die Diagramme A bis G besitzen in der Spalte für PROFIsafe den Level (“gut“/“schlecht“), den Eingangsstatus des Kanals mit dem Level (“Fehlerwert“/“Prozesswert“), den Ausgangsstatus des Kanals mit dem Level (“Fehlerwert“/“Prozesswert“),die PROFIsafe-Benutzeranforderung mit dem Level (“nicht aktiv“/“aktiv“),die PROFIsafe Quittierung mit dem Level (“nicht aktiv“/“aktiv“), die Fehlerkanalbenutzeranforderung mit dem Level (“nicht aktiv“/“aktiv“), die Fehlerkanalquittierung mit dem Level (“nicht aktiv“/“aktiv“),die Eingangs-/ Ausgangskanaldiagnose mit dem Level (“gut“/“schlecht“), den Eingangsstatus des Kanals mit dem Level (“Fehlerwert/schlecht“/“Prozesswert/gut“),den Ausgangsstatus des Kanals mit dem Level (“Fehlerwert/schlecht“/“Prozesswert/gut“). Die untere Abbildung zeigt das Diagramm A mit der Benutzerquittierung (User Ack) und der Fehlerkanalbenutzeranforderung (ChF_Ack_Req**).** Das Diagramm zeigt, dass die Diagnose einen Eingangskanalfehler erkennt und keinen Ausgangskanalfehler besitzt.

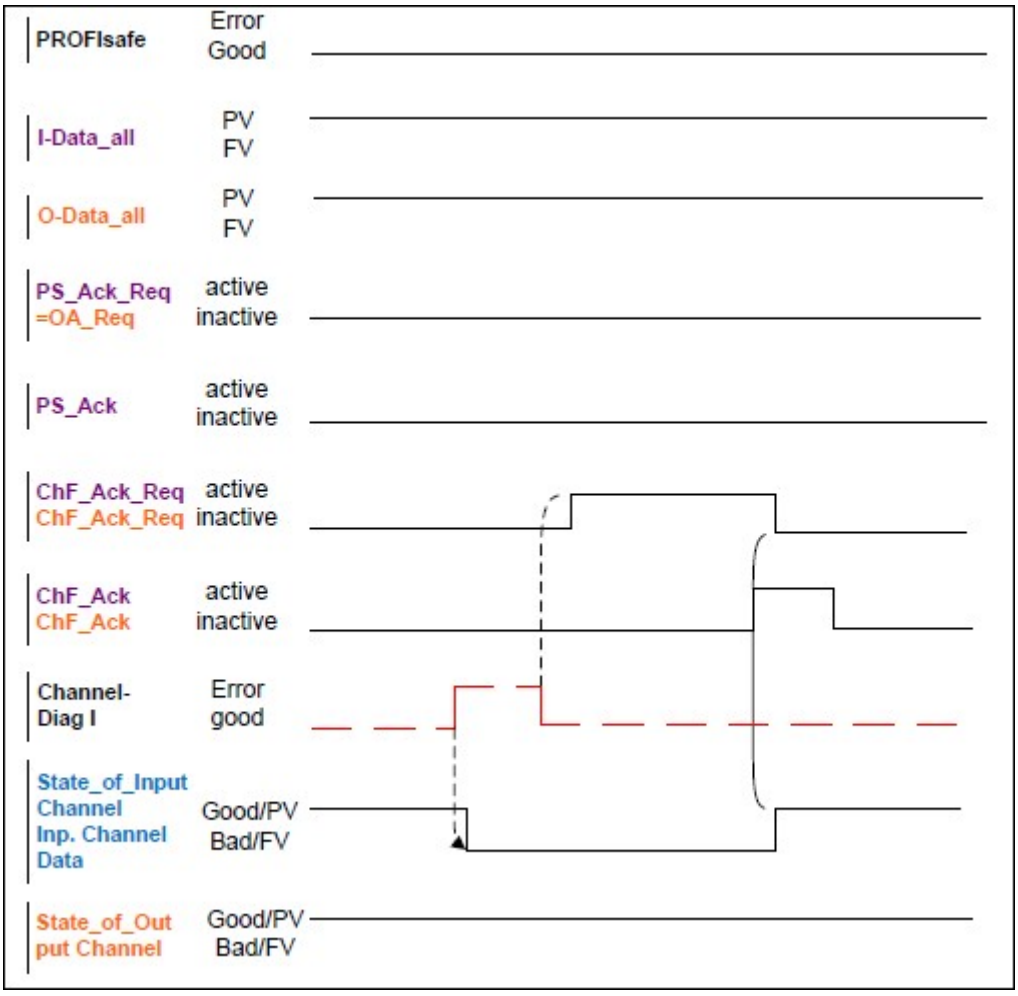

Abbildung 25 Benutzerquittierung (User Ack) und der Fehlerkanalbenutzeranforderung (ChF_Ack_Req)

Die Abbildung zeigt wie eine Eingangskanaldiagnose(Channel-Diag I) gestoppt werden kann, wenn die Fehlerkanalbenutzeranforderung (ChF_Ack_Req**)** „high“ wird. Erst bei der Fehlerkanalquittierung (ChF_Ack) von “low“ auf “high“ wird der Eingangskanalfehler auf “high“ gesetzt und somit quittiert. Weitere Beispiele sind im Dokument[44] nachzulesen.

Bei den fehlerbeherrschenden Maßnahmen[45] müssen nach dem vorherigen Kapitel Anmerkung an die Softwaresicherheitsvalidation innerhalb einer sicherheitsgerichteten Datenkommunikation Übertragungsfehler, Wiederholung, Verlust, Einfügung, falsche Abfolge, Nachrichtenverfälschung, zeitliche Verzögerung und Maskierung aufgenommen und

[44][RIO2017]

[45][FBus2001]

beherrscht werden. Bei einem Übertragungsfehler handelt es sich um eine Signalverfälschung, welche eine Wiederholung darstellt, wenn ein Signal fälschlicherweise innerhalb eines Zeitintervalls noch einmal gesendet wird. Ein Verlust ist es, wenn ein Signal durch einen Fehler komplett gelöscht wird. Eine Einfügung ist ein verfälschtes Signal innerhalb einer Nachricht. Eine falsche Abfolge beschreibt die Information einer Nachricht, die zum späteren Zeitpunkt beim Empfänger ankommt. Eine Nachrichtenverfälschung ist ein Fehler im PSD der unzulässig verfälscht wird. Durch einen Fehler tritt eine Zeitverzögerung beim Empfänger sowie die Maskierung bei einer Adressverfälschung sicherer und nicht sicherer Busteilnehmer auf. Das nächste Kapitel beschreibt die Zusammensetzung des PROFIsafe Nachrichtentelegramms und die Berechnung der Restfehlerrate.

Die sichere Kommunikation zwischen F-Host und F-Slave ist im Aufbau des F-Telegramms in der unteren Abbildung zu erkennen. Hier werden das Eingangstelegramm mit dessen Statusbyte und des CRCs sowie das Ausgangstelegramm mit dessen Steuerbyte und des CRCs dargestellt.

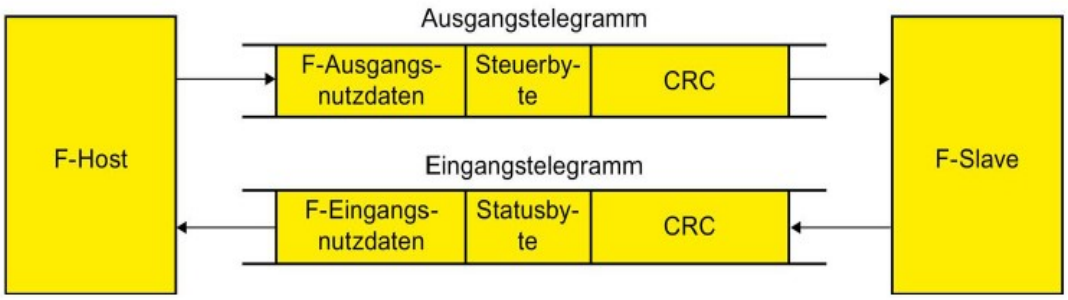

Abbildung 26 Aufbau der F- Telegramme

Wie vorher schon angesprochen werden die F-Nachrichten[46] zwischen einem F-Host und seinem F-Device als Nutzfracht in PROFIBUS- oder PROFINET-Telegrammen transportiert. Im Fall eines modularen F-Slave mit mehreren F-Modulen besteht die Nutzfracht aus mehreren F-Nachrichten.In der unteren Tabelle wird dies gezeigt und als Grundlage der nachfolgenden Berechnungen berücksichtigt.

F-Input/ F-Output Data	Status-Steuerbyte	CRC
Sichere Prozessdaten max. 12 oder 123	1 Byte Informationen	CRC24 oder CRC32

Tabelle 12 PROFIsafe Nachrichtenformat

[46][Syst2007]

Der Eingangs-CRC wird über F-Input Data, dessen Statusbyte und den zugehörigen F-Parametern berechnet. Dagegen wird der Ausgangs-CRC über den F-Output Data, dessen Steuerbyte und den zugehörigen F – Parametern berechnet. Der CRC24 kommt bei Prozessdaten bis zu 12 Bytes Länge und der CRC32 bei Prozessdaten bis zu 123 Bytes Länge zum Einsatz. Eine Berechnung der zyklischen Blockprüfung (CRC) dient zur Erkennung von fehlerhaften Datenbits. Für weitere Ausfallberechnungen werden die notwendigen Betrachtungen der Restfehlerwahrscheinlichkeit in den Festlegungen der DIN EN 61508 herangezogen, die die zulässige Fehlerwahrscheinlichkeit von Sicherheitsfunktionen beziffert. PROFIsafe orientiert sich an den Festlegungen dieser Norm. Die untere Abbildung ist definiert mit einen SIL 3 und dessen Ausfallwahrscheinlichkeit von$10^{-7}/_{h}$. Mit Hilfe eines ausgewählten 32 Bit CRC – Generatorpolynom wird die gefährliche Ausfallwahrscheinlichkeit $10^{-9}/_{h}$ durch die 1%-Regel sichergestellt.

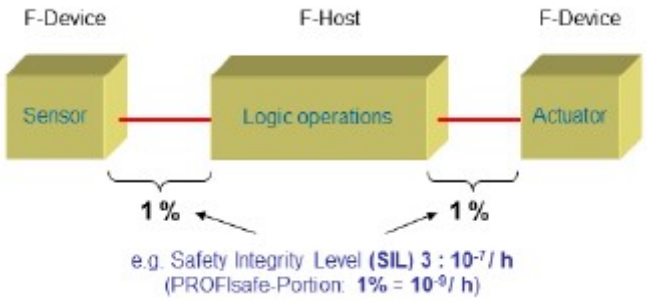

Abbildung 27 1 %-Regel

Nach dem Ansatz [DIN32010] gilt für ein binär-symmetrisches Übertagungssystem pro Kanal eine Restfehlerwahrscheinlichkeit pro Stunde. Bei einem Modell der Gaußstörung, bedient man sich der Hilfe einer Bernoulli-Verteilung.

p = Bitfehlerwahrscheinlichkeit

$$R(p,n,k) \approx \frac{1}{2^k}\sum_{e=d}^{n+k}\binom{n+k}{e}p^e(1-p)^{n+k-e}$$

e = Anzahl der Fehler

k = Anzahl der Prüfsummenbits

n = Anzahl der Datenbits

d = Hamming Distanz

Tabelle 13 Bestimmung der Gaußstörung

R = Restfehlerwahrscheinlichkeit

$$\Lambda = 3600 * R * \mathrm{v} * (\mathrm{m} - 1)$$[47]

m = Anzahl der sicheren Busteilnehmer

v = Anzahl der Nachrichtenzyklen pro Sekunde

Tabelle 14 Berechnung der Restfehlerwahrscheinlichkeit

Wie oben schon durch die gefährliche Ausfallwahrscheinlichkeit erwähnt ist, ist noch die Frage nach der Ausfallrate λ offen. Um die Ausfallrate λ zu erzielen, bedient man sich -wie oben genannt- der Restfehlerwahrscheinlichkeit. Liefert λ_D 1% von PFD, dann kann das Bussystem bei der Betrachtung des gefährlichen Ausfalls λ_{DD} vernachlässigt werden.

5.1.8 Kompatibilität und Migration

Wie im vorherigen Kapitel schon erwähnt, werden hier die PSD-Schnittstellenfunktionen[48] in ihren Funktionen mit Prototypen, dessen Übergabeparameter und Rückgabeparameter definiert.

Die Funktion psd_CRC32() besitzt den Prototyp psd_CRC32(P_Dword psd_CRC32, P_DWordLength, P_DWordStartValue, const P_Byte_Data_Ptr[], P_ByteCalcDirection). Die Übergabeparameter Length, StartValue, Data_Ptr und CalcDirection beschreiben die Berechnungslänge, den Berechnungsstartwert, den Zeiger dessen und die Berechnungsrichtung. Der Rückgabeparameterals P_Dword gibt einen 32Bit-CRC-Wert an.

Die Funktion psd_InitInstance() besitzt den Prototyp P_Bytepsd_Intistance(P_WordInst_ID, P_ByteDynamicTeleLenConf). Die Übergabeparameter Inst_ID und DynamicTeleLenConf beschreiben die Instanz sowie die Konfiguration mit und ohne die Funktion psd_Config(). Der Rückgabeparameter als P_Bytegibt an, ob der Treiber instanziiert wurde oder nicht.

Die Funktion psd_GetState() besitzt den Prototyp P_Bytepsd_GetState(P_Bytepsd_GetState(P_WordInst_ID). Der Übergabeparameter

---

[47][FBus2001]

[48][Siem2010], [Siem2019]

Inst_ID beschreibt die Instanz. Der Rückgabeparameter als P_Byte gibt an, ob der Treiber instanziiert, parametriert oder im zyklischen Austausch ist.

Die Funktion psd_GetFPar() besitzt den Prototyp P_Bytepsd_GetFPar(P_WordInst_ID, FwPsdFPar*FwFParamPtr). Die Übergabeparameter Inst_IDund FwFParamPtr beschreiben die Instanz sowie den Zeiger auf die F-Parameterstruktur.Der Rückgabeparameter als P_Byte gibt an, ob der Aufruf korrekt oder nicht korrekt ist.

Die Funktion psd_Stop() besitzt den Prototyp P_Bytepsd_Stop(P_WordInst_ID). Der Übergabeparameter Inst_ID beschreibt die Instanz. Der Rückgabeparameter als P_Byte gibt an, ob der Aufruf korrekt ist oder einen hardwaremäßigen Fehler hat.

Die Funktion psd_Run() besitzt den Prototyp P_Bytepsd_Run(P_WordInst_ID). Der Übergabeparameter Inst_ID beschreibt die Instanz. Der Rückgabeparameter als P_Byte gibt an, ob ein zyklischer Datenaustausch gestartet werden kann, ob eine F-Parametrierung fehlt oder ein hardwaremäßiger Fehler vorliegt.

Die Funktion psd_GetFOutData() besitzt den Prototyp P_Bytepsd_GetFOutData (P_WordInst_ID, Fw_OutputDataCB P_GLOBDAT_ATTR_POINTER*,FwOutputDataCB_Ptr, P_Byte P_GLOBDAT_ATTR_POINTER* OutDataArray, P_ByteOutDataLen). Die Übergabeparameter Inst_ID,FwOutputDataCB_Ptr, OutDataArray, OutDataLen beschreiben die Instanz, die Steuerbyteerkennung, die Ablage der F-Ausgangsnutzdaten und die Anzahl der F-Ausgangsnutzdaten. Der Rückgabeparameter als P_Byte gibt an, ob ein Toogle gesteuert wird, ob eine Passivierung ausgeführt wurde oder ob eine Inkonsistenz vorliegt.

Die Funktion psd_RecvFOutTele() besitzt den Prototyp P_Byte psd_RecvFOutTele(P_WordInst_ID, P_WordClockValue, P_Word ClockValueC2). Der Übergabeparameter Inst_ID, ClockValue, ClockValuC2 beschreibt die Instanz und die 1 ms Zählerwerte zur jeweiligen CPU. Der Rückgabeparameter als P_Byte gibt an, ob das Ausgangstelegramm übernommen wurde.

Die Funktion psd_SendFInTele() besitzt den Prototyp P_Byte psd_SendInTele(P_WordInst_ID). Der Übergabeparameter Inst_ID beschreibt die Instanz. Der Rückgabeparameter als P_Byte gibt an, ob das F-Eingangstelegram am Stack-Interface übergeben wird.

Die Funktion psd_SetFInData() besitzt den Prototyp P_Byte psd_SetFInData(P_WordInst_ID, constFw_InputtDataSB P_GLOBDAT_ATTR_POINTER*,FwInputDataSB_Ptr,constP_Byte P_GLOBDAT_ATTR_POINTER*, InDataArray,,P_ByteInDataLen). Der Übergabeparameter Inst_ID, FwInDataSB_Ptr, InDataArray, InDataLen beschreibt die Instanz, die Steuerbyteerkennung, die Ablage der F-Eingangsnutzdaten und die Anzahl der F-

Eingangsnutzdaten. Der Rückgabeparameter als P_Byte gibt an, ob die Daten übernommen wurden.

Die Funktion psd_Config () besitzt den Prototyp P_Byte psd_Config (P_WordInst_ID). Der Übergabeparameter Inst_ID beschreibt die Instanz. Der Rückgabeparameter als P_Byte gibt an, ob die Konfiguration erfolgreich war.

Die Funktion psd_FParBuild() besitzt den Prototyp P_Byte psd_FParBuild(P_WordInst_ID, Fw_FParFwPar, P_Word psd_CRC3). Der Übergabeparameter Inst_ID, FwPar, beschreibt die Instanz, den Aufbau der Geräteparameterstruktur und den 16iParameter-CRC. Der Rückgabeparameter als P_Byte gibt an, ob die Zieladressen sowie Quelladressen ungültig sind, der Watchdog ungültig ist, die parametrierte SIL –Klasse größer ist als die F – Applikation dem PSD übergeben hat oder ein F- Parameter-CRC Fehler vorliegt.

In Anlage 3 wird die Migration in Form von Codeauszügen der genannten Rumpf-Funktionen im Programmeditor gezeigt. Der Code-Schnipsel bedient sich der Referenz [Syst2007]. Hier wird die PSD - Schnittstelle in den wichtigsten Abarbeitungen der F - Applikation für die Stackbearbeitung gezeigt und als CRC – Generatorpolynom eine ausfallsichere Software für die Bustechnik, die mit Hilfe des PROFIsafe-Telegramms fungiert, umgesetzt. Es handelt sich hier nicht um einlaufendes Programm sondern es zeigt nur wie die einzelnen Prototypen aus den Regelwerken zu einem sicheren Ablauf benutzt werden können, da es sich hier um Firmengeheimnisse handelt.

6 Ergebnisse und Ausblick

6.1 Ergebnisse

Die Intension dieser Diplomarbeit [DIN32010] basiert auf die Weiterbehandlung meiner ersten Diplomarbeit [Reich2016] aufgrund des Software Designs, was nur partiell angesprochen wurde und hier ausgiebig über das V-Modell diskutiert wird. Bei den Bussystemen als ausfallsichere Systeme gilt nach der Programmiersprache für die Entwicklung von sicherheitsbezogener Software die Übereinstimmung mit angemessenen Programmierrichtlinien. Eine Programmiersprache muss gute Programmiertechniken beschreiben, unsichere Sprachmerkmale aufweisen, die Codeverständlichkeit fördern, die Verifikation und Tests erleichtern und die Verfahren für die Dokumentation des Quellcodes beschreiben. Wo diese durchzuführen ist, müssen mindestens folgende Informationen im Quellcode enthalten sein: Die juristische Person, die Beschreibung, die Ein- und Ausgaben sowie die Vorgeschichte des Konfigurationsmanagements. Speziell die Sicherheits-Integrität von Bussystemen, die einen Level von SIL 3 als Minimum haben, müssen einen Compiler mit einem Zertifikat eines technischen Überwachungsvereins aufweisen und bei der Empfehlung des eingebetteten Systems keine Programmiersprache ’C’ wie Tabelle C1 (Empfehlungen für bestimmte Programmiersprachen) nach [DIN72010] zu Verfügung stellen, da diese Möglichkeiten wegen der Zeigerarithmetik korrupte Speicherbelegungen und Pufferüberläufe im Normalfall haben. Dies zeigt zum Einen Tabelle A4 (Softwareentwurf und Softwareentwicklung – Detaillierter Entwurf als defensive Programmierung), die im oberen Kapitel schon erörtert wurde. Zum Anderen zeigen dies Tabelle B1 (Entwurfs- und Programmierrichtlinien) wie Einschränkung von Verwendungen von Zeigern, Einschränkung von Verwendungen von Interrupts und Einschränkung von Verwendungen von Rekursionen fungieren. Weitere Beispiele sind in Tabelle B7 (Semiformale Methoden), Tabelle B8 (Statische Analyse) sowie Tabelle B9 (Modularer Ansatz) hinterlegt. Die nächst höhere Ebene ist die, der diskreten Ereignisse, wo Register der Übertragung, der logischen Schaltungen sowie für RAM-Prüfungen definiert sind. Der Entscheidungscode liegt in der Programmiersprache “C“ im Modell an oberster Stelle und ist in der heuristische Ebene zu finden. Die physikalische Situation des Verhaltens über die universelle Programmiersprache mit den [MISRA2013] Werkzeugen ist in Blöcke modelliert. Die Sicherheitslogik wird mit Hilfe von den oben erörterten Zustandsautomaten und deren bekannten Methoden spezifiziert und formal verifiziert und an zustandsbasierte Systeme zur Prüfung weitergegeben. Prüfungen implementieren effizient Algorithmen, die das modellierte System automatisch durchlaufen, um das definierte System zu überprüfen. Die grafisch spezifizierten Zustandsautomaten werden in eine Textdarstellung umgewandelt, die die ausgewählte Prüfungssoftware verarbeiten kann. Die Definition der benötigten Systemeigenschaften bietet das Werkzeug einer grafischen Benutzerschnittstelle und generiert automatisch eine formale Darstellung der definierten Eigenschaften. Mit diesem Ergebnis ist

die F – Anwendung im Stack mit der User-Software zu realisieren und in ihren Abläufen zu definieren.

6.2 Bewertung der Arbeit

Die Frage die auftaucht [DIN72010] ist, wie es möglich ist bei einer solchen Sprache wie ’C’ unerkannte gefährliche Ausfälle zu entdecken, um von einem sicheren Bussystem zu sprechen. Bei den meisten Programmierern gilt das Argument: „Wir wissen, wo die kritischen Stellen sind und wir können mit diesen umgehen“. Das ist leider der falsche Weg, denn umso höher die Sicherheitsanforderungen, desto spezifischer ist der Umfang. Dies wird im Regelwerk [MISRA2013] für die Einschränkung, um unsichere Folgen zu verhindern und zu sperren, erörtert. Die Modelprüfung verwendet die formale Spezifikation der Sicherheitslogik und das erforderliche System hat die Eigenschaften, um mathematisch nachzuweisen, dass die Sicherheitslogik die definierten Eigenschaften besitzt. Wenn die Verifizierung scheitert, müssen zwei verschiedene mögliche Ursachen erneut analysiert werden: Die modellierten Zustandsautomaten können falsch sein oder die angegebenen Eigenschaften könnten möglicherweise zu streng sein. Ein Experte muss die fehlgeschlagene bewertete Eigenschaft besitzen zu entscheiden, ob die Spezifikation der Sicherheitslogik oder die definierten Eigenschaften verändert werden. Dies bedeutet, dass die Spezifikationsphase und die Verifizierungsphase nicht streng getrennt werden dürfen. Als Referenz ist in der Abbildung 4 “Systematische Eignung der Software und Entwicklungslebenszyklus (V-Modell)“ zu sehen, dass sie abwechselnd ausgeführt werden müssen bis alle Eigenschaften erfüllt sind. Ein Projekt bezieht sich auf sechs Schritte, um die Sicherheit zu gewährleisten. Am Anfang eines Projektes wird ein Arbeitskreis der Sicherheit gebildet. Die übergeordneten Anforderungen des Produktmanagements werden gesammelt und in eine Fehlermöglichkeitsanalyse und eine Fehlereinflussanalyse gebracht, wo die Sicherheitsfunktionen beschrieben und bewertet werden. Der nächste Punkt ist das Grundlagendasein für das Management, da der Produktmanager nur das Produkt verfolgt und der Entwickler nur auf seine Aufgabenstellung und Aufgabenerledigung fixiert ist. Das Anforderungskonzept für das Sicherheitskonzept wurde schon in früheren Kapiteln sowie in[Reich2016] partiell diskutiert. Dieser Sicherheitsplan legt fest, in welcher Sicherheit sich das Produkt befinden muss. Er ist ein zentrales Dokument und steht am Anfang jeder Entwicklung. Im Verifikations- und Validationsplan kann gelesen werden, wie, wann, von wem, was verifiziert und am Ende validiert wird. Entlang jeder Lebenszyklusphase werden so Verantwortungen, Qualitätsmanagement, Konfigurationsmanagement, Änderungsmanagement, Werkzeuge und Maßnahmen zur Fehlervermeidung dokumentiert. Sichere Anforderungen müssen systematisch (bei mittleren und großen Projekten mit einem datenbankbasierten Werkzeug) erfasst werden. Die Dokumente werden nach einer Neuansicht mit dem Ziel der Vollständigkeit und Verständlichkeit zur Konzeptprüfung eingereicht. Die eigentliche Konzeptprüfung wird als nächstes angesprochen und ist unter anderem ein Statusbericht in Form einer Mängelliste über den Verifikations- und Validati-

onsplan. Nachdem die Mängel beseitigt sind, wird mit dem V-Modell weitergefahren. Parallel läuft der Zugriffsprozess mit allen geplanten Überarbeitungen und Berichten an. Der darauf folgende Punkt ist das Design, welches bei der Firmware ein Werkzeug für hilfreiche Anbindung im Anforderungsstatus einer Datenbank sein kann. Hierdurch können weitere Fehler vermieden und eine Nachvollziehbarkeit sichergestellt werden. Bei Fertigstellung wird das Design durch eine erneute Fehlereinflussanalyse mit Methoden der Zuverlässigkeit betrachtet. Hierbei werden alle Operationen in Bezug auf ihren Einfluss auf sicherheitskritische Funktionalität in Kontroll- und Datenfluss klassifiziert. Zusätzlich wird jede Operation analysiert. Es kommen ebenso Prüfungen in der Hardware vor, um die Ausfallwahrscheinlichkeit zu bestimmen. Danach ist das Design abgeschlossen und die darauf folgenden Testfälle in White- und Blackbox werden anhand der Anforderungen sichergestellt. Der folgende Schritt ist die Softwareintegration, wo die Qualitätssicherung durch statische Codeanalysen mit Softwaremetriken und Code Coverage Tests unerlässlich ist. Zusätzlich sollte nun die defensive Programmierung angestrebt werden. Der Abschluss bildet ein Design-Review, der sowohl über eine qualifizierte Person gewährleistet sein muss als auch alle Anforderungen bestätigt und verifiziert, sodass der Zusammenhang zwischen den Methoden und Anforderungen zur Minderung des Bussystemrisikos nach den oben genannten Regelwerken gezeigt wird. Der größte Teil des Themas der sicheren Busdokumentation wird übergeordnet in der nachfolgenden Abbildung 3 “Feldbus und Sicherheitsnormen für Prozessautomatisierung“ dargestellt [Syst2007]. In Kapitel 7 “Testspezifikation“ [Siem2019] wird die spezifische Umsetzung der PSD Fehleraufdeckungsmechanismen über die FIT nachgewiesen. Es wird eine ein kanalige und zwei kanalige Generiervariante vorgestellt. In der einkanaligen Variante erkennt der PSD einen verfälschten Kontrollwert des Algorithmus. Bei der zweikanaligen Variante erkennt der PSD einen verfälschten Rückgabewert der Funktion. Der letzte Test ist der PROFIsafe-Conformance-Test und wird mit Hilfe eines qualifizierten Layer-Testprogramms und maschinell erzeugten Test-Telegrammen durchgeführt. Die fehlerfreie Kommunikation zwischen F-Host und F-Device wird durch diesen Test geprüft. Somit wird die Treiberfunktionalität und Integration des PSD in ihrer F – Applikation ebenso verifiziert.

6.3 Ausblick

Wie in dieser Diplomarbeit gezeigt wird, ist ein wichtiger Punkt die technische Dokumentation, die in den Themen Zielen, Anforderungen der Bereiche im Management sowie in ihrer Beurteilung verankert ist. Ziel ist es bei der technischen Dokumentation, die notwendigen Informationen festzulegen, die dokumentiert werden müssen, damit alle oben beschriebenen Phasen des Softwaresicherheitslebenszyklus wirkungsvoll ausgeführt werden können. Im Wesentlichen befasst sich die technische Dokumentation mit Informationen und nicht mit deren Gestalt und Aussehen. Die Anforderungen sind, dass die Dokumentation hinreichende Informationen zu jeder abgeschlossenen Phase des Softwaresicherheitslebenszyklus enthalten muss, welche für eine wirkungsvolle Erfüllung nachfolgender Phasen und Verifikationstätigkeiten notwendig sind. Da diese Phasen über die Arbeit hinweg ausgiebig diskutiert wurden, können nun die eventuellen auftretenden Prob-

leme erörtert werden. In Kapitel 4 wird eine einfache Kommunikation für die F – Anwendung vorgestellt. Es gibt jedoch nicht nur die Buskommunikation zwischen einer F - Anwendung sondern mehrere F–Anwendungen [RIO2017] untereinander. Jede zertifizierte F-Anwendung besitzt eine Schnittstelle zur Übergabe [FBus2001], die zwischen den Buskopplern siehe Abbildung “6-6 Nachrichtenfluss zwischen den Anwendungen“ zusätzlich zertifiziert werden müssen. Auch hier werden die obengenannten fehlerbeherrschten Maßnahmen für die einzelnen Feldbusteilnehmer in Anspruch genommen, wobei der gesamte Übertragungsweg von Datenschnittstelle zu Datenschnittstelle bewertet werden muss. Im Normalfall wird nur die Übertragungsstrecke über den CRC abgesichert. Somit sind Fehler im Buskoppler nicht zu erkennen, da es sich nicht um einen zertifizierten Standardfeldbus handelt und somit ein anderer Weg gefunden werden muss. Die fehlervermeidenden Maßnahmen sind nur in der Einzelprüfung vom Hersteller möglich. Wie in der Diplomarbeit [Reich2016] schon erörtert, fehlen die notwendigen Angaben der hardwaremäßigen Ausfallwahrscheinlichkeit “HTF“, um eine geeignete Qualifizierung vorzunehmen. Um den nicht zertifizierten Standardbus zu bewerten, bedient man sich der funktionalen Sicherheit, wobei der Buskoppler in eine Sicherheitskette geschliffen und über den $MTTF_d$ neu bewertet wird. Eine zweite Betrachtung ist eine reine Softwarebetrachtung. Die Funktion psd_RecvFOutTele() besitzt als Übergabeparameter zwei Zähler mit je 1 ms Zählwert für die jeweile CPU und deren Telegramm. Durch eine falsche Zykluszeit treten die fehlerbeherrschten Maßnahmen wie die Maskierung nicht ein und es kann zu Vermischungen zwischen nicht sicherheitsrelevanten und sicherheitsrelevanten Daten kommen. Als Abhilfe kann man einen Hardware - Watchdog einsetzen, da die Zykluszeit über eine falsche projektierte Hardware kommen kann. Zum Schluss ist noch zu sagen, dass es keine geeignete Lösung der Fehlererkennungsmaßnahmen gibt, da jede F–Applikation in ihrer Hardware verschieden ist.

Index

A

Akkumulator	Bestandteil einer CPU
ANSI-Standard C90, C99	Varianten der Programmiersprache C
Antivalenz	Exklusive ODER-Verknüpfung (EXOR)
Assembler	eine Programmiersprache, die auf den Befehlsvorrat eines bestimmten Computertyps ausgerichtet ist.
Assessment	Bewertung, Einschätzung
Applikation	werden Programme bezeichnet, die genutzt werden, um eine nützliche oder gewünschte nicht systemtechnische Funktionalität zu bearbeiten oder zu unterstützen
Ausfallrate λ	Die Ausfallrate λ ist die Addition $\lambda_s + \lambda_D$ mit $\lambda_s = \lambda_{SD} + \lambda_{SU}$ und $\lambda_D = \lambda_{DD} + \lambda_{DU}$, wobei λ_s die ungefährliche Ausfallrate und λ_D die gefährliche Ausfallrate bestimmt werden. λ_{SD} gilt der entdeckten und λ_{SU} der unentdeckten ungefährlichen Ausfälle. λ_D gilt der entdeckten und λ_{DU} der unentdeckten gefährlichen Ausfälle. siehe [Reich2016]
Ausfall-wahrscheinlichkeit F_t	Die Ausfallwahrscheinlichkeit $F_{(t)} = 1 - e^{-\lambda t}$ mit Annahme λ = konst. und $\lambda \cdot t$ << 1 ergibt die Ausfall-wahrscheinlichkeit $F_{(t)} = \lambda \cdot t$ siehe [Reich2016]
Äquivalenzklasse	Test von Partitionen des Eingangsbereiches

B

Bernoulli-Verteilung	Zufallsgrößen mit einer Bernoulli-Verteilung benutzt man zur Beschreibung von zufälligen Ereignissen, bei denen es nur zwei mögliche Versuchsausgänge gibt.
binär-symmetrisches Übertagungs-	Der binäre Übertragungskanal ist ein wertdiskreter

system	Übertragungskanal. Somit ist seine maximale Kanalkapazität auf ein Bit limitiert
Bitfehlerwahrscheinlichkeit	Die Bitfehlerhäufigkeit ist ein Maß der Nachrichtentechnik für die Qualität der Übertragung über einen Kanal oder einer digitalen Übertragungsstrecke.
Black-Box Verfahren	Dynamisches Verhalten unter realen Bedingungen und der Aufdeckung der Nichteinhaltung der Spezifikation
Branch Coverage	Diese Metrik wird als Zweigüberdeckung bezeichnet
Big Endian	Bytereihenfolge (Ablegung der höchstwertigen Speicheradresse im Hauptspeicher)
Bottom-Up-Methode	Analytische Methoden von unten nach oben (Fehlerbaumanalyse)

C

Code-Review	Mit dem Review werden Arbeitsergebnisse der Softwareentwicklung manuell geprüft
Compiler	Ein Compiler ist ein Computerprogramm, das Quellcodes einer bestimmten Programmiersprache in eine Form übersetzt
Condition Coverage	Diese Metrik wird als Bedingungsabdeckung bezeichnet
Conformance-Test	Bezeichnet ein Element der Konformitätsprüfungen
Controller	Elektronische Komponente der Computer, welche Steuerungsaufgaben übernehmen.
CPU	Der Begriff wird auch Zentraleinheit genannt, da es sich hier um einen zentralen Hauptspeicher handelt
CRC	Eine Berechnung der zyklischen Blockprüfung (siehe Generatorpolynom)

D

Defensive Programmierung	Erstellung von Programmen, die anomale Steuerflüsse, Datenflüsse oder Datenwerte während ihrer Ausführung erkennen und auf diese in vorbestimmter und annehmbarer Art und Weise
Diversität	Verwendung unterschiedlicher Methoden, um die gleichen Ergebnisse zu erreichen
Device	engl. für ein Gerät

E

Echtzeitzeitbetriebssystem	Betriebssystem für Daten, die direkt abgreifbar sind
Einbitfehler	Fehler auf erster Bitstelle ungleich Eins
Einerkomplement	Operation mit bitweiser Negation
Einflussanalyse	Eine Abhängigkeit zwischen Bauteile oder Systeme
erodiert	Qualitäts- und Zuverlässigkeitsprüfung

F

Fail-Safe*	Eine Sicherheitsfunktion, die mit F – gezeigt wird
Funktionale Sicherheit	Störfestigkeit von sicherheitsbezogenen Systemen und von Betriebsmitteln, die vorgesehen sind sicherheitsbezoge Funktionen auszuführen

G

Generatorpolynom	Die zyklische Redundanzprüfung ist ein Verfahren zur Bestimmung eines Prüfwerts für Daten, um Fehler bei der Übertragung oder Speicherung erkennen zu können
Grenzwertanalyse	Eine Grenzwertanalyse oder Kontrollflussanalyse kann die Testfälle auf eine akzeptable Anzahl reduzieren.

H

Hamming Distanz	Unterschiedlichkeit von Zeichenketten
Hardware-Interrupt	eine kurzfristige Unterbrechung der normalen Programmausführung durch die Hardware
heuristische Ebene	Statische Auswertung von Ausschlußverfahren
High-Priority-Task	Aufgabe mit höchster Anforderung
Host	definiert als eine Verarbeitungseinheit mit einem Betriebssystem innerhalb eines Netzwerks, das seine Clients bedient.

I

Integrationstest	in der Softwareentwicklung eine aufeinander abgestimmte Reihe von Einzeltests
IOPS	Ein Leistungswert von elektronischen Speichermedien
iParameter	Es sind individuelle oder technologiespezifische Geräteparameter.

J

K

Kapselung	Verhinderung des unbeabsichtigten Zugriffs auf Daten und Prozeduren und dadurch Unterstützung einer guten Programmstruktur.
Konformität	ist die Übereinstimmung einer Person mit den Normen
Kontextwechsel	Unterbrechung eines Prozesses innerhalb des Betriebssystems
Kontrollflussgraph	Gerichteter Graph, um den Kontrollfluss zu erkennen

L

Layer	engl. für Schichten
Linker	ein Computerprogramm, das einzelne Programmmodule zu einem ausführbaren Programm zusammenstellt.
Little Endian	Bytereihenfolge (Ablegung der niederwertigen Speicheradresse im Hauptspeicher)
Low-Priority-Task	Aufgabe mit niedrigster Anforderung

M

Mehrbitfehler	Fehler auf mehreren Bitstellen ungleich Eins
Mid-Priority-Task	Aufgabe mit mittlerer Anforderung
MISRA C	ein C-Programmierstandard aus der Automobilindustrie
MMU	benennt eine Hardwarekomponente eines Prozessors, die den Zugriff auf den Arbeitsspeicher verwaltet.
Monitoring-Number	Nummer der Vorgangsüberwachung
Modell der Gaußstörung	Es dient zur mathematischen Modellierung des Verhaltens von nicht-deterministischen Systemen auf der Basis von statistischen Beobachtungen
Modul	Adressierbare Ein-/Ausgabeeinheit innerhalb eines modularen IO-Device
Modultest	in der Softwareentwicklung angewendet, um die funktionalen Einzelteile von Computerprogrammen zu testen
MTTF, $MTTF_d$	Die mittlere Dauer bis zum (gefahrbringenden) Ausfall

N

Nominativ	Bezug Kasus des Subjekts im Satz (wer oder was)

O

OSI-Referenz	Es handelt sich um ein Referenzmodell, was die Schichtenarchitektur in den Netzwerkprotokollen darstellt.

P

Parity-Bit	belegte Bitfolge als gerade oder ungerade zu ergänzen
PFD	Wahrscheinlichkeit eines gefahrbringenden Ausfalls bei Anforderung
Polynom	Eine Termsumme kn^n (mathematisch)
Polynomdivision	Ein Polynom wird durch ein anderes Polynom geteilt
Präemptiv (preemptiv)	Es kann dem Prozess Ressourcen bereits vor Beenden wieder entziehen, um sie zwischenzeitlich anderen Prozessen zuzuteilen
private function	In der Programmierung gibt es private Funktionen nur für den internen Zugriff
Process Value	Dies sind Ein- und Ausgabedaten, die zur Steuerung eines automatisierten Prozesses im Regelbetrieb benötigt werden.
PROFIBUS	es ist ein Standard für die Feldbus-Kommunikation in der Automatisierungstechnik
PROFIsafe	Es bestimmt auf Sicherheit ausgerichtete Geräte über PROFIBUS so kommunizieren zu lassen, dass in Sicherheitsautomatisierungs-Aufgaben bis SIL 3 eingesetzt werden können.
PROFINET	es ist der offene Industrial-Ethernet-Standard der PROFIBUS-Nutzerorganisation e. V. für die Automatisierung
Prototyp	Es ist ein funktionsfähiges, aber vereinfachtes Versuchsmodell eines geplanten Produktes.
prove on use	Nachweis verwenden
PSD / PST	dient als Schnittstelle zwischen dem Stack und der F – Applikation (siehe Abbildung Architekturübersicht PROFIsafe)

public function	In der Programmierung gibt es öffentliche Funktionen für den externen Zugriff
Q	
Qualifizierer	Definition in gut, schlecht und in den Werten PV, SV
R	
RAM	Definiert als Arbeitsspeicher
Redundanz	Erkennung von Ausfällen durch Bereitstellung mehrerer funktionaler Einheiten durch Überwachung des Verhaltens jeder dieser Einheiten, um Ausfälle zu erkennen und durch Einleiten eines Übergangs in einen sicheren Zustand, wenn irgendeine Unstimmigkeit im Verhalten erkannt wird
Regressionstest	Wiederholung von Testfällen, um sicherzustellen, dass Modifikationen in bereits getesteten Teilen der Software keine neuen Fehler verursachen
rekursiv	prinzipiell unendlicher Vorgang, der sich selbst als Teil enthält oder mithilfe von sich selbst definierbar ist
Restfehlerwahrscheinlichkeit	die wiederholte Nachricht wird bei mehreren Durchläufen korrekt übertragen. Eine Rückleitung muss vorhanden sein
ROM	Ein Festwertspeicher oder Nur-Lesespeicher
Rückgabeparameter	Variablen, die auf bestimmte Werte eingestellt wurden und zurückgegeben werden
Rückwirkungsfreiheit	Ausgangsgröße hat keine Rückwirkung auf Eingangsgröße
Rückwärtsverfolgbarkeit	vom Validierungsplan der Sicherheit der Software zur Spezifikation der Anforderungen an die Sicherheit der Software
RT-Ethernet	PROFInet Protokolle und Dienste für die UDP Kommunikation (RTA, RTC)
RTOS	Echzeitbetriebssystem
S	
Saldierung	Addition über den/ die Akkumulator‘en
Sandwich- Methode	es ist ein Hybrid zwischen der Top-Down-Methode und der Bottom-Up Methode

Safety-Task	Sicherheitsbezogene Aufgabe
Scheduler	regelt die zeitliche Ausführung mehrerer Prozesse in Betriebssystemen
SFF	Anteil der sicheren Ausfälle
SIL	Sicherheitsanforderungsstufe
Slave	PROFIBUS-Synonym für IO-Device.
Source Code	engl. Wort für Quellcode
SRS	Eine sicherheitsbezogene Spezifikation
Stack	in der Informatik bezeichnet man ein Stapelspeicher als eine häufig eingesetzte dynamische Datenstruktur
Statement Coverage	Diese Metrik wird als Anweisungsüberdeckung bezeichnet
Submodule	es ist eine PROFINET-Definition für die kleinste adressierbare Einheit.
Substitute Value	Ein Ersatzwert ist der Wert, den ein Gerät oder ein Host ausgibt, wenn kein gültiger Prozesswert verfügbar ist.
T	
TCP/IP	Siehe TCP/IP Referenz
Timer Tick	Ausführung von Zeitprozeduren in festgelegten Abständen (Intervall)
Token Passing	Weitergabe an den nächsten Teilnehmer nach seiner Sendung
Top-Down-Methode	Analytische Methoden von oben nach unten (Fehlerbaumanalyse)
Transition	Alternativverzweigung
U	
UDP	Siehe UDP-Referenz
Übergabeparameter	Variablen, die auf bestimmte Werte eingestellt werden können
UML	ist eine grafische Modellierungssprache zur Spezifikation
unabhängige Personen	z.B. Behörden (Technischer Überwachungsverein
V	
V-Modell	Vorgehensmodell in der Softwareentwicklung
Validation	Anforderungen für eine spezielle beabsichtigte Ver-

	wendung, die erfüllt worden ist
Verifikation	Bereitstellung eines Nachweises, dass die Anforderungen erfüllt worden sind
Vorwärtsverfolgbarkeit	von der Spezifikation der Anforderungen an die Sicherheit der Software zum Validierungsplan der Sicherheit der Software

W

Walk-Through	einhalten von Spezifikation und Ausführung aufgedeckt
Watchdog	Der Begriff bezeichnet eine Funktion zur Ausfallerkennung eines digitalen Systems
White-Box Verfahren	Beim White-Box-Verfahren handelt es sich um die ermittelten Fehlersymptome beim dynamischen Verhalten in den inneren Funktionsbedingungen.

X

Y

Z

Zustandsmaschine/Zustandsautomat	Verhaltensmodell besteht aus Zuständen, Zustandsübergängen und Aktionen

Literatur

[DIN12010]	DIN EN 61508-1:<Funktionale Sicherheit sicherheitsbezogenere-lektrischer/elektronischer/programmierbarer elektronischer Syste-me –Teil 1: Allgemeine Anforderungen an sicherheitsbezogenee-lektrische/elektronische/programmierbare elektronische Syste-me>(IEC 61508-1:2010);Deutsche Fassung EN 61508-1:2010
[DIN22010]	DIN EN 61508-2:<Funktionale Sicherheit sicherheitsbezogenere-lektrischer/elektronischer/programmierbarer elektronischer Syste-me –Teil 2: Anforderungen an sicherheitsbezogeneelektri-sche/elektronische/programmierbare elektronische Systeme>(IEC 61508-2:2010);Deutsche Fassung EN 61508-2:2010
[DIN32010]	DIN EN 61508-3:<Funktionale Sicherheit sicherheitsbezogenere-lektrischer/elektronischer/programmierbarer elektronischer Syste-me –Teil 3: Anforderungen an Software> (IEC 61508-3:2010);Deutsche Fassung EN 61508-3:2010
[DIN72010]	DIN EN 61508-7:<Funktionale Sicherheit sicherheitsbezogenere-lektrischer/elektronischer/programmierbarer elektronischer Syste-me –Teil 7: Überblick über Verfahren und Maßnahmen> (IEC 61508-7:2010);Deutsche Fassung EN 61508-7:2010
[MISR2013]	MISRA C:2012: <Guidlines for the use of the C language in critical systems>UnterlagenAuftraggeberMurrelektronik GmbH Oppen-weiler
EZB52018]	Echtzeitbetriebssysteme – Einführung und Konzepte. URL: < https://www.embedded-software-engineering.de/echtzeitbetriebssysteme-einfuehrung-und-konzepte-a-651685/ >, verfügbar am 05.2018
[TUEV2018]	Schulung Funktionale Sicherheit: <Software nach IEC 61508 Mo-dul 3> Unterlagen Auftraggeber Murrelektronik GmbH Oppenwei-ler, verfügbar am 05.2010
[Syst2007]	PROFIsafe Systembeschreibung Technologie und Anwendungen Vergleich. URL: < https://www.profibus.com >, verfügbar am 09.2007
[Siem2010]	PROFIsafe Treiber:<V2.1 für F-Slaves> Vergleich. URL: <silo.tips_siemens-simatic-profisafe-treiber-v21-fr-f-slaves-inhaltsverzeichnis-vorwort-1-produktbersicht-2-funktionsweise-des-psd-3.pdf>, verfügbar am 05.2010
[Siem2019]	PROFIsafe Treiber:<V2.2.2 für F-Slaves> Vergleich. URL: < https://support.industry.siemens.com>>, verfüg-bar am 04.2019

[Kunb2019] Kunbus: <Feldbus Grundlagen eigene Bibliothek>Unterlagen Auftraggeber Kunbus GmbH Denkendorf
Vergleich. URL: <https://www.kunbus.de/feldbus-grundlagen.html>, verfügbar am 07.2020

[RIO2017] Kunbus GmbH: <RIO-FA_3242_V109_Mar17>
Vergleich. URL: <https://www.profibus.com>, verfügbar am 03.2017

[FBus2001] Forschungsvorhaben 35/00: <Anwendung der Bussysteme in der Anlagensicherheit der Chemie-Industrie>
Vergleich. URL: < https://docplayer.org/19698582-Landesumweltamt-nordrhein-westfalen-forschungsvorhaben-35-00-anwendung-der-bussysteme-in-der-anlagensicherheit-der-chemie-industrie.html
Untersuchungsvorhaben.pdf>, verfügbar am xx.2001

[Reich2016] Ernst Henry Reichert: < Die essentielle Sicherheitsbetrachtung in Kraftwerken und ihre Ausfallwahrscheinlichkeiten >verfügbar am 07.2016

[TUB2003] Technische Universität Berlin: <Automatisiertes spezifikationsbasiertes Testen von Software>Vergleich. URL:
<https://docplayer.org/13290597-Technische-universitaet-berlin-ss-2003-sadik-cs-tu-berlin-de.html
>, verfügbar am xx.2003

Anlagen

Anlagen, Teil 1

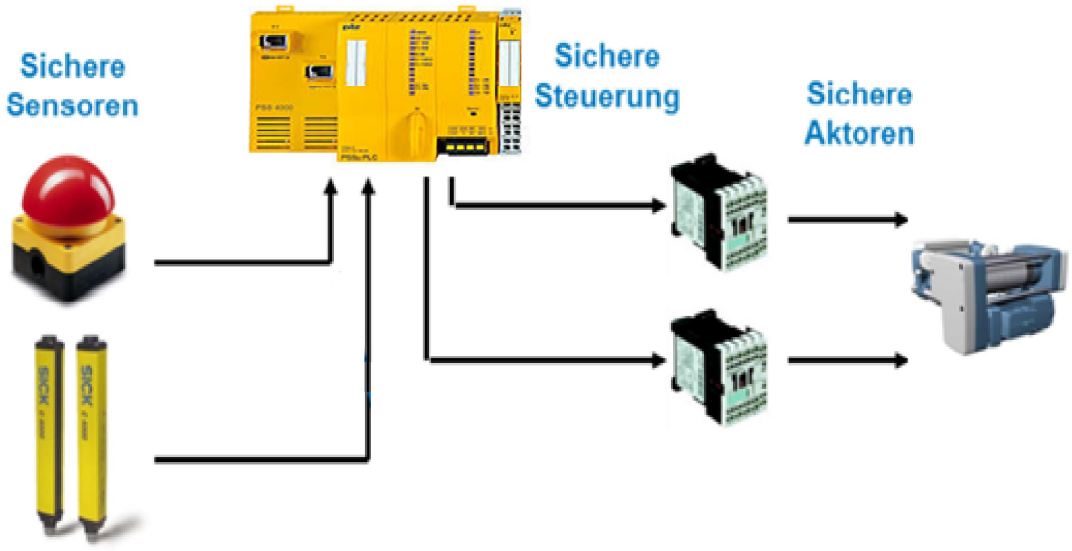

Abbildung 28 Sicherheitstechnik ohne Bussystem

Als Erinnerung an meiner vorigen Arbeit[49]…Die mittlere Ausfallwahrscheinlichkeit der Sicherheitsfunktionen basiert auf der Summe der einzelnen mittleren Ausfallwahrscheinlichkeiten der Teilsysteme. Eine Teilsystemstruktur besteht aus den Teilsystemen Erfassung (PFD_S), Auswertung (PFD_L) und Ausführung (PFD_{FE})…

Das Teilsystem (PFD_S) wird durch die sicheren Sensoren, das Teilsystem (PFD_L) wird durch die sichere Steuerung und das Teilsystem (PFD_{FE}) wird durch die sichere Aktoren dargestellt. Wie die sicherheitsgerichtete Software über den internen Bus (Rückwandbus), dem Systembus als ProfiBus mit dem Sicherheitstelegram ProfiSafe und dessen Hilfe des Internetprotokolls das PROFINET unterstützt, wird im Kapitel sicherheitsgerichtete Busübersicht erörtert.

[49][Reich2016]

Anlagen, Teil 2

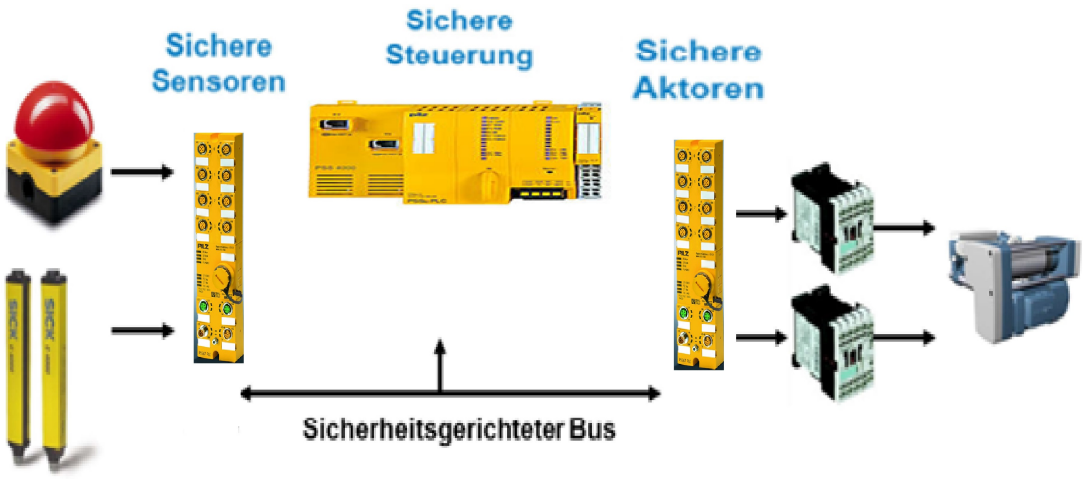

Abbildung 29 Sicherheitstechnik mit Bussystem

Wie im vorherigen Kapitel angesprochen, erkennt man hier, dass die Fragen der einzelnen Phasen bei dieser Vielfalt von Regelwerken nur ein Prozentanteil betrifft. Die hier abgebildeten Komponenten werden nach verschiedenen normativen Richtlinien gestaltet. Die einzelnen Richtlinien zählen unter den harmonischen Regelwerken. Für den PROFIBUS werden

- DIN EN IEC 61158-5-3
  Industrielle Kommunikationsnetze
  Teil 5: Dienstfestlegung der Anwendungsschicht
- DIN EN IEC 61158-6-3
  Teil 6: Protokollspezifikation der Anwenderschicht
- DIN EN IEC 61158-1:2019-01
  Industrielle Kommunikationsnetze – Profile
  Teil 1: Feldbusprofile

diese Regelwerke zu Rate gezogen.

Für das ProfiNet werden

- DIN EN IEC 61158-5-10
  Industrielle Kommunikationsnetze
  Teil 5: Dienstfestlegung der Anwendungsschicht
- DIN EN IEC 61158-6-10
  Teil 6: Protokollspezifikation der Anwenderschicht
- DIN EN IEC 61784-2:2020-04

Industrielle Kommunikationsnetze – Profile
Teil 2: Zusätzliche Feldbusprofile für Echtzeitnetzwerke

diese Regelwerke zu Rate gezogen.

Für das ProfiSafe wird

- DIN EN IEC 61784-3-3
  Industrielle Kommunikationsnetze – Profile
  Teil 3: Funktional sichere Übertragung bei Feldbussen – Allgemeine Regeln und Festlegungen für Profile

dieses Regelwerk zu Rate gezogen.

Anlagen, Teil 3

Siehe Programmbeispiel fProfisafe.c

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Düsseldorf, den 21. April 2021

Ernst Henry Reichert