Applikationssoftware für sichere (Safety-) Maschinensteuerungen erstellen

Dipl.-Ing. Torsten Borowski, Institut für Arbeitsschutz der Deutschen Gesetzlichen Unfallversicherung – BGIA, Fachbereich 5 – Unfallverhütung und Produktsicherheit

Development of application software for safety machine controls

More and more machinery requires flexible safety functionalities. Safety controllers (PLC) with application programming are essential for individual safety solutions. Errors in the programming of safety functions, however, can have dramatic repercussions for occupational safety on machinery and plant. They might contribute to accidents. Since troubleshooting during commissioning is almost impossible, application software development should put stronger emphasis on the specification, design and maintenance phases in order to avoid software faults.
The presented new standards ISO 13849 and IEC 62061 impose requirements concerning safety-related software development (for machinery) and can also be seen as a programmers support to implement application software.

software standards, fault avoidance, measures, v-model, tools

1. Kontext / Problemstellung

Wettbewerbsbedingt sind die Anforderungen an Steuerungssysteme für die Maschinen- und Anlagenautomation immens. Der Markt verlangt die Realisierung immer umfangreicherer, leistungsfähigerer und sowieso vernetzbarer, funktional flexibler und schneller Lösungen. Heutige Entwicklungen moderner Sicherheitssteuerungen unterstützen eine Gesamtintegration von Sicherheitstechnik und eine individuelle Implementierung maschinenbezogener Sicherheitsfunktionen. Dabei gewinnt deren Programmierbarkeit, d.h. die Erstellung von Applikations- bzw. Anwendungssoftware mit Sicherheitsaufgaben, immer mehr an Bedeutung. Aber manchmal sollen erst am Ende wirksame Sicherheitskonzepte integriert werden. Doch was ist mit Fehlern in der Applikationssoftware, die aufgrund mangelhafter Planung, fehlender Leistungs- und Anforderungsbeschreibung, fehlerbehafteter Implementierung und sorgfaltsloser Pflege bzw. Modifikation auftreten können oder besser: eingebracht werden? Da selbst die sicherste Steuerung auch ein fehlerhaftes Anwendungsprogramm ausführen wird, muss es – will man mögliche (neue) Ursachenschwerpunkte durch Applikationssoftware verhindern – geregelte Abläufe und konkrete Maßnahmen geben, mit denen man zuverlässige, qualitative und eben möglichst fehlerfreie Applikationssoftware realisiert!

Welche Regeln, Standards und Maßnahmen können Projekteure und Programmierer also für eine sicherheitsgerichtete Entwicklung von Applikationssoftware für Maschinen heranziehen? Und wodurch erhalten sie dabei Unterstützung?

2. Stand der Technik, Lösungsansatz

2.1 Sicherheitsfunktionen mit Applikationssoftware realisieren

Was macht Sicherheits-(Safety-)Steuerungstechnik aus? …. Kein Exkurs, sondern knapp: sie realisiert Sicherheitsfunktionen, die zwecks Risikominderung von Gefährdungen für Personen und der Umwelt maschinen- bzw. anlagenindividuell zugemessen werden und verfügt über die Eigenschaft, auf zufällige Ausfälle wie auch auf systematische Fehler reagieren zu können. Anders als bei „fertig“ bzw. embedded programmierten Serienprodukten hängt die “funktionale Sicherheit” bei (frei) anwenderprogrammierbaren Steuerungen – i.d.R. SPS – wesentlich von einer korrekten Umsetzung der spezifizierten Funktionalität für die jeweilige, ggf. nur einmalig realisierte Maschine oder Anlage von der Applikationssoftware ab – sie ist Mittelpunkt des Sicherheitskonzeptes! Das dies (im besten Sinne des Wortes) nicht funktionieren kann, wenn die Softwareplanung der Maschine/Anlage erst vor Ort, „last minute“ erfolgt, soll dieser Beitrag untermauern. Er zeigt ebenso diejenigen Maßnahmen auf, die im Entwicklungsverlauf von sicherheitsgerichteter Applikationssoftware angewendet werden sollen und selbst noch bei unvermeidlichen Anpassungen der Software in der Inbetriebnahmephase greifen (können), um Fehler „aus dem System zu holen“.

2.2 Applikationssoftware ohne Fehler?!

Einmal losgelöst von der Hardware – diese unterliegt natürlich ebenso wie das Betriebssystem (OS) bzw. hardwarenahe Firmwareschichten der Fehlerbetrachtung – können Fehler in der Applikationssoftware enthalten sein. Wann sich diese offenbaren ist so leicht nicht zu prognostizieren; oft wird von „Verkettung unglücklicher Zustände“, also von unvorhergesehenen betriebsmäßigen Eintrittsbedingungen, gesprochen. Aber gelänge eine exakte Prognose, dann ließe sich der Fehler doch „von vorn herein“ ausschließen! Genau dies ist für Softwarefehler festzuhalten: sie sind „von vorn herein“ enthalten (!), treten nicht zufällig auf (!) und werden sich unter „gegebenen Umständen“ auch reproduzierbar einstellen und auf die Sicherheitsfunktion(en) auswirken! Also müssen und können ausschließlich die Spezifikationsphase, die Entwurfs- und Codierungsphase, die Implementierungsphase und last but not least die Änderungsphase verantwortlich gemacht werden. Genauer noch: nicht die Phasen, sondern Diejenigen (Personen und Engineeringmittel), die sie ausführen bzw. dafür verantwortlich sind! Bild 1 zeigt (ohne Anspruch auf Vollständigkeit) Bereiche, denen man Softwarefehler zumeist zuordnen kann. Personenbezogene Fehlermöglichkeiten wie mangelnde Kenntnisse, Erfahrung und Verantwortungswahrnehmung, organisatorische Defizite, abweichende Interpretationen, Bedienfehler, „Stress“ bei nahender Übergabe etc. seien hier nicht weiter thematisiert.

Bild 1: Fehlerbereiche für Software

Mit den Fehlerbereichen vor Augen sind priore Ziele schnell formuliert: ein möglichst fehlerfreies und „anwendungsrobustes“ Progamm..

2.3 Wie kann Applikationssoftware sicher gestaltet werden?

2.3.1 Anwendbare Standards und Kriterien

Standards für programmierbare Systeme und Steuerungen mit Sicherheitsaufgaben gibt es inzwischen zahlreiche. Prinzipiell thematisch sortierbar finden sich Regelwerke z.B. für Bahn-/Signalanlagentechnik, Automotive-Lösungen, Prozess- und Prozessleittechnik und die Luftfahrtindustrie. Darunter – z.B. in der DIN EN 61511 – auch solche mit Anforderungen an Applikationssoftware. Auf den Maschinensektor übertragen sind diese Festlegungen zwar als informativ wertvoll anzusehen, aufgrund der teils stark abweichenden Erfordernisse zu funktionalem Verhalten (Reaktionen, sichere Zustände), Zeit- oder Zyklusaspekten, Kommunikations- und Wartungsstrategien u.a.m. aber nur bedingt anwendbar. Daher ist die Maschinenbranche sehr an eigenen internationalen Standards für ihren Sektor interessiert. Für programmierbare Sicherheitssteuerungen ist dies mit den recht neuen Normen DIN EN ISO 13849-1 [1] und DIN EN 62061 [2] inzwischen erfolgt. Applikationssoftware und eher noch der Applikationssoftware-Entwicklungsprozess ist in beiden Normen aufgegriffen. Entsprechend sind Anforderungen aufgestellt. Diese dürfen und können – so propagiere ich es gern – aber auch als wertvolle Hilfestellung in dieser nicht immer glasklaren Materie gelten!
Die Grundideen der Standards führt auf das Ziel, verständliche, gut strukturierte, einfach lesbare, testbare und pflegbare Software zu erhalten. Davon profitieren alle Beteiligten enorm und es ist zu erwarten, dass weniger Fehler vorhanden sind.

2.3.2 Allgemeine Regeln für die Softwareerstellung

Softwareerstellung ist mehr als Programmierung. Soll ein Softwareprojekt guter Qualität entstehen, so muss der gesamte Engineering- und Pflegeaspekt – man spricht vom „Software-Safety-Lifecycle“ – betrachtet werden. Erfahrungen zeigen, dass Softwarefehler überwiegend in der Spezifikationsphase, beim Entwurf und bei Modifikationen eingebracht werden. Und so berücksichtigt folgerichtig z.B. die DIN EN ISO 13849-1 hierzu Schwerpunkte. Prinzipiell lassen es die Normen zu, Anforderungen bzw. deren Wirksamkeit abhängig von einem Risikominderungsmaß abzustufen. Über *PL*s und *SIL*s soll hier aber nicht weiter differenziert werden. Die Beschreibungen erlauben es, generell zuzutreffen.

- Die Software spezifizieren

… um eine vollständige, widerspruchsfreie bzw. eindeutige und jederzeit rekapitulierbare Anforderungs- und Maßnahmenbeschreibung zu allen funktionalen, systemischen und leistungsbezogenen Belangen der Applikationssoftware in guter, verständlicher Beschreibungsqualität zu erhalten. Die Spezifikation ist Grundlage für die weitere Gestaltung, Prüfung und Abnahme. Hilfen zur Erstellung einer Software-Anforderungsspezifikation (Software Requirement Specification – SRS) finden sich genügend; z.B. bei IEEE [3]. Der Teil der Sicherheitsanforderungen – zufällig oder leider auch mit SRS (Safety Requirement Specification) abgekürzt – beinhaltet alle für die Software geltenden Festlegungen und Eigenschaften zu Sicherheitsfunktionen, den in der Anlage aufzudeckenden Fehlern und der erforderlichen Reaktion darauf sowie alle vorgesehenen Maßnahmen, Fehler im Software-Entwicklungsablauf zu vermeiden; einschließlich einer Prüfplanung. Bei der Erstellung einer SRS kann der Einsatz semiformaler Beschreibungsmittel (z.B. UML, ggf. CASE ... auch schon eine Checkliste hilft) und eine Nutzung von Hilfsmitteln/Tools z.B. zur Anforderungsverfolgung mehr als erwartet gute Dienste leisten.

- Modular und strukturiert entwerfen

… denn es wäre schlecht, wenn die spezifizierten Sicherheitsfunktionen nicht mehr leicht erkennbar wären. Den Programmaufbau und die Programmeinheiten (Module) mit dem Fokus auf Einzelfunktionen, bei reduzierten Schnittstellen und nachvollziehbaren Daten und Verknüpfungen in einen übersichtlichen Ablauf gebracht, erhöhen die Verständlichkeit nicht nur für den Erstprogrammierer ungemein. Und auch die Testbarkeit profitiert von klaren und einfachen Programmstrukturen.

- Defensiv programmieren und Fehlererkennung einsetzen

… damit die betriebsmäßigen Auswirkungen auf die Software selbst bei anormalen Konstellationen von Datenwerten, Datenflüssen und Steuerflüssen erkannt und in jedem Fall definiert reagiert wird. Defensiv programmieren meint kurz gesprochen fehlerrobuste Software zu erstellen, die sogar zu eigenen Entwurfsunzulänglichkeiten etwas entgegenzusetzen hat. Probate Softwaregestaltung, die damit gleichsam Fehlererkennung leistet, nutzt hierzu hauptsächlich Plausibilitätsprüfungen, Zusicherungen (Assertions) und Programmlaufüberwachung.

- Programmierrichtlinien / Programmierregeln anwenden

Sie beherrschen viele Programmiertricks (?), halten Programmkommentierung für überflüssig (?) und dokumentieren nur auf Anordnung? Dann sind Sie ….. falsch in einem Projekt mit Sicherheitsaufgaben. Programmierregeln haben den ernsthaften Hintergrund, Fehler in der Codierung durch allgemeingültige und auch firmenspezifische Konventionen zu vermeiden und zu der Erfüllung schon genannter Kriterien für Software deutlich beizutragen. Die Regelbereiche sind umfassend; von der Variablenverwendung bis zur Kommentierung. Alle Regeln (wer kennt die schon) an dieser Stelle aufzuschreiben, würde den Rahmen sprengen. Als Anfang einer Recherche nach allgemeinen Richtlinien sei hier folgende Literaturreferenz, [4] und [5], gegeben. Firmeneigene Regeln sollten Erfahrung zusammentragen, die „gelebte“ aber auch gelenkte Programmierpraxis wiedergeben und können z.B. mit einem einheitlichen „Style Guide“ beginnen. Für Applikationssoftware – gerade wenn die unten aufgeführten Punkte berücksichtigt wurden – wird man feststellen, dass sich ein Regelsatz auf ein sehr überschaubares Maß reduziert.

- Bewährte Entwicklungstools verwenden

„What You See Is Not What You Get“ … wer kennt es nicht? Und nach einem Fehler, der vom Variablen- oder Programmeditor, dem Compiler oder beim Download eingebracht wird, wird man erst gar nicht suchen oder ihn bei der Inbetriebnahme nur noch durch glückliche Zufälle feststellen. Also gilt die Devise, vertraute Tools mit bekannten und somit zu umgehenden „Bugs“ einzusetzen, die auch ansonsten weit verbreitet sind. Kommen redundante Steuerungsarchitekturen zur Anwendung, liegt eine Möglichkeit in der Verwendung unterschiedlicher (somit diversitärer) Tools. Ist dies nicht der Fall, müssten anzunehmende Tool-/PC-Fehler zur Implementierung „händisch“ gesucht werden. Ersparen wir uns, Such- und Aufwandsargumente gegenüberzustellen. Der Rat lautet: auf inzwischen verfügbare Safety-Programmiersysteme zurückzugreifen (s.u.).

- Die Entwicklungsschritte verifizieren / das Ergebnis validieren

Warum mit Fehleraufdeckung bis zur Betriebsphase, im GAU-Fall bis zur Unfallanalyse warten, wenn doch schon zuvor prinzipiell alle Softwarefehler eliminierbar sind? Die DIN EN ISO 13849-1, wie auch andere Standards, systematisieren diese Maßnahmen zur Vermeidung von Fehlern in Form der Berücksichtigung von Verifikationen und der Validierung im sogenannten „V – Modell“ für die Softwareentwicklung.

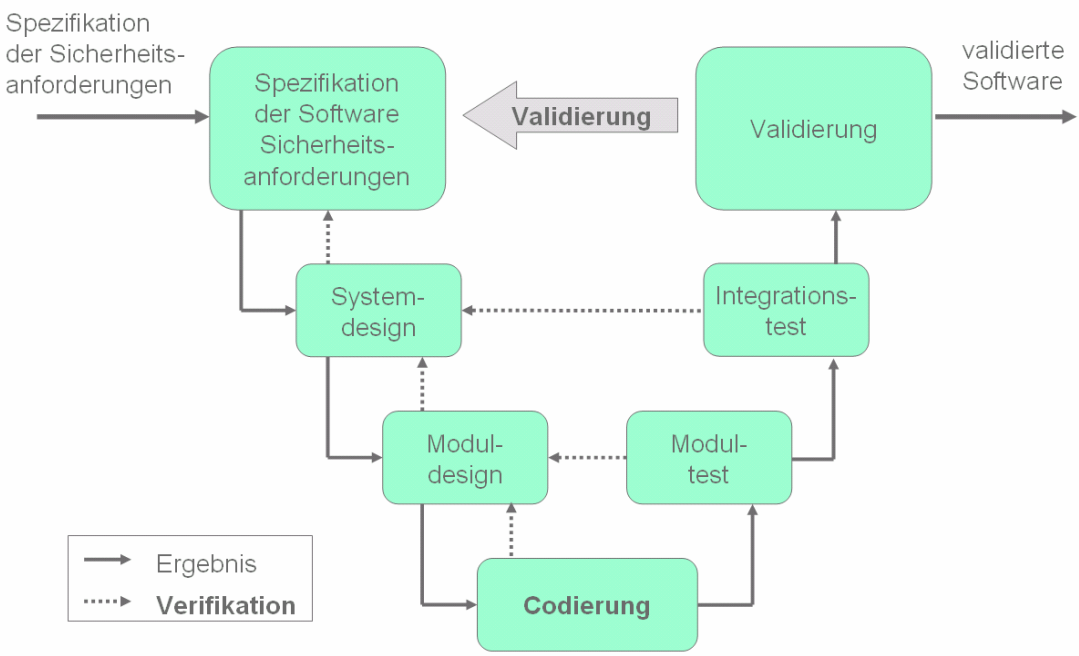

Bild 2: Vereinfachtes „V – Modell“ zur Softwareentwicklung in DIN EN ISO 13849-1

Mit Anwendung der DIN EN ISO 13849-1 für Applikationssoftware hat man gar den „Bonus“, nicht alles Machbare an Test- und Analysemaßnahmen durcharbeiten zu müssen, sondern **Reviews** und **Inspektionen** der Software-Spezifikation und der Softwarestruktur, eine gewisse Fehleranalytik (**FMEA**) auch für die Applikation, eine quasi Nachbetrachtung (**Analyse**) für editierte Daten- und Steuerflüsse, der **E/A-Test**, **Funktionstest** und eine sog. **erweiterte Funktionsprüfung** spätestens zur Inbetriebnahme reichen aus, das vereinfachte V-Modell (siehe Bild 2) zu durchlaufen. Aber: bitte Sorgfalt vor Schnelle! Welche Sorgfalt bei der Codierung aufgewendet wurde, kann noch mittels **Qualitätsmetriken** gemessen werden. Ein erstes kostenloses Tool hierzu ist z.B. unter www.bgia.de, Webcode 2386566 zu bekommen.
Nicht dem V-Modell zu entnehmen, jedoch keinesfalls zu vernachlässigen ist die Phase der Softwaremodifikation. Hier auch die Konsequenzen und Auswirkungen auf die vollständige Sicherheitsfunktionalität der Maschine/Anlage samt einzelner Eigenschaften zu erwägen (mittels **Einflussanalyse**) sollte selbstverständlich sein. Teamwork an dieser Stelle auch! Ist das Programm dann modifiziert, steht das o.g. „Handwerkzeug“ für Neuverifikationen und die erneute Validierung zur Verfügung.

- Die Software „ordentlich“ managen

… damit alles „glatt läuft“ bei der Realisierung des Applikationssoftware-Projekts. Auch hier gilt: nicht alles was QS/QM so zu bieten hat, muss zum Einsatz kommen. Zum „Grundgerüst“ des Vermeidens systematischer Softwarefehler gehört zweifelsohne eine adäquate **Projektdokumentation**. Hierzu zählen: die Spezifikation und das Programm (Struktur, Listing, verwendete Bibliotheken, Variablen), angewandte Programmierregeln und Sprachteilmengen, ein Nachweis durchgeführter Verifikationen und der Validierung sowie ein Abnahmeprotokoll. Ebenfalls elementar ist ein **Konfigurationsmanagement** (Versionierung,Identifikationen, Wer/Wann/Was/Womit) und die **Archivierung**.

Ein gutes Management zeichnet aus, wenn diese „Basics“ auch für Anpassungen, die sich erst in der Inbetriebnahmephase als erforderlich erweisen sowie bei Modifikationen/Änderungen greifen. Daher seien sie für diese Phasen nochmals extra „ans Herz gelegt“.

2.3.3 Applikationssoftware im Besonderen

- Geeignete Programmiersprache, Sprachenteilmengen und sprachenspezifische Programmierregeln wählen

Unter den SPS-Sprachen der IEC 61131-3 existieren Sprachen mit bereits limitiertem bzw. weniger variablem Befehlsvolumen, die sogenannten LVL (Limited Variability Languages). Dies sind Funktionsbausteinsprache (FBS) und Kontaktplan (KOP). Bei diesen graphischen Sprachen entfallen schon einige Konstrukte, die als fehleranfällig oder „schwer zu durchschauen“ gelten. Der „Not“(wendigkeit) großer Freiheitsgrade sei mit dem Argument begegnet, dass selbst sehr komplexe Projekte, z.B. in der chemischen Industrie, in FBS realisiert sind. LVL genießen auch den Vorteil intuitiver und verständlicher zu sein. Solche Kriterien gelten als einer geringen Fehlerwahrscheinlichkeit sehr nützlich. Dies gilt auch für Sprachenteilmengen, denn besonders problembehaftete oder schwierig analysierbare Sprachenkonstrukte werden damit ausgefiltert. Hier hilft eine international anerkannte Spezifikation von PLCopen, die einen Marktstandard für anwendbare SPS-Sprachen, Sprachelemente, Programmierrichtlinien und Sicherheits-Funktionsbausteine für Maschinensteuerungen darstellt [6].

- Funktionsbausteine und Bibliotheken einsetzen

Funktionsbausteine (FB) dienen insbesondere der höheren Transparenz der Programmierung, die sich insgesamt deutlich vereinfachen und verkürzen lässt, wenn FB verwendet werden. Sie können einzeln schon einen Teil der geforderten Funktionalität („fertig“) erfüllen, lassen sich kombinieren und zu neuen, größeren Funktionen entwickeln. Sind FB schon einmal validiert worden oder gar zertifiziert, so verbleibt der deutlich reduzierte Aufwand beim Testen der FB-Einbindung. Nicht zuletzt die Wiederverwendbarkeit spricht für eine Programmierung mit FB.
Die Spezifikation [6] standardisiert schon eine Vielzahl von Sicherheits-Funktionsbausteinen wie beispielsweise *Sicherer Halt*, *Schutztürüberwachung*, *BWS-Anbindung*, *Zweihandbedienung*, *Sicher begrenzte Geschwindigkeit*, u.a.m. Sie eignet sich natürlich auch als Vorlage für die Erstellung eigener FB-Spezifikationen.

- Programmierwerkzeuge (-Tools) auswählen

Wie – aus sicherheitsfunktionaler Sicht – leistungsfähig **Safety-Programmiersysteme** sind, lässt sich ihnen erst „auf den zweiten Blick“ anmerken. Zunächst wird nicht sonderlich viel auffallen, denn sie sind im Benutzungsverhalten und dem „Look & Feel“ bewusst an vertraute Standard-Tools angelehnt. Doch je weiter man in einer Safety-Projektierung voranschreitet, umso mehr Elemente der fehlervermeidenden und fehlerbeherrschenden Eigenschaften werden offenkundig. Vielleicht zunächst die Benutzerverwaltung samt

Zugangsmanagement, dann Namenskonventionen für Projekt-Objekte, Sprachenselektion und Sprachteilmengen … und dann evt. automatisch wirkende Verknüpfungsregeln und Platzierungs-Konventionen im Programmeditor. Dies wird sich fortsetzen bis hin zum Software-Abnahmeprotokoll. Doch ein solches Tool kann noch einiges mehr. Es prüft selbsttätig die Einhaltung einiger Programmierregeln, verfügt über Maßnahmen der sicheren Zielcode-/Maschinencodeerzeugung (Compilierung) und sorgt für Datenintegrität beim Download. Für gefahrloses Prüfen in der Anlage finden sich in der Debug-Ebene Funktionen zur Programmsimulation … und andere Dinge mehr. Als wesentliches „Feature" seien noch Trennungsmechanismen wie z.B. Safety-Datentypen und definierte Verknüpfungselemente zu bzw. mit nicht sicherheitsrelevanten Programmen und Variablen hervorgehoben. Damit wird das Problem der Rückwirkungen weitestgehend entschärft. Und wenn dann noch eine automatisch mitgenerierte Programmlaufüberwachung für den Applikationscode enthalten ist (siehe auch den Punkt „defensive Programmierung"), dann darf man im Kontext von Fehlervermeidung doch von „besten Diensten" reden.

- Wohldefinierte Trennung / Schnittstellen zur Standardsoftware

… best case: die soeben beschriebenen Safety-Programmiersysteme einsetzen.

- Fehlererkennung „einbauen"

Auch mit Applikationssoftware ist es möglich, „aktive" Fehlererkennung zu betreiben. Plausibilitätsprüfungen für E/A-Zustände, Bereichs- und Werteprüfung für Variablen (Ein- und Ausgangsvariablen) bis hin zu rein zu Fehlererkennungszwecken eingelesenen Anlagen-/Systemzuständen sollten immer möglich sein. Auch können diese Zustände über E/A-Variablen auf ihr Ablaufverhalten (die logische bzw. zeitliche Folge) an Erwartungshaltungen gebunden werden. Für eine Programmlaufüberwachung der Applikation wird es ggf. einer Unterstützung durch das Programmiersystem (s.o.) und durch das Betriebssystem bedürfen.

3. Ausblick

Mit der Verbreitung der beiden Normen [1] und [2] werden sich deren Anforderungen zur Softwareerstellung zunehmend in Lastenheften finden und auch in die Praxis des Applizierens eingehen. Mehr und mehr standardisierte Sicherheits-Funktionsbausteine und -Bibliotheken werden für eine vereinfachte wie fehlerarme Umsetzung in Softwareprojekten sorgen können. Alle Beteiligten können von einer höheren Planungssicherheit, kürzeren Entwicklungs- und Abnahmezeiten profitieren. Und wenn auch weiterhin genügende Mitarbeit und Unterstützung für Marktstandards wie [6] mobilisiert wird, dann muss es einem „Unfallschützer" nicht bange werden. Unter den noch zu leistenden Punkten wäre aus Sicht des Autors ein Workshop mit dem Thema **„**Aus Softwarefehlern lernen**"** sinnvoll, in dem aus einem breiten Teilnehmerkreis Erfahrungen zu Softwarefehlern zusammengetragen werden und für die Praxis zu fokussierende Aspekte konkretisiert werden.

Literaturverzeichnis

[1] DIN EN ISO 13849-1:02.2007, Sicherheit von Maschinen - Sicherheitsbezogene Teile von Steuerungen, Allgemeine Gestaltungsleitsätze, Beuth Verlag, Berlin

[2] DIN EN 62061:10.2005, Sicherheit von Maschinen - Funktionale Sicherheit sicherheitsbezogener elektrischer, elektronischer und programmierbarer elektronischer Steuerungssysteme, Beuth Verlag, Berlin

[3] IEEE Guide to Software Requirements Specification, ANSI/IEEE Std 830-1984, IEEE Press, Piscataway, New York, 1984.

[4] MISRA: MISRA-C:2004 - Guidelines for the use of the C language in critical systems, ISBN 0952415623; *www.misra.org.uk*

[5] Programmierregeln für die Erstellung von Software für Steuerungen mit Sicherheitsaufgaben, Schriftenreihe der BAuA, Dortmund, Fb 812, ISBN 3-89701-212-X

[6] Safety Software – Technical Specification, Part 1: Concepts and Function Blocks; Version 1.0, PLCopen, TC5, Gorinchem, NL; *www.plcopen.org/pages/tc5_safety*

[7] in Vorbereitung:

Inhalt:

- Risikobetrachtung, Beispiele zur Risikoabschätzung
- Performance Level Kategorie/MTTF/DC/CCF Quantifizierung
- **Entwicklungsablauf, Softwareerstellung**
- Steuerungsbeispiele (alle/gemischte Technologien)
- Verifizieren & Validieren
- Fehlerlisten
- erforderliche Dokumentation, Beispielspezifikation
- .... viele Informationen