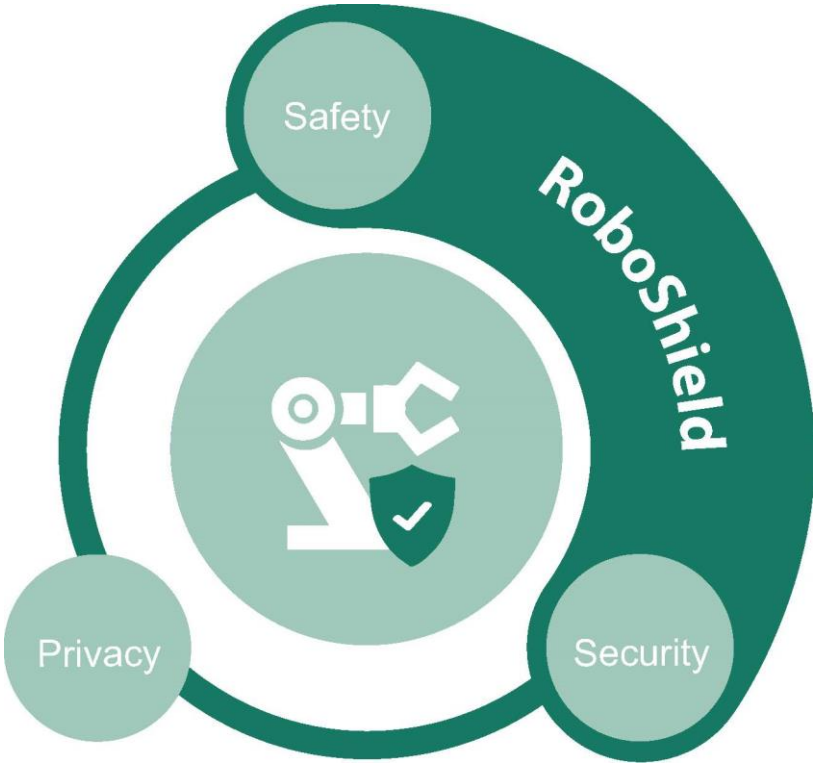
# Einführung in die Entwicklung sicherheitszertifizierter Software

entstanden im Rahmen eines
RoboShield-QuickChecks




Durchgeführt von:

Patrick Schlosser, M.Sc.
Tom Huck, M.Sc.
Dr.-Ing. Christoph Ledermann

Karlsruher Institut für Technologie (KIT)
Institut für Anthropomatik und Robotik –
Intelligente Prozessautomation und Robotik (IAR-IPR)

## Haftungsausschluss & Ziel des Dokuments

Das vorliegende Dokument dient lediglich zur Orientierung sowie zu Informationszwecken und soll einen ersten Einblick in Anforderungen und das Vorgehen bei der Entwicklung sicherheitszertifizierter Software liefern. Es stellt keine Grundlage für die tatsächliche Entwicklung und/oder Zertifizierung eines sicherheitszertifizierten Produkts dar und darf hierfür nicht verwendet werden. Enthaltene Informationen wurden sorgfältig recherchiert, es besteht jedoch weder ein Anspruch auf Korrektheit noch auf Vollständigkeit. Zu Teilen enthält das Dokument Interpretationen der Anforderungen aus Normen durch die Autoren. Es wird keinerlei Haftung für Schäden und jedwede weitere Folgen, die sich aus der Verwendung dieses Dokuments ergeben, übernommen.

Die alleinige Grundlage für die Entwicklung eines sicherheitszertifizierten Produkts stellen die jeweils geltenden gesetzlichen Vorschriften sowie die gegebenenfalls hierzu harmonisierte Normen dar.

## Nennung von Firmen und Produkten

Im Rahmen des Dokuments werden verschiedene Firmen und teilweise durch sie vermarktete Produkte aufgeführt. Diese dienen lediglich als Beispiele und zur Veranschaulichung und sollen keine Empfehlung darstellen. Die Erstellung dieses Dokuments wurde durch genannte Firmen nicht finanziell unterstützt.

# 1. Einführung

Bei der Entwicklung sicherer Software geht es darum, Software zu entwickeln, die mit den gesetzlichen Sicherheitsvorschriften des jeweiligen Landes konform ist. Sicherheit bezieht sich im Rahmen dieses Dokuments lediglich auf die funktionale Sicherheit; Informationen zur IT-Sicherheitszertifizierung können in Deutschland beispielsweise beim Bundesamt für Sicherheit in der Informationstechnik (BSI) eingeholt werden.

Grundsätzlich stellt sich die Frage, für welche Anwendungen die Entwicklung sicherer Software notwendig ist und wie sich diese Entwicklung gestaltet. Diese Frage wird innerhalb der EU durch die Maschinenrichtlinie (Richtlinie 2006/42/EG) beantwortet, welche im deutschen Recht vornehmlich im Produktsicherheitsgesetz und in der Maschinenverordnung verankert ist. Stark vereinfacht ausgedrückt ist die Entwicklung sicherer Software notwendig, sobald ein Verletzungsrisiko für den Menschen besteht - je höher und schwerwiegender dieses Risiko ist, desto schärfere Sicherheitsvorschriften müssen beachtet werden. Konkret werden die einzuhaltenden Regeln in Abhängigkeit zum Anwendungsrisiko festgelegt, beispielsweise durch die Safety Integrity Level (SIL) der Norm IEC 61508, durch die Performance Leveln (PL) der Norm ISO 13849 oder durch die Automotive Safety Integrity Level (ASIL) der Norm ISO 26262. Welche konkrete Norm anzuwenden ist, hängt vom Anwendungsfall ab.

Um die Einhaltung der gesetzlichen Gegebenheiten zu vereinfachen, existieren auf EU-Ebene sogenannte harmonisierte Normen für jeweilige Einsatzgebiete. Diese harmonisierten Normen enthalten Regeln und Vorschriften für die Produktentwicklung, bei deren Einhaltung davon ausgegangen werden darf, dass das Produkt die gesetzlich verankerten Anforderungen erfüllt. Für die Industrierobotik ist beispielsweise die harmonisierte Norm ISO 10218 vorgesehen, welche unter anderem auf die bereits erwähnten Normen IEC 61508 und ISO 13849 verweist. Zu beachten ist, dass die Einhaltung der gesetzlichen Vorschriften für die vorgesehenen Anwendungen im Regelfall in Eigenverantwortung (“Selbstüberprüfung”) geschieht: Erst wenn ein Schadensfall eintritt, muss die Einhaltung nachgewiesen werden können.

Für den Einstieg in die Entwicklung sicherer Software ist vor allem anzumerken, dass sich der Entwicklungsprozess sicherer Software im Regelfall drastisch von heute gängigen Entwicklungsprozessen (wie z.B. SCRUM) unterscheidet und oftmals ein Vielfaches des Entwicklungsaufwands von regulärer Software bedeutet. Von Beginn der Softwareentwicklung an müssen die Anforderungen für das angestrebte Sicherheitsniveau beachtet werden, der Entwicklungs-, Dokumentations und Testprozess muss gegebenenfalls angepasst werden.

Gerade für kleine und junge Unternehmen stellt die kostenintensive Entwicklung eines sicherheitszertifizierten Produkts eine finanzielle Herausforderung dar, die kaum zu bewältigen ist. Oftmals existiert daher die Notwendigkeit, ein funktionierendes Produkt zu entwickeln und zu vermarkten, mit dem Wunsch, dessen Anwendungsbereich zu einem späteren Zeitpunkt auf sicherheitskritische Anwendungen ausweiten zu können. Dies widerspricht jedoch fundamental dem Grundgedanken der Entwicklung sicherer Software: Hier soll von vorneherein die sichere Entwicklung auf einen konkreten Anwendungsfall und/oder ein konkretes Sicherheitsniveau zugeschnitten erfolgen. Dennoch ist es möglich, eine nachträgliche Sicherheitszertifizierung zu erreichen. Vereinfacht ausgedrückt muss nachgewiesen werden, dass der angewendete Entwicklungsprozess gleichwertig oder besser zum vorgeschriebenen Entwicklungsprozess ist; an Stellen, an denen dies nicht gegeben ist, müssen zusätzliche Nachweise erbracht werden.

Um vor allem KMU einen Einblick in die Welt der Entwicklung sicherer Software zu gewähren und gängige Fragen zur sicheren Softwareentwicklung zu beantworten, werden im Rahmen dieses

Dokuments zuerst Grundlage zu Normen und Sicherheitslevels vorgestellt. Anschließend werden basierend auf den gängigen Safety-Normen (primär IEC 61508) die Wege zur Entwicklung sicherer Software vorgestellt. Abschließend wird auf häufige Fragen im Hinblick auf den Einsatz von Compilern und Open-Source-Software bei sicherheitskritischer Software eingegangen.

Hinweis: Abgesehen von den Anforderungen an den SW-Entwicklungsprozess an sich werden auch Anforderungen an die Organisationsstruktur im Unternehmen gestellt. So soll z.B. ein Safety-Manager unternehmensweit safety-konforme Entwicklungsprozesse definieren und Templates erstellen, um die Überwachung und Dokumentation des Entwicklungsprozesses einfach und nachvollziehbar zu gestalten. Weitere Mitarbeiter sollen als dedizierte Safety-Ingenieure den Entwicklungs- und Dokumentationsprozess entsprechend der Norm begleiten und überwachen. Generell soll bei allen beteiligten Entwicklern das grundsätzliche Bewusstsein für Safety geschärft sein. Diese allgemeinen organisatorischen Randbedingungen gilt es ebenfalls zu beachten, sie sind jedoch nicht Gegenstand des vorliegenden Dokuments.

## 2. Normen und Sicherheitslevels

**Allgemeines:** Obwohl grundsätzliche Konzepte und Vorgehensweisen in der Entwicklung funktional sicherer Systeme ähnlich sind, unterscheiden sich konkrete Anforderungen, zu beachtende Normen und erforderliche Sicherheitslevels je nach Anwendungsfall. Es ist daher wichtig, Anwendungsszenarien für ein Produkt frühzeitig zu definieren. Dieser Abschnitt soll einen kurzen Überblick über die verschiedenen Normen zur funktionalen Sicherheit sowie einige Beispiele geben, erhebt jedoch keinen Anspruch auf Vollständigkeit.

**IEC 61508 und verwandte Normen:** In diesem Dokument liegt der Fokus auf der Norm IEC 61508. Sie gilt gemeinhin als die "Basisnorm" zur funktionalen Sicherheit und eignet sich gut dafür, grundsätzliche Vorgehensweisen darzustellen. Wie bereits in der Einleitung erwähnt, gibt es jedoch für viele Anwendungsbereiche eigene Normen, die zumeist branchenspezifische Abwandlungen der IEC 61508 sind oder auf diese Norm verweisen. Einige dieser Normen sind in Tabelle 1 beispielhaft aufgeführt. Weiterhin existieren auch spezialisierte Normen, die auf bestimmte Produkte oder Produktgruppen zugeschnitten sind. Grundsätzlich gilt: Für einen gegebenen Anwendungsfall haben die Bestimmungen der jeweils spezifischsten Norm Vorrang. Einen tiefergehenden Überblick über die Normenlandschaft bietet die Infografik "Functional Safety in a Nutshell" des TÜV Süd[1].

| **Branche** | **Beispiele branchenspezifischer Normen**<br>(abgeleitet von, oder mit Verweis auf IEC 61508) |
|---|---|
| Automatisierungstechnik, Maschinenbau | IEC 62061: Maschinensteuerungen (E/E/PE[2])<br>ISO 13849: Maschinensteuerungen allgemein[3] (E/E/PE, aber auch hydraulisch, pneumatisch, mechanisch)<br>ISO 13850: Not-Aus<br>ISO 10218: Industrieroboter |
| Transport | ISO 26262: Straßenfahrzeuge<br>EN 5012x: Eisenbahn |
| Medizintechnik | IEC 60601 |
| Energie | IEC 62109 |

Tabelle 1: Beispiele für branchenspezifische Sicherheitsnormen.

**Sicherheitslevels und -ziele:** Hat man die für die jeweilige Anwendung relevante Norm bestimmt, muss das geeignete Sicherheitslevel ausgewählt werden. Dieses bestimmt, welche Maßnahmen bei der sicherheitsgerichteten Entwicklung getroffen werden müssen. Die Definition der Sicherheitslevels variiert mit den jeweiligen Normen. Beispiele sind SIL ("Safety Integrity Level") aus der IEC 61508, ASIL ("Automotive Safety Integrity Level") aus der ISO 26262 oder PL ("Performance Level") aus der ISO 13849. Diese Sicherheitslevel bestimmen die Maßnahmen, die

[1] TÜV Süd AG: *Functional Safety in a Nutshell*, 2019. Download unter https://www.tuvsud.com/en/resource-centre/infographics/functional-safety-regulation-landscape
[2] E/E/PE: elektrisch (E), elektronisch (E) und programmierbar elektronisch (PE)
[3] Zu den Unterschieden zwischen IEC 61508, IEC 62061 und ISO 13849 siehe den Artikel von Steffens und Schepers: *Die drei Standards der Maschinensicherheit: EN ISO 13849, EN 62061 und IEC 61508*, 2016. Verfügbar unter: https://www.elektrotechnik.vogel.de/die-drei-standards-der-maschinensicherheit-en-iso-13849-en-62061-und-iec-61508-a-394828/

getroffen werden müssen. Die Einhaltung dieser Maßnahmen soll zum Erreichen bestimmter Fehlerwahrscheinlichkeiten führen. Die Wahrscheinlichkeiten sind als Auftretenswahrscheinlichkeit gefährlicher Fehler pro Stunde (“Probability of Dangerous Failure per Hour”, PFH) oder - z.B. für selten in Anspruch genommene Sicherheitsfunktionen - pro Inanspruchnahme der Funktion (“Probability of Failure on Demand”, PFD) definiert. Außerdem gibt es, je nach Norm, noch weitere Kriterien wie zum Beispiel die “Kategorie” (ein Maß für die Redundanz und Möglichkeit der Fehlererkennung) in ISO 13849 oder die “Systematic Capability” (SC) in der IEC 61508 (die SC ist ein Maß für die Vermeidungsmöglichkeit systematischer Fehler und wird in Kapitel III eingehend erläutert).

An dieser Stelle sei darauf hingewiesen, dass die Normen - zumindest im Kontext der Software-Entwicklung - primär Maßnahmen empfehlen, die durch Verbesserung der Software-Qualität zur Einhaltung der Sicherheitsziele beitragen. Ein rechnerischer Nachweis, dass das System tatsächlich die gewünschte Fehlerwahrscheinlichkeit erreicht, lässt sich für Software nicht führen. Dies ist nur im Bereich der Hardware möglich, und dann auch nur, wenn entsprechende Daten zur Ausfallwahrscheinlichkeiten von Bauteilen vorliegen. Dennoch sind in Tabelle 2 PFH und PFD-Werte aufgeführt, um einen Eindruck über die Größenordnung der angestrebten Fehlerraten zu erhalten.

Die Vorgehensweise zur Bestimmung des Sicherheitslevels variiert mit der anzuwenden Norm. Meist basiert Sie jedoch auf einer Gefährdungsanalyse wie HAZOP (siehe IEC 61882) oder einer Risikobeurteilung (siehe ISO 12100). Dabei werden Ausfälle oder Fehlfunktionen identifiziert, die zu einem Schadensereignis führen können. Für die entsprechenden Gefährdungen werden Sicherheitsfunktionen definiert, die das System bei Auftreten des Gefährdungsereignisses in einen sicheren Zustand überführen. Den Sicherheitsfunktionen werden dann, in Abhängigkeit von Eintrittswahrscheinlichkeit und Schwere des Schadens, Sicherheitslevel zugewiesen. Ein Beispiel hierfür ist der Risikograph aus Anhang A der ISO 13849. In manchen Fällen gibt es auch produktspezifische Normen, die - unabhängig von einer Gefährdungsanalyse - ein konkretes Sicherheitslevel definieren. So wird beispielsweise in ISO 10218 für Sicherheitsfunktionen von Industrierobotern generell SIL 2 bzw. PL d gefordert.

| SIL | PFH | PFD |
|---|---|---|
| 1 | $\geq 10^{-6}$ bis $< 10^{-5}$ | $\geq 10^{-2}$ bis $< 10^{-1}$ |
| 2 | $\geq 10^{-7}$ bis $< 10^{-6}$ | $\geq 10^{-3}$ bis $< 10^{-2}$ |
| 3 | $\geq 10^{-8}$ bis $< 10^{-7}$ | $\geq 10^{-4}$ bis $< 10^{-3}$ |
| 4 | $\geq 10^{-9}$ bis $< 10^{-8}$ | $\geq 10^{-5}$ bis $< 10^{-4}$ |

Tabelle 2: Fehlerraten für SIL 1-4, gemäß IEC 61508-1, Kapitel 7, Tabelle 2

Zusammengefasst beginnt die Entwicklung eines sicherheitszertifizierten Produktes also damit, das Einsatzgebiet festzulegen, die anzuwendende Norm auszuwählen, Sicherheitsrisiken zu identifizieren und zu klassifizieren und anschließend geeignete Sicherheitsfunktionen zur Reduzierung/Vermeidung von Risiken zu definieren.

# 3. Wege zur sicherheitszertifizierten Entwicklung

Im Rahmen dieses Abschnitts werden die Möglichkeiten zur Entwicklung sicherer Software anhand der gängigen Norm IEC 61508-3 vorgestellt. Bevor Sie sich an den Vorschriften dieser Norm orientieren gehen Sie sicher, dass die zur Maschinenrichtlinie harmonisierte Norm für Ihr geplantes Einsatzgebiet die Anwendung von IEC 61508-3 vorsieht.

Zur Entwicklung sicherer Software gibt es prinzipiell drei verschiedene Vorgehensweisen: “Compliant Development”, “Proven in Use” und “Assessment of non-compliant Development”. Von diesen drei Möglichkeiten ist die Vorgehensweise “Proven in Use” für Software in aller Regel unpraktikabel: Als Voraussetzungen müsste hierzu die Software in unveränderter Form seit geraumer Zeit im Einsatz sein - eine Voraussetzung, die durch Patches und die regelmäßige Veröffentlichung neuerer Versionen zumindest heutzutage so gut wie niemals gegeben ist. Daher wird dieser Pfad zur Entwicklung sicherer Software im Rahmen dieses Dokumentes nicht weiter behandelt. Bei Interesse kann jedoch die Norm IEC 61508-2, Abschnitt 7.4.10 konsultiert werden.

Als in der Praxis praktikable Wege für Software stehen folglich “Compliant Development” und “Assessment of non-compliant Development” zur Verfügung.
“Compliant Development”, im Folgenden auch als “Weg 1” bezeichnet, sieht die Entwicklung von Software in Übereinstimmung mit den in IEC 61508-3 enthaltenen Vorschriften vor. Dieser Weg sollte gewählt werden, falls sich das Produkt noch in der Planung befindet.
Sollte ein Produkt bereits existieren, kann der Weg “Assessment of non-compliant Development”, im Folgenden auch “Weg 3” genannt, beschritten werden. Kernaspekt von Weg 3 ist es nachzuweisen, dass der verwendete Entwicklungsprozess die Vorschriften für Weg 1 einhält oder übertrifft. An Stellen, an denen dies nicht der Fall ist, muss nachgebessert werden (z.B. bei der Dokumentation) und/oder weitere Nachweise müssen erbracht werden. Auch wenn bereits ein Produkt existiert, ist Weg 3 nicht immer der beste Weg: Der Zeitaufwand für Vergleiche von Verfahren, Nachbesserungen und Nachweise kann den einer Neuentwicklung übertreffen; zudem ist der Erfolg hier nicht garantiert.

Um einen besseren Überblick über Weg 1 und Weg 3 zu erhalten und den mit ihnen verbundenen Aufwand besser abschätzen zu können, werden die Wege im Folgenden detaillierter vorgestellt.

## 3.a Neuentwicklung in Übereinstimmung mit bestehenden Sicherheitsnormen

Dieser Abschnitt befasst sich mit Weg 1 zur Zertifizierung: “Compliant Development”. Dabei befolgt man von Beginn an die Regeln und Vorschriften in der anzuwendenden harmonisierten Norm. In der Praxis ist dies der am häufigsten gewählte Weg zur Sicherheitszertifizierung. Im Rahmen dieses Dokuments wird das generelle Vorgehen anhand der Basisnorm IEC 61508 beschrieben. In der Praxis muss jedoch zuerst der Anwendungsbereich festgelegt und die zutreffende harmonisierte Norm ausgewählt werden, die im Regelfall jedoch sehr ähnlich zur Basisnorm IEC 61508 ist.

Bevor die eigentliche Entwicklung beginnen kann, müssen einige organisatorische Voraussetzungen erfüllt werden. So müssen diverse Anforderungen an die Verwaltung des Projekts erfüllt sein (IEC 61508-1 6.x), beispielsweise die klare Benennung von Verantwortlichen und ihre Aufgabengebiete (IEC 61508-1 6.2.1) sowie die Einführung von Verfahren zur Überprüfung der Kompetenz besagter Personen (IEC 61508-1 6.2.13). Ein Software-Safety-Lifecycle-Modell, das bestimmte Anforderungen erfüllen muss, muss eingesetzt werden (siehe IEC 61508-3 7.1); dabei gilt es auch, ein für Sicherheit geeignetes Modell zur Softwareentwicklung zu wählen, beispielsweise das V-Modell (siehe IEC 61508-3 7.1.2.4). Ebenso muss die

Dokumentation der Software von Beginn an geplant und in ausreichendem Umfang durchgeführt werden. Hierzu zählt auch festzulegen, welche Informationen relevant sind und daher dokumentiert werden müssen (IEC 61508-1 5.2.1 - 5.2.3). Generell gilt der Leitsatz, dass die Dokumentation einer jeden durchgeführten Phase so ausführlich sein muss, dass die nachfolgenden Phasen und durchzuführenden Aktivitäten effektiv durchgeführt werden können (IEC 61508-1 5.2.1 bis 5.2.3 & 5.2.5).

Sind die organisatorischen Voraussetzungen erfüllt, kann der eigentliche Entwicklungsprozess (unter Beachtung des gewählten Lifecycle-Modells) beginnen. Als ein erster Schritt müssen die Sicherheitsanforderungen an die Software herausgearbeitet und dokumentiert werden. Diese beziehen sich auf alle Anforderungen, die nötig sind, um letztendlich das gewünschte Sicherheitsniveau (hier Safety Integrity Level, SIL) und die gewünschte systematische Sicherheitsintegrität (gemessen durch Systematic Capability, SC) zu erreichen[4]. Ebenso sind Bedarf und die Anforderungen an Sicherheitsfunktionen zu ermitteln. Um sich einen Überblick über den Umfang der zu bedenkenden Anforderungen zu verschaffen, empfiehlt sich der Blick auf IEC 61508-3 7.2.2.5, 7.2.2.8, 7.2.2.10, 7.2.2.12 & 7.2.2.13 - dort finden sich beispielsweise Anforderungen bezüglich der nötigen Sicherheitsfunktionen, der Architektur, der Echtzeitfähigkeit und den Schnittstellen zu nicht sicherheitsrelevanten Funktionalitäten. Insbesondere der erwähnte Absatz 7.2.2.10 führt wichtigen Bedarf an Sicherheitsfunktionen an, die im ersten Moment übersehen werden können, beispielsweise den Bedarf an Funktionen zur Selbstüberwachung der Software.

Als nächster Schritt muss ein Plan für die Validierung der sicherheitsbezogenen Aspekte der Software erstellt werden - hierbei ist die Integration mit Hardware und Systemumgebung zu beachten, da diese verschiedene Aspekte wie Echtzeitanforderungen beeinflussen können. In IEC 61508-3 befasst sich Abschnitt 7.3 mit der Planung. Generelles Ziel dieser Phase ist es einen Plan zu erstellen, wie die Einhaltung der Sicherheitsanforderungen nachgewiesen werden kann. Die Planung umfasst beispielsweise die Festlegung von Verfahren, die zur Evaluation (IEC 61508-3 7.3.2.2 e)) eingesetzt werden sollen, und das Festlegen von Kriterien, wann die Validierung erfolgreich oder fehlgeschlagen ist (IEC 61508-3 7.3.2.2 h) & 7.3.2.5).

Ist der Plan für die Validierung der Software festgelegt, folgt der Grobentwurf der Software. Anforderungen an das Entwurfsverfahren und den Entwurf werden in IEC 61508-3 7.4.2 festgelegt und stimmen in Teilen mit der allgemeinen Auffassung guten Software-Designs überein (Einfachheit, Modularität, Testbarkeit, Modifizierbarkeit, …). Insbesondere ist es wichtig, Sicherheitsfunktionen von nicht sicherheitsrelevanten Funktionen getrennt zu halten, sodass sie sich nicht gegenseitig beeinflussen können - wird dem nicht Folge geleistet, ist auch die nicht sicherheitsrelevante Funktion wie eine Sicherheitsfunktion zu behandeln (IEC 61508-3 7.4.2.8). Generell gilt der Grundsatz: Wo Funktionen verschiedenen Sicherheitslevels oder Sicherheitsfunktionen und nicht sicherheitsrelevante Funktionen gemeinsam vorhanden sind, muss nachgewiesen werden, dass sich die Funktionen nicht gegenseitig beeinflussen und unabhängig voneinander sind, ansonsten ist das jeweils höchste vorhandene Sicherheitslevel unter den Sicherheitsfunktionen für alle Funktionen anzuwenden (IEC 61508-3 7.4.2.8 & 7.4.2.9). Daher ist es äußerst wünschenswert, eine entsprechende Modularität und Unabhängigkeit der

[4] Das SIL bezieht sich dabei auf die Wahrscheinlichkeit des Auftretens zufälliger, schwerwiegender Fehler, während sich die SC mit der Wahrscheinlichkeit des Auftretens systematischer Fehler befasst und somit ein Maß für die systematische Sicherheitsintegrität ist. In der Praxis muss SC >= SIL gelten (beachte hierzu auch IEC 61508-3 7.4.2.10), ansonsten wird das SIL auf die SC abgewertet (beispielsweise SIL 3 bei SC 2 wird zu SIL 2). Einen tieferen und verständlichen Einblick in die SC gibt Gerry Creech: *IEC 61508 Systematic Capability*, 2014, verfügbar unter https://journals.sagepub.com/doi/pdf/10.1177/0020294014528895

Komponenten durch das angewendete Design zu erzielen und nachzuweisen. Ebenso erwähnenswert ist es, dass auch eingesetzte Daten im Entwurfsprozess bedacht werden müssen (IEC 61508-3, 7.4.2.14), beispielsweise bei Konfigurationsdaten, wenn diese die Funktionalität der Sicherheitsfunktionen stark beeinflussen. Weitere Anforderungen, direkt auf den Entwurf der Softwarearchitektur bezogen, können darüber hinaus IEC 61508-3, 7.4.3 entnommen werden. Zu einem späteren Zeitpunkt und vor Beginn der Implementierung ist der gesamte Entwurf noch weiter zu verfeinern (im Gegensatz zum iterativen Vorgehen bei agilen Methoden), indem die Elemente des vorangeganenen Grobentwurfs weiter aufgeteilt und modularisiert werden. Während sich der Grobentwurf eher auf einer abstrakten Ebene abspielt, werden im Feinentwurf beispielsweise die individuellen Softwaremodule entworfen, aus denen sich die abstrakten, funktionalen Elemente des Grobentwurfs zusammensetzen. Besonders erwähnenswert ist, dass im Rahmen des Feinentwurfs direkt die Maßnahmen zur Verifikation der Softwaremodule (IEC 61508-3 7.4.5.4) und Tests im Rahmen der Integration (IEC 61508-3 7.4.5.5, 7.4.8.1 & 7.4.8.2) festgelegt werden müssen. Weitere Informationen zum Feinentwurf können in IEC 61508-3 Absatz 7.4.5 nachgeschlagen werden.

Auf den Entwurf der Software folgt die Auswahl geeigneter Werkzeuge und Programmiersprachen, die eingesetzt werden dürfen (IEC 61508-3 7.4.4.). Eingesetzte Werkzeuge sind prinzipiell in zwei Kategorien aufzuteilen: Offline-Werkzeuge, die während der Entwicklung zum Einsatz kommen, und Online-Werkzeuge, die während der Laufzeit Einfluss auf das System haben. Der Einsatz von Online-Werkzeugen ist eher kritisch, da diese als Teil der Software gesehen werden (IEC 61508-3 7.4.4.1). Dies hat zur Folge, dass entweder nachgewiesen werden muss, dass das Online-Tool keinen Einfluss auf Sicherheitsfunktionen hat, oder dass Anforderungen wie bei Sicherheitsfunktionen beachtet werden müssen. Anders verhält es sich bei der Verwendung von Offline-Werkzeugen, deren Einsatz im Rahmen von IEC 61508-3 explizit empfohlen wird (Absatz 7.4.4.2), falls sie einen Mehrwert bieten (Absatz 7.4.4.3), beispielsweise zur Vermeidung von Programmierfehlern oder zur Unterstützung bei Tests der Software. Offline-Werkzeuge lassen sich weiter in Kategorien einteilen: T1 - T3[5], welche in IEC 61508-4 3.2.11 spezifiziert werden. Vor Allem der Einsatz mächtigerer Werkzeuge der Kategorie T2 & T3 ist weiter reglementiert, beispielsweise muss das Verhalten der Offline-Werkzeuge der Kategorie T2-T3 klar dokumentiert sein (IEC 61508-3 7.4.4.4), für Werkzeuge der Klasse T3 muss darüber hinaus nachgewiesen sein, dass das tatsächliche Verhalten mit der Dokumentation übereinstimmt (IEC 61508-3 7.4.4.4). Bezüglich der Programmiersprachen finden sich Anforderungen beispielsweise in IEC 61508-3 7.4.4.10 - 7.4.4.13. Wichtige erwähnte Aspekte für die Wahl einer Programmiersprache sind beispielsweise, dass geeignete Übersetzer (dies schließt Compiler mit ein) vorhanden sind und dass ein geeigneter Programmierstil für die Programmiersprache vorhanden ist, wobei das Verbot nicht sicherer Funktionen der Programmiersprache einen Kernpunkt darstellt (eine Liste der Anforderungen findet sich in Absatz 7.4.4.13).

Für die eigentliche Implementierung, die auf die Auswahl der Werkzeuge und der Programmiersprache(n) folgt, kommen mit IEC 61508-3 Absatz 7.4.6 wenige neue Vorschriften hinzu. Zuvor festgelegte Anforderungen, beispielsweise aus dem Design oder dem gewählten Programmierstil, müssen selbstverständlich eingehalten werden, ansonsten gilt es wie bereits beim Entwurf der Architektur erwünschte Eigenschaften wie Verständlichkeit und Testbarkeit zu erzielen. Neu kommt jedoch in Absatz 7.4.6.1 hinzu, dass der geschriebene Code einer Review unterzogen werden muss.

[5] Beispiele für die Level T1-T3 nach IEC 61508-4 3.2.11 : T1 umfasst beispielsweise “einfache” Werkzeuge wie reine Texteditoren ohne weitere Funktionalitäten, T2 umfasst beispielsweise Analysewerkzeuge, wie solche zur statischen Codeanalyse und T3 beispielsweise Werkzeuge mit Einfluss auf den Code, wie optimierende Compiler.

Anschließend ist die implementierte Software zu testen; hierfür sind die im Rahmen des Feinentwurfs spezifizierten Tests zur Verfifikation der einzelnen Softwaremodule (IEC 61508-3 7.4.7) sowie die Integrationstests (IEC 61508-3 7.4.8 & 7.4.9) durchzuführen, wobei bei letzteren die Integration der einzelnen Module zu einer finalen Software und die Integration von Software und Hardware zu unterscheiden ist. Prinzipiell soll hierbei nachgewiesen werden, dass die einzelnen Softwaremodule korrekt funktionieren, dass das Zusammenspiel der einzelnen Softwaremodule im finalen Softwareprodukt korrekt funktioniert und dass das Zusammenspiel von Hardware und Software wie gewünscht abläuft. Alle Testvorgänge haben hierbei gemeinsam, dass ihre Ergebnisse dokumentiert werden müssen (IEC 61508-3 7.4.7.3, 7.4.8.4, 7.5.2.7 & 7.5.2.8). Zu den empfohlenen Verifikationsmaßnahmen zählt beispielsweise das Erreichen einer Testabdeckung (“Code Coverage”) von 100%, wobei jedoch in Sonderfällen Ausnahmen von den 100% möglich sind (wenn 100% nicht erreicht werden können) und sich die Art der nötigen Testabdeckung je nach SIL-Level unterscheidet. Für SIL 1 müssen lediglich alle Funktionen zumindest ein Mal aufgerufen werden (“entry points structural test coverage”), ein Erfüllen umfassender Testabdeckungskriterien ist nicht verpflichtend, sondern lediglich empfohlen (Niveau “Recommended”). Bei SIL 4 müssen hingehen eine Vielzahl von Testabdeckungskriterien erfüllt werden, bis hin zu beispielsweise der komplexen MC/DC[6] Testabdeckung. Für weitere Details siehe IEC 61508-3 Tabelle B.2.

Neben der Pflicht zur Dokumentation ist auch die Pflicht zur Einflussanalyse (engl. “Impact Analysis”) besonders hervorzuheben, die im Rahmen der Integrationstests gilt. Sie besagt, dass bei vorgenommenen Änderungen an der Software alle Teile der Software identifiziert werden müssen, die von dieser Änderung betroffen sind (beispielsweise direkt, da tatsächlich etwas geändert wurde, oder indirekt, da sich das Zusammenspiel der Komponenten durch die Änderung verändert hat). Im Anschluss müssen die betroffenen Teile erneut verifiziert werden, eventuell müssen auch Änderungen am Entwurf vorgenommen werden (IEC 61508-3 7.4.8.5 & 7.5.2.8). Das Prinzip der Einflussanalyse sollte verinnerlicht werden, da sie auch beim fertigen Produkt durchgeführt werden muss, sollten Änderungen nötig sein (IEC 61508-3 7.8.2.3).

Mit den Tests ist der eigentliche Entwicklungsprozess und somit auch das vorliegende Kapitel III.a abgeschlossen. Zum Abschluss sollen jedoch noch einige wichtige Teile der Norm IEC 61508-3 aufgelistet werden, die ebenfalls beachtet werden müssen:

- Für den späteren Betrieb und die Modifikation existierender Software existieren auch Vorschriften. Diese sind in Absatz 7.6 und 7.8 zu finden.
- Absatz 7.7 befasst sich mit der eigentlichen sicherheitsbezogenen Validierung der Software im finalen Produkt.
- Absatz 7.9 befasst sich mit der Verifikation der einzelnen Phasen des Lifecycles bei der Softwareentwicklung. Er liefert einen geeigneten Einstiegspunkt, um sich mit dem allgemeinen Konzept der Verifikation in IEC 61508-3 vertraut zu machen.
- Die finale Abschätzung der funktionalen Sicherheit wird in Absatz 8 geregelt.
- Die Norm IEC 61508-3 verfügt über einige Anhänge. Dort sind empfohlene Maßnahmen für die einzelnen Aktivitäten bezüglich des jeweiligen angestrebten SIL-Levels zu finden.

[6] Definition und Erklärung siehe https://www.rapitasystems.com/mcdc-coverage

## 3.b Nachträgliche Sicherheitszertifizierung existierender Softwareprodukte

Dieser Abschnitt befasst sich mit Weg 3 zur Zertifizierung: “Assessment of non-compliant Development”. Er kommt zum Einsatz, falls existierende Software nicht direkt nach den Vorschriften von IEC 61508 entwickelt wurde, aber dennoch Sicherheitsfunktionen übernehmen soll. Die nötigen Anforderungen werden ausführlich in Abschnitt 7.4.2.13 von IEC 61508-3 beschrieben, darüber hinaus ist ebenfalls Abschnitt 7.4.2.13 b) zu beachten.
Strebt man die Sicherheitszertifizierung über Weg 3 an, sind primär 3 verschiedene Aufgaben zu erledigen:

- Dokumentation, um die Übereinstimmung mit der Sicherheitsnorm zu erreichen und/oder nachzuweisen, wo dies nötig ist
- Erbringen von Nachweisen, dass Sicherheitsanforderungen eingehalten werden
- Nachbessern, wo dies nötig ist

Die zu erbringende Dokumentation im Falle von Weg 3 umfasst Punkte, die auch bei Weg 1 dokumentiert werden müssen. Zuerst gilt es, die Sicherheitsanforderungen an die Software, abhängig von ihrem konkreten angestrebten Einsatz, zu formulieren. Hierbei muss ebenfalls auf das funktionale Verhalten und das Sicherheitsverhalten der Software eingegangen werden. Anschließend gilt es schlüssig zu dokumentieren, wie das Softwaredesign die Sicherheitsanforderungen erfüllt. Zu beachtende Regeln und empfohlene Maßnahmen hierzu können der Norm entnommen werden. Besonders empfehlenswert ist die Partitionierung der Software (falls noch nicht geschehen), wie sie auch in der Norm mehrfach empfohlen wird, insbesondere in sicherheitskritische und nicht sicherheitskritische Teile. Diese Partitionierung erleichtert sowohl die Dokumentation als auch einige der später nötigen Nachweise.

Nachweise müssen im Rahmen von Weg 3 erbracht werden, um aufzuzeigen, dass sicherheitsrelevante Aspekte bedacht und eingehalten wurden, auch wenn das in der Norm vorgeschriebene Vorgehen nicht exakt angewendet wurde. Ein besonders wichtiger Teil dieser Nachweise bezieht sich darauf, dass die Software benötigte Sicherheitseigenschaften erfüllt. Eine Liste der einzuhaltenden Vorschriften kann IEC 61508-3, Abschnitt 7.4.2.13 b) entnommen werden. Diese beziehen sich beispielsweise auf die Architektur der Software, die Testbarkeit des Codes und Validierung der Sicherheit. Es ist insbesondere zu beachten, dass die Nachweise nicht die Software allein, sondern auch das Zusammenspiel mit der Hardware umfassen (beispielsweise relevant für das Einhalten von Echtzeitanforderungen). Ebenso wichtig ist der Nachweis eines systematischen Verifikations- und Validierungsverfahrens, welches zur Auswertung der Software angewendet wurde. Im Rahmen des Verfahrens müssen sowohl Code als auch die Architektur betrachtet worden sein. Essentiell ist hierbei, dass die Ergebnisse von Tests und Reviews schriftlich festgehalten wurden - ein Vorgehen, das zeitaufwendig ist und bei den meisten kleineren Unternehmen einen der ersten Punkte darstellen dürfte, an dem nachgebessert werden muss. Weitere Nachweise umfassen beispielsweise, dass nicht-sicherheitsbezogene Funktionen die sicherheitsbezogenen Funktionen nicht behindern bzw. negativ beeinflussen und dass alle realistischen Fehlerfälle identifiziert und angemessene Gegenmaßnahmen ergriffen wurden. Eine Liste mit den erwähnten und weiteren zu erbringenden Nachweisen kann Abschnitt 7.4.2.13 von IEC 61508-3 entnommen wurden.

Sollten beim Erstellen/Erweitern der Dokumentation und dem Führen der Nachweise Lücken aufgefallen sein, gilt es, diese in der Software entsprechend nachzubessern. Wurde Sicherheit beim ursprünglichen Entwurf des Systems nicht beachtet, kann dies einen erheblichen Zeitaufwand bedeuten. Insbesondere ist zu beachten, dass Anpassungen an der Software bedeuten, dass auch Dokumentation und bereits erzielte Nachweise neu erbracht werden müssen.

Für nicht neu erbrachte Nachweise sollte gezeigt werden, dass sie nicht von den Änderungen an der Software betroffen sind, da die Analyse des Einflusses von Änderungen an der Software ein essentieller Bestandteil bei der Entwicklung sicherer Software ist (siehe u.a. IEC 61508-3, Abschnitt 6.2.3 d), 7.5.2.6 und insb. 7.5.2.8). Es empfiehlt sich folglich, offensichtlich notwendige Änderungen und Nachbesserungen an Stellen, die das Führen von Nachweisen potentiell behindern könnten, vorab vorzunehmen, um Mehrarbeit zu verhindern.

Prinzipiell empfiehlt sich die Wahl von Weg 3 primär in zwei Fällen:

1. Eine Neuentwicklung des Produkts unter Beachtung des in den Normen vorgeschriebenen Vorgehens ist nicht möglich oder mit unvertretbarem Aufwand verbunden. Ein Beispiel für ein solches Produkt stellt beispielsweise das Open-Source Betriebssystem Linux dar, für welches die Organisation OSADL im Projekt SIL2LinuxMP versucht hat, das Sicherheitsniveau SIL 2 über das Beschreiten von Weg 3 für einen konkreten Anwendungsfall zu erreichen.
2. Bei der Entwicklung wurden bereits allerhöchste Qualitätsstandards angelegt. Es fand eine ausführliche Planung des Systems unter Beachtung von Risiken statt, eine sinnvolle, partitionierende Softwarearchitektur wurde etabliert, Maßnahmen zur Qualitätssicherung, wie regelmäßige Überprüfungen des Codes und Tests, werden bereits angewendet, und verwaltungstechnische Punkte, wie das Vorhandensein eines Änderungsmanagements und einer detaillierten Dokumentation, wurden beachtet. Mit Blick auf die anzuwendenden Sicherheitsnormen sind wenig Änderungen an Dokumentation und Software zu erwarten, die zu erwartende Hauptarbeit entfällt auf das Erbringen der Nachweise.

## 4. Schritte, die gegebenenfalls eine nachträgliche Zertifizierung erleichtern können

Vor allem bei KMU taucht die Frage auf, ob bei der Softwareentwicklung eines neuen Produkts bereits nicht allzu aufwendige Schritte unternommen werden können, die zu einem späteren Zeitpunkt eine Zertifikation über Weg 3 erleichtern können. Hintergrund dieser Frage ist, dass bei KMU oftmals initial nicht die Ressourcen zur Verfügung stehen, direkt eine Entwicklung über Weg 1 anzustreben, sondern zuerst ein kommerziell nutzbares Produkt entstehen soll, dessen Einsatzspektrum gegebenenfalls zu einem späteren Zeitpunkt auf sicherheitskritische Szenarien ausgeweitet werden soll. Das vorliegende Kapitel spiegelt lediglich Ideen der Autoren basierend aus dem Vorgehen von Weg 1 und den Anforderungen von Weg 3 wieder - ein Mehrwert bei einem späteren Beschreiten von Weg 3 kann daher nicht garantiert werden.

Eine besondere Herausforderung beim Festlegen sinnvoller Schritte ergibt sich daraus, dass der spätere, sicherheitskritische Anwendungsfall unbekannt ist, denn sowohl die konkret anzuwendenden Normen als auch die Bestimmung der sicherheitskritischen Funktionen des Systems hängen vom konkreten Anwendungsfall ab. Man betrachte beispielsweise ein Modul aus Software und Sensorik, das zur Detektion von Fahrzeugen und der Bestimmung des Abstands zwischen Fahrzeugen eingesetzt wird. Eine solche Software könnte beispielsweise als Erweiterung von Anlagen zur Verkehrsüberwachung zum Einsatz kommen, um neben Verstößen gegen das geltende Tempolimit auch Verstöße gegen den Mindestabstand ahnden zu können - zumindest für die physische Sicherheit von Menschen existieren hier keine Sicherheitsbedenken. Ebenso könnte das Modul jedoch bei einem autonomen Fahrzeug eingesetzt werden, um den Abstand zu anderen Verkehrsteilnehmern zu überwachen - ein äußerst sicherheitskritischer Einsatz des Moduls. Auch Szenarien, bei denen nur Teile des Moduls (z.B. die Detektion von Fahrzeugen ohne die Abstandsbestimmung) sicherheitskritisch sind, sind denkbar. Es wird daher versucht, möglichst allgemeingültige Schritte zu finden, die den Grundprinzipien sicherer Softwareentwicklung entsprechen, welche auch ohne Kenntnis des konkreten Anwendungsfalls Anwendung finden können:

1. **Nutzung einer modularen Softwarearchitektur mit so geringen Abhängigkeiten zwischen den Modulen wie möglich.** Wie bereits in Kapitel III.a aufgezeigt, wird eine modulare Softwarearchitektur für die Entwicklung sicherheitszertifizierter Software dringend empfohlen. So muss auf Komponenten, die nicht unabhängig sind, das jeweils höchste unter ihnen vorhandene Sicherheitslevel angewendet werden - daher ist die Modularisierung in viele, möglichst unabhängige Komponenten wünschenswert, vor allem wenn man noch nicht weiß, welche Komponenten später in einem konkreten Anwendungsfall als sicherheitskritisch zu betrachten sind. Eine angemessene Modularisierung der Software sollte darüber hinaus mit vertretbarem Aufwand umzusetzen sein und erleichtert generell spätere Anpassungen im Code.
2. **Festlegen eines einheitlichen Vorgehens bei der Dokumentation und gewissenhafte Durchführung der Dokumentation über alle Phasen der Entwicklung.** Ein Kernaspekt sicherer Software ist die angemessene Dokumentation von u.a. Architektur, Entwicklung & Tests. Es empfiehlt sich daher von Beginn an festzulegen, welche Aspekte relevant sind und dokumentiert werden sollen und diese Dokumentation gewissenhaft durchzuführen. Der Aufwand für eine angemessene Dokumentation kann durchaus erhöht sein, bietet jedoch auch Zeitersparnisse in späteren Projektphasen und bei Einbezug neuer Mitarbeiter.
3. **Festlegen von Tests parallel zur Entwicklung.** Im Rahmen der Entwicklung sicherer Software gilt es, für jede Phase der Softwareentwicklung direkt passende Tests festzulegen, um den Erfolg der jeweiligen Phase evaluieren zu können. Abhängig davon,

wie ansonsten getestet werden würde, geht dieser Schritt mit variablem Zeitaufwand einher, wobei sich der erhöhte Testaufwand in einem Produkt mit weniger Fehlern, folglich einem höherwertigen Produkt, niederschlagen sollte. Hinweis: Parallelität bedeutet nicht Personalunion! Auch wenn es bequemer erscheinen mag, die Tests von der gleichen Person festlegen zu lassen, die die Funktion implementiert hat, da diese “sich ja damit auskennt” - wer sich selbst testet, testet schlecht!

4. **Festlegen einheitlicher (Entwicklungs-)Werkzeuge & eines Programmierstils sowie Reduktion auf Werkzeuge, die tatsächlich einen Mehrwert bringen.** Da der Einsatz von Werkzeugen im Rahmen der sicherheitszertifizierten Entwicklung gerechtfertigt werden muss, sollte hier Homogenität geschaffen werden und ein einheitlicher Satz an Werkzeugen für alle Entwickler genutzt werden. Ebenso sollte ein gemeinsamer, geeigneter Programmierstil gewählt werden, der im Rahmen der sicherheitszertifizierten Entwicklung sowieso genutzt werden muss. Dies erspart zu einem späteren Zeitpunkt den Aufwand der Rechtfertigung für weitere Werkzeuge sowie Codeanpassungen. Der Zeitaufwand hierfür sollte sich gering gestalten, während durch Einheitlichkeit im besten Fall auch eine höhere Codequalität erzielt werden kann. Auf Online-Werkzeuge, sofern nicht zwingend nötig, sollte möglichst verzichtet werden, da der Rechtfertigungsaufwand für deren Einsatz höher ist.

Die vorgeschlagenen Punkte haben gemeinsam, dass durch sie theoretisch der Aufwand für Weg 3 verringert werden sollte. Selbst falls dies nicht der Fall ist, sind die Punkte so ausgewählt, dass sie voraussichtlich mit vertretbarem Aufwand realisiert werden können und einen positiven Einfluss auf das finale Softwareprodukt haben sollten. Prinzipiell können mögliche Maßnahmen selbst aus dem Vorgehen aus Weg 1 und den Anforderungen aus Weg 3 abgeleitet werden.

# 5. Umgang mit Compilern und Open-Source-Software bei sicherheitszertifizierten Softwareprodukten

Im Rahmen dieses Abschnitts soll die Frage beantwortet werden, wie im Kontext von sicherheitskritischer Software mit Compilern und Open-Source-Software umgegangen werden muss und welche Möglichkeiten für deren Einsatz bestehen.

In der Norm IEC 61508-3 fallen Compiler in die Gruppe der Softwarewerkzeuge, deren Einsatz in Absatz 7.4.4 geregelt wird. Der Einsatz eines konkreten Compilers muss daher, wie bei anderen Softwarewerkzeugen, gerechtfertigt werden. Weiterhin muss er einer ausgiebigen, zu dokumentierenden Validierung unterzogen werden. Empfehlungen für Programmiersprachen und Compiler werden darüber hinaus in Tabelle A.3 der Norm aufgelistet - der Einsatz zertifizierter Compiler (als Teil der übergeordneten Gruppe der “Translator”) wird hierbei über alle SIL-Level hinweg empfohlen. Besonders häufig tritt im Zusammenhang mit diesen zertifizierten Compilern die Frage auf, ob solche im Hinblick auf neueste Programmiersprachenstandards (z.B. neue C++ Versionen) existieren. Diese Frage kann, eventuelle Sonderregelungen in fachbereichsspezifischen Normen ausgenommen, mit “Nein” beantwortet werden: Tabelle A.3 Punkt 4b von IEC 61508-3 sieht für alle SIL-Level vor, dass unterstützend zur Validierung ausreichend positive Nutzungsdaten für Compiler vorliegen[7] - beispielsweise von einer hohen Anzahl von nicht sicherheitskritischen Projekten, bei denen keine Probleme durch die Nutzung des spezifischen Compilers aufgetreten sind. Compiler für neue Programmiersprachenstandards verfügen noch nicht über ein ausreichendes Portfolio an positiven Nutzungserfahrungen und eignen sich somit nicht für die Entwicklung sicherheitszertifizierter Softwareprodukte.

Bezüglich Informationen zum Vorgehen bei der Zertifizierung von Werkzeugen, insbesondere Compilern, kann IEC 61508-7 Abschnitt C.4.3 herangezogen werden. Darüber hinaus findet sich in IEC 61508-3, welche Eigenschaften ein sicherheitszertifizierter Compiler erfüllen muss, damit er im Entwicklungsprozess für ein sicherheitszertifiziertes Softwareprodukt eingesetzt werden darf. Für IEC 61508-3 finden sich die Anforderungen in Annex C, Tabelle C.3. Demnach muss die Verwendung und Funktionalität eindeutig sein, ebenso muss die Erzeugung von Ausgaben durch den Compiler korrekt und wiederholbar sein. Für beides wird ein Rigour-Level der Stufe 2 festgelegt (siehe Annex C, C 1.1 & C1.2), was bedeutet, dass der Nachweis dieser Eigenschaften nicht über formale Beweise erfolgen muss, sondern dass lediglich nachgewiesen werden muss, dass die benötigten Eigenschaften mit höchster Wahrscheinlichkeit[8] erreicht wurden. Ein geeignetes Nachweisverfahren können beispielsweise ausgiebige Tests mit sehr hoher oder vollständiger Testabdeckung (“Code Coverage”) sein - hier können unentdeckte Fehler noch vorhanden sein, die Wahrscheinlichkeit ist jedoch sehr gering. Detektierte Fehler müssen behoben oder ausreichend dokumentiert und gerechtfertigt werden.

[7] Konkrete Zahlen sind hier schwer zu finden. Das Sicherheitsteam von Hitex UK spricht beispielsweise davon, dass der Compiler unter anderem von einer großen Anzahl an Nutzern über einen längeren Zeitraum in verschiedenen Projekten eingesetzt worden sein muss (“[...] by showing that it has been used in different products by a large user base over an extended period [...]”, siehe https://hitex.co.uk/fileadmin/uk-files/pdf/IEC61508_proofed.pdf). Viele Quellen geben lediglich die Anforderungen aus IEC-61508-7 Abschnitt C.4.4 direkt oder indirekt wieder: Es sollte ein Compiler verwendet werden, der über viele Projekte kein Fehlverhalten gezeigt hat; ebenso sollten Compiler ohne ausreichend Praxiserfahrung oder mit schweren bekannten Mängeln gemieden werden. (Original: “A translator is used, where there has been no evidence of improper performance over many prior projects. Translators without operating experience or with any serious known faults should be avoided unless there is some other assurance of correct performance.”)

[8] Eine allgemeingültige Definition von „höchster Wahrscheinlichkeit“ wurde im Rahmen der Recherche zu dieser Arbeit leider nicht gefunden, der Begriff scheint nicht eindeutig definiert zu sein.

Für die Auswahl und den Einsatz eines Compilers in der Praxis gibt es im Hinblick auf kleinere Unternehmen zwei praktikable Wege: Die Verwendung eines zertifizierten Compilers (mit Tool Qualification Kit), der entsprechend der Vorschriften seines Safety-Manuals eingesetzt wird und viel der benötigten Dokumentation und Rechtfertigung für den Einsatz bereits mitliefert, oder die Zusammenarbeit mit einem Dienstleister (z.B. Validas, Heicon, Tasking), um die Qualifizierung des Compilers mit Erstellung eines eigenen Qualification KITs[9] und Safety-Manuals durchzuführen. Eine Auswahl an zertifizierten Compilern findet sich in Anhang A. Die Anforderungen an Compiler und den erzeugten Objektcode weichen jedoch in verschiedenen Normen ab. So sieht Norm DO-178B (Luftfahrt) beispielsweise vor, dass Compiler nicht qualifiziert werden müssen, wenn der erzeugte Objektcode durch Tests verifiziert wird (Allgemein: Wenn die Ausgabe eines Tools verifiziert wird, muss das Tool nicht qualifiziert werden)[10]. Zur Verifikation könnte der Objektcode beispielsweise unter anderem Tests sowie Reviews unterzogen werden. Darüber hinaus räumt die neueste Version der Norm (DO 178C) auch explizit die Möglichkeit ein, die notwendige Testabdeckung auf dem Objektcode zu erzielen ("Structural coverage analysis may be performed on the Source Code, Object Code or Executable Object Code.", DO 178C, 6.4.4.2b), wobei für das höchste Sicherheitslevel (Level A) Einschränkungen/Erweiterungen gelten.

Neben dem Einsatz von Compilern gilt es im Rahmen dieses Abschnitts auch die Möglichkeiten zum Einsatz von Open-Source-Software zu klären. Betrachtet wird der gängige Fall, dass die Open-Source-Software über keine Sicherheitszertifizierung verfügt - ansonsten wäre sie so zu handhaben wie andere sicherheitszertifizierte Softwarebausteine. Darüber hinaus wird davon ausgegangen, dass die Open-Source-Software nicht selbst entwickelt wurde. Unter diesen Annahmen ist lediglich die Anwendung von Weg 3, wie in Abschnitt III.b beschrieben, praktikabel. Die besondere Herausforderung bei Open-Source-Software ist, dass oftmals wenig über den Entwicklungsprozess bekannt bzw. dokumentiert ist und sich dieser in der Regel vom erwünschten Vorgehen gemäß IEC 61508-3 unterscheidet. Dennoch ist es möglich, eine Sicherheitszertifizierung für eingesetzte Open-Source-Software im Rahmen des eigenen Projekts zu erzielen, wenngleich dieses Vorgehen schwieriger und zeitaufwendiger ist als bei eigener Software. Um einen Eindruck über das mögliche Vorgehen und die Herausforderungen zu erhalten, werden die Erkenntnisse aus dem SIL2LinuxMP-Projekt (Start im Jahr 2013) herangezogen. Bei diesem Projekt wurde versucht, eine Sicherheitszertifizierung für ein Open-Source Linux Betriebssystem im Kontext einer konkreten Anwendung zu erzielen (Anm.: Nur für die tatsächlich benötigten Teile des Systems im Kontext der Anwendung). Erkenntnisse der ersten 3 Jahre des Projekts wurden von Platschek et al. veröffentlicht[11]; diese Veröffentlichung dient als Grundlage für den vorliegenden Abschnitt.

Wie bereits zuvor erwähnt besteht die große Herausforderung bei der nachträglichen Zertifizierung von Open-Source-Software darin, dass nicht die üblichen Nachweise über den Entwicklungsprozess geführt werden können. SIL2LinuxMP geht dieses Problem unter anderem dadurch an, dass das empfohlene V-Modell für den gesamten Entwicklungsprozess angepasst wird. Anforderungsanalyse und grober Architekturentwurf werden gemäß Weg 1 (siehe Abschnitt III.a) entwickelt, der nachfolgende Teil des V-Modells wird durch einen Selektionsprozess für Open-Source-Komponenten ersetzt. Dieses neue Vorgehen bedingt wiederum, dass die vorgeschlagenen Methoden zur Zertifizierung angepasst beziehungsweise durch neue,

[9] Ein Qualification Kit ist eine Sammlung an Dokumenten, die für die Qualifizierung eines Werkzeugs wie eines Compilers notwendig sind/diese unterstützen. Siehe beispielsweise https://www.tasking.com/products/compiler-qualification-kit

[10] Siehe Hilderman et al.: *Avionics Certification: A Complete Guide to DO-178 (software), DO-254 (hardware)*, 2011, S. 216

[11] Platschek et al.: *Certifying Linux: Lessons Learned in Three Years of SIL2LinuxMP*, 2018, siehe https://www.bmw-carit.com/downloads/publications/EWC2018_Certifying-Linux-Lessons-Learned.pdf

vergleichbare Methoden ersetzt werden. Als Beispiel für die Notwendigkeit führen Platschek et al. den Fall an, dass die Spezifikation für ein Softwareelement nachträglich erstellt wird. Ziel dieser Spezifikation ist es jedoch dafür zu sorgen, dass im Softwareprodukt keine intrinsischen Spezifikationsfehler auftauchen - ein Ziel, zu dem eine nachträgliche Spezifikation keinen Beitrag leisten kann. Folglich muss dieses Ziel über eine alternative Methode erreicht werden, die dieses Ziel gleichwertig oder zuverlässiger erreicht. Um dieses Problem systematisch anzugehen, wurde im Rahmen von SIL2LinuxMP ein allgemeiner Prozess entworfen, wie bei der Argumentation bezüglich neuer/angepasster Methoden vorzugehen ist und wie gezeigt werden kann, dass die durch IEC 61508 gewünschten Ziele durch die neuen Methoden erreicht werden. Details können der Arbeit von Platschek et al. entnommen werden. Die Arbeit von Platschek et al. zu SIL2LinuxMP enthält darüber hinaus noch weitere Vorgehensweisen, für die wir auf die entsprechende Arbeit verweisen. Besonders erwähnenswert ist in jedem Fall noch das Prinzip der Minimierung: Sollte Open-Source-Software in einem spezifischen Projekt eingesetzt werden, empfiehlt es sich, den Code auf das absolut nötige Minimum zu reduzieren, um somit den Aufwand für Zertifizierungstätigkeiten beträchtlich zu minimieren. SIL2LinuxMP geht dabei sogar so weit, dass ein automatisiertes Werkzeug für diese Codeminimierung entworfen wurde.

Die genannten Beispiele dienen lediglich zur Orientierung. Die vorgestellten Vorgehensweisen zeigen klar, dass Open-Source-Software nicht leichtfertig eingesetzt werden kann und ein hoher Aufwand nötig ist, um sie im Rahmen eines Projekts einzusetzen. Daher sollte eingehend geprüft werden, ob der Einsatz einer bestimmten Open-Source-Software in Hinblick auf den zu erwartenden Zertifizierungsaufwand gerechtfertigt ist, oder ob genutzte Funktionalitäten, besonders falls sie nicht umfangreich sind, mit geringerem Aufwand selbst implementiert werden können.

# 6. Schlusswort

Im Rahmen dieses Dokuments wurde eine Einführung in die Welt der sicherheitszertifizierten Softwareentwicklung gegeben, um kleinen Unternehmen einen Einblick zu gewähren, womit bei der Entwicklung eines sicherheitszertifizierten Softwareprodukts zu rechnen ist. Die Fülle an Vorschriften und notwendigen Maßnahmen wirkt auf den ersten Blick überwältigend: Organisatorische Maßnahmen, besondere Anforderungen an den Entwicklungsprozess, die Überprüfung aller Prozessschritte und des Softwareprodukts, die Anforderungen an die eingesetzten Werkzeuge und vieles mehr. Die vorgestellten Maßnahmen und Vorschriften sollen vermitteln, dass die Entwicklung von sicherheitszertifizierter Software nicht etwas ist, was man “schnell nebenbei” erledigen kann, bzw. dass man nicht “schnell nachträglich” ein existierendes Softwareprodukt zertifizieren kann. Beide Pfade können beschritten werden, benötigen jedoch volles Engagement und sind mit hohem Aufwand verbunden.
Da der Einstieg in die sicherheitszertifizierte Entwicklung nicht leicht ist empfiehlt es sich, die Ziele zu Beginn nicht zu hoch zu stecken: Ohne Erfahrung wird die Entwicklung eines SIL 3 bzw. 4 oder ASIL D Produkts ohne fachmännische Hilfe kaum gelingen. Mit Projekten, die niedrigere SIL-Level anstreben oder der Entwicklung von Produkten für Anwendungsbereiche, die niedrige Sicherheitsanforderungen haben, lassen sich gezielt Erfahrungen mit der Arbeit mit Normen sammeln. Auch gemeinsame Projekte mit Partnern, die Erfahrung auf dem Gebiet der sicherheitszertifizierten Entwicklung haben, können einen guten Einstieg bieten. Trotz der Einstiegshürde sollte man sich nicht vom Ziel der sicherheitszertifizierten Entwicklung abbringen lassen: Mit dem entsprechenden Engagement und den richtigen Ambitionen gelingt auch der Einstieg in die Welt der sicherheitszertifizierten Entwicklung.

## Anhang A: Tabelle sicherheitszertifizierte Compiler

| Compiler | Zertifikate | Version |
|---|---|---|
| Arm Compiler Version 6 | IEC 61508 (SIL 1-4)<br>EN 50128 (SIL 1-4)<br>ISO 26262 (ASIL A-D)<br>IEC 62304 | C++14 |
| Arm Compiler Version 5 | IEC 61508 (SIL 1-3)<br>ISO 26262 (ASIL A-D) | C++03 |
| Green Hill Compiler | IEC 61508 (SIL 1-4)<br>EN 50128 (SIL 1-4)<br>ISO 26262 (ASIL A-D) | C++03<br>C++11<br>C++14 |
| Wind River Diab Compiler | IEC 61508 (SIL 1-4)<br>ISO 26262 (ASIL A-D) | C89<br>C99<br>C++03<br>C++11<br>C++14 |
| IAR Embedded Workbench (functional safety edition) enthält Compiler | IEC 61508 (SIL 1-4)<br>ISO 26262 (ASIL A-D)<br>EN 50128<br>EN 50657<br>(teilweise IEC 62304) | C89<br>C11<br>C++14<br>C++17<br>C18 |

## Anhang B: Nützliche Links/Weitere Informationen

Häufige Fehler bezüglich funktionaler Sicherheit, zusammengetragen von TÜV Süd:
https://www.tuvsud.com/en/services/functional-safety/top-misunderstandings-about-functional-safety

Paper *IEC 61508-3 Software Assessments – Lessons Learned since 2010* von Lloyd, enthält in Abschnitt 2.2 Projekte, deren Konformität mit der Norm IEC überprüft wurde. Es werden das erreichte/zu erreichende SIl-Level, das Maß der Konformität in den einzelnen Phasen des Lebenszyklus und existierende/gelöste Herausforderungen angegeben. Abschnitt 3 fasst darüber hinaus Erkenntnisse aus den Projekten zusammen:
https://www.researchgate.net/publication/329074682_IEC_61508-3_Software_Assessments_-_Lessons_Learned_since_2010