DGUV
Deutsche Gesetzliche
Unfallversicherung
Spitzenverband

IFA Report 1/2020

Praxisgerechte Umsetzung der Anforderungen für sicherheitsbezogene Embedded-Software nach DIN EN ISO 13849-1

Verfasser: Thomas Bömer, Karl-Heinz Büllesbach, Michael Hauke, Stefan Otto, Christian Werner
Institut für Arbeitsschutz der Deutschen Gesetzlichen Unfallversicherung (IFA),
Sankt Augustin

Herausgeber: Deutsche Gesetzliche Unfallversicherung e. V. (DGUV)
Glinkastr. 40
10117 Berlin
Telefon: 030 13001-0
Telefax: 030 13001-9876
Internet: www.dguv.de
E-Mail: info@dguv.de

– April 2020 –

Publikationsdatenbank: www.dguv.de/publikationen

ISBN (online): 978-3-948657-00-0
ISSN: 2190-7994

Kurzfassung

Praxisgerechte Umsetzung der Anforderungen für sicherheitsbezogene Embedded-Software nach DIN EN ISO 13849-1

Dieser Report richtet sich an Software-Entwickler, die sicherheitsbezogene Embedded-Software (SRESW) für Maschinen im Rahmen der DIN EN ISO 13849-1 erstellen und überprüfen. Die Erläuterungen dienen der Interpretation der normativen Anforderungen und sollen eine Empfehlung und Richtschnur für den Weg durch die verschiedenen Phasen des Software-Sicherheitslebenszyklus sein.

Abstract

Practicable implementation of the requirements concerning safety-related embedded software to EN ISO 13849-1

This report is intended for software developers producing and checking safety-related embedded software (SRESW) for machinery in the context of EN ISO 13849-1. The explanations support interpretation of the normative requirements and are intended as recommendations and guidance through the various phases of the software life cycle.

Résumé

Mise en pratique des exigences relatives aux logiciels intégrés relatifs à la sécurité selon EN ISO 13849-1

Ce rapport s'adresse aux concepteurs de logiciels qui élaborent et contrôlent les logiciels intégrés relatifs à la sécurité (SRESW) pour des machines, conformément à la norme EN ISO 13849-1. Les commentaires servent à l'interprétation des exigences normatives et visent à servir de recommandation et de ligne directrice pour le déroulement des différentes phases du cycle de vie des logiciels.

Resumen

Implementación práctica de los requisitos de Embedded Software para funciones de seguridad según EN ISO 13849-1

Este informe va dirigido a desarrolladores de software que crean y verifican Embedded Software para funciones de seguridad (SRESW) para maquinaria en el marco de la normativa EN ISO 13849-1. Las explicaciones sirven para facilitar la implementación de los requisitos normativos y pretenden ser una recomendación y una referencia sobre el camino a seguir a lo largo de las diversas fases del ciclo de vida útil del software.

Inhaltsverzeichnis

1 Einleitung

Dieser Report richtet sich an Software-Entwickler, die sicherheitsbezogene Embedded-Software (SRESW) für Maschinen im Rahmen der DIN EN ISO 13849-1 [1] erstellen und überprüfen. Die hierfür geltenden normativen Anforderungen sind in den Abschnitten 4.6.1 und 4.6.2 der Norm genannt. Die Anforderungen sind allerdings sehr allgemein und knapp formuliert. Während zu sicherheitsbezogener Anwendungsprogrammierung (SRASW) bereits ausführliche Hilfen des IFA existieren [2 bis 4], bleibt für SRESW demgegenüber in der praktischen Umsetzung oft unklar, was die normativen Anforderungen konkret implizieren.

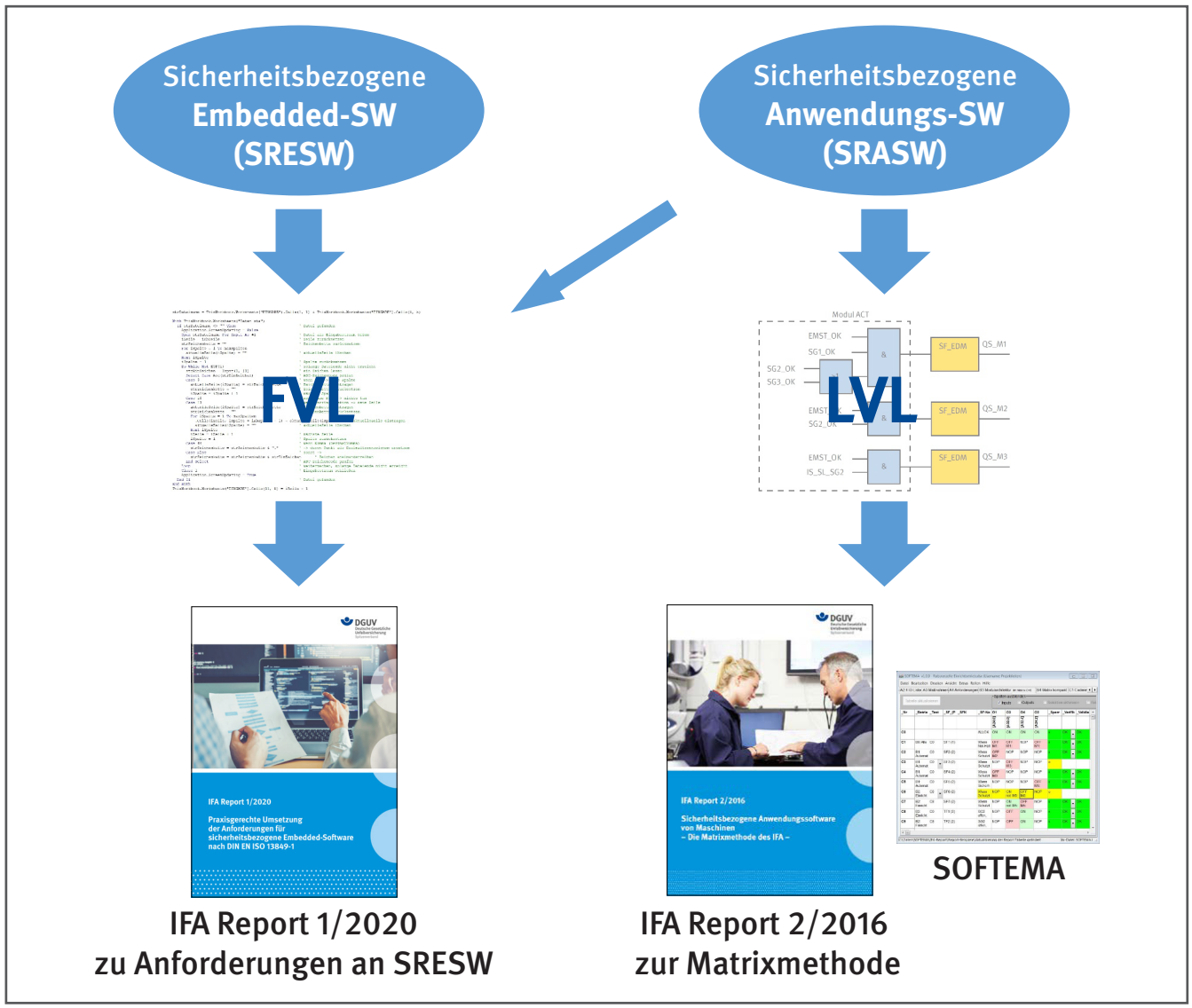
Die folgenden Erläuterungen dienen der Interpretation der normativen Anforderungen und sollen eine Empfehlung und Richtschnur für den Weg durch die verschiedenen Phasen des Software-Sicherheitslebenszyklus sein.

Einige der im Weiteren dargestellten Inhalte wurden bereits in der aktuellen Revision der Norm für ihre kommende vierte Auflage aufgegriffen, allerdings nicht in der hier beschriebenen Detailtiefe.

Da auch für Anwendungssoftware, die unter Verwendung einer Programmiersprache mit nicht eingeschränktem Sprachumfang (FVL) programmiert wurde, die in der Norm für SRESW genannten Anforderungen gelten, ist der vorliegende Report auch für deren Programmierung anwendbar.

Abbildung 1 gibt eine Übersicht über die bereits erschienenen Hilfen des IFA für die Programmierung sicherheitsbezogener Software.

Abbildung 1:
Hilfen des IFA für die Programmierung sicherheitsbezogener Software (SW: Software)



2 Softwareentwicklung nach DIN EN ISO 13849-1

2.1 V-Modell des Software-Sicherheitslebenszyklus

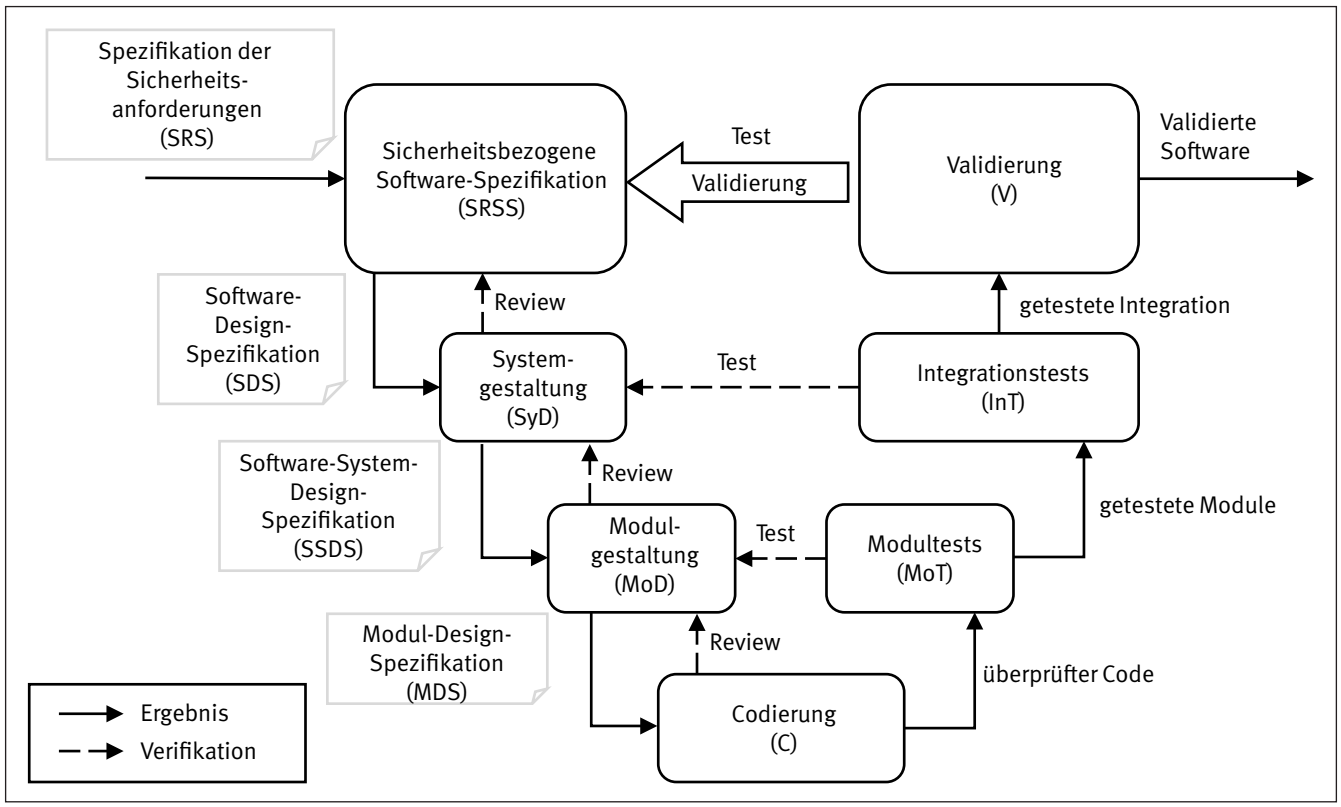
Zur Veranschaulichung des Entwicklungs- und Verifikationsprozesses sicherheitsrelevanter Software verwendet die DIN EN ISO 13849-1 in Abschnitt 4.6.1[1] das V-Modell (Abbildung 2). Das V-Modell basiert auf der Grundidee, dass es sich bei der Entwicklung und der Testung von Software um aufeinander bezogene, gleichwertige Tätigkeiten handelt. Bildlich wird dies durch den absteigenden und den aufsteigenden Ast des Buchstabens „V“ dargestellt.

Der linke, absteigende Ast steht für die einzelnen Entwicklungsschritte von der Spezifikation der Sicherheitsanforderungen (SRS) [5] über die Entwurfsphasen auf System- und Modulebene bis hin zur eigentlichen Codierung. Der rechte, aufsteigende Ast steht für die korrespondierenden Aufgaben in den Integrations- und Testphasen, in deren Verlauf die Module unter kontinuierlichen Tests zu größeren Teilsystemen zusammengesetzt und schließlich auf die Zielhardware transferiert werden. Dort wird das Gesamtsystem schließlich im Hinblick auf die sicherheitsfunktionalen Anforderungen validiert. Die Dokumentation spielt bei der Entwicklung nach dem V-Modell eine wichtige Rolle. Jede Ebene der Software-Entwicklung (absteigender Ast) hat als Output eine Spezifikation der Anforderungen für die nachfolgende Ebene. Die Spezifikation ist gleichzeitig die Vorgabe, gegen die die Software nach der Codierung auf den jeweiligen Integrationsebenen getestet wird.

Ziel der Entwicklung nach V-Modell ist es, lesbare, verständliche, testbare und wartbare Software zu erhalten. Kontinuierliche Reviews und Tests auf allen Ebenen des Software-Sicherheitslebenszyklus ermöglichen zusätzlich eine möglichst vollständige und fehlerfreie Validierung der Softwareentwicklung.

Für die Embedded-Software zugekaufter Standardkomponenten kann das V-Modell nicht angewendet werden, da die Software nicht speziell für den Einsatz in Sicherheitsfunktionen entwickelt wurde und auch für den Entwickler sicherheitsgerichteter Software nicht zugänglich ist. Hierfür werden am Ende des Abschnitts 4.6.2 in [1] alternative Anforderungen beschrieben, die aber nicht Gegenstand dieses Reports sind.

Abbildung 2:
Software-Sicherheitslebenszyklus nach dem V-Modell (Abkürzungen siehe Tabelle 1)



[1] Alle Referenzen beziehen sich auf die zum Zeitpunkt der Veröffentlichung dieses Dokuments gültige deutsche Ausgabe der Norm vom Juni 2016.

Tabelle 1:
Erläuterungen der Abkürzungen in Abbildung 2

Abkürzung	Bedeutung	Ergebnis
SRSS	Sicherheitsbezogene Software-Spezifikation: Spezifikation der Software unter Berücksichtigung der zu betrachtenden Sicherheitsfunktionen nach den Vorgaben der Spezifikation der Sicherheitsanforderungen (SRS)	Software-Design-Spezifikation (SDS)
SyD	Systemgestaltung: Entwurf der Software unter Berücksichtigung der sicherheitsgerichteten Hardwarestruktur nach den Vorgaben der Software-Design-Spezifikation (SDS)	Software-System-Design-Spezifikation (SSDS)
MoD	Modulgestaltung: Gestaltung der einzelnen Softwaremodule nach den Vorgaben der Software-System-Design-Spezifikation (SSDS)	Modul-Design-Spezifikation (MDS)
C	Codierung: Codierung der einzelnen Softwaremodule nach den Vorgaben und anschließendes Review auf Übereinstimmung mit der Modul-Design-Spezifikation (MDS)	überprüfter Programmcode der Module (mit Kommentaren)
MoT	Modultests: Test der Module auf Übereinstimmung mit der Modul-Design-Spezifikation (MDS)	getestete Module
InT	Integrationstests: Test der Software-Schnittstellen und – wenn möglich – der auf die Zielhardware transferierten Gesamt-Software auf Übereinstimmung mit der Software-System-Design-Spezifikation (SSDS)	getestete Integration
V	Validierung: Prüfen der Software und – wenn möglich – der Hardware gegen die Software-Design-Spezifikation (SDS) und die Spezifikation der Sicherheitsanforderungen (SRS)	validierte Software

Anmerkung: Abbildung 2 basiert auf Bild 6 der DIN EN ISO 13849-1. Die dort dargestellten Entwicklungs- und Testphasen wurden um die zugehörigen Spezifikationsdokumente sowie eine detailliertere Zuordnung von Review- und Testschritten erweitert. Die Kürzel hinter den Entwicklungs- und Testschritten dienen im folgenden Kapitel zur Zuordnung der beschriebenen Maßnahmen zu den Phasen des V-Modells.

2.2 Basismaßnahmen und zusätzliche Maßnahmen

Zusätzlich zum V-Modell listet [1] in Abschnitt 4.6.2 Maßnahmen auf, die bei der Entwicklung sicherheitsrelevanter Software in Abhängigkeit des erforderlichen Performance Levels (PL_r) anzuwenden sind[2]:

Basismaßnahmen (für Bauteile mit einem PL_r a bis d):

- Software-Sicherheitslebenszyklus mit Verifikation und Validierung (1.1),
- Dokumentation der Spezifikation und des Entwurfs (1.2),
- modulare und strukturiere Entwicklung und Codierung (1.3),
- Beherrschung systematischer Ausfälle (1.4),
- bei Verwendung softwarebasierter Maßnahmen zur Beherrschung zufälliger Hardwareausfälle: Verifikation der korrekten Implementierung (1.5),
- funktionale Tests, zum Beispiel Black-Box-Tests (1.6),
- geeignete Aktivitäten für den Software-Sicherheitslebenszyklus nach Änderungen (1.7).

[2] Sprachliche Unklarheiten in der deutschen Fassung der Norm wurden auf der Basis der englischen Originalfassung korrigiert.

Zusätzliche Maßnahmen (für Bauteile mit einem PL_r c bis d):

- Projekt- und Qualitätsmanagement vergleichbar mit zum Beispiel der Reihe DIN EN 61508 oder DIN EN ISO 9001 (2.1),
- Dokumentation aller relevanten Aktivitäten während des Software-Sicherheitslebenszyklus (2.2),
- Konfigurationsmanagement, um alle Konfigurationsmerkmale und Dokumente mit Bezug zu einer SRESW-Freigabe festzustellen (2.3),
- strukturierte Spezifikation mit Sicherheitsanforderungen und Entwicklung (2.4),
- Verwendung geeigneter Programmiersprachen und rechnergestützter Werkzeuge mit Betriebsbewährung (2.5),
- modulare und strukturierte Programmierung, Abgrenzung von nicht sicherheitsbezogener Software, beschränkte Modulgröße mit vollständig definierten Schnittstellen, Verwendung von Entwurfs- und Codierungsrichtlinien (2.6),
- Verifikation des Codes durch ein Walk-Through-Review (Überprüfung) mit einer Kontrollflussanalyse (2.7),
- erweiterte Funktionstests (zum Beispiel Grey-Box-Tests), Leistungstests oder Simulation (2.8)
- Einflussanalyse und angemessene Software-Sicherheitslebenszyklus-Aktivitäten nach Änderungen (2.9).

Anmerkung: Die Nummer in den Klammern hinter den jeweiligen Maßnahmen gibt ihre Fundstelle in Abschnitt 4.6.2 in [1] an. Die erste Zahl kennzeichnet, ob es sich um eine Basismaßnahme (1) oder eine zusätzliche Maßnahme (2) nach Norm handelt, die zweite Zahl steht für den jeweils referenzierten Spiegelstrich in dem betreffenden Absatz. Beispielsweise steht die Referenz 1.2 für die Basismaßnahme „Dokumentation der Spezifikation und des Entwurfs“.

Da die Auflistung der Maßnahmen in der Norm ohne Kommentierung erfolgt, ist für die an Entwicklung und Test Beteiligten nicht immer ohne Weiteres zu erkennen, welche Tätigkeiten an welcher Stelle im Lebenszyklus der Software aus den Maßnahmen abzuleiten sind.

Der vorliegende Report ordnet im folgenden Kapitel 3 die Maßnahmen aus Abschnitt 4.6.2 nach [1] den einzelnen Entwicklungs- und Testphasen des V-Modells zu. Zusätzlich wird erläutert, welche Tätigkeiten aus jeder Maßnahme abzuleiten sind. Übergeordnete Maßnahmen, die nicht dem V-Modell zugeordnet werden können, werden in Kapitel 4 gesondert aufgeführt.

Da die einzelnen Maßnahmen mitunter mehreren Phasen des V-Modells zuzuordnen sind, wird der betreffende Normentext in der nachfolgenden Tabelle gegebenenfalls auf den jeweils relevanten Teil gekürzt, siehe etwa in Abschnitt 3.1 die oben genannte Basismaßnahme „Dokumentation der Spezifikation“.

Anmerkung: Für Software mit PL_r = e verweist [1] auf die Anforderungen für SIL 3 in Abschnitt 7 der DIN EN 61508-3 [6]. Eine Ausnahme bilden Architekturen der Kategorien 3 und 4, in denen für beide Kanäle Diversität in Spezifikation, Entwurf und Codierung verwendet wird. In diesem Fall ist die Anwendung der in diesem Dokument beschriebenen Basis- und zusätzlichen Maßnahmen ausreichend.

3 Maßnahmen im Software-Sicherheitslebenszyklus nach dem V-Modell

3.1 Sicherheitsbezogene Software-Spezifikation (SRSS)

Referenz	Maßnahme	Erläuterung
SRSS_1	Dokumentation der Spezifikation (1.2)	Auf der Basis der Spezifikation der Sicherheitsanforderungen (SRS), in der u. a. die Sicherheitsfunktionen detailliert beschrieben sind, ist die Software-Design-Spezifikation zu erstellen. Unabhängig von der konkreten Realisierung wird darin allgemein beschrieben, welche Funktionen die Software zur Umsetzung der Sicherheitsfunktionen auszuführen hat. Es wird eine strukturierte Form der Software-Design-Spezifikation erwartet. Diese bildet die Grundlage für die folgende Phase der Systemgestaltung. Dazu enthält sie u. a. die Festlegung der zu verwendenden rechnergestützten Werkzeuge mit Betriebsbewährung (siehe C_5, MoT_2, InT_3).
SRSS_2	Strukturierte Spezifikation mit Sicherheitsanforderungen und Entwicklung (2.4)	Es wird sowohl eine strukturierte Spezifikation als auch eine strukturierte Entwicklung gefordert. Durch die Maßnahme SRSS_2 ergeben sich keine zusätzlichen, über SRSS_1 hinausgehenden Aktivitäten. Es wird erwartet, dass die Sicherheitsanforderungen an die Software bereits mit der Software-Design-Spezifikation (SDS) bereitgestellt werden. Die Planung der Entwicklung sollte im Projektmanagement (siehe QM_2) abgebildet werden. Die darauffolgende Durchführung der strukturierten Entwicklung ergibt sich bei Anwendung des V-Modells (siehe QM_1) automatisch.

3.2 Systemgestaltung (SyD)

Referenz	Maßnahme	Erläuterung
SyD_1	Dokumentation des Entwurfs (1.2)	Es ist die Software-System-Design-Spezifikation zu erstellen. Diese beschreibt die Umsetzung der festgelegten Sicherheitsfunktionen auf Systemebene in der Software. Es ist zu beschreiben, welche Module vorgesehen sind und wie diese in der Software-Architektur zusammenwirken. Die Software-System-Design-Spezifikation bildet somit die Grundlage für die Modulgestaltung. Der Test der Systemgestaltung einschließlich der softwarebasierten Maßnahmen zur Beherrschung zufälliger Hardwareausfälle istg zu planen und zu dokumentieren (siehe InT_1).
SyD_2	Dokumentation aller relevanten Aktivitäten während des Software-Sicherheitslebenszyklus (2.2)	Ergänzend zu SyD_1 ist das Review der Systemgestaltung gegen die Software-Design-Spezifikation zu dokumentieren. Die Dokumentation aller relevanten Aktivitäten während dieser Entwicklungsphase hilft, Fehler bei der Erstellung der sicherheitsrelevanten Software zu vermeiden und unterstützt bei deren Beurteilung. Die Dokumentation sollte in verständlicher und natürlicher Sprache unter Zuhilfenahme grafischer Darstellungen und eines Fachwortverzeichnisses erfolgen.
SyD_3	Modulare und strukturierte Entwicklung (1.3)	Das Ziel ist die Erstellung einer hierarchischen Beschreibung von Teilanforderungen („Modularisierung“) der zu realisierenden sicherheitsrelevanten Funktionalität, d. h. einer Beschreibung vom Groben ins Feine. Dies ermöglicht es, diese Teilanforderungen einzeln zu codieren und zu verifizieren. Im Rahmen der Systemgestaltung wird auch über die Verwendung von Interrupts entschieden. Es sind folgende Aspekte zu berücksichtigen: • Beschreibung und Begrenzung der Funktionalität: Idealerweise setzt ein Software-Modul eine einzelne, definierte und zu erfüllende Funktion um. • Schnittstellen (Eingabe und Ausgabe): Die Verbindungen zwischen den Softwaremodulen müssen begrenzt sowie eindeutig definiert und beschrieben werden. • Die Aufrufbeziehungen der Module untereinander müssen eindeutig sein. Sogenannte polymorphe Aufrufstrukturen, bei denen erst zur Laufzeit entschieden wird, welches Modul bzw. welche Methode aufgerufen wird, sind zu vermeiden. Ausnahmen sind nachvollziehbar zu begründen und zu dokumentieren.

3.3 Modulgestaltung (MoD)

Referenz	Maßnahme	Erläuterung
MoD_1	Dokumentation des Entwurfs (1.2)	Es ist die Modul-Design-Spezifikation zu erstellen. Diese beschreibt die Umsetzung der sicherheitsrelevanten Funktionen in der Codierung der einzelnen Module. Die Beschreibung sollte durch Ablaufdiagramme, Flussdiagramme, Datenflusspläne, Zeitablaufdiagramme etc. unterstützt werden. Einzelne Lösungsansätze werden dabei im Detail beschrieben. Es wird (wenn notwendig), auf die hardwarespezifische Softwareumgebung eingegangen, zum Beispiel RAM/ROM- Bereiche, Adressbereiche der I/O, Interruptverwendung und deren Vektoren, Reaktionszeiten der Hardware (mit Bezug etwa zu Eingangsfiltern), Kommunikation und Protokoll mit einem ggf. vorhandenen Parallelkanal. Die Modul-Design-Spezifikation bildet damit die Grundlage für die Codierung.
MoD_2	Dokumentation aller relevanten Aktivitäten während des Softwaresicherheitslebenszyklus (2.2)	Ergänzend zu MoD_1 ist das Review der Modulgestaltung gegen die Software-System-Design-Spezifikation zu dokumentieren. Siehe auch SyD_2.
MoD_3	Modulare und strukturierte Entwicklung (1.3)	Die in der Software-System-Design-Spezifikation beschriebene Realisierung der Sicherheitsfunktionen wird durch Aufteilung in einzelne, in sich abgeschlossene Unterprogramme (Softwaremodule) konkretisiert. Dieser Schritt ermöglicht eine modulare und strukturierte Codierung (siehe C_3). Eine strukturierte Codierung bezieht sich auf den inneren Aufbau eines Programms oder Programmteils, eine modulare Codierung betrachtet dessen äußere Eigenschaften (zum Beispiel Schnittstellen). Neben der Fehlerverringerung ist das Ziel, die Komplexität eines Programms zu verringern sowie schlechte oder unübersichtliche Programmstrukturen zu vermeiden. Programme sollten leicht gelesen, gewartet und geändert werden können. Hieraus resultieren folgende Anforderungen: • Das Programm sollte in angemessen kleine Softwaremodule (Modularisierung) unterteilt sein. • Der Ablauf eines Programms sollte nur folgende Konstrukte enthalten: — Abfolge (Sequenz), — Wiederholung (Iteration), — Auswahl (Selektion). • Entsprechend sollte die grafische Darstellung des Kontrollflusses des Programms nur folgende Elemente enthalten (siehe auch MoT_3): — Verzweigungsknoten, — Anweisungen, — Vereinigungsknoten. • Es sollte nur möglichst wenig Pfade durch ein Softwaremodul und einfache Beziehungen zwischen den Eingabe- und Ausgabeparametern geben. • Komplizierte Berechnungen als Grundlage von Verzweigungen und Schleifenbedingungen sollten vermieden werden. • Komplizierte Verzweigungen und insbesondere bedingungslose Sprünge (Goto) sollten vermieden werden. Diese führen zu unstrukturiertem Code. Unzugänglicher Code (englisch: dead code) ist nicht zulässig. • Interrupts sollten aus Gründen der Minimierung von Echtzeiteinflüssen und zur besseren Übersichtlichkeit des Programmablaufs nicht verwendet werden. Abweichungen von diesem Entwurfsprinzip sind nur zulässig, wenn sich dadurch eine wesentliche Vereinfachung des Programms ergibt. Die Gründe sind im Einzelnen zu dokumentieren. • Eine modulare Codierung zeichnet sich durch eine Begrenzung der Softwaremodulgröße sowie vollständig definierte Schnittstellen aus. Entsprechende Anforderungen finden sich in MoD_5. Zur Gewährleistung dieser Eigenschaften ist in der Entwurfsphase eine Reihe von Regeln festzuhalten, die als verbindliche Codierregeln in die folgende Phase der Codierung der sicherheitsbezogenen Software eingehen. Anforderungen hierzu finden sich in MoD_6.

Referenz	Maßnahme	Erläuterung
MoD_4	Abgrenzung von nicht sicherheitsbezogener Software (2.6)	Bei der späteren Codierung ist darauf zu achten, dass nicht sicherheitsbezogene Software die Ausführung von Sicherheitsfunktionen nicht beeinträchtigen kann. Hierzu sind bereits in der Phase der Modulgestaltung Regeln, Entwurfs- und Codierungsrichtlinien aufzustellen, die eine Kapselung der sicherheitsbezogenen Software erleichtern. Die sicherheitsbezogene Software stellt die softwaretechnische Realisierung einer Sicherheitsfunktion dar und beginnt an dem Punkt in der Gesamtsoftware, an dem ein spezifiziertes, sicherheitsgerichtetes Datum gelesen wird. Sie enthält darüber hinaus den Teil der Software, der das Datum verarbeitet, und endet an der Stelle, an der ein sicherheitsgerichtetes Datum ausgegeben wird. Die sicherheitsbezogene Software schließt die Datenverwaltung (zum Beispiel Deklaration, Initialisierung) der relevanten Daten mit ein. Weiterhin zählen zu sicherheitsgerichteter Software auch die softwaretechnischen Realisierungen von Testroutinen, die für eine ausreichende Fehlererkennung (Diagnosedeckungsgrad) der Funktionskanäle benötigt werden.
MoD_5	Beschränkte Modulgröße mit vollständig definierten Schnittstellen (2.6)	Softwaremodule sollten über eine beschränkte Modulgröße und vollständig definierte Schnittstellen verfügen. Im Einzelnen ergeben sich daraus folgende konkrete Anforderungen: • Ein Softwaremodul sollte im Idealfall nur eine einzige Aufgabe oder zu erfüllende Funktion haben. • Schnittstellen zwischen Softwaremodulen sollten begrenzt und vollständig definiert werden; jede Schnittstelle eines Softwaremoduls sollte nach Möglichkeit nur die Parameter enthalten, die für dessen Funktion notwendig sind. • Zu einer vollständig definierten Schnittstelle (für Aufrufparameter und Rückgabewerte) gehören u. a. — Anzahl der übergebenen Parameter, — Datentypen, — vorgesehene Wertebereiche der Parameter. • Softwaremodule sollten mit anderen Softwaremodulen allein durch ihre Schnittstellen kommunizieren. Variablen in einem Modul sollten privat sein. Werden globale oder gemeinsame Variablen verwendet, sollten diese gut strukturiert sein, der Zugriff auf sie kontrolliert erfolgen und ihre Verwendung in jedem Fall gerechtfertigt sein. Es empfiehlt sich, bei globalen Variablen aussagekräftige Bezeichnungenn zu verwenden. • Alle Schnittstellen der Softwaremodule sollten (auch im Quelltext) vollständig dokumentiert werden. • Die Größe eines Softwaremoduls sollte einen bestimmten Wert nicht überschreiten (empfohlene maximale Größe: 50 Zeilen); eine Ausnahme hiervon bildet sequentieller Code gleicher Funktionalität (zum Beispiel Selbsttest der einzelnen Befehle eines Mikroprozessors). • Die Schachtelungstiefe von Modulaufrufen ist möglichst gering zu halten (empfohlen wird eine maximale Schachtelungstiefe von fünf Aufrufen). • Softwaremodule sollten nur einen Eingang und einen Ausgang haben (für die Fehlerbehandlung kann ein weiterer Ausgang vorgesehen werden).
MoD_6	Verwendung von Entwurfs- und Codierungsrichtlinien (2.6)	Für die Codierung ist die Erstellung und Verwendung einer Codierungsrichtlinie unbedingt notwendig. Die Verwendung von Codierungsrichtlinien reduziert einerseits die Fehlerwahrscheinlichkeit und erleichtert andererseits die spätere Verifikation. In die Codierungsrichtlinien sollten alle in den vorhergehenden Abschnitten genannten, die Codierung betreffenden Anforderungen aufgenommen werden. Die Codierungsrichtlinie sollte mindestens folgende Bereiche abdecken und in Einzelanforderungen konkretisieren: • Modularer Ansatz: Begrenzung der Softwaremodulgröße und vollständig definierte Schnittstellen (siehe MoD_5), • Verständlichkeit des Codes: zum Beispiel eindeutige und schlüssige Benennung von Variablen, Richtlinien zur Dokumentation des Codes (siehe C_1), • Verifizierbarkeit und Prüfbarkeit: zum Beispiel Schutzmechanismen für kritische Bibliotheksfunktionen, Vermeidung von unzugänglichem Code, • gute Programmiertechnik: zum Beispiel Prüfen von Vor- und Nachbedingungen und Rücksprungbedingungen, Abfangen ungültiger Zustände.

3.4 Codierung (C)

Referenz	Maßnahme	Erläuterung
C_1	Dokumentation des Entwurfs (1.2)	Die Modul-Design-Spezifikation bildet die Grundlage für die Codierung. Mit der Dokumentation des Entwurfs ist die Bereitstellung des Quelltextes (englisch: codelistings) einschließlich der Kommentierung gemeint. Eine ausreichende Kommentierung des Codes ist zwingend erforderlich, um lesbaren, verständlichen, testbaren und wartbaren Code zu erhalten. Die Kommentierung des Codes sollte zum Beispiel eine Beschreibung der Algorithmen, weiterer Besonderheiten und einen Modulkopf/Header bereitstellen. Der Modulkopf/Header sollte mindestens folgende Angaben (siehe Beispiel im Anhang A) enthalten: • Nennung des Autors, • Beschreibung der Zielsetzung und Funktion im Gesamtzusammenhang des Moduls (nicht nur des Inhalts), • Ein- und Ausgaben, • Historie des Konfigurationsmanagements.
C_2	Dokumentation aller relevanten Aktivitäten während des Software-Sicherheits-lebenszyklus (2.2)	Ergänzend zu C_1 ist das Review der Codierung (siehe C_6) gegen die Modul-Design-Spezifikation zu dokumentieren. Siehe auch SyD_2.
C_3	modulare und strukturierte Codierung (1.3; 2.6)	Es sind die im Rahmen der Modulgestaltung (siehe MoD_3) festgelegten Maßnahmen, falls erforderlich mit Iterationen, auf Codeebene umzusetzen.
C_4	Verwendung geeigneter Programmiersprachen (2.5)	Die Eignung der verwendeten Programmiersprache bezieht sich auf den Nachweis der Eindeutigkeit und Nachvollziehbarkeit jedes Programms oder Programmteils innerhalb seiner Programmlaufzeit. Zum Beispiel muss bei einem Walk-Through der Zustand einer Variablen oder das Verhalten des betrachteten Moduls immer eindeutig definiert werden können. Gegebenenfalls ist der Sprachumfang so zu begrenzen, dass besonders fehlerträchtige Sprachkonstrukte ausgeschlossen werden.
C_5	Verwendung geeigneter rechnergestützter Werkzeuge mit Betriebsbewährung (2.5)	Die Verwendung rechnergestützter Werkzeuge im Rahmen der Codierung ist als wichtige Maßnahme zur Fehlervermeidung erforderlich. Die zu verwendenden Werkzeuge werden in der Software-Design-Spezifikation festgelegt (siehe SRSS_1). Der Begriff „Betriebsbewährung“ (englisch: confidence from use) wird in diesem Kontext so interpretiert, dass die Programmiererinnen und Programmierer im Umgang mit den Werkzeugen und in der Entwicklungsumgebung geschult wurden und die Werkzeuge in vergangenen Projekten ohne Mängel eingesetzt wurden. Hierfür sind Nachweise erforderlich.
C_6	Verifikation des Codes durch ein Walk-Through/ Review (2.7)	Nach Abschluss der Codierung sind die einzelnen Module im Rahmen eines Code-Reviews oder Code-Walk-Throughs gegen die Modul-Design-Spezifikation zu prüfen. Die Begutachtung erfolgt durch den bzw. die Programmiererinnen und Programmierer zusammen mit einer oder mehreren Personen mit gleichem technischen Sachverstand. Ziel des Reviews bzw. des Walk-Throughs ist die Überprüfung des Codes auf folgende Punkte: • Fehler oder potenzielle Fehler, • Qualität der Kommentare, • Einhaltung der Codierungsrichtlinien (kann auch toolunterstützt durchgeführt werden), • Übersichtlichkeit und Lesbarkeit, • Vollständigkeit bezüglich der zu implementierenden Funktionalität. Die durchzuführenden Maßnahmen bei einem Walk-Through und einem Review werden als identisch angesehen.

3.5 Modultests (MoT)

Referenz	Maßnahme	Erläuterung
MoT_1	Dokumentation aller relevanten Aktivitäten während des Software-Sicherheits-lebenszyklus (2.2)	Es sind die Modultests (siehe MoT_3 bis MoT_5) zu dokumentieren. Ziel ist der Nachweis, dass alle Anforderungen aus der Modul-Design-Spezifikation umgesetzt wurden. Siehe auch SyD_2.
MoT_2	Verwendung geeigneter rechnergestützter Werkzeuge mit Betriebsbewährung (2.5)	Auch im Rahmen der Modultests sind möglichst rechnergestützte Werkzeuge als wichtige Maßnahme zur Fehlervermeidung erforderlich. Die zu verwendenden Werkzeuge werden in der Software-Design-Spezifikation festgelegt (siehe SRSS_1). Zum Begriff „Betriebsbewährung“ siehe C_5. Die Betriebsbewährung muss entsprechend nachgewiesen und dokumentiert sein und bezieht sich an dieser Stelle speziell auf den Einsatz im Rahmen des Modultests.
MoT_3	Verifikation des Codes durch Kontrollflussanalyse (2.7)	Der Kontrollfluss des Programms – hier: eines Moduls – ist mithilfe eines Kontrollflussgraphen (Programmablaufplans) zu analysieren (und anhand der im Folgenden genannten Kriterien zu bewerten). Die Kontrollflussanalyse ist ein Werkzeug der statischen Codeanalyse und dient der Aufdeckung möglicher struktureller Mängel innerhalb eines Programmablaufs. Ein Kontrollflussgraph ist ein gerichteter Graph mit einem Anfangs- und einem Endknoten (Anfangs- und Endanweisung des Moduls). Dazwischen liegen weitere Knoten (Anweisungen), die über sogenannte Kanten miteinander verbunden sind. Die Kanten beschreiben den möglichen Kontrollfluss zwischen den Anweisungen. Die grafische Darstellung des Kontrollflusses darf neben Anfangs- und Endknoten nur folgende Elemente enthalten: a) Anweisungen (Knoten mit jeweils nur einer ein- und einer ausgehenden Kante), b) Verzweigungen (Knoten mit mindestens zwei Nachfolgeknoten), c) Vereinigungen (Knoten mit mindestens zwei eingehenden Kanten). Der Kontrollflussgraph muss schrittweise auf einen einzelnen Knoten reduzierbar sein. Ist dies nicht möglich, ist der Code nicht gut strukturiert und sollte optimiert werden.
MoT_4	Funktionale Tests, zum Beispiel Black-Box-Tests (1.6)	Funktionale Tests auf Modulebene sind durchzuführen, um die Übereinstimmung mit der Modul-Design-Spezifikation zu verifizieren. Dazu kann z. B. die Entwicklungsumgebung genutzt werden. Für die Durchführung der Tests ist unter Umständen die Programmierung eines Testrahmens (englisch: test bed) erforderlich. Testdaten, eventuelle Vor- und Nachbedingungen sowie Testergebnisse (z. B. Ausgaben, Änderung interner Zustände etc.) sind zu protokollieren. Ziel des Modultests ist die Sicherstellung, dass die Module ihrer in der Modul-Design-Spezifikation festgelegten Funktionalität gerecht werden. Die Funktionalität eines Moduls entspricht dabei ihrem Ein- und Ausgabeverhalten. Um dies zu überprüfen, werden jedem Modul Eingabedaten übergeben, die die erwartete Funktion angemessen wiedergeben. Diese sollten mindestens die folgenden Bereiche abdecken: • Daten aus zulässigen Bereichen, • Daten aus unzulässigen Bereichen, • Daten aus Grenzbereichen. Zulässige Bereiche und Bereichsgrenzen sind aus der Modul-Design-Spezifikation abzuleiten.

Referenz	Maßnahme	Erläuterung
MoT_5	erweiterte Funktionstests, zum Beispiel Grey-Box-Tests, Leistungstests oder Simulation (2.8)	Neben den in MoT_4 genannten Eingabedaten sind für sicherheitsrelevante Software in PL c und d auch ungültige Eingabedaten zu berücksichtigen. Es ist zu überprüfen, ob diese durch die Programmierung des Moduls abgefangen werden oder zu unerwünschten bzw. unerwarteten Reaktionen führen. Dies betrifft beispielsweise die folgenden Eingaben: • Divisor Null, • ASCII-Leerzeichen, • leerer Stack oder leeres Listenelement, • volle Matrix, • leerer Tabelleneintrag. In Testdurchläufen zur Fehleraufdeckung sollte ein möglichst hoher Anteil des Codes zur Ausführung gebracht werden. Dazu werden auf der Basis der Elemente des Kontrollflussgraphen (MoT_3) Überdeckungstests durchgeführt. Folgende Testmethoden sind zu unterscheiden: • Anweisungstest, • Zweig- oder Entscheidungstest, • Pfadtest, • Bedingungstest. Ziel sollte nach Möglichkeit eine hundertprozentige Abdeckung sein, da bei nicht ausgeführten Anweisungen keine Aussage zu deren Korrektheit möglich ist. Gibt es Anweisungen, die durch keinen Testfall zur Ausführung gebracht werden können, so kann dies ein Hinweis auf unerreichbaren Code (Dead Code) sein. Werden durch den Test bedingungslose Sprünge aufgedeckte, so ist der Code zu überarbeiten, um diese zu vermeiden.

3.6 Integrationstests (InT)

Referenz	Maßnahme	Erläuterung
InT_1	bei Verwendung softwarebasierter Maßnahmen zur Beherrschung von zufälligen Hardwareausfällen: Verifikation der korrekten Implementierung (1.5)	In Abhängigkeit von der gewählten Systemstruktur (Kategorie 2, 3 und 4) können zur Erfüllung des notwendigen Diagnosedeckungsgrades (englisch: diagnostic coverage) softwarebasierte Maßnahmen zur Beherrschung von zufälligen Hardwareausfällen notwendig werden. Softwarebasierte Maßnahmen beinhalten zum Beispiel • RAM-, ROM- und CPU-Tests, • Tests von Hardwarekomponenten, • Plausibilitätstests von Rechenergebnissen und Eingabewerten. Die Wirksamkeit und korrekte Implementierung solcher erforderlichen softwarebasierten Maßnahmen zur Fehlererkennung ist zu verifizieren. Dazu ist es nötig, die in SyD_1 definierten Testfälle auszuführen, um die jeweils angesprochenen Testroutinen zu verifizieren. Das Ergebnis dieser Verifikation ist zu dokumentieren. Für Realisierungen in Kategorie B (PL a oder b) fordert die Norm keine Beherrschung zufälliger Hardwareausfälle. Daher werden typischerweise auch keine entsprechenden softwarebasierten Maßnahmen umgesetzt.
InT_2	Dokumentation aller relevanter Aktivitäten während des Software-Sicherheitslebenszyklus (2.2)	Die Durchführung und die Ergebnisse der Integrationstests (siehe InT_1 und InT_4 bis InT_6) sind zu dokumentieren. Ziel ist der Nachweis, dass alle Anforderungen aus der Software-System-Design-Spezifikation umgesetzt wurden. Siehe auch SyD_2.
InT_3	Verwendung geeigneter rechnergestützter Werkzeuge mit Betriebsbewährung (2.5)	Die Verwendung rechnergestützter Werkzeuge im Rahmen der Integrationstests ist als wichtige Maßnahme zur Fehlervermeidung erforderlich. Die zu verwendenden Werkzeuge werden in der Software-Design-Spezifikation festgelegt (siehe SRSS_1). Zum Begriff „Betriebsbewährung“ siehe C_5. Die Betriebsbewährung muss entsprechend nachgewiesen und dokumentiert sein und bezieht sich an dieser Stelle speziell auf den Einsatz im Rahmen der Integrationstests.

Referenz	Maßnahme	Erläuterung
InT_4	Verifikation des Codes durch Kontrollflussanalyse (2.7)	Auf der Ebene der Gesamt-Software ist die statische Kontrollflussanalyse der verschiedenen Module mithilfe eines Aufrufdiagramms (englisch: call graph) durchzuführen. Das Aufrufdiagramm dient der Darstellung von Aufrufbeziehungen der einzelnen Module untereinander. Zu komplexe Aufrufstrukturen sind zu vermeiden. Wurden polymorphe Aufrufstrukturen verwendet, sind alle möglichen Aufrufmöglichkeiten anhand der vorliegenden Dokumentation zu analysieren (siehe SyD_3).
InT_5	funktionale Tests, zum Beispiel Black-Box-Tests (1.6)	Integrationstests betreffen in dieser Phase zum einen den Zusammenschluss der einzelnen Software-Module zu einer Gesamt-Software (Integrationstest auf Software-Ebene) und zum anderen – wenn möglich – die Integration der Gesamt-Software auf der Zielhardware (Integrationstest auf Hardware-Ebene). Mit den im Folgenden beschriebenen Tests ist die Übereinstimmung der Software mit der Software-System-Design-Spezifikation zu verifizieren. Der Integrationstest setzt voraus, dass die einzelnen Komponenten im Rahmen der Codierung und der Modultests bereits getestet und mögliche Mängel bereits behoben wurden. Die Komponenten werden dann zu immer größer werdenden Teilsystemen zusammengefügt. Tests auf Software-Ebene betreffen die Schnittstellen und das Zusammenspiel der einzelnen Komponenten. Da bis zur Fertigstellung der Gesamtsoftware die Teilsysteme für sich genommen unter Umständen noch nicht lauffähig sind, kann es notwendig werden, für die noch fehlenden Komponenten einen Testrahmen (siehe auch MoT_4) zu erstellen. Tests auf Integrationsebene sollen folgende Fehler aufdecken: • inkompatible Schnittstellenformate und Protokollfehler, • widersprüchliche Interpretation von Daten, • Daten werden zu spät oder in zu kurzen Intervallen übergeben. Die Testumgebung des Integrationstests auf Hardware-Ebene sollte der zu erwartenden späteren Betriebsumgebung möglichst nahekommen. Entsprechend sollte als Testumgebung installierte Soft- und Hardware-Peripherie möglichst der später tatsächlich zum Einsatz kommenden Peripherie entsprechen. Dem System werden Eingabedaten übergeben, die den erwarteten Normalbetrieb angemessen wiedergeben. Die funktionalen Tests müssen alle realistischerweise vorhersehbaren Eingangsbedingungen abdecken und die spezifizierten Ergebnisse (inklusive zum Beispiel der Reaktionszeit) liefern. Die Ergebnisse werden beobachtet und mit den in der Software-System-Design-Spezifikation festgelegten Vorgaben verglichen. Dabei sind gegebenenfalls vorher nicht schriftlich festgelegte Anforderungen nachzutragen.
InT_6	erweiterte Funktionstests, zum Beispiel Grey-Box-Tests, Leistungstests oder Simulation (2.8)	Neben den in InT_5 beschriebenen Tests sind erweiterte Funktionstests durchzuführen. Im Rahmen der erweiterten Funktionstests wird das Verhalten der festgelegten Sicherheitsfunktionen im Falle seltener oder nicht festgelegter Eingaben verifiziert. Es können zum Beispiel • Funktionstests mit Eingabebedingungen durchgeführt werden, die nur selten erwartet werden oder die außerhalb der festgelegten Eigenschaften der Sicherheitsfunktionen liegen, • Funktionstests mit Randbedingungen unter Grenzbedingungen (zum Beispiel extreme Umgebungsbedingungen, Bauteildimensionierung an den Grenzwerten, maximale thermische Bauteilauslastung, maximale Prozessorauslastung) durchgeführt werden, • Funktionstests auf der Basis des Wissens der internen Codierung der gesamten Software durchgeführt werden, • Funktionstests von Hardware-Elementen der Sicherheitsfunktionen (zum Beispiel elektronische Schaltkreise) durch Softwaresimulation ausgeführt werden.

3.7 Validierung (V)

Referenz	Maßnahme	Erläuterung
V_1	Dokumentation aller relevanten Aktivitäten während des Software-Sicherheits-lebenszyklus (2.2)	Die Validierung (siehe V_2) ist zu dokumentieren. Ziel ist der Nachweis, dass alle Anforderungen aus der Software-Design-Spezifikation umgesetzt wurden. Siehe auch SyD_2.
V_2	Funktionale Tests (1.6)	Im Sinne der Norm ist die Validierung der finale Abgleich der durchgeführten Prüfungen und deren Testergebnisse in den einzelnen Phasen des V-Modells gegen die Anforderungen der Software-Design-Spezifikation. Wenn bereits die Integration auf der Ziel-Hardware stattgefunden hat, umfasst die Validierung auch den Abgleich gegen die Spezifikation der Sicherheitsanforderungen (SRS). Bei Einzelkomponenten wird die Validierung vor dem Inverkehrbringen durchgeführt. In der Regel finden keine rein validierungsbezogenen funktionalen Tests mehr statt. In diesem Fall fallen Integrationstest und Validierung zusammen. Bei Maschinen ist die Validierung Teil der Installation und des Abnahmetests beim Betreiber. Der Abgleich gegen die Spezifikation der Sicherheitsanforderungen (SRS) kann an dieser Stelle durch zusätzliche funktionale Tests in der realen Umgebung erfolgen.

4 Allgemeine Maßnahmen

4.1 Qualitätsmanagement (QM)

Referenz	Maßnahme	Erläuterung
QM_1	Software-Sicherheitslebenszyklus mit Verifikation und Validierung, Anwendung des V-Modells (1.1)	Das V-Modell (Abbildung 2) ist mit dem Ziel anzuwenden, die Entwicklung der Software und den Nachweis ihrer Eignung in definierte Phasen und Tätigkeiten zu strukturieren. Dies führt im absteigenden Ast des V-Modells schrittweise zu detaillierteren Spezifikationen. Nach der Codierung der einzelnen Software-Module werden diese im aufsteigenden Ast integriert und getestet.
QM_2	Projektmanagement- und Qualitätsmanagementsystem vergleichbar mit zum Beispiel der Reihe DIN EN 61508 oder DIN EN ISO 9001 (2.1)	Projekt- und Qualitätsmanagement sind anzuwenden, um Fehler im Rahmen der Softwareentwicklung zu vermeiden. Wichtige Maßnahmen zur Umsetzung dieser Anforderung sind: • Erstellung eines Qualitätssicherungshandbuchs, das ein Organisationsmodell speziell für die Qualitätssicherung beschreibt, • Definition einer Entwurfsorganisation mit Definition der Aufgaben und Verantwortlichkeiten der organisatorischen Einheiten, Befugnisse der Abteilung für die Qualitätssicherung, Unabhängigkeit der Qualitätssicherung (interne Inspektion) von der Entwicklung, • Definition des Ablaufplans (Phasenmodelle) mit Festlegung aller Tätigkeiten, die während der Durchführung des Projekts sachdienlich sind, einschließlich der internen Inspektionen und deren Zeitplanung sowie der Projektaktualisierung, • Definition eines feststehenden Ablaufs einer internen Inspektion mit Planung, Ausführung und Überprüfung der Inspektion, Freigabemechanismen für Teilprodukte, Sicherung von Wiederholungsinspektionen, • Konfigurationsmanagement mit Verwaltung und Kontrolle der Versionen nach QM_3, • Einführung von rechnerunterstützten Werkzeugen, Durchführung und Nachweis der Personalschulung, • durchgängige Anwendung des Vier-Augen-Prinzips. Wird Software nach DIN EN 61508 entwickelt, so sind die genannten Anforderungen bereits berücksichtigt. Liegt der Nachweis eines Qualitätsmanagementsystems nach DIN EN ISO 9001 vor, so sind für eine Software-Entwicklung die genannten Maßnahmen speziell umzusetzen.
QM_3	Konfigurationsmanagement (2.3)	Um Fehler während des Entwicklungsprozesses sicherheitsrelevanter Software zu vermeiden und ein nachfolgendes Review der gesamten Software unter Berücksichtigung der zu betrachtenden Sicherheitsfunktionen zu ermöglichen, ist ein Software-Konfigurationsmanagement anzuwenden. Es müssen Verfahren existieren: a) zur eindeutigen Identifikation aller Bestandteile eines Softwareprojekts, mit u. a. • Versionsverwaltung des Source-Codes der einzelnen Softwaremodule und der gesamten Software, • Spezifikation der Software und Entwurfsdokumente, • Testplänen und Testergebnissen, • Verifikationsdokumenten, • bereits existierenden Softwareelementen, sofern vorhanden, • rechnerunterstützten Werkzeugen und Entwicklungsumgebungen, die zur Erstellung der Software verwendet werden, b) zur Verwaltung von Software-Modifikationen, um sicherzustellen, dass die spezifizierten Anforderungen bei Änderungen weiterhin erfüllt werden, mit u. a. Maßnahmen • zur Verhinderung von nicht autorisierten Modifikationen, • zur Verhinderung des Einsatzes von nicht zugelassenen Komponenten, • zur Analyse des Einflusses von beabsichtigten Modifikationen, • zur Freigabe von beabsichtigten Modifikationen, c) zur Aufbewahrung der Software und aller zugehörigen Dokumentation. Das Software-Konfigurationsmanagement kann in der einfachsten Ausprägung manuell gehandhabt werden, sinnvoller und wirksamer ist allerdings der Einsatz geeigneter rechnerunterstützter Werkzeuge und Entwicklungsumgebungen. d) zur Freigabe der gesamten Software nach Abschluss des Entwicklungsprozesses.

4.2 Systematische Ausfälle (SyF)

Referenz	Maßnahme	Erläuterung
SyF_1	Beherrschung systematischer Ausfälle (1.4)	In einigen Fällen, die in DIN EN ISO 13849-1, Abschnitt G.2 genannt werden, dienen softwaretechnische Maßnahmen zur Beherrschung systematischer Ausfälle. Zur Verifikation der korrekten und vollständigen Umsetzung sind ausgehend von der Spezifikation der Sicherheitsanforderungen (SRS) die für das Projekt anzuwendenden Maßnahmen in der Software-Design-Spezifikation zu beschreiben und deren Umsetzung in der Software-System-Design-Spezifikation und der Modul-Design-Spezifikation zu dokumentieren. Diese Anforderungen betreffen die folgenden in Abschnitt G.2 genannten Maßnahmen: • Überwachung des Programmablaufs: Je nach gewählter Hardwarestruktur können verschiedene Überwachungen möglich sein, die sich am PL_r orientieren sollten. Beispiele sind ein externer Hardware-Watchdog, der aus der Software heraus getriggert wird oder eine zeitlich-logische Programmlaufüberwachung, die durch die Kommunikation mit einem Parallelkanal realisiert wird. Unabhängig von der Realisierung der Überwachung muss bei einem erkannten Fehler der sichere Zustand (zum Beispiel der energielose Zustand) eingenommen werden. • Maßnahmen zur Beherrschung von Fehlern im Datenkommunikationsprozess: Wenn die Ausführung einer Sicherheitsfunktion sichere Datenkommunikation (z. B. über Bussysteme) erfordert, müssen Maßnahmen zur Beherrschung der folgenden Fehlermodelle für sicherheitsgerichtete Nachrichten getroffen werden: — Wiederholung, — Verlust, — Einfügung, — falsche Abfolge, — Verfälschung, — Verzögerung, — Maskerade. Zu weiteren Details sicherer Datenkommunikation siehe DIN EN 61784-3 [7]. Die korrekte Implementierung der Maßnahmen zur Beherrschung von systematischen Ausfällen ist in jeder Phase des V-Modells zu berücksichtigen und zu verifizieren, insbesondere in den Phasen der Spezifikation und Validierung.

4.3 Modifikationsmanagement (MM)

Referenz	Maßnahme	Erläuterung
MM_1	geeignete Aktivitäten für den Software-Sicherheitslebenszyklus nach Änderungen (1.7)	Vor der Änderung der Software sollte eine Analyse der möglichen Auswirkungen – eine sogenannte Einflussanalyse – ausgeführt sowie die betroffenen Softwaremodule ermittelt werden. Eine nicht zulässige Auswirkung könnte etwa eine Minderung der festgelegten Eigenschaften der Sicherheitsfunktionen zum Beispiel in Bezug auf Ansprechzeit oder -verhalten sein. Nach Abschluss der Einflussanalyse ist in Abhängigkeit des PL_r zu entscheiden, auf welchen Ebenen des V-Modells welche Tests nach Änderungsabschluss wiederholt werden müssen. Dies kann anhand der folgenden Kriterien geschehen: • Wiederholung nur der Tests, denen eine hohe Priorität zugeordnet wird • bei Funktionstests Verzicht auf gewisse Varianten (Sonderfälle) • Einschränkung der Tests auf bestimmte Konfigurationen • Einschränkung der Tests auf bestimmte Teilsysteme Um die Grundlage für die Bewertung dieser Kriterien zu schaffen, ist auf den Testebenen des V-Modells (Abschnitt 3.5 bis 3.7) eine detaillierte Dokumentation, Begründung und Bewertung der (je Softwaremodul) durchgeführten Testfälle notwendig. Hierbei sollten auch die Tests gekennzeichnet werden, die bei einer notwendigen Wiederholung nach Änderung berücksichtigt werden sollten.
MM_2	Einflussanalyse und angemessene Software-Sicherheitslebenszylus-Aktivitäten nach Änderungen (2.9)	Die in MM_1 beschriebene Einflussanalyse muss zwingend durchgeführt werden. Die dort genannten Kriterien sind wegen des höheren PL_r strenger auszulegen, sodass eine größere Zahl von Tests wiederholt werden müssen. Die durchgeführten Verifikationsschritte sind zu dokumentieren.

Literatur

[1] DIN EN ISO 13849-1: Sicherheit von Maschinen – Sicherheitsbezogene Teile von Steuerungen – Teil 1: Allgemeine Gestaltungsleitsätze (06.16). Berlin: Beuth 2016.

[2] *Huelke, M.; Becker, N.; Eggeling, M.:* Sicherheitsbezogene Anwendungssoftware von Maschinen – Die Matrixmethode des IFA. IFA Report 2/2016. Hrsg.: Deutsche Gesetzliche Unfallversicherung e. V. (DGUV), Berlin 2016.
www.dguv.de, Webcode: d1023063

[3] Software-Assistent SOFTEMA: Spezifikation zur IFA-Matrixmethode bei sicherheitsbezogener Anwendungssoftware. Hrsg.: Deutsche Gesetzliche Unfallversicherung e. V . (DGUV), Sankt Augustin.
www.dguv.de, Webcode: d1082520

[4] *Hauke, M.; Schaefer, M.; Apfeld, R.; Bömer, T.; Huelke, M.; Borowski, T.* et al.: Funktionale Sicherheit von Maschinensteuerungen – Anwendung der DIN EN ISO 13849. IFA Report 2/2017. Hrsg.: Deutsche Gesetzliche Unfallversicherung e. V. (DGUV), Berlin 2017.
www.dguv.de, Webcode: d1099283

[5] *Apfeld, R.; Hauke, M.; Otto, S.:* Das SISTEMA-Kochbuch 6: Definition von Sicherheitsfunktionen – Was ist wichtig? Hrsg.: Deutsche Gesetzliche Unfallversicherung e. V. (DGUV), Berlin 2015.
www.dguv.de, Webcode: d109240

[6] DIN EN 61508-5-3: Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme – Teil 3: Anforderungen an Software (02.11). Berlin: Beuth 2011

[7] DIN EN 61784-3: Industrielle Kommunikationsnetze – Profile – Teil 3: Funktional sichere Übertragung bei Feldbussen – Allgemeine Regeln und Festlegungen für Profile (09/17). Berlin: Beuth 2017

Anhang A:

Beispiel Modulkopf/Header

Beispiel Modulkopf/Header

Gemäß der ab PL_r a verpflichtend anzuwendenden Basismaßnahme C_1 muss jedem Modul (jeder Methode bei objektorientierter Programmierung) ein eigener Modulkopf vorangestellt werden, der das „wer, wann, was, warum und wie“ festhält. Der Modulkopf sollte als Minimum eine Auflistung folgender Angaben enthalten:

- Modulname,
- Kurzbeschreibung der Funktion,
- Versionsnummer,
- Legende der Änderungen,
- Datum der letzten Änderung,
- Bearbeitende,
- Eingabeparameter,
- Ausgabeparameter,
- veränderte Register und Speicherbereiche,
- Fehleraussprung,
- übergeordnete Module (called by),
- untergeordnete Module (calls),
- Stacktiefe (zum Beispiel bei Assembler),
- veränderte Flags (zum Beispiel bei Assembler)

und weiterhin, falls erforderlich:

- verwendete Registerbank (z. B. bei Assembler),
- Durchlaufzeit,
- Timeouts,
- besondere Hinweise.

Beispiel eines Modulkopfes

```
/* ****************************************** */
/*                                            */
/* Modul:                                     */
/*                                            */
/* Version: ... Bearbeitet durch: ... Datum:  */
/* Version: ... Bearbeitet durch: ... Datum:  */
/* Version: ... Bearbeitet durch: ... Datum:  */
/*                                            */
/* Funktion des Moduls:                       */
/*                                            */
/*                                            */
/*                                            */
/* Eingabeparameter:                          */
/* Ausgabeparameter:                          */
/*                                            */
/* Veränderte Register:                       */
/* Veränderte Speicherbereiche:               */
/* Veränderte Flags:                          */
/*                                            */
/* Fehleraussprung:                           */
/*                                            */
/* Übergeordnete Module:                      */
/* Untergeordnete Module:                     */
/*                                            */
/* Stacktiefe:                                */
/*                                            */
/* Verwendete Registerbank:                   */
/*                                            */
/* Besondere Hinweise:                        */
/*                                            */
/* ****************************************** */
```

Anhang B:

Erläuterung der Abkürzungen

Erläuterung der Abkürzungen	
C	Codierung (englisch: **C**oding)
CPU	**C**entral **P**rocessing **U**nit
FVL	Programmiersprache mit nicht eingeschränktem Sprachumfang (englisch: **F**ull **V**ariability **L**anguage)
InT	Integrationstests (englisch: **In**tegration **T**esting)
LVL	Programmiersprache mit eingeschränktem Sprachumfang (englisch: **L**imited **V**ariability **L**anguage)
MDS	Modul-Design-Spezifikation (englisch: **M**odule **D**esign **S**pecification)
MM	Modifikationsmanagement (englisch: **M**odification **M**anagement)
MoD	Modulgestaltung (englisch: **M**odule **D**esign)
MoT	Modultests (englisch: **M**odule **T**esting)
PL	**P**erformance **L**evel
PL_r	Erforderlicher Performance Level (englisch: **r**equired **P**erformance **L**evel)
RAM	**R**andom **A**ccess **M**emory
ROM	**R**ead **O**nly **M**emory
SDS	Software-Design-Spezifikation (englisch: **S**oftware **D**esign **S**pecification)
SRASW	**S**afety **R**elated **A**pplication **S**oft**w**are
SRESW	**S**afety **R**elated **E**mbedded **S**oft**w**are
SRS	Spezifikation der Sicherheitsanforderungen (englisch: **S**afety **R**equirement **S**pecification)
SRSS	Sicherheitsbezogene Software-Spezifikation (englisch: **S**afety **R**elated **S**oftware **S**pecification)
SSDS	Software-System-Design-Spezifikation (englisch: **S**oftware **S**ystem **D**esign **S**pecification)
SyD	Systemgestaltung (englisch: **Sy**stem **D**esign)
SyF	Systematische Ausfälle (englisch: **Sy**stematic **F**ailure)
QM	Qualitätsmanagement (englisch: **Q**uality **M**anagement)
V	Validierung (englisch: **V**alidation)