Softwarebasierte Anlagenabsicherung in Chemieanlagen
- Erfahrungen bei der Erstellung von SSPS-Applikationssoftware -

Dirk Hablawetz

BASF Aktiengesellschaft, Fachstellen für Prozessleittechnik
D-67056 Ludwigshafen
dirk.hablawetz@basf-ag.de

Abstract: Es gibt viele Wege, die Risiken eines kritischen Prozesses zu minimieren. Der beste Weg ist sicherlich einen in sich sicheren Prozess zu entwerfen. Wo dies nicht möglich ist, übernehmen PLT-Schutzeinrichtungen die Aufgabe dieses Risiko zu minimieren. In den letzten 10 Jahren entwickelten sich sicherheitsgerichtete speicherprogrammierbare Steuerungen (SSPS) zunehmend zum Mittel der Wahl bei der Absicherung von kritischen Prozessen in der chemischen Industrie. Das Spektrum von softwareintensiven PLT-Schutzsystemen reicht von einfachen Systemen, welche lediglich bei einer großen Anzahl von E/A-Punkten die traditionellen VPS-Systeme aus ökonomischen Gründen substituieren, über Systeme zur Absicherung von Batch-Anlagen bis hin zu komplexen Schutzsystemen.
Während triviale Standard-Sicherheits-Schaltungen wie „Druck-zu-Hoch–Überwachung“ einfach zu beherrschen sind, erfordern komplexe PLT-Schutzsysteme weitergehende Entwurfs-, Entwicklungs-, Dokumentations- sowie Verifizierungs- und Validierungsmethoden. In diesem Beitrag wird insbesondere auf Erfahrungen in der Spezifikationsphase komplexer sicherheitsrelevanter Applikationssoftware im Umfeld der Prozessindustrie eingegangen.

1. Software in SSPS-Systemen

Die Software von SSPS-Systemen setzt sich aus einer Vielzahl von Programmen mit sicherheitsrelevantem Aspekt zusammen:

- dem Betriebssystem,
- der Firmware,
- der Kommunikationssoftware,
- dem Benutzerinterface und diverse Hilfsprogramme,
- der Software-Engineeringtools,
- und nicht zuletzt die Applikationssoftware, die anlagenspezifischen Funktionen enthaltende, Software.

Während a priori bis auf die Applikationssoftware alle anderen Programme im Rahmen der Systemzertifizierung (Nachweis der Eignung bzw. Zulassung für Sicherheitsaufgaben) als „sicher“ gelten, wird die Qualität und Zuverlässigkeit des Applikationsprogramms scheinbar maßgeblich von der „Qualität“ des Programmierers und seinem Umfeldes bestimmt. Der Grundstein für ein qualitativ hochwertiges und damit sicheres Applikationsprogramm wird jedoch schon in der Spezifikationsphase gelegt.

Allgemein werden in der SSPS-Praxis drei Typen von Sprachen verwendet:

- Fixed Program Languages (FPL) – Feste Programmiersprachen:
  Dazu zählen Parametriersprachen wie man sie oft in Smart Transmittern oder Geräten mit beschränkter Funktionsvielfalt wiederfindet. In vielen Fällen können nur bestimmte Parameter oder einzelne Funktionen aktiviert oder verändert werden.

- Limited Variability Languages (LVL) – Programmiersprachen mit begrenzter Variabilität:
  Typische Vertreter dieser Sprachen sind die Funktionsbausteinsprache (FBD), Kontaktplan (LL) oder Ablaufdiagramme (SFC). Diese Sprachen erlauben dem Anwender in einem relativ weitem Rahmen Funktionen miteinander zu kombinieren oder neue Funktionen zu entwickeln um damit die geforderte Funktionalität des Schutzsystems zu gewährleisten. Die Grundfunktionen dieser Sprachen sind meist in Bibliotheken enthalten, welche mit dem SSPS-System mitgeliefert werden.

- Full Variability Languages (FVL) – Programmiersprachen mit voller Variabilität:
  Diese Sprachen sind am ehesten mit allgemeinen Programmiersprachen wie C++ oder Pascal zu vergleichen. Ein Vertreter im SSPS-Bereich ist die Sprache „Strukturierter Text“ (ST). Typisch für diese Sprachen ist der Zugriff auf betriebssystemnahe Funktionen und ihre hohe Flexibilität. Die Nutzung dieser Sprachen sollte allerdings nur auf Spezialisten mit ausreichender Kenntnis und bei Anwendungen mit einem hohen Komplexitätsgrad beschränkt sein.

Die bevorzugte Sprache in der Prozessindustrie ist die Funktionsbausteinsprache. Mit dieser Sprache können sowohl triviale als auch komplexe Funktionen relativ einfach visualisiert und damit programmiert werden. Gleichzeitig wird sie in der betrieblichen Praxis als Hilfsmittel zur Darstellung der Funktionen im Rahmen der Spezifikation verwendet.

Generell werden folgende Mindestanforderungen an sicherheitsrelevante Applikationsprogramme gestellt:

- eine modulare und klare (Programm-)Struktur und damit einfache Testbarkeit,
- Verständliche Darstellung der Funktionen,
  - für den Operator auf dem Bildschirm (Navigation),
  - Lesbarkeit des späteren Dokumentationsausdrucks
- symbolische und aussagekräftige Variablennamen und Kommentare,
- Verwendung einfacher Funktionen (keine indirekte Adressierung, keine Variablenfelder),

- defensive Programmierung,
- leichte Erweiterbarkeit (Anpassung).

Oberstes Ziel bei sicherheitsrelevanter Programmierung ist eine hohe Verständlichkeit der programmierten Funktionen, d.h. es sind klare und kompakte Darstellungen bzw. Programmkonstrukte notwendig. Hauptproblem ist dabei oftmals die historisch bedingte Hardwareorientierung und die dadurch geprägte Denkweise der Programmierer und Planer im SPS-Bereich.

Beispielhaft zwei Programmmodule (mit identischer Funktion!) zur Mittelwertbildung einer Prozessgröße über die Zeit [Jü02]:

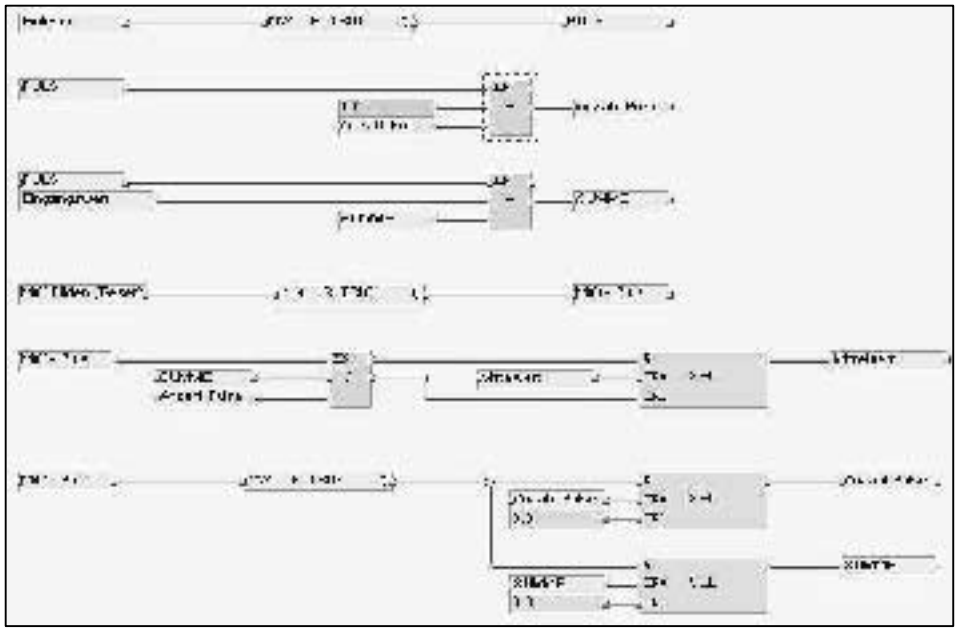

Abbildung 1: Hardwareorientierte, oftmals typische Darstellung

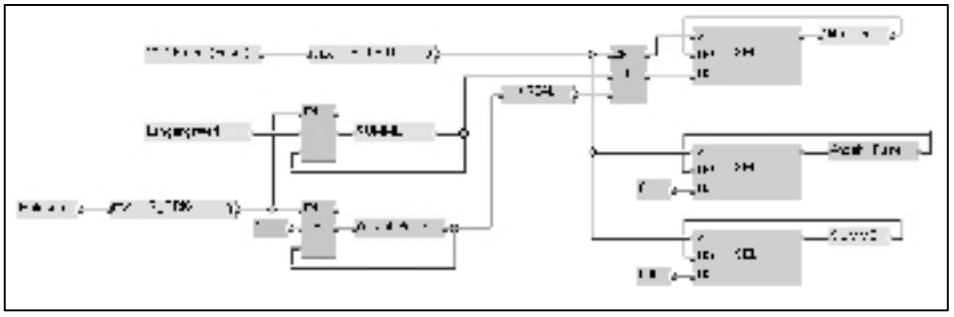

Abbildung 2: besser: kompakte, funktionsorientierte Darstellung

Die Codierungsverfahren und Implementierungsstrategien sowie die Programmierung stellen heute prinzipiell kein Problem mehr dar, sind in der Fachliteratur ausreichend beschrieben [Ha98], [Jü02], [Br00], [Su02] und in der Praxis bekannt. Es gibt jedoch durchaus Unterschiede in der Qualität der Programmierung von SSPS-Systemen. Dabei spielt insbesondere die Erfahrung der ausführenden Personen/Firma im sicherheitstechnischen Bereich eine wichtige Rolle.

Vor der Codierung kommen jedoch die Phasen der Spezifikation sowie das Design. In der Designphase wird aus der eigentlichen Aufgabenstellung (der Spezifikation) abgeleitet, wie das Programm zu strukturieren ist und welche Sprachen und Datentypen verwendet werden. In der betrieblichen Praxis entfällt oftmals diese Phase durch überwiegend zeitliche Zwänge zugunsten einer sofortigen Codierung. Das Ergebnis ist in vielen Fällen „Spaghetti-Code“. Umso wichtiger wird somit die Spezifikation als die einzige Basis der Programmerstellung.

2. IEC61511

Erstmalig für die Prozessindustrie beschreibt IEC61511 [IEC61511] für Applikationssoftware einen eigenen umfassenden Softwareentwicklungszyklus, welcher die Entwicklung von Applikationssoftware auf eine systematische Grundlage (V-Modell) stellt. Diese Vorgehensweise orientiert sich sehr eng an der Softwareentwicklungsmethodik, wie wir sie in anderen SW-Entwicklungsmodellen, wie z.B. SW-CMM (Capability Maturity Model for Software) oder SPICE (Software Process Improvement and Capability dEtermination - ISO/IEC 15504) wiederfinden und umfasst alle Phasen der Softwareentwicklung von der Erstellung der Spezifikation bis zum Integrationstest.

Aus sicherheitstechnischem Blickwinkel ist eine solche systematische Vorgehensweise ganz klar zu unterstützen, jedoch muss der Umfang der einzelnen Entwicklungsschritte, auch unter Beachtung der wesentlich geringeren Komplexität der Programme, an die Gegebenheiten in der Prozessindustrie angepasst werden. Dies geschieht in vielen Fällen durch firmeninterne Richtlinien, welche sowohl auf aktuellen Standards basieren aber auch die langjährigen firmeninternen Erfahrungen berücksichtigen.

3. Die Spezifikation - Herausforderung in der tägliche Praxis

Applikationssoftware wird in der Praxis üblicherweise nicht mehr durchgängig in einem Unternehmen entwickelt. Die Entwicklung selbst ist immer ein iterativer Prozess.

Unternehmen in der Prozessindustrie als Auftraggeber liefern die Spezifikation und definieren den Verifikations- und Validationsumfang. SSPS-Hersteller, Kontraktoren, Systemintegratoren oder Ing.-Büros programmieren und implementieren die Software - eine typische Kunden-Lieferanten-Beziehung. Damit wird sehr schnell deutlich: Alles, was nicht in der Spezifikation an Anforderungen enthalten und nicht ausreichend und widerspruchsfrei beschrieben ist, wird im Extremfall entweder nicht realisiert oder kann

durch Interpretationen fehlerhaft sein. Softwarefehler sind in jedem Fall systematische Fehler und oftmals auf eine mangelnde Spezifikation zurückzuführen.

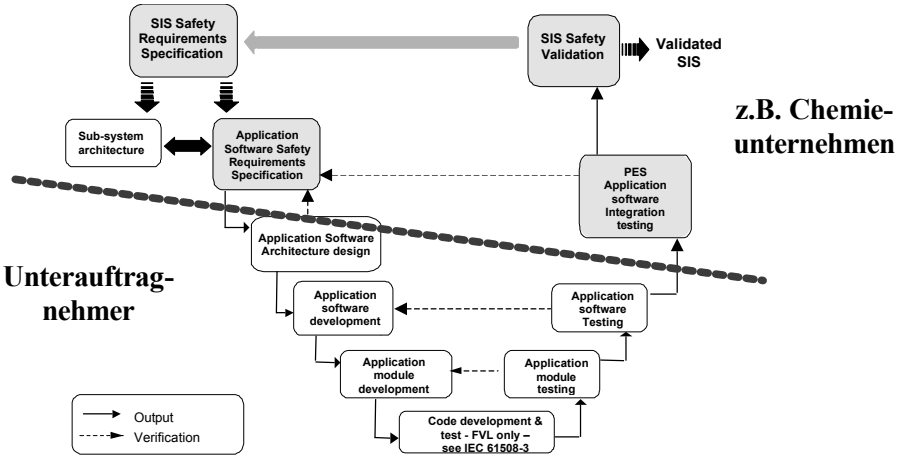

Abbildung 3: Softwareentwicklungszyklus nach IEC61511

Was gehört in eine Spezifikation für sicherheitsrelevante Software? Eine Darstellung aus den MISRA-Guidelines (Motor Industry Software Reliability Association) [MISRA] gibt einen guten Überblick:

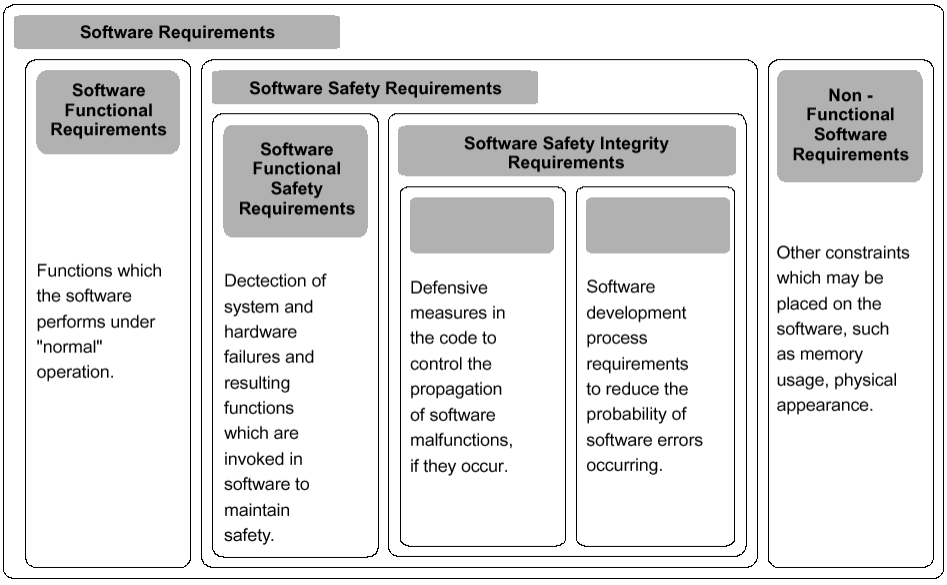

Abbildung 4 : Spezifikation sicherheitsrelevanter Software

Die Formulierung der funktionalen Anforderungen an die Applikationssoftware bereitet in vielen Fällen keine Probleme. Die Definition der Anforderungen an die Sicherheit der Applikationssoftware scheint jedoch nicht so einfach zu sein. Warum?

Viele Projektierer oder Planer agierten bis vor einigen Jahren in einem nahezu ausschließlich von Hardware geprägtem Umfeld. (Hardware-) Diagnose und andere Sicherheitsmechanismen waren fest vorgegeben und im Rahmen der Zertifizierungen verifiziert. Man konnte sich ausschließlich auf die Funktionalität konzentrieren. In modernen SSPS stimmt dies zwar auch noch – nur ist es reduziert auf die Grundfunktionen der SSPS, am einfachsten zu vergleichen mit dem Betriebssystem eines PC. Die Anforderungen an die Applikationssoftware der SSPS, die in jedem Fall ein Unikat ist, umfassen jedoch beides – die applikationsspezifische Funktionalität und die Sicherheitsanforderungen (Safety Requirements Specification – SRS).

Der überwiegende Teil der SSPS-Applikationsprogrammierer sind keine Softwareingenieure bzw. nicht im Softwareengineering ausgebildet, sondern in vielen Fällen Praktiker, wie Meister, Techniker oder Ingenieure, die überwiegend funktions- und lösungsorientiert denken. Dies ist aber für sicherheitsrelevante Lösungen nicht ausreichend. Sicherheit wird im wesentlichen durch das Erkennen von Fehlern im System bzw. potenziell gefährlichen Zuständen in der Applikation erreicht. Deshalb ist es insbesondere bei komplexen Systemen notwendig, in der Spezifikationsphase darüber nachzudenken,

- welche Defekte oder Fehler auftreten können und welche Konsequenzen diese haben (z.B. verfälschter Messwert aus dem Prozess),
- wie Defekte oder Fehler erkannt werden können (z.B. Vergleich mit zweitem, messtechnisch homogenem, Messwert),
- welche Reaktionen eingeleitet werden müssen (z.B. Alarm ohne Konsequenz, Reparaturanforderung, Abschaltung).

Die Antworten auf solche oder ähnliche Fragen spiegeln sich in der Spezifikation als definierte Anforderungen wider, diese (klar benannten!) Fehler zu erkennen und zu beherrschen.

O.g. Vorgehen verlangt phasenweise eine recht destruktive Denk- und Betrachtungsweise (siehe Murphy: „Alles was schief gehen kann, wird schief gehen."). Es gelingt nur, wenn derjenige, der die Spezifikation erstellt, die eigentliche Funktion, die sicherheitstechnische Aufgabe sowie das anlagentechnische Umfeld sehr genau kennt und verstanden hat sowie potenzielle Risiken erkennen und beurteilen kann. Kritischen Phasen im Betrieb einer Anlage sind z.B. der Start- oder Wiederanfahrprozess. Werden diese Phasen nicht gut genug verstanden, kann es schwierig werden, die Anlage später wieder zu starten. Fehlinterpretation können schwerwiegende Folgen haben und in eine inkonsistente Programmlogik oder falsche Abschaltaktionen münden.

Die Suche nach kritische Konstellationen verlangt oft eine gewisse Kreativität und Erfahrung. Zwei Beispiele sollen dies verdeutlichen:

1. Die Sommer-/Winterzeitumstellung im Zusammenhang mit der verwendeten Hardware können zu einer potentiell gefährlichen Situation führen: Prozessleitsystem (PLS) und SSPS sind miteinander verkoppelt und arbeiten, durch das PLS synchronisiert, mit derselben Uhrzeit. In der SSPS wird ein mengenabhängiger Koeffizient für einen Grenzwert ermittelt. Bei Erreichen des Grenzwertes muss die Anlage abgeschaltet werden. Die Menge wird über einen bestimmten Zeitraum integriert, die dafür verwendete Zeit basiert auf der Zykluszeit der SSPS und kann über komplexe Timerkonstrukte aus der Systemzeit der verwendeten SSPS ermittelt werden. Im PLS erfolgt die Sommer-/ Winterzeitumstellung. Die nachfolgende Synchronisation definierte die Systemzeit in der SSPS neu und über den vorn erwähnten Weg änderte sich damit auch der Integrationszeitraum um genau eine Stunde. Das Ergebnis ist ein falsch berechneter Koeffizient und damit ein falscher sicherheitsrelevanter Grenzwert.

2. Wenn implizites Wissen im Rahmen der Spezifikation nicht in klar definierte Anforderungen überführt wird, kann dies ungewollt Raum für Interpretationen schaffen: Ein Planer kann z.B. die Anforderung „Abschaltung bei Erreichen eines Grenzdruckes“ formulieren, er erwähnt jedoch nicht die in diesem Fall (ihm durchaus bewusste) besondere sicherheitstechnische Bedeutung der Nachkommastellen des gemessenen Druckwertes. Der Programmierer indes weist diesem Druckwert den Datentyp INTEGER zu. Die Nachkommastellen werden somit nicht verarbeitet und eine verzögerte Reaktionen der Schutzeinrichtung ist vorprogrammiert.

4. Defensive Programmierung

Wie geht man mit Zuständen oder Konstellationen um, die sicherheitstechnisch bedeutsam sein können und von der Spezifikation nicht oder nicht vollständig abgedeckt werden, ggf. sogar vergessen wurden? In den Mindestanforderungen an Applikationssoftware wurde „defensive Programmierung“ gefordert. Defensive Programmierung verlangt Softwarekonstrukte, die eine Überwachung der Software auf Fehler, welche zur Laufzeit des Programms auftreten können (falsche Eingaben, korrupte Daten, falsche Berechnungen [Formel falsch, Überlauf, ...], Speichermangel, ...) ermöglichen. An bestimmten Stellen im Programmablauf werden Bedingungen eingefügt, deren Ergebnisse Zusicherungen (Assertions) für die Gültigkeit der z.B. Eingabewerte, der Vorergebnisse oder des Programmablaufes darstellen.

Beispiele dafür sind

- die Parameter, die eine Prozedur oder Funktion übergeben bekommt, erfüllen bestimmte Bedingungen (z.B. der Messwert liegt im Bereich von 4...20 mA),

- die Rückgabe einer Prozedur/ Funktion erfüllt bestimmte Bedingungen (z.B. Druckberechnung: Bei dieser Reaktandenmenge X kann max. ein bestimmter Druck Y entstehen),
- für eine Menge von Variablen gelten bestimmte Plausibilitäten (z.B. wenn ein Ventil angesteuert ist, muss auch eine entsprechende Rückmeldung vom Ventil kommen, dass es geöffnet hat),
- das Programm wird in der richtigen Reihenfolge und zeitfolgerichtig abgearbeitet (z.B. Programmlaufüberwachung).

Diese technisch bzw. funktional scheinbar nutzlosen „Zwischenprüfungen“, von dessen Notwendigkeit ein eingefahrener Praktiker anfänglich nur schwer zu überzeugen ist („... erzeugt doch nur höhere Komplexität...“, „...ist das wirklich notwendig...“, „...haben wir noch nie so gemacht...“), geben komplexen Applikationsprogrammen jedoch die notwendige Sicherheit und Robustheit. Oftmals werden damit auch Fehler abgefangen, die vorher nie betrachtet wurden. Die Implementierung von Assertions kann direkt in den funktionalen Ablauf integriert werden oder (besser lesbar) als „Parallelprogramm“, quasi zur Überwachung, erfolgen.

Ein Programm ist nur dann sicher und zuverlässig, wenn es gleichzeitig fehlertolerant ist, d.h. es mit möglichst vielen (und überwiegend als wahrscheinlich angenommenen) Fehlern klarkommt, diese ausgleichen kann, in jedem Fall definiert reagiert oder den Anwender informiert (z.B. Weiterarbeiten mit Default-Werten, kontrollierter Abbruch, Fehlermeldung, ...).

Eine gute Spezifikation ist nicht nur unabdingbare Voraussetzung für die sichere Erstellung der Applikationssoftware selbst. Sie ist gleichzeitig die Grundlage für die spätere Prüfung der Applikationssoftware, insbesondere im Rahmen des Tests bei der Herstellerfirma (FAT – Factory Acceptance Test). In vielen Fällen ist dieser Test der umfangreichste Test vor der eigentlichen Inbetriebnahme einer Schutzeinrichtung. Neben den geforderten Funktionen müssen gleichzeitig die Anforderungen an Sicherheit und Robustheit getestet werden. Dabei kommen i.a. alle o.g. Forderungen an die Spezifikation zum Tragen und garantieren einen systematischen(!) und umfassenden Test, dessen Testkriterien aus den einzelnen Anforderungen der Spezifikation direkt abgeleitet werden können.

5. Zusammenfassung

Die Absicherung von prozesstechnischen Anlagen in der chemischen Industrie durch softwareintensive Systeme, wie z.B. sicherheitsgerichtete speicherprogrammierbare Steuerungen, wird zukünftig noch stärker zunehmen. Die Anzahl Schutzeinrichtungen steigt insgesamt auf Grund schärferer Gesetze und Umweltauflagen sowie dem Ersatz von mechanischen Schutzeinrichtungen an. Parallel dazu wirken aber auch wirtschaftliche Zwänge: Anlagenausbeute sowie Produktionsausstoß sollen weiter optimiert werden. Dies erfordert neue, in den meisten Fällen komplexere, Schutzeinrichtungen, welche bei gleicher Sicherheit wie heute, einen kleineren Sicherheitsabstand bis zur

Belastbarkeitsgrenze der Anlagen ermöglichen. Bei hoher Komplexität dieser Schutzeinrichtungen wird sich dies überwiegend in SSPS-Applikationssoftware niederschlagen.

Im Gegensatz zu mechanischen Einrichtungen nutzt sich Software nicht ab bzw. altert nicht. Softwarefehler sind keine zufälligen Fehler – sie sind grundsätzlich systematisch und treten somit unter identischen Randbedingungen immer wieder auf. Alle Fehler in einer Software werden prinzipiell in das System „hineinentwickelt". Um diese Besonderheiten von Software zu beherrschen, müssen Fehler primär in der Entstehungsphase vermieden werden. Eine sorgfältige und vollständige Spezifikation ist dabei der wichtigste Schritt.

Gleichzeitig ist abzusehen, dass es im Planungsbereich in naher Zukunft einen Paradigmenwechsel von einer hardwareorientierten hin zu einer funktionsorientierten Denkweise geben muss. Diese „neue" Betrachtungsweise muss durch alle am Entwicklungsprozess von SSPS-Applikationssoftware Beteiligten unterstützt werden, sei es durch zusätzliche Ausbildung, Entwicklungs- und Spezifikationsrichtlinien oder geeignete Software-Engineeringtools.

Literaturverzeichnis

[Ha98] Hablawetz, D.: Applikationssoftware und Systemsicherheit in speicherprogrammierbaren Systemen, at – Automatisierungstechnik Heft 2 1998, R.Oldenburg Verlag, München, 1998

[Jü02] Jülly, U.: Projektierung mit ELOPII, Firmenschrift HIMA Paul Hildebrandt GmbH +Co KG, nicht veröffentlicht, Brühl, Juni 2002

[Br00] Brendel, W.: Strategies for Software Tools and Programming Languages for PLCs in Safety related Applications, Vortragsskript, 4. Internationales Symposium für programmierbare Systeme für sicherheitsgerichtete Anwendungen, Köln, Mai 2000

[Su02] Summers, Angela E.: Software-implemented Safety Logic, Process Safety ProgressJune 2002, Vol. 21, No. 2, , AIChE, New York, 2002

[IEC61511] draft IEC 61511: Functional safety: Safety Instrumented Systems for the process industry sector, Part 1: Framework, definitions, system, hardware and software requirements, Geneva, February 2002

[MISRA] MISRA Electrical Group: Development Guidelines for Vehicle Based Software, Nuneaton, Warwickshire, November 1994